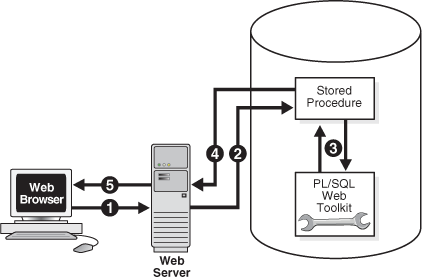
**Invoking a PL/SQL Web Application**

Typically, a Web application written in PL/SQL is a set of stored procedures that interact with Web browsers through HTTP. A set of interlinked, dynamically generated HTML pages forms the user interface of a web application.

The program flow of a PL/SQL Web application is similar to that in a CGI Perl script. Developers often use CGI scripts to produce Web pages dynamically, but such scripts are often not optimal for accessing Oracle Database. Delivering Web content with PL/SQL stored procedures provides the power and flexibility of database processing. For example, you can use DML, dynamic SQL, and cursors. You also eliminate the process overhead of forking a new CGI process to handle each HTTP request.

[Figure 11-1](http://docs.oracle.com/cd/B19306_01/appdev.102/b14251/adfns_web.htm#BGBHFAHI) illustrates the generic process for a PL/SQL Web application.

***Figure 11-1 PL/SQL Web Application***



The process includes the following steps:

1. A user visits a Web page, follows a hypertext link, or submits data in a form, which causes the browser to send a HTTP request for a URL to an HTTP server.
2. The HTTP server invokes a stored procedure on an Oracle database according to the data encoded in the URL. The data in the URL takes the form of parameters to be passed to the stored procedure.
3. The stored procedure calls subprograms in the PL/SQL Web Toolkit. Typically, subprograms such as HTP.Print generate Web pages dynamically. A generated Web page varies depending on the database contents and the input parameters.
4. The subprograms pass the dynamically generated page to the Web server.
5. The Web server delivers the page to the client.

## Implementing PL/SQL Web Applications

You can implement a web browser-based application entirely in PL/SQL with these Oracle Database components:

* [PL/SQL Gateway](http://docs.oracle.com/cd/E11882_01/appdev.112/e41502/adfns_web.htm#CHDJDIII)
* [PL/SQL Web Toolkit](http://docs.oracle.com/cd/E11882_01/appdev.112/e41502/adfns_web.htm#CHDFIIHB)

### PL/SQL Gateway

The PL/SQL gateway enables a web browser to invoke a PL/SQL stored subprogram through an HTTP listener. The gateway is a platform on which PL/SQL users develop and deploy PL/SQL web applications.

#### mod\_plsql

mod\_plsql is one implementation of the PL/SQL gateway. The module is a plug-in of Oracle HTTP Server and enables web browsers to invoke PL/SQL stored subprograms. Oracle HTTP Server is a component of both Oracle Application Server and the database.

The mod\_plsql plug-in enables you to use PL/SQL stored subprograms to process HTTP requests and generate responses. In this context, an HTTP request is a URL that includes parameter values to be passed to a stored subprogram. PL/SQL gateway translates the URL, invokes the stored subprogram with the parameters, and returns output (typically HTML) to the client.

Some advantages of using mod\_plsql over the embedded form of the PL/SQL gateway are:

* You can run it in a firewall environment in which the Oracle HTTP Server runs on a firewall-facing host while the database is hosted behind a firewall. You cannot use this configuration with the embedded gateway.
* The embedded gateway does not support mod\_plsql features such as dynamic HTML caching, system monitoring, and logging in the Common Log Format.

#### Embedded PL/SQL Gateway

You can use an embedded version of the PL/SQL gateway that runs in the XML DB HTTP Listener in the database. It provides the core features of mod\_plsql in the database but does not require the Oracle HTTP Server. You configure the embedded PL/SQL gateway with the DBMS\_EPG package in the PL/SQL Web Toolkit.

Some advantages of using the embedded gateway over mod\_plsql are as follows:

* You can invoke PL/SQL web applications such as Application Express without installing Oracle HTTP Server, thereby simplifying installation, configuration, and administration of PL/SQL based web applications.
* You use the same configuration approach that is used to deliver content from Oracle XML DB in response to FTP and HTTP requests.

## Using Embedded PL/SQL Gateway

The embedded gateway functions very similar to the mod\_plsql gateway. Before using the embedded version of the gateway, familiarize yourself with the[*Oracle HTTP Server mod\_plsql User's Guide*](http://www.oracle.com/pls/topic/lookup?ctx=db112&id=YPMOD). Much of the information is the same or similar.

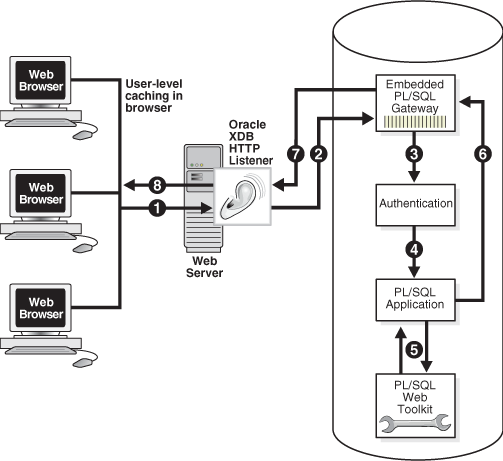
**Topics:**

* [How Embedded PL/SQL Gateway Processes Client Requests](http://docs.oracle.com/cd/E11882_01/appdev.112/e41502/adfns_web.htm#CHECCDEB)
* [Installing Embedded PL/SQL Gateway](http://docs.oracle.com/cd/E11882_01/appdev.112/e41502/adfns_web.htm#CHEDJACD)
* [Configuring Embedded PL/SQL Gateway](http://docs.oracle.com/cd/E11882_01/appdev.112/e41502/adfns_web.htm#CHEIGJIG)
* [Invoking PL/SQL Stored Subprograms Through Embedded PL/SQL Gateway](http://docs.oracle.com/cd/E11882_01/appdev.112/e41502/adfns_web.htm#CHEICAAC)
* [Securing Application Access with Embedded PL/SQL Gateway](http://docs.oracle.com/cd/E11882_01/appdev.112/e41502/adfns_web.htm#CHEBAEFB)
* [Restrictions in Embedded PL/SQL Gateway](http://docs.oracle.com/cd/E11882_01/appdev.112/e41502/adfns_web.htm#CHEEGIED)
* [Using Embedded PL/SQL Gateway: Scenario](http://docs.oracle.com/cd/E11882_01/appdev.112/e41502/adfns_web.htm#CHEHBJHI)

### How Embedded PL/SQL Gateway Processes Client Requests

[Figure 9-2](http://docs.oracle.com/cd/E11882_01/appdev.112/e41502/adfns_web.htm#BGBBAEIG) illustrates the process by which the embedded gateway handles client HTTP requests.

***Figure 9-2 Processing Client Requests with Embedded PL/SQL Gateway***



**PL/SQL Web Toolkit**

This set of PL/SQL packages is a generic interface that enables you to use stored subprograms invoked by mod\_plsql at run time.

In response to a browser request, a PL/SQL subprogram updates or retrieves data from Oracle Database according to the user input. It then generates an HTTP response to the browser, typically in the form of a file download or HTML to be displayed. The PL/SQL Web Toolkit API enables stored subprograms to perform actions such as:

* Obtain information about an HTTP request
* Generate HTTP headers such as content-type and mime-type
* Set browser cookies
* Generate HTML pages

[Table 9-1](http://docs.oracle.com/cd/E11882_01/appdev.112/e41502/adfns_web.htm#CHEBCGGF) describes commonly used PL/SQL Web Toolkit packages.

