



---

# **Oracle 9iAS: Develop Web-Based Applications with PL/SQL**

**Student Guide**

---

40050GC10  
Production 1.0  
January 2001  
D31418

**ORACLE®**

**Authors**

Shena Deuchars  
Brian Fry  
Ashesh Parekh

**Technical Contributors**

Ed Dowgiallo  
Peter Driver  
Larry Frazier  
Lek Kajornsak  
Istvan Kiss

**Publisher**

Carol Brown  
Don Griffin

**Copyright © Oracle Corporation, 2001. All rights reserved.**

This documentation contains proprietary information of Oracle Corporation. It is provided under a license agreement containing restrictions on use and disclosure and is also protected by copyright law. Reverse engineering of the software is prohibited. If this documentation is delivered to a U.S. Government Agency of the Department of Defense, then it is delivered with Restricted Rights and the following legend is applicable:

**Restricted Rights Legend**

Use, duplication or disclosure by the Government is subject to restrictions for commercial computer software and shall be deemed to be Restricted Rights software under Federal law, as set forth in subparagraph (c)(1)(ii) of DFARS 252.227-7013, Rights in Technical Data and Computer Software (October 1988).

This material or any portion of it may not be copied in any form or by any means without the express prior written permission of Oracle Corporation. Any other copying is a violation of copyright law and may result in civil and/or criminal penalties.

If this documentation is delivered to a U.S. Government Agency not within the Department of Defense, then it is delivered with "Restricted Rights," as defined in FAR 52.227-14, Rights in Data-General, including Alternate III (June 1987).

The information in this document is subject to change without notice. If you find any problems in the documentation, please report them in writing to Education Products, Oracle Corporation, 500 Oracle Parkway, Box SB-6, Redwood Shores, CA 94065. Oracle Corporation does not warrant that this document is error-free.

Oracle is a registered trademark and SQL, PL/SQL, and Oracle Internet Application Server are trademarks or registered trademarks of Oracle Corporation.

All other products or company names are used for identification purposes only, and may be trademarks of their respective owners.

# Contents

## Preface

- Profile 3
- Related Publications 4
- Typographic Conventions 5

## Lesson 1: Introduction to PL/SQL Web Applications

- Objectives 1-2
- Overview 1-3
- iAS Architecture 1-4
- Benefits of the PL/SQL Gateway 1-5
- The Flow of HTTP Requests 1-6
- Course Application 1-7
- Summary 1-10
- Practice 1 Overview 1-11

## Lesson 2: Creating a Web Page with HTML

- Objectives 2-2
- Overview 2-3
- Uniform Resource Locator (URL) 2-4
- HTML Tags 2-5
- Basic Elements of a Web Page 2-6
- Basic HTML: Example Code 2-7
- Basic HTML: Example Output 2-8
- Block Element Tags 2-9
- List Tags 2-10
- Formatting for Special Characters 2-11
- Creating Document Links 2-12
- Adding In-Line Images 2-13
- Retrieving the Default Web Page 2-14
- Summary 2-15
- Practice 2 Overview 2-16

## Lesson 3: Using the PL/SQL Web Toolkit

- Objectives 3-2
- Overview 3-3
- PL/SQL Package Syntax 3-4
- Passing Parameters to a Procedure 3-5
- The PL/SQL Web Toolkit 3-6
- Benefits of the PL/SQL Web Toolkit 3-7
- HTP and HTF Packages 3-8
- Generating HTML Using HTP 3-9

Generating HTML Using HTF	3-10
Passing Parameters to a Toolkit Procedure	3-11
Creating a Basic Page	3-12
Calling a Procedure from a URL	3-14
Building a Generic HTML Document Header	3-15
Building a Basic HTML Document Body	3-16
Building a Generic HTML Document Footer	3-17
Creating a Document Link	3-18
Including Dynamic Data in an HTML Page	3-20
Summary	3-21
Practice 3 Overview	3-22

## **Lesson 4: Building a Dynamic HTML Table**

Objectives	4-2
Overview	4-3
Basic Table Elements	4-4
Table Tags	4-5
Displaying Information in a Table	4-6
Table Attributes	4-7
Building a Dynamic Table	4-8
Selecting Table Data	4-9
Opening and Closing a Table	4-10
Adding a Table Caption	4-11
Creating Rows and Headers	4-12
Loading Dynamic Data into a Table	4-13
Printing a Database Table Using owa_util.tableprint	4-14
Printing Part of a Database Table	4-15
Printing SQL Query Results Using owa_util.cellsprint	4-16
Table Tips	4-17
Summary	4-18
Practice 4 Overview	4-19

## **Lesson 5: Building Dynamic Forms**

Objectives	5-2
Overview	5-3
Defining a Form in HTML	5-4
Text and Password Fields	5-5
Check Box and Radio Button Fields	5-6
Text Area Fields	5-7
Select List Fields	5-8
Hidden Fields	5-9
Submit and Reset Buttons	5-10
Opening a Form	5-11
Creating a Text Field	5-12
Creating a Radio Group	5-13
Creating a Select List	5-14
Creating a Hidden Field	5-15

- Submitting a Form 5-16
- Resetting a Form 5-17
- Closing a Form 5-18
- Passing Multiple Values 5-19
- Handling Multiple Values 5-20
- Flexible Parameter Passing 5-21
- Creating an Image Map 5-22
- Summary 5-23
- Practice 5 Overview 5-24

## **Lesson 6: Adding Functionality with Toolkit Utilities**

- Objectives 6-2
- Overview 6-3
- Viewing a Stored Procedure 6-4
- CGI Environment Variables 6-5
- CGI Information 6-6
- Retrieving an Environment Variable 6-7
- Viewing CGI Environment Variables 6-8
- Printing a Calendar 6-9
- Entering Dates in Forms 6-10
- The `owa_pattern` Package 6-11
- Calling Other Applications 6-13
- Summary 6-15
- Practice 6 Overview 6-16

## **Lesson 7: Enhancing Client Interaction**

- Objectives 7-2
- Overview 7-3
- Cookies 7-4
- The `owa_cookie` Package 7-5
- Using Cookies 7-6
- Cookies and Application Navigation 7-7
- Using Image Maps 7-8
- Opening and Closing the Map 7-9
- Creating the Links 7-10
- Specifying the Image 7-11
- Redirecting the URL 7-12
- Uploading and Downloading Files 7-13
- Document Table Structure 7-14
- Uploading Files 7-15
- Downloading Files 7-17
- Summary 7-18
- Practice 7 Overview 7-19

## **Lesson 8: Validating Data and Executing Transactions**

- Objectives 8-2
- Overview 8-3
- DML in Web Applications 8-4
- Data Validation Methods 8-5
- Concurrency Problem 8-6
- Possible Concurrency Solutions 8-7
- Copying the Data 8-8
- Checking Copied Data 8-9
- Using a Row Checksum 8-10
- Verifying the Checksum 8-11
- Web Transaction Architecture 8-12
- Transactions in iAS 8-13
- Summary 8-14
- Practice 8 Overview 8-15

## **Lesson 9: Creating Display Areas with Frames**

- Objectives 9-2
- Overview 9-3
- What Is a Frame? 9-4
- Layout Documents Versus Content Documents 9-5
- Creating a Layout Document 9-6
- Creating a Frameset 9-8
- Frameset Dimensions 9-9
- Defining the Frame 9-10
- Creating a Container for Frame-Incapable Browsers 9-11
- Directing Links to a Frame 9-12
- Populating a Frame with `http.base` 9-13
- Summary 9-14
- Practice 9 Overview 9-15

## **Appendix A: Practice Solutions**

- Practice 1 Solutions A-2
- Practice 2 Solutions A-4
- Practice 3 Solutions A-5
- Practice 4 Solutions A-10
- Practice 5 Solutions A-13
- Practice 6 Solutions A-18
- Practice 7 Solutions A-19
- Practice 8 Solutions A-21
- Practice 9 Solutions A-24

## **Appendix B: PL/SQL Web Toolkit Reference**

---

## Preface

---





## Profile

### Before You Begin This Course

Before you begin this course, you should have the following qualifications:

- Thorough knowledge of SQL, PL/SQL, and SQL\*Plus or Procedure Builder
- Working experience with HTML and Web technology

### Prerequisites

- *Introduction to Oracle: SQL and PL/SQL* (ILT)
- *Develop PL/SQL Program Units* (ILT)

### Suggested Prerequisites

*iAS: Administration* (ILT)

### Related Media Courses

- *Oracle SQL and SQL\*Plus: Basic Select Statements* (CBT)
- *Oracle SQL and SQL\*Plus: DML and DDL* (CBT)
- *PL/SQL Basics* (CBT)

ILT=Instructor-Led Training, CBT=Computer-Based Training

### How This Course Is Organized

*9iAS: Develop Web-Based Applications with PL/SQL* is an instructor-led course featuring lecture and hands-on exercises. Online demonstrations and written practice sessions reinforce the concepts and skills introduced.

## **Related Publications**

- System release bulletins
- Installation and user's guides
- readme files
- International Oracle User's Group (IOUG) articles
- *Oracle Magazine*

## Typographic Conventions

### Typographic Conventions in Text

Convention	Element	Example
Bold italic	Glossary term (if there is a glossary)	The <b><i>algorithm</i></b> inserts the new key.
Caps and lowercase	Buttons, check boxes, triggers, windows	Click the Executable button. Select the Can't Delete Card check box. Assign a When-Validate-Item trigger to the ORD block. Open the Master Schedule window.
Courier new, case sensitive (default is lowercase)	Code output, directory names, filenames, passwords, pathnames, URLs, user input, usernames	Code output: <code>debug.set ( 'I', 300 );</code> Directory: <code>bin</code> (DOS), <code>\$FMHOME</code> (UNIX) Filename: Locate the <code>init.ora</code> file. Password: Use <code>tiger</code> as your password. Pathname: Open <code>c:\my_docs\projects</code> URL: Go to <code>http://www.oracle.com</code> User input: Enter <code>300</code> Username: Log on as <code>scott</code>
Initial cap	Graphics labels (unless the term is a proper noun)	Customer address ( <i>but</i> Oracle Payables)
Italic	Emphasized words and phrases, titles of books and courses, variables	Do <i>not</i> save changes to the database.  For further information, see <i>Oracle7 Server SQL Language Reference Manual</i> .  Enter <code>user_id@us.oracle.com</code> , where <i>user_id</i> is the name of the user.
Quotation marks	Interface elements with long names that have only initial caps; lesson and chapter titles in cross-references	Select "Include a reusable module component" and click Finish.  This subject is covered in Unit II, Lesson 3, "Working with Objects."
Uppercase	SQL column names, commands, functions, schemas, table names	Use the SELECT command to view information stored in the LAST_NAME column of the EMP table.

Convention	Element	Example
Arrow	Menu paths	Select File—> Save.
Brackets	Key names	Press [Enter].
Commas	Key sequences	Press and release keys one at a time: [Alternate], [F], [D]
Plus signs	Key combinations	Press and hold these keys simultaneously: [Ctrl]+[Alt]+[Del]

### Typographic Conventions in Code

Convention	Element	Example
Caps and lowercase	Oracle Forms triggers	When-Validate-Item
Lowercase	Column names, table names	SELECT last_name FROM s_emp;
	Passwords	DROP USER scott IDENTIFIED BY tiger;
	PL/SQL objects	OG_ACTIVATE_LAYER (OG_GET_LAYER ('prod_pie_layer'))
Lowercase italic	Syntax variables	CREATE ROLE <i>role</i>
Uppercase	SQL commands and functions	SELECT userid FROM emp;

# 1

## **Introduction to PL/SQL Web Applications**

Copyright © Oracle Corporation, 2001. All rights reserved.

**ORACLE®**

# Objectives

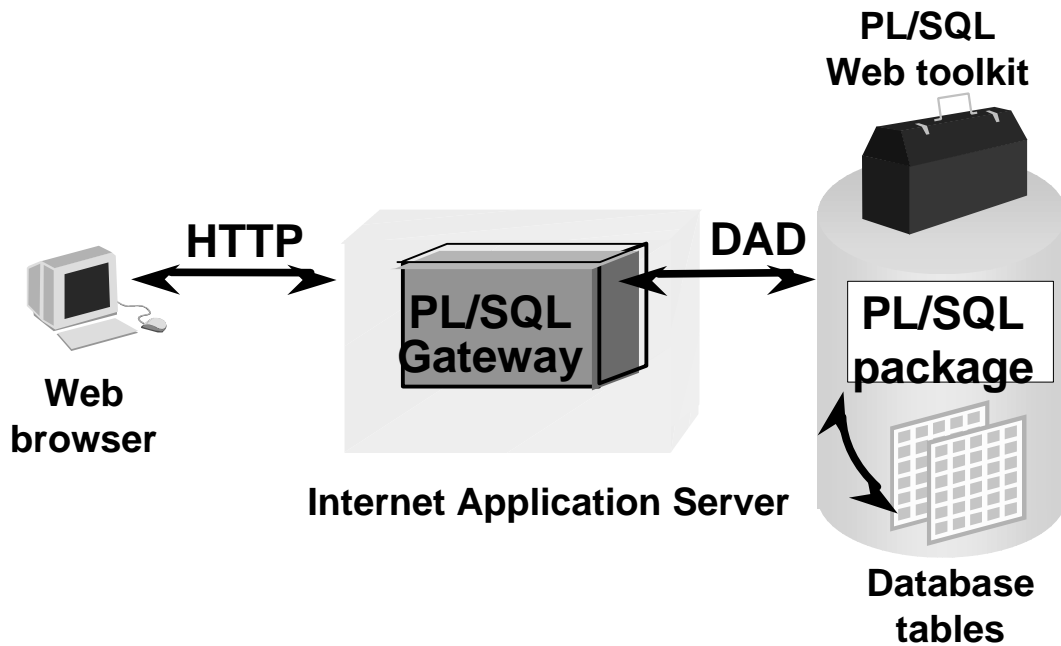
**After completing this lesson, you should be able to do the following:**

- **Identify the main Internet Application Server components used to build PL/SQL Web applications**
- **Use the course application to insert and display data**

## **Lesson Aim**

To build Web applications, you can use PL/SQL packages and other mechanisms provided by the Internet Application Server (*iAS*). This lesson describes the application-related components and introduces the application that you build during this course.

# Overview



1-3

Copyright © Oracle Corporation, 2001. All rights reserved.

ORACLE®

## Overview

### On the Client Tier

Clients access PL/SQL Web applications through a browser using the Web protocol HTTP. The application components generate HTML, which is returned to the browser and displayed as Web pages.

### On the Database Tier

PL/SQL Web applications are developed as PL/SQL packages and stored in an Oracle database. You can access database tables through these packages and present the data as dynamic information in your generated Web pages.

Oracle9 Internet Application Server (Oracle 9iAS) provides a collection of PL/SQL packages called the PL/SQL Web Toolkit. These packages are also stored in the database and are used by your application packages to generate Web page components and other related functionality.

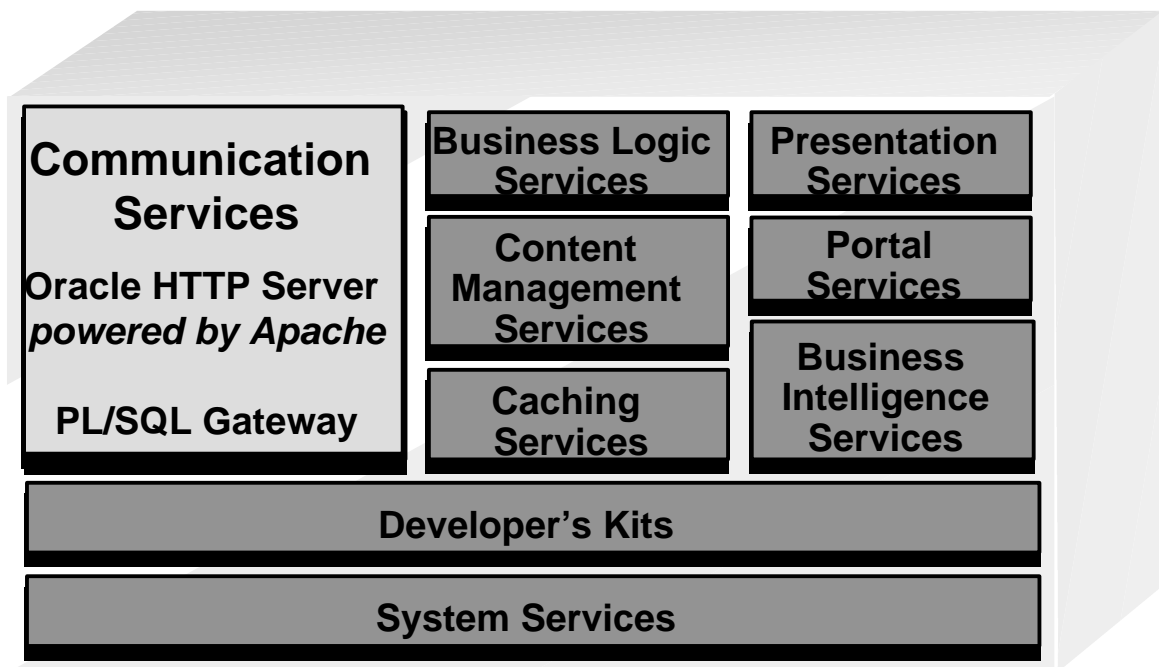
**Note:** By default, the PL/SQL Web toolkit is installed in the OAS\_PUBLIC schema.

### On the Application Server Tier: The PL/SQL Gateway

The PL/SQL Gateway enables you to call PL/SQL programs from a Web browser. The PL/SQL programs run on the server and return HTML to the browser.

The database access descriptor (DAD) is a mechanism defined through iAS that maps the PL/SQL Gateway user to the correct database schema. Once connected to the database, your PL/SQL code can access the database tables and call PL/SQL subprograms in the PL/SQL Web toolkit.

# iAS Architecture



1-4

Copyright © Oracle Corporation, 2001. All rights reserved.

ORACLE®

## Oracle 9iAS Architecture

Oracle9 Internet Application Server provides a set of services that manage the technological complexity of a complete middle-tier Internet infrastructure.

The *iAS* Communication Services include a number of modules that handle application modules written in a variety of languages, such as PERL and PL/SQL. This course concentrates on the PL/SQL Gateway module (also known as `mod_plsql`).

- The Oracle HTTP Server *powered by Apache* provides fast, reliable Web listening services that make your Web content accessible to your users.
- The PL/SQL Gateway enables PL/SQL applications to produce your Web content.

The *iAS* Presentation Services include Oracle PL/SQL Server Pages (Oracle PSP), a translator and run time environment. Oracle PSP is implemented in PL/SQL and runs in either the Oracle database server or the Oracle9i Application Server. You can write an HTML page with embedded PL/SQL code and compile it as a PL/SQL server page. The PL/SQL server page can call procedures from the PL/SQL Web Toolkit. The PL/SQL Gateway can access the PSP procedure.



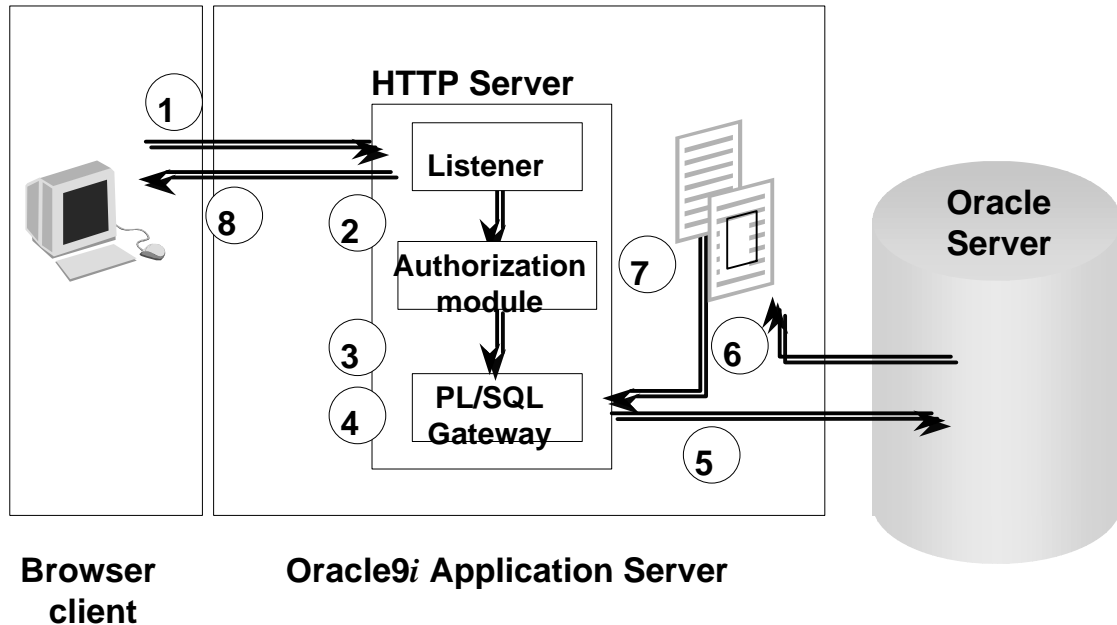
# Benefits of the PL/SQL Gateway

- **Efficiency**
- **Security**
- **Reusability**
- **Productivity**

## **The PL/SQL Gateway**

- The PL/SQL Gateway is the most efficient way to generate HTML from SQL.
- The PL/SQL Gateway provides data security: users need access to procedures but not to tables
- The PL/SQL Gateway enables you to reuse existing code and in-house PL/SQL skills.
- The PL/SQL Gateway increases productivity: Oracle Portal and Oracle Designer have PL/SQL generators; if you have a lot of HTML, you can use Oracle PL/SQL Server Pages to develop dynamic content rapidly. If you have a large body of PL/SQL code that produces formatted output, you can change the print statements to call procedures of the PL/SQL Web Toolkit.

# The Flow of HTTP Requests



1-6

Copyright © Oracle Corporation, 2001. All rights reserved.

ORACLE®

## Communication Flow: The Path of HTTP Requests

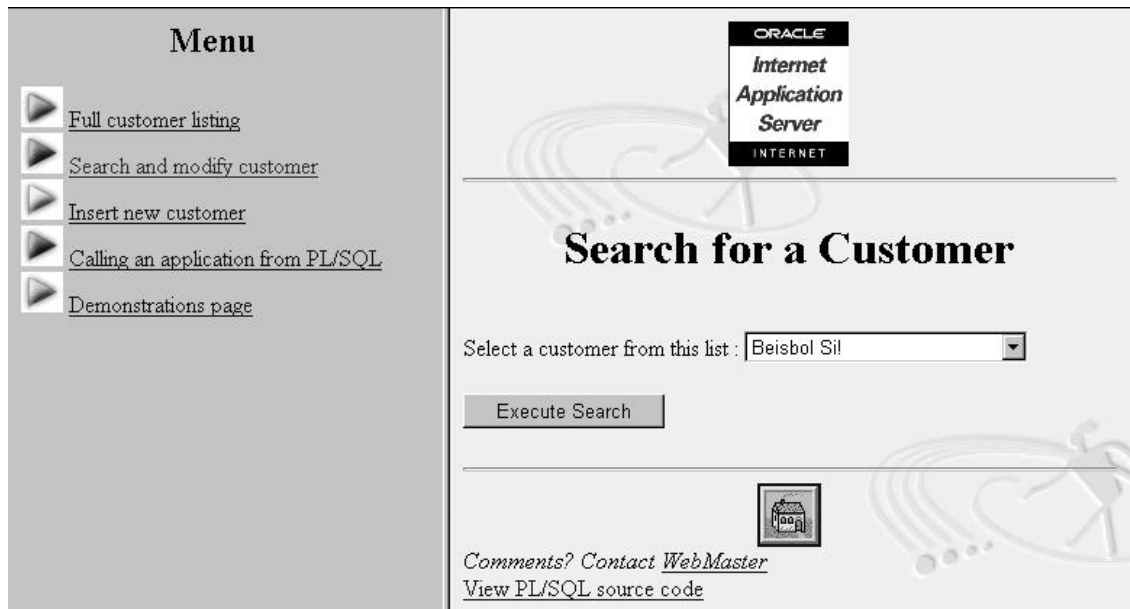
1. The browser sends a URL to the listener. The listener examines the URL and determines that the request is for the PL/SQL Gateway.
2. If authentication is required, the listener contacts an authorization module.
3. The authorization module validates the request and returns the result to the PL/SQL Gateway.
4. The PL/SQL Gateway uses the database access descriptor (DAD) to connect to the database.
5. The PL/SQL Gateway connects to the database, prepares the call parameters, and invokes the named PL/SQL procedure.
6. The PL/SQL procedure generates an HTML page, which can include dynamic data accessed from tables in the database as well as static data.
7. The output from the procedure is returned by way of the response buffer to the PL/SQL Gateway.
8. The Oracle HTTP Server sends the response back to the client.

# Course Application

- **Maintains customer and employee information**
- **Content:**
  - **Data stored in the database**
  - **Images stored in the file system**
  - **HTML pages**
- **PL/SQL packages use the PL/SQL Web toolkit to generate HTML on demand**

# Course Application

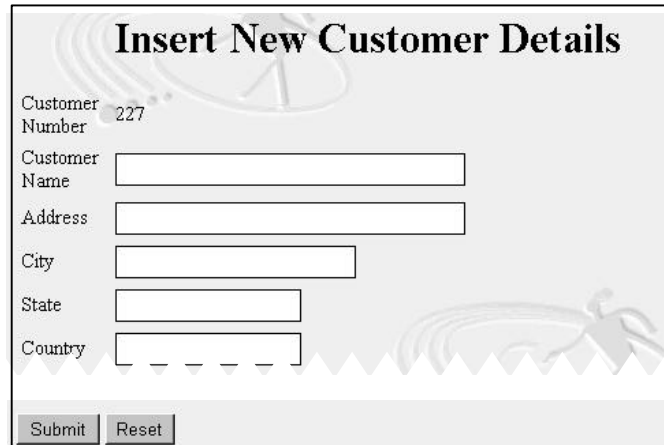
- Separate frames contain independent information
- Links change main frame contents



# Course Application

## Fill-in form:

- Text, numeric, and pop-up list fields
- Dynamic retrieval of results



The screenshot shows a web form titled "Insert New Customer Details". The form contains the following fields and controls:

- Customer Number:** A text field containing the value "227".
- Customer Name:** A text input field.
- Address:** A text input field.
- City:** A text input field.
- State:** A text input field.
- Country:** A text input field.
- Buttons:** At the bottom of the form are two buttons labeled "Submit" and "Reset".

The background of the form has a faint, stylized graphic of a person running.

# Summary

- **PL/SQL Web applications:**
  - Are defined as database packages
  - Use the PL/SQL Web toolkit to generate HTML
  - Are registered with Internet Application Server
  - Are controlled by the PL/SQL Gateway at run time
- **The course application maintains customer information**

# Practice 1 Overview

**This practice covers the following topics:**

- **Previewing the completed course application**
- **Entering data into the course application**

## **Practice 1 Overview**

For solutions to this practice, see Appendix A, “Practice Solutions.”

## Practice 1

Your instructor will provide you with login information. Make a note of this information here. You will need to refer to it throughout this course:

1. Database (username/password@connect\_string)
  - username:
  - password:
  - connect string:
2. UNIX server (or WinNT server)
  - hostname or IP:
  - username:
  - password:
  - \$HOME:
3. Client machine
  - username:
  - domain:
  - password:
4. iAS Administration pages
  - URL: `http://`  
(Usually `http://hostname.domain/pls/dad/admin_/gateway.htm`)
  - username:
  - password:
5. URLs for your application
  - Network port:
  - PL/SQL virtual path:  
(Usually `pls/ias_dad`)
  - Home page:  
(Usually `http://hostname.domain/index.html`)
6. URLs for the course applications
  - Summit application: `http://`  
(Usually `http://hostname.domain/pls/ias_dad/summit.main_page`)
  - Practice application: `http://`  
(Usually `http://hostname.domain/pls/ias_dad/lab.main_page`)



### Practice 1 (continued)

7. Use the URL provided in Question 6 to access the Summit demonstration application. Then perform the following tasks to familiarize yourself with the application:
  - a. View a listing of all customers.
  - b. Beisbol Si!, customer number 209, has just been taken over by a major multinational company. Change the telephone number to 447-9967 and add a comment: Pending new credit details no more orders are to be taken for this customer.
  - c. Add your own company to the customer listings.
  - d. Navigate to the frame-based version of the application. View a list of all customers from the frames version of the application.



# 2

## Creating a Web Page with HTML

Copyright © Oracle Corporation, 2001. All rights reserved.

**ORACLE®**

# Objectives

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

- **Identify the main components of a URL**
- **Create a simple home page using HTML tags**

## Lesson Aim

To format and display Web pages, applications must generate instructions in Hypertext Markup Language (HTML). A basic understanding of HTML is needed to help you use the related mechanisms and PL/SQL packages of Internet Application Server.

This lesson covers the basics of creating a static Web page.

For more information on writing HTML, see:

- *NCSA Beginner's Guide to HTML* at  
<http://www.ncsa.uiuc.edu/General/Internet/WWW/HTMLPrimer.html>
- *The Official HTML 4 Reference Specification* at  
<http://www.w3.org/TR/REC-html40/>

# Overview

- **Uniform resource locators (URLs)**
- **Elements of a Web page**
- **HTML body tags: Headings, lists, formatting**
- **Creating document links**
- **Adding in-line images**
- **Setting up a home page**
- **General guidelines**

# Uniform Resource Locator (URL)

***service://server:port/application***

- **Required components: server name**
- **Default service: HTTP**
- **Purposes:**
  - **Link to a page or a location within the same document**
  - **Download files**
  - **Send e-mail messages**

## Uniform Resource Locator (URL)

The URL is the address of a resource on the Web. It is made up of a number of components:

- The service indicates the protocol used to access information. To access a Web page, you use the HTTP or HTTPS services.
- The server name is the host (domain) name, or the IP address, of a machine connected to the network.
- The port is a number that identifies a communication channel associated with a specific program. The default HTTP port is 80.
- The application identifies the document to be served to the client.

### Example

`http://www.oracle.com/index.html`

# HTML Tags

- Mark the document elements for display
- Have this format: `<tag_name>`
- May be paired:  
`<H1>some_text</H1>`
- Contain attributes: Alignment, size, width, height
- May not be supported by all Web browsers

## HTML Tags

An element is a fundamental component of the structure of a text document. Some examples of elements are heads, tables, paragraphs, and lists. You use HTML tags to mark the elements of a file for your browser. An element may include other elements.

To denote the various elements in an HTML document, you use tags. HTML tags consist of a left angle bracket (<), a tag name, and a right angle bracket (>). Tags are usually paired (for example, <H1> and </H1>) to start and end the instruction. The end tag looks just like the start tag except that a slash (/) precedes the tag name.

Some elements may include attributes, in the form of additional information that is included inside the start tag. For example, you can specify the background color for a Web page using the BGCOLOR attribute of the body tag:

```
<BODY BGCOLOR="AQUA">
```

Not all tags are supported by all Web browsers. If a browser does not support a tag, it usually just ignores it.

