Oracle® Internet Application Server

Using mod_plsql

Release1.0.1

July 18, 2000

Part No. A83590-02

ORACLE®

Using mod_plsql for iAS Release1.0.1

Part No. A83590-02

Primary Author: Dave Mathews

Contributors: Ron Decker, Pushkar Kapasi, Sanjay Khanna, Eric Lee, Kannan Muthukkaruppan

Contents

1 mod_plsql Overview

Send Us Your Comments

Using mod_plsql for Oracle Internet Application Server Release 1.0.1

Part No. A83590-02

Oracle Corporation welcomes your comments and suggestions on the quality and usefulness of this document. Your input is an important part of the information used for revision.

- Did you find any errors?
- Is the information clearly presented?
- Do you need more information? If so, where?
- Are the examples correct? Do you need more examples?
- What features did you like most?

If you find any errors or have any other suggestions for improvement, then indicate the document title and part number, and the chapter, section, and page number (if available). You can send comments to us in the following way:

Electronic mail: webdbdoc@us.oracle.com

If you would like a reply, please give your name, address, and telephone number.

If you have problems with the software, contact your local Oracle Support Services.

Preface

This manual describes how to install, configure, and maintain mod_plsql for iAS 1.0.1. It contains the following chapters:

Chapter 1	Provides an overview of mod_plsql and its features.
Chapter 2	Explains how to install mod_plsql.
Chapter 3	Describes global mod_plsql settings and those for individual Data Access Descriptors .
Chapter 4	Describes special considerations for running Oracle WebDB versions 2.0, 2.1, and 2.2 with mod_plsql.
Chapter 5	Describes how to install the PL/SQL Web Toolkit. Before you can use mod_plsql, you must install the packages in the PL/SQL Web Toolkit in a common schema called owa_public in your Oracle database.
Chapter 6	Provides step-by-step instructions for guide creating and invoking a simple application that displays the contents of a database table in an HTML page.

Related Documents

For more information, see the following manuals:

- *Internet Application Server, Release 1.0.1 - Migrating from Oracle Application Server A83709-02*
- *Internet Application Server, Release 1.0.1 - Overview A83707-02*

- *Oracle Internet Application Server Installation Guide A83708-02*
- *Release Notes for Solaris and Windows NT A83588-02*

1

mod_plsql Overview

Oracle Internet Application Server (iAS) consolidates Oracle's middle-tier products into a single solution for development and deployment of Web applications. The standard version of iAS version 1.0 includes:

- Oracle HTTP Server (powered by Apache) and Servlet Engine
- Oracle Java Server Pages (JSP) Engine
- mod_plsql
- Oracle 8i Cache
- Oracle Tools (included in Enterprise Edition)

mod_plsql provides support for building and deploying PL/SQL-based applications on the Web. PL/SQL stored procedures can retrieve data from database tables and generate HTTP responses containing data and code to display in a Web browser. mod_plsql supports other Oracle products such as WebDB 2.2 and includes a number of new features.

1.1 Stateless and Stateful modes

The database session state includes the state of PL/SQL package variables, application state, and transaction state.

In a stateless environment, each HTTP request from a client maps to a new database session. Application state is typically maintained in HTTP cookies or database tables. Transaction state cannot span across requests. If a PL/SQL procedure executes successfully, an implicit commit is performed. If it executes with an error, an implicit rollback is performed.

In a stateful environment, each HTTP request from a client maps to the same database session. Application state is preserved in PL/SQL package variables. A

transaction can span across requests because no implicit commits or rollbacks are performed

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

- iAS plus mod_plsql supports running in stateless mode. This is the recommended configuration for users who want to develop stateless PL/SQL-based Web applications.
- iAS plus mod_OSE supports running in both stateless and stateful mode. This is the recommended configuration for users who want to develop stateful PL/SQL- and Java-based Web applications. When using mod_OSE, the stateful mode is preferable because a new database session does not have to be created and destoyed for every HTTP request. For more information, see the mod_OSE documentation.

1.2 Database Access Descriptors

Each mod_plsql request is associated with a database access descriptor (DAD), a named set of configuration values used for database access. A DAD specifies information such as:

- the database alias (SQL*Net V2 service name).
- a connect string if the database is remote.
- a procedure for uploading and downloading documents.

You can also specify a username and password information in a DAD; if they are not specified, the user will be prompted to enter a username and password when the URL is invoked.

1.3 Processing client requests

The following occurs when a server receives a request:

1. The Web server receives a mod_plsql request from a client and forwards the request to mod_plsql.
2. mod_plsql uses the DAD's configuration values (see "Configuring mod_plsql" on page 3-7 for more information) to determine how to connect to the database.
3. mod_plsql connects to the database, prepares the call parameters, and invokes the PL/SQL procedure in the database.

4. The PL/SQL procedure generates an HTML page, which can include dynamic data accessed from tables in the database as well as static data.
5. The output from the procedure is returned via the response buffer back to mod_plsql and the client.

The procedure that mod_plsql invokes should return HTML data back to the client. To simplify this task, mod_plsql comes with the PL/SQL Web Toolkit, a set of packages that you can use in your stored procedure to get information about the request, construct HTML tags, and return header information to the client. You install the toolkit in a common schema so that all users can access it. See "Using the PL/SQL Web Toolkit" on page 5-13 for more information.

1.4 Invoking mod_plsql

To invoke mod_plsql in a Web browser, the URL must typically be in the following format:

protocol`://`*hostname*`[:`*port*`]/`*prefix*`/`*DAD*`/[[!][`*schema*`.][`*package*`.]`*proc_name*`[?`*query_string*`]]`

where:

protocol can be either `http` or `https`. For SSL, use https.

hostname is the machine where the Web server is running.

port is the port at which the application server is listening. If omitted, port 80 is assumed.

prefix is a virtual path to handle PL/SQL requests that you have configured in the Web server. `pls` is the default setting for this parameter. For example, you can configure the Web server to set `pls` as the prefix so that all requests containing the `pls` prefix are routed to mod_plsql.

DAD is the DAD entry to be used for this URL.

! character, if present, indicates that flexible parameter passing scheme must be used. See"Flexible Parameter Passing" on page 1-22 for more information.

schema is the database schema name. If omitted, name resolution for *package.proc_name* occurs based on the database user that the URL request is processed as.

package is the package that contains the PL/SQL stored procedure. If omitted, the procedure is stand-alone.

proc_name specifies the PL/SQL stored procedure to run. This must be a procedure and not a function. It can accept only IN arguments.

?query_string specifies parameters (if any) for the stored procedure. The string follows the format of the GET method. For example:

- Multiple parameters are separated with the & character, and space characters in the values to be passed in are replaced with the + character.
- If you use HTML forms to generate the string (as opposed to generating the string yourself), the formatting will be done automatically for you.
- The HTTP request may also choose the HTTP POST method to post data to mod_plsql. See "POST and GET Methods" on page 1-15 for more information.

For example, if a Web server is configured with pls as a prefix and the browser sends the following URL:

```
http://www.acme.com:9000/pls/mydad/mypackage.myproc
```

the Web server running on `www.acme.com` and listening at port `9000` would handle the request. When the Web server receives the request, it will pass the request to mod_plsql. This is because the pls prefix indicates that the Web server is configured to invoke mod_plsql. mod_plsql then uses the DAD associated with mydad and runs the myproc procedure stored in mypackage.

You can specify a URL without a DAD, schema or stored procedure name. For example, if you specify

```
http://www.acme.com:9000/pls/mydad
```

then the default home page for the `mydad` DAD (as specified on the mod_plsql configuration page) displays.

If you specify

```
http://www.acme.com:9000/pls
```

the default DAD's default home page is invoked.

Generally, you do not need to be concerned with the order in which PL/SQL parameters are given in the URL or the HTTP header, because the parameters are passed by name. However, there are some exceptions to this rule. Please refer to Parameter passing on page 1-20 for more information.

1.4.1 POST and GET Methods

POST and GET methods in the HTTP protocol instruct browsers how to pass parameter data (usually in the form of name-value pairs) to applications. The parameter data are usually generated by HTML forms.

mod_plsql applications can use either method. The method that you use is as secure as the underlying transport protocol (http or https).