***Table 9-1 Commonly Used Packages in the PL/SQL Web Toolkit***

| **Package** | **Description of Contents** |
| --- | --- |
| [HTF](http://docs.oracle.com/cd/E11882_01/appdev.112/e40758/w_htf.htm#ARPLS390) | Function versions of the subprograms in the htp package. The function versions do not directly generate output in a web page. Instead, they pass their output as return values to the statements that invoke them. Use these functions when you must nest function calls. |
| [HTP](http://docs.oracle.com/cd/E11882_01/appdev.112/e40758/w_htp.htm#ARPLS391) | Subprograms that generate HTML tags. For example, the procedure htp.anchor generates the HTML anchor tag, <A>. |
| [OWA\_CACHE](http://docs.oracle.com/cd/E11882_01/appdev.112/e40758/w_cache.htm#ARPLS386) | Subprograms that enable the PL/SQL gateway cache feature to improve performance of your PL/SQL web application.  You can use this package to enable expires-based and validation-based caching with the PL/SQL gateway file system. |
| [OWA\_COOKIE](http://docs.oracle.com/cd/E11882_01/appdev.112/e40758/w_cook.htm#ARPLS387) | Subprograms that send and retrieve HTTP cookies to and from a client web browser. Cookies are strings a browser uses to maintain state between HTTP calls. State can be maintained throughout a client session or longer if a cookie expiration date is included. |
| [OWA\_CUSTOM](http://docs.oracle.com/cd/E11882_01/appdev.112/e40758/w_custom.htm#ARPLS388) | The authorize function used by cookies. |
| [OWA\_IMAGE](http://docs.oracle.com/cd/E11882_01/appdev.112/e40758/w_image.htm#ARPLS392) | Subprograms that obtain the coordinates where a user clicked an image. Use this package when you have an image map whose destination links invoke a PL/SQL gateway. |
| [OWA\_OPT\_LOCK](http://docs.oracle.com/cd/E11882_01/appdev.112/e40758/w_optlock.htm#ARPLS393) | Subprograms that impose database optimistic locking strategies to prevent lost updates. Lost updates can otherwise occur if a user selects, and then attempts to update, a row whose values were changed in the meantime by another user. |
| [OWA\_PATTERN](http://docs.oracle.com/cd/E11882_01/appdev.112/e40758/w_patt.htm#ARPLS398) | Subprograms that perform string matching and string manipulation with regular expressions. |
| [OWA\_SEC](http://docs.oracle.com/cd/E11882_01/appdev.112/e40758/w_sec.htm#ARPLS395) | Subprograms used by the PL/SQL gateway for authenticating requests. |
| [OWA\_TEXT](http://docs.oracle.com/cd/E11882_01/appdev.112/e40758/w_text.htm#ARPLS396) | Subprograms used by package OWA\_PATTERN for manipulating strings. You can also use them directly. |
| [OWA\_UTIL](http://docs.oracle.com/cd/E11882_01/appdev.112/e40758/w_util.htm#ARPLS397) | These types of utility subprograms:   * Dynamic SQL utilities to produce pages with dynamically generated SQL code. * HTML utilities to retrieve the values of CGI environment variables and perform URL redirects. * Date utilities for correct date-handling. Date values are simple strings in HTML, but must be properly treated as an Oracle Database data type. |
| [WPG\_DOCLOAD](http://docs.oracle.com/cd/E11882_01/appdev.112/e40758/w_wdoclo.htm#ARPLS200) | Subprograms that download documents from a document repository that you define using the DAD configuration. |

## Generate HTML Output from PL/SQL?

Traditionally, PL/SQL web applications have used function calls to generate each HTML tag for output, using the PL/SQL web toolkit packages that come with Oracle9i Application Server (iAS), Oracle Application Server (OAS), and WebDB:

owa\_util.mime\_header('text/html');

htp.htmlOpen;

htp.headOpen;

htp.title('Title of the HTML File');

htp.headClose;

htp.bodyOpen( cattributes => 'TEXT="#000000" BGCOLOR="#FFFFFF"');

htp.header(1, 'Heading in the HTML File');

htp.para;

htp.print('Some text in the HTML file.');

htp.bodyClose;

htp.htmlClose;

## How Do I Pass Parameters to a PL/SQL Web Application?

To be useful in a wide variety of situations, a web application must be interactive enough to allow user choices. To keep the attention of impatient web surfers, you should streamline the interaction so that users can specify these choices very simply, without a lot of decision-making or data entry.

The main methods of passing parameters to PL/SQL web applications are:

* Using HTML form tags. The user fills in a form on one web page, and all the data and choices are transmitted to a stored procedure when the user clicks the Submit button on the page.
* Hardcoded in the URL. The user clicks on a link, and a set of predefined parameters are transmitted to a stored procedure. Typically, you would include separate links on your web page for all the choices that the user might want.

#### Passing List and Dropdown List Parameters from an HTML Form

List boxes and dropdown lists are implemented using the same HTML tag (<SELECT>).

Use a list box for a large number of choices, where the user might have to scroll to see them all, or to allow multiple selections. List boxes are good for showing items in alphabetical order, so that users can find an item quickly without reading all the choices.

Use a dropdown list for a small number of choices, or where screen space is limited, or for choices in an unusual order. The dropdown captures the first-time user's attention and makes them read the items. If you keep the choices and order consistent, users can memorize the motion of selecting an item from the dropdown list, allowing them to make selections quickly as they gain experience.

#### Passing Radio Button and Checkbox Parameters from an HTML Form

Radio buttons pass either a null value (if none of the radio buttons in a group is checked), or the value specified on the radio button that is checked.

To specify a default value for a set of radio buttons, you can include the CHECKED attribute in one of the INPUT tags, or include a DEFAULT clause on the parameter within the stored procedure. When setting up a group of radio buttons, be sure to include a choice that indicates "no preference", because once the user selects a radio button, they can still select a different one, but they cannot clear the selection completely. For example, include a "Don't Care" or "Don't Know" selection along with "Yes" and "No" choices, in case someone makes a selection and then realizes it was wrong.

Checkboxes need special handling, because your stored procedure might receive a null value, a single value, or multiple values:

All the checkboxes with the same NAME attribute make up a checkbox group. If none of the checkboxes in a group is checked, the stored procedure receives a null value for the corresponding parameter.

If one checkbox in a group is checked, the stored procedure receives a single VARCHAR2 parameter.

If more than one checkbox in a group is checked, the stored procedure receives a parameter with the PL/SQL type TABLE OF VARCHAR2. You must declare a type like this, or use a predefined one like OWA\_UTIL.IDENT\_ARR. To retrieve the values, use a loop:

CREATE OR REPLACE PROCEDURE handle\_checkboxes ( checkboxes owa\_util.ident\_arr )

AS

BEGIN

...

FOR i IN 1..checkboxes.count

LOOP

htp.print('<p>Checkbox value: ' || checkboxes(i));

END LOOP;

...

END;

/

show errors;

#### Passing Entry Field Parameters from an HTML Form

Entry fields require the most validation, because a user might enter data in the wrong format, out of range, and so on. If possible, validate the data on the client side using dynamic HTML or Java, and format it correctly for the user or prompt them to enter it again.

For example:

* You might prevent the user from entering alphabetic characters in a numeric entry field, or from entering characters once a length limit is reached.
* You might silently remove spaces and dashes from a credit card number if the stored procedure expects the value in that format.
* You might inform the user immediately when they type a number that is too large, so that they can retype it.

Because you cannot always rely on such validation to succeed, code the stored procedures to deal with these cases anyway. Rather than forcing the user to use the Back button when they enter wrong data, display a single page with an error message and the original form with all the other values filled in.

For sensitive information such as passwords, a special form of the entry field, <INPUT TYPE=PASSWORD>, hides the text as it is typed in.

For example, the following procedure accepts two strings as input. The first time it is called, the user sees a simple form prompting for the input values. When the user submits the information, the same procedure is called again to check if the input is correct. If the input is OK, the procedure processes it. If not, the procedure prompts for new input, filling in the original values for the user.

-- Store a name and associated zip code in the database.

CREATE OR REPLACE PROCEDURE associate\_name\_with\_zipcode

(

name VARCHAR2 DEFAULT NULL,

zip VARCHAR2 DEFAULT NULL

)

AS

booktitle VARCHAR2(256);

BEGIN

-- Both entry fields must contain a value. The zip code must be 6 characters.

-- (In a real program you would perform more extensive checking.)

IF name IS NOT NULL AND zip IS NOT NULL AND length(zip) = 6 THEN

store\_name\_and\_zipcode(name, zip);

htp.print('<p>The person ' || name || ' has the zip code ' || zip || '.');

-- If the input was OK, we stop here and the user does not see the form again.

RETURN;

END IF;

-- If some data was entered, but it is not correct, show the error message.

IF (name IS NULL AND zip IS NOT NULL)

OR (name IS NOT NULL AND zip IS NULL)

OR (zip IS NOT NULL AND length(zip) != 6)

THEN

htp.print('<p><b>Please re-enter the data. Fill in all fields, and use a

6-digit zip code.</b>');

END IF;

-- If the user has not entered any data, or entered bad data, prompt for

-- input values.

-- Make the form call the same procedure to check the input values.

htp.formOpen( 'scott.associate\_name\_with\_zipcode', 'GET');

htp.print('<p>Enter your name:</td>');

htp.print('<td valign=center><input type=text name=name value="' || name ||

'">');

htp.print('<p>Enter your zip code:</td>');

htp.print('<td valign=center><input type=text name=zip value="' || zip ||

'">');

htp.formSubmit(NULL, 'Submit');

htp.formClose;

END;

/

show errors;

#### Passing Hidden Parameters from an HTML Form

One technique for passing information through a sequence of stored procedures, without requiring the user to specify the same choices each time, is to include hidden parameters in the form that calls a stored procedure. The first stored procedure places information, such as a user name, into the HTML form that it generates. The value of the hidden parameter is passed to the next stored procedure, as if the user had entered it through a radio button or entry field.

Other techniques for passing information from one stored procedure to another include:

* Sending a "cookie" containing the persistent information to the browser. The browser then sends this same information back to the server when accessing other web pages from the same site. Cookies are set and retrieved through the HTTP headers that are transferred between the browser and the web server before the HTML text of each web page.
* Storing the information in the database itself, where later stored procedures can retrieve it. This technique involves some extra overhead on the database server, and you must still find a way to keep track of each user as multiple users access the server at the same time.

#### Submitting a Completed HTML Form

By default, an HTML form must have a Submit button, which transmits the data from the form to a stored procedure or CGI program. You can label this button with text of your choice, such as "Search", "Register", and so on.

You can have multiple forms on the same page, each with its own form elements and Submit button. You can even have forms consisting entirely of hidden parameters, where the user makes no choice other than clicking the button.

Using Javascript or other scripting languages, you can do away with the Submit button and have the form submitted in response to some other action, such as selecting from a dropdown list. This technique is best when the user only makes a single selection, and the confirmation step of the Submit button is not essential.