**Note:** HTML is not case sensitive; <title> is equivalent to <TITLE> or <TiTIE>.

# Basic Elements of a Web Page

Required tags for a basic Web page:

<b>&lt;HTML&gt;</b>	<b>&lt;/HTML&gt;</b>
<b>&lt;HEAD&gt;</b>	<b>&lt;/HEAD&gt;</b>
<b>&lt;TITLE&gt;</b>	<b>&lt;/TITLE&gt;</b>
<b>&lt;BODY&gt;</b>	<b>&lt;/BODY&gt;</b>

## Basic Elements of a Web Page

The following tag pairs are used to construct a basic Web page:

<b>&lt;HTML&gt;</b>	<b>&lt;/HTML&gt;</b>
<b>&lt;HEAD&gt;</b>	<b>&lt;/HEAD&gt;</b>
<b>&lt;TITLE&gt;</b>	<b>&lt;/TITLE&gt;</b>
<b>&lt;BODY&gt;</b>	<b>&lt;/BODY&gt;</b>

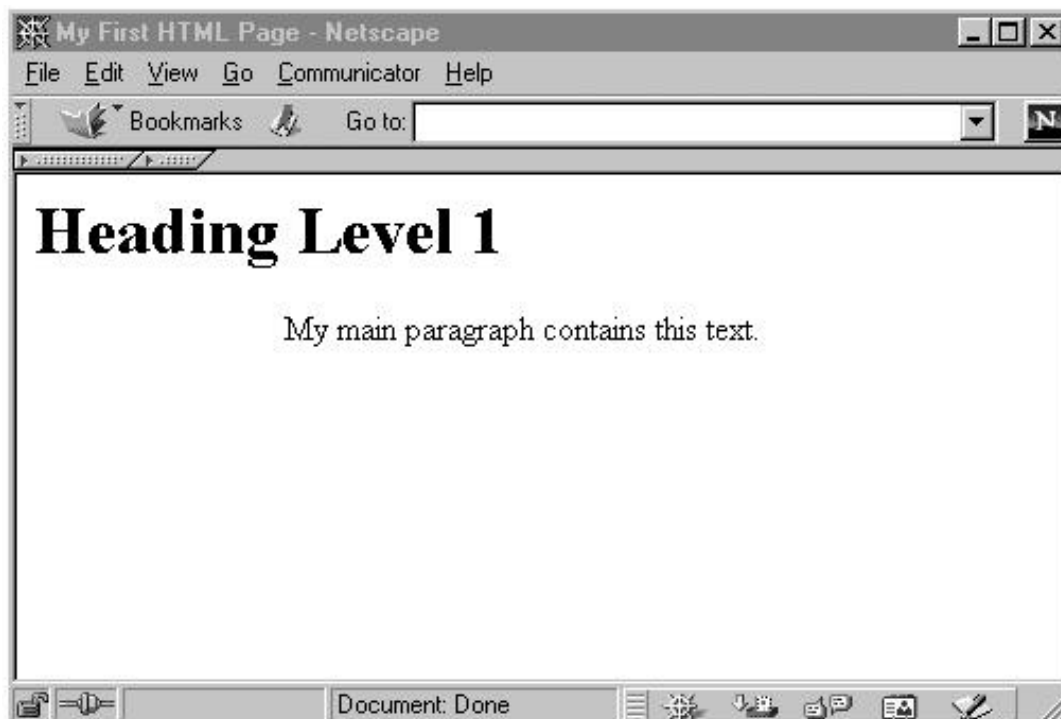
- The head section contains HEAD and TITLE elements.
- The body section contains paragraphs, lists, links, and other elements. Graphic elements consist of background images, in-line images, image maps, icons, and links.



## Basic HTML: Example Code

```
<HTML>
<HEAD>
<TITLE>My First HTML Page</TITLE>
</HEAD>
<!--This is the body of the page-->
<BODY>
<H1>Heading Level 1</H1>
<P ALIGN="CENTER">My main paragraph
contains this text.</P>
</BODY>
</HTML>
```

## Basic HTML: Example Output



## Block Element Tags

Opening	Closing	Definition
<b>&lt;P&gt;</b>	<b>&lt;/P&gt;</b>	<b>Paragraph delimiter</b>
<b>&lt;PRE&gt;</b>	<b>&lt;/PRE&gt;</b>	<b>Preformatted text</b>
<b>&lt;BR&gt;</b>	<b>N/A</b>	<b>Line break</b>
<b>&lt;CENTER&gt;</b>	<b>&lt;/CENTER&gt;</b>	<b>Center tool</b>
<b>&lt;HR&gt;</b>	<b>N/A</b>	<b>Horizontal rule</b>
<b>&lt;FORM&gt;</b>	<b>&lt;/FORM&gt;</b>	<b>HTML form</b>
<b>&lt;TABLE&gt;</b>	<b>&lt;/TABLE&gt;</b>	<b>HTML table</b>
<b>&lt;UL&gt;</b>	<b>&lt;/UL&gt;</b>	<b>Unordered list</b>
<b>&lt;OL&gt;</b>	<b>&lt;/OL&gt;</b>	<b>Ordered list</b>
<b>&lt;DL&gt;</b>	<b>&lt;/DL&gt;</b>	<b>Descriptive list</b>

### Meta Tags

The tag that expires the page from the browser cache is:

`<META HTTP-EQUIV="Pragma" CONTENT="no-cache">`

The refresh tag refreshes the same page (no URL, only time in seconds):

`<META HTTP-EQUIV="Refresh" CONTENT="30">`

# List Tags

## Ordered

```
<OL TYPE=i>
<LI>First item</LI>
<LI>Second item</LI>
</OL>
```

## Unordered

```
<UL>
<LI>First item</LI>
<LI>Second item</LI>
</UL>
```

### Attributes for Ordered Lists

- The `type` attribute determines the numbering style.

Type	Numbering style	Example
1	Arabic numbers	1, 2, 3, ...
a	Lower alphabetic	a, b, c, ...
A	Upper alphabetic	A, B, C, ...
i	Lower roman numerals	i, ii, iii, ...
I	Upper roman numerals	I, II, III, ...

- The `start` attribute indicates the starting number.
- The `compact` attribute reduces the inter-item spacing.

### Attributes for Unordered Lists

- The `type` attribute determines the bullet style: disc, circle, or square.
- The `compact` attribute reduces the inter-item spacing.

## Formatting for Special Characters

- **HTML special characters:**

<	&lt;
>	&gt;
&	&amp;

- **Other special characters:**

ö	&ouml;
ñ	&ntilde;
È	&Egrave;

### Special Characters

Special characters are defined by case-sensitive escape sequences: &LT; is not equivalent to &lt; .

- Copyright (©):               &#169;
- Double quotation mark ("): &#34; or &quot;
- Registered trademark (®): &#174;
- Degree sign (°):             &#176;
- Plus or minus (±):           &#177;
- Pounds sterling (£):         &#163;
- Yen sign (¥):                &#165;

For more information on the HTML coded character set, see:

<http://www.w3.org/TR/REC-html40/charset.html>

## Creating Document Links

- **Link to your Web pages:**

- **Create a target:**

```
<A NAME="top">Introduction</A>
```

- **Create a link:**

```
<A HREF="index.html#top">  
Top of page</A>
```

- **Link to other documents using URLs:**

```
<A HREF="http://www.oracle.com">  
Oracle Home Page</A>
```

### Creating Document Links

Embedded URLs create a link from text on the screen to a document (or URL).

- The browser highlights this text, usually with color and underlining.
- Most browsers also display the URL on the status line when the cursor moves over the text.
- When the user selects this text, the browser submits the new URL to the specified server to request a new file.

The application developer can embed an absolute URL (containing the entire URL, including the protocol and machine name) or a relative URL (containing only a trailing portion of the URL).

The anchor element specifies the link or creates a named anchor.

## Adding In-Line Images

- Place an image in the HTML page:

```
<IMG SRC="/gif/oracle_logo.gif"  
  ALT="Oracle Logo" ALIGN="BOTTOM"  
  BORDER=0 WIDTH=240 HEIGHT=38>
```

- Create a link from the image:

```
<A HREF="http://www.oracle.com">  
  <IMG SRC="/gif/oracle_logo.gif"  
    ALT="Oracle Logo" ALIGN="BOTTOM"  
    BORDER=0 WIDTH=240 HEIGHT=38>  
</A>
```

### Embedding Images in HTML Documents

Most browsers can handle Graphics Interchange Format (GIF) and Joint Photographics Experts Group (JPEG) images.

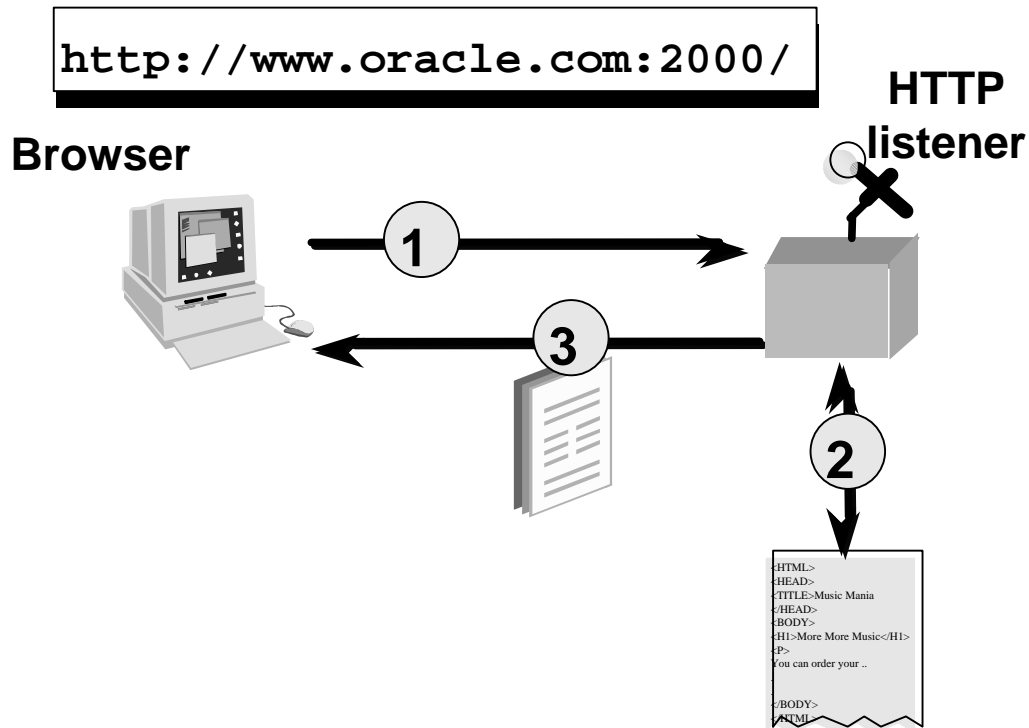
To embed an image in a page, use the <IMG> tag. It is good practice to use the ALT attribute to specify text for browsers that do not display images. The SRC attribute specifies the URL that points to the file that contains the image. For example:

```
<IMG SRC="image_name.ext" ALT="text" ALIGN=alignment HEIGHT=value  
  WIDTH=value>
```

where alignment can be TOP, MIDDLE, BOTTOM, LEFT, or RIGHT.

The URL pointing to the image can be absolute or relative. Relative URLs are more common for image files, because images usually exist on the same server as the HTML file in which they are embedded. However, if you specify an absolute URL, you can embed images that exist on a different server. When a Web client processes an HTML file that contains the appropriate image markup, the client establishes a new connection to the specified server and automatically downloads the image file.

## Retrieving the Default Web Page



2-14

Copyright © Oracle Corporation, 2001. All rights reserved.

ORACLE®

### Retrieving the Default Web Page

1. A user with a Web browser submits a request without specifying a filename.
2. The HTTP listener finds out the name of the initial file from the Apache HTTP server configuration settings (the value of `DirectoryIndex`). In this instance, the initial file is set to `wwwIndex.html`, so a request for `http://www.oracle.com:2000` returns the file `http://www.oracle.com:2000/wwwIndex.html`.
3. The HTTP listener also checks the directory mapping for the virtual path used in the request. In this case, the user specified the virtual path `"/`, which is also known as the Document Root. If the Document Root is mapped in the listener to the physical path `/home/ias1/` then the listener returns the file `/home/ias1/wwwIndex.html`.
4. The Web browser receives the document. If the document is HTML, then the browser interprets the tags and displays them accordingly.

Note: The default page is usually called `index.html`.



# Summary

- **Identify the main components of a URL**
- **Define HyperText Markup Language (HTML)**
- **Identify the basic HTML tags and attributes**

## Practice 2 Overview

**This practice covers the following topics:**

- **Creating a home page (`index.html`) with HTML**
- **Including a link to the *iAS* administration page**

### **Practice 2 Overview**

For solutions to this practice, see Appendix A, “Practice Solutions.”

## Practice 2

In this practice you create your application's home page in HTML.

1. Copy the `index.html` file that is located in the `labsPLSQL` directory to your home directory. Modify the file. You can be as creative as you want because the aim of this exercise is to ensure that you are familiar with HTML.
2. Use the anchor tag, `<A>`, to create a link to the 9iAS Administrative pages. See Practice 1, Question 4 for the URL.
3. Create a link on your home page to the finished application. See Practice 1, Question 6 for the URL.
4. Create a link on your home page to the application you will build during the course. See Practice 1, Question 6 for the URL.
5. Open your browser and test your application home page by issuing the following URL (see Practice 1, Question 5):  
`http://hostname.domain:your_port_number/`



# 3

## Using the PL/SQL Web Toolkit

Copyright © Oracle Corporation, 2001. All rights reserved.

**ORACLE®**

# Objectives

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

- **Create a PL/SQL package containing Web page elements**
- **Recreate the home page using the procedures and functions of the PL/SQL Web toolkit**
- **Select data into a page**

## **Lesson Aim**

The PL/SQL Web toolkit enables you to generate HTML pages by writing application components in PL/SQL. This lesson shows you how to structure such components and discusses the PL/SQL procedures and functions that enable you to create Web pages.

# Overview

- **Writing PL/SQL procedures**
- **Passing parameters to procedures**
- **Creating a basic page with HTP and HTF**
- **Accessing a procedure through a URL**
- **Building a package with a reusable header and footer**
- **Creating a dynamic page**

# PL/SQL Package Syntax

**Package  
specification**

Public

Procedure A  
Declaration

**Package  
body**

Private

Procedure B  
Definition (Private)  
Function C  
Definition (Private)

Procedure A  
Definition (Public)

3-4

Copyright © Oracle Corporation, 2001. All rights reserved.

ORACLE®

## Review of PL/SQL Package Syntax

CREATE PACKAGE *name* AS—specification (visible part)  
public type and object declarations  
subprogram specifications  
END [*name*];

CREATE PACKAGE BODY *name* AS—body (hidden part)  
private type and object declarations  
subprogram bodies  
[BEGIN  
initialization statements]  
END [*name*];

- Public subcomponents are declared in the package specification and can be referenced from any Oracle server environment as `packageA.procX`.
- Private subcomponents are declared and defined in the package body. They can be referenced only by other constructs that are part of the same package.
- Keeping a component private hides the implementation from external sources. When designing Web packages, remember that if an HTML form needs to call a component, then that component must be declared as public.



# Passing Parameters to a Procedure

```
PROCEDURE test (a IN NUMBER DEFAULT NULL,  
                b IN NUMBER DEFAULT NULL,  
                c IN NUMBER DEFAULT NULL) IS ...
```

- **Positional notation**

```
test;  
test(1);  
test(1, 2);  
test(1, 2, 3);  
test(1, NULL, 3);
```

- **Association operator**

```
test(1,c=>3);
```

3-5

Copyright © Oracle Corporation, 2001. All rights reserved.

ORACLE®

## Passing Parameters to Procedures

A procedure defined as:

```
PROCEDURE test (a IN NUMBER DEFAULT NULL,  
                b IN NUMBER DEFAULT NULL,  
                c IN NUMBER DEFAULT NULL) IS
```

can be called using:

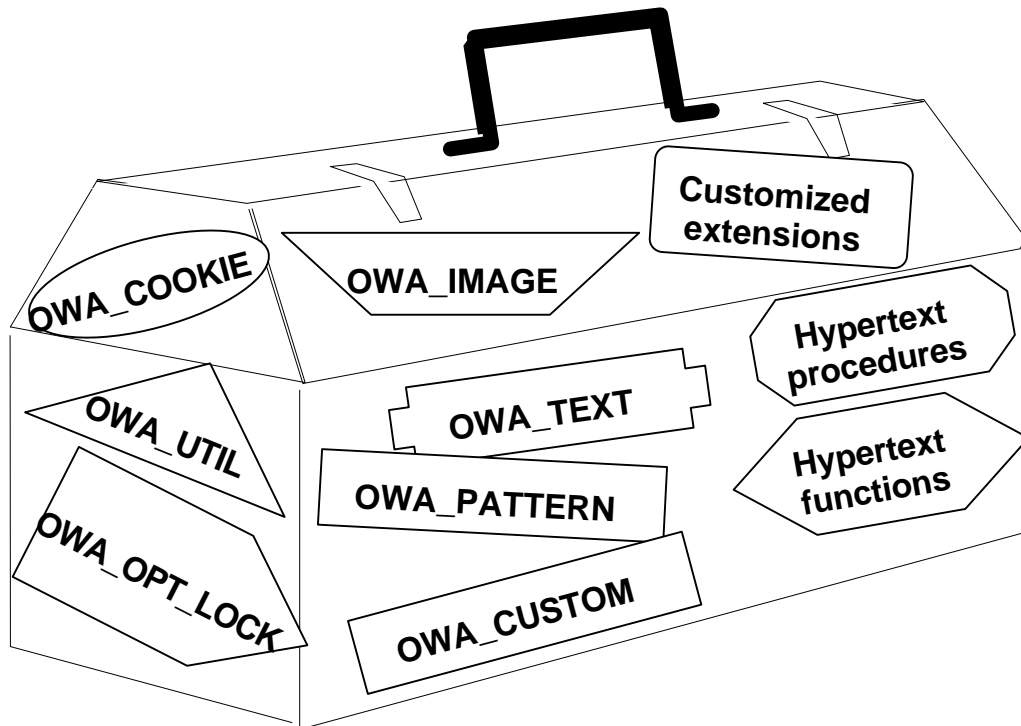
- test;
- test(1);
- test(1,2);
- test(1,2,3);

You cannot call it using test(1,,3);

If you want to supply only the first and third parameters, you need to pass NULL as the middle argument or use the association operator:

- test(1,NULL,3);
- test(1,c=>3);

# The PL/SQL Web Toolkit



3-6

Copyright © Oracle Corporation, 2001. All rights reserved.

ORACLE®

## The PL/SQL Web Toolkit

Before you can use the PL/SQL Gateway, you need to load packages that define data types, functions, and procedures for it. The HTP and HTF packages take care of the most time-consuming and error-prone activity: generating the proper HTML tags around variables. For most HTP procedures, there is a corresponding HTF function with identical parameters.

The PL/SQL Web toolkit contains the following packages in addition to the HTP and HTF packages:

- owa\_cookie: Enables you to send and retrieve cookies in HTTP headers
- owa\_custom: Contains the authorize function and the time zone constants used by cookies
- owa\_image: Enables you to create an image map that invokes a PL/SQL procedure
- owa\_opt\_lock: Implements an optimistic locking scheme that you can use in your stored procedure to check whether the row that the user wants to update was changed by someone else
- owa\_pattern: Enables you to locate text patterns within strings and replace the matched string with another string; you can use regular expressions with the subprograms in this package
- owa\_sec: Contains subprograms used by the gateway for authenticating requests
- owa\_text: Contains subprograms used by owa\_pattern for manipulating strings; they are externalized so that you can use them directly
- owa\_util: Contains subprograms for performing operations such as getting the value of CGI environment variables and printing the results of a query in an HTML table

## Benefits of the PL/SQL Web Toolkit

- **Integrate SQL and HTML**
- **Use existing PL/SQL expertise**
- **Create correct HTML more easily**
- **Use the same packages in other tools**

### Benefits of the Toolkit

It is easier to create the correct HTML using the packages because the default parameters define the most common values for the HTML tags.

Oracle Portal and the Web PL/SQL Generator of Designer 6i also use the PL/SQL Web Toolkit.

Note: You can find documentation about the PL/SQL Web Toolkit on the Oracle Technology Network at [www.technet.oracle.com](http://www.technet.oracle.com).

# HTP and HTF Packages

**Low-level, one-to-one mapping between HTML tags and PL/SQL procedures and functions:**

- **Structure tags**
- **Header tags**
- **Body tags**
- **Print procedures**
- **Form tags**
- **Anchor tags**

## Generating HTML Using HTP

An HTP procedure generates a line of HTML that contains the HTML tag corresponding to its name.

```
http.header(1,'Overview');  
http.print('Hello World');  
http.para;  
http.bold('Life is Great');
```

Procedure Call

```
<H1>Overview</H1>  
Hello World  
<P>  
<B>Life is Great</B>
```

Generated HTML

## Generating HTML Using HTF

- An HTF function returns an HTML tag that corresponds to its name.
- The HTF result must be passed to HTP for output to a browser.

```
http.listItem(htf.italic('Company: ')||  
             'Oracle, Sun, Netscape');
```

Procedure Call

```
<LI><I>Company: </I>Oracle, Sun, Netscape </LI>
```

Generated HTML

## Passing Parameters to a Toolkit Procedure

- The first letter indicates the data type:
  - cname IN VARCHAR2
  - nsize IN INTEGER
  - dbuf IN DATE
- Many parameters have a default value
- The cattributes parameter passes exact text:

```
http.bodyOpen(cattributes =>'BGCOLOR="#FFFFFF"')
```

### Passing Parameters to a Toolkit Procedure

The http.header procedure has the following parameters:

```
nsize      IN INTEGER
cheader    IN VARCHAR2
calign     IN VARCHAR2 DEFAULT NULL
cnowrap    IN VARCHAR2 DEFAULT NULL
cclear     IN VARCHAR2 DEFAULT NULL
cattributes IN VARCHAR2 DEFAULT NULL
```

Default values are provided for many of the parameters. This means that you need not specify them; if you do not supply a value for the parameter, the default value is used. As a minimum, you need to provide values for the first two parameters when you call http.header:

```
http.header(1, 'iAS Application');
```

You may supply values for the other parameters using positional notation or the association operator. The following two statements are equivalent:

```
http.header(1, 'iAS Application', NULL, NULL, 'YES');
http.header(1, 'iAS Application', cclear=>'YES');
```

## Creating a Basic Page

<pre>CREATE PACKAGE basic_http AS   PROCEDURE show; END basic_http;  CREATE PACKAGE BODY basic_http AS   . . .   PROCEDURE show   BEGIN     header;     body;     footer;   END show;   . . . END basic_http;</pre>	<pre>&lt;HTML&gt; &lt;HEAD&gt;&lt;TITLE&gt;Sample Page&lt;/TITLE&gt; &lt;BODY bgcolor="#000000"&gt;   &lt;H1&gt;Heading Level 1   &lt;HR&gt;   Sample text   &lt;HR&gt;   &lt;ADDRESS&gt;&amp;#169Oracle Education&lt;/ADDRESS&gt; &lt;/BODY&gt; &lt;/HTML&gt;</pre>
---	--

**Output**

**Generated HTML**

3-12

Copyright © Oracle Corporation, 2001. All rights reserved.

ORACLE®

### Creating a Basic Page

When creating a basic Web page in PL/SQL, you define a package specification and a package body. Basic elements of the Web page that are in the package body include:

- A header that starts the HTML and gives a title to the page
- A body that displays data and graphics on the page
- A footer that identifies the author and closes the HTML

To execute the combined elements, create a procedure in the package body, such as `show`, which also is listed in the package specification.

The header and footer elements are often the same throughout an application. Consider creating them as generic elements in a reusable package.



## Web Page Code Example

```
CREATE OR REPLACE PACKAGE basic_http AS
    PROCEDURE show;
END basic_http;

/

CREATE OR REPLACE PACKAGE BODY basic_http AS
    PROCEDURE header IS
    BEGIN
        http.htmlOpen;
        http.headOpen;
        http.title ('Sample Page');
        http.headClose;
        http.comment ('This is a comment');
    END header;

    PROCEDURE body IS
    BEGIN
        http.bodyOpen(attributes=>'bgcolor="#00FFFF"');
        http.hr;
        http.header (1,'Heading Level 1');
        http.print ('Sample text');
    END body;

    PROCEDURE footer IS
    BEGIN
        http.hr;
        http.address ('&#169;Oracle Education');
        http.bodyClose;
        http.htmlClose;
    END footer;

    PROCEDURE show IS
    BEGIN
        header;
        body;
        footer;
    END show;
END basic_http;

/
```

## Calling a Procedure from a URL

`http://servername:2000/pls/ias_dad/basic_http.show`

<b>host</b>	<b>The name of the machine where the application server is running</b>
<b>port</b>	<b>The number of the port where the application server listens</b>
<b>pls</b>	<b>The indicator that invokes the PL/SQL Gateway</b>
<b>dad</b>	<b>The database access descriptor (DAD)</b>
<b>package</b>	<b>The name of the package containing the procedure</b>
<b>proc</b>	<b>The stored procedure to be run</b>
<b>QUERY_STRING</b>	<b>Parameters for the stored procedure</b>

### Calling a Procedure from a URL

To invoke a PL/SQL procedure using the Gateway, the URL must be in the following format:  
`http://host[:port]/pls/dad/[package.]proc_name[?QUERY_STRING]`

- The `host` specifies the machine (server) where the application server is running.
- The (optional) `port` specifies the port at which the application server is listening. If this specification is omitted, port 80 is assumed.
- `pls` specifies that the call is directed to the PL/SQL Gateway. This is the default value but you can specify a different value for `<Location <MOUNT_PATH>>` in the `plssql.conf` file.
- The `dad` specifies the database access descriptor that enables a connection to the database.
- The (optional) `package` specifies the name of the package that contains the procedure. If this specification is omitted, the procedure stands alone.
- The `proc` entry specifies the stored procedure to be run. This must be a procedure and not a function.
- The `QUERY_STRING` entry specifies parameters for the stored procedure.

#### Example

`http://serv1/pls/dad/browse.user_tnv?cuser=STUDENT&ntable_type=1`

# Building a Generic HTML Document Header

```
-- Generic document header in the package body
PROCEDURE header
  (p_title IN VARCHAR2 DEFAULT NULL) IS
BEGIN
  http.htmlOpen;
  http.headOpen;
  http.title(p_title);
  http.headClose;
END header;
```

## Building a Generic HTML Document Header

The `http.htmlOpen`, `http.headOpen`, and `http.headClose` procedures do not accept any parameters. They generate `<HTML>`, `<HEAD>`, and `</HEAD>`, respectively.

### Syntax

```
http.title (ctitle);
```

### Parameters

```
ctitle IN VARCHAR2
```

### Generated Code

```
<TITLE>ctitle</TITLE>
```

## Building a Basic Document Body

```
-- Basic body procedure in the package body
PROCEDURE body IS
BEGIN
    http.bodyOpen(cattributes =>
        'BGOLOR="00FFFF"');
    http.header (1,'Heading Level 1');
    http.paragraph ('CENTER');
    http.print ('My paragraph');
END body;
```

### Building a Generic HTML Document Body

#### Syntax

```
http.header (nsize, cheader, calign, cnowrap, cclear, cattributes);
http.paragraph (calign, cnowrap, cclear, cattributes);
```

#### Parameters

nsize	IN INTEGER
cheader	IN VARCHAR2
calign	IN VARCHAR2 DEFAULT NULL
cnowrap	IN VARCHAR2 DEFAULT NULL
cclear	IN VARCHAR2 DEFAULT NULL
cattributes	IN VARCHAR2 DEFAULT NULL

Values for nsize are 1 through 6. Values for calign are LEFT, RIGHT, and CENTER. Values for cclear and cnowrap are either NULL or any value.

#### Generated Code

```
<Hnsize ALIGN="calign" NOWRAP CLEAR="cclear" cattributes>cheader
</Hnsize>
<P ALIGN="calign" NOWRAP CLEAR="cclear" cattributes>
```

Each returned value is enclosed in double quotation marks (").

# Building a Generic HTML Document Footer

```
-- Example document footer in the package body
PROCEDURE footer IS
BEGIN
    http.line;
    http.address('Comments ? ' ||
        htf.anchor('mailto:admin@oracle.com','Admin'));
    http.anchor('/', 'Home Page');
    http.bodyClose ;
    http.htmlClose ;
END footer ;
```

## Building a Generic HTML Document Footer

The `http.htmlClose` and `http.bodyClose` procedures do not accept any parameters. They generate `</HTML>` and `</BODY>`, respectively.

### Syntax

```
http.line (cclear, csrc, cattributes);
http.address(cvalue, cnowrap, cclear, cattributes);
```

### Parameters

<code>cclear</code>	IN VARCHAR2 DEFAULT NULL
<code>cvalue</code>	IN VARCHAR2
<code>cnowrap</code>	IN VARCHAR2
<code>csrc</code>	IN VARCHAR2 DEFAULT NULL
<code>cattributes</code>	IN VARCHAR2 DEFAULT NULL

Values for `cclear` and `cnowrap` are either NULL or any value. You can represent the line as an image by specifying an image for `csrc`.

### Generated Code

```
<HR CLEAR="cclear" SRC="csrc" cattributes>
<ADDRESS CLEAR="cclear" NOWRAP cattributes>cvalue </ADDRESS>
```

# Creating a Document Link

```
-- Sample hypertext link
http.anchor('http://host/pls/dad/lab.main_page',
            'iAS Application Menu');

-- Sample graphic link
http.anchor('http://www.oracle.com',
            htf.img('/img/oraonweb.gif',null,
                    'Oracle Home',
                    cattributes=>'WIDTH=33 HEIGHT=33'));
```

## Anchor Element

### Syntax

```
http.anchor (curl, ctext, cname, cattributes);
```

### Parameters

curl	IN VARCHAR2
ctext	IN VARCHAR2
cname	IN VARCHAR2 DEFAULT NULL
cattributes	IN VARCHAR2 DEFAULT NULL

You must specify either `curl` or `cname`. The visible text or graphical link is designated by the value of `ctext` and the anchor's target name is designated by the value of `cname`.

### Generated Code

```
<A HREF="curl" NAME="cname" cattributes>ctext</A>
```

The `HREF` attribute specifies the URL when the link is selected. The `NAME` attribute specifies a named reference for a specific section in a document.

## Image Element

### Syntax

```
http.img (curl, calign, calt, cismap, cattributes);
```

### Parameters

curl	IN VARCHAR2 DEFAULT NULL
calign	IN VARCHAR2 DEFAULT NULL
calt	IN VARCHAR2 DEFAULT NULL
cismap	IN VARCHAR2 DEFAULT NULL
cattributes	IN VARCHAR2 DEFAULT NULL

### Generated Code

```
<IMG SRC="curl" ALIGN="calign" ALT="calt" ISMAP cattributes>
```

The SRC attribute specifies the URL that identifies the image. The ALIGN attribute specifies the alignment of text in relation to the image. The cismap parameter indicates that the image is an image map.

## Including Dynamic Data in an HTML Page

```
PROCEDURE including_data IS
  CURSOR employee IS
    SELECT first_name, last_name FROM s_emp;
BEGIN
  header;
  FOR current_employee IN employee LOOP
    http.Bold(current_employee.first_name||' '||
              current_employee.last_name);

    http.Br;
  END LOOP;
  footer;
END including_data;
```

### Including Dynamic Data in an HTML Page

Because the procedures are written in PL/SQL, you can access, display and manipulate database data. The simplest way to display records is to select them in a cursor and use a cursor FOR loop to print them. You can use any Toolkit procedure or function in conjunction with the data.



# Summary

- **PL/SQL package and procedure syntax**
- **Parameters passed to procedures**
- **HTP and HTF packages and their benefits**
- **URL syntax**
- **PL/SQL application elements:**
  - **Document header, body, and footer**
  - **Fill-in form**
  - **Anchors as targets and links**

# Practice 3 Overview

**This practice covers the following topics:**

- **Creating a PL/SQL packaged procedure to generate the header and footer for HTML documents**
- **Creating your application's main menu using HTP procedures**
- **Creating procedures to indicate the success or failure of an action**

## Practice 3 Overview

For solutions to this practice, see Appendix A, “Practice Solutions.”

### Instructor Note

From this lesson, the practices require the participants to write a lot of PL/SQL code. The practices also build on the code written in earlier lessons. This means that an uncompleted practice may have a negative effect on future practices.

However, the students have access to .sql files containing the solutions to each practice in their solnPLSQL directory. This enables them to recover from an unsuccessful practice and continue the next practice with a clean copy of the code.

To recover from an incomplete practice, follow these steps:

1. Ask the student to copy the solution file for the unsuccessful practice from the solnPLSQL directory to their home directory.
2. Rename the body.sql file to body.old.
3. Rename the solution file to body.sql.

For example, to start lesson 5 with a clean copy, they should copy sol4\_3.sql to body.sql.

### Practice 3

Any components you create from this point forward should form part of a PL/SQL package called LAB. For PL/SQL login information, refer to Practice 1, Question 1. Copy the template files, `spec.sql` and `body.sql`, from the `labsPLSQL` directory to your home directory and use them as a guide to the content of the LAB package.

