Oracle9*i*AS Single Sign-On

Application Developer's Guide

Release 2 (9.0.2)

January 2002

Part No. A96114-01

ORACLE

Oracle9*i*AS Single Sign-On Application Developer's Guide, Release 2 (9.0.2)

Part No. A96114-01

Primary Author: Henry Abrecht

Graphic Artist: Valarie Moore

Contributors: Kamalendu Biswas

Contents

List of Figures

List of Tables

Send Us Your Comments

Oracle9*i*AS Single Sign-On Application Developer's Guide, Release 2 (9.0.2)

Part No. A96114-01

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, please 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 ways:

- Electronic mail: infodev_us@oracle.com
- FAX: (650) 506-7227 Attn: Server Technologies Documentation Manager
- Postal service:
  Oracle Corporation
  Server Technologies Documentation
  500 Oracle Parkway, Mailstop 4op11
  Redwood Shores, CA 94065
  USA

If you would like a reply, please give your name, address, telephone number, and (optionally) electronic mail address.

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

Preface

Oracle9iAS Single Sign-On Application Developer's Guide is directed at those developers who modify applications for Oracle Single Sign-On. This modification is accomplished using either the HTTP module mod_osso or the Oracle9*i*AS Single Sign-On Software Development Kit (SDK).

This preface contains these topics:

Audience

This document assumes the following knowledge or capabilities:

- access to a working copy of Oracle9*i*AS, Release 2, or the ability to install one
- an understanding of Oracle9*i*AS concepts
- proficiency in the PL/SQL or Java programming language

Organization

This document contains:

Chapter 1, "Introduction"

This chapter introduces mod_osso and the Single Sign-On SDK. It provides a brief description of other Single Sign-On components.

Chapter 2, "Developing Applications Using Mod_osso"

This chapter introduces the HTTP authentication module mod_osso, explaining how it is used to protect applications enabled by Oracle Single Sign-On. The chapter provides code that demonstrates how applications are integrated with mod_osso.

Chapter 3, "Single Sign-On Software Development Kit"

This chapter lists and describes the PL/SQL application programming interfaces (APIs) for integrating applications with Oracle Single Sign-On. The SDK also contains Java APIs, which can be found at Oracle Technology Network. The chapter explains how SDK authentication works.

Chapter 4, "Using the PL/SQL and Java APIs"

This chapter explains how to write partner applications using PL/SQL and Java. Code examples are provided for both languages.

Related Documentation

For more information, see:

- *Oracle9iAS Single Sign-On Administrator's Guide*
- *Oracle9iAS Single Sign-On API Reference (Javadoc)*

In North America, printed documentation is available for sale in the Oracle Store at

`http://oraclestore.oracle.com/`

Customers in Europe, the Middle East, and Africa (EMEA) can purchase documentation from

`http://www.oraclebookshop.com/`

Other customers can contact their Oracle representative to purchase printed documentation.

To download free release notes, installation documentation, white papers, or other collateral, please visit the Oracle Technology Network (OTN). You must register online before using OTN; registration is free and can be done at

`http://otn.oracle.com/admin/account/membership.html`

If you already have a username and password for OTN, then you can go directly to the documentation section of the OTN Web site at

`http://otn.oracle.com/docs/index.htm`

To access the database documentation search engine directly, please visit

`http://tahiti.oracle.com`

Conventions

This section describes the conventions used in the text and code examples of this documentation set. It describes:

- Conventions in Text
- Conventions in Code Examples
- Conventions for Microsoft Windows Operating Systems

Conventions in Text

We use various conventions in text to help you more quickly identify special terms. The following table describes those conventions and provides examples of their use.