#### Handling Missing Input from an HTML Form

When an HTML form is submitted, your stored procedure receives null parameters for any form elements that are not filled in. For example, null parameters can result from an empty entry field, a set of checkboxes, radio buttons, or list items with none checked, or a VALUE parameter of "" (empty quotation marks).

Regardless of any validation you do on the client side, always code stored procedures to handle the possibility that some parameters are null:

* Use a DEFAULT clause in all parameter declarations, to prevent an exception when the stored procedure is called with a missing form parameter. You can set the default to zero for numeric values (when that makes sense), and use DEFAULT NULL when you want to check whether or not the user actually specifies a value.
* Before using an input parameter value that has a DEFAULT NULL declaration, check if it is null.
* Make the procedure generate sensible results even when not all input parameters are specified. You might leave some sections out of a report, or display a text string or image in a report to indicate where parameters were not specified.
* Provide a way to fill in the missing values and run the stored procedure again, directly from the results page. For example, you could include a link that calls the same stored procedure with an additional parameter, or display the original form with its values filled in as part of the output.

#### Maintaining State Information Between Web Pages

Web applications are particularly concerned with the idea of **state**, the set of data that is current at a particular moment in time. It is easy to lose state information when switching from one web page to another, which might result in asking the user to make the same choices over and over.

You can pass state information between dynamic web pages using HTML forms. The information is passed as a set of name-value pairs, which are turned into stored procedure parameters for you.

If the user has to make multiple selections, or one selection from many choices, or it is important to avoid an accidental selection, use an HTML form. After the user makes and reviews all the choices, they confirm the choices with the Submit button. Subsequent pages can use forms with hidden parameters (<INPUT TYPE=HIDDEN> tags) to pass these choices from one page to the next.

If the user is only considering one or two choices, or the decision points are scattered throughout the web page, you can save the user from hunting around for the Submit button by representing actions as hyperlinks and including any necessary name-value pairs in the query string (the part following the ? within a URL).

An alternative way to main state information is to use the Oracle9i Application Server and its mod\_ose module, as described in Oracle Servlet Engine User's Guide. This approach lets you store state information in package variables that remain available as a user moves around a Web site.

# Developing PL/SQL Server Pages (PSP)

This chapter explains how to develop PL/SQL Server Pages (PSP), which let you include dynamic content in web pages.

**Topics:**

* [What Are PL/SQL Server Pages and Why Use Them?](http://docs.oracle.com/cd/E11882_01/appdev.112/e41502/adfns_psp.htm#g2475337)
* [Prerequisites for Developing and Deploying PL/SQL Server Pages](http://docs.oracle.com/cd/E11882_01/appdev.112/e41502/adfns_psp.htm#g2475353)
* [PL/SQL Server Pages and the HTP Package](http://docs.oracle.com/cd/E11882_01/appdev.112/e41502/adfns_psp.htm#g2475369)
* [PL/SQL Server Pages and Other Scripting Solutions](http://docs.oracle.com/cd/E11882_01/appdev.112/e41502/adfns_psp.htm#g2475381)
* [Developing PL/SQL Server Pages](http://docs.oracle.com/cd/E11882_01/appdev.112/e41502/adfns_psp.htm#BHAFACCG)
* [Loading PL/SQL Server Pages into the Database](http://docs.oracle.com/cd/E11882_01/appdev.112/e41502/adfns_psp.htm#BHAIFHAG)
* [Querying PL/SQL Server Page Source Code](http://docs.oracle.com/cd/E11882_01/appdev.112/e41502/adfns_psp.htm#g2478973)
* [Running PL/SQL Server Pages Through URLs](http://docs.oracle.com/cd/E11882_01/appdev.112/e41502/adfns_psp.htm#BHAHIJCH)
* [Examples of PL/SQL Server Pages](http://docs.oracle.com/cd/E11882_01/appdev.112/e41502/adfns_psp.htm#BHAEBFEE)
* [Debugging PL/SQL Server Pages](http://docs.oracle.com/cd/E11882_01/appdev.112/e41502/adfns_psp.htm#BHAFHEHC)
* [Putting PL/SQL Server Pages into Production](http://docs.oracle.com/cd/E11882_01/appdev.112/e41502/adfns_psp.htm#BHAFGEJE)

## What Are PL/SQL Server Pages and Why Use Them?

PL/SQL Server Pages (PSP) are server-side scripts that include dynamic content, including the results of SQL queries, inside web pages. You can author the web pages in an HTML authoring tool and insert blocks of PL/SQL code.

[Example 10-1](http://docs.oracle.com/cd/E11882_01/appdev.112/e41502/adfns_psp.htm#BHADDBII) shows a simple PL/SQL server page called simple.psp.

***Example 10-1 simple.psp***

<%@ page language="PL/SQL" %>

<%@ page contentType="text/html" %>

<%@ plsql procedure="show\_employees" %>

<%-- This example displays the last name and first name of every

employee in the hr.employees table. --%>

<%!

CURSOR emp\_cursor IS

SELECT last\_name, first\_name

FROM hr.employees

ORDER BY last\_name;

%>

<html>

<head>

<meta http-equiv="Content-Type" content="text/html">

<title>List of Employees</title>

</head>

<body TEXT="#000000" BGCOLOR="#FFFFFF">

<h1>List of Employees</h1>

<table width="40%" border="1">

<tr>

<th align="left">Last Name</th>

<th align="left">First Name</th>

</tr>

<% FOR emp\_record IN emp\_cursor LOOP %>

<tr>

<td> <%= emp\_record.last\_name %> </td>

<td> <%= emp\_record.first\_name %> </td>

</tr>

<% END LOOP; %>

</table>

</body>

</html>

You can compile and load a PL/SQL server page into the database with the loadpsp command-line utility. This command loads simple.psp into the hrschema, replacing the show\_employees procedure if it exists:

loadpsp -replace simple.psp

Enter Password: *password*

Browser users can run the show\_employees procedure through a URL. An HTML page that displays the last and first names of employees in the hr.employeestable is returned to the browser through the PL/SQL gateway.

Deploying content through PL/SQL Server Pages has these advantages:

* For developers familiar with PL/SQL, the server pages are the easiest way to create professional web pages that include database-generated content. You can develop web pages as you usually do and then embed PL/SQL code in the HTML.
* PL/SQL Server Pages can be more convenient than using the HTP and HTF packages to write out HTML content line by line.
* Because processing is performed on the database server, the client browser receives a plain HTML page with no special script tags. You can support all browsers and browser levels equally.
* Network traffic is efficient because use of PL/SQL Server Pages minimizes the number of database round-trips.
* You can write content quickly and follow a rapid, iterative development process. You maintain central control of the software, with only a web browser required on the client system.

## Prerequisites for Developing and Deploying PL/SQL Server Pages

To develop and deploy PL/SQL server pages, you must meet these prerequisites:

* To write a PL/SQL server page you need access to a text editor or HTML authoring tool for writing the script. No other development tool is required.
* To load a PL/SQL server page you need:
  + An account on the database in which to load the server pages.
  + Execution rights to the loadpsp command-line utility, which is located in $ORACLE\_HOME/bin.
* To deploy the server pages you must use mod\_plsql. As explained in ["Using mod\_plsql Gateway to Map Client Requests to a PL/SQL Web Application"](http://docs.oracle.com/cd/E11882_01/appdev.112/e41502/adfns_web.htm#CHEIAHAE), the gateway uses the PL/SQL Web Toolkit.

**See Also:**

* ["Using mod\_plsql Gateway to Map Client Requests to a PL/SQL Web Application"](http://docs.oracle.com/cd/E11882_01/appdev.112/e41502/adfns_web.htm#CHEIAHAE)

## 

## PL/SQL Server Pages and the HTP Package

You can enable browser users to run PL/SQL units through HTTP in these ways:

* By writing an HTML page with embedded PL/SQL code and compiling it as a PL/SQL server page. You might invoke subprograms from the PL/SQL Web Toolkit, but not to generate every line of HTML output.
* By writing a complete stored subprogram that produces HTML by invoking the HTP and OWA\_\* packages in the PL/SQL Web Toolkit. For information about this technique, see ["Generating HTML Output with PL/SQL"](http://docs.oracle.com/cd/E11882_01/appdev.112/e41502/adfns_web.htm#i1006207).

Thus, you must choose which technique to use when writing your web application. The key factors in choosing between these techniques are:

* What source are you using as a starting point?
  + If you have a large body of HTML, and want to include dynamic content or make it the front end of a database application, then use PL/SQL Server Pages.
  + If you have a large body of PL/SQL code that produces formatted output, then you might find it more convenient to produce HTML tags by changing your print statements to invoke the HTP package of the PL/SQL Web Toolkit.
* What is the fastest and most convenient authoring environment for your group?
  + If most work is done using HTML authoring tools, then use PL/SQL Server Pages.
  + If you use authoring tools that produce PL/SQL code, then it might be less convenient to use PL/SQL Server Pages.

## PL/SQL Server Pages and Other Scripting Solutions

Scripting solutions can be client-side or server-side. JavaScript is a very popular client-side scripting languages. PL/SQL Server Pages fully support JavaScript. Because any kind of tags can be passed unchanged to the browser through a PL/SQL server page, you can include JavaScript or other client-side script code in a PL/SQL server page.