When you use the POST method, parameters are passed in the request body. When you use the GET method, parameters are passed using a query string. These methods are described in the HTTP 1.1 specification, which is available at the W3C web site at:

```
http://www.w3.org/Protocols/HTTP/1.1/draft-ietf-http-v11-spec-rev-01.txt
```

The limitation of the GET method is that the length of the value in a name-value pair cannot exceed the maximum length for the value of an environment variable, as imposed by the underlying operating system. In addition, operating systems have a limit on how many environment variables you can define.

Generally, if you are passing large amounts of parameter data to the server, you should use the POST method instead.

1.5 Overview of mod_plsql Features

1.5.1 Authentication

mod_plsql provides levels of authentication in addition to those provided by the Web Server itself. Whereas the Web server protects documents, virtual paths, etc., mod_plsql protects users logging into the database or running a PL/SQL Web application.

1.5.1.1 Database Controlled Authentication

mod_plsql supports authentication at the database level. It uses HTTP Basic Authentication but authenticates credentials by using them to attempt to log on to the database. Authentication is verified against a user database account, using user names and passwords that are either:

- stored in the DAD. The end user is not required to log in. This method is useful for Web pages that provide public information
- provided by the users via a browser-based basic HTTP authentication dialog box. The end user must provide a username and password in the dialog box.

1.5.1.2 Deauthentication

mod_plsql allows users to log off (clear HTTP authentication information) programatically through a PL/SQL procedure without having to exit all instances of the browser. Because of the use of cookies, this feature is supported on Netscape 3.0 or higher and Internet Explorer. On other browsers, the user may have to exit the browser to deauthenticate.

Another method of deauthentication is to add `/logmeoff` after the DAD in the URL, for example

```
http://myhost:2000/pls/myDAD/logmeoff
```

1.5.1.3 Custom Authentication

Custom authentication enables applications to authenticate users within the application itself, not at the database level.

You can enable custom authentication using the **Custom Authentication** parameter on mod_plsql configuration page, or using the custom_auth parameter. This parameter can be set to one of the following values:

- Basic - authentication is performed using basic HTTP authentication. Most applications will use Basic authentication.
- Global Owa - authorization id performed in the OWA package schema.
- Custom Owa - authorization is performed using packages and procedures in the user's schema, or if not found, in the OWA package schema
- PerPackage - authentication is performed by packages and procedures in the user's schema
- Single Sign-On - authentication is performed using the Oracle Single Sign-On feature of the Login Server. You can use this mode only if your application is set up to work with the Login Server

1.5.1.3.1 Implementing the authorize function

Custom authentication needs a static username/password to be stored in a configuration file, and cannot be combined with the dynamic username/password authentication.

The syntax of the authorize function is:

```
function authorize return boolean;
```

To enable custom authentication, you must

1. Set the level of authentication by editing the privcust.sql file
2. Reload it
3. Implement the authentication function.

mod_plsql uses the username/password provided in the DAD to log into the database. Once the login is complete, authentication control is passed to the application. Application-level PL/SQL hooks (callback functions) are then called. The implementations for these callback functions are left to the application developers. The return value of the callback function determines if the authentication succeeded or failed: if the function returns TRUE, authentication succeeded. If it returns FALSE, authentication failed and code in the application is not executed.

You can place the authentication function in different locations, depending on when it is to be invoked:

If you want the same authentication function to be invoked for all users and for all procedures, change the line in the privcust.sql file to:

```
owa_sec.set_authorization(OWA_SEC.GLOBAL)
```

and implement the owa_custom.authorize function in the OWA Package schema, which contains the PL/SQL Web Toolkit.

If you want a different authentication function to be invoked for each user and for all procedures, change the line in the privcust.sql file to:

```
owa_sec.set_authorization(OWA_SEC.CUSTOM)
```

and implement the owa_custom.authorize function in each user's schema. For users who do not have that function in their schema, the owa_custom.authorize function in the OWA package schema will be invoked instead.

For 3.0 users: if you implemented owa_init.authorize in each user's schema, you need to migrate the function to each user's owa_custom package.

If you want the authentication function to be invoked for all users but only for procedures in a specific package or for anonymous procedures, change the line in the privcust.sql file to:

```
owa_sec.set_authorization(OWA_SEC.PER_PACKAGE)
```

and implement the authorize function in that package in each user's schema. If the procedure is not in a package, then the anonymous authorize function is called instead. The following table summarizes the parameter values:

Value for parameter	Access control scope	Callback function
OWA_SEC.NO_CHECK	N / A	N / A
OWA_SEC.GLOBAL	All packages	owa_custom.authorize in the OWA package schema
OWA_SEC.PER_PACKAGE	Specified package	packageName.authorize in the user's schema
OWA_SEC.PER_PACKAGE	Anonymous procedures	authorize in the user's schema
OWA_SEC.CUSTOM	All package	owa_custom.authorize in the user's schema, or, if not found, in the OWA package schema

When you use custom authentication, you can use the subprograms in the owa_sec package. You should not use owa_sec if you are not using custom authentication.

1.5.2 Transaction model

After processing a URL request for a procedure invocation, mod_plsql performs a rollback if there were any errors. Otherwise, the Gateway performs a commit. This mechanism does not allow a transaction to span across multiple HTTP requests. In this stateless model, applications typically maintain state using HTTP cookies or database tables.

1.5.3 Parameter passing

mod_plsql supports:

- Parameter passing by name

  Each parameter in a URL that invokes procedure or functions identified by a unique name. Overloaded parameters are supported. See "Overloaded parameters" on page 1-20 for more information.

- Flexible parameter passing

  Parameters are prefixed by a ! character. See "Flexible Parameter Passing" on page 1-22 for more information.

- Large (up to 32K) parameters.

  See "Large parameters" on page 1-23 for more information.

1.5.3.1 Overloaded parameters

Overloading allows multiple subprograms (procedures or functions) to have the same name, but differ in the number, order, or the datatype family of the parameters. When you call an overloaded subprogram, the PL/SQL compiler determines which subprogram to call based on the data types passed.

PL/SQL allows you to overload local or packaged subprograms; stand-alone subprograms cannot be overloaded. See the *PL/SQL User's Guide* in the Oracle Server documentation for more information on PL/SQL overloading.

You must give parameters different names for overloaded subprograms that have the same number of parameters. Because HTML data is not associated with datatypes, it is impossible for mod_plsql to know which version of the subprogram to call.

For example, PL/SQL allows you to define the two procedures in the example below. If parameter names for these procedures are the same, an error occurs when you try to use them with mod_plsql:

```
-- legal PL/SQL, but not for mod_plsql
CREATE PACKAGE my_pkg AS
  PROCEDURE my_proc (val IN VARCHAR2);
  PROCEDURE my_proc (val IN NUMBER);
END my_pkg;
```

To avoid the error, name the parameters differently. For example:

```
-- legal PL/SQL and also works for mod_plsql
CREATE PACKAGE my_pkg AS
  PROCEDURE my_proc (valvc2 IN VARCHAR2);
  PROCEDURE my_proc (valnum IN NUMBER);
END my_pkg;
```

The URL to invoke the first version of the procedure looks something like:

```
http://www.acme.com/pls/myDAD/my_pkg.my_proc?valvc2=input
```

The URL to invoke the second version of the procedure looks something like:

```
http://www.acme.com/pls/myDAD/my_pkg.my_proc?valnum=34
```

1.5.3.2 Overloading and PL/SQL Arrays

If you have overloaded PL/SQL procedures where the parameter names are identical, but where the data type is *owa_util.ident_arr* (a table of varchar2*)* for one procedure and a scalar type for another procedure, mod_plsql can still distinguish between the two procedures. For example, if you have the following procedures:

```
CREATE PACKAGE my_pkg AS
  PROCEDURE my_proc (val IN VARCHAR2); -- scalar data type
  PROCEDURE my_proc (val IN owa_util.ident_arr); -- array data type
END my_pkg;
```

Each of these procedures has a single parameter of the same name, `val`.

When mod_plsql gets a request that has only one value for the val parameter, it invokes the procedure with the scalar data type. When it gets a request with more than one value for the val parameter, it then invokes the procedure with the array data type.

Example 1: If you send the following URL:

```
http://www.acme.com/pls/myDAD/my_proc?val=john
```

the scalar version of the procedure executes.