Convention	Meaning	Example
Bold	Bold typeface indicates terms that are defined in the text or terms that appear in a glossary, or both.	When you specify this clause, you create an **index-organized table**.
Italics	Italic typeface indicates book titles or emphasis.	*Oracle9i Database Concepts* Ensure that the recovery catalog and target database do *not* reside on the same disk.
`UPPERCASE monospace (fixed-width) font`	Uppercase monospace typeface indicates elements supplied by the system. Such elements include parameters, privileges, datatypes, RMAN keywords, SQL keywords, SQL*Plus or utility commands, packages and methods, as well as system-supplied column names, database objects and structures, usernames, and roles.	You can specify this clause only for a `NUMBER` column. You can back up the database by using the `BACKUP` command. Query the `TABLE_NAME` column in the `USER_TABLES` data dictionary view. Use the `DBMS_STATS.GENERATE_STATS` procedure.
`lowercase monospace (fixed-width) font`	Lowercase monospace typeface indicates executables, filenames, directory names, and sample user-supplied elements. Such elements include computer and database names, net service names, and connect identifiers, as well as user-supplied database objects and structures, column names, packages and classes, usernames and roles, program units, and parameter values. **Note:** Some programmatic elements use a mixture of UPPERCASE and lowercase. Enter these elements as shown.	Enter `sqlplus` to open SQL*Plus. The password is specified in the `orapwd` file. Back up the datafiles and control files in the `/disk1/oracle/dbs` directory. The `department_id`, `department_name`, and `location_id` columns are in the `hr.departments` table. Set the `QUERY_REWRITE_ENABLED` initialization parameter to `true`. Connect as `oe` user. The `JRepUtil` class implements these methods.
`lowercase italic monospace (fixed-width) font`	Lowercase italic monospace font represents placeholders or variables.	You can specify the *`parallel_clause`*. Run `U`*`old_release`*`.SQL` where *`old_release`* refers to the release you installed prior to upgrading.

Conventions in Code Examples

Code examples illustrate SQL, PL/SQL, SQL*Plus, or other command-line statements. They are displayed in a monospace (fixed-width) font and separated from normal text as shown in this example:

```
SELECT username FROM dba_users WHERE username = 'MIGRATE';
```

The following table describes typographic conventions used in code examples and provides examples of their use.

Convention	Meaning	Example
[ ]	Brackets enclose one or more optional items. Do not enter the brackets.	`DECIMAL (`*`digits`*` [ , `*`precision`*` ])`
{ }	Braces enclose two or more items, one of which is required. Do not enter the braces.	`{ENABLE | DISABLE}`
|	A vertical bar represents a choice of two or more options within brackets or braces. Enter one of the options. Do not enter the vertical bar.	`{ENABLE | DISABLE}` `[COMPRESS | NOCOMPRESS]`
. . .	Horizontal ellipsis points indicate either: ■ That we have omitted parts of the code that are not directly related to the example ■ That you can repeat a portion of the code	`CREATE TABLE ... AS `*`subquery`*`;` `SELECT `*`col1`*`, `*`col2`*`, ... , `*`coln`*` FROM employees;`
. . .	Vertical ellipsis points indicate that we have omitted several lines of code not directly related to the example.	
Other notation	You must enter symbols other than brackets, braces, vertical bars, and ellipsis points as shown.	`acctbal NUMBER(11,2);` `acct CONSTANT NUMBER(4) := 3;`
Italics	Italicized text indicates placeholders or variables for which you must supply particular values.	`CONNECT SYSTEM/`*`system_password`* `DB_NAME = `*`database_name`*

UPPERCASE	Uppercase typeface indicates elements supplied by the system. We show these terms in uppercase in order to distinguish them from terms you define. Unless terms appear in brackets, enter them in the order and with the spelling shown. However, because these terms are not case sensitive, you can enter them in lowercase.	`SELECT last_name, employee_id FROM employees;`
		`SELECT * FROM USER_TABLES;`
		`DROP TABLE hr.employees;`

Convention	Meaning	Example

`lowercase`	Lowercase typeface indicates programmatic elements that you supply. For example, lowercase indicates names of tables, columns, or files.
	
	Note: Some programmatic elements use a mixture of UPPERCASE and lowercase. Enter these elements as shown.
		`SELECT last_name, employee_id FROM employees;`
		
		`sqlplus hr/hr`
		
		`CREATE USER mjones IDENTIFIED BY ty3MU9;`

Conventions for Microsoft Windows Operating Systems

The following table describes conventions for Microsoft Windows operating systems and provides examples of their use.