Java Server Pages (JSP) and Active Server Pages (ASP) are two of the most popular server-side scripting solutions. Compared to PL/SQL Server Pages:

* Java server pages are loosely analogous to PL/SQL Server Pages pages; Java servlets are analogous to PL/SQL packages. PL/SQL Server Pages use the same script tag syntax as JSP to make it easy to switch back and forth.
* PL/SQL Server Pages use syntax that is similar to ASP, although not identical. Typically, you must translate from VBScript or JScript to PL/SQL. The best candidates for migration are pages that use the Active Data Object (ADO) interface to perform database operations.

**Note:**

You cannot mix PL/SQL server pages with other server-side script features, such as server-side includes. Often, you can get the same results by using the corresponding PL/SQL Server Pages features.

## Developing/Writing PL/SQL Server Pages

To develop a PL/SQL server page, you can start with an existing web page or with an existing stored subprogram. Either way, with a few additions and changes you can create dynamic web pages that perform database operations and display the results.

The file for a PL/SQL server page must have the extension .psp. It can contain whatever content you choose, with text and tags interspersed with PL/SQL Server Pages directives, declarations, and scriptlets. A server page can take these forms:

* In the simplest case, it is an HTML file. Compiling it as a PL/SQL server page produces a stored subprogram that outputs the same HTML file.
* In the most complex case, it is a PL/SQL subprogram that generates all the content of the web page, including the tags for title, body, and headings.
* In the typical case, it is a mixture of HTML (providing the static parts of the page) and PL/SQL (providing the dynamic content).

The order and placement of the PL/SQL Server Pages directives and declarations is usually not significant. It becomes significant only when another file is included. For ease of maintenance, Oracle recommends that you put the directives and declarations near the beginning of the file.

[Table 10-1](http://docs.oracle.com/cd/E11882_01/appdev.112/e41502/adfns_psp.htm#BHAGCCAJ) lists the PL/SQL Server Pages elements and directs you to the section that explains how to use them. The section ["Using Quotation Marks and Escaping Strings in a PSP Script"](http://docs.oracle.com/cd/E11882_01/appdev.112/e41502/adfns_psp.htm#BHACAEGJ) describes how to use quotation marks in strings that are used in various PL/SQL Server Pages elements.

***Table 10-1 PSP Elements***

| **PSP Element** | **Name** | **Specifies . . .** | **Section** |
| --- | --- | --- | --- |
| <%@ page ... %> | Page Directive | Characteristics of the PL/SQL server page. | ["Specifying Basic Server Page Characteristics"](http://docs.oracle.com/cd/E11882_01/appdev.112/e41502/adfns_psp.htm#BHACJDBE) |
| <%@ parameter ... %> | Parameter Directive | The name, and optionally the type and default, for each parameter expected by the PSP stored procedure. | ["Accepting User Input"](http://docs.oracle.com/cd/E11882_01/appdev.112/e41502/adfns_psp.htm#BHAGEBDA) |
| <%@ plsql ... %> | Procedure Directive | The name of the stored procedure produced by the PSP file. | ["Naming the PL/SQL Stored Procedure"](http://docs.oracle.com/cd/E11882_01/appdev.112/e41502/adfns_psp.htm#BHAGDGBA) |
| <%@ include ... %> | Include Directive | The name of a file to be included at a specific point in the PSP file. | ["Including the Contents of Other Files"](http://docs.oracle.com/cd/E11882_01/appdev.112/e41502/adfns_psp.htm#BHAEFHGG) |
| <%! ... %> | Declaration Block | The declaration for a set of PL/SQL variables that are visible throughout the page, not just within the next BEGIN/END block. | ["Declaring Global Variables in a PSP Script"](http://docs.oracle.com/cd/E11882_01/appdev.112/e41502/adfns_psp.htm#BHAIABJJ) |
| <% ... %> | Code Block | A set of PL/SQL statements to be executed when the procedure is run. | ["Specifying Executable Statements in a PSP Script"](http://docs.oracle.com/cd/E11882_01/appdev.112/e41502/adfns_psp.htm#BHAFIBIA) |
| <%= ... %> | Expression Block | A single PL/SQL expression, such as a string, arithmetic expression, function call, or combination of these. | ["Substituting Expression Values in a PSP Script"](http://docs.oracle.com/cd/E11882_01/appdev.112/e41502/adfns_psp.htm#BHAIEHEE) |
| <%-- ... --%> | Comment | A comment in a PSP script. | ["Including Comments in a PSP Script"](http://docs.oracle.com/cd/E11882_01/appdev.112/e41502/adfns_psp.htm#BHAIEJIF) |

**Note:**

If you are familiar with dynamic HTML, you can go directly to ["Examples of PL/SQL Server Pages"](http://docs.oracle.com/cd/E11882_01/appdev.112/e41502/adfns_psp.htm#BHAEBFEE).

**Topics:**

* [Specifying Basic Server Page Characteristics](http://docs.oracle.com/cd/E11882_01/appdev.112/e41502/adfns_psp.htm#BHACJDBE)
* [Accepting User Input](http://docs.oracle.com/cd/E11882_01/appdev.112/e41502/adfns_psp.htm#BHAGEBDA)
* [Naming the PL/SQL Stored Procedure](http://docs.oracle.com/cd/E11882_01/appdev.112/e41502/adfns_psp.htm#BHAGDGBA)
* [Including the Contents of Other Files](http://docs.oracle.com/cd/E11882_01/appdev.112/e41502/adfns_psp.htm#BHAEFHGG)
* [Declaring Global Variables in a PSP Script](http://docs.oracle.com/cd/E11882_01/appdev.112/e41502/adfns_psp.htm#BHAIABJJ)
* [Specifying Executable Statements in a PSP Script](http://docs.oracle.com/cd/E11882_01/appdev.112/e41502/adfns_psp.htm#BHAFIBIA)
* [Substituting Expression Values in a PSP Script](http://docs.oracle.com/cd/E11882_01/appdev.112/e41502/adfns_psp.htm#BHAIEHEE)
* [Using Quotation Marks and Escaping Strings in a PSP Script](http://docs.oracle.com/cd/E11882_01/appdev.112/e41502/adfns_psp.htm#BHACAEGJ)
* [Including Comments in a PSP Script](http://docs.oracle.com/cd/E11882_01/appdev.112/e41502/adfns_psp.htm#BHAIEJIF)

### Specifying Basic Server Page Characteristics

Use the <%@ page ... %> directive to specify characteristics of the PL/SQL server page such as:

* What scripting language it uses.
* What type of information (MIME type) it produces.
* What code to run to handle all uncaught exceptions. This might be an HTML file with a friendly message, renamed to a .psp file. You must specify this same file name in the loadpsp command that compiles the main PSP file. You must specify the same name in both the errorPage directive and in theloadpsp command, including any relative path name such as ../include/.

This code shows the syntax of the page directive (the attribute names contentType and errorPage are case-sensitive):

<%@ page

language='PL/SQL'

contentType='*content\_type\_string*'

charset='*encoding*'

errorPage='*file*.psp'

%>

**Topics:**

* [Specifying the Scripting Language](http://docs.oracle.com/cd/E11882_01/appdev.112/e41502/adfns_psp.htm#BHAIEFHD)
* [Returning Data to the Client Browser](http://docs.oracle.com/cd/E11882_01/appdev.112/e41502/adfns_psp.htm#BHAGHFED)
* [Handling Script Errors](http://docs.oracle.com/cd/E11882_01/appdev.112/e41502/adfns_psp.htm#BHAEEBEC)

#### Specifying the Scripting Language

To identify a file as a PL/SQL server page, include this directive somewhere in the file:

<%@ page language="PL/SQL" %>

This directive is for compatibility with other scripting environments. [Example 10-1](http://docs.oracle.com/cd/E11882_01/appdev.112/e41502/adfns_psp.htm#BHADDBII) shows an example of a simple PL/SQL server page that includes the language directive.

#### Returning Data to the Client Browser

**Options:**

* [Returning HTML](http://docs.oracle.com/cd/E11882_01/appdev.112/e41502/adfns_psp.htm#BHAIDFCI)
* [Returning XML, Text, and Other Document Types](http://docs.oracle.com/cd/E11882_01/appdev.112/e41502/adfns_psp.htm#BHAIIBFJ)
* [Returning Pages Containing Different Character Sets](http://docs.oracle.com/cd/E11882_01/appdev.112/e41502/adfns_psp.htm#BHACJICD)

##### Returning HTML

The PL/SQL parts of a PL/SQL server page are enclosed within special delimiters. All other content is passed exactly as it is—including any white space—to the browser. To display text or HTML tags, write it as you would write a typical web page. You need not invoke any output functions. As illustration, the server page in [Example 10-1](http://docs.oracle.com/cd/E11882_01/appdev.112/e41502/adfns_psp.htm#BHADDBII) returns the HTML page shown in [Example 10-2](http://docs.oracle.com/cd/E11882_01/appdev.112/e41502/adfns_psp.htm#BHAECBCF), except that it includes the table rows for the queried employees.