Example 2: If you send the following URL:

```
http://www.acme.com/pls/myDAD/my_proc?val=john&val=sally
```

the array version of the procedure executes.

To ensure that the array version of the procedure executes, use hidden form elements on your HTML page to send dummy values that are checked and discarded in your procedure.

1.5.3.3 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, or you could insert hidden form elements to ensure that the names in the query string are consistent each time, regardless of which elements the user chooses.

mod_plsql makes this easier by supporting a flexible parameter passing scheme. In order to use flexible parameter passing for a URL-based procedure invocation, prefix the name of the procedure with a '!' character in the URL. The procedure must have the following signature:

```
procedure [proc_name] is
      name_array  IN  [array_type]
      value_array IN  [array_type],
where:
```

[*proc_name]* is the name of the PL/SQL procedure that you are invoking.

name_array specifies the names from the query string (indexed from 1) in the order submitted.

value_array specifies the values from the query string (indexed from 1) in the order submitted.

[array_type] is any PL/SQL index-by table of varchar2 type (e.g., owa.vc_arr).

Note The above is a two parameter interface, which is recommended for use with mod_plsql. A four parameter interface is supported for compatibility.

Example 1: If you send the following URL:

```
http://www.acme.com/pls/myDAD/!scott.my_proc?x=john&y=10&z=doe
```

The '!' prefix tells mod_plsql that it must use flexible parameter passing. It will invoke procedure *scott.myproc* and pass it the following two arguments:

```
num_entries ==] 3

reserved ==] ()
```

Example 2: If you send the following URL, where the *query_string* has duplicate occurrences of the name "x":

```
http://www.acme.com/pls/myDAD/!scott.my_pkg.my_proc?x=a&y=b&x=c
```

The '!' prefix tells mod_plsql that it must use flexible parameter passing. It will invoke procedure `scott.my_pkg.myproc` and pass it the following four arguments:

```
num_entries ==] 3
name_array ==] ('x', 'y', 'x');
values_array ==] ('a', 'b', 'c')
reserved ==] ()
```

1.5.3.4 Large parameters

Section 1.5.3.2 and Section 1.5.3.3 above indicate that you can use mod_plsql to invoke procedures with either scalar or index-by table of varchar2 arguments. The values passed as scalar arguments and values that are passed as elements to the index-by table of varchar2 arguments can be up to 32K in size.

For example, when using flexible parameter passing (described in "Flexible Parameter Passing" on page 1-22), each name or value in the *query_string* portion of the URL gets passed as an element of the `name_array` or `value_array` argument to the procedure being invoked. These names or values can be up to 32KB in size.

1.5.4 File Upload and Download

mod_plsql allows you to:

- Upload and download files as raw byte streams without any character set conversions. The files are uploaded into the document table. A primary key is passed to the PL/SQL upload handler routine so that it can retrieve the appropriate row of the table.
- Specify one or more tables per application for uploaded files so that files from different applications are not mixed together.
- Provide access to files in these tables via a URL format that doesn't use query strings, for example

  ```
  http://myhost/mysite/pls/docs/cs250/lecture1.htm
  ```

  This is required to support uploading a set of files that have relative references to each other.
- Upload multiple files per form submission.
- Upload files into LONG RAW and BLOB types of columns in the document table.

1.5.4.1 Document Table Definition

mod_plsql enables you to specify the document storage table on a per DAD basis. The document storage table must have the following definition:

```
CREATE TABLE [table_name] (
NAME VARCHAR2(256) UNIQUE NOT NULL,
MIME_TYPE VARCHAR2(128),
DOC_SIZE NUMBER,
DAD_CHARSET VARCHAR2(128),
LAST_UPDATED DATE,
content_typeVARCHAR2(128),
[content_column_name] [content_column_type]
[ , [content_column_name] [content_column_type]]*
);
```

Users can choose the table_name. The content_column_type type must be either LONG RAW or BLOB.

The content_column_name depends on the corresponding content_column_type:

- If content_column_type is LONG RAW, the content_column_name must be CONTENT.
- If content_column_type is BLOB, the content_column_ name must be CONTENT_BLOB.

An example of legal document table definition is:

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

1.5.4.1.1 Semantics of the CONTENT column

The actual contents of the table will be stored in a content column. There can be more than one content columns in a document table. However, for each row in the document table, only one of the content column is used. The other content columns are set to NULL.

1.5.4.1.2 Semantics of the CONTENT_TYPE column

The content_type column is used to track which content column the document is stored in. When a document is uploaded, mod_plsql will set the value of this column to be the type name (i.e. the [content_column_type] of the content column into which the document is uploaded.

For example, if a document was uploaded into the BLOB content column, then the content_type column for the document will be set to the string ‘BLOB’.

1.5.4.1.3 Semantics of the LAST_UPDATED column

The LAST_UPDATED column reflects a document's creation or last modified time.When a document is uploaded, mod_plsql will set the LAST_UPDATED column for the document to be the database server time (as obtained from sysdate()) at the time of upload. If an application subsequently modifies or replaces the contents or attributes of the document, it must also update the LAST_UPDATED time.

The LAST_UPDATED column is used by mod_plsql to check and indicate to the HTTP client (e.g., a browser) if it is okay for the HTTP client to use a previously cached version of the document. This helps reduce network traffic and response times and improves server performance and scalability.

1.5.4.1.4 Semantics of the DAD_CHARSET column

The DAD_CHARSET column keeps track of the character set setting at the time of the file upload.

1.5.4.2 Old Style Document Table Definition

For backward capability with the document model used by older releases of WebDB 2.X, mod_plsql will also support the following old definition of the document storage table where the content_type DAD_CHARSET and LAST_UPDATED columns are not present.

```
/* older style document table definition (DEPRECATED) */
CREATE TABLE [table_name]
(
NAME VARCHAR2(128),
MIME_TYPE VARCHAR2(128),
DOC_SIZE NUMBER,
CONTENT LONG RAW
;
```

1.5.4.3 Relevant Parameters

For each DAD, the following configuration parameters are relevant for file upload / download.

1.5.4.3.1 document_table (Document Table Name)

The document_table parameter specifies the name of the table to be used for storing documents when file uploads are performed via this DAD.

Syntax

```
document_table = [document_table_name]
```

Examples

```
document_table = my_documents
```

or,

```
document_table = scott.my_document_table
```

1.5.4.4 document_path (Document Access Path)

This specifies the path element to immediately follow the DAD name in the URL to access a document. For example, if the document access path is `docs`, then the URL to access a document might look like:

```
http://neon/myDAD/docs/myfile.htm
```

where `myDAD` is the DAD name and `myfile.htm` is the file name. The document access path mechanism enables the standard-style document access URLs required for WebDB's features for building Web sites.

Syntax

```
document_path = [document_access_path_name]
```

1.5.4.4.1 document_proc (Document Access Procedure):

This is an application-specified procedure, with no parameters, that processes a URL request with the document access path. The document access procedure should call `wpg_docload.download_file(filename)` to initiate download of a file. It should figure out the filename based on the complete URL specification. This can be used by an application, for example, to implement file-level access controls and versioning. An example of such an application is shown in "File Download" on page 1-31.

Syntax

```
document_proc = [document_access_procedure_name]
```

Examples

```
document_proc = my_access_procedure
```

or,

```
document_proc = scott.my_pkg.my_access_procedure
```

1.5.4.4.2 upload_as_long_raw

The DAD parameter `upload_as_long_raw` is used to configure file uploads based on their file extensions. The value of an `upload_as_long_raw` DAD parameter is a (,) comma separated list of file extensions. Files with these extensions will be uploaded by mod_plsql into the content column of `long_raw` type in the document table.

The file extensions can be text literals (jpeg, gif, etc.). In addition, an asterisk (*) can be used as a special file extension and matches any file whose extension has not been explicitly listed in an `upload_as_long_raw` setting.

Syntax

```
upload_as_long_raw = [file_extension][,[file_extension]]*
```

where **[file_extension]** is an extension for a file (with or without the '.' character, e.g., 'txt' or '.txt') or the wild card character *.