Convention	Meaning	Example
Choose Start >	How to start a program.	To start the Oracle Database Configuration Assistant, choose Start > Programs > Oracle - *HOME_NAME* > Configuration and Migration Tools > Database Configuration Assistant.
File and directory names	File and directory names are not case sensitive. The following special characters are not allowed: left angle bracket (<), right angle bracket (>), colon (:), double quotation marks ("), slash (/), pipe (|), and dash (-). The special character backslash (\) is treated as an element separator, even when it appears in quotes. If the file name begins with \\, then Windows assumes it uses the Universal Naming Convention.	`c:\winnt"\"system32` is the same as `C:\WINNT\SYSTEM32`
`C:\>`	Represents the Windows command prompt of the current hard disk drive. The escape character in a command prompt is the caret (^). Your prompt reflects the subdirectory in which you are working. Referred to as the *command prompt* in this manual.	`C:\oracle\oradata>`

Convention	Meaning	Example

	The backslash (\) special character is sometimes required as an escape character for the double quotation mark (") special character at the Windows command prompt. Parentheses and the single quotation mark (') do not require an escape character. Refer to your Windows operating system documentation for more information on escape and special characters.	`C:\>exp scott/tiger TABLES=emp QUERY=\"WHERE job='SALESMAN' and sal<1600\"`
		`C:\>imp SYSTEM/`*`password`*
		`FROMUSER=scott TABLES=(emp, dept)`

HOME_NAME	Represents the Oracle home name. The home name can be up to 16 alphanumeric characters. The only special character allowed in the home name is the underscore.	`C:\> net start Oracle`*`HOME_NAME`*`TNSListener`

Convention	Meaning	Example
ORACLE_HOME and *ORACLE_BASE*	In releases prior to Oracle8*i* release 8.1.3, when you installed Oracle components, all subdirectories were located under a top level *ORACLE_HOME* directory that by default used one of the following names: ■ `C:\orant` for Windows NT ■ `C:\orawin95` for Windows 95 ■ `C:\orawin98` for Windows 98 This release complies with Optimal Flexible Architecture (OFA) guidelines. All subdirectories are not under a top level *ORACLE_HOME* directory. There is a top level directory called *ORACLE_BASE* that by default is `C:\oracle`. If you install Oracle9*i* release 1 (9.0.1) on a computer with no other Oracle software installed, then the default setting for the first Oracle home directory is `C:\oracle\ora90`. The Oracle home directory is located directly under *ORACLE_BASE*. All directory path examples in this guide follow OFA conventions. Refer to *Oracle9i Database Getting Starting for Windows* for additional information about OFA compliances and for information about installing Oracle products in non-OFA compliant directories.	Go to the *ORACLE_BASE*`\`*ORACLE_HOME*`\rdbms\admin` directory.

Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible, with good usability, to the disabled community. To that end, our documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Standards will continue to evolve over time, and Oracle Corporation is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be accessible to all of our customers. For additional information, visit the Oracle Accessibility Program Web site at

`http://www.oracle.com/accessibility/`

Accessibility of Code Examples in Documentation JAWS, a Windows screen reader, may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, JAWS may not always read a line of text that consists solely of a bracket or brace.

Accessibility of Links to External Web Sites in Documentation This documentation may contain links to Web sites of other companies or organizations that Oracle Corporation does not own or control. Oracle Corporation neither evaluates nor makes any representations regarding the accessibility of these Web sites.

1

Introduction

Oracle9*i*AS Single Sign-On is a component of Oracle9*i* Application Server (*i*AS) that enables users to log in to all features of the *i*AS product complement, as well as to other Web applications, using a single user name and password.

In Oracle9*i*AS, Release 2, application developers can use either mod_osso, an Oracle HTTP authentication module, or the Single Sign-On Software Development Kit (SDK) to enable applications for Single Sign-On. This chapter introduces these two authentication options. The remainder of the book explains how to integrate applications with mod_osso and with the SDK.

The chapter covers the following topics:

What Is Mod_osso?

Mod_osso is an HTTP module that provides authentication to *i*AS applications. It is a simpler alternative to the Single Sign-On SDK, used in earlier releases of Oracle Single Sign-On to integrate partner applications. Mod_osso simplifies the authentication process by serving as the sole partner application to the Single Sign-On server, rendering authentication transparent for *i*AS applications. Thus, the administrator for these applications is spared the burden of integrating them with the Single Sign-On SDK.

Mod_osso can be registered with the Single Sign-On server when *i*AS is installed.