***Example 10-2 Sample Returned HTML Page***

<html>

<head>

<meta http-equiv="Content-Type" content="text/html">

<title>List of Employees</title>

</head>

<body TEXT="#000000" BGCOLOR="#FFFFFF">

<h1>List of Employees</h1>

<table width="40%" border="1">

<tr>

<th align="left">Last Name</th>

<th align="left">First Name</th>

</tr>

<!-- result set of query of hr.employees inserted here -->

</table>

</body>

</html>

Sometimes you might want to display one line of output or another, or change the value of an attribute, based on a condition. You can include control structures and variable substitution inside the PSP delimiters, as shown in this code fragment from [Example 10-1](http://docs.oracle.com/cd/E11882_01/appdev.112/e41502/adfns_psp.htm#BHADDBII):

<% FOR emp\_record IN emp\_cursor LOOP %>

<tr>

<td> <%= emp\_record.last\_name %> </td>

<td> <%= emp\_record.first\_name %> </td>

</tr>

<% END LOOP; %>

##### Returning XML, Text, and Other Document Types

By default, the PL/SQL gateway transmits files as HTML documents so that the browser interprets the HTML tags. If you want the browser to interpret the document as XML, plain text (with no formatting), or some other document type, then include this directive:

<%@ page contentType="*MIMEtype*" %>

The attribute name contentType is case-sensitive. Insert text/html, text/xml, text/plain, image/jpeg, or some other MIME type that the browser or other client program recognizes. Users might have to configure their browsers to recognize some MIME types. An example of a directive for an Excel spreadsheet is:

<%@ page contentType="application/vnd.ms-excel" %>

Typically, a PL/SQL server page is intended to be displayed in a web browser. It can also be retrieved and interpreted by a program that can make HTTP requests, such as a a Java or PERL client.

##### Returning Pages Containing Different Character Sets

By default, the PL/SQL gateway transmits files with the character set defined by the PL/SQL gateway. To convert the data to a different character set for browser display, include this directive:

<%@ page charset="encoding" %>

Specify Shift\_JIS, Big5, UTF-8, or another encoding that the client program recognizes.

You must also configure the character set setting in the database accessor descriptor (DAD) of the PL/SQL gateway. Users might have to select the same encoding in their browsers to see the data displayed properly. For example, a database in Japan might have a database character set that uses the EUCencoding, but the web browsers are configured to display Shift\_JIS encoding.

#### Handling Script Errors

When writing PL/SQL server pages, you can get these types of errors:

* HTML syntax errors

The browser handles these errors. The loadpsp utility does not check for them.

* PL/SQL syntax errors

The loadpsp utility stops and displays the line number, column number, and a brief message. You must fix the error before continuing.

Any previous version of the stored subprogram can be erased when you attempt to replace it with a script that contains a syntax error. You might want to use one database for prototyping and debugging, and then load the final stored subprogram into a different database for production. You can switch databases using a command-line flag without changing any source code.

* Runtime errors

To handle database errors that occur when the script runs, you can include PL/SQL exception-handling code within a PSP file and have any unhandled exceptions start a special PL/SQL server page. Use the errorPage attribute (the name is case-sensitive) of the <%@ page ... %> directive to specify the page name.

The page for unhandled exceptions is a PL/SQL server page with extension .psp. The error subprogram does not receive any parameters, so to determine the cause of the error, it can invoke the SQLCODE and SQLERRM functions. You can also display a standard HTML page without any scripting when an error occurs, but you must still give it the extension .psp and load it into the database as a stored subprogram.

This line specifies errors.psp as the page to run when errors are encountered:

<%@ page language="PL/SQL" contentType="text/html" **errorPage="errors.psp"** %>

### Accepting User Input

To set up parameter passing for a PL/SQL server page, include a directive with this syntax:

<%@ plsql parameter="*parameter\_name*" [type="*PL/SQL\_type*"] [default="*value*"] %>

The default *PL/SQL\_type* is VARCHAR2. This directive specifies that the parameter p\_employee\_id is of the type NUMBER:

<%@ plsql parameter="p\_employee\_id" type="NUMBER" %>

Specifying a default value for a parameter makes the parameter optional. The default value is substituted directly into a PL/SQL statement, so any strings must be enclosed in single quotation marks, and you can use special values such as NULL. This directive specifies that the parameter p\_last\_name has the default value NULL:

<%@ plsql parameter="p\_last\_name" default="NULL" %>

User input comes encoded in the URL that retrieves the HTML page. You can generate the URL by hard-coding it in an HTML link, or by invoking your page as the action of an HTML form. Your page receives the input as parameters to a PL/SQL stored subprogram.

[Example 10-3](http://docs.oracle.com/cd/E11882_01/appdev.112/e41502/adfns_psp.htm#BHACEBEE) is like [Example 10-1](http://docs.oracle.com/cd/E11882_01/appdev.112/e41502/adfns_psp.htm#BHADDBII), except that it uses a parameter, p\_employee\_id. If the PL/SQL gateway is configured so that you can run procedures by invoking http://www.host.com/pls/*proc\_name*, where *proc\_name* is the name of a procedure, then you can pass 200 for parameter p\_employee\_id as follows:

http://www.example.com/pls/show\_employees?p\_employee\_id=200

***Example 10-3 simplewithuserinput.psp***

<%@ page language="PL/SQL" %>

<%@ page contentType="text/html" %>

**<%@ plsql parameter="p\_employee\_id" default="null" type="NUMBER" %>**

<%@ plsql procedure="show\_employees" %>

<%-- This example displays the last name and first name of every

employee in the hr.employees table. --%>

<%!

CURSOR emp\_cursor IS

SELECT last\_name, first\_name

FROM hr.employees

**WHERE employee\_id = p\_employee\_id**

ORDER BY last\_name;

%>

<html>

<head>

<meta http-equiv="Content-Type" content="text/html">

<title>List of Employees</title>

</head>

<body TEXT="#000000" BGCOLOR="#FFFFFF">

<h1>List of Employees</h1>

<table width="40%" border="1">

<tr>

<th align="left">Last Name</th>

<th align="left">First Name</th>

</tr>

<% FOR emp\_record IN emp\_cursor LOOP %>

<tr>

<td> <%= emp\_record.last\_name %> </td>

<td> <%= emp\_record.first\_name %> </td>

</tr>

<% END LOOP; %>

</table>

</body>

</html>

### Naming the PL/SQL Stored Procedure

Each top-level PL/SQL server page corresponds to a stored procedure within the server. When you load the page with loadpsp, the utility creates a PL/SQL stored procedure. If the server page is *name*.psp, the default procedure name is *name*. For example, if the server page is hello\_world.psp, then the default procedure name is hello\_world.

To specify a procedure name, use this directive, where *procname* is the name for the procedure:

<%@ plsql procedure="*procname*" %>

In [Example 10-1](http://docs.oracle.com/cd/E11882_01/appdev.112/e41502/adfns_psp.htm#BHADDBII), this directive gives the stored procedure the name show\_employees:

<%@ plsql procedure="show\_employees" %>

It is the name of the procedure, not the name of the PSP script, that you include in the URL.

### Including the Contents of Other Files

You can set up an include mechanism to pull in the contents of other files, typically containing either static HTML content or more PL/SQL scripting code. Insert this directive at the point where the content of the other file is to appear, replacing *filename* with the name of the file to be included:

<%@ include file="*filename*" %>

The included file must have an extension other than .psp. You must specify the same name in both the include directive and in the loadpsp command, including any relative path name such as ../include/.

Because the files are processed when you load the stored procedure into the database, the substitution is performed only once, not whenever the page is served. Therefore, changes to the included files that occur after the page is loaded into the database are not displayed when the procedure is executed.

You can use the include feature to pull in libraries of code, such as a navigation banners, footers, tables of contents, and so forth into multiple files. Alternatively, you can use this feature as a macro capability to include the same section of script code in multiple places in a page. This example includes an HTML footer:

<%@ include file="footer.htm" %>

When you use included files:

* You can use any names and extensions for the included files. For example, you can include a file called products.txt.
* If the included files contain PL/SQL scripting code, then they do not need their own set of directives to identify the procedure name, character set, and so on.
* When specifying the names of files to the loadpsp utility, you must include the names of all included files also. Specify the names of included files before the names of any .psp files.

### Declaring Global Variables in a PSP Script

You can use the <%! ... %> directive to define a set of PL/SQL variables that are visible throughout the page, not just within the next BEGIN/END block. This element typically spans multiple lines, with individual PL/SQL variable declarations ended by semicolons. The syntax for this directive is as follows:

<%! *PL/SQL declaration;*

[ *PL/SQL declaration;* ] ... %>

The usual PL/SQL syntax is allowed within the block. The delimiters server as shorthand, enabling you to omit the DECLARE keyword. All declarations are available to the code later in the file. [Example 10-1](http://docs.oracle.com/cd/E11882_01/appdev.112/e41502/adfns_psp.htm#BHADDBII) includes this cursor declaration:

<%!

CURSOR emp\_cursor IS

SELECT last\_name, first\_name

FROM hr.employees

ORDER BY last\_name;

%>

You can specify multiple declaration blocks; internally, they are all merged into a single block when the PSP file is created as a stored procedure.

You can also use explicit DECLARE blocks within the <% ... %> delimiters that are explained in ["Specifying Executable Statements in a PSP Script"](http://docs.oracle.com/cd/E11882_01/appdev.112/e41502/adfns_psp.htm#BHAFIBIA). These declarations are only visible to the BEGIN/END block that follows them.

**Note:**

To make things easier to maintain, keep all your directives and declarations near the beginning of a PL/SQL server page.