Examples

```
upload_as_long_raw = html, txt
upload_as_long_raw = *
```

1.5.4.5 File Upload

To upload files from a client machine to a database, you create an HTML page that contains:

- A FORM tag whose *enctype* attribute is set to `multipart/form-data` and whose action attribute is associated with a mod_plsql procedure call, referred to as the "action procedure".

- 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.

When a user clicks the submit button to trigger the form action, the following events occur:

1. The browser uploads the contents of the file specified by the user as well as other form data to the server.
2. mod_plsql stores the file contents in the database in the document storage table. The table name is derived from the document_table DAD setting.
3. The action procedure specified in the ACTION attribute of the FORM is run similar to invoking a mod_plsql procedure without file upload.

The following example shows an HTML form that enables a user to select a file from the file system to upload. The form contains other fields that allow the user to provide information about the file.

```
<html>
<head>
<title>test upload</title>
</head>
<body>
 <FORM enctype="multipart/form-data"
action="/sample/plsql/write_info"
method="POST">
<p>Author's Name:<INPUT type="text" name="who">
<p>Description:<INPUT type="text" name="description"><br>
<p>File to upload:<INPUT type="file" name="file"><br>
<p><INPUT type="submit">
</FORM>
</body>
</html>
```

When a user clicks a Submit button on the form, the browser uploads the file listed in the `INPUT type=file` element.

The `write_info` procedure then runs. The procedure writes information from the form fields to a table in the database and returns a page to the user. The action procedure 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.

A sample write_info procedure might look like:

```
procedure write_info (
who         in varchar2,
description in varchar2,
file        in varchar2) as
begin
insert into myTable values (who, description, file);
htp.htmlopen;
htp.headopen;
htp.title('File Uploaded');
htp.headclose;
htp.bodyopen;
htp.header(1, 'Upload Status');
htp.print('Uploaded ' || file || ' successfully');
htp.bodyclose;
htp.htmlclose;
end;
```

The filename obtained from the browser is prefixed with a generated directory name to reduce the possibility of name conflicts. The "action procedure" specified in the form should rename this name to what it wants. So, for instance, when `/private/minutes.txt` is uploaded, the name stored in the table by the gateway would look like `F9080/private/minutes.txt`. The application can rename this to whatever it wants in the called stored procedure. For instance, the application can rename it to `scott/minutes.txt`.

1.5.4.6 Specifying Attributes (Mime Types) of Uploaded Files

In addition to renaming the uploaded file, the stored procedure that is the action target of the form can alter other attributes relating to the file. For example, the form in the example shown in section 1.5.4.5 on page 28 could display a field for allowing the user to input the uploaded document's mime type.

The mime type could be received as a parameter in write_info. The document table could then store the mime type for the document instead of the default mime type that is parsed from the multipart form by mod_plsql when uploading the file.

1.5.4.7 Uploading Multiple Files

To upload multiple files per submit action, the upload form must include multiple <INPUT type="file" name="file"> elements. If more than one file INPUT element defines `name` to be of the same name, then the action procedure must declare that parameter name to be of type owa.vc_arr. The names defined in the file INPUT elements could also be unique, in which case the action procedure must declare

each of them to be of varchar2. For example, if a form contained the following elements:

```
<INPUT type="file" name="textfiles">
<INPUT type="file" name="textfiles">
<INPUT type="file" name="binaryfile">
```

then the action procedure must contain the following parameters:

```
procedure handle_text_and_binary_files(textfiles IN owa.vc_arr,
binaryfile IN varchar2).
```

1.5.4.8 File Download

After you have uploaded files to the database, you can download them, delete them from the database, and read and write their attributes.

To download a file, create a stored procedure with no parameters that calls wpg_docload.download_file(file_name) to initiate the download. The document download packages are in docload.sql. See "Installing required packages" on page 2-2 for more information about docload.sql.

The HTML page presented to the user will simply have a link to a URL which includes the Document Access Path and specifies the file to be downloaded.

For example, if the webview DAD specifies that the Document Access Path is docs and the Document Access Procedure is webview.process_download, then the webview.process_download procedure will be called when the user clicks on a URL such as

```
http://acme/pls/webview/docs/myfile.htm.
```

An example implementation of process_download is:

```
procedure process_download is
v_filename varchar2(255);
begin
  -- getfilepath() uses the SCRIPT_NAME and PATH_INFO cgi
  -- environment variables to construct the full pathname of
  -- the file URL, and then returns the part of the pathname
  -- following '/docs/'
  v_filename := getfilepath;
  select name into v_filename from plsql_gateway_doc
  where UPPER(name) = UPPER(v_filename);
  -- now we call docload.download_file to initiate
  -- the download.
  wpg_docload.download_file(v_filename);
exception
  when others then
v_filename := null;
end process_download;
```

Any time you call wpg_docload.download_file(filename) from a procedure running in the gateway, a download of the file *filename* will be initiated. The restriction, however, is that when a file downloaded is initiated, no other HTML (produced via HTP interfaces) generated by the procedure, will be passed back to the browser.

mod_plsql looks up for the file filename in the document table. There must be a unique row in the document table whose NAME column matches filename. mod_plsql generates appropriate HTTP response headers based on the information in the MIME_TYPE column of the document table. The content_type column's value determines which of the content columns get the document's content from. The contents of the document are sent as the body of the HTTP response.

1.6 CGI Environment Variables

The OWA_UTIL package provides an API to get the values of CGI environment variables, which serve to provide a kind of context to the procedure being executed via mod_plsql. Although mod_plsql is not operated through CGI, the PL/SQL application invoked from mod_plsql can access these CGI environment variables.

mod_plsql provides the following CGI environment variables:

- REMOTE_USER
- DAD_NAME
- DOC_ACCESS_PATH
- PATH_INFO
- SCRIPT_NAME
- SERVER_PORT
- SERVER_NAME
- REQUEST_METHOD
- REMOTE_HOST
- REMOTE_ADDR
- SERVER_PROTOCOL
- HTTP_USER_AGENT
- HTTP_PRAGMA
- HTTP_HOST
- HTTP_ACCEPT
- HTTP_ACCEPT_ENCODING
- HTTP_ACCEPT_LANGUAGE
- HTTP_ACCEPT_CHARSET
- REQUEST_CHARSET (see "REQUEST_CHARSET CGI environment variable" on page 1-34 for more information)
- REQUEST_IANA_CHARSET
- DOCUMENT_TABLE (See "document_table (Document Table Name)" for more information)
- AUTHORIZATION

A PL/SQL application can get the value of a CGI environment variable using the owa_util.get_cgi_env interface.

Syntax:

```
owa_util.get_cgi_env(param_name in varchar2) return varchar2;
```

where

param_name is the name of the CGI environment variable. param_name is case-insensitive.

1.6.1 NLS

The NLS extensions are part of the DAD or global settings in the Gateway configuration and they provide a flexible infrastructure to request and retrieve values to and from Oracle databases in different languages/formats. Even when the database is configured with other NLS settings, all the conversions are handled implicitly by the database and mod_plsql.

For example, if you have a database that is configured with US or NLS Currency but you want to present the values in Japanese Yen to the user, all you need to do is set NLS Currency to Japanese Yen. When the data is retrieved from the database, it will be presented as Japanese Yen.

1.6.1.1 REQUEST_CHARSET CGI environment variable

Every request to mod_plsql is associated with a DAD. The CGI environment variable REQUEST_CHARSET will be set as per the following rules:

- Otherwise, if NLS_LANG is specified as part of the Gateway's global configuration information, then the REQUEST_CHARSET CGI environment variable will be set to the character set portion of the global NLS_LANG parameter.
- Otherwise, the REQUEST_CHARSET will be set to the default character set in use.
  - For the embedded gateway this will be the database's default character set.
  - For the gateway deployed in the middle-tier (as part of WebDB listener or Oracle HTTP Server) this will be the character set information derived from the NLS_LANG environment variable of the WebDB listener process.

The PL/SQL application can access this information via a function call of the form:

```
owa_util.get_cgi_env('REQUEST_CHARSET');
```

1.6.1.2 REQUEST_IANA_CHARSET CGI environment variable

This is the IANA (Internet Assigned Number Authority) equivalent of the REQUEST_CHARSET CGI environment variable. IANA is an authority that globally coordinates the standards for charsets used on the Internet.