1. Log into SQL\*Plus and compile `spec.sql`. This creates the package specification for LAB.
2. In `body.sql`, add code to the `HEADER` private procedure. This procedure accepts a single IN parameter (`v_title`) and creates the HTML tags that begin every page in your application. It should:
  - a. Open the HTML document.
  - b. Open the Head section.
  - c. Use `v_title` to provide a Title tag.
  - d. Close the Head section.
  - e. Open the Body section (with a suitable background such as `/image/owasbkg.jpg`).
  - f. Include the Internet Application Server main logo (`/image/ias_logo.gif`).
  - g. Draw a horizontal rule.
3. Add code to the `FOOTER` private procedure. This procedure accepts no parameters, and creates HTML tags to end every page. It should:
  - a. Draw a horizontal rule.
  - b. Create a clickable image (use `/image/co_home.gif`), which has a link to the home page.
  - c. Add a link to allow users to mail comments to the page designer.
  - d. Close the Body section.
  - e. Close the HTML document.
4. Add code to the `MAIN_PAGE` public procedure. This procedure accepts no parameters and is the application home page. You need to use the `HEADER` and `FOOTER` private procedures to start and end this procedure.
5. Log into SQL\*Plus and compile `body.sql`. This creates the package body for LAB.
6. Test your application's main page by entering the appropriate URL. (See Practice 1, Question 6 for the URL.) Remember that only public procedures and functions can be invoked directly, so to test `HEADER` and `FOOTER`, just make sure that the relevant HTML is displayed on `MAIN_PAGE`.
7. Test the link from your home page by issuing the following URL and clicking on the link to "My Summit Application":  
`http://hostname.domain:your_port_number/`
8. Add code to the `ERROR_PAGE` public procedure. This procedure is used by the application if an error condition arises. It should accept a parameter named `v_message` to allow appropriate messages to be displayed. Use `/image/error.gif` to highlight that this is an error page.

### Practice 3 (continued)

9. Add code to the SUCCESS\_PAGE public procedure. Its purpose is to signify successful completion of an operation. It should accept a parameter named v\_message to allow appropriate messages to be displayed. Use /image/success.gif to highlight that this is a success page.
10. Test SUCCESS\_PAGE and ERROR\_PAGE by entering the appropriate URLs. Remember that the SUCCESS\_PAGE and ERROR\_PAGE procedures require a parameter, and you must tag this onto the end of the URL when testing (with any spaces replaced with +). Here is a sample call to test LAB.SUCCESS\_PAGE:  
`http://hostname.domain:port/pls/dad/lab.success_page?v_message=test+message`

# 4

## Building a Dynamic HTML Table

Copyright © Oracle Corporation, 2001. All rights reserved.

ORACLE®

# Objectives

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

- **Create HTML tables in a Web page using HTP and HTF commands**
- **Display database information within an HTML table**

## Lesson Aim

Web pages often need to show data in a tabular format. In this lesson, you learn how to generate HTML tables through the PL/SQL Web toolkit and include dynamic data from SQL queries.

For more information on writing HTML, see the *NCSA Beginner's Guide to HTML* at:

<http://www.ncsa.uiuc.edu/General/Internet/WWW/HTMLPrimer.html>

# Overview

- **HTML tags for tables**
- **Building a table using HTP and HTF**
- **Using SQL cursors to populate tables dynamically**
- **Producing a table using `owa_util.cellsprint` and `owa_util.tableprint`**

## Basic Table Elements

- Table
- Caption
- Header
- Rows
- Data
- Border

<i>Employee Table</i>		
Employee Name	Employee ID	Department
Ngao	2	Operations
Urguhart	6	Operations
Maduro	16	Operations
Smith	17	Operations

### Basic Table Elements

Tables are a standard feature of HTML 3.0. However, some browsers may not support all table elements.

Cells in tables can contain lists, paragraphs, graphics, and nested tables. You can format your tables in a number of ways; for example, you can set borders, fonts, and alignment.

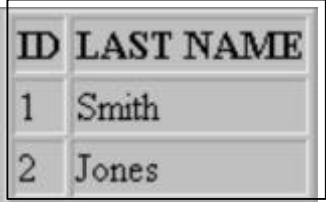


# Table Tags

- **Table delimiters** `<TABLE> </TABLE>`
- **Row delimiters** `<TR> </TR>`
- **Column header delimiters** `<TH> </TH>`
- **Data cell delimiters** `<TD> </TD>`
- **Attributes can determine:**
  - **Borders**
  - **Width and height**
  - **Alignment**

## Displaying Information in a Table

<code>&lt;TABLE BORDER=1&gt;</code>	→	Open table and display border
<code>&lt;TR&gt;</code>		
<code>&lt;TH&gt;ID&lt;/TH&gt;</code>	→	Create a header row
<code>&lt;TH&gt;LAST NAME&lt;/TH&gt;</code>		
<code>&lt;/TR&gt;</code>		
<code>&lt;TR&gt;</code>	→	Enter data row 1
<code>&lt;TD&gt;1&lt;/TD&gt;</code>		
<code>&lt;TD&gt;Smith&lt;/TD&gt;</code>		
<code>&lt;/TR&gt;</code>		
<code>&lt;TR&gt;</code>	→	Enter data row 2
<code>&lt;TD&gt;2&lt;/TD&gt;</code>		
<code>&lt;TD&gt;Jones&lt;/TD&gt;</code>		
<code>&lt;/TR&gt;</code>		
<code>&lt;/TABLE&gt;</code>	→	Close table



### Displaying Information in a Table

Tables are useful for displaying data, especially data returned from a database.

You must start and end a table with the `<TABLE>` and `</TABLE>` tags. Additional table attributes include `ALIGN`, `VALIGN`, `COLSPAN`, `ROWSPAN`, and `NOWRAP`.

You can present information without the border around a table to make images appear as one element or to align graphics and text.

#### Required Table Elements

- `<TABLE>` starts the table definition.
- `<TR>` indicates a table row; you can have any number of rows in a table.
- `<TD>` specifies table data, which is left aligned and vertically centered by default; you can define any number of cells in a row.

#### Optional Table Elements

- `<TH>` creates a table header cell, which is displayed in bold and centered by default.
- `<CAPTION>` defines a caption for the title of the table, which appears centered at the top of the table by default.

Not all browsers can interpret all table elements.

## Table Attributes

Attribute	Value	Description
ALIGN	LEFT, CENTER, RIGHT	Sets the horizontal alignment of a cell
VALIGN	TOP, MIDDLE, BOTTOM	Sets the vertical alignment of a cell
COLSPAN	<i>n</i> (number)	Indicates the number of columns a cell spans
ROWSPAN	<i>n</i> (number)	Indicates the number of rows a cell spans
NOWRAP		Prevents line breaking within a cell
BORDER	BORDER= <i>n</i>	Draws borders; <i>n</i> represents line weight

4-7

Copyright © Oracle Corporation, 2001. All rights reserved.

ORACLE®

### Table Attributes

- <TR>, <TH>, and <TD> tags have a DP attribute, which allows you to set the decimal point character as a period (.) or comma (,).
- COLSPAN and ROWSPAN can be used in <TD> and <TH> tags to make the cell span a number of columns or rows.

# Building a Dynamic Table

```
PROCEDURE show_emps(the_deptno IN VARCHAR2) IS
  CURSOR hcurs IS
    SELECT s_emp.last_name, s_emp.id, s_dept.name
    FROM   s_emp, s_dept
    WHERE  s_emp.dept_id = s_dept.id
    AND    s_emp.dept_id = TO_NUMBER(the_deptno);
BEGIN
  http.tableOpen('border="1"');
  http.tableCaption('Employee Table', 'CENTER');
  http.tableRowOpen;
    http.tableHeader('Employee Name');
    ...
  http.tableRowClose;
  FOR hrec IN hcurs loop
    http.tableRowOpen;
    http.tableData(hrec.last_name);
    ...
    http.tableRowClose;
  END LOOP;
  http.tableClose;
END show_emps;
```

## Selecting Table Data

```
PROCEDURE show_emps(the_deptno IN VARCHAR2) IS
  CURSOR hcurs IS
    SELECT s_emp.last_name, s_emp.id, s_dept.name
    FROM    s_emp, s_dept
    WHERE   s_emp.dept_id = s_dept.id
    AND     s_emp.dept_id = TO_NUMBER(the_deptno);
BEGIN
  ...
END show_emps;
```

# Opening and Closing a Table

```
PROCEDURE show_emps(the_deptno IN VARCHAR2) IS
  CURSOR hcurs IS
    ...
BEGIN
  http.tableOpen( 'BORDER="1"' );
  ...
  http.tableClose;
END show_emps;
```

Procedure Call

```
<TABLE BORDER=1>
```

```
...
```

```
</TABLE>
```

Generated HTML

## Opening and Closing a Table

### Syntax

```
http.tableOpen (cborder, calign, cnowrap, cclear, cattributes);
http.tableClose;
```

### Parameters

```
cborder      IN VARCHAR2 DEFAULT NULL
calign       IN VARCHAR2 DEFAULT NULL
cnowrap      IN VARCHAR2 DEFAULT NULL
cclear       IN VARCHAR2 DEFAULT NULL
cattributes  IN VARCHAR2 DEFAULT NULL
```

For the cclear and cnowrap parameters, enter any value, or leave them set to NULL.

### Generated Code

```
<TABLE cborder NOWRAP ALIGN="calign" CLEAR="clear"
  cattributes>
</TABLE>
```

# Adding a Table Caption

```
PROCEDURE show_emps(the_deptno IN VARCHAR2) IS
  CURSOR hcurs IS
    ...
BEGIN
  http.tableOpen('border="1"');
  http.tableCaption('Employee Table', 'CENTER');
  ...
  http.tableClose;
END show_emps;
```

Procedure Call

```
<TABLE border=1>
<CAPTION ALIGN="CENTER">Employee Table</CAPTION>
...
</TABLE>
```

Generated HTML

## Adding a Table Caption

### Syntax

```
http.tableCaption(ccaption, calign, cattributes);
```

### Parameters

```
ccaption      IN VARCHAR2
calign        IN VARCHAR2 DEFAULT NULL
cattributes   IN VARCHAR2 DEFAULT NULL
```

### Generated Code

```
<CAPTION ALIGN="calign" cattributes>ccaption</CAPTION>
```

### Implementing Table Captions

There are two ways to implement optional table captions:

- Execute the `http.tableCaption` command.
- Execute the `http.tableData` command with the appropriate row and column spans.

# Creating Rows and Headers

```
PROCEDURE show_emps(the_deptno IN VARCHAR2) IS
  CURSOR hcurs IS ...
BEGIN
  http.tableOpen('border="1"'); http.tableCaption('Employee Table', 'CENTER');
  http.tableRowOpen;
  http.tableHeader('Employee
  Name');
  http.tableHeader('Employee ID');
  http.tableHeader('Department');
  http.tableRowClose;
END show_emps;
```

Procedure Call

```
<TABLE border=1>
  <CAPTION ALIGN="CENTER">Employee Table</CAPTION>
  <TR>
    <TH>Employee Name</TH>
    <TH>Employee ID</TH>
    <TH>Department</TH>
  </TR>
... </TABLE>
```

Generated HTML

## Creating Rows and Headers

### Syntax

```
http.rowOpen (calign, valign, cdp, cnowrap, cattributes);
http.tableHeader (cvalue, calign, cdp, cnowrap, crowspan,
                  ccolspan, cattributes);
http.rowClose;
```

### Parameters

calign	IN VARCHAR2 DEFAULT NULL
valign	IN VARCHAR2 DEFAULT NULL
cdp	IN VARCHAR2 DEFAULT NULL
cnowrap	IN VARCHAR2 DEFAULT NULL
crowspan	IN VARCHAR2 DEFAULT NULL
ccolspan	IN VARCHAR2 DEFAULT NULL
cattributes	IN VARCHAR2 DEFAULT NULL

### Generated Code

```
<TR ALIGN="calign" VALIGN="valign" DP="cdp" NOWRAP cattributes>
<TH ALIGN="calign" DP="cdp" ROWSPAN="crowspan" COLSPAN="ccolspan"
  NOWRAP cattributes>cvalue</TH>
</TR>
```



# Loading Dynamic Data into a Table

```
PROCEDURE show_emps(the_deptno IN VARCHAR2) IS CURSOR hcurs IS ...
BEGIN
    http.tableOpen('border="1"'); http.tableCaption('Employee Table', 'CENTER');
    http.tableRowOpen; http.tableHeader('Employee Name'); ... http.tableRowClose;

    FOR hrec IN hcurs LOOP
        http.tableRowOpen;
        http.tableData(hrec.last_name);
        http.tableData(hrec.id);
        http.tableData(hrec.name);
        http.tableRowClose;
    END LOOP;

    http.tableClose; END show_emps;
```

Procedure Call

```
<TABLE border=1>
  <CAPTION ALIGN="CENTER">Employee Table</CAPTION>
  <TR> <TH>Employee Name</TH> ... </TR>

  <TR>
    <TD>Ngao</TD>
    <TD>2</TD>
    <TD>Operations</TD>
  </TR> ...

</TABLE>
```

Generated HTML

## Loading Dynamic Data Into a Table

### Syntax

```
http.tableData (cvalue, calign, cdp, crowspan, ccolspan, cnowrap,
               cattributes);
```

### Parameters

cvalue	IN VARCHAR2 DEFAULT NULL
calign	IN VARCHAR2 DEFAULT NULL
cdp	IN VARCHAR2 DEFAULT NULL
crowspan	IN VARCHAR2 DEFAULT NULL
ccolspan	IN VARCHAR2 DEFAULT NULL
cnowrap	IN VARCHAR2 DEFAULT NULL
cattributes	IN VARCHAR2 DEFAULT NULL

### Generated Code

```
<TD ALIGN="calign" DP="cdp" ROWSPAN="crowspan" COLSPAN="ccolspan"
NOWRAP cattributes>cvalue</TD>
```

## Printing a Database Table Using owa\_util.tableprint

```
PROCEDURE show_depts IS
```

```
  x  BOOLEAN;
```

```
BEGIN
```

```
  x := OWA_UTIL.tablePrint ('s_dept');
```

```
END show_depts;
```

Function Call

```
<TABLE  BORDER>
```

```
<TR>
```

```
<TH>ID</TH><TH>NAME</TH><TH>REGION</TH>
```

```
</TR>
```

```
<TR>
```

```
<TD>10</TD><TD>Finance</TD><TD>1</TD>
```

```
</TR>
```

Generated HTML

### Example: Printing a Database Table Using OWA\_UTIL.tablePrint

owa\_util.tableprint takes the name of a table as a parameter.

You can quickly and easily print tables with the owa\_util.tableprint function. Depending on your Web browser capabilities, you can specify that the table be printed preformatted or HTML.

You can specify multiple tables to perform a joined query and you can specify a WHERE clause to restrict data and an ORDER BY clause to sort data.

## Printing Part of a Database Table

```
PROCEDURE show_emps(the_deptno IN VARCHAR2) IS
  x BOOLEAN;
BEGIN
  x:= owa_util.tablePrint(
    's_emp, s_dept','BORDER', owa_util.html_Table,
    's_emp.last_name, s_emp.id, s_dept.name',
    'where s_emp.dept_id=s_dept.id',
    'Surname, Employee Number, Department');
END show_emps;
```

Function Call

```
<TABLE BORDER>
<TR>
  <TH>Employee</TH>
  <TH>Employee Number</TH>
  <TH>Department</TH>
</TR>
<TR>
  <TD ALIGN="LEFT">Ngao</TD>
  <TD ALIGN="LEFT">2</TD>
  <TD ALIGN="LEFT">Operations</TD>
</TR>
```

Generated HTML

## Printing Part of a Database Table

### Syntax

```
owa_util.tablePrint (ctable, cattributes, ntable_type, ccolumns,
                    cclauses, ccol_aliases, nrow_min, nrow_max);
```

### Parameters

ctable	IN VARCHAR2
cattributes	IN VARCHAR2 DEFAULT NULL
ntable_type	IN INTEGER DEFAULT HTML_TABLE
ccolumns	IN VARCHAR2 DEFAULT '*'
cclauses	IN VARCHAR2 DEFAULT NULL
ccol_aliases	IN VARCHAR2 DEFAULT NULL
nrow_min	IN NUMBER DEFAULT 0
nrow_max	IN NUMBER DEFAULT NULL

For the ntable\_type parameter, enter owa\_util.PRE\_TABLE for a preformatted table.

### Return Value

TRUE is returned if more rows are available beyond the nrow\_max value requested.

## Printing SQL Query Results Using owa\_util.cellsprint

- Produces HTML table cells from a query
- Allows a degree of control over returned values:
  - Limit the number of rows
  - Skip the first x rows
  - Page through data sets
  - Format number columns

```
OWA_UTIL.cellsPrint  
( 'select * from emp', 25, 'YES' );
```

- Does not produce <TABLE> and </TABLE> tags

### Printing SQL Query Results Using owa\_util.cellsprint

There are eight versions of this procedure. The first and second versions display rows (up to the specified maximum) returned by the query cursor. The third and fourth versions allow you to exclude the specified number of rows from the HTML table. You can also use the third and fourth versions to scroll through result sets by saving the last row seen in hidden form element.

The fifth through eighth versions are the same as the first four versions, except that they return a row count output parameter.

#### Syntax

```
owa_util.cellsprint(p_theQuery in varchar2  
                   p_max_rows in number DEFAULT 100  
                   p_format_numbers in varchar2 DEFAULT NULL);  
  
owa_util.cellsprint(p_theCursor in integer  
                   p_max_rows in number DEFAULT 100  
                   p_format_numbers in varchar2 DEFAULT NULL);  
  
owa_util.cellsprint(p_theQuery in varchar2  
                   p_max_rows in number DEFAULT 100  
                   p_format_numbers in varchar2 DEFAULT NULL  
                   p_skip_rec in number default 0  
                   p_more_data out boolean);
```

## Table Tips

- **Tables can be nested**
- **Tables need not have borders**
- **Tables can enhance layout**
- **Tables can contain form input items**
- **Large tables can take a long time to display**
- **`owa_util.tableprint` and `owa_util.cellsprint` are less efficient than the cursor method using HTP procedures**

### Table Tips

`owa_util.tableprint` and `owa_util.cellsprint` use dynamic SQL, which can be less efficient than using the cursor method with HTP procedures.

## Summary

- **HTML table tags**
- **HTP commands**
- `owa_util.tableprint`
- `owa_util.cellsprint`

## Practice 4 Overview

**This practice covers the following topics:**

- **Creating a procedure that generates an HTML table with information about every customer in the S\_CUSTOMER table**
- **Creating a procedure that generates an HTML table with information about a specific customer in the S\_CUSTOMER table**

### **Practice 4 Overview**

For solutions to this practice, see Appendix A, “Practice Solutions.”

## Practice 4

1. Add code to the `SHOW_ALL_CUST` public procedure to list all customers in the `S_CUSTOMER` table sorted by customer name. This procedure accepts no parameters. You may choose to use a cursor and the HTML-table related tags in the HTP package or to use `owa_util.cellsprint`. Display the customer's ID, name, country, credit rating, and sales representative ID.
2. Test your procedure. Make sure that the corresponding link in `LAB.MAIN_PAGE`, "Full customer listing", works correctly.
3. Add code to the `SHOW_CUSTOMER` public procedure to list information about a particular customer in the `S_CUSTOMER` table. This procedure accepts one parameter, `v_id`, the customer ID whose details you need to display. Copy the code from `SHOW_ALL_CUST` and make necessary modifications to it so the table looks like the following:

Id	Customer	Address	Telephone	Rating
209	Beisbol Sil	792 Playa Del Mar San Pedro de Macon's Costa Rica	809-352689	EXCELLENT

Test your procedure using:

`http://hostname.domain:port/pls/dad/lab.show_customer?v_id=209`

**Note:** If necessary, replace the value 209 with another valid customer ID.



# 5

## Building Dynamic Forms

Copyright © Oracle Corporation, 2001. All rights reserved.

**ORACLE®**

# Objectives

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

- **Identify the basic elements of an HTML form**
- **Create dynamic HTML forms using the PL/SQL Web toolkit**
- **Use an image to submit a form**

## **Lesson Aim**

Users can interact with the database by passing data through HTML forms. This lesson introduces the HTML form elements and explains how to create forms dynamically using the PL/SQL Web toolkit.

# Overview

- **Defining the elements of a form**
- **Defining a form in HTML**
- **Creating form elements with HTP and HTF**
- **Defining the relationship between input elements and ACTION procedure parameters**
- **Handling multiple values**
- **Creating a clickable image**

## Defining a Form in HTML

- **<FORM> ... </FORM>**
  - The action specifies the URL invoked when the submit button is clicked
  - The method is either GET or POST
- **<INPUT> defines the field type and name**

```
<FORM ACTION="form_http.show"
      METHOD="POST">
  <INPUT TYPE="text" NAME="p_data">
  <INPUT TYPE="submit">
</FORM>
```

5-4

Copyright © Oracle Corporation, 20001 All rights reserved.

ORACLE®

### Defining a Form in HTML

Name the fields to match the parameters in your called application.

The METHOD attribute specifies the way the encoded results are passed to the procedure or program:

- GET appends information from the form to the end of the action URL. Your program receives the encoded form input in the QUERY\_STRING environment variable. You should not use this method because it uses operating system environment variables that place limits on the length of QUERY\_STRING.
- POST transmits all form input information immediately after the requested URL. Your program receives the encoded form information as standard input. This is the preferred method.

For more information on creating HTML forms, see *Instantaneous Introduction to CGI Scripts and HTML Forms* at

<http://www.cc.ukans.edu/~acs/docs/other/forms-intro.shtml>

## Text and Password Fields

- **Text**

Username

```
Username: <INPUT TYPE="text"
NAME="p_user" SIZE="8">
```

- **Password**

Password

```
Password: <INPUT TYPE="password"
NAME="p_pass" SIZE="8">
```

### Text Fields

Text is the default input type. A text field displays a simple line of text:

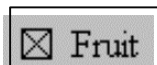
- NAME defines the identifier for the data (required).
- SIZE specifies the size of the field in number of characters.
- MAXLENGTH specifies the maximum number of characters allowed.
- VALUE defines the default text.

### Password Fields

A password field is a modified text field. It displays typed characters as stars. Its options are identical to those available for text fields.

## Check Box and Radio Button Fields

- **Check box**



```
<INPUT TYPE="checkbox" NAME="p_ck"
VALUE="fruit" CHECKED> Fruit
```

- **Radio button**



```
<INPUT TYPE="radio" NAME="p_radio"
VALUE="red" CHECKED> Red
<INPUT TYPE="radio" NAME="p_radio"
VALUE="yellow"> Yellow
```

### Check Boxes

The simple check box can be checked or empty; use it when the choice is yes or no.

- NAME defines the identifier for the data (required).
- VALUE specifies the value returned to the server if the box is checked.
- CHECKED sets checked as the default status for the check box.

If the form is submitted with the box unchecked, then no name-value pair is sent.

### Radio Buttons

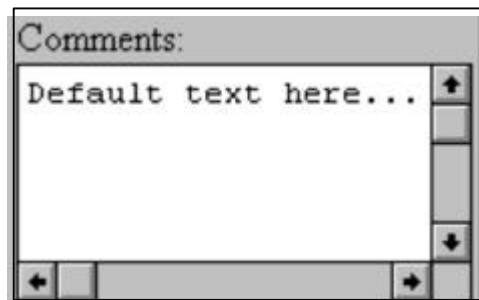
Radio buttons are more complex versions of the check box; only one button in the set can be selected.

- NAME defines the identifier for the data (required).
- VALUE specifies the value returned to the server if the button is selected.
- CHECKED sets selected as the default status for the radio button.

All radio buttons that you want to group together as mutually exclusive must share the same NAME value.

## Text Area Fields

```
Comments:<BR>  
<TEXTAREA NAME = "p_text" ROWS=4  
  COLS=40>  
  Default text here...  
</TEXTAREA>
```



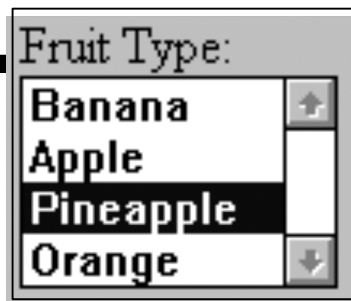
### Text Area Fields

A text area is a text field that allows multiple-line input.

- NAME defines the identifier for the data (required).
- ROWS sets the number of rows in the field.
- COLS sets the width of the field (in characters).
- DEFAULT TEXT specifies the default text that appears in the field. Even if you do not enter default text, you must still close the text area tag.
- WRAP sets the wrap style to NONE, PHYSICAL, or VIRTUAL

## Select List Fields

```
Fruit Type:<BR>
<SELECT NAME="p_type" SIZE=4 MULTIPLE>
<OPTION VALUE="1"> Banana
<OPTION VALUE="2"> Apple
<OPTION SELECTED VALUE="3"> Pineapple
<OPTION VALUE="4"> Orange
</SELECT>
```



5-8

Copyright © Oracle Corporation, 20001 All rights reserved.

ORACLE®

### Select List Fields

A select list field is a pop-up or scrollable list that displays multiple choices.

- NAME defines the identifier for the data (required).
- SIZE determines the number of choices shown.
- MULTIPLE allows multiple selections.

List options include a value and the text string displayed in the list. If you want one of the choices to be the default, use the SELECTED attribute. If you omit the VALUE attribute from the <OPTION> tag, the corresponding text is returned as the selected value.



# Hidden Fields

```
<INPUT TYPE="hidden" NAME="p_rowid"  
VALUE="00003105.0000.0001">
```

- The field is not visible on the screen
- The hidden value is sent to the server
- Hidden does not mean secure

## Hidden Fields

Use hidden fields to:

- Pass a user ID and password from page to page to revalidate the user's access to those pages  
**Note:** Use carefully because the information is clearly visible if the user views the document source.
- Store the URL to determine from which form the user entered information
- Collect information, for example the ROWID value, to be used in a later database transaction

Although most browsers support hidden fields, some browsers do not hide them, which can lead to user confusion.

## Submit and Reset Buttons

- Reset button



```
<INPUT TYPE="reset" VALUE="Reset">
```

- Submit button



```
<INPUT TYPE="submit" NAME="p_submit"  
VALUE="Execute">
```

### Reset Button

The reset button clears all the data in the form to its original value. VALUE specifies the button label.

### Submit Button

The submit button sends the data in the form to the server to be processed. The data is sent in name-value pairs.

- The NAME attribute specifies the name of the data item. It must correspond to a formal parameter name in the procedure referenced by the FormOpen procedure.
- The VALUE attribute specifies the value of the parameter.

If you have a button with a NAME attribute, then a name-value pair is generated for it when the user clicks the button.

The Execute button on the slide submits the following parameter:

```
p_submit=Execute
```

This enables you to create several submit buttons on a single form, each of which carries out the specified action when selected.

**Note:** If the user activates the button by pressing Return, the name-value pair is not generated.

# Opening a Form

```
http.formOpen( 'form_http.show' );
```

Procedure Call

```
<FORM ACTION="form_http.show" METHOD="POST">
```

Generated HTML

5-11

Copyright © Oracle Corporation, 20001 All rights reserved.

**ORACLE®**

## Opening a Form with the HTP Package

### Syntax

```
http.formOpen (curl, cmethod, ctargget, cencntype, cattributes);
```

### Parameters

```
curl           IN VARCHAR2
cmethod        IN VARCHAR2 DEFAULT 'POST'
ctargget       IN VARCHAR2 DEFAULT NULL
cencntype      IN VARCHAR2 DEFAULT NULL
cattributes    IN VARCHAR2 DEFAULT NULL
```

### Generated Code

```
<FORM ACTION="curl" METHOD="cmethod" TARGET="ctargget "
ENCTYPE="cencntype" cattributes>
```

## Creating a Text Field

```
http.formOpen('form_http.show');  
http.formText('p_data');
```

Procedure Call

```
<FORM ACTION="form_http.show" METHOD="POST">  
<INPUT TYPE="text" NAME="p_data"...>
```

Generated HTML

### Creating a Text Field with HTP

#### Syntax

```
http.formText (cname, csize, cmaxlength, cvalue, cattributes);
```

#### Parameters

cname	IN VARCHAR2
csize	IN VARCHAR2 DEFAULT NULL
cmmaxlength	IN VARCHAR2 DEFAULT NULL
cvalue	IN VARCHAR2 DEFAULT NULL
cattributes	IN VARCHAR2 DEFAULT NULL

#### Generated Code

```
<INPUT TYPE="text" NAME="cname" SIZE="csize" MAXLENGTH="cmmaxlength"  
VALUE="cvalue" cattributes>
```

To create a password field, use `http.formPassword`. The syntax is similar to that for `http.formText`.

## Creating a Radio Group

```
http.formOpen('form_http.show');  
http.formText ('p_data');
```

```
http.formRadio ('p_color','Red','YES');  
http.p ('Red');  
http.formRadio ('p_color', 'Yellow');  
http.p ('Yellow');
```

Procedure Call

```
<FORM ACTION="form_http.show" METHOD="POST">  
<INPUT TYPE="text" NAME="p_data"...>
```

```
<INPUT TYPE="radio" NAME="p_color"  
      VALUE="Red" CHECKED> Red  
<INPUT TYPE="radio" NAME="p_color"  
      VALUE="Yellow"> Yellow
```

Generated HTML

### Creating a Radio Group with HTP

#### Syntax

```
http.formRadio (cname, cvalue, cchecked, cattributes);
```

#### Parameters

cname	IN VARCHAR2
cvalue	IN VARCHAR2
cchecked	IN VARCHAR2 DEFAULT NULL
cattributes	IN VARCHAR2 DEFAULT NULL

#### Generated Code

```
<INPUT TYPE="radio" NAME="cname" VALUE="cvalue" CHECKED cattributes>
```

Any value in the cchecked parameter means CHECKED; leave this parameter set to NULL to indicate the unchecked state.

Ensure that the NAME attribute is the same for all buttons in a radio group.

To create a check box, use `http.formCheckbox`. The syntax is similar to that for `http.formRadio`.

## Creating a Select List

```
http.formOpen('form_http.show'); http.formText('p_data');
http.formRadio('p_color','Red','YES'); http.p('Red');
http.formRadio('p_color','Yellow'); http.p('Yellow');

http.formSelectOpen('p_fruit','Pick a Fruit:', 4);
http.formSelectOption('Banana','SELECTED',
                      cattributes=>'VALUE="1"');
http.formSelectOption('Apple',
                      cattributes=>'VALUE="2"'); ...
http.formSelectClose;
```

Procedure Call

```
<FORM ACTION="form_http.show" METHOD="POST">
<INPUT TYPE="text" NAME="p_data"...>
<INPUT TYPE="radio" NAME="p_color"...>
Pick a fruit:
<SELECT NAME="p_fruit" SIZE=4>
<OPTION SELECTED VALUE="1"> Banana
<OPTION VALUE="2"> Apple ...
</SELECT>
```

Generated HTML

### Creating a Select List with HTP

#### Syntax

```
http.formSelectOpen (cname, cprompt, nsize, cattributes);
http.formSelectOption (cvalue, cselected, cattributes);
http.formSelectClose;
```

#### Parameters

cname	IN VARCHAR2
cprompt	IN VARCHAR2 DEFAULT NULL
nsize	IN INTEGER DEFAULT NULL
cvalue	IN VARCHAR2
cselected	IN VARCHAR2 DEFAULT NULL
cattributes	IN VARCHAR2 DEFAULT NULL

Any value in the cselected parameter means SELECTED; leave this parameter set to NULL to indicate the unselected state.

#### Generated Code

```
<SELECT NAME="cname" SIZE="nsize" cattributes>
<OPTION SELECTED cattributes>cvalue
</SELECT>
```

## Creating a Hidden Field

```
http.formOpen('form_http.show'); http.formText('p_data');  
http.formRadio('p_color','Red','YES'); http.p('Red');  
http.formRadio('p_color','Yellow'); http.p('Yellow');  
http.formSelectOpen('p_fruit','Pick a Fruit:', 4);  
...  
http.formSelectClose;
```

```
http.formHidden ('p_val','tiger');
```

Procedure Call

```
<FORM ACTION="form_http.show" METHOD="POST">  
<INPUT TYPE="text" NAME="p_data"...>  
<INPUT TYPE="radio" NAME="p_color"...>  
Pick a fruit:  
<SELECT NAME="p_fruit" SIZE=4>  
...  
</SELECT>
```

```
<INPUT TYPE="hidden" NAME="p_val" VALUE="tiger">
```

Generated HTML

### Creating a Hidden Field with HTP

#### Syntax

```
http.formHidden (cname, cvalue, cattributes);
```

#### Parameters

```
cname          IN VARCHAR2  
cvalue         IN VARCHAR2 DEFAULT NULL  
cattributes    IN VARCHAR2 DEFAULT NULL
```

#### Generated Code

```
<INPUT TYPE="hidden" NAME="cname" VALUE="cvalue" cattributes>
```

## Submitting a Form

```
http.formOpen('form_http.show'); http.formText('p_data');
http.formRadio('p_color','Red','YES'); http.p('Red');
http.formRadio('p_color','Yellow'); http.p('Yellow');
http.formSelectOpen('p_fruit','Pick a Fruit:', 4);
...
http.formSelectClose; http.formHidden ('p_val','tiger');
http.formSubmit(cvalue=>'Execute');
```

Procedure Call

```
<FORM ACTION="form_http.show" METHOD="POST">
<INPUT TYPE="text" NAME="p_data"...>
<INPUT TYPE="radio" NAME="p_color"...>
Pick a fruit:
<SELECT NAME="p_fruit" SIZE=4> ... </SELECT>
<INPUT TYPE="hidden" NAME="p_val" VALUE="tiger">
<INPUT TYPE="submit" VALUE="Execute">
```

Generated HTML

### Submitting a Form with HTP

The method indicated in the form element can be either POST or GET.