### Specifying Executable Statements in a PSP Script

You can use the <% ... %> code block directive to run a set of PL/SQL statements when the stored procedure is run. This code shows the syntax for executable statements:

<% *PL/SQL statement;*

[ *PL/SQL statement;* ] ... %>

This element typically spans multiple lines, with individual PL/SQL statements ended by semicolons. The statements can include complete blocks, as in this example, which invokes the OWA\_UTIL.TABLEPRINT procedure:

<% OWA\_UTIL.TABLEPRINT(CTABLE => 'hr.employees', CATTRIBUTES => 'border=2',

CCOLUMNS => 'last\_name,first\_name', CCLAUSES => 'WHERE employee\_id > 100'); %>

The statements can also be the bracketing parts of IF/THEN/ELSE or BEGIN/END blocks. When a code block is split into multiple directives, you can put HTML or other directives in the middle, and the middle pieces are conditionally executed when the stored procedure is run. This code from [Example 10-11](http://docs.oracle.com/cd/E11882_01/appdev.112/e41502/adfns_psp.htm#BHAECEIB) provides an illustration of this technique:

<% FOR ITEM IN (SELECT product\_name, list\_price, catalog\_url

FROM product\_information

WHERE list\_price IS NOT NULL

ORDER BY list\_price DESC) LOOP

IF item.list\_price > p\_minprice THEN

v\_color := '#CCCCFF';

ELSE

v\_color := '#CCCCCC';

END IF;

%>

<TR BGCOLOR="<%= v\_color %>">

<TD><A HREF="<%= item.catalog\_url %>"><%= item.product\_name %></A></TD>

<TD><BIG><%= item.list\_price %></BIG></TD>

</TR>

<% END LOOP; %>

All the usual PL/SQL syntax is allowed within the block. The delimiters server as shorthand, letting you omit the DECLARE keyword. All the declarations are available to the code later on in the file.

**Note:**

To share procedures, constants, and types across different PL/SQL server pages, compile them into a package in the database by using a plain PL/SQL source file. Although you can reference package procedures, constants, and types from PSP scripts, the PSP scripts can only produce standalone procedures, not packages.

### Substituting Expression Values in a PSP Script

An expression directive outputs a single PL/SQL expression, such as a string, arithmetic expression, function call, or combination of these things. The result is substituted as a string at that spot in the HTML page that is produced by the stored procedure. The expression result must be a string value or be able to be cast to a string. For any types that cannot be implicitly cast, such as DATE, pass the value to the PL/SQL TO\_CHAR function.

The syntax of an expression directive is as follows, where the *expression* placeholder is replaced by the desired expression:

<%= *expression* %>

You need not end the PL/SQL expression with a semicolon.

[Example 10-1](http://docs.oracle.com/cd/E11882_01/appdev.112/e41502/adfns_psp.htm#BHADDBII) includes a directive to print the value of a variable in a row of a cursor:

<%= emp\_record.last\_name %>

Compare the preceding example to the equivalent htp.print call in this example (note especially the semicolon that ends the statement):

<% HTP.PRN (emp\_record.last\_name); %>

The content within the <%= ... %> delimiters is processed by the HTP.PRN function, which trims leading or trailing white space and requires that you enclose literal strings in single quotation marks.

You can use concatenation by using the twin pipe symbol (||) as in PL/SQL. This directive shows an example of concatenation:

<%= 'The employee last name is ' || emp\_record.last\_name %>

### Using Quotation Marks and Escaping Strings in a PSP Script

PSP attributes use double quotation marks to delimit data. When values specified in PSP attributes are used for PL/SQL operations, they are passed exactly as you specify them in the PSP file. Thus, if PL/SQL requires a string enclosed in single quotation marks, then you must specify the string enclosed in single quotation marks, and enclose the whole thing in double quotation marks.

For example, your PL/SQL procedure might use the string Babe Ruth as the default value for a variable. For the string to be used in PL/SQL, you must enclose it in single quotation marks as 'Babe Ruth'. If you specify this string in the default attribute of a PSP directive, you must enclose it in double quotation marks, like this:

<%@ plsql parameter="in\_players" default="'Babe Ruth'" %>

You can also enclose strings that are enclosed in single quotation marks in another set of single quotation marks. In this case, you must escape the inner single quotation marks by specifying the sequence \'. For example:

<%@ plsql parameter="in\_players" default="'Walter \'Big Train\' Johnson'" %>

You can include most characters and character sequences in a PSP file without having them changed by the PSP loader. To include the sequence %>, specify the escape sequence %\>. To include the sequence <%, specify the escape sequence <\%. For example:

<%= 'The %\> sequence is used in scripting language: ' || lang\_name %>

<%= 'The <\% sequence is used in scripting language: ' || lang\_name %>

### Including Comments in a PSP Script

To put a comment in the HTML portion of a PL/SQL server page for the benefit of those reading the PSP source code, use this syntax:

<%-- *PSP comment text* --%>

Comments in the preceding form do not appear in the HTML output from the PSP and also do not appear when you query the PL/SQL source code inUSER\_OBJECTS.

To create a comment that is visible in the HTML output and in the USER\_OBJECTS source, place the comment in the HTML and use the normal HTML comment syntax:

<!-- *HTML comment text* -->

To include a comment inside a PL/SQL block within a PSP, and to make the comment invisible in the HTML output but visible in USER\_OBJECTS, use the normal PL/SQL comment syntax, as in this example:

-- Comment in PL/SQL code

[Example 10-4](http://docs.oracle.com/cd/E11882_01/appdev.112/e41502/adfns_psp.htm#BHADEJAH) shows a fragment of a PSP file with the three types of comments.

***Example 10-4 Sample Comments in a PSP File***

<p>Today we introduce our new model XP-10.

**<%--**

This is the project with code name "Secret Project".

Users viewing the HTML page do not see this PSP script comment.

The comment is not visible in the USER\_OBJECTS source code.

**--%>**

**<!--**

Some pictures of the XP-10.

Users viewing the HTML page source see this comment.

The comment is also visible in the USER\_OBJECTS source code.

**-->**

<%

FOR image\_file IN (SELECT pathname, width, height, description

FROM image\_library WHERE model\_num = 'XP-10')

**--** Comments interspersed with PL/SQL statements.

**--** Users viewing the HTML page source do not see these PL/SQL comments.

**--** These comments are visible in the USER\_OBJECTS source code.

LOOP

%>

<img src="<%= image\_file.pathname %>" width=<% image\_file.width %>

height=<% image\_file.height %> alt="<% image\_file.description %>">

<br>

<% END LOOP; %>

## Loading PL/SQL Server Pages into the Database

Use the loadpsp utility, which is located in $ORACLE\_HOME/bin, to load one or more PSP files into the database as stored procedures. Each .psp file corresponds to one stored procedure. The pages are compiled and loaded in one step, to speed up the development cycle. The syntax of the loadpsp utility is:

loadpsp [-replace] [*include\_file\_name*...] [*error\_file\_name*] *psp\_file\_name*...

Enter Password: *password*

When you load a PSP file, the loader performs these actions:

1. Logs on to the database with the specified user name, password, and net service name
2. Creates the stored procedures in the user schema

-replace creates procedures with CREATE OR REPLACE syntax.

*include\_file\_name* is the name of a file that is specified in the PSP include directive.

*error\_file\_name* is the name of the file that is specified in the errorPage attribute of the PSP page directive.

*psp\_file\_name* is the name of a file that is specified in a PSP page directive.

The filenames on the loadpsp command line must exactly match the names specified in the PSP include and page directives, including any relative path name such as ../include/.

[Example 10-5](http://docs.oracle.com/cd/E11882_01/appdev.112/e41502/adfns_psp.htm#BHAJGEHF) shows a sample PSP load command.

***Example 10-5 Loading PL/SQL Server Pages***

loadpsp -replace -user joe/abc123@/db3 banner.inc error.psp display\_order.psp

In [Example 10-5](http://docs.oracle.com/cd/E11882_01/appdev.112/e41502/adfns_psp.htm#BHAJGEHF):

* The stored procedure is created in the database db3. The database is accessed as user joe with password abc123, both to create the stored procedure and when the stored procedure is executed.
* banner.inc is a file containing boilerplate text and script code that is included by the .psp file. The inclusion occurs when the PSP is loaded into the database, not when the stored procedure is executed.
* error.psp is a file containing code, text, or both that is processed when an unhandled exception occurs, to present a friendly page rather than an internal error message.
* display\_order.psp contains the main code and text for the web page. By default, the corresponding stored procedure is named display\_order.

## Querying PL/SQL Server Page Source Code

The code that loadpsp generates is different from the code in the source file. It has calls to the HTP package, which generates the HTML tags for the web page.

After loading a PSP file, you can see the generated source code by querying the static data dictionary views \*\_SOURCE. For example, suppose that you load the script in [Example 10-1](http://docs.oracle.com/cd/E11882_01/appdev.112/e41502/adfns_psp.htm#BHADDBII) with this command:

loadpsp -replace -user hr simple.psp

Enter Password: *password*

If you log on to the database as user hr, you can view the source code of the PSP as shown in [Example 10-6](http://docs.oracle.com/cd/E11882_01/appdev.112/e41502/adfns_psp.htm#BHAGDCCH).