2

Installing mod_plsql

2.1 System Requirements

The following are the recommended and minimum requirements for installing and running mod_plsql:

Operating Systems

- Windows NT 4.0 with Service Pack 3 or above
- Solaris 2.6 and above
- IBM AIX 4.3.2 / 4.3.3
- Compaq Tru64 4.0d
- Solaris Intel 2.7

Oracle Database

- Oracle8*i* (Release 8.1.6)

  Note mod_plsql requires the Oracle 8.1.6 client libraries to be installed in the same Oracle Home as mod_plsql. If these libraries are installed, you can still run mod_plsql against Oracle 8.0.5 or above. For example, you can use mod_plsql to run PL / SQL procedures installed in a remote 8.0.5 database.

Web Listener

- On Solaris - Oracle HTTP Server (powered by Apache) 1.3.9 for iAS version 1.0.0
- On Windows NT - Oracle HTTP Server (powered by Apache) 1.3.12 for iAS version 1.0.1

Web Browsers

- Netscape 4.0.8 and above
- Microsoft Internet Explorer 4.0.1 with Service Pack 1 and above

2.2 Before you begin

Before you install mod_plsql using the Internet Application Server (IAS) v1.0 Oracle Universal Installer, you must satisfy the following prerequisite requirements:

- You must have a SYS user password on the database where you plan to load Oracle Web Agent (OWA) packages required by mod_plsql.
- The database to which you plan to connect mod_plsql must be up and running.
- You must have enough disk space on the machine where you plan to run the Oracle Universal Installer.
- You must have write permissions to the directory where the Oracle Universal Installer is writing its oraInventory data.

2.3 Installation

To begin the Oracle Universal Installer, execute the runInstaller application located on your product CD or stage area. Follow the instructions in each step of the installation application, including choosing a directory where you want to install iAS v1.0.1. This install directory will be referred to as <IAS_ROOT> after you choose.

2.4 Installing required packages

After installation, you must manually install additional required packages using the owaload.sql script.

1. Navigate to the directory where the owaload.sql and docload.sql files are located. This directory should be <IAS_ROOT>/Apache/modplsql/owa.
2. Log into the Oracle 8.1.6 database as the SYS user.

3. At a SQL prompt, run the following command:

   ```
   @owaload.sql log_file
   ```

   where

   log_file is the installation log file.

   owaload.sql installs the OWA packages into the SYS schema. It also creates public synonyms and makes the packages public so that all users in the database have access to them. Therefore, only one installation per database is needed.

2.5 Configuring the Oracle HTTP Server Listener

The iAS installation creates configuration files that you can edit, including the following that affect mod_plsql:

<IAS_ROOT>/Apache/Apache/bin/httpdsctl

This script is used to start and stop Oracle HTTP Server. Inside this file, there are three parameters that affect mod_plsql:

- ORACLE_HOME - the Oracle Home in which mod_plsql runs. Default: <IAS_ROOT>
- LD_LIBRARY_PATH - the Oracle libraries needed by mod_plsql. This should point to an Oracle 8.1.6 installation. This parameter is for Solaris only. Default: <IAS_ROOT>/lib
- WV_GATEWAY_CFG - mod_plsql configuration file.
  Default on Solaris: <IAS_ROOT>/Apache/modplsql/cfg/wdbsvr.app
  Default on Windows NT <APACHE_HOME>/modplsql/cfg/wdbsvr.app

If you want to have mod_plsql running in another Oracle Home, remember to change both the ORACLE_HOME and LD_LIBRARY_PATH settings.

On Solaris, if you want mod_plsql to use a different configuration file, just update the httpdsctl file to point to the new configuration file. On Windows NT, you can click Start->Settings->Control Panel->System. Click the Environment tab, then create a System variable called WV_GATEWAY_CFG that points to the new configuration file.

<IAS_ROOT>/Apache/Apache/conf/httpds.conf

This configuration file defines the behavior of Oracle HTTP Server (powered by Apache). You can set your port number as well as other server settings.

<IAS_ROOT>/Apache/modplsql/cfg/plsql.conf

This configuration file describes settings for the mod_plsql module. There settings are configurable:

- LoadModule plsql_module <MOD_PATH> - the location of the mod_plsql module.
  Default on Solaris: <IAS_ROOT> / Apache / modplsql / bin / modplsql.so
  Default on Windows NT: <IAS_ROOT> / Apache / modplsql / bin / modplsq.dll located in $ORACLE_HOME / bin
- <Location <MOUNT_PATH>> - the prefix in the URL for which mod_plsql is invoked.
  Default: / pls

<IAS_ROOT>/Apache/modplsql/cfg/wdbsvr.app

This is the main mod_plsql configuration file. It contains all the DAD information. Please do not edit this file directly. Use mod_plsql configuration page, which you can access through your browser as shown below.

2.6 Accessing the mod_plsql configuration page

To access to mod_plsql configuration page, enter the following URL in your Web browser:

```
http://<hostname>:<port>/pls/DAD/<admin_path>/gateway.htm
```

where:

<hostname> is the machine where the application server is running.

<port> specifies the port at which the application server is listening. If omitted, port 80 is assumed.

<admin_path> specifies the URL path element that identifies an admin page. The default is admin_. For example, if you specify the default of admin_, the following URL will invoke mod_plsql configuration page, given that the invoking user is listed in the administrators configuration setting:

```
http://www.myserver.com/pls/admin_/gateway.htm
```

Configuration settings are protected by the administration security settings. The web administration page can only be invoked by those users whose user names appear in the Administrators setting of the configuration file. See "Configuring mod_plsql" on page 3-7 for more information.

2.6.1 pls.conf configuration file

The Oracle HTTP Listener configuration file includes the modplsql configuration file plsql.conf. The contents of plsql.conf are:

```
#
# Directives added for mod_plsql
#
LoadModule plsql_module %APACHE_HOME%/modplsql/bin/modplsql.so

#
# Enable handling of all virtual paths beginning with "/pls" by mod-plsql
#
<Location /pls>
  SetHandler pls_handler
  Order deny,allow
  Allow from all
</Location>
```

2.7 Starting and stopping the Oracle HTTP Server Listener

To start the Apache listener, type:

```
<IAS_ROOT>/Apache/Apache/bin/httpdsctl start
```

To start the Apache listener with SSL. support, type:

```
<IAS_ROOT>/Apache/Apache/bin/httpdsctl startssl
```

To stop the Apache listener, type:

```
> <IAS_ROOT>/Apache/Apache/bin/httpdsctl stop
```

3

Configuring mod_plsql

mod_plsql provides a Web page for configuring Database Access Descriptors (DADs). A DAD is a set of values that specify how mod_plsql connects to a database server to fulfill an HTTP request.

You can access mod_plsql configuration page at

`http://<hostname>:<port>/pls/admin_/gateway.htm`

3.1 mod_plsql Settings

Global Settings	
Default Database Access Descriptor (DAD)	Specify a path that points to the default DAD. If the end user enters a URL without specifying the DAD name, the home page for the default DAD will be displayed. **Default** = none You can change the DAD name by typing a new one in this field.
Administrators	Specifies who can view the admin pages. By default, this is set to ALL which means anyone can view the admin pages. This should be changed to a comma separated list of users to enforce security on the admin pages, for example *scott, mike* where *scott* and *mike* are local database user names. Or, *scott, mike@orcl* where orcl is a connect string for a remote database. **Note** This setting is accessible through the configuration file only, not through mod_plsql Web page.

Admin Path	Specifies the URL path element that identifies an admin page. This should normally be left unchanged as */admin_/*. **Note** This setting is accessible through the configuration file only, not through mod_plsql Web page.
Database Access Descriptor Settings	
Database Access Descriptor Name	Displays the name for this DAD. The name is set at installation time or during creation of new web sites. You can change the name by typing a new one in this field.
Oracle User Name	Displays the Oracle database account user name. The user name is typically set at installation or during creation of new web sites. You can change it by typing a new name in this entry field.
Oracle Password	Displays the Oracle database account password. The password is typically set at installation, but you change it by typing a new password in this entry field. **Notes** The **Oracle User Name** and **Password** are the default user name and password for logging in to a Web site or page. If you leave the **Oracle User Name** and **Oracle Password** entry fields blank**,** the user will be prompted to enter a user name and password when first logging in.
Oracle Connect String	Enter a SQL*Net alias if you are using a remote database. Leave this field blank if the database is local.
Authentication Mode	This parameter can be set to one of the following values: ■ Basic - authentication is performed using basic HTTP authentication. Most applications will use Basic authentication. ■ Global Owa - authorization id performed in the OWA package schema. ■ Custom Owa - authorization is performed using packages and procedures in the user's schema, or if not found, in the OWA package schema ■ PerPackage - authentication is performed by packages and procedures in the user's schema ■ Single Sign-On - authentication is performed using the Oracle Single Sign-On feature of the Login Server. You can use this mode only if your application is is set up to work with the Login Server.