#### Syntax

```
http.formSubmit (cname, cvalue, cattributes);
```

#### Parameters

```
cname          IN VARCHAR2 DEFAULT NULL
cvalue         IN VARCHAR2 DEFAULT 'Submit'
cattributes    IN VARCHAR2 DEFAULT NULL
```

#### Generated Code

```
<INPUT TYPE="submit" NAME="cname" VALUE="cvalue" cattributes>
```



## Resetting a Form

```
http.formOpen('form_http.show'); http.formText('p_data');
http.formRadio('p_color','Red','YES'); http.p('Red');
http.formRadio('p_color','Yellow'); http.p('Yellow');
http.formSelectOpen('p_fruit','Pick a Fruit:', 4);
... http.formSelectClose;
http.formHidden('p_val','tiger');
http.formSubmit(cvalue=>'Execute');
```

```
http.formReset;
```

Procedure Call

```
<FORM ACTION="form_http.show" METHOD="POST">
<INPUT TYPE="text" NAME="p_data"...>
<INPUT TYPE="radio" NAME="p_color"...>
Pick a fruit:
<SELECT NAME="p_fruit" SIZE=4> ... </SELECT>
<INPUT TYPE="hidden" NAME="p_val" VALUE="tiger">
<INPUT TYPE="submit" VALUE="Execute">
```

```
<INPUT TYPE="reset" VALUE="Reset">
```

Generated HTML

### Resetting a Form with HTP

The `formReset` procedure creates a reset button that resets all the data in the form to its original value when it is clicked.

#### Syntax

```
http.formReset (cvalue, cattributes);
```

#### Parameters

```
cvalue          IN VARCHAR2 DEFAULT 'Reset'
cattributes IN VARCHAR2 DEFAULT NULL
```

#### Generated Code

```
<INPUT TYPE="reset" VALUE="cvalue" cattributes>
```

## Closing a Form

```
http.formOpen('form_http.show'); http.formText('p_data');  
http.formRadio('p_color','Red','YES'); http.p('Red');  
http.formRadio('p_color','Yellow'); http.p('Yellow');  
http.formSelectOpen('p_fruit','Pick a Fruit:', 4);  
... http.formSelectClose;  
http.formHidden('p_val','tiger');  
http.formSubmit(cvalue=>'Execute');  
http.formReset;  
http.formClose;
```

Procedure Call

```
<FORM ACTION="form_http.show" METHOD="POST">  
<INPUT TYPE="text" NAME="p_data"...>  
<INPUT TYPE="radio" NAME="p_color"...>  
Pick a fruit:  
<SELECT NAME="p_fruit" SIZE=4> ... </SELECT>  
<INPUT TYPE="hidden" NAME="p_val" VALUE="tiger">  
<INPUT TYPE="submit" VALUE="Execute">  
<INPUT TYPE="reset" VALUE="Reset">  
</FORM>
```

Generated HTML

### Closing a Form with HTTP

#### Syntax

```
http.formClose;
```

#### Generated Code

```
</FORM>
```

## Passing Multiple Values

```
http.formSelectOpen('p_depts', nsize=>3,  
cattributes=>'MULTIPLE');  
...
```

Procedure Call

```
<SELECT NAME="p_depts" SIZE=3 MULTIPLE>  
<OPTION>Sales  
<OPTION>Marketing  
...
```

Generated HTML

- The receiving procedure must provide a PL/SQL table
- Pass at least one value from the form; use a hidden field if required

### Passing Multiple Values

Multiple values can be accepted by select lists and by other input types (such as check boxes) if you create these items with the same NAME attribute.

Multiple values are handled in PL/SQL tables. When creating the procedure that receives the form values, you must define a parameter as a PL/SQL table to collect the corresponding values from the form.

If a value is not provided (that is, if the user fails to select an option), an error occurs. You cannot provide a default for a PL/SQL table, so if you cannot guarantee that at least one value is submitted, you must use a hidden field to provide the first value.

# Handling Multiple Values

```
PROCEDURE multi (p_depts IN owa_util.ident_arr)
IS
  v_ct  NUMBER(3);
BEGIN
  v_ct := p_depts.COUNT;
  http.print('You selected ' || v_ct - 1 || ' choices');
  http.print('Those choices are :');
  http.olistOpen;
  FOR counter IN 2..v_ct LOOP
    http.listItem(p_depts(counter));
  END LOOP;
  http.olistClose;
END;
```

5-20

Copyright © Oracle Corporation, 20001 All rights reserved.

ORACLE®

## Handling Multiple Values

The OWA\_UTIL.IDENT\_ARR type is a VARCHAR2(30) table, indexed by binary\_integer values. You do not need to use this supplied data type, but you must use a VARCHAR2 table.

The user may not select any values from the list, so you must ensure that a hidden field passes a value:

```
http.FormOpen('pkg.multi');
http.FormHidden('p_depts','None');
http.FormSelectOpen('p_depts',null,3,cattributes=>'MULTIPLE');
http.FormSelectOption('Sales');
http.FormSelectOption('Marketing');
...
```

When you look at the passed values, you must ignore this hidden value. In the example on the slide, 1 is subtracted from the count and the loop begins at the second element.

## Flexible Parameter Passing

- You can define a procedure that has the following signature:

```
proc_name(name_array IN OWA.vc_arr,  
          value_array IN OWA.vc_arr)
```

- You use ! in the URL to indicate flexible parameters:

```
http://host/pls/mydad/!myproc?par1=val1&par2=val2
```

- The PL/SQL Gateway invokes the procedure with the parameters:

```
name_array ==> ('par1', 'par2');  
value_array ==> ('val1', 'val2');
```

### Flexible Parameter Passing

You can have HTML forms from which users can select any number of elements. If these elements have different names, you must create overloaded procedures to handle each possible combination. The PL/SQL Gateway makes this process easier by supporting flexible parameters. You can use a two parameter interface or a four parameter interface but the two parameter interface is recommended for performance reasons.

To use the two parameter interface, your procedure must have this signature:

```
proc_name (name_array IN OWA.vc_arr,  
          value_array IN OWA.vc_arr)
```

- proc\_name is the name of the PL/SQL procedure that you are invoking
- name\_array specifies the names from the query string
- value\_array specifies the values from the query string

You invoke the procedure with the following URL format:

```
http://host/pls/mydad/!package.procedure?par1=val1&par2=val2
```

The '!' tells the PL/SQL Gateway to invoke the procedure with the following parameters:

```
name_array ==> ('par1', 'par2');  
value_array ==> ('val1', 'val2');
```

## Creating an Image Map

- **<INPUT TYPE="image"> enables you to specify a clickable image.**
- **The form data pairs include x, y coordinates:**

```
PROCEDURE process_form (p_data    IN VARCHAR2,  
                        p_image    IN owa_image.point) IS  
BEGIN  
    http.p ('p_data=' || p_data);  
    http.p ('x=' || owa_image.get_x(p_image));  
    http.p ('y=' || owa_image.get_y(p_image));  
END process_form;
```

### Submitting a Form Using an Image

The clickable image differs from an image map because data can be sent with the x, y coordinates.

You use the `owa_image.get_x` and `owa_image.get_y` functions to retrieve the x and y positions of a mouse-click on an image. The coordinates are sent as the `owa_image.point` data type. After they are retrieved, you can use these positions to determine the part of the image that was clicked and what procedure to call.

#### Example

- Submit a form using a clickable image field. Use the point data type in a procedure to create a variable to hold the x and y positions from a form ACTION value.
- Retrieve locations in a database within a 5-kilometer radius based on the longitude and latitude coordinates determined from the position of the mouse-click.

## Summary

- **HTML forms:**
  - **Form definition**
  - **Input fields**
  - **Submit button**
  - **Reset button**
- **Creating a form using HTP and HTF**
- **Handling multiple values**
- **Flexible parameter passing**
- **Creating a clickable image**

## Practice 5 Overview

**This practice covers the following topics:**

- **Creating a procedure that generates an HTML form with a list of customers selected from the S\_CUSTOMER table**
- **Creating an HTML form with two submit buttons**
- **Modifying an HTML form to insert, update, and delete data in the S\_CUSTOMER table**

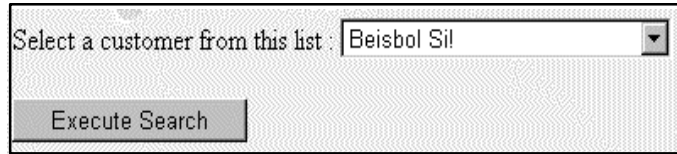
### **Practice 5 Overview**

For solutions to this practice, see Appendix A, “Practice Solutions.”



## Practice 5

1. Add code to the public procedure `FIND_CUSTOMER` to allow the user to select a customer from a drop-down (select) list. This procedure accepts no parameters. Display the customer's name in the list, but transmit the customer ID (`v_id`) to the action procedure of this HTML form. Call the `SHOW_CUSTOMER` procedure that you wrote in Practice 4, Question 3 as the action of this HTML form.



The screenshot shows a web form with a label "Select a customer from this list :". To the right of the label is a dropdown menu with "Beisbol Sil" selected. Below the dropdown is a button labeled "Execute Search".

2. Test your procedures. Make sure that the corresponding link, "Search and modify customer," in `LAB.MAIN_PAGE` works correctly.
3. Modify `SHOW_CUSTOMER` to add an HTML form that displays Update and Delete buttons after displaying customer information. The form has no other visual input items besides the two buttons. Set the action of the form to call `LAB.MODIFY_CUSTOMER` with two parameters: the ID of the customer whose record is displayed on this screen (use a hidden field) and the value of the button selected by the user.
4. Examine the code of `LAB.MODIFY_CUSTOMER`. Note that the main functionality of this procedure is to generate HTML forms for inserting, updating, and deleting customers.
  - a. If the action passed to the form is 'Delete', then this procedure generates an HTML form that prompts the user to confirm the delete.
  - b. If the action passed to the form is 'Update', then this procedure generates an HTML form that displays the customer information so that the user can modify it.
  - c. If the action passed to the form is 'Insert', then this procedure presents a blank HTML form so that the user can add a new customer.
5. Test your procedure. Invoke "Search and modify customer" from `LAB.MAIN_PAGE` and ensure that your buttons appear correctly.

### If you have time:

6. Add the following functionality to the `LAB.MODIFY_CUSTOMER` procedure:
  - a. Convert the `sales_rep_id` and `region_id` text fields into select lists.
  - b. Convert the `credit_rating` text field into a radio group.

**Hint:** Remember that when updating data, this form needs to display the current values initially. Select lists normally display the first item in the list. Therefore you need to ensure that the `SELECTED` attribute is set for the current value. Similarly, for the radio group, ensure that the `CHECKED` value is set for the current value.
7. Test `LAB.MODIFY_CUSTOMER`.
  - a. Invoke the procedure from the "Insert new customer" link in `LAB.MAIN_PAGE`.
  - b. Invoke the procedure from the Update and Delete buttons on the `LAB.SHOW_CUSTOMER` form.



# 6

## **Adding Functionality with Toolkit Utilities**

Copyright © Oracle Corporation, 2001. All rights reserved.

**ORACLE®**

# Objectives

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

- **Describe the utility packages provided with the PL/SQL Web toolkit**
- **Output the source of a specified PL/SQL program unit**
- **Dynamically retrieve CGI environment variables**

## **Lesson Aim**

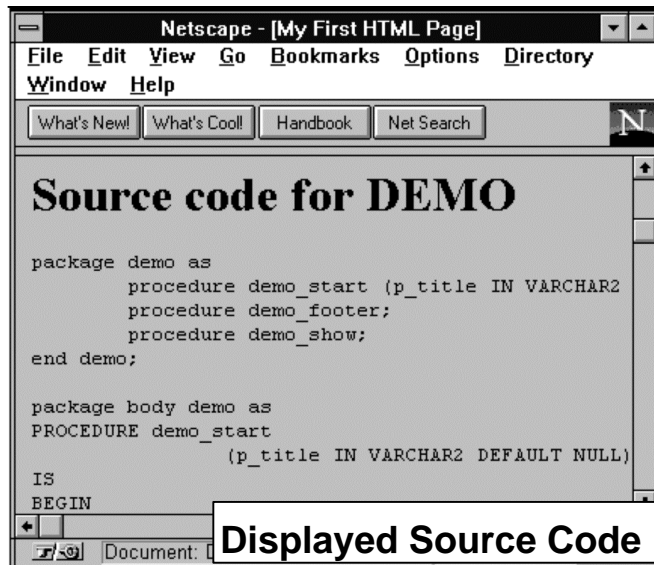
This lesson introduces some additional utilities provided by the PL/SQL Web toolkit. These allow you to use extra features in your Web pages, such as environment information.

# Overview

- Using additional `owa_util` procedures
- Entering dates in forms with `owa_util`
- Pattern matching
- Calling one type of application from another

## Viewing a Stored Procedure

```
owa_util.showsource('demo'); Procedure Call
```



6-4

Copyright © Oracle Corporation, 2001. All rights reserved.

ORACLE®

### Viewing a Stored Procedure

#### Syntax

```
owa_util.showsource(cname);
```

#### Parameters

cname IN VARCHAR2

Specifies the package or procedure name

#### Generated Code

When you invoke `owa_util.showsource`, the source code for the specified PL/SQL procedure is displayed in the Web browser. If you specify a procedure that belongs to a package, the source code for the whole package is returned.

### Viewing Procedure Output

The `owa_util.showpage` procedure prints the HTML output of a procedure in SQL\*Plus, SQL\*DBA, or Oracle Server Manager. The procedure must use the HTP and HTF packages to generate the HTML page and the `owa_util.showpage` procedure must be issued after the procedure has been called and before any other HTP or HTF subprograms are directly or indirectly called. This method is useful for generating pages filled with static data.

#### Syntax

```
owa_util.showpage;
```

## CGI Environment Variables

Variable	Description
SERVER_NAME	The server host name, DNS alias, or IP address
HTTP_ACCEPT	The MIME types that the client accepts
HTTP_USER_AGENT	The client's browser
PATH_INFO	Virtual path (/pls/dad)
REMOTE_ADDR	IP address making the request
REMOTE_HOST	Name of the host making the request
REMOTE_USER	Authenticated user name
SCRIPT_NAME	Name of the package and procedure requested by the user
SERVER_PORT	Port number to which the request was sent

6-5

Copyright © Oracle Corporation, 2001. All rights reserved.

**ORACLE®**

### CGI Environment Variables

You can use environment variables to pass data about the information request from the server to a script. These variables are set when the server uses the PL/SQL Gateway:

AUTHORIZATION	HTTP_HOST	REQUEST_IANA_CHARSET
DAD_NAME	HTTP_PRAGMA	REQUEST_METHOD
DOC_ACCESS_PATH	HTTP_USER_AGENT	REQUEST_PROTOCOL
DOCUMENT_TABLE	PATH_ALIAS	SCRIPT_NAME
HTTP_ACCEPT	PATH_INFO	SCRIPT_PREFIX
HTTP_ACCEPT_CHARSET	REMOTE_ADDR	SERVER_NAME
HTTP_ACCEPT_ENCODING	REMOTE_HOST	SERVER_PORT
HTTP_ACCEPT_LANGUAGE	REMOTE_USER	SERVER_PROTOCOL
HTTP_COOKIE	REQUEST_CHARSET	

## CGI Information

Utility	Description
<code>get_cgi_env</code>	Retrieves the value of a specified CGI environment variable
<code>print_cgi_env</code>	Prints values of all CGI environment variables

### CGI Information

These `owa_util` utilities give you access to CGI information.



# Retrieving an Environment Variable

Keep track of:

- Who visited your Web site
- The requesting page

```
INSERT INTO web_visitors
  (ip_address, page_name)
VALUES
  ( owa_util.get_cgi_env ('REMOTE_ADDR'),
    owa_util.get_cgi_env ('HTTP_REFERER'));
```

6-7

Copyright © Oracle Corporation, 2001. All rights reserved.

ORACLE®

## Retrieving an Environment Variable

### Syntax

```
owa_util.get_cgi_env(param_name);
```

### Parameters

```
param_name  IN VARCHAR2
```

See the list of values for param\_name on previous slides.

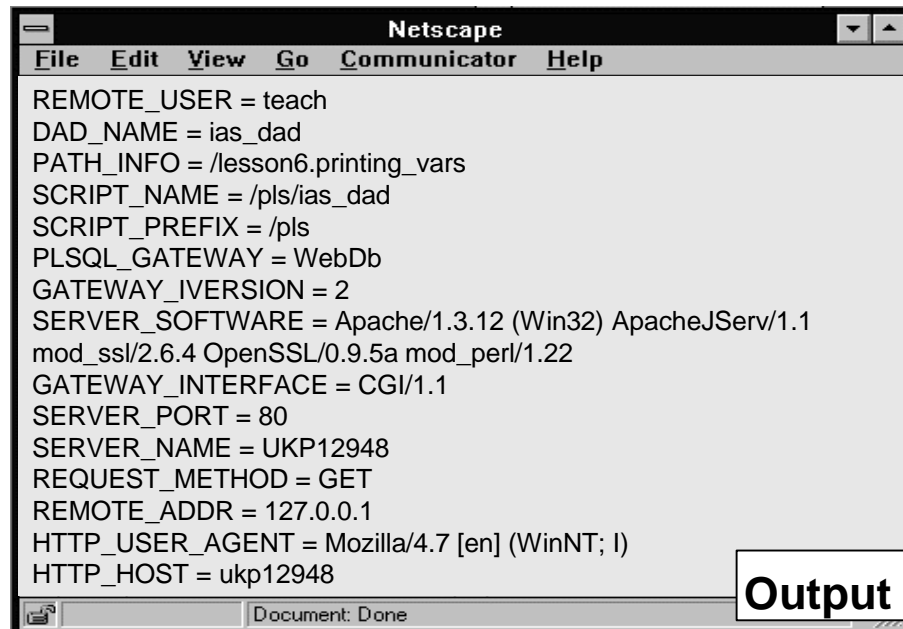
### Return Value

This function returns the value of the CGI environment variable for the PL/SQL procedure. If the value is not set, the returned value is NULL. Although the PL/SQL Gateway is not operated through CGI, it can use CGI environment variables.

# Viewing CGI Environment Variables

```
owa_util.print_cgi_env;
```

Procedure Call



Output

6-8

Copyright © Oracle Corporation, 2001. All rights reserved.

ORACLE®

## Viewing CGI Environment Variables

The `owa_util.print_cgi_env` procedure is used when testing to view the CGI environment variables.

### Syntax

```
owa_util.print_cgi_env;
```

### Return Value

This procedure returns all the CGI environment variables made available by the PL/SQL Gateway. Variables that have no value are not displayed by `owa_util.print_cgi_env`.

# Printing a Calendar

- Outputs dates in calendar style
- Accepts a query as an argument
- Returns:
  - The date
  - The text for the calendar entry
  - An optional link from the calendar entry

```
owa_util.calendarPrint('
    SELECT date_hired, name,
              'emp.htm#' || id ||
    ' FROM employee ORDER BY date_hired');
```

6-9

Copyright © Oracle Corporation, 2001. All rights reserved.

ORACLE®

## Printing a Calendar

### Syntax

```
owa_util.calendarprint( p_query in varchar2, p_mf_only in varchar2
                        DEFAULT 'N' );
```

```
owa_util.calendarprint( p_cursor in integer, p_mf_only in varchar2
                        DEFAULT 'N' );
```

### Parameters

p_query	Specifies the query
p_cursor	Specifies a cursor ID for a dynamic query (returned by owa_util.bind_variables)
p_mf_only	If this parameter is set to N, the generated calendar includes Sunday through Saturday; otherwise, it includes Monday through Friday only

This procedure creates a calendar in HTML. Each date in the calendar can contain any number of hypertext links. To achieve this effect, design your query as follows:

- The first column should be a DATE column. This is used to correlate the information produced by the query with the calendar output automatically generated by the procedure.  
**Note:** The query output must be sorted on this column using ORDER BY.
- The second column contains the text, if any, that you want printed for that date.
- The third column contains the destination for automatically generated links. Each item in the second column becomes a hypertext link to the destination given in this column.

## Entering Dates in Forms

- Form elements for day, month, and year
- The receiving parameter must be of type `owa_util.datetype`:

```
owa_util.choose_date('my_date',  
                     sysdate);
```

- The `owa_util.to_date` function converts `owa_util.datetype` to standard date type:

```
http.print('Today is ' ||  
          owa_util.to_date (my_date));
```

### Entering Dates in Forms

#### Syntax

```
owa_util.choose_date( p_name in varchar2, p_date in date  
                     DEFAULT SYSDATE);
```

#### Parameters

<code>p_name</code> IN VARCHAR2	Specifies the name for the form elements
<code>p_date</code> IN DATE	Specifies the initial date (defaults to today's date)

This procedure generates three HTML form elements that allow the user to select the day, month, and year. The parameter in the procedure that receives the data from these elements should be an `owa_util.datetype` data type. You can use the `owa_util.to_date` function to convert the `owa_util.datetype` data type value to the standard Oracle DATE data type.

# The owa\_pattern Package

- **match** Returns TRUE if the pattern is contained within a string
- **amatch** Returns an integer indicating the location of the pattern in the string
- **change** Changes the pattern within the string to a new pattern
- **Uses the owa\_text package**

## owa\_pattern Package

The owa\_pattern package contains procedures that match or replace Perl-style regular expressions. You specify a regular expression by creating the string that you want to match interspersed with various wildcard tokens and quantifiers.

Here is an example of MATCH.

```
owa_pattern.match ('BATMAN', 'Bat.*', i);
```

This is how the function is interpreted: BATMAN is searched for the regular expression Bat.\* and the function returns TRUE. Bat.\* means that we are looking for 'Bat', followed by any set of characters not including a newline. The i at the end is a flag indicating that case is to be ignored in the search.

**Note:** If multiple overlapping strings can match the regular expression, OWA\_PATTERN takes the longest match.

### Syntax

```
owa_pattern.match(string, expression, flags)
```

### Parameters

string IN VARCHAR2	The string to be searched
expression IN VARCHAR2	The expression to be found
flags IN VARCHAR2 DEFAULT NULL	If 'i', the search is case-insensitive

## **owa\_pattern Package (continued)**

### **Special Datatype**

The owa\_pattern package uses the following special datatype as an alternative way to store expressions:

pattern        A PL/SQL table of VARCHAR2 strings, indexed by BINARY INTEGER.

This has the advantage that you can use a pattern as both an input and output parameter. Thus, you can pass the same regular expression to several OWA\_PATTERN function calls and it only has to be parsed once.

The owa\_pattern.getpat procedure converts a string into the special pattern datatype.

```
owa_pattern.getpat(string, pattern)
```

### **Supported Wildcard Tokens**

<b>^</b>	New line or the beginning of the target
<b>\$</b>	New line or the end of the target
<b>\n</b>	New line
<b>.</b>	Any character except a new line character
<b>\t</b>	Tab
<b>\d</b>	Digits (0 to 9)
<b>\D</b>	Not digits (not 0 to 9)
<b>\w</b>	Alphanumeric (word) characters (0 to 9, a to z, A to Z or _)
<b>\W</b>	Not alphanumeric characters (not 0 to 9, a to z, A to Z or _)
<b>\s</b>	White-space characters (blank, tab, or new line)
<b>\S</b>	Not white-space characters (not blank, tab, or new line)
<b>\b</b>	Word boundaries (between \w and \W)
<b>\x&lt;HEX&gt;</b>	The value in the current character set of the hexadecimal number
<b>\&lt;OCT&gt;</b>	The value in the current character set of the octal number
<b>\</b>	Matches any character not covered by another case
<b>&amp;</b>	Applies only to CHANGE. This character causes the string that matched the regular expression to be included in the string that replaces it.

Any token (except &) can have its meaning extended by any of the following qualifiers:

<b>?</b>	0 or 1 occurrence
<b>*</b>	0 or more occurrences
<b>+</b>	1 or more occurrences
<b>{n}</b>	Exactly n occurrences
<b>{n,}</b>	At least n occurrences
<b>{n,m}</b>	At least n, but not more than m occurrences

### **The owa\_text Package**

This package provides ways of handling large amounts of text. It is chiefly used by the owa\_pattern package, but the functions are public so that you can use them directly.

# Calling Other Applications

- Use the UTL\_HTTP package to call other applications using the appropriate URL:

```
PROCEDURE test IS
  x utl_http.html_pieces;
BEGIN
  x:=utl_http.request_pieces
      ('http://www.oracle.com/',100);
  http.print(x.count || ' pieces were retrieved.');
```

```
FOR i IN 1..x.Count LOOP
  http.Print(x(i));
END LOOP;
END;
```

## The utl\_http Package

You can call other types of application from within your procedure using the utl\_http package, which makes HTTP calls from PL/SQL. You can use it to access data on the Internet or to call iAS applications.

You can find more details about the package in the *Oracle8i Supplied Packages Reference*.

### Syntax

```
UTL_HTTP.request_pieces(url in varchar2, max_pieces natural
                        default 32767) return html_pieces;
```

This procedure returns a PL/SQL table of 2000-byte pieces of the data retrieved from the given URL. The optional second argument places a bound on the number of pieces retrieved.

```
UTL_HTTP.request (url in varchar2) return varchar2;
```

This procedure returns up to the first 2000 bytes of the data retrieved from the given URL.

## Calling Other Applications (continued)

The following procedure retrieves up to 100 pieces of data from the URL. It displays the number of pieces retrieved and the total length, in bytes, of the data retrieved:

```
PROCEDURE test IS
  x utl_http.html_pieces;
BEGIN
  x := utl_http.request_pieces('http://www.oracle.com/', 100);
  http.print(x.count || ' pieces were retrieved. ');
  http.print('with total length ');
  If x.count < 1 THEN
    http.print('0');
  ELSE
    http.print((2000*(x.count - 1))+length(x(x.count)));
  END IF;
  FOR i IN 1..x.Count LOOP
    http.Print(x(i));
  END LOOP;
END;
```



# Summary

- **owa\_util:**
  - **Test method for CGI environment variables**
  - **Date handling**
- **owa\_pattern: Text pattern matching**
- **utl\_http: Application calls**

## Practice 6 Overview

**This practice covers the following topics:**

- **Modifying the footer procedure to include a reusable signature with CGI environment variables**
- **Calling another URL from PL/SQL**

### **Practice 6 Overview**

For solutions to this practice, see Appendix A, “Practice Solutions.”

## Practice 6

1. Modify the FOOTER procedure to include a reusable signature that allows the user to display the PL/SQL source code for the current page.
2. Add code to the `call_http` public procedure to display the contents of your `index.html` file. Use `owa_util.get_cgi_env` to dynamically construct the URL for the call. When finished, test the procedure by using the link on the LAB.MAIN\_PAGE.





## **Enhancing Client Interaction**

Copyright © Oracle Corporation, 2001. All rights reserved.

**ORACLE®**

# Objectives

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

- **Use cookies in applications**
- **Include image maps in Web pages**
- **Upload and download files**

## **Lesson Aim**

This lesson describes how status information can be preserved across client requests and used in your application. The lesson also discusses Web toolkit utilities for creating interactive images.

# Overview

- Using cookies
- Image maps
- Redirecting URLs
- Uploading and downloading files

# Cookies

- **A cookie is a string:**
  - **Sent by the server on connection**
  - **Stored by the browser**
  - **Returned to the server on future connections**
- **The developer can access the information using the `owa_cookie` package.**

## Cookies

Because HTTP is a stateless protocol, it is impossible to differentiate between visits to your Web site unless you can somehow mark them. The cookie mechanism stores and retrieves information on the client side of the connection. This simple, persistent, client-side state significantly extends the capabilities of Web applications.

For more information about cookies, visit the following URLs:

<http://www.cookiecentral.com/index2.html>

[http://home.netscape.com/newsref/std/cookie\\_spec.html](http://home.netscape.com/newsref/std/cookie_spec.html)

### How Cookies Work

When a server returns an HTTP object to the client, it may also send a piece of state information, a cookie, which the client stores in the browser's cookie file. The cookie includes a range of URLs for which it is valid. When the client makes HTTP requests that fall into the range, the client sends the cookie back to the server.

This simple mechanism provides a powerful tool:

- Sites can store registration information that saves the user from typing access details on every connection
- Sites can store user preferences that provide customization for the user
- Shopping applications can store information about the currently selected items



## owa\_cookie Package

- Cookies are sent in the HTTP header.
- Functions manipulate cookies:

Function	Description
send	Sends a value to the target cookie for the next browser call
get	Gets the value of the target cookie
remove	Deletes an expired cookie

### Setting the Header

As cookies are sent in the HTTP header, you must provide all of the necessary header information. The `owa_util.mime_header` procedure allows you to set the MIME header. The `owa_util.http_header_close` procedure closes the HTTP header.

#### Syntax

```
owa_util.mime_header(ccontent_type in varchar2 DEFAULT
                    'text/html',
                    bclose_header in boolean DEFAULT TRUE,
                    ccharset in varchar2 DEFAULT NULL);
```

#### Parameters

<code>ccontent_type</code>	The MIME type to be generated
<code>bclose_header</code>	If TRUE, the HTTP header is closed; if the HTTP header remains open, you close it explicitly using <code>owa_util.http_header_close</code>
<code>ccharset</code>	The character set to be used

#### Generated Code

Content-type: <ccontent\_type>\n

# Using Cookies

```
CREATE PROCEDURE counter IS
    v_cookie      owa_cookie.cookie;
BEGIN
    v_cookie := owa_cookie.get('counter');
    owa_util.mime_header('text/html',FALSE);
    IF v_cookie.num_vals > 0 THEN
        owa_cookie.send('counter',v_cookie.vals(1)+1);
    ELSE owa_cookie.send('counter',1);
    END IF;
    owa_util.http_header_close;
    http.print('You have visited here
               '||v_cookie.vals(1)||' times');
END;
```

## Using Cookies

- `owa_cookie.send` stores the cookie information.
- `owa_cookie.get` returns the requested cookie as a record of special datatype `COOKIE`.

### Syntax

`owa_cookie.send (name,value,expires,path,domain,secure)`

This procedure adds a Set Cookie line to the HTTP header:

```
Set-Cookie: NAME=VALUE; expires=DATE; domain=DOMAIN; path=PATH;
secure
```

- `EXPIRES` is the time at which the cookie information is forgotten by the browser.
- `DOMAIN` and `PATH` are used to determine if there are any valid cookies for a server.
- `SECURE` indicates that the cookie is to be passed only over an SSL connection.

You can only use this procedure between `owa_util.mime_header` and `owa_util.http_header_close`.

# Cookies and Application Navigation

- Recognize and authenticate user
- Maintain a record of where the user has been:
  - Set a cookie as a page is requested
  - Check that the cookie exists before letting the user visit another page

## Cookies and Application Navigation

Another use of cookies is to ensure that pages in an application are visited in a particular order or only after any prerequisite pages are visited.

**Note:** Users can disable the support for cookies in their browser.

# Using Image Maps

- **In development:**
  - **Name the map**
  - **Divide it into clickable areas**
  - **Specify the image**
- **At runtime:**
  - **Click on the image**
  - **The browser processes the coordinates**

## Using Image Maps

An image map is a picture that has clickable regions: when the user clicks a region, the appropriate link is invoked. To create an image map, you must use the following PL/SQL procedures:

```
http.mapOpen(map_name);           -- Name the map
    http.area(...);               -- Specify the regions
http.mapClose;                    -- Close the map
http.img2('map.gif', cismap=>TRUE, -- Specify the image and link to the regions
          cusemap=>map_name);
```

## Opening and Closing the Map

```
http.mapOpen( 'map1' );  
    .  
    .  
    .  
    .  
http.mapClose;  
    .  
    .
```

Procedure Call

```
<MAP NAME="map1">  
    .  
    .  
    .  
    .  
</MAP>  
    .
```

Generated HTML

7-9

Copyright © Oracle Corporation, 2001. All rights reserved.