***Example 10-6 Querying PL/SQL Server Page Source Code***

Query:

SELECT TEXT

FROM USER\_SOURCE

WHERE NAME = 'SHOW\_EMPLOYEES'

ORDER BY LINE;

Result:

PROCEDURE show\_employees AS

CURSOR emp\_cursor IS

SELECT last\_name, first\_name

FROM hr.employees

ORDER BY last\_name;

BEGIN NULL;

owa\_util.mime\_header('text/html'); htp.prn('

');

htp.prn('

');

htp.prn('

');

htp.prn('

');

htp.prn('

');

htp.prn('

<html>

<head>

<meta http-equiv="Content-Type" content="text/html">

<title>List of Employees</title>

</head>

<body TEXT="#000000" BGCOLOR="#FFFFFF">

<h1>List of Employees</h1>

<table width="40%" border="1">

<tr>

<th align="left">Last Name</th>

<th align="left">First Name</th>

</tr>

');

FOR emp\_record IN emp\_cursor LOOP

htp.prn('

<tr>

<td> ');

htp.prn( emp\_record.last\_name );

htp.prn(' </td>

<td> ');

htp.prn( emp\_record.first\_name );

htp.prn(' </td>

</tr>

');

END LOOP;

htp.prn('

</table>

</body>

</html>

');

END;

## Running PL/SQL Server Pages Through URLs

After the PL/SQL server page is turned into a stored procedure, you can run the procedure by retrieving an HTTP URL through a web browser or other Internet-aware client program. The virtual path in the URL depends on the way the PL/SQL gateway is configured.

The parameters to the stored procedure are passed through either the POST method or the GET method of the HTTP protocol. With the POST method, the parameters are passed directly from an HTML form and are not visible in the URL. With the GET method, the parameters are passed as name-value pairs in the query string of the URL, separated by & characters, with most nonalphanumeric characters in encoded format (such as %20 for a space). You can use theGET method to invoke a PSP page from an HTML form, or you can use a hard-coded HTML link to invoke the stored procedure with a given set of parameters.

Using METHOD=GET, the syntax of the URL looks something like this:

http://*sitename*/*schemaname*/*procname*?*parmname1*=*value1*&*parmname2*=*value2*

For example, this URL includes a p\_lname and p\_fname parameter:

http://www.example.com/pls/show\_employees?p\_lname=Ashdown&p\_fname=Lance

Using METHOD=POST, the syntax of the URL does not show the parameters:

http://*sitename*/*schemaname*/*procname*

For example, this URL specifies a procedure name but does not pass parameters:

http://www.example.com/pls/show\_employees

The METHOD=GET format is more convenient for debugging and allows visitors to pass the same parameters when they return to the page through a bookmark.

The METHOD=POST format allows a larger volume of parameter data, and is suitable for passing sensitive information that must not be displayed in the URL. (URLs linger on in the browser's history list and in the HTTP headers that are passed to the next-visited page.) It is not practical to bookmark pages that are invoked this way.

## Examples of PL/SQL Server Pages

This section shows how you might start with a very simple PL/SQL server page, and produce progressively more complicated versions as you gain more confidence.

As you go through each step, you can follow the instructions in ["Loading PL/SQL Server Pages into the Database"](http://docs.oracle.com/cd/E11882_01/appdev.112/e41502/adfns_psp.htm#BHAIFHAG) and ["Running PL/SQL Server Pages Through URLs"](http://docs.oracle.com/cd/E11882_01/appdev.112/e41502/adfns_psp.htm#BHAHIJCH) to test the examples.

**Topics:**

* [Setup for PL/SQL Server Pages Examples](http://docs.oracle.com/cd/E11882_01/appdev.112/e41502/adfns_psp.htm#BHACCEDE)
* [Printing the Sample Table with a Loop](http://docs.oracle.com/cd/E11882_01/appdev.112/e41502/adfns_psp.htm#BHACBCFF)
* [Allowing a User Selection](http://docs.oracle.com/cd/E11882_01/appdev.112/e41502/adfns_psp.htm#BHAEHEIC)
* [Using an HTML Form to Invoke a PL/SQL Server Page](http://docs.oracle.com/cd/E11882_01/appdev.112/e41502/adfns_psp.htm#BHAFDADH)
* [Including JavaScript in a PSP File](http://docs.oracle.com/cd/E11882_01/appdev.112/e41502/adfns_psp.htm#g2479352)

### Setup for PL/SQL Server Pages Examples

These examples use the PRODUCT\_INFORMATION table in the OE schema, which is described as follows:

SQL\*Plus command:

DESCRIBE PRODUCT\_INFORMATION;

Result:

Name Null? Type

----------------------------------------- -------- ----------------------------

PRODUCT\_ID NOT NULL NUMBER(6)

PRODUCT\_NAME VARCHAR2(50)

PRODUCT\_DESCRIPTION VARCHAR2(2000)

CATEGORY\_ID NUMBER(2)

WEIGHT\_CLASS NUMBER(1)

WARRANTY\_PERIOD INTERVAL YEAR(2) TO MONTH

SUPPLIER\_ID NUMBER(6)

PRODUCT\_STATUS VARCHAR2(20)

LIST\_PRICE NUMBER(8,2)

MIN\_PRICE NUMBER(8,2)

CATALOG\_URL VARCHAR2(50)

The examples assume:

* You have set up mod\_plsql as described in ["Using mod\_plsql Gateway to Map Client Requests to a PL/SQL Web Application"](http://docs.oracle.com/cd/E11882_01/appdev.112/e41502/adfns_web.htm#CHEIAHAE).
* You have created a DAD for static authentication of the OE user.
* You can access PL/SQL stored procedures created in the OE schema through this URL, where proc\_name is the name of a stored procedure:http://www.example.com/pls/proc\_name

For debugging purposes, you can display the complete contents of a SQL table with a call to OWA\_UTIL.TABLEPRINT, as in [Example 10-7](http://docs.oracle.com/cd/E11882_01/appdev.112/e41502/adfns_psp.htm#BHAJFGJA). Later examples show other techniques that give more control over the presentation.

***Example 10-7 show\_prod\_simple.psp***

<%@ plsql procedure="show\_prod\_simple" %>

<HTML>

<HEAD><TITLE>Show Contents of product\_information (Complete Dump)</TITLE></HEAD>

<BODY>

<%

DECLARE

dummy BOOLEAN;

BEGIN

dummy := OWA\_UTIL.TABLEPRINT('oe.product\_information','border');

END;

%>

</BODY>

</HTML>

Load the PSP in [Example 10-7](http://docs.oracle.com/cd/E11882_01/appdev.112/e41502/adfns_psp.htm#BHAJFGJA) at the command line as follows:

loadpsp -replace -user oe/*password* show\_prod\_simple.psp

Enter Password: *password*

Access the PSP through this URL:

http://www.example.com/pls/show\_prod\_simple

### Printing the Sample Table with a Loop

[Example 10-7](http://docs.oracle.com/cd/E11882_01/appdev.112/e41502/adfns_psp.htm#BHAJFGJA) loops through the items in the product\_information table and adjusts the SELECT statement to retrieve only a subset of the rows or columns. This example uses a very simple presentation, a set of list items, to avoid any problems from mismatched or unclosed table tags.

***Example 10-8 show\_catalog\_raw.psp***

<%@ plsql procedure="show\_prod\_raw" %>

<HTML>

<HEAD><TITLE>Show Products (Raw Form)</TITLE></HEAD>

<BODY>

<UL>

<% FOR item IN (SELECT product\_name, list\_price, catalog\_url

FROM product\_information

WHERE list\_price IS NOT NULL

ORDER BY list\_price DESC) LOOP %>

<LI>

Item = <%= item.product\_name %><BR>

Price = <%= item.list\_price %><BR>

URL = <%= item.catalog\_url %><BR>

<% END LOOP; %>

</UL>

</BODY>

</HTML>

[Example 10-9](http://docs.oracle.com/cd/E11882_01/appdev.112/e41502/adfns_psp.htm#BHAJBEEB) shows a more sophisticated variation of [Example 10-8](http://docs.oracle.com/cd/E11882_01/appdev.112/e41502/adfns_psp.htm#BHACCAAA) in which formatting is added to the HTML to improve the presentation.

***Example 10-9 show\_catalog\_pretty.psp***

<%@ plsql procedure="show\_prod\_pretty" %>

<HTML>

<HEAD><TITLE>Show Products (Better Form)</TITLE></HEAD>

<BODY>

<UL>

<% FOR item IN (SELECT product\_name, list\_price, catalog\_url

FROM product\_information

WHERE list\_price IS NOT NULL

ORDER BY list\_price DESC) LOOP %>

<LI>

Item = <A HREF=<%= item.catalog\_url %>><%= item.product\_name %></A><BR>

Price = <BIG><%= item.list\_price %></BIG><BR>

<% END LOOP; %>

</UL>

</BODY>

</HTML>

### Allowing a User Selection

In [Example 10-7](http://docs.oracle.com/cd/E11882_01/appdev.112/e41502/adfns_psp.htm#BHAJFGJA), [Example 10-8](http://docs.oracle.com/cd/E11882_01/appdev.112/e41502/adfns_psp.htm#BHACCAAA), and [Example 10-9](http://docs.oracle.com/cd/E11882_01/appdev.112/e41502/adfns_psp.htm#BHAJBEEB), the HTML page remains the same unless the PRODUCT\_INFORMATION table is updated. [Example 10-10](http://docs.oracle.com/cd/E11882_01/appdev.112/e41502/adfns_psp.htm#BHAHGAAF):

* Makes the HTML page accept a minimum price, and presents only the items that are more expensive. (Your customers' buying criteria might vary.)
* Sets the default minimum price to 100 units of the appropriate currency.