Session Cookie Name	Enter a session cookie name only for Oracle Portal 3.X installations that participate in a distributed environment.
Create a Stateful Session?	Choose **Yes** to preserve the database package / session state for each database request. Choose **No** to reset it after each request. For mod_plsql, this parameter must be set to **No**.
Keep Database Connection Open Between Requests?	Choose whether, after processing one URL request, the database connection should be kept open to process future requests. In most configurations, choose **Yes** for maximum performance. The mod_plsql cleanup thread cleans up database sessions that have not been used for 15 minutes.
Maximum Number of Open Connections	Enter the size of the connection pool. This is the maximum number of database connections kept open at one time for this DAD. If a request for another connection comes in after the maximum number is reached, one of the connection is closed to serve this request. **Tip Y**ou'll need to adjust this number depending on your server, its capacity, and the number of connected users. As a rule of thumb, set this number at between 5 and 20 at a medium sized installation (approximately 200 users). **Notes** - This field is ignored when the Unix Oracle HTTP Server (powered by Apache) is used with mod_plsql. In a Unix configuration, each server process keeps one database connection pooled for each DAD. Thus, the maximum number of Oracle HTTP Server (powered by Apache) processes currently alive is the maximum size of the connection pool for each DAD. If the number of processes grows, the pool size grows, and the Gateway creates new connection. When a process dies, connections are closed. The maximum number of server processes can be configured through Oracle HTTP Server (powered by Apache) Configuration files. - When NT Oracle HTTP Server (powered by Apache) Server is used with mod_plsql, configuration files govern the maximum number of threads that will simultaneously be serving requests. The **Maximum Number of Open Connections** field governs the maximum number of connection that can be kept open. Therefore, to ensure correct behavior on NT, specify a value that is equal to maximum number of threads specified in the Apache server configuration file. If this number is smaller, some requests may be rejected if threads are idle to serve but maximum connection limit has already been reached.

Keep Database Connection Open Between Requests	Choose whether, after processing one URL request, the database connection should be kept open to process future requests. In most configurations, specify **Yes** for maximum performance.
Default (Home) Page	Enter the PL/SQL procedure that will be invoked when none is specified as part of the URL itself. For example, if you specify a default home page of `myapp.home` and an end user enters this URL in a browser: http://myapp.myserver.com:2000/pls/myapp/ will automatically update the URL to: http://myapp.myserver.com:2000/pls/myapp/myapp.home
Document Table	Enter the name of the database table into which files uploaded to a web site created with will be stored. The default value in this entry field is based on the name of the schema in which you created the site.
Document Access Path	Enter a path in the URL installation that is used to indicate a document is being referenced. In the following URL, for example: http://myapp.myserver.com:2000/pls/my_site/docs/folder1/presentation.htm `docs` is the document access path.
Document Access Procedure	Enter the procedure that will be used to upload and download documents.
Extensions to be Uploaded as LONGRAW	Specify extensions for files to be uploaded as LONGRAW.
Path Alias	To be used by PL/SQL applications for path aliasing. **WebDB 2.X Note** You must leave this field blank if the DAD is for an existing WebDB 2.x Web site.
Path Alias Procedure	To be used by PL/SQL applications for path aliasing. **WebDB 2.X Note** You must leave this field blank if the DAD is for an existing WebDB 2.x Web site.

4

Setting up WebDB to run with mod_plsql

This section is for WebDB users who plan to run WebDB version 2.x (2.0, 2.1, 2.2) through mod_plsql.

4.1 Before You Begin

- Use the latest OWA packages shipped with mod_plsql in your WebDB 2.x database. Re-execute owaload.sql with the proper parameters if you are in doubt. **Note:** This may invalidate some of your existing PL/SQL procedures. You may need to recompile them. See "Installing required packages" on page 2-2 for more information
- Set the following in the DAD configuration for the WebDB 2.x schema in wdbsvr.app configuration file.

  Authentication Mode = Basic

  Document Table = schema.wwv_document

  upload_as_long_raw = *

  If you set up your DAD using the **Add for WebDB 2.x configuration** page (http://<hostname>:<port>/pls/admin_/gateway.htm), these settings will automatically be set.
- To enable WebDB 2.x sites, please connect to the database as the owner of the site and run wwvdocs.sql and wwvdocb.plb. These files are located in the same directory as the owaload.sql and docload.sql files See "Installing required packages" on page 2-2 for more information.

5

Using the PL/SQL Web Toolkit

Before you can use mod_plsql, you must install the packages in the PL/SQL Web Toolkit in a common schema called owa_public in your Oracle database. Public synonyms are used to enable users to execute the objects in the common schema. Users execute the objects in the common schema with their own privileges, rather than with the privileges of the common schema.

If multiple instances of the PL/SQL Web Toolkit are installed in the database, it is recommended that you drop earlier packages from the individual schemas.

5.1 PL/SQL Web Toolkit Installation

If you did not install the PL/SQL Web Toolkit when you installed mod_plsql, you can install it using the owaload .sql installation script. See "Installing required packages" on page 2-2 for more information.

5.2 Packages in the Toolkit

The PL/SQL Web Toolkit contains the following packages:

Package	Description
htf and htp	The htp (hypertext procedures) package contains procedures that generate HTML tags. For instance, the htp.anchor procedure generates the HTML anchor tag, <A>. The htf (hypertext functions) package contains the function version of the procedures in the htp package. The function versions do not directly generate output in your web page. Instead, they pass their output as return values to the statements that invoked them. Usethese functions when you need to nest calls. To print the output of htf functions, call them from within the htp.print procedure, which simply prints its parameter values to the generated web page.

Package	Description
owa	Contains subprograms required by mod_plsql.
owa_content	Contains functions and procedures that let you query the content service repository and manipulate document properties.
owa_sec	Contains subprograms used by mod_plsql for authenticating requests. **Note** This package is included when you install the Toolkit with OAS. mod_plsql does not use it.
owa_util	Contains utility subprograms. It is divided into the following areas: ■ Dynamic SQL utilities enable you to produce pages with dynamically generated SQL code. ■ HTML utilities enable you to retrieve the values of CGI environment variables and perform URL redirects. ■ Date utilities enable correct date-handling. Date values are simple strings in HTML, but should be properly treated as a data type by the Oracle database.
owa_pattern	Contains subprograms that you can use to perform string matching and string manipulation with regular expression functionality.
owa_text	Contains subprograms used by owa_pattern for manipulating strings. They are externalized so you can use them directly
owa_image	Contains subprograms that get the coordinates of where the user clicked on an image. Use this package when you have an imagemap whose destination links invoke a mod_plsql.
owa_cookie	Contains subprograms that enable you to send HTTP cookies to and get them from the client's browser. Cookies are opaque strings sent to the browser to maintain state between HTTP calls. State can be maintained 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_custom package.
owa_opt_lock	Contains subprograms that enable you to impose database optimistic locking strategies, so as to prevent lost updates. Lost updates can occur if a user selects and then attempts to update a row whose values have been changed in the meantime by another user.
owa_custom	Contains the authorize function and the time zone constants used by cookies. **Note** This package is included when you install the Toolkit with OAS. mod_plsql does not use it.