ORACLE®

### Opening a Map

#### Syntax

```
http.mapOpen(cname, cattributes);
```

#### Parameters

cname in varchar2

cattributes in varchar2 DEFAULT NULL

#### Generated Code

```
<MAP NAME="cname" cattributes>
```

### Closing a Map

#### Syntax

```
http.mapClose;
```

#### Generated Code

```
</MAP>
```

## Creating the Links

```
http.mapOpen( 'map1' );  
    http.area( '0,0,50,50', 'rect',  
              'http://education.oracle.com' );  
    http.area( '0,50,50,100', 'rect',  
              'http://www.oracle.com' );  
http.mapClose;
```

Procedure Call

```
<MAP NAME="map1">  
    <AREA SHAPE="rect" COORDS="0,0,50,50"  
        HREF="http://education.oracle.com">  
    <AREA SHAPE="rect" COORDS="0,50,50,100"  
        HREF= "http://www.oracle.com">  
  
</MAP>
```

Generated HTML

7-10

Copyright © Oracle Corporation, 2001. All rights reserved.

ORACLE®

### Creating Clickable Regions

A clickable region may be:

- A rectangle, defined by the coordinates of its top left and bottom right corners
- A circle, defined by the coordinates of its centre and the radius
- A polygon, defined by the coordinates of its corners

#### Syntax

```
http.area(ccoords, cshape, chref, cnohref, ctargert, cattributes);
```

#### Parameters

```
ccoords in varchar2  
cshape in varchar2 DEFAULT NULL  
chref in varchar2 DEFAULT NULL  
cnohref in varchar2 DEFAULT NULL  
ctargert in varchar2 DEFAULT NULL  
cattributes in varchar2 DEFAULT NULL
```

#### Generated Code

```
<AREA COORDS="ccoords" SHAPE="cshape" HREF="chref" NOHREF  
    TARGET="ctargert" cattributes>
```

# Specifying the Image

```
http.mapOpen('map1');
    http.area('0,0,50,50', 'rect',
              'http://education.oracle.com');
    http.area('0,50,50,100', 'rect',
              'http://www.oracle.com');
http.mapClose;
http.img2('map.gif', cismap=>1, cusemap=>'#map1');
```

Procedure Call

```
<MAP NAME="map1">
  <AREA SHAPE="rect" COORDS=0,0,50,50
        HREF= "http://education.oracle.com">
  <AREA SHAPE="rect" COORDS=0,50,50,100
        HREF= "http://www.oracle.com">
</MAP>

<IMG SRC="map.gif" ISMAP USEMAP="#map1">
```

Generated HTML

## Image Element

### Syntax

```
http.img2(curl, calign, calt, cismap, cusemap, cattributes);
```

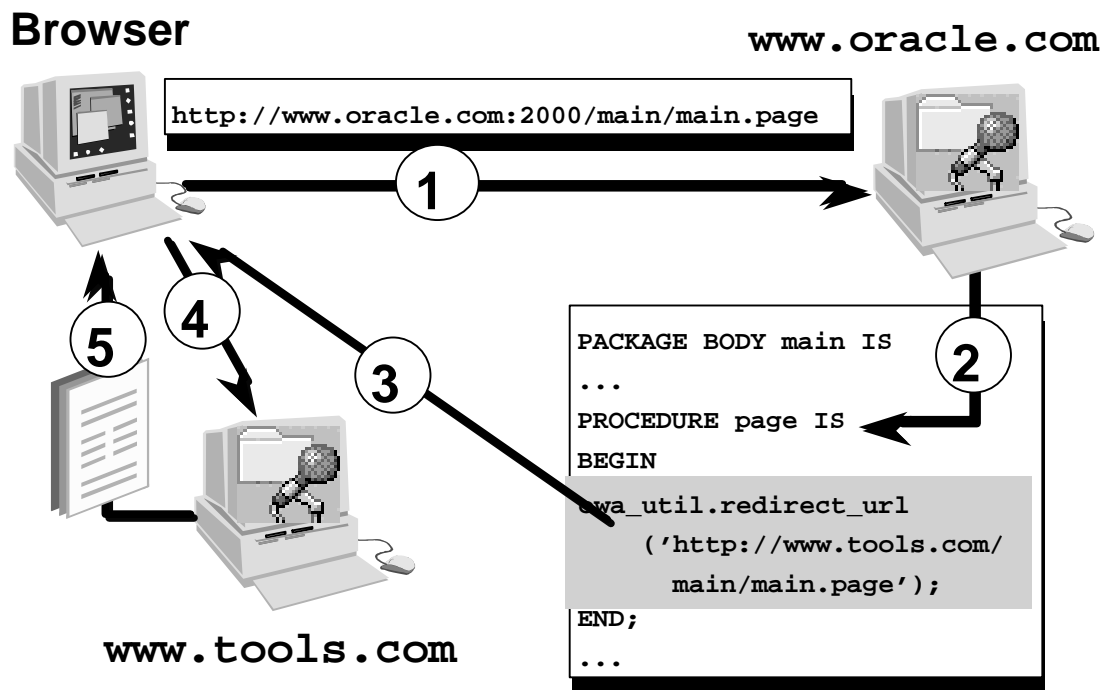
### Parameters

curl	IN VARCHAR2 DEFAULT NULL	the URL that identifies the image
calign	IN VARCHAR2 DEFAULT NULL	the alignment of text
calt	IN VARCHAR2 DEFAULT NULL	
cismap	IN VARCHAR2 DEFAULT NULL	if not null, this is an image map
cusemap	IN VARCHAR2 DEFAULT NULL	use the areas in specified <map>
cattributes	IN VARCHAR2 DEFAULT NULL	

### Generated Code

```
<IMG SRC="curl" ALIGN="calign" ALT="calt" ISMAP USEMAP="cusemap"
cattributes>
```

# Redirecting a URL



7-12

Copyright © Oracle Corporation, 2001. All rights reserved.

ORACLE®

## Redirecting the URL

You can use `owa_util.redirect_url` to redirect a request to another target. The URL may specify either a Web page to be returned or a program to be executed. This procedure must come before any `http.print` or `http.prn` calls in order to tell the PL/SQL Gateway to perform the redirect operation.

### Syntax

```
owa_util.redirect_url(curl in varchar2, bclose_header in boolean
                      DEFAULT TRUE);
```

### Parameters

<code>curl</code>	IN VARCHAR2	Specifies the URL to visit.
<code>bclose_header</code>	IN BOOLEAN	Specifies whether or not to close the HTTP header.



# Uploading and Downloading Files

- **DAD configuration parameters**
  - **document\_table**
  - **document\_path**
  - **document\_proc**
- **Document table**

## Uploading and Downloading Files

The PL/SQL Gateway allows you to transfer files from a client machine to and from a database (uploading and downloading). The files are uploaded into a document table and a primary key is passed to the handler routine so that it can retrieve the appropriate row of the table. You can perform the following operations through the PL/SQL Gateway:

- Upload multiple files per form submission
- Upload and download files as raw byte streams without any character set conversions
- Specify a document table per DAD, so that files from different applications are not mixed

## DAD Configuration

In order to upload and download files, the DAD associated with the PL/SQL Gateway must be configured to use the appropriate table and handling procedures:

- Document table: table in which the documents will be stored
- Document access path: path entered into URL to indicate document transfer
- Document access procedure: procedure used to download documents

## Document Table Structure

```
CREATE TABLE documents (  
    NAME                VARCHAR2(128)  
                        UNIQUE NOT NULL,  
    MIME_TYPE           VARCHAR2(128),  
    DOC_SIZE            NUMBER,  
    DAD_CHARSET         VARCHAR2(128),  
    LAST_UPDATED       DATE,  
    CONTENT_TYPE        VARCHAR2(128),  
    CONTENT             LONG RAW,  
    BLOB_CONTENT        BLOB  
);
```

### Document Table Structure

You can name the table anything that you like but the columns must be named as shown. You do not need both the CONTENT and BLOB\_CONTENT columns; for any one row, only one of the content columns is used, as indicated by the CONTENT\_TYPE column. You can set the upload\_as\_long\_raw and upload\_as\_blob DAD parameters to specify into which column the content is loaded, based on the file extension.

The value of each of these parameters is a comma separated list of file extensions. Files with extensions listed in the upload\_as\_long\_raw parameter are uploaded into the CONTENT column. Files with extensions listed in the upload\_as\_blob parameter are uploaded into the BLOB\_CONTENT column.

The file extensions can be text literals (such as jpeg and gif). In addition, you can add an asterisk (\*) to match any file extension that is not explicitly listed.

#### Syntax

upload\_as\_long\_raw = file\_extension, file\_extension, \*

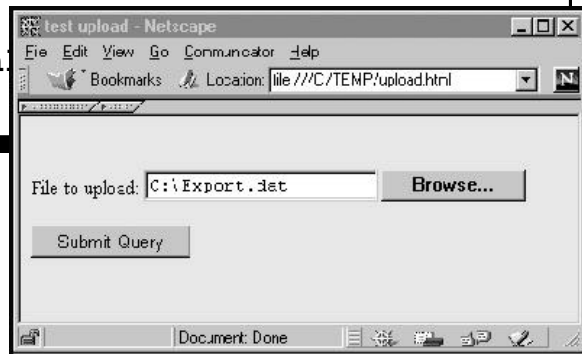
#### Examples

upload\_as\_long\_raw = html, txt, \*

upload\_as\_long\_blob = gif

# Uploading Files

```
http.Title('Test upload');  
http.formOpen('write_info',  
              enctype=>'multipart/form-data');  
http.print('File to upload: ');  
http.formFile('file');  
http.Para;  
http.FormSubmit(cva);  
http.formClose;
```



7-15

Copyright © Oracle Corporation, 2001. All rights reserved.

ORACLE®

## Uploading Files

To upload files from a client machine to a database, you create an HTML form:

- Set the `enctype` attribute of the `FORM` element to `multipart/form-data` and the `action` attribute to the procedure that writes the data to the table:  

```
http.formOpen('write_info', enctype=>'multipart/form-data');
```
- Create an `INPUT` element whose `type` and `name` attributes are set to `file`. The `INPUT type=file` element enables a user to browse and select files from the file system.  

```
http.formFile('file');
```

When a user clicks the Submit button, the action procedure (for example, `write_info`) is called:

1. The file named in the `INPUT type=file` element is uploaded to the table named in the `DAD document_table` parameter.
2. The action procedure returns a page to the user. It does not have to return anything to the user, but it is a good idea to let the user know whether the upload operation succeeded or failed.

# Uploading Files

```
http.Title('Test upload');
http.formOpen( 'write_info',
               'cncv' || 'multipart/form-data');
http.pr
h

PROCEDURE write_info (file in varchar2) IS
BEGIN
    http.Print('Uploaded ' || file || ' successfully');
END write_info;
```

## Uploading Files (continued)

The action procedure must have an input parameter whose name is the same as the file element submitted by the form. If the user can upload only one file at a time, the file parameter is of type VARCHAR2. If the user can upload more than one file at a time, the file parameter is of type owa.vc\_arr.

The filename obtained from the browser (and saved in the documents table) is prefixed with a generated directory name to reduce the possibility of name conflicts. For instance, when /private/minutes.txt is uploaded, the name stored in the table by the gateway may be F9080/minutes.txt.

## Uploading Multiple Files

To upload multiple files in one submit action, the upload form must include multiple INPUT elements of type file. If they also have the same name, then they are passed to the action procedure as an array of type owa.vc\_arr. If they have different names, the action procedure must declare each of them as type VARCHAR2.

For example, if a form contains the following elements:

```
<INPUT type="file" name="tfiles">
<INPUT type="file" name="tfiles">
<INPUT type="file" name="bfile">
```

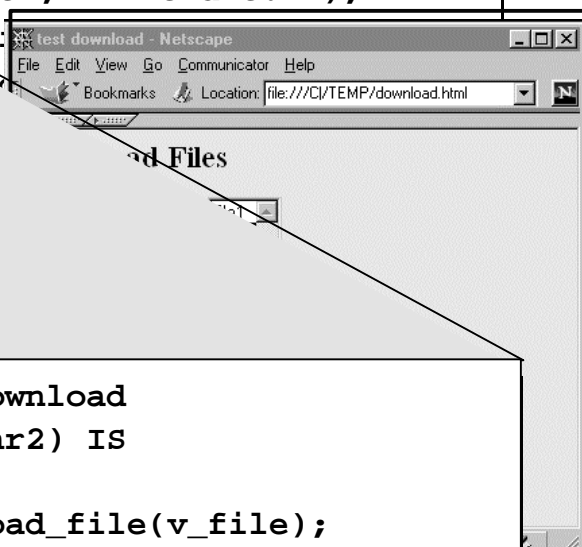
then the action procedure must contain the following parameters:

```
procedure write_files(tfiles IN owa.vc_arr, bfile IN varchar2)
```

# Downloading Files

```
http.Title('Test download');  
http.formOpen(' process_download ');  
  http.formSelectOn('v_file', 'Filename: ');  
  FOR current  
    http.form  
  END LOOP;  
  http.form  
  http.Form'  
http.formC'
```

```
PROCEDURE process_download  
  (v_file in varchar2) IS  
BEGIN  
  wpg_docload.download_file(v_file);  
END process_download;
```



7-17

Copyright © Oracle Corporation, 2001. All rights reserved.

ORACLE®

## Downloading Files

After you have uploaded files to the database, you can download them, delete them from the database, and read and write their attributes. You cannot download multiple files.

The HTML page presented to the user has a link to a URL which includes the document access path for the DAD and specifies the file to be downloaded. For example, if the document access path is docs and the file is x.txt, the URL would be:

```
http://www.oracle.com/pls/ias_dad/docs/x.txt
```

When the user presses the submit button, the named file is downloaded from the document table and passed back to the browser.

To download a file, you must create a stored procedure that calls the document download packages included in the PL/SQL Web Toolkit:

```
wpg_docload.download_file(file_name)
```

The procedure that downloads files can invoke queries and other procedures, but it cannot write strings that are returned to the browser. It can return only the contents of a file.

For information about more utilities that enable you to manipulate the database file repository, refer to *The owa\_content Package*.

# Summary

- **Cookie management subprograms**
- **Handling images**
- **Redirecting URLs**
- **Uploading and downloading files**

## Practice 7 Overview

**This practice covers the following topics:**

- **Creating procedures to set a cookie and to get the value of a cookie**
- **Modifying the main menu procedure that uses cookies to display the number of times a particular individual has visited that page**

### **Practice 7 Overview**

For solutions to this practice, see Appendix A, “Practice Solutions.”

## Practice 7

The purpose of this practice is to use cookies to count the number of times a particular individual has visited LAB.MAIN\_PAGE. Note that you are counting the number of times a particular individual has visited this page and not the total number of hits for the page.

1. Add code to the GET\_COOKIE private function, which accepts one parameter (v\_cookie\_name) and returns the value of the cookie, v\_cookie\_name, or 0, if the value of the cookie has not been set.
2. Add code to the SET\_COOKIE private procedure, which accepts two parameters (v\_cookie\_name and v\_cookie\_val). The v\_cookie\_name parameter is the name of the cookie that is being created. The v\_cookie\_val parameter is the cookie's value. Make sure that the cookie expires once the user exits the browser.
3. Modify procedure MAIN\_PAGE so that it sets a cookie value and displays the number of times an individual has visited the page using SET\_COOKIE and GET\_COOKIE. Test your procedure by reloading LAB.MAIN\_PAGE several times and verify that the cookie is incremented.



# 8

## **Validating Data and Executing Transactions**

Copyright © Oracle Corporation, 2001. All rights reserved.

**ORACLE®**

# Objectives

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

- **Validate data**
- **Perform DML in Web applications**

## **Lesson Aim**

This lesson shows you how to use data manipulation language (DML) commands in Web applications and explains the methods available for data validation. The lesson also discusses how database transactions can be controlled in applications.

# Overview

- **DML and data validation issues**
- **Handling concurrency:**
  - **Locking**
  - **Using a checksum**
- **Web transactions**

# DML in Web Applications

- **Browsers have no native validation methods.**
- **The HTTP protocol is stateless.**
- **Applications cannot use:**
  - **Ordinary locking facilities**
  - **Transactions**

## DML in Web Applications

One of the biggest disadvantages of the Web as a medium for application distribution has been the inability of applications to address the issues listed in the slide. Oracle Internet Application Server breaks new ground in its ability to get around these issues in a robust and reliable manner.

# Data Validation Methods

- Database integrity constraints and triggers
- PL/SQL in the database
- A scripting language in the browser

## Data Validation Methods

Data validation should:

- Be simple to execute
- Provide immediate feedback
- Not waste resources
- Be consistent across all applications that access your data

The solutions discussed in this part of the lesson all satisfy one or more of these criteria, but none can satisfy them all. You can choose to:

- Send the data to the database and execute DML; the database integrity constraints and triggers then check the data
- Send the data to the database and use PL/SQL to validate the data before executing DML
- Use a scripting language (for example, Javascript) to check the data in the browser

Your choice of validation method may depend on the available resources and desired outcome.

## Concurrency Problem

1. User A calls up employee data for Scott.
2. User B modifies Scott's data.
3. User B submits form and commits changes.
4. User A decides to modify data.
5. User A submits form.

**Should user A be made aware of the changes user B has made?**

### Concurrency Problem

In a multiuser environment, concurrency is a major issue. Unlike other Oracle products (such as Oracle Forms), there is no automatic mechanism for ensuring concurrency; instead, the developer is responsible for implementing the appropriate mechanisms.

# Possible Concurrency Solutions

- **Keep a copy of the data**
- **Construct a row checksum**

## Possible Concurrency Solutions

To maintain concurrency, you can:

- Keep a copy of the selected data in hidden fields and check it with the current database values before executing a DML command
- Construct a checksum on a row

Both of these techniques are “optimistic”: they do not prevent other users from performing updates, but they reject the current update if an intervening update has occurred. The `owa_opt_lock` package supports both mechanisms.

# Copying the Data

- Store column values in hidden fields

**old\_***tablename*

```
CURSOR selected_emp IS
SELECT rowid, emp.* FROM emp WHERE empno=1206;
...
http.formOpen('app.process_form');
FOR x IN selected_emp LOOP
owa_opt_lock.store_values(user,'emp',x.rowid);
... <build other fields here>
END LOOP;
http.formClose;
```

8-8

Copyright © Oracle Corporation, 2001. All rights reserved.

ORACLE®

## Copying the Data

The example above assumes that only one row is identified by the SELECT statement.

### Syntax

```
owa_opt_lock.storevalues (p_owner,p_tname,p_rowid);
```

### Parameters

p_owner	IN VARCHAR2	Specifies the owner of the table
p_tname	IN VARCHAR2	Specifies the name of the table
p_rowid	IN ROWID	Specifies the identifier of the row for which you want to store values

When you use the owa\_opt\_lock package, the record is locked.

This procedure generates a series of hidden fields in the HTML form. The fields are accessed by referencing old\_p\_tname.

When the form is submitted, the subprogram that handles the data must handle the hidden fields as a PL/SQL table. The owa\_opt\_lock package provides the owa\_opt\_lock.vcarray data type (a VARCHAR2 PL/SQL table).



## Checking Copied Data

```
PROCEDURE do_update
    (old_emp IN owa_opt_lock.vcArray,
     <all the new values>) IS
    the_rowid rowid;
BEGIN
    the_rowid := owa_opt_lock.get_rowid(old_emp);
    IF (owa_opt_lock.verify_values(old_emp)) THEN
        UPDATE <new values> where rowid=the_rowid;
    ELSE <Tell user the data has changed>;
    END IF;
END;
```

### Checking Copied Data

#### Syntax

```
owa_opt_lock.verify_values (p_old_values);
owa_opt_lock.get_rowid (p_old_values);
```

#### Parameters

p_old_values	IN	vcArray	Specifies a PL/SQL table containing hidden field values
--------------	----	---------	---

The `verify_values` function returns TRUE if the values stored in the PL/SQL table match the database values.

The `get_rowid` function returns the ROWID of the row being modified.

# Using a Row Checksum

```
CURSOR selected_emp IS
SELECT rowid, emp.* FROM emp WHERE empno=1206;
...
http.formOpen('app.process_data');
FOR x IN selected_emp LOOP
    http.formHidden('the_rowid',x.rowid);
    http.formHidden('the_checksum',
        owa_opt_lock.checksum(user,'emp',x.rowid));
    ...<build other fields here>
END LOOP;
http.formClose;
```

## Using a Row Checksum

The example above assumes that only one row is identified by the SELECT statement.

### Syntax

```
owa_opt_lock.checksum (p_owner, p_tname, p_rowid);
```

### Parameters

p_owner	IN VARCHAR2	Specifies the owner of the table
p_tname	IN VARCHAR2	Specifies the name of the table
p_rowid	IN ROWID	Specifies the ROWID of the row for which you want to calculate the checksum

This function returns a checksum value for a specified string or for a row in a table. For a row in a table, the function calculates the checksum value based on the values of the columns in the row. This function comes in two versions:

- The checksum is based on a specified string; this is a “pure” 32-bit checksum
- The checksum is based on the values of a row in a table; this is an “impure” 32-bit checksum

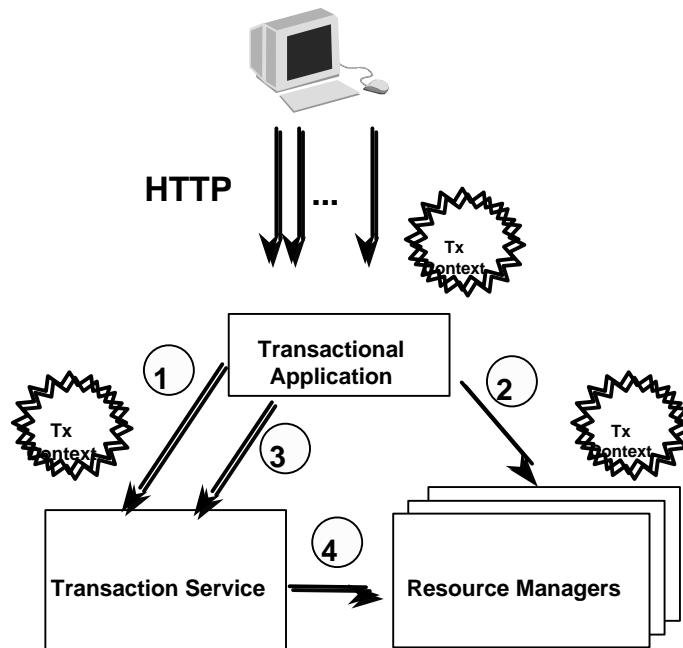
## Verifying the Checksum

```
PROCEDURE do_update(the_checksum IN number,  
                    the_rowid    IN varchar2,  
                    <all the new values>) IS  
BEGIN  
    IF the_checksum = owa_opt_lock.checksum  
        (user,'emp',x.rowid))  
    THEN  
        UPDATE <new values> where rowid=the_rowid;  
    ELSE  
        <Tell user data has changed>;  
    END IF;  
END;
```

### Verifying the Checksum

The checksum is recalculated on the row with the stored ROWID. Only if the values match should the data be modified.

# Web Transaction Architecture



8-12

Copyright © Oracle Corporation, 2001. All rights reserved.

ORACLE®

## Web Transaction Architecture

A Web application is invoked by an HTTP request originating from a browser, using a stateless protocol. Although HTTP is stateless, *iAS* must map requests originating from the same browser to a transaction context object.

There are two possible solutions:

- Map requests from one browser to the same session. This way the transaction context can be stored in the Gateway state and reused by consecutive calls to the same instance.  
*iAS* provides a similar solution for JServlet. Here the instance is not locked to one browser but a separate session state object is maintained for all of the requesting browser sessions. The transaction state is stored in the session object on the application server.
- Propagate the transaction context to the browser. The browser sends back the locally stored context with the next request to the Gateway. In practice, it is not the transaction context that is propagated to the browser, but an external representation of the reference to a transaction context object. After receiving this external representation, the Web application can recreate the object reference.

In both solutions, cookies are usually involved to maintain mapping of a browser to a server object.

# Transactions in iAS

- **The PL/SQL Gateway: stateless**
- **The Oracle Servlet Engine: stateless and stateful**

## Transactions in iAS

iAS provides two configurations for deploying PL/SQL-based Web applications:

- The PL/SQL Gateway runs only in stateless mode but it has a connection pooling mechanism that can keep database sessions open between HTTP requests.
- The Oracle Servlet Engine (OSE) of Oracle 8i supports both stateless and stateful modes. This means that a new database session does not have to be created and destroyed for every HTTP request. For more information, see the *Oracle8i Oracle Servlet Engine User's Guide*.

# Summary

- **Using DML**
- **Data validation**
- **Concurrency**
- **Transactions**

## Practice 8 Overview

**This practice covers the following topics:**

- **Computing a checksum for updating or deleting customer records**
- **Modifying procedures for updating, deleting, and inserting customer records to use `owa_opt_lock`**

### **Practice 8 Overview**

For solutions to this practice, see Appendix A, “Practice Solutions.”

## Practice 8

1. Modify the LAB.MODIFY\_CUSTOMER procedure to compute a checksum for the Update and Delete options and pass them as hidden fields to UPDATE\_CUSTOMER and DELETE\_CUSTOMER. Note that the Insert option does not require a checksum.
2. Modify the LAB.UPDATE\_CUSTOMER procedure to first compute a checksum to verify that the update should be allowed. Use the ERROR\_PAGE or SUCCESS\_PAGE procedure, with appropriate messages, to indicate the success or failure of the update.
3. Modify the LAB.DELETE\_CUSTOMER procedure to first compute a checksum to verify that the delete should be allowed. Use the ERROR\_PAGE or SUCCESS\_PAGE procedure, with appropriate messages, to indicate the success or failure of the update.



# 9

## Creating Display Areas with Frames

Copyright © Oracle Corporation, 2001. All rights reserved.

**ORACLE®**

# Objectives

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

- **Present Web site documents within frames**
- **Provide a no-frames alternative for displaying documents**

## Lesson Aim

One powerful technique for displaying complex sets of information is to partition them into frames. In this lesson, you learn how to design and assign content to documents with frames.

For more information on frames, see:

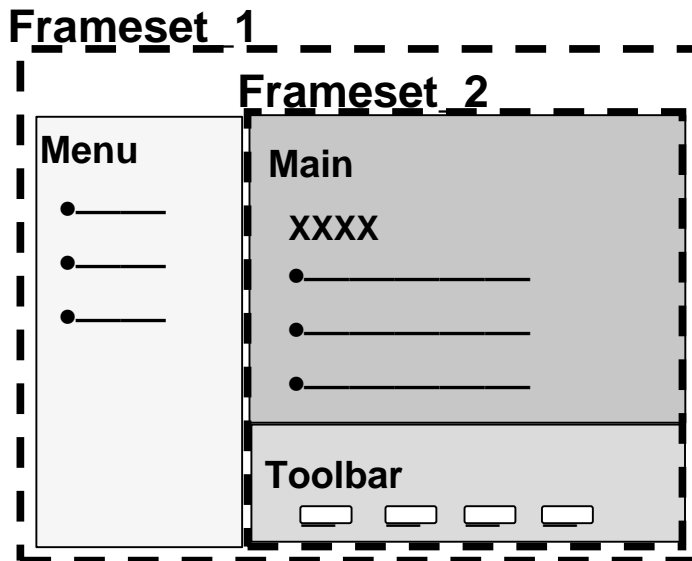
- *Frames: An Introduction* at  
[http://home.netscape.com/assist/net\\_sites/frames.html](http://home.netscape.com/assist/net_sites/frames.html)
- *Netscape Frames* at  
<http://sharky.nomius.com/>
- *Framing the Web* at  
<http://webreference.com/>

# Overview

- **Frame concepts:**
  - **Framesets**
  - **Layout versus content documents**
- **Creating a frame-based document using HTP**
- **Creating a frameset**
- **Directing links to a frame**
- **Frame design considerations**

# What Is a Frame?

- Content area
- Frameset
- Frame
- Target
- No frame



9-4

Copyright © Oracle Corporation, 2001. All rights reserved.

ORACLE®

## Frame Terminology

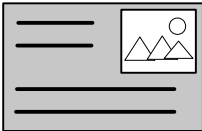
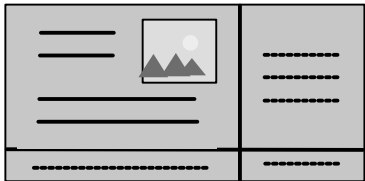
The content area is the whole window of the Web browser. You can partition it into frames.

- The frameset divides the content area into groups of frames.
- The frame is a named content container.
- The target specifies the location for the requested content.
- No frame specifies how the page is presented when viewed by a Web browser that does not support frames.

Not all Web browsers support frames.

# Layout Documents Versus Content Documents

Layout Document	Content Document
<ul style="list-style-type: none"><li>• Frame dimensions (columns and rows)</li><li>• Frame name (target)</li><li>• Initial content document for frame</li><li>• A no-frame alternative</li></ul>	<ul style="list-style-type: none"><li>• User information</li></ul>



# Creating a Layout Document

```
PACKAGE BODY f_pac AS ...
  PROCEDURE show is BEGIN
    http.htmlopen; http.headopen; http.title('Frame'); http.headclose;
    http.framesetopen(NULL, '30%', '*');
    http.frame('/pls/dad/f_pack.toc');
    http.framesetopen('*',65');
    http.frame('/pls/dad/f_pack.home', 'main_win');
    http.frame('/pls/dad/f_pack.foot', cscrolling=>'NO');
    http.framesetclose;
    http.framesetclose;
    http.noframesopen;
    http.bodyopen; ... http.bodyclose; http.noframesclose;
    http.htmlclose;
  END show;
END f_pack;
```

Procedure

## Creating a Layout Document

`http://servername:2000/pls/dad/`

`f_pack.show`

```
http.htmlopen;  
  http.headopen; http.title('Frame'); http.headclose;  
  ...
```

```
http.htmlclose;
```

**Procedure Call**

```
<HTML>  
  <HEAD><TITLE>Frame</TITLE></HEAD>  
  ...
```

```
</HTML>
```

**Generated HTML**

# Creating a Frameset

```
http.htmlOpen; ...
  http.framesetOpen (NULL,'30%', '*');
  ...
    http.framesetOpen ('*',65');
    ...
    http.framesetClose;
  http.framesetClose;
...
```

**Procedure Call**

```
<HTML> ...
  <FRAMESET COLS="30%,*">
  ...
    <FRAMESET ROWS="*,65">
    ...
    </FRAMESET>
  </FRAMESET>
  ...
```

**Generated HTML**

9-8

Copyright © Oracle Corporation, 2001. All rights reserved.

**ORACLE®**

## Creating a Frameset

### Syntax

```
http.framesetOpen (crows, ccols, cattributes);
```

### Parameters

crows	IN VARCHAR2 DEFAULT NULL
ccols	IN VARCHAR2 DEFAULT NULL
cattributes	IN VARCHAR2 DEFAULT NULL

Values for crows and ccols are \*, number, and number%.

### Generated HTML

```
<FRAMESET ROWS="crows" COLS="ccols" cattributes>
```

You must specify either crows or ccols.

## Closing a Frameset

If you have nested framesets, you must close each one.

### Syntax

```
http.framesetClose;
```

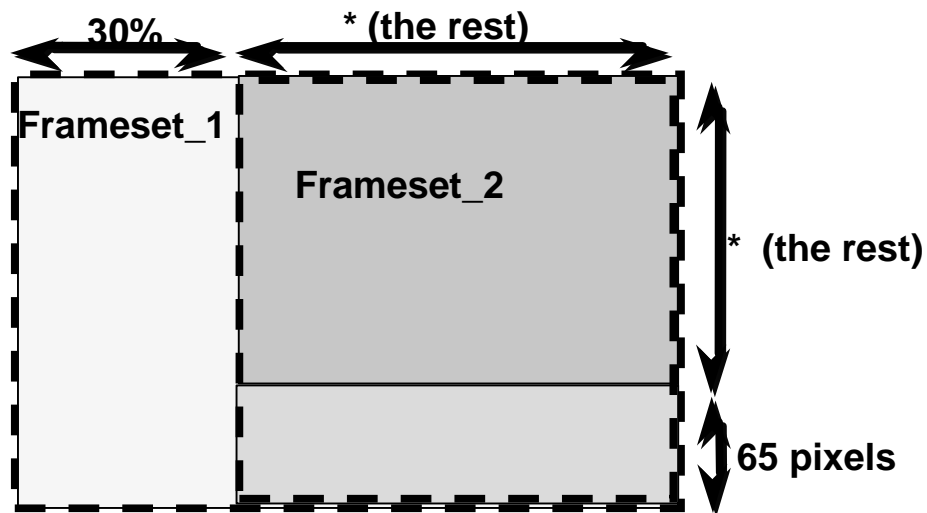
### Generated HTML

```
</FRAMESET>
```



## Frameset Dimensions

```
http.framesetOpen (NULL, '30%', '*');  
http.framesetOpen ( '*', 65 );
```



9-9

Copyright © Oracle Corporation, 2001. All rights reserved.