***Example 10-10 show\_product\_partial.psp***

<%@ plsql procedure="show\_product\_partial" %>

<%@ plsql parameter="p\_minprice" default="100" %>

<HTML>

<HEAD><TITLE>Show Items Greater Than Specified Price</TITLE></HEAD>

<BODY>

<P>This report shows the items whose price is greater than <%= p\_minprice %>.

<UL>

<% FOR ITEM IN (SELECT product\_name, list\_price, catalog\_url

FROM product\_information

WHERE list\_price > p\_minprice

ORDER BY list\_price DESC)

LOOP %>

<LI>

Item = <A HREF="<%= item.catalog\_url %>"><%= item.product\_name %></A><BR>

Price = <BIG><%= item.list\_price %></BIG><BR>

<% END LOOP; %>

</UL>

</BODY>

</HTML>

After loading [Example 10-10](http://docs.oracle.com/cd/E11882_01/appdev.112/e41502/adfns_psp.htm#BHAHGAAF) into the database, you can pass a parameter to the show\_product\_partial procedure through a URL. This example specifies a minimum price of 250:

http://www.example.com/pls/show\_product\_partial?p\_minprice=250

Filtering results is appropriate for applications such as search results, where users might be overwhelmed by choices. But in a retail situation, you might want to use the alternative technique illustrated in [Example 10-11](http://docs.oracle.com/cd/E11882_01/appdev.112/e41502/adfns_psp.htm#BHAECEIB), so that customers can still choose to purchase other items:

* Instead of filtering the results through a WHERE clause, retrieve the entire result set and then take different actions for different returned rows.
* Change the HTML to highlight the output that meets their criteria. [Example 10-11](http://docs.oracle.com/cd/E11882_01/appdev.112/e41502/adfns_psp.htm#BHAECEIB) uses the background color for an HTML table row. You can also insert a special icon, increase the font size, or use another technique to call attention to the most important rows.
* Present the results in an HTML table.

***Example 10-11 show\_product\_highlighed.psp***

<%@ plsql procedure="show\_product\_highlighted" %>

<%@ plsql parameter="p\_minprice" default="100" %>

<%! v\_color VARCHAR2(7); %>

<HTML>

<HEAD><TITLE>Show Items Greater Than Specified Price</TITLE></HEAD>

<BODY>

<P>This report shows all items, highlighting those whose price is

greater than <%= p\_minprice %>.

<P>

<TABLE BORDER>

<TR>

<TH>Product</TH>

<TH>Price</TH>

</TR>

<% FOR ITEM IN (SELECT product\_name, list\_price, catalog\_url

FROM product\_information

WHERE list\_price IS NOT NULL

ORDER BY list\_price DESC) LOOP

IF item.list\_price > p\_minprice THEN

v\_color := '#CCCCFF';

ELSE

v\_color := '#CCCCCC';

END IF;

%>

<TR BGCOLOR="<%= v\_color %>">

<TD><A HREF="<%= item.catalog\_url %>"><%= item.product\_name %></A></TD>

<TD><BIG><%= item.list\_price %></BIG></TD>

</TR>

<% END LOOP; %>

</TABLE>

</BODY>

</HTML>

### Using an HTML Form to Invoke a PL/SQL Server Page

[Example 10-12](http://docs.oracle.com/cd/E11882_01/appdev.112/e41502/adfns_psp.htm#BHAGJABB) shows a bare-bones HTML form that allows the user to enter a price. The form invokes the show\_product\_partial stored procedure illustrated in [Example 10-10](http://docs.oracle.com/cd/E11882_01/appdev.112/e41502/adfns_psp.htm#BHAHGAAF) and passes it the entered value as the p\_minprice parameter.

To avoid coding the entire URL of the stored procedure in the ACTION= attribute of the form, you can make the form a PSP file so that it resides in the same directory as the PSP file that it invokes. Even though this HTML file contains no PL/SQL code, you can give it a .psp extension and load it as a stored procedure into the database. When the product\_form stored procedure is executed through a URL, it displays the HTML exactly as it appears in the file.

***Example 10-12 product\_form.psp***

<HTML>

<BODY>

<FORM method="POST" action="show\_product\_partial">

<P>Enter the minimum price you want to pay:

<INPUT type="text" name="p\_minprice">

<INPUT type="submit" value="Submit">

</FORM>

</BODY>

</HTML>

### Including JavaScript in a PSP File

To produce an elaborate HTML file, perhaps including dynamic content such as JavaScript, you can simplify the source code by implementing it as a PSP. This technique avoids having to deal with nested quotation marks, escape characters, concatenated literals and variables, and indentation of the embedded content.

[Example 10-13](http://docs.oracle.com/cd/E11882_01/appdev.112/e41502/adfns_psp.htm#BHAGACAH) shows a version of [Example 10-10](http://docs.oracle.com/cd/E11882_01/appdev.112/e41502/adfns_psp.htm#BHAHGAAF) that uses JavaScript to display the order status in the browser status bar when the user moves his or her mouse over the product URL.

***Example 10-13 show\_product\_javascript.psp***

<%@ plsql procedure="show\_product\_javascript" %>

<%@ plsql parameter="p\_minprice" default="100" %>

<HTML>

<HEAD>

<TITLE>Show Items Greater Than Specified Price</TITLE>

<SCRIPT language="JavaScript">

<!--hide

var text=" ";

function overlink (text)

{

window.status=text;

}

function offlink (text)

{

window.status=text;

}

//-->

</SCRIPT>

</HEAD>

<BODY>

<P>This report shows the items whose price is greater than <%= p\_minprice %>.

<P>

<UL>

<% FOR ITEM IN (SELECT product\_name, list\_price, catalog\_url, product\_status

FROM product\_information

WHERE list\_price > p\_minprice

ORDER BY list\_price DESC)

LOOP %>

<LI>

Item =

<A HREF="<%= item.catalog\_url %>"

onMouseover="overlink('PRODUCT STATUS: <%= item.product\_status %>');return true"

onMouseout="offlink(' ');return true">

<%= item.product\_name %>

</A>

<BR>

Price = <BIG><%= item.list\_price %></BIG><BR>

<% END LOOP; %>

</UL>

</BODY>

</HTML>

## Debugging PL/SQL Server Pages

As you begin experimenting with PL/SQL Server Pages, and as you adapt your first simple pages into more elaborate ones, keep these guidelines in mind when you encounter problems:

* The first step is to get all the PL/SQL syntax and PSP directive syntax right. If you make a mistake here, the file does not compile.
  + Use semicolons to terminate lines where required.
  + If a value must be quoted, quote it. You might need to enclose a value in single quotation marks (which PL/SQL needs) inside double quotation marks (which PSP needs).
  + Mistakes in the PSP directives are usually reported through PL/SQL syntax messages. Check that your directives use the right syntax, that directives are closed properly, and that you are using the right element (declaration, expression, or code block) depending on what goes inside it.
  + PSP attribute names are case-sensitive. Most are specified in all lowercase; contentType and errorPage must be specified as mixed-case.
* When using a URL to request a PSP, you might get an error that the file is not found. In this case:
  + Be sure you are requesting the right virtual path, depending on the way the web gateway is configured. Typically, the path includes the host name, optionally a port number, the schema name, and the name of the stored procedure (with no .psp extension).
  + If you use the -replace option when compiling the file, the old version of the stored procedure is erased. So, after a failed compilation, you must fix the error or the page is not available. You might want to test scripts in a separate schema, then load them into the production schema.
  + If you copied the file from another file, remember to change any procedure name directives in the source to match the correct file name.
  + When you get one file-not-found error, request the latest version of the page the next time. The error page might be cached by the browser. You might need to force a page reload in the browser to bypass the cache.
* When the PSP script is run, and the results come back to the browser, use standard debugging techniques to check for and correct wrong output. The difficult part is to configure the interface between different HTML forms, scripts, and CGI programs so that the right values are passed into your page. The page might return an error because of a parameter mismatch.

Guidelines:

* + To determine exactly what is being passed to your page, use METHOD=GET in the invoking form so that the parameters are visible in the URL.
  + Ensure that the form or CGI program that invokes your page passes the correct number of parameters, and that the names specified by theNAME= attributes on the form match the parameter names in the PSP file. If the form includes any hidden input fields, or uses the NAME= attribute on the Submit or Reset buttons, then the PSP file must declare equivalent parameters.
  + Ensure that the parameters can be cast from string into the correct PL/SQL types. For example, do not include alphabetic characters if the parameter in the PSP file is declared as a NUMBER.
  + Ensure that the query string of the URL consists of name-value pairs, separated by equals signs, especially if you are passing parameters by constructing a hard-coded link to the page.
  + If you are passing a lot of parameter data, such as large strings, you might exceed the volume that can be passed with METHOD=GET. You can switch to METHOD=POST in the invoking form without changing your PSP file.
  + Although the loadpsp command reports line numbers correctly when there is a syntax error in your source file, line numbers reported for runtime errors refer to a transformed version of the source and do not match the line numbers in the original source. When you encounter errors that produce an error trace instead of the expected web page, you must locate the error through exception handlers and by printing debug output.