5.2.1 htp and htf packages

The htp and htf packages provide subprograms that enable you to generate HTML tags from your stored procedure. For example, the following commands generate a simple HTML document:

```
create or replace procedure hello AS
BEGIN
    htp.htmlopen;           -- generates <HTML>
    htp.headopen;           -- generates <HEAD>
    htp.title('Hello');     -- generates <TITLE>Hello</TITLE>
    htp.headclose;          -- generates </HEAD>
    htp.bodyopen;           -- generates <BODY>
    htp.header(1, 'Hello'); -- generates <H1>Hello</H1>
    htp.bodyclose;          -- generates </BODY>
    htp.htmlclose;          -- generates </HTML>
END;
```

These packages also provide print procedures (such as htp.print), which writes its argument to the current document. You can use these print procedures to generate non-standard HTML, to display the return value of functions, or to pass hard-coded text that appears in the HTML document as-is. The generated text is passed to mod_plsql, which then sends it to the user's browser.

5.2.2 owa_image package

The owa_image package contains subprograms that get the coordinates of where the user clicked on an image. You use this for image maps that invoke mod_plsql. Your procedure would look something like:

```
create or replace procedure process_image
    (my_img in owa_image.point)
    x integer := owa_image.get_x(my_img);
    y integer := owa_image.get_y(my_img);
begin
    /* process the coordinate */
end
```

5.2.3 owa_opt_lock

The owa_opt_lock package contains subprograms that enable you to impose database optimistic locking strategies, so as to prevent lost updates. Lost updates can occur if a user selects and then attempts to update a row whose values have been changed in the meantime by another user.

mod_plsql cannot use conventional database locking schemes because HTTP is a stateless protocol. The owa_opt_lock package works around this by giving you two ways of dealing with the lost update problem:

- The hidden fields method stores the previous values in hidden fields in the HTML page. When the user requests an update, mod_plsql checks these values against the current state of the database. The update operation is performed only if the values match. To use this method, call the owa_opt_lock.store_values procedure.
- The checksum method stores a checksum rather than the values themselves. To use this method, call the owa_opt_lock.checksum function.

These methods are optimistic. That is, they do not prevent other users from performing updates, but they do reject the current update if an intervening update has occurred.

5.2.4 owa_custom

Note This package is included when you install the Toolkit with OAS. mod_plsql does not use it.

The owa_custom package contains the authorize function and the time zone constants used by cookies. Cookies use expiration dates defined in Greenwich Mean Time (GMT). If you are not on GMT, you can specify your time zone using one of these two constants:

If your time zone is recognized by Oracle, you can specify it directly using dbms_server_timezone. The value for this is a string abbreviation for your time zone. (See *Oracle Server SQL Reference* for a list of recognized time zones. For example, if your time zone is Pacific Standard Time, you can use the following:

```
dbms_server_timezone constant varchar2(3) := 'PST'
```

If your time zone is not recognized by Oracle, use dbms_server_gmtdiff to specify the offset of your time zone from GMT. Specify a positive number if your time zone is ahead of GMT, otherwise use a negative number.

```
dbms_server_gmtdiff constant number := NULL
```

After making the appropriate changes, you need to reload the package.

5.2.5 owa_content

Note This package is included when you install the Toolkit with OAS. mod_plsql does not use it.

The owa_content package contains functions and procedures that let you query the content service repository and manipulate document properties. You can use this package to perform tasks, like:

- set a document description
- delete documents
- delete document attributes
- retrieve attribute information
- list document attributes
- retrieve content type of a document

When compiling PL/SQL procedures and packages that use the owa_content package, you may get the following error message:

```
PLS-00201
identifier 'WEBSYS.OWA_CONTENT' must be declared
```

To avoid this error, when creating a new DAD that uses a non local database, you must enter the SYS username and corresponding password when prompted for a DBA user. Entering the SYSTEM user will not allows the correct grant and rights to be assigned to the database user. If you have entered SYSTEM as the DBA user then you must explicitly perform the grant privilege option as shown below:

```
SQL>grant all on WEBSYS.OWA_CONTENT to scott
```

If you are creating a DAD using an existing database user, you must perform the manual grant privilege shown above before using the OWA_CONTENT package.

The PL/SQL samples use the OWA_CONTENT package; so, these steps must be performed before installing the PL/SQL samples.

5.3 Conventions for Parameter Names in the Toolkit

In the PL/SQL Web Toolkit, the first letter of the parameter name indicates the data type of the parameter:

Table 5–1

First character	Datatype	Example
c	VARCHAR2	cname IN VARCHAR2
n	INTEGER	nsize IN INTEGER
d	DATE	dbuf IN DATE

5.4 HTML Tag attributes

Many HTML tags have a large number of optional attributes that, if passed as individual parameters to the hypertext procedures or functions, would make the calls cumbersome. In addition, some browsers support non-standard attributes. Therefore, each hypertext procedure or function that generates an HTML tag has as its last parameter `cattributes`, an optional parameter. This parameter enables you to pass the exact text of the desired HTML attributes to the PL/SQL procedure.

For example, the syntax for htp.em is:

```
htp.em(ctext, cattributes);
```

A call that uses HTML 3.0 attributes might look like the following:

```
htp.em('This is an example','ID="SGML_ID" LANG="en"');
```

which would generate the following:

```
<EM ID="SGML_ID" LANG="en">This is an example</EM>
```

5.5 mod_plsql and Applets

When you reference an applet using the APPLET tag in an HTML file, the server looks for the applet class file in the directory containing the HTML file. If the applet class file is in another directory, you use the CODEBASE attribute of the APPLET tag to specify that directory.

When you generate an HTML page from mod_plsql and the page references an applet, you must specify the CODEBASE attribute because mod_plsql does not have a concept of a current directory and does not know where to look for the applet class file.

The following example uses htp.appletopen to generate an APPLET tag. It uses the `cattributes` parameter to specify the CODEBASE value.

```
htp.appletopen('myapplet.class', 100, 200, 'CODEBASE="/applets"')
```

generates

```
<APPLET CODE="myapplet.class" height=100 width=200 CODEBASE="/applets">
```

/applets is a virtual path that contains the myapplet.class file.

5.6 Cookies

Cookies can be used to maintain persistent state variables from the client browser:

```
http://home.netscape.com/newsref/std/cookie_spec.html
http://www.virtual.net/Projects/Cookies/
```

The owa_cookie package enables you to send and retrieve cookies in HTTP headers. It contains the following subprograms that you can use to set and get cookie values:

- owa_cookie.cookie data type contains cookie name-value pairs.
- owa_cookie.get function gets the value of the specified cookie.
- owa_cookie.get_all procedure gets all cookie name-value pairs.
- owa_cookie.remove procedure removes the specified cookie.

5.7 LONG Data Type

If you use values of the LONG data type in procedures/functions such as htp.print, htp.prn, htp.prints, htp.ps, or owa_util.cellsprint, be aware that only the first 32K of the LONG data is used. This reason for this limitation is that the LONG data is bound to a varchar2 data type in the procedure/function.

5.8 Extensions to the htp and htf Packages

The htp and htf packages allow you to use customized extensions. Therefore, as the HTML standard changes, you can add new functionality similar to the hypertext procedure and function packages to reflect those changes.

Here is an example of customized packages using non-standard <BLINK> and imaginary <SHOUT>tags:

```
create package nsf as
    function blink(cbuf in varchar2) return varchar2;
    function shout(cbuf in varchar2) return varchar2;
end;

create package body nsf as
    function blink(cbuf in varchar2) return varchar2 is
        begin return ('<BLINK>' || cbuf || '</BLINK>');
    end;
    function shout(cbuf in varchar2) return varchar2 is
        begin return ('<SHOUT>' || cbuf || '</SHOUT>');
    end;
end;

create package nsp as
    procedure blink(cbufin varchar2);
    procedure shout(cbufin varchar2);
end;

create package body nsp as
    procedure blink(cbufin varchar2) is
        begin htp.print(nsf.blink(cbuf));
    end;
    procedure shout(cbufin varchar2) is
        begin htp.print(nsf.shout(cbuf));
    end;
end;
```

Now you can begin to use these procedures and functions in your own procedure.

```
create procedure nonstandard as
begin
    nsp.blink('Gee this hurts my eyes!');
    htp.print('And I might ' || nsf.shout('get mad!'));
end;
```

5.9 String Matching and Manipulation

The owa_pattern package contains procedures and functions that you can use to perform string matching and string manipulation with regular expression functionality. The package provides the following subprograms:

- The owa_pattern.match function determines whether a regular expression exists in a string. It returns TRUE or FALSE.
- The owa_pattern.amatch function is a more sophisticated variation of the owa_pattern.match function. It lets you specify where in the string the match has to occur. This function returns the end of the location in the string where the regular expression was found. If the regular expression is not found, it returns 0.
- The owa_pattern.change function and procedure lets you replace the portion of the string that matched the regular expression with a new string. If you call it as a function, it returns the number of times the regular expression was found and replaced.