ORACLE®

### Frameset Dimensions

In the example in the slide, a two-column frameset contains a frame and a two-row nested frameset.

## Defining the Frame

```
http.htmllopen; ...
```

```
    http.framesetopen(NULL, '30%', '*');
```

```
        http.frame ('/pls/dad/f_pack.toc');
```

```
    http.framesetopen('*', '65');
```

```
        http.frame('/pls/dad/f_pack.home', 'main_win');
```

```
        http.frame('/pls/dad/f_pack.foot',  
                   cscrolling=>'NO');
```

```
    http.framesetclose; ...
```

```
http.htmlclose;
```

**Procedure Call**

```
<HTML> ...
```

```
<FRAMESET COLS="30%,*">
```

```
    <FRAME SRC="/pls/dad/f_pack.toc">
```

```
<FRAMESET ROWS="*,65">
```

```
    <FRAME SRC="/pls/dad/f_pack.home" NAME="main_win">
```

```
    <FRAME SRC="/pls/dad/f_pack.foot" SCROLLING="NO">
```

```
</FRAMESET>
```

```
...
```

**Generated HTML**

9-10

Copyright © Oracle Corporation, 2001. All rights reserved.

**ORACLE®**

## Defining the Frame

### Syntax

```
http.frame(csrc, cname, cmarginwidth, cmarginheight, cscrolling,  
           cnoresize, cattributes);
```

### Parameters

csrc	IN VARCHAR2
cname	IN VARCHAR2 DEFAULT NULL
cmarginwidth	IN VARCHAR2 DEFAULT NULL
cmarginheight	IN VARCHAR2 DEFAULT NULL
cscrolling	IN VARCHAR2 DEFAULT NULL
cnoresize	IN VARCHAR2 DEFAULT NULL
cattributes	IN VARCHAR2 DEFAULT NULL

Values for cscrolling are YES, NO, and AUTO (default).

### Generated HTML

```
<FRAME SRC="csrc" NAME="cname" MARGINWIDTH= "cmarginwidth"  
MARGINHEIGHT="cmarginheight" SCROLLING="cscrolling" NORESIZE  
cattributes>
```

## Creating a Container for Frame-Incapable Browsers

```
http.noframesOpen;  
...  
http.noframesClose;
```

Procedure Call

```
<NOFRAMES>  
<BODY>  
...  
</BODY>  
</NOFRAMES>
```

Generated HTML

### Creating a Container for Frame-Incapable Browsers

Remember to include a no-frame alternative for browsers that cannot handle frames.

The no-frame alternative is ignored by the browsers that can handle frames. It contains the body of the document to be displayed without frames.

#### Syntax

```
http.noframesOpen;  
http.noframesClose;
```

#### Generated HTML

```
<NOFRAMES>  
</NOFRAMES>
```

# Directing Links to a Frame

- **Set the TARGET attribute:**
  - *name*
  - `_self`
  - `_parent`
  - `_top`
  - `_blank`
- **Use the `http.anchor2` procedure**
- **Use the `http.base` procedure**

## Populating a Frame

The value of the target attribute determines the window in which the document is to appear. It may have the following values:

- *name* populates the named window or a new window if no window by that name currently exists.
- `_self` repopulates the window in which the link was selected.
- `_parent` loads into the immediate parent window.
- `_top` fills the entire browser window, replacing the frames.
- `_blank` populates a new, unnamed browser window.

The `http.base` procedure establishes the default target window for all links in this document. The `http.anchor2` procedure specifies the TARGET attribute as well as the URL and name.

The TARGET attribute of the anchor tag (`<A>`) overrides the target indicated in the `<BASE>` tag.

## Populating a Frame with `http.base`

```
http.base ('main_win');
```

Procedure Call

```
...
```

```
<BODY>
```

```
<BASE HREF="fruit.html" TARGET="main_win">
```

```
...
```

Generated HTML

### Populating a Frame with `http.base`

You can embed the BASE element in:

- The header section, to include the absolute path name for the URL
- The body section, to determine the default window name for the document

#### Syntax

```
http.base(ctarget, cattributes);
```

#### Parameters

ctarget                    IN VARCHAR2 DEFAULT NULL

cattributes               IN VARCHAR2 DEFAULT NULL

Values for `ctarget` are `name`, `_self`, `_parent`, `_top`, and `_blank`.

#### Generated HTML

```
<BASE HREF="current_URL" TARGET="ctarget" cattributes>
```

# Summary

## **PL/SQL toolkit frame tag procedures:**

- **Open and close framesets**
- **Specify frame contents**
- **Open and close the no-frame container**

## Practice 9 Overview

**This practice covers the following topics:**

- **Creating a document that contains:**
  - One frame for the main menu
  - One frame for the query form and results
  - One frame for a footer
- **Creating no-frame alternatives:**
  - A link to the no-frame version
  - Support for frame-incapable browsers

### **Practice 9 Overview**

For solutions to this practice, see Appendix A, “Practice Solutions.”

## Practice 9

1. Create a public procedure named LAB.FRAME\_VERSION that displays the application main page using frames. This procedure accepts no parameters. It subdivides the content area into three framesets. The first frameset creates two columns, where the first column occupies 25% of the area. The second frameset creates two rows, where the second row occupies 25% of the area.

Name the large frame MAIN\_FRAME. It should use the LAB.FRAME\_INTRODUCTION procedure as its source. The side frame should contain the LAB.FRAME\_TOC procedure, and the bottom frame should contain the LAB.FRAME\_FOOTER procedure.

**Note:** Be sure to code a no-frames alternative!

2. Create a public procedure named LAB.FRAME\_TOC. This procedure is based on the anchors in MAIN\_PAGE (except the last one, which is used to switch to the frames version of the application) but should use the HTP.BASE procedure to set the default TARGET of the document to MAIN\_FRAME.
3. When finished, test the procedure by using the link on LAB.MAIN\_PAGE.



---

**A**

.....

## **Practice Solutions**

## Practice 1 Solutions

Your instructor will provide you with login information. Make a note of this information here. You will need to refer to it throughout this course:

1. Database (username/password@connect\_string):
  - username:
  - password:
  - connect string:
2. UNIX server (or WinNT server)
  - hostname or IP:
  - username:
  - password:
  - \$HOME:
3. Client machine
  - username:
  - domain:
  - password:
4. iAS Administration pages
  - URL: `http://`  
(Usually `http://hostname.domain/pls/dad/admin_/gateway.htm`)
  - username:
  - password:
5. URLs for your application
  - Network port:
  - PL/SQL virtual path:  
(Usually `pls/dad`)
  - Home page:  
(Usually `http://hostname.domain/index.html`)
6. URLs for the course applications
  - Submit application: `http://`  
(Usually `http://hostname.domain/pls/ias_dad/submit.main_page`)
  - Practice application: `http://`  
(Usually `http://hostname.domain/pls/ias_dad/lab.main_page`)

### **Practice 1 Solutions (continued)**

7. Use the URL provided in Question 6 to access the Summit demonstration application. Perform the following tasks to familiarize yourself with the application:

- a. View a listing of all customers.
- b. Beisbol Si!, customer number 209, has just been taken over by a major multinational company. Change the telephone number to 447-9967 and add a comment: Pending new credit details no more orders are to be taken for this customer.
- c. Add your own company to the customer listings.
- d. Navigate to the frame-based version of the application. View a list of all customers from the frames version of the application.

**URL: See Practice 1, Question 6.**

7. a.
  - i. Click Full Customer Listing.
  - ii. Click the Home icon to return to the main page.
- b.
  - i. Click Search and Modify Customer.
  - ii. Select Beisbol Si! in the drop-down list.
  - iii. Click the Execute Search button.
  - iv. Click the Update button.
  - v. Modify the customer record and click Submit.
  - vi. Click the Home icon to return to the main page.
- c.
  - i. Click Insert a new customer.
  - ii. Enter new customer record and click Submit.
  - iii. Click the Home icon to return to the main page.
- d.
  - i. Click the Frames version link.
  - ii. Click Full Customer Listing.

## Practice 2 Solutions

In this practice you create your application's home page in HTML.

1. Copy the `index.html` file that is located in the `labsPLSQL` directory to your home directory. Modify the file. You can be as creative as you want because the aim of this exercise is to ensure that you are familiar with HTML.
2. Use the anchor tag, `<A>`, to create a link to the 9iAS Administrative pages. See Practice 1, Question 4 for the URL.
3. Create a link on your home page to the finished application. See Practice 1, Question 6 for the URL.
4. Create a link on your home page to the application you will build during the course. See Practice 1, Question 6 for the URL.
5. Open your browser and test your application home page by issuing the following URL (see Practice 1, Question 5):

`http://hostname.domain:your_port_number/`

**Solution file `sol12.html` (for Questions 2–4; only the relevant code is shown):**

```
...
<P>
<!--Put in a link to iAS Admin pages here:-->
<A HREF="http://<machine>.<domain>:<Admin_port>">Click here
to go to the iAS Admin pages</A>
<P>
<!--Put in a link to summit.main_page here:-->
<A
HREF="http://<machine>.<domain>:<User_port>/pls/<dad>/summit.
main_page">Click here to go to the completed version of
the Summit Sporting Goods Customer Application</A>
<P>
<!--Put in a link to lab.main_page here:-->
<A HREF="http://<machine>.<domain>:<User_port>/pls/<dad>
/lab.main_page">Click here to go to my Summit Sporting Goods
Customer Application</A>
<HR>
<ADDRESS>
If you have questions, please contact:
<A HREF="mailto:your_name@your_company.com"> Webmaster </A>
</ADDRESS>
...
```

### Practice 3 Solutions

Any components you create from this point forward should form part of a PL/SQL package called LAB. For PL/SQL login information, refer to Practice 1, Question 1. Copy the template files, `spec.sql` and `body.sql`, from the `labsPLSQL` directory to your home directory and use them as a guide to the content of the LAB package.

1. Log into SQL\*Plus and compile `spec.sql`. This creates the package specification for LAB.
2. In `body.sql`, add code to the HEADER private procedure. This procedure accepts a single IN parameter (`v_title`) and creates the HTML tags that begin every page in your application. It should:
  - a. Open the HTML document.
  - b. Open the Head section.
  - c. Use `v_title` to provide a Title tag.
  - d. Close the Head section.
  - e. Open the Body section (with a suitable background such as `/image/owasbkg.jpg`).
  - f. Include the Internet Application Server main logo (`/image/ias_logo.gif`).
  - g. Draw a horizontal rule.

#### **Solution file `so13_2.sql`:**

**Note:** Only the code relevant to the question is shown here:

```
PROCEDURE header
  (v_title IN VARCHAR2)
IS
BEGIN
  http.HtmlOpen;
  http.HeadOpen;
  http.Title(v_title);
  http.HeadClose;
  http.BodyOpen(cbackground=>'/image/owasbkg.jpg');
  http.Center(htf.Img('/image/ias_logo.gif',
    calt=>'ias text logo'));
  http.hr;
END header;
```

### Practice 3 Solutions (continued)

3. Add code to the FOOTER private procedure. This procedure accepts no parameters, and creates HTML tags to end every page. It should:
  - a. Draw a horizontal rule.
  - b. Create a clickable image (/image/co\_home.gif), which has a link to the home page.
  - c. Add a link to allow users to mail comments to the page designer.
  - d. Close the Body section.
  - e. Close the HTML document.

#### Solution file sol3\_3.sql:

**Note:** The variables shown in the solution will be used in a later practice. Only the code relevant to the question is shown here:

```
PROCEDURE footer
IS
    v_path  varchar2(50);
    v_procedure_name varchar2(50);
BEGIN
    http.Hr;
    http.Center(htf.Anchor('lab.main_page',
        htf.Img('/image/co_home.gif')));
    http.Address('Comments? Contact ' ||
        htf.Anchor('mailto:xyz@my_org.com', 'WebMaster'));
    http.BodyClose;
    http.HtmlClose;
END footer;
```

### Practice 3 Solutions (continued)

4. Add code to the MAIN\_PAGE public procedure. This procedure accepts no parameters and is the application home page. You need to use the HEADER and FOOTER private procedures to start and end this procedure.

**Solution file sol3\_4.sql:**

**Note:** Only the code relevant to the question is shown.

```
PROCEDURE main_page
IS
    v_cookie varchar2(15);
BEGIN
    header('iAS Practice Application');
    .
    .
    .
    footer;
END main_page;
```

5. Log into SQL\*Plus and compile body.sql. This creates the package body for LAB.

**No formal solution for this practice.**

6. Test your application's main page by entering the appropriate URL. (See Practice 1, Question 6 for the URL.) Remember that only public procedures and functions can be invoked directly, so to test HEADER and FOOTER, just make sure that the relevant HTML is displayed on MAIN\_PAGE.

**No formal solution for this practice.**

7. Test the link from your application home page by issuing the following URL and clicking on the link to "My Summit Application":  
`http://hostname.domain:your_port_number/`

**No formal solution for this practice.**

### Practice 3 Solutions (continued)

8. Add code to the ERROR\_PAGE public procedure. This procedure is used by the application if an error condition arises. It should accept a parameter named v\_message to allow appropriate messages to be displayed. Use /image/error.gif to highlight that this is an error page.

#### Solution file sol3\_8.sql:

**Note:** Only the code relevant to the question is shown.

```
PROCEDURE error_page
    (v_message IN VARCHAR2) IS
BEGIN
    header('Application Error Page');
    http.Img('/image/error.gif');
    http.Print('An error has been raised by the application.' ||
    'If you are unable to determine the cause from the ' ||
    'information below then contact the help desk.');
```

http.Br;

http.Header(3,v\_message);

footer;

END error\_page;



### Practice 3 Solutions (continued)

9. Add code to the SUCCESS\_PAGE public procedure. Its purpose is to signify successful completion of an operation. It should accept a parameter named v\_message to allow appropriate messages to be displayed. Use /image/success.gif to highlight that this is a success page.

#### Solution file sol13\_9.sql:

**Note:** Only the code relevant to the question is shown.

```
PROCEDURE success_page
    (v_message IN VARCHAR2)
IS
BEGIN
    header('Action Completed Successfully');
    http.Image('/image/success.gif');
    http.Print('The action requested has been completed
    successfully');
    http.Br;
    http.Header(3,v_message);
    footer;
END success_page;
```

10. Test SUCCESS\_PAGE and ERROR\_PAGE by entering the appropriate URLs. Remember that the SUCCESS\_PAGE and ERROR\_PAGE procedures require a parameter, and you must tag this onto the end of the URL when testing (replace any spaces with +). Here is a sample call to test LAB.SUCCESS\_PAGE:  
`http://hostname.domain:port/virtual_path/lab.success_page?  
v_message=test+message`

**No formal solution for this practice.**

## Practice 4 Solutions

1. Add code to the SHOW\_ALL\_CUST public procedure to list all customers in the S\_CUSTOMER table sorted by customer name. This procedure accepts no parameters. You may choose to use a cursor and the HTML-table related tags in the HTP package or to use owa\_util. Display the customer's ID, name, country, credit rating and sales representative ID.

### Solution file sol4\_1.sql:

**Note:** Only the code relevant to the question is shown, the cursor version first and the owa\_util version second.

```
PROCEDURE show_all_cust
IS
    CURSOR cust_cur IS
        SELECT id, name, NVL(country, '-----') country,
               credit_rating, sales_rep_id
        FROM s_customer
        ORDER BY name;
BEGIN
    header('Customer List');
    http.CenterOpen;
    http.TableOpen('BORDER');
    http.TableCaption('Customer Listing sorted by Name');
    http.TableRowOpen;
    http.TableHeader('Id');
    http.TableHeader('Name');
    http.TableHeader('Country');
    http.TableHeader('Rating');
    http.TableHeader('RepId');
    http.TableRowClose;
    FOR cust_rec IN cust_cur LOOP
        http.TableRowOpen;
        http.TableData(cust_rec.id);
        http.TableData(cust_rec.name);
        http.TableData(cust_rec.country);
        http.TableData(cust_rec.credit_rating);
        http.TableData(cust_rec.sales_rep_id);
        http.TableRowClose;
    END LOOP;
    http.TableClose;
    http.CenterClose;
    footer;
END show_all_cust;
```

#### Practice 4 Solutions (continued)

```
PROCEDURE show_all_cust
IS
    x BOOLEAN;
BEGIN
    header('Customer List');
    http.CenterOpen;
    x := owa_util.tablePrint('s_customer', 'BORDER',
        owa_util.html_Table,
        'id, name, country, credit_rating, sales_rep_id',
        'ORDER BY name',
        'Id, Name, Country, Rating, RepId');
    http.CenterClose;
    footer;
END show_all_cust;
```

2. Test your procedure. Make sure that the corresponding link in LAB.MAIN\_PAGE, “Full customer listing”, works correctly.

**No formal solution for this practice.**

3. SHOW\_CUSTOMER public procedure to list information about a particular customer in the S\_CUSTOMER table. This procedure accepts one parameter, v\_id, the customer ID whose details you need to display. Copy the code from SHOW\_ALL\_CUST and make necessary modifications to it so the table looks like the following:

Id	Customer	Address	Telephone	Rating
209	Beisbol Sil	792 Playa Del Mar San Pedro de Macon's Costa Rica	809-352689	EXCELLENT

Test your procedure using:

[http://hostname.domain:port/pls/dad/lab.show\\_customer?v\\_id=209](http://hostname.domain:port/pls/dad/lab.show_customer?v_id=209)

**Note:** If necessary, replace the value 209 with some valid customer ID.

## Practice 4 Solutions (continued)

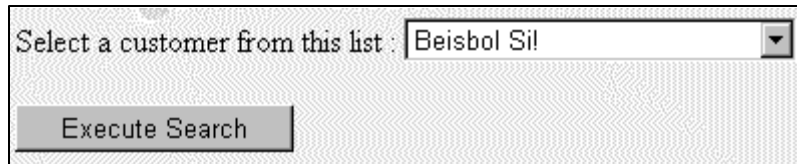
### Solution file sol14\_3.sql:

**Note:** Only the code relevant to the question is shown:

```
PROCEDURE show_customer
  (v_id in VARCHAR2)
IS
  CURSOR cust_cur IS
    SELECT id, name, address, city, state, country,
           phone, credit_rating
    FROM s_customer
    WHERE id = TO_NUMBER(v_id);
BEGIN
  header('Customer Listing');
  http.CenterOpen;
  http.Header(1,'Customer Details');
  http.Br;
  http.Br;
  http.TableOpen('BORDER=2',
                 cattributes=>'VALIGN="TOP"');
  http.TableRowOpen;
  http.TableHeader('Id');
  http.TableHeader('Customer');
  http.TableHeader('Address');
  http.TableHeader('Telephone');
  http.TableHeader('Rating');
  http.TableRowClose;
  FOR cust_rec IN cust_cur LOOP
    http.TableRowOpen;
    http.TableData(cust_rec.id);
    http.TableData(cust_rec.name);
    http.TableData(cust_rec.address || htf.br ||
                   cust_rec.city || ' ' || cust_rec.state || htf.br ||
                   cust_rec.country);
    http.TableData(cust_rec.phone);
    http.TableData(cust_rec.credit_rating);
    http.TableRowClose;
  END LOOP;
  http.TableClose;
  http.CenterClose;
  footer;
END show_customer;
```

## Practice 5 Solutions

1. Add code to the public procedure `FIND_CUSTOMER` to allow the user to select a customer from a drop-down (select) list. This procedure accepts no parameters. Display the customer's name in the list, but transmit the customer ID (`v_id`) to the action procedure of this HTML form. Call the `SHOW_CUSTOMER` procedure that you wrote in Practice 4, Question 3 as the action of this HTML form.

The screenshot shows a web form with a label "Select a customer from this list :". To the right of the label is a drop-down menu with "Beisbol Sil" selected. Below the label and the drop-down menu is a button labeled "Execute Search".

### Solution file `sol5_1.sql`:

**Note:** Only the code relevant to the question is shown:

```
PROCEDURE find_customer
IS
    CURSOR cust_cur IS
        SELECT id, name
        FROM s_customer
        ORDER BY name;
BEGIN
    header('Customer Search');
    http.Header(1,'Search for a Customer','CENTER');
    http.Br;
    http.FormOpen('lab.show_customer');
    http.FormSelectOpen('v_id',
        'Select a customer from this list : ');
    FOR cust_rec IN cust_cur LOOP
        http.FormSelectOption(cust_rec.name,
            cattributes=>'VALUE="||cust_rec.id||"'');
    END LOOP;
    http.FormSelectClose;
    http.Br;
    http.Br;
    http.FormSubmit(cvalue=>'Execute Search');
    http.FormClose;
    footer;
END find_customer;
```

### Practice 5 Solutions (continued)

2. Test your procedures. Make sure that the corresponding link, “Search and modify customer,” in LAB.MAIN\_PAGE works correctly.

#### No formal solution for this practice.

3. Modify SHOW\_CUSTOMER to add an HTML form that displays Update and Delete buttons after displaying customer information. The form has no other visual input items besides the two buttons. Set the action of the form to call LAB.MODIFY\_CUSTOMER with two parameters: the ID of the customer whose record is displayed on this screen (use a hidden field) and the value of the button selected by the user.

#### Solution file `sol5_3.sql`:

**Note:** Only the code relevant to the question is shown:

(from LAB.SHOW\_CUSTOMER)

```
...
    http.TableClose;
    http.Br;
    http.Br;
    http.Italic('Select one of the buttons below to Update ' ||
        ' or Delete this customer's information');
    http.Br;
    http.Br;
    http.FormOpen('lab.modify_customer');
    http.FormHidden('v_id',v_id);
    http.FormSubmit('v_action','Update');
    http.FormSubmit('v_action','Delete');
    http.CenterClose;
    footer;
END show_customer;
```

## Practice 5 Solutions (continued)

4. Examine the code of LAB.MODIFY\_CUSTOMER. Note that the main functionality of this procedure is to generate HTML forms for inserting, updating, and deleting customers.
  - a. If the action passed to the form is 'Delete', then this procedure generates an HTML form that prompts the user to confirm the delete.
  - b. If the action passed to the form is 'Update', then this procedure generates an HTML form that displays the customer information so the user can modify it.
  - c. If the action passed to the form is 'Insert', then this procedure presents a blank HTML form so the user can add a new customer.

**No formal solution for this practice.**

5. Test your procedures. Invoke "Search and modify customer" from LAB.MAIN\_PAGE and ensure that your buttons appear correctly.

**No formal solution for this practice.**

6. Add the following functionality to the LAB.MODIFY\_CUSTOMER procedure:
  - a. Convert the sales\_rep\_id and region\_id text fields into select lists.
  - b. Convert the credit\_rating text field into a radio group.

**Hint:** Remember that when updating data, this form needs to display the current values initially. Select lists normally display the first item in the list. Therefore you need to ensure that the SELECTED attribute is set for the current value. Similarly, for the radio group, ensure that the CHECKED value is set for the current value.

**Solution file sol5\_6.sql:**

**Note:** Only the relevant code is shown here (from LAB.MODIFY\_CUSTOMER):

```
...
http.TableRowOpen;
http.TableData('Credit Rating');
http.print ('<TD>');
IF current_customer.credit_rating = 'EXCELLENT' THEN
    http.FormRadio('v_credit_rating', 'EXCELLENT', 'CHECKED');
ELSE
    http.FormRadio('v_credit_rating', 'EXCELLENT');
END IF;
http.print('Excellent');
IF current_customer.credit_rating = 'GOOD' THEN
    http.FormRadio('v_credit_rating', 'GOOD', 'CHECKED');
ELSE
    http.FormRadio('v_credit_rating', 'GOOD');
END IF;
http.print('Good');
```

### Practice 5 Solutions (continued)

```
IF current_customer.credit_rating = 'POOR' THEN
    http.FormRadio('v_credit_rating', 'POOR', 'CHECKED');
ELSE
    http.FormRadio('v_credit_rating', 'POOR');
END IF;
http.print('Poor');
IF current_customer.credit_rating IS NULL THEN
    http.FormRadio('v_credit_rating', 'Unknown', 'CHECKED');
ELSE
    http.FormRadio('v_credit_rating', 'Unknown');
END IF;
http.print('Unknown');
http.print('</TD>');
http.TableRowOpen;
http.TableData('Sales Rep');
http.print('<TD>');
http.FormSelectOpen('v_sales_rep_id');
FOR rep_rec IN rep_cur LOOP
    IF rep_rec.id = current_customer.sales_rep_id THEN
        http.FormSelectOption(rep_rec.sales, 'SELECTED',
            cattributes=>'VALUE="' || rep_rec.id || '"');
    ELSE
        http.FormSelectOption(rep_rec.sales,
            cattributes=>'VALUE="' || rep_rec.id || '"');
    END IF;
END LOOP;
http.FormSelectClose;
http.print('</TD>');
http.TableRowClose;
http.TableRowOpen;
http.TableData('Region');
http.print('<TD>');
http.FormSelectOpen('v_region_id');
FOR region_rec IN region_cur LOOP
    IF region_rec.id = current_customer.region_id THEN
        http.FormSelectOption(region_rec.name, 'SELECTED',
            cattributes=>'VALUE="' || region_rec.id || '"');
    ELSE
        http.FormSelectOption(region_rec.name,
            cattributes=>'VALUE="' || region_rec.id || '"');
    END IF;
END LOOP;
```



## **Practice 5 Solutions (continued)**

7. Test LAB.MODIFY\_CUSTOMER.
  - a. Invoke the procedure from the “Insert new customer” link in LAB.MAIN\_PAGE.
  - b. Invoke the procedure from the Update and Delete buttons on the LAB.SHOW\_CUSTOMER form.

**No formal solution for this practice.**

## Practice 6 Solutions

1. Modify the FOOTER procedure to include a reusable signature that allows the user to display the PL/SQL source code for the current page.

### Solution file `sol6_1.sql`:

**Note:** Only the code relevant to the question is shown here:

```
PROCEDURE footer IS
    v_path  varchar2(50);
    v_procedure_name varchar2(50);
BEGIN
    http.Hr;
    http.Center(htf.Anchor('lab.main_page',
        htf.Image('/image/co_home.gif')));
    http.Address('Comments? Contact ' ||
        htf.Anchor('mailto:xyz@my_org.com', 'WebMaster'));
    v_path := owa_util.Get_Cgi_Env('PATH_INFO');
    v_procedure_name := substr(v_path,2);
    http.anchor('owa_util.showsource?cname=' ||
        v_procedure_name, 'View PL/SQL source code');
    http.BodyClose;
    http.HtmlClose;
END footer;
```

2. Add code to the call\_http public procedure to display the contents of the simple.html file, which is in your html directory. Use owa\_util.get\_cgi\_env to dynamically construct the URL for the call. When finished, test the procedure by using the link on the LAB.MAIN\_PAGE.

### Solution file `sol6_2.sql`:

**Note:** Only the code relevant to the question is shown here:

```
PROCEDURE call_http IS
    dummy  varchar2(2000);
    urlval varchar2(100):='http://';
BEGIN
    header('Application Call Example');
    http.header(1,'This is an example of calling an HTTP' ||
        ' application from PL/SQL');
    urlval:=urlval||owa_util.Get_Cgi_Env('SERVER_NAME')||': ' ||
        owa_util.Get_Cgi_Env('SERVER_PORT');
    urlval:=urlval||'/index.html';
    dummy := utl_http.Request(urlval);
    http.Print(dummy);
    footer;
END call_http;
```

## Practice 7 Solutions

The purpose of this practice is to use cookies to count the number of times a particular individual has visited LAB.MAIN\_PAGE. Note that you are counting the number of times a particular individual has visited this page and not the total number of hits for the page.

1. Create a private function GET\_COOKIE that accepts one parameter (v\_cookie\_name) and returns the value of the cookie, v\_cookie\_name, if it has a valid value. If the cookie does not have a valid value (has not been set), the function should return the value zero.

### Solution file sol17\_1.sql:

**Note:** Only the code relevant to the question is shown here:

```
FUNCTION get_cookie
(v_cookie_name IN VARCHAR2)
RETURN VARCHAR2
IS
    v_cookie_val owa_cookie.Cookie;
BEGIN
    v_cookie_val := owa_cookie.Get(v_cookie_name);
    IF v_cookie_val.Num_Vals > 0 THEN
        RETURN(v_cookie_val.Vals(1));
    ELSE
        RETURN('zero');
    END IF;
END get_cookie;
```

2. Create a private procedure SET\_COOKIE that accepts two parameters (v\_cookie\_name and v\_cookie\_val). The v\_cookie\_name parameter is the name of the cookie that is being created. The v\_cookie\_val parameter is the cookie's value. Make sure that the cookie expires once the user exits the browser.

### Solution file sol17\_2.sql:

**Note:** Only the code relevant to the question is shown here:

```
PROCEDURE set_cookie
(v_cookie_name IN VARCHAR2,
v_cookie_val IN VARCHAR2)
IS
BEGIN
    owa_util.Mime_Header('text/html',FALSE);
    owa_cookie.Send(v_cookie_name,v_cookie_val);
    owa_util.Http_Header_Close;
END set_cookie;
```

### Practice 7 Solutions (continued)

3. Modify procedure MAIN\_PAGE so that it sets a cookie value and displays the number of times an individual has visited the page using SET\_COOKIE and GET\_COOKIE. Test your procedure by reloading LAB.MAIN\_PAGE several times and verify that the cookie is incremented.

**Solution file sol17\_3.sql:**

**Note:** Only the code relevant to the question is shown here:

```
PROCEDURE main_page
IS
  v_cookie varchar2(15);
BEGIN
  v_cookie := get_cookie('SUMMITCOUNT');
  IF v_cookie = 'zero' THEN
    set_cookie('SUMMITCOUNT',1);
  ELSE
    set_cookie('SUMMITCOUNT',to_number(v_cookie)+1);
  END IF;
  ...
```

**Note:** The rest of the code is the same as sol13\_3.sql

## Practice 8 Solutions

1. Modify the MODIFY\_CUSTOMER procedure to compute a checksum for the Update and Delete options and pass them as hidden fields to UPDATE\_CUSTOMER and DELETE\_CUSTOMER. Note that the Insert option does not require a checksum.

**Solution file sol8\_1.sql:**

**Note:** Only the code relevant to the question is shown here:

Insert one line of code for the Delete option in MODIFY\_CUSTOMER as highlighted here:

```
...  
  
    http.FormOpen('lab.delete_customer');  
    http.FormHidden('v_id',v_id);  
    http.FormHidden('v_rowid',current_rowid);  
    http.FormHidden('v_checksum',  
        owa_opt_lock.checksum(user,'s_customer',  
        current_rowid));  
    http.FormSubmit('v_action','Delete');  
    http.FormSubmit('v_action','Do not delete!');  
    http.FormClose;
```

...

Insert another line of code for the Update option in MODIFY\_CUSTOMER as highlighted here:

```
...  
  
    http.FormOpen('lab.update_customer');  
    http.Header(1,'Modify Customer Details','CENTER');  
    http.FormHidden('v_rowid',current_rowid);  
    http.FormHidden('v_checksum',  
        owa_opt_lock.checksum(user,'s_customer',  
        current_rowid));  
    END IF;
```

...

## Practice 8 Solutions (continued)

2. Modify the LAB.UPDATE\_CUSTOMER procedure to first compute a checksum to verify that the update should be allowed. Use the ERROR\_PAGE or SUCCESS\_PAGE procedure, with appropriate messages, to indicate the success or failure of the update.