These subprograms are overloaded. That is, there are several versions of each, distinguished by the parameters they take. Specifically, there are six versions of MATCH, and four each of AMATCH and CHANGE. The subprograms use the following 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 owa_text.multi_line data type.
- pat - This is the pattern that the subprograms attempt to locate in line. The pattern can contain regular expressions. Note in the
- owa_pattern.change function and procedure, this parameter is called from_str.
- flags - This specifies whether the search is case-sensitive or if substitutions are to be done globally.

5.10 owa_pattern.match

The regular expression in this function can be either a VARCHAR2 or a owa_pattern.pattern data type. You can create a owa_pattern.pattern data type from a string using the owa_pattern.getpat procedure.

You can create a multi_line data type from a long string using the owa_text.stream2multi procedure. If a multi_line is used, the rlist parameter specifies a list of chunks where matches were found.

If the line is a string and not a multi_line, you can add an optional output parameter called backrefs. This parameter is a row_list that holds each string in the target that was matched by a sequence of tokens in the regular expression. Here is an example of the owa_pattern.match function:

```
boolean foundMatch;
foundMatch := owa_pattern.match('KAZOO', 'zoo.*', 'i');
```

This is how the function works: KAZOO is the target where it is searching for the `zoo.*` regular expression. The period indicates any character other than newline, and the asterisk matches 0 or more of the preceding characters. In this case, it matches any character other than the newline.

Therefore, this regular expression specifies that a matching target consists of `zoo`, followed by any set of characters neither ending in nor including a newline (which does not match the period). The i is a flag indicating that case is to be ignored in the search. In this case, the function returns TRUE, which indicates that a match had been found.

5.11 owa_pattern.change

owa_pattern.change can be a procedure or a function, depending on how it is invoked. As a function, it returns the number of changes made. If the flag 'g' is not used, this number can only be 0 or 1. The flag 'g' specifies that all matches are to be replaced by the regular expression. Otherwise, only the first match is replaced.

The replacement 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:

```
owa_pattern.change('Cats in pajamas', 'C.+in', '& red ')
```

The regular expression matches the substring 'Cats in'. It then replaces this string with '& red'. The ampersand character, &, indicates 'Cats in', since that's what matched the regular expression. Thus, this procedure replaces the string 'Cats in pajamas' with 'Cats in red'. If you called this as a function instead of a procedure, the value it would return would not be 'Cats in red' but 1, indicating that a single substitution had been made.

6

mod_plsql Tutorial

This section provides a step-by-step guide on creating and invoking a simple application that displays the contents of a database table as an HTML table. The application consists of one PL/SQL cartridge. The cartridge invokes a stored procedure that calls functions and procedures defined in the PL/SQL Web Toolkit.

This tutorial assumes the following:

- You have completed the section, "Installing required packages" on page 2-2.
- You can log in as the admin user on the server. This is required because you will be adding new settings to the server configuration. The database to which you will be connecting already has the PL/SQL Web Toolkit installed. See "PL/SQL Web Toolkit Installation" on page 5-13 for more information.
- You have the SCOTT schema in your Oracle database. The PL/SQL cartridge logs into the database using scott/tiger as the username and password. If you do not have the SCOTTschema, you can use an existing schema on your database, or you can create SCOTT using the CREATE SCHEMA command.

  A schema is a user account containing as a collection of database objects such as tables, views, procedures, and functions. Each object in the schema can access other objects in the same schema.

6.1 Creating and Loading the Stored Procedure onto the Database

The stored procedure that the application invokes is current_users (defined below). The procedure retrieves the contents of the all_users table and formats it as an HTML table.

To create the stored procedure, save the text of the procedure in a file called current_users.sql, and then run Oracle Server Manager to read and execute the statements in the file.

1. Type the following lines and save it in a file called current_users.sql. The current_users procedure retrieves the contents of the all_users table and formats it as an HTML table.

```
create or replace procedure current_users
    AS
        ignore boolean;
    BEGIN
        htp.htmlopen;
        htp.headopen;
        htp.title('Current Users');
        htp.headclose;
        htp.bodyopen;
        htp.header(1, 'Current Users');
        ignore := owa_util.tablePrint('all_users');
        htp.bodyclose;
        htp.htmlclose;
    END;
    /
    show errors
```

   This procedure uses functions and procedures from the htp and owa_util packages to generate the HTML page. For example, the htp.htmlopen procedure generates the string <html>, and htp.title('Current Users') generates <title>Current Users</title>.

   The owa_util.tablePrint function queries the specified database table, and formats the contents as an HTML table.

2. Start up Server Manager in line mode. ORACLE_HOME is the directory that contains the Oracle database files.

```
prompt> $ORACLE_HOME/bin/svrmgrl
```

3. Connect to the database as "scott". The password is "tiger".

```
SVRMGR> connect scott/tiger
```

4. Load the current_users stored procedure from the current_users.sql file. You need to provide the full path to the file if you started up Server Manager from a directory different than the one containing the current_users.sql file.

```
SVRMGR> @  Name of script file: current_users.sql
```

5. Exit Server Manager.

```
SVRMGR> exit
```

6. Configure a DAD to point to the schema where PL/SQL applications that you want to run with mod_plsql are stored, with the parameters shown in the following table:

Table 6–1

Parameter	Value
Database Access Descriptor Name	Scott
Schema	Scott
Oracle User Name	Scott
Oracle Password	Tiger
Oracle Connect String	htmlperf-tcp
Authentication Mode	Basic
Session Cookie Name	
Create a Stateful Session?	No
Keep Database Connections Open Between Requests	Yes
Maximum Number of Worker Threads	10
Default (Home) Page	Scott.home
Document Table	Scott.wwdoc_document
Document Access Path	docs
Document Access Procedure	Scott.wpg_testdoc.process_download
Extensions to be Uploaded as LONGRAW	*
Path Alias	
Path Alias Procedure	

Notes

- You need to configure only one DAD per schema.
- If you want require a user to log on to the database containing the application, leave the **Oracle User Name** and **Oracle Password** fields blank.

6.2 Creating an HTML Page to Invoke the Application

To run the current_users procedure, enter the following URL in your browser:

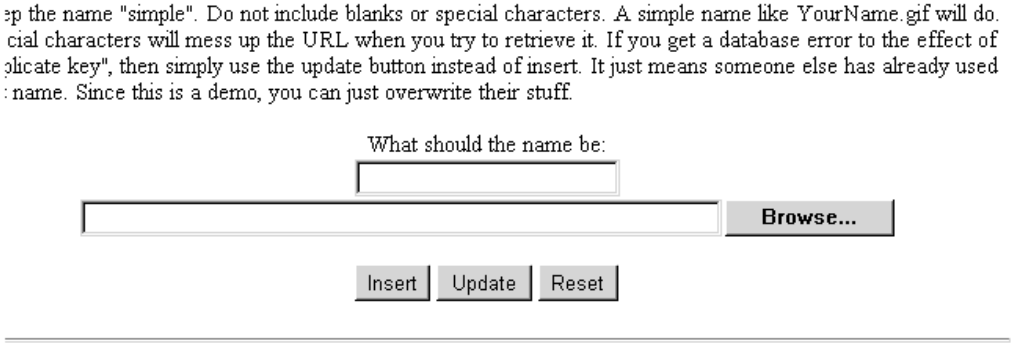
```
http://<host>:<port>//pls/mydad/scott.current_users
```

It is more common, however, to invoke the procedure from an HTML page. For example, the following HTML page has a link that calls the URL.

```
<HTML>
<HEAD>
<title>Current Users</title>
</HEAD>

<BODY>
<H1>Current Users</H1>
<p><a href="http://hal.us.oracle.com:9999/simpleApp1/cart1/current_
users">Run
current_users</a>
</BODY>
</HTML>
```

The figure below shows the source page (the page containing the link that invokes the stored procedure), and the page that is generated by the current_users stored procedure.



Index

H

I

L

M

N

O

P

R

S

T

U

W