### Solution file sol8\_2.sql:

**Note:** Only the code relevant to the question is shown here:

```
PROCEDURE update_customer
  (v_id          IN  VARCHAR2,
   ...
   v_comments    IN  VARCHAR2  DEFAULT NULL)
IS
  v_message  VARCHAR2(150);
BEGIN
  IF TO_NUMBER(v_checksum)=
    owa_opt_lock.Checksum(user,'s_customer',v_rowid) THEN
    UPDATE s_customer
      ...
      COMMIT;
    v_message := 'Customer record '||v_id||
      ' has been updated';
    success_page(v_message);
  ELSE
    v_message := 'Customer record '||v_id|| 'has changed '||
      'while you were modifying the record. <P> Your ' ||
      'modifications are not allowed - Please re-query';
    error_page(v_message);
  END IF;
  ...
```

### Practice 8 Solutions (continued)

3. Modify the LAB.DELETE\_CUSTOMER procedure to first compute a checksum to verify that the delete should be allowed. Use the ERROR\_PAGE or SUCCESS\_PAGE procedure, with appropriate messages, to indicate the success or failure of the update.

#### Solution file sol18\_3.sql:

**Note:** Only the code relevant to the question is shown here:

```
PROCEDURE delete_customer
  (v_id          IN  VARCHAR2,
   v_rowid       IN  VARCHAR2,
   v_checksum    IN  VARCHAR2  DEFAULT NULL,
   v_action      IN  VARCHAR2  DEFAULT NULL)
IS
  v_message varchar2(200);
BEGIN
  ...
  IF TO_NUMBER(v_checksum) =
    owa_opt_lock.Checksum(user,'s_customer',v_rowid) THEN
    DELETE FROM s_customer WHERE rowid = v_rowid;
    COMMIT;
    success_page ('Customer ' || v_id || ' successfully deleted');
  ELSE
    v_message := 'Customer record ' || v_id ||
      'has changed between the time you queried the record ' ||
      'and deleted it. <P> Your delete is not allowed - ' ||
      'Please re-query';
    error_page(v_message);
  END IF;
  ...
```

## Practice 9 Solutions

1. Add code to the public procedure named LAB.FRAME\_VERSION that displays the application main page using frames. This procedure accepts no parameters. It subdivides the content area into three framesets. The first frameset creates two columns, where the first column occupies 25% of the area. The second frameset creates two rows, where the second row occupies 25% of the area.

Name the large frame MAIN\_FRAME. It should use the LAB.FRAME\_INTRODUCTION procedure as its source. The side frame should contain the LAB.FRAME\_TOC procedure, and the bottom frame should contain the LAB.FRAME\_FOOTER procedure.

**Note:** Be sure to code a no-frames alternative!

### Solution file `so19_1.sql`:

**Note:** Only the code relevant to the question is shown here:

```
PROCEDURE frame_version
IS
BEGIN
    http.HtmlOpen;
    http.HeadOpen;
    http.Title('Frames version of Application');
    http.HeadClose;
    http.FrameSetOpen(ccols=>'25%,*');
        http.FrameSetOpen(crows=>'*,25%');
            http.Frame('lab.frame_toc');
            http.Frame('lab.frame_footer');
        http.FrameSetClose;
        http.Frame('lab.frame_introduction','main_frame');
    http.FrameSetClose;
    http.NoFramesOpen;
        http.BodyOpen;
        http.header(2,'Your browser doesn't support HTML Frames');
        http.anchor('lab.main_page',
            'Please click here to return to the main page');
        http.BodyClose;
    http.NoFramesClose;
    http.HtmlClose;
END frame_version;
```



## Practice 9 Solutions (continued)

2. Add code to the public procedure named LAB.FRAME\_TOC. This procedure is based on the anchors in MAIN\_PAGE (except the last one, which is used to switch to the frames version of the application), but should use the HTP.BASE procedure to set the default TARGET of the document to MAIN\_FRAME.

### Solution file `sol9_2.sql`:

**Note:** Only the code relevant to the question is shown here:

```
PROCEDURE frame_toc
IS
BEGIN
    http.HtmlOpen;
    http.BodyOpen(cattributes=>'bgcolor="Silver"');
    http.Base('main_frame');
    http.Header(2, 'Menu', 'CENTER');
    http.Img('/image/blue_tri.gif');
    http.Anchor('lab.show_all_cust',
        'Full customer listing');
    http.Br;
    http.Img('/image/purple_tri.gif');
    http.Anchor('lab.find_customer',
        'Search and modify customer');
    http.Br;
    http.Img('/image/yellow_tri.gif');
    http.Anchor('lab.modify_customer?v_action=Insert',
        'Insert new customer');
    http.Br;
    http.Img('/image/red_tri.gif');
    http.Anchor('lab.icx', 'PL/SQL to LiveHTML ICX');
    http.Br;
    http.Img('/image/green_tri.gif');
    http.Anchor('demo.bookmark', 'Demonstrations page');
    http.BodyClose;
    http.HtmlClose;
END frame_toc;
```

3. When finished, test the procedure by using the link on LAB.MAIN\_PAGE.

**No formal solution for this practice.**



---

# B

---

## **PL/SQL Web Toolkit Reference**

## Overview

This appendix describes the PL/SQL packages that make up the PL/SQL Web Toolkit. The name of each package is a mnemonic for its contents:

- Hypertext procedures and functions (HTP and HTF packages)
- OWA utilities (OWA\_UTIL package)
- Locking utilities (OWA\_OPT\_LOCK package)
- Pattern matching utilities (OWA\_PATTERN package)
- Text manipulation utilities (OWA\_TEXT package)
- Image map utilities (OWA\_IMAGE package)
- Cookie utilities (OWA\_COOKIE package)

## Hypertext Procedures (the HTP Package)

A hypertext procedure generates a line in an HTML document that contains the HTML tag that corresponds to its name. The HTP procedures are grouped into the following categories:

- Print Procedures
- Head Related Tags
- General Body Tags
- List Tags
- Character Format Tags
- Physical Format Tags
- Form Tags
- Table Tags

## Hypertext Functions (the HTF Package)

A hypertext function returns the HTML tag that corresponds to its name but is not passed to the browser. The output of an HTF function must be passed to `http.print` in order to be output to an HTML document. Every hypertext function (HTF) has a corresponding hypertext procedure (HTP) with the same name and parameters. HTF functions are generally used only when you need to nest calls.

**Note:** To look up an HTF function, see the entry for the corresponding HTP procedures.

## HTP Print Procedures

The following print procedures are used in conjunction with HTF functions to generate a line in the HTML document being constructed. They can also be passed hard-coded text that will appear in the HTML document as-is. The generated line is passed to the PL/SQL Agent, which sends it to standard output. As documented in the CGI 1.1 specification, the Oracle Web Listener takes the contents of standard output and returns it to the Web browser that requested the dynamic HTML document.

### **http.print Syntax**

```
http.print(cbuf | dbuf | nbuf); -- generates a line containing the value of the parameter  
http.prn (cbuf | dbuf | nbuf);
```

### **http.prints Syntax**

```
http.prints (ctext);  
http.ps (ctext);
```

Generates a line and replaces all occurrences of the following special characters with the escape characters shown. If not replaced, the special characters would be interpreted as HTML control characters, and would produce garbled output.

- < is replaced with `&lt;`;
- > is replaced with `&gt;`;
- " is replaced with `&quot;`;
- & is replaced with `&amp;`;

## HTP Head Procedures

The following procedure tags should be placed between the `http.headOpen` and `http.headClose` procedure tags.

### **http.title Syntax**

Most Web Browsers display the `ctitle` value at the top of the document viewing window.

```
http.title (ctitle);          -- generates <TITLE>ctitle</TITLE>
```

Parameters: `ctitle` in `varchar2`

### **http.base Syntax**

```
http.base (ctarget, cattributes); -- generates absolute pathname of current document
```

Parameters: `ctarget` in `varchar2` DEFAULT NULL  
            `cattributes` in `varchar2` DEFAULT NULL

### **http.isindex Syntax**

```
http.isindex (cprompt, curl);  
                -- generates <ISINDEX PROMPT="cprompt" HREF="curl">
```

Parameters: `cprompt` in `varchar2` DEFAULT NULL  
            `curl` in `varchar2` DEFAULT NULL

### **http.linkRel Syntax**

```
http.linkRel (crel, curl, ctitle);  
                -- generates <LINK REL="crel" HREF="curl" TITLE="ctitle">
```

Parameters: `crel` in `varchar2`  
            `curl` in `varchar2`  
            `ctitle` in `varchar2` DEFAULT NULL

### **http.linkRev Syntax**

```
http.linkRev (crel, curl, ctitle);  
                -- generates <LINK REV="crev" HREF="curl" TITLE="ctitle">
```

Parameters: `crev` in `varchar2`  
            `curl` in `varchar2`  
            `ctitle` in `varchar2` DEFAULT NULL

### **http.meta Syntax**

```
http.meta (chttp_equiv, cname, ccontent);  
        -- generates <META HTTP-EQUIV="chttp_equiv" NAME ="cname" CONTENT="ccontent">
```

Parameters: `chttp_equiv` in `varchar2`  
            `cname` in `varchar2`  
            `ccontent` in `varchar2`

Example: `http.meta ('Refresh', NULL, 120); <META HTTP-EQUIV="Refresh" CONTENT=120>`

On some browsers, this causes the current URL to be reloaded automatically every 120 seconds.

## HTP Body Procedures

Body tags are used in the main text of your HTML page. They can format a paragraph, allow you to add hidden Comments to your text, and add images within the body of your HTML text.

### http.line Syntax

```
http.line (cclear, csrc, cattributes);  
http.hr          -- generates <HR CLEAR="cclear" SRC="csrc" cattributes>
```

Parameters: cclear in varchar2 DEFAULT NULL  
              csrc in varchar2 DEFAULT NULL  
              cattributes in varchar2 DEFAULT NULL

### http.nl Syntax

```
http.nl (cclear, cattributes);  
http.br          -- generates <BR CLEAR="cclear" cattributes>
```

Parameters: cclear in varchar2 DEFAULT NULL,  
              cattributes in varchar2 DEFAULT NULL

### http.header Syntax

```
http.header (nsize, cheader, calign, cnowrap, cclear, cattributes);  
          -- generates <Hnsize ALIGN="calign" NOWRAP CLEAR="cclear"  
                      cattributes>cheader</Hnsize>
```

Parameters: nsize in integer                      Valid values are 1 through 6  
              cheader in varchar2  
              calign in varchar2 DEFAULT NULL  
              cnowrap in varchar2 DEFAULT NULL  
              cclear in varchar2 DEFAULT NULL  
              cattributes in varchar2 DEFAULT NULL

Example: http.header (1,'Overview'); produces <H1>Overview</H1>

### http.anchor Syntax

```
http.anchor (curl, ctext, cname, cattributes);  
          -- generates <A HREF="curl" NAME="cname" cattributes>ctext</A>
```

Parameters: curl in varchar2  
              ctext in varchar2  
              cname in varchar2 DEFAULT NULL  
              cattributes in varchar2 DEFAULT NULL

## HTP Body Procedures (continued)

### http.anchor2 Syntax

```
http.anchor2 (curl, ctext, cname, ctarget, cattributes);  
    -- generates <A HREF="curl" NAME="cname" TARGET = "ctarget"  
                                cattributes>ctext</A>
```

Parameters: curl in varchar2  
            ctext in varchar2  
            cname in varchar2 DEFAULT NULL  
            ctarget in varchar2 DEFAULT NULL  
            cattributes in varchar2 DEFAULT NULL

### http.mailto Syntax

```
http.mailto (caddress, ctext, cname, cattributes);  
    -- generates <A HREF="mailto:caddress" cattributes>ctext</A>
```

Parameters: caddress in varchar2  
            ctext in varchar2  
            cname in varchar2  
            cattributes in varchar2 DEFAULT NULL

Example: http.mailto('pres@white\_house.gov','Email the President');  
          <A HREF="mailto:pres@white\_house.gov">Email the President</A>

### http.img Syntax

Places an image into the HTML page

```
http.img (curl, calign, calt, cismap, cattributes);  
    -- generates <IMG SRC="curl" ALIGN="calign" ALT="calt" ISMAP cattributes>
```

Parameters: curl in varchar2 DEFAULT NULL  
            calign in varchar2 DEFAULT NULL  
            calt in varchar2 DEFAULT NULL  
            cismap in varchar2 DEFAULT NULL  
            cattributes in varchar2 DEFAULT NULL

### http.img2 Syntax

Places an image into the HTML page that is used as an image map

```
http.img (curl, calign, calt, cismap, cusemap, cattributes);  
    -- generates <IMG SRC="curl" ALIGN="calign" ALT="calt" ISMAP  
USEMAP="cusemap" cattributes>
```

Parameters: curl in varchar2 DEFAULT NULL  
            calign in varchar2 DEFAULT NULL  
            calt in varchar2 DEFAULT NULL  
            cismap in varchar2 DEFAULT NULL  
            cusemap in varchar2 DEFAULT NULL  
            cattributes in varchar2 DEFAULT NULL



## HTP Body Procedures (continued)

### http.para Syntax

http.para; -- generates <P>

### http.paragraph Syntax

http.paragraph (calign, cnowrap, cclear, cattributes);  
-- generates <P ALIGN="*calign*" NOWRAP CLEAR="*cclear*" *cattributes*>

Parameters: calign in varchar2 DEFAULT NULL  
cnowrap in varchar2 DEFAULT NULL  
cclear in varchar2 DEFAULT NULL  
cattributes in varchar2 DEFAULT NULL

### http.address Syntax

http.address (cvalue, cnowrap, cclear, cattributes);  
-- generates <ADDRESS CLEAR="*cclear*" NOWRAP *cattributes*>cvalue</ADDRESS>

Parameters: cvalue in varchar2  
cnowrap in varchar2 DEFAULT NULL  
cclear in varchar2 DEFAULT NULL  
cattributes in varchar2 DEFAULT NULL

### http.Comment Syntax

http.Comment (ctext); -- generates <!-- *ctext* -->

Parameters: ctext in varchar2

### http.preOpen Syntax

http.preOpen (cclear, cwidth, cattributes);  
-- generates <PRE CLEAR="*cclear*" WIDTH="*cwidth*" *cattributes*>

Parameters: cclear in varchar2 DEFAULT NULL  
cwidth in varchar2 DEFAULT NULL  
cattributes in varchar2 DEFAULT NULL

### http.preClose Syntax

http.preClose; -- generates </PRE>

### http.blockquoteOpen Syntax

http.blockquoteOpen (cnowrap, cclear, cattributes);  
-- generates <BLOCKQUOTE CLEAR="*cclear*" NOWRAP *cattributes*>

Parameters: cnowrap in varchar2 DEFAULT NULL  
cclear in varchar2 DEFAULT NULL  
cattributes in varchar2 DEFAULT NULL;

### http.blockquoteClose Syntax

http.blockquoteClose; -- generates </BLOCKQUOTE>

## HTP Body Procedures (continued)

### http.base Syntax

http.base(ctarget, cattributes);

-- generates <BASE HREF="<current URL>" TARGET="target" cattributes>

Parameters: ctarget in varchar2 DEFAULT NULL  
cattributes in varchar2 DEFAULT NULL

### http.area Syntax

http.area(ccoords, cshape, chref, cnohref, ctarget, cattributes);

-- generates <AREA COORDS="ccoords" SHAPE="cshape" HREF="chref" NOHREF  
TARGET="ctarget" cattributes>

Parameters: ccoords in varchar2  
cshape in varchar2 DEFAULT NULL  
chref in varchar2 DEFAULT NULL  
cnohref in varchar2 DEFAULT NULL  
ctarget in varchar2 DEFAULT NULL  
cattributes in varchar2 DEFAULT NULL

### http.mapOpen Syntax

http.mapOpen(cname, cattributes);

-- generates <MAP NAME="cname" cattributes>

Parameters: cname in varchar2  
cattributes in varchar2 DEFAULT NULL

### http.mapClose Syntax

http.mapClose; -- generates </MAP>

### http.bgsound Syntax

http.bgsound(csrc, cloop, cattributes);

-- generates <BGSOUND SRC="csrc" LOOP="cloop" cattributes>

Parameters: csrc in varchar2  
cloop in varchar2 DEFAULT NULL  
cattributes in varchar2 DEFAULT NULL

### http.div Syntax

http.div(calign, cattributes);

-- generates <DIV ALIGN="calign" cattributes>

Parameters: calign in varchar2 DEFAULT NULL  
cattributes in varchar2 DEFAULT NULL

### http.listingOpen Syntax

http.listingOpen; -- generates <LISTING>

### http.listingClose Syntax

http.listingClose; -- generates </LISTING>

## HTP Body Procedures (continued)

### http.nobr Syntax

http.nobr(ctext); -- generates <NOBR>ctext</NOBR>

Parameters: ctext in varchar2

### http.wbr Syntax

http.wbr; -- generates <WBR>

### http.center Syntax

http.center(ctext); -- generates <CENTER>ctext</CENTER>

Parameters: ctext in varchar2

### http.centerOpen Syntax

http.centerOpen; -- generates <CENTER>

### http.centerClose Syntax

http.centerClose; -- generates </CENTER>

### http.dfn Syntax

http.dfn(ctext); -- generates <DFN>ctext</DFN>

Parameters: ctext in varchar2

### http.big Syntax

http.big(ctext, cattributes);  
-- generates <BIG cattributes>ctext</BIG>

Parameters: ctext in varchar2  
cattributes in varchar2 DEFAULT NULL

### http.small Syntax

http.small(ctext, cattributes);  
-- generates <SMALL cattributes>ctext</SMALL>

Parameters: ctext in varchar2  
cattributes in varchar2 DEFAULT NULL

### http.sub Syntax

http.sub(ctext, calign, cattributes);  
-- generates <SUB ALIGN="calign" cattributes>ctext</SUB>

Parameters: ctext in varchar2  
calign in varchar2 DEFAULT NULL  
cattributes in varchar2 DEFAULT NULL

### http.sup Syntax

http.sup(ctext, calign, cattributes);  
-- generates <SUP ALIGN="calign" cattributes>ctext</SUP>

Parameters: ctext in varchar2  
calign in varchar2  
cattributes in varchar2 DEFAULT NULL

## HTP Body Procedures (continued)

### http.basefont Syntax

```
http.basefont(nsize); -- generates <BASEFONT SIZE="nsize">
```

Parameters: nsize in integer

### http.fontOpen Syntax

```
http.fontOpen(ccolor, cface, csize, cattributes);  
-- generates <FONT COLOR="ccolor" FACE="cface" SIZE="csize" cattributes>
```

Parameters: ccolor in varchar2 DEFAULT NULL  
            cface in varchar2 DEFAULT NULL  
            csize in varchar2 DEFAULT NULL  
            cattributes in varchar2 DEFAULT NULL

### http.fontClose Syntax

```
http.fontClose; -- generates </FONT>
```

### http.plaintext Syntax

```
http.plaintext(ctext, cattributes);  
-- generates <PLAINTEXT cattributes>ctext</PLAINTEXT>
```

Parameters: ctext in varchar2  
            cattributes in varchar2 DEFAULT NULL

### http.s and http.strike Syntax

Specifies that the text is rendered in strikethrough type.

```
http.s(ctext, cattributes); -- generates <S cattributes>ctext</S>  
http.strike(ctext, cattributes);  
-- generates <STRIKE cattributes>ctext</STRIKE>
```

Parameters: ctext in varchar2  
            cattributes in varchar2 DEFAULT NULL

## HTP List Procedures

List tags allow you to display information in the following ways:

- *ordered*: these lists have numbered items
- *unordered*: these lists have bullets to mark each item
- *definition*: these lists alternate a term with its definition

### http.listHeader Syntax

```
http.listHeader (ctext, cattributes);  
-- generates <LH cattributes>ctext</LH>
```

Parameters: ctext in varchar2  
            cattributes in varchar2 DEFAULT NULL

## HTP List Procedures (continued)

### http.listItem Syntax

```
http.listItem (ctext, cclear, cdingbat, csrc, cattributes);  
    -- generates <LI CLEAR="cclear" DINGBAT="cdingbat" SRC="csrc"  
    cattributes>ctext
```

Parameters: ctext in varchar2 DEFAULT NULL  
cclear in varchar2 DEFAULT NULL  
cdingbat in varchar2 DEFAULT NULL  
csrc in varchar2 DEFAULT NULL  
cattributes in varchar2 DEFAULT NULL

### http.ulistOpen Syntax

```
http.ulistOpen (cclear, cwrap, cdingbat, csrc, cattributes);  
    -- generates <UL CLEAR="cclear" WRAP="cwrap" DINGBAT="cdingbat"  
    SRC="csrc" cattributes>
```

Parameters: cclear in varchar2 DEFAULT NULL  
cwrap in varchar2 DEFAULT NULL  
cdingbat in varchar2 DEFAULT NULL  
csrc in varchar2 DEFAULT NULL  
cattributes in varchar2 DEFAULT NULL

### http.ulistClose Syntax

```
http.ulistClose;    -- generates </UL>
```

### http.olistOpen Syntax

```
http.olistOpen (cclear, cwrap, cattributes);  
    -- generates <OL CLEAR="cclear" WRAP="cwrap" cattributes>
```

Parameters: cclear in varchar2 DEFAULT NULL  
cwrap in varchar2 DEFAULT NULL  
cattributes in varchar2 DEFAULT NULL

### http.olistClose Syntax

```
http.olistClose;    -- generates </OL>
```

### http.dlistOpen Syntax

```
http.dlistOpen (cclear, cattributes);  
    -- generates <DL CLEAR="cclear" cattributes>
```

Parameters: cclear in varchar2 DEFAULT NULL  
cattributes in varchar2 DEFAULT NULL

### http.dlistClose Syntax

```
http.dlistClose    -- generates </DL>
```

## HTP List Procedures (continued)

### http.dlistDef Syntax

```
http.dlistDef (ctext, cclear, cattributes);  
-- generates <DD CLEAR="cclear" cattributes>ctext
```

Parameters: c<sub>text</sub> in varchar2 DEFAULT NULL  
              c<sub>clear</sub> in varchar2 DEFAULT NULL  
              c<sub>attributes</sub> in varchar2 DEFAULT NULL

### http.dlistTerm Syntax

```
http.dlistTerm (ctext, cclear, cattributes);  
-- generates <DT CLEAR="cclear" cattributes>ctext
```

Parameters: c<sub>text</sub> in varchar2 DEFAULT NULL  
              c<sub>clear</sub> in varchar2 DEFAULT NULL  
              c<sub>attributes</sub> in varchar2 DEFAULT NULL

### http.menulistOpen Syntax

```
http.menulistOpen; -- generates <MENU>
```

### http.menulistClose Syntax

```
http.menulistClose; -- generates </MENU>
```

### http.dirlistOpen Syntax

```
http.dirlistOpen; -- generates <DIR>
```

### http.dirlistClose Syntax

```
http.dirlistClose; -- generates </DIR>
```

## HTP Character Format Procedures

The character format tags are used to specify or alter the appearance of the marked text. Character format tags have opening and closing elements, and affect only the text that they surround.

Character format tags tell the browser how a character or character string should appear, but each browser determines its actual appearance. Essentially, they place text into categories such that all text in a given category is given the same special treatment, but the browser determines what that treatment is. For example, the HTML string <STRONG>Here is some text</STRONG> might appear as bold in some browsers, or might flash instead. If a specific text attribute, such as bold is desired, a physical format tag may be necessary. See the section, "Physical Format Tags," for more information.

### http.cite Syntax

```
http.cite (ctext, cattributes); -- generates <CITE cattributes>ctext</CITE>
```

Parameters: c<sub>text</sub> in varchar2  
              c<sub>attributes</sub> in varchar2 DEFAULT NULL

### http.code Syntax

```
http.code (ctext, cattributes); -- generates <CODE cattributes>ctext</CODE>
```

Parameters: c<sub>text</sub> in varchar2  
              c<sub>attributes</sub> in varchar2 DEFAULT NULL

## HTP Character Format Procedures (continued)

### http.emphasis Syntax

```
http.emphasis (ctext, cattributes); -- generates <EM cattributes>ctext</EM>  
http.em (ctext, cattributes);
```

Parameters: ctext in varchar2  
              cattributes in varchar2 DEFAULT NULL

### http.keyboard Syntax

```
http.keyboard (ctext, cattributes);  
http.kbd (ctext, cattributes); -- generates <KBD cattributes>ctext</KBD>
```

Parameters: ctext in varchar2  
              cattributes in varchar2 DEFAULT NULL

### http.sample Syntax

```
http.sample (ctext, cattributes);  
-- generates <SAMP cattributes>ctext</SAMP>
```

Parameters: ctext in varchar2  
              cattributes in varchar2 DEFAULT NULL

### http.strong Syntax

```
http.strong (ctext, cattributes);  
-- generates <STRONG cattributes>ctext</STRONG>
```

Parameters: ctext in varchar2  
              cattributes in varchar2 DEFAULT NULL

### http.variable Syntax

```
http.variable (ctext, cattributes);  
-- generates <VAR cattributes>ctext</VAR>
```

Parameters: ctext in varchar2  
              cattributes in varchar2 DEFAULT NULL

## HTP Physical Format Procedures

The physical format tags are used to specify the format of the marked text.

### http.bold Syntax

```
http.bold (ctext, cattributes); -- generates <B cattributes>ctext</B>
```

Parameters: ctext in varchar2  
              cattributes in varchar2 DEFAULT NULL

### http.italic Syntax

```
http.italic (ctext, cattributes); -- generates <I cattributes>ctext</I>
```

Parameters: ctext in varchar2  
              cattributes in varchar2 DEFAULT NULL

## HTP Physical Format Procedures (continued)

### http.teletype Syntax

`http.teletype (ctext, cattributes);` -- generates `<TT cattributes>ctext</TT>`

Parameters: `ctext` in `varchar2`  
`cattributes` in `varchar2` DEFAULT NULL

### HTP Form Procedures

The form tags are used to create and manipulate an HTML form. Forms are used to allow interactive data exchange between a Web Browser and a CGI program. Forms can have the following types of elements:

- Input: used for a large variety of types of input fields, for example:
  - single line text
  - single line password fields
  - checkboxes
  - radio buttons
  - submit buttons
- Text area: used to create a multi-line input field.
- Select: used to allow the user to chose one or more of a set of alternatives described by textual labels. Usually rendered as a pulldown, pop up, or a fixed size list.

### http.formOpen Syntax

`http.formOpen (curl, cmethod, ctarget, cenctype, cattributes);`  
-- generates `<FORM ACTION="curl" METHOD="cmethod" TARGET="ctarget" ENCTYPE="cenctype" cattributes>`

Parameters: `curl` in `varchar2`  
`cmethod` in `varchar2` DEFAULT 'POST'  
`ctarget` in `varchar2`  
`cenctype` in `varchar2` DEFAULT NULL  
`cattributes` in `varchar2` DEFAULT NULL

### http.formClose Syntax

`http.formClose;` -- generates `</FORM>`

### http.formCheckbox Syntax

`http.formCheckbox (cname, cvalue, cchecked, cattributes);`  
-- generates `<INPUT TYPE="checkbox" NAME="cname" VALUE="cvalue" CHECKED cattributes>`

Parameters: `cname` in `varchar2`  
`cvalue` in `varchar2` DEFAULT 'on'  
`cchecked` in `varchar2` DEFAULT NULL  
`cattributes` in `varchar2` DEFAULT NULL



## HTP Form Procedures (continued)

### http.formHidden Syntax

```
http.formHidden (cname, cvalue, cattributes);  
-- generates <INPUT TYPE="hidden" NAME="cname" VALUE="cvalue" cattributes>  
Parameters:  cname in varchar2  
             cvalue in varchar2 DEFAULT NULL  
             cattributes in varchar2 DEFAULT NULL
```

### http.formImage Syntax

```
http.formImage (cname, csrc, calign, cattributes);  
-- generates <INPUT TYPE="image" NAME="cname" SRC="csrc" ALIGN="calign"  
cattributes>  
Parameters:  cname in varchar2  
             csrc in varchar2  
             calign in varchar2 DEFAULT NULL  
             cattributes in varchar2 DEFAULT NULL;
```

### http.formPassword Syntax

```
http.formPassword (cname, csize, cmaxlength, cvalue, cattributes);  
-- generates <INPUT TYPE="password" NAME="cname" SIZE="csize"  
MAXLENGTH="cmmaxlength" VALUE="cvalue" cattributes>  
Parameters:  cname in varchar2  
             csize in varchar2  
             cmaxlength in varchar2 DEFAULT NULL  
             cvalue in varchar2 DEFAULT NULL  
             cattributes in varchar2 DEFAULT NULL
```

### http.formRadio Syntax

```
http.formRadio (cname, cvalue, cchecked, cattributes);  
-- generates <INPUT TYPE="radio" NAME="cname" VALUE="cvalue" CHECKED  
cattributes>  
Parameters:  cname in varchar2  
             cvalue in varchar2  
             cchecked in varchar2 DEFAULT NULL  
             cattributes in varchar2 DEFAULT NULL
```

### http.formReset Syntax

```
http.formReset (cvalue, cattributes);  
-- generates <INPUT TYPE="reset" VALUE="cvalue" cattributes>  
Parameters:  cvalue in varchar2 DEFAULT 'Reset'  
             cattributes in varchar2 DEFAULT NULL
```

## HTP Form Procedures (continued)

### http.formSubmit Syntax

```
http.formSubmit (cname, cvalue, cattributes);  
-- generates <INPUT TYPE="submit" NAME="cname" VALUE="cvalue" cattributes>  
Parameters:  cname in varchar2 DEFAULT NULL  
             cvalue in varchar2 DEFAULT 'Submit'  
             cattributes in varchar2 DEFAULT NULL
```

### http.formText Syntax

```
http.formText (cname, csize, cmaxlength, cvalue, cattributes);  
-- generates <INPUT TYPE="text" NAME="cname" SIZE="csize"  
             MAXLENGTH="cmmaxlength" VALUE="cvalue" cattributes>  
Parameters:  cname in varchar2  
             csize in varchar2 DEFAULT NULL  
             cmaxlength in varchar2 DEFAULT NULL  
             cvalue in varchar2 DEFAULT NULL  
             cattributes in varchar2 DEFAULT NULL
```

### http.formSelectOpen Syntax

```
http.formSelectOpen (cname, cprompt, nsize, cattributes);  
-- generates cprompt <SELECT NAME="cname" PROMPT="cprompt"  
                     SIZE="nsize" cattributes>
```

```
Parameters:  cname in varchar2  
             cprompt in varchar2 DEFAULT NULL  
             nsize in integer DEFAULT NULL  
             cattributes in varchar2 DEFAULT NULL
```

```
Example:  http.formSelectOpen('greatest_player',  
                             'Pick the greatest player:');  
          http.formSelectOption('Messier');  
          http.formSelectOption('Howe');  
          http.formSelectOption('Hull');.  
          http.formSelectOption('Gretzky');.  
          http.formSelectClose;
```

```
generates: Pick the greatest player:  
<SELECT NAME="greatest_player">  
          <OPTION>Messier  
          <OPTION>Howe  
          <OPTION>Hull  
          <OPTION>Gretzky  
</SELECT>
```

## HTP Form Procedures (continued)

### http.formSelectOption Syntax

```
http.formSelectOption (cvalue, cselected, cattributes);  
                        -- generates <OPTION SELECTED cattributes>cvalue
```

Parameters: cvalue in varchar2  
            cselected in varchar2 DEFAULT NULL  
            cattributes in varchar2 DEFAULT NULL

### http.formSelectClose Syntax

```
http.formSelectClose;      -- generates </SELECT>
```

### http.formTextarea Syntax

```
http.formTextarea (cname, nrows, ncolums, calign, cattributes);  
    -- generates <TEXTAREA NAME="cname" ROWS="nrows" COLS="ncolums"  
                  ALIGN="calign" cattributes></TEXTAREA>
```

Parameters: cname in varchar2  
            nrows in integer  
            ncolums in integer  
            calign in varchar2 DEFAULT NULL  
            cattributes in varchar2 DEFAULT NULL

### http.formTextarea2 Syntax

```
http.formTextarea(cname, nrows, ncolums, calign, cwrap, cattributes);  
    -- generates <TEXTAREA NAME="cname" ROWS="nrows" COLS="ncolums"  
                  ALIGN="calign" WRAP="cwrap" cattributes></TEXTAREA>
```

Parameters: cname in varchar2  
            nrows in integer  
            ncolums in integer  
            calign in varchar2 DEFAULT NULL  
            cwrap in varchar2 DEFAULT NULL  
            cattributes in varchar2 DEFAULT NULL

### http.formTextareaOpen Syntax

```
http.formTextareaOpen (cname, nrows, ncolums, calign, cattributes);  
    -- generates <TEXTAREA NAME="cname" ROWS="nrows" COLS="ncolums"  
                  ALIGN="calign" cattributes>
```

Parameters: cname in varchar2  
            nrows in integer  
            ncolums in integer  
            calign in varchar2 DEFAULT NULL  
            cattributes in varchar2 DEFAULT NULL

## HTP Form Procedures (continued)

### http.formTextareaOpen2 Syntax

```
http.formTextareaOpen (cname, nrows, ncolumns, calign, cwrap,  
cattributes);
```

```
-- generates <TEXTAREA NAME="cname" ROWS="nrows" COLS="ncolumns"  
ALIGN="calign" WRAP = "cwrap" cattributes>
```

Parameters:   cname in varchar2  
              nrows in integer  
              ncolumns in integer  
              calign in varchar2 DEFAULT NULL  
              cwrap in varchar2 DEFAULT NULL  
              cattributes in varchar2 DEFAULT NULL

### http.formTextareaClose Syntax

```
http.formTextareaClose;               -- generates </TEXTAREA>
```

## HTP Table Procedures

The Table tags allow the user to insert tables and manipulate the size and columns of the table in a document.

### http.tableOpen Syntax

```
http.tableOpen (cborder, calign, cnowrap, cclear, cattributes);  
              -- generates <TABLE "cborder" NOWRAP ALIGN="calign"  
                                                          CLEAR="cclear" cattributes>
```

Parameters:   cborder in varchar2 DEFAULT NULL  
              calign in varchar2 DEFAULT NULL  
              cnowrap in varchar2 DEFAULT NULL  
              cclear in varchar2 DEFAULT NULL  
              cattributes in varchar2 DEFAULT NULL;

### http.tableClose Syntax

```
http.tableClose;                      -- generates </TABLE>
```

### http.tableCaption Syntax

```
http.tableCaption (ccaption, calign, cattributes);  
              -- generates <CAPTION ALIGN="calign" cattributes>ccaption</CAPTION>
```

Parameters:   ccaption in varchar2  
              calign in varchar2 DEFAULT NULL  
              cattributes in varchar2 DEFAULT NULL

## HTP Table Procedures (continued)

### http.tableRowOpen Syntax

```
http.tableRowOpen (calign, cvalign, cdp, cnowrap, cattributes);  
    -- generates <TR ALIGN="calign" VALIGN="cvalign" DP="cdp"  
        NOWRAP cattributes>
```

Parameters: calign in varchar2 DEFAULT NULL  
cvalign in varchar2 DEFAULT NULL  
cdp in varchar2 DEFAULT NULL  
cnowrap in varchar2 DEFAULT NULL  
cattributes in varchar2 DEFAULT NULL

### http.tableRowClose Syntax

```
http.tableRowClose;          -- generates </TR>
```

### http.tableHeader Syntax

```
http.tableHeader(cvalue, calign, cdp,  
    cnowrap, crowspan, ccolspan, cattributes);  
    -- generates <TH ALIGN="calign" DP="cdp" ROWSPAN="crowspan"  
        COLSPAN="ccolspan" NOWRAP cattributes>cvalue</TH>
```

Parameters: cvalue in varchar2 DEFAULT NULL  
calign in varchar2 DEFAULT NULL  
cdp in varchar2 DEFAULT NULL  
cnowrap in varchar2 DEFAULT NULL  
crowspan in varchar2 DEFAULT NULL  
ccolspan in varchar2 DEFAULT NULL  
cattributes in varchar2 DEFAULT NULL

### http.tableData Syntax

```
http.tableData (cvalue, calign, cdp, crowspan,  
    ccolspan, cnowrap, cattributes);  
    -- generates <TD ALIGN="calign" DP="cdp" ROWSPAN="crowspan"  
        COLSPAN="ccolspan" NOWRAP cattributes>cvalue</TD>
```

Parameters: cvalue in varchar2 DEFAULT NULL  
calign in varchar2 DEFAULT NULL  
cdp in varchar2 DEFAULT NULL  
crowspan in varchar2 DEFAULT NULL  
ccolspan in varchar2 DEFAULT NULL  
cnowrap in varchar2 DEFAULT NULL  
cattributes in varchar2 DEFAULT NULL

## OWA Utilities (OWA\_UTIL Package)

This is a collection of useful utility procedures and functions. It is divided into the following areas:

- OWA\_UTIL HTML Utilities range from printing a signature tag on HTML pages to retrieving the values of CGI environment variables and performing URL redirects.
- OWA\_UTIL Dynamic SQL Utilities enable you to produce Web pages with dynamically generated SQL code.
- OWA\_UTIL Date Utilities make it easier to properly handle dates, which are simple strings in HTML, but are properly treated as a datatype by the Oracle RDBMS.

### OWA\_UTIL HTML Utilities

#### owa\_util.signature Syntax

Generates: a signature line in the HTML document that might look like the following:

"This page was produced by the **Oracle PL/SQL Agent** on August 9, 1995 09:30"

```
owa_util.signature;
```

```
owa_util.signature(cname); -- generates a signature line with a link to owa_util.showsources
```

Parameters: cname in varchar2

#### owa\_util.showsources Syntax

```
owa_util.showsources(cname); -- generates the source code of the procedure
```

Parameters: cname in varchar2

#### owa\_util.showpage Syntax

```
owa_util.showpage;
```

Generates static pages in SQL\*Plus that can then be accessed as a standard HTML page

#### owa\_util.get\_cgi\_env Syntax

This function returns the value of the specified CGI environment variable for PL/SQL procedure. If the value is not set, returns *null*.

```
owa_util.get_cgi_env(param_name in varchar2);
```

Parameters: param\_name in varchar2      Note that param\_name is case-insensitive

#### owa\_util.print\_cgi\_env Syntax

```
owa_util.print_cgi_env;
```

Prints the CGI environment variables made available by the PL/SQL Agent to the PL/SQL procedures.

#### owa\_util.mime\_header Syntax

```
owa_util.mime_header(ccontent_type, bclose_header);
```

-- generates: Content-type: <ccontent\_type>\n\n

Parameters: ccontent\_type in varchar2

bclose\_header in boolean DEFAULT TRUE

## **OWA\_UTIL HTML Utilities (continued)**

### **owa\_util.redirect\_url Syntax**

```
owa_util.redirect_url(curl, bclose_header);  
-- generates Location: <curl>\n\n
```

Parameters: curl in varchar2  
bclose\_header in boolean DEFAULT TRUE

### **owa\_util.status\_line Syntax**

```
owa_util.status_line(nstatus, creason, bclose_header);  
-- generates Status: <nstatus> <creason>\n\n
```

Parameters: nstatus in integer,  
creason in varchar2 DEFAULT NULL  
bclose\_header in boolean DEFAULT TRUE

### **owa\_util.http\_header\_close Syntax**

```
owa_util.http_header_close; -- generates a newline, which terminates the header
```

### **owa\_util.get\_owa\_service\_path Syntax**

This function returns the name of the currently active path with its full virtual path, plus the currently active DCD. For example, a call to get\_owa\_service\_path could return /ows-bin/myservice/owa/.

```
owa_util.get_owa_service_path;
```

### **owa\_util.tableprint Syntax**

```
owa_util.tablePrint; -- generates a preformatted or HTML table
```

This function returns True if there are more rows available beyond the nrow\_max requested.

Parameters: ctable in varchar2  
cattributes in varchar2 DEFAULT NULL  
ntable\_type in integer DEFAULT HTML\_TABLE  
Valid values are owa\_util.html\_table or owa\_util.pre\_table  
ccolumns in varchar2 DEFAULT `\*`  
cclasses in varchar2 DEFAULT NULL  
ccol\_aliases in varchar2 DEFAULT NULL  
nrow\_min in number DEFAULT 0  
nrow\_max in number DEFAULT NULL

## **OWA\_UTIL Dynamic SQL Utilities**

### **owa\_util.bind\_variables Syntax**

This utility is available only as a function. It prepares a SQL query, binding variables to it, and stores the output in an opened cursor. The integer returned by the function is a unique identifier for that cursor. You normally use this function as a parameter to a procedure to which you desire to send a dynamically generated query. You can use up to 25 bind variables.

```
owa_util.bind_variables(theQuery, bv1Name, bv1Value, bv2Name,  
bv2Value, ..., bv25Name, bv25Value)
```

Parameters: theQuery in varchar2 DEFAULT NULL  
bv1Name in varchar2 DEFAULT NULL  
bv1Value in varchar2 DEFAULT NULL  
bv2Name in varchar2 DEFAULT NULL  
bv2Value in varchar2 DEFAULT NULL  
...  
bv25Name in varchar2 DEFAULT NULL  
bv25Value in varchar2 DEFAULT NULL

### **owa\_util.cellsprint Syntax**

This procedure Generates: an HTML table from the output of a SQL query. SQL atomic data items are mapped to HTML cells and SQL rows to HTML rows. You must write the code to begin and end the table.

There are 4 versions of this procedure. Versions 1 and 3 each use a varchar2 parameter (p\_theQuery) that contains a query prepared with the owa\_util.bind\_variables function. Versions 2 and 4 use an open cursor. Versions 3 and 4 enable you to scroll through result sets.

#### **Version 1**

```
owa_util.cellsprint(p_theQuery, p_max_rows, p_format_numbers)
```

Generates: 

```
<tr><td>QueryResultItem</td><td>QueryResultItem</td></tr>  
<tr><td>QueryResultItem</td><td>QueryResultItem</td></tr>
```

Parameters: p\_theQuery in varchar2  
p\_max\_rows in number DEFAULT 100  
p\_format\_numbers in varchar2 DEFAULT NULL

#### **Version 2**

```
owa_util.cellsprint(p_theCursor, p_max_rows, p_format_numbers)
```

Generates: 

```
<tr><td>QueryResultItem</td><td>QueryResultItem</td></tr>  
<tr><td>QueryResultItem</td><td>QueryResultItem</td></tr>
```

Parameters: p\_theCursor in integer  
p\_max\_rows in number DEFAULT 100  
p\_format\_numbers in varchar2 DEFAULT NULL



## **OWA\_UTIL Dynamic SQL Utilities (continued)**

### **owa\_util.cellsprint Syntax (continued)**

#### **Version 3**

```
owa_util.cellsprint(p_theQuery, p_more_data, p_max_rows, p_skip_rec,  
p_format_numbers)
```

Generates: `<tr><td>QueryResultItem</td><td>QueryResultItem</td></tr>`  
`<tr><td>QueryResultItem</td><td>QueryResultItem</td></tr>`

Parameters: `p_theQuery` in varchar2  
`p_more_data` out boolean  
`p_max_rows` in number DEFAULT 100  
`p_skip_rec` in number DEFAULT 0  
`p_format_numbers` in varchar2 DEFAULT NULL

#### **Version 4**

```
owa_util.cellsprint(p_theQuery, p_more_data, p_max_rows, p_skip_rec,  
p_format_numbers)
```

Generates: `<tr><td>QueryResultItem</td><td>QueryResultItem</td></tr>`  
`<tr><td>QueryResultItem</td><td>QueryResultItem</td></tr>`

Parameters: `p_theCursor` in integer  
`p_more_data` out boolean  
`p_max_rows` in number DEFAULT 100  
`p_skip_rec` in number DEFAULT 0  
`p_format_numbers` in varchar2 DEFAULT NULL

### **owa\_util.listprint Syntax**

This procedure generates: an HTML pop-up selection list from the output of a SQL query. The parameter `p_cname` is the name of the list. The parameter `p_nsize` is the number of items in the list. Lastly, `p_multiple` is a boolean indicating whether multiple selection is permitted.

There are 2 versions of this procedure. Version 1 uses a parameter (`p_theQuery`) that contains a hardcoded SQL query . Version 2 uses a dynamic query prepared with the `owa_util.bind_variables` function.

#### **Version 1**

```
owa_util.listprint(p_theQuery, p_cname, p_nsize, p_multiple)
```

-- generates `<SELECT NAME="p_cname" SIZE="p_nsize">`  
`<OPTION value=QueryResultRow>...`  
`</SELECT>`

Parameters: `p_theQuery` in varchar2  
`p_cname` in varchar2  
`p_nsize` in number  
`p_multiple` in boolean DEFAULT FALSE

## OWA\_UTIL Dynamic SQL Utilities (continued)

### owa\_util.listprint Syntax (continued)

#### Version 2

```
owa_util.listprint(p_theCursor, p_cname, p_nsize, p_multiple)
-- generates  <SELECT NAME="p_cname" SIZE="p_nsize">
               <OPTION value=QueryResultRow>...
               </SELECT>
```

Parameters: p\_theCursor in integer  
            p\_cname in varchar2  
            p\_nsize in number  
            p\_multiple in boolean DEFAULT FALSE

### owa\_util.who\_called\_me Syntax

This procedure returns information about the PL/SQL code unit that invoked it:

- The name of the program unit: the package or procedure or function
- The owner of the program unit
- The line number within the program unit where the call was made
- The type of program unit that made the call: package body, anonymous block, procedure, or function

```
owa_util.who_called_me(owner, name, lineno, caller_t)
```

Parameters: owner out varchar2  
            name out varchar2  
            lineno out number  
            caller\_t out varchar2

## OWA\_UTIL Date Utilities

### Datatypes

**dateType** This is a PL/SQL table of varchar2(10) indexed by binary integer. It is intended to hold date information. The function owa\_util.todate converts an item of this type to the type DATE, which is understood and properly handled as a date by the database. The procedure owa\_util.choose\_date enables the user to select the desired date.

### owa\_util.calendarprint Syntax

This procedure creates a calendar in HTML. Each date in the calendar can contain any number of hypertext links. The query should have the following three columns:

1. A DATE that is used to correlate the information produced by the query with the calendar output automatically generated by the procedure. **Note:** the query output must be sorted on this column.
2. The text, if any, that you want printed for that date.
3. The destination for the links automatically generated in the second column.

This procedure has 2 versions. Version 1 uses a hard-coded query stored in a varchar2 string. Version 2 uses a dynamic query prepared with the owa\_util.bind\_variables function.

## **OWA\_UTIL Date Utilities (continued)**

### **owa\_util.calendarprint Syntax (continued)**

#### **Version 1**

```
owa_util.calendarprint(p_query, p_mf_only)
```

Parameters: p\_query in varchar2  
p\_mf\_only in varchar2 DEFAULT 'N'

#### **Version 2**

```
owa_util.calendarprint(p_cursor, p_mf_only)
```

Parameters: p\_cursor in integer  
p\_mf\_only in varchar2 DEFAULT 'N'

### **owa\_util.choose\_date Syntax**

```
owa_util.choose_date(p_name, p_date)
```

-- displays a date in HTML and allows the user to select a date

Parameters: p\_name in varchar2  
p\_date in date DEFAULT SYSDATE

### **owa\_util.todate Syntax**

This function converts the dateType datatype to the standard database DATE datatype.

```
owa_util.todate(p_datearray)
```

-- generates a standard PL/SQL date from p\_datearray

Parameters: p\_datearray in dateType

## **Optimistic Locking (OWA\_OPT\_LOCK Package)**

Since HTTP is a stateless protocol, conventional database locking schemes cannot be used directly. The owa\_opt\_lock package works around this by giving you a choice of two ways of dealing with the lost update problem: the problem caused if a user selects and then attempts to update a row whose value has been changed in the meantime by another user. The two techniques this package provides are as follows:

- The hidden fields method. This stores the previous values in hidden fields in the HTML page. When the update is performed, it checks these values against the current state of the database. This is implemented with the procedure owa\_opt\_lock.hidden\_fields.
- The checksum method. This stores a checksum rather than the values themselves. This is implemented with the procedure owa\_opt\_lock.checksum.

Both of these techniques are "optimistic". That is to say, they do not prevent other users from performing updates, but reject the current update if an intervening update has occurred.

## **Datatypes**

```
vcArray                    table of varchar2(2000) index by binary integer
```

This datatype is intended to hold rowids.

## **Optimistic Locking (OWA\_OPT\_LOCK Package) (continued)**

### **owa\_opt\_lock.checksum Syntax**

#### **Version 1**

This function returns a checksum based on the information passed. p\_buff should include the primary key of the table being "locked". This is a "pure" 32-bit checksum executed by the database and based on the Internet 1 protocol.

```
owa_opt_lock.checksum(p_buff)           -- generates a checksum of type number
```

Parameters: p\_buff in varchar2

#### **Version 2**

This function returns a checksum based on the information passed. The input Parameters: are the owner, name, and rowid of the table and row being "locked". Use the owa\_opt\_lock.get\_rowid function to convert vcArray values to proper rowids for processing by this function. This is a "impure" 32-bit checksum based on the Internet 1 protocol.

```
owa_opt_lock.checksum(p_owner, p_tname, p_rowid)
                                           -- generates a checksum of type number
```

Parameters: p\_owner in varchar2

p\_tname in varchar2

p\_rowid in rowid

### **owa\_opt\_lock.get\_rowid Syntax**

```
owa_opt_lock.get_rowid(p_old_values) -- generates a ROWID
```

Parameters: p\_old\_values in vcArray

### **owa\_opt\_lock.hidden\_fields Syntax**

```
owa_opt_lock.hidden_fields(p_owner, p_tname, p_rowid)
```

Stores the owner, name, and rowids of the table and rows being "locked" with the current values in hidden HTML fields in the Web page. At update time, you can use owa\_opt\_lock.verify\_hidden\_fields to reject the update if these values have changed.

Parameters: p\_owner in varchar2

p\_tname in varchar2

p\_rowid in rowid

### **owa\_opt\_lock.verify\_hidden\_fields Syntax**

```
owa_opt_lock.verify_hidden_fields(p_old_values)
```

This function takes a vcArray containing rowid values and returns a boolean indicating whether the associated rows have changed since the fields were hidden.

Parameters: p\_old\_values in vcArray

## Pattern Matching Utilities (OWA\_PATTERN Package)

The OWA\_PATTERN package enables you to do sophisticated string manipulation using regular expressions. OWA\_PATTERN provides the following three operations:

- **MATCH.** This determines whether a regular expression exists in a string. This is a function that returns TRUE or FALSE.
- **AMATCH.** This is a more sophisticated variation on MATCH that lets you specify *where* in the string the match has to occur. This is a function that returns as an integer the end of the location in the string where the regular expression was found. If the regular expression is not found, it returns 0.
- **CHANGE.** This lets you replace the portion of the string that matched the regular expression with a new string. CHANGE can be either a procedure or a function. If a function, it returns the number of times the regular expression was found and replaced.

The OWA\_PATTERN operations all use the following three parameters:

- *line.* This is the target to be examined for a match. Despite the name, it can be more than one line of text or can be a PL/SQL table of type multi\_line
- *pat.* This is the regular expression the functions attempts to locate in *line*. This regular expression uses the special tokens explained shortly. Note: in CHANGE, this is called *from\_str*.
- *flags.* These are arguments that control how the search is to be performed.

MATCH, AMATCH, and CHANGE are overloaded. That is to say, there are several versions of each, distinguished by the parameters they take.

### Regular Expressions

You specify a regular expression by creating the string you want to match interspersed with various wildcard tokens and quantifiers. The wildcard tokens all match something other than themselves, and the quantifiers modify the meaning of tokens or of literals by specifying such things as how often each is to be applied.

### Wildcard Tokens

<code>^</code>	Matches newline or the beginning of the target.
<code>\$</code>	Matches newline or the end of the target.
<code>\n</code>	Matches newline.
<code>.</code>	Matches any character except newline.
<code>\t</code>	Matches tab.
<code>\d</code>	Matches digits [0-9]
<code>\D</code>	Matches non-digits [not 0-9]
<code>\w</code>	Matches word characters (alphanumeric) [0-9, a-z, A-Z or _]
<code>\W</code>	Matches non-word characters [not 0-9, a-z, A-Z or _]
<code>\s</code>	Matches whitespace characters [blank, tab, or newline]
<code>\S</code>	Matches non-whitespace characters [not blank, tab, or newline]
<code>\b</code>	Matches "word" boundaries (between \w and \W)
<code>\x&lt;HEX&gt;</code>	Matches the value in the current character set of the two hexadecimal digits

## Pattern Matching Utilities (OWA\_PATTERN Package) (continued)

### Wildcard Tokens (continued)

`\<OCT>` Matches the value in the current character set of the two or three octal digits  
`\` Followed by any character not covered by another case matches that character  
& Applies only to CHANGE. This causes the string that matched the regular expression to be included in the string that replaces it. This differs from the other tokens in that it specifies how a target is changed rather than how it is matched. This is explained further under CHANGE.

### Quantifiers

Any of the above tokens except & can have its meaning extended by any of the following quantifiers. You can also apply these quantifiers to literals.

? 0 or 1 occurrences  
\* 0 or more occurrences  
+ 1 or more occurrences  
{n} Exactly *n* occurrences  
(n,) At least *n* occurrences  
{n,m} At least *n*, but not more than *m*, occurrences

### Flags

In addition to targets and regular expressions, the OWA\_PATTERN functions and procedures can use flags to affect how they are interpreted. The recognized flags are as follows:

- i This indicates a case-insensitive search.
- g This applies only to CHANGE. It indicates a global replace. That is to say, all portions of the target that match the regular expression are replaced.

### Datatypes

The following special datatype is used by OWA\_PATTERN:

**pattern** A PL/SQL table of 4 byte VARCHAR2 strings, indexed by BINARY INTEGER. This is an alternative way to store your regular expression rather than in simple VARCHAR2 strings. The advantages of this is that you can use a pattern as both an input and output parameter. Thus, you can pass the same regular expression to several subsequent OWA\_PATTERN function calls, and it only has to be parsed once.

### Example of Using MATCH

```
MATCH ('BATMAN', 'Bat.*', i);
```

This is how the function is interpreted: BATMAN is the target where we are searching for the regular expression. Bat.\* is the regular expression we are attempting to find. The period (.) indicates any character other than newline, and the asterisk (\*) indicates any 0 or more of such. Therefore, this regular expression specifies that a matching target consists of 'Bat', followed by any set of characters neither ending in nor including a newline (which does not match the period). The *i* at the end is a flag indicating that case is to be ignored in the search. This example returns TRUE, indicating that a match had been found.

If multiple overlapping strings can match the regular expression, OWA\_PATTERN takes the longest match.

## Pattern Matching Utilities (OWA\_PATTERN Package) (continued)

### owa\_pattern.match

MATCH is a function that returns TRUE or FALSE depending on whether a match was found.

The target can be either a simple VARCHAR2 string of less than 32K or a multi\_line. If a multi\_line is used, there is a parameter called *rlist*, after the regular expression, but before the flags. This is a list of the chunks where matches were found. A VARCHAR2 implies use of one of the first four versions of this function. A multi\_line implies use of version 5 or 6.

The regular expression can be either a VARCHAR2 string or a *pattern*. You can create a *pattern* from a string using the *getpat* function described later in this section. If you use a pattern, you are using one of the even-number versions of the function.

If the line is a string, then you can add an optional output parameter called *backrefs*. This goes after the regular expression, but before the flags. You cannot use *backrefs* if you pass a multi\_line to MATCH, because this is the same place in the parameter list that the *rlist* parameter would go. The *backrefs* parameter is a row\_list that holds each string in the target that was matched by a sequence of tokens in the regular expression. If you use *backrefs*, you are using versions 3 or 4 of the function.

#### Version 1 Syntax

```
owa_pattern.match(line, pat, flags)
```

Parameters: line in varchar2  
            pat in varchar2  
            flags in varchar2 DEFAULT NULL

#### Version 2 Syntax

```
owa_pattern.match(line, pat, flags)
```

Parameters: line in varchar2  
            pat in out pattern  
            flags in varchar2 DEFAULT NULL

#### Version 3 Syntax

```
owa_pattern.match(line, pat, backrefs, flags)
```

Parameters: line in varchar2  
            pat in varchar2  
            backrefs out owa\_text.vc\_arr  
            flags in varchar2 DEFAULT NULL

#### Version 4 Syntax

```
owa_pattern.match(line, pat, backrefs, flags)
```

Parameters: line in varchar2  
            pat in out pattern  
            backrefs out owa\_text.vc\_arr  
            flags in varchar2 DEFAULT NULL

## Pattern Matching Utilities (OWA\_PATTERN Package) (continued)

### owa\_pattern.match (continued)

#### Version 5 Syntax

```
owa_pattern.match(mline, pat, rlist, flags)
```

Parameters: mline in owa\_text.multi\_line  
pat in varchar2  
rlist out owa\_text.row\_list  
flags in varchar2 DEFAULT NULL

#### Version 6 Syntax

```
owa_pattern.match(mline, pat, rlist, flags)
```

Parameters: mline in owa\_text.multi\_line  
pat in out pattern  
rlist out owa\_text.row\_list  
flags in varchar2 DEFAULT NULL

### owa\_pattern.amatch

AMATCH is a function that returns the number of characters from the beginning of the target to the end of the first match found (AMATCH stops searching after the first match). If no match is found, it returns 0.

Following the target, but preceding the regular expression, is the input parameter *from\_loc*. This indicates how many characters from the beginning of the target the search should commence.

The regular expression can be either a VARCHAR2 string or a pattern. You can create a pattern from a string using the *getpat* function described later in this section.. If you use a pattern, you are using one of the even-number versions of the function.

After the regular expression, but before the flags, you can add an optional output parameter called "backrefs". This is a PL/SQL table that will hold each string in the target that was matched by a sequence of tokens in the regular expression. If you use *backrefs*, you are using the third or fourth version of the function.

#### Version 1 Syntax

```
owa_pattern.match(line, from_loc, pat, flags)
```

Parameters: line in varchar2  
from\_loc in integer  
pat in varchar2  
flags in varchar2 DEFAULT NULL

#### Version 2 Syntax

```
owa_pattern.match(line, from_loc, pat, flags)
```

Parameters: line in varchar2  
from\_loc in integer  
pat in out pattern  
flags in varchar2 DEFAULT NULL



## Pattern Matching Utilities (OWA\_PATTERN Package) (continued)

### owa\_pattern.amatch (continued)

#### Version 3 Syntax

owa\_pattern.match(line, from\_loc, pat, backrefs, flags)

Parameters: line in varchar2  
from\_loc in integer  
pat in varchar2  
backrefs out owa\_text.vc\_arr  
flags in varchar2 DEFAULT NULL

#### Version 4 Syntax

owa\_pattern.match(line, from\_loc, pat, backrefs, flags)

Parameters: line in varchar2  
from\_loc in integer  
pat in out pattern  
backrefs out owa\_text.vc\_arr  
flags in varchar2 DEFAULT NULL

### owa\_pattern.change

CHANGE can be either a procedure or a function, depending on how it is invoked. If a function, it returns the number of changes made. If the flag 'g' is not used, this number can be only 0 or 1.

The target can be either a simple VARCHAR2 string of less than 32K or a multi\_line.

Unlike MATCH and AMATCH, the regular expression can only be a VARCHAR2 string, not a pattern.

Following the regular expression is the string that is to replace it. This string can use the token ampersand (&), which indicates that the portion of the target that matched the regular expression is to be included in the expression that replaces it. For example:

```
CHANGE('Cats in pajamas', 'C.+in', '& red ')
```

The regular expression matches the substring 'Cats in'. It then replaces this string with '& red'. & indicates 'Cats in', since that's what matched the regular expression. Thus, this procedure replaces the string 'Cats in pajamas' with 'Cats in red pajamas'. Of course, we used a literal here for clarity. In actuality, 'Cats in pajamas' would be the value of a variable, and that value would be changed. Were this a function rather than a procedure, the value it would return would not be 'Cats in red pajamas' but 1, indicating that a single substitution had been made.

The flag 'g' indicates that all matching portions of the target are to be replaced by the regular expression. Otherwise, only the first match is replaced.

#### Version 1 Syntax

Syntax owa\_pattern.change(line, from\_str, to\_str, flags)

Parameters: line in out varchar2  
from\_str in varchar2  
to\_str in varchar2  
flags in varchar2 DEFAULT NULL

## **Pattern Matching Utilities (OWA\_PATTERN Package) (continued)**

### **owa\_pattern.change (continued)**

#### **Version 2 Syntax**

```
owa_pattern.change(line, from_str, to_str, backrefs, flags)
```

Parameters: mline in out multi\_line  
            from\_str in varchar2  
            to\_str in varchar2  
            flags in varchar2 DEFAULT NULL

### **owa\_pattern.getpat Syntax**

This procedure converts a VARCHAR2 string into the pattern datatype.

```
owa_pattern.getpat(arg, pat)
```

Parameters: arg in varchar2  
            pat in out pattern

## Text Manipulation Utilities (OWA\_TEXT Package)

The OWA\_TEXT package is chiefly used by OWA\_PATTERN, but the functions are externalized so that you can use them directly if desired.

### Datatypes

`vc_arr` A PL/SQL table of 32K VARCHAR2 strings, indexed by BINARY INTEGER. This is a component of *owa\_text.multi\_line*

`multi_line` A record

`int_array` A PL/SQL table of INTEGER indexed by BINARY INTEGER. This is a component of *owa\_text.row\_list*

`row_list` A record

### owa\_text.stream2multi Syntax

`owa_text.stream2multi(stream, mline)` -- generates a multi\_line from a string

Parameters: `stream` in varchar2

`mline` out multi\_line

### owa\_text.add2multi Syntax

`owa_text.add2multi(stream, mline, continue)`

Adds content to a multi\_line. The continue parameter specifies whether to begin appending within the previous final chunk (assuming it is less than 32K) or to start a new chunk.

Parameters: `stream` in varchar2

`mline` out multi\_line

`continue` in boolean DEFAULT TRUE

### owa\_text.new\_row\_list Syntax

`owa_text.new_row_list(rlist)`

If called as a function it takes no parameters and outputs a new, initially empty, row\_list. If a procedure, it places into the rlist parameter a new, initially empty, row\_list.

Parameters: `rlist` in row\_list

### owa\_text.print\_multi Syntax

`owa_text.print_multi(mline)` -- uses http.print to print the multi\_line

Parameters: `mline` in multi\_line

### owa\_text.print\_row\_list Syntax

`owa_text.print_row_list(rlist)` -- uses http.print to print the row\_list

Parameters: `rlist` in row\_list

## **Image Map Utilities (OWA\_IMAGE Package)**

This is a set of datatypes and functions for manipulating HTML image maps.

### **Datatypes**

**point** This provides the x and y coordinates of a user's click on an image map.

### **Package Variables**

**null\_point** This is a variable used to default point parameters. X and Y both are NULL.

### **owa\_image.get\_x Syntax**

`owa_image.get_x(p)`                      -- returns the X coordinate of the image map.

Parameters: `p` in `point`

### **owa\_image.get\_y Syntax**

`owa_image.get_y(p)`                      -- returns the Y coordinate of the image map.

Parameters: `p` in `point`

## Cookie Utilities (OWA\_COOKIE Package)

OWA\_COOKIE provides wrappers so that you can send cookies to and get them from the client's browser. Cookies are strings that are opaque to the client, but which maintain state throughout the client's session, or longer if an expiration date is included. Your system date is calculated with reference to the information specified in the OWA\_INIT package.

### Datatypes

`cookie` Since the HTTP standard is that cookie names can be overloaded, that is, multiple values can be associated with the same cookie name, this is a PL/SQL RECORD holding all values associated with a given cookie name. The fields are as follows:

<code>name</code>	<code>varchar2(4K)</code>
<code>vals</code>	<code>vc_arr</code>
<code>numvals</code>	<code>integer</code>

### `owa_cookie.send` Syntax

```
owa_cookie.send(name, value, expires, path, domain, secure)
    -- generates Set-Cookie: <name>=<value> expires=<expires>
    path=<path> domain=<domain> secure
```

Parameters: `name` in `varchar2`  
`value` in `varchar2`  
`expires` in `date` DEFAULT NULL  
`path` in `varchar2` DEFAULT NULL  
`domain` in `varchar2` DEFAULT NULL  
`secure` in `varchar2` DEFAULT NULL

### `owa_cookie.get` Syntax

```
owa_cookie.get(name)                                -- generates a cookie
```

Parameters: `name` in `varchar2`

### `owa_cookie.get_all` Syntax

```
owa_cookie.get_all(names, vals, num_vals)
```

Generates arrays of the names and values of the cookies, in the order received, and the count of the combinations

Parameters: `names` out `vc_arr`  
`vals` out `vc_arr`  
`num_vals` out `integer`

### `owa_cookie.remove` Syntax

```
owa_cookie.remove(name, value, path)
    -- generates Set-Cookie: <name>=<value> expires=01-JAN-1990 path=<path>
```

Parameters: `name` in `varchar2`  
`value` in `varchar2`  
`path` in `varchar2` DEFAULT NULL