What's in the Single Sign-On SDK

The Single Sign-On SDK consists of the following components:

- PL/SQL Application Programming Interface
- Java Application Programming Interface
- Sample code for implementing the PL/SQL and Java APIs, specifically procedures for constructing the application URL and retrieving the URLC token. The Java code includes examples of servlet and JSP partner applications

Other Components of Oracle Single Sign-On

- Oracle Single Sign-On Server (9*i*AS, Release 2)

  Program logic in the *i*AS database that enables users to log in securely to Single-Sign-On-enabled applications such as expense reports, mail, and benefits

- Partner Applications

  *i*AS applications that delegate the authentication function to the Single Sign-On server

- External Applications

  Web applications that do not delegate authentication to the Single Sign-On server. Instead, they display HTML login forms that ask for application user names and passwords

> **See Also:** Chapter 1, "Single Sign-On Basics" in *Oracle9iAS Single Sign-On Administrator's Guide*

2

Developing Applications Using Mod_osso

This chapter explores the role that the HTTP authentication module mod_osso plays in Oracle Single Sign-On. In addition, it explains how to develop applications to work with mod_osso.

The chapter covers the following topics:

Why Mod_osso?

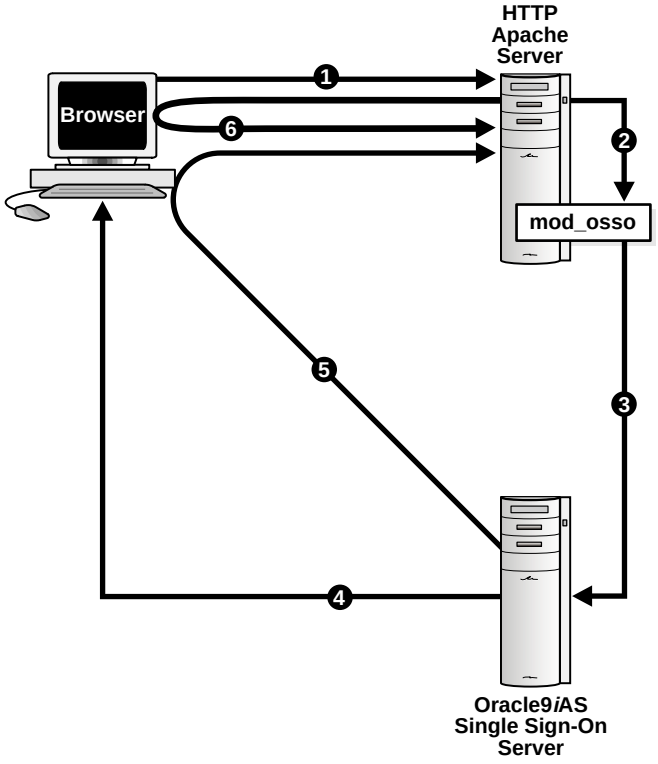
In earlier releases of Oracle Single Sign-On, applications were expected to integrate with the Single Sign-On server using the Single Sign-On SDK. The SDK gave them full control over their interaction with the server, but it exacted a setup cost for registering an application. In Oracle9*i*AS, Release 2, an HTTP module called mod_osso, not the application, provides client-side Single Sign-On functionality.

Mod_osso eases Single Sign-On integration, but it also takes away the granular control that applications have when they use the SDK. Fortunately, a new communication mechanism between applications and mod_osso restores this granular control. This mechanism is called the dynamic directive.

How Mod_osso Works

Figure 2–1 on page 2-3 shows what happens when a user requests a URL protected by mod_osso. To understand how the process differs from SDK-enabled authentication, see "Authentication Flow in SDK-Enabled Single Sign-On" in Chapter 3, "Single Sign-On Software Development Kit."

Figure 2–1 Single Sign-On with Mod_osso



1. The user requests a URL through a Web browser
2. The Web server looks for a mod_osso cookie for the user. If the cookie exists, the Web server extracts the user's information and uses it to log the user in to the requested application.
3. If the cookie does not exist, mod_osso redirects the user to the Single Sign-On server.
4. The Single Sign-On server looks for its own cookie in the browser. If it finds none, it tries to authenticate the user with a user name and password. If authentication is successful, the Single Sign-On server creates a cookie in the browser as a reminder that the user has been authenticated. If a cookie exists, the Single Sign-On server will authenticate using the cookie.

5. The Single Sign-On server returns the user's encrypted information to mod_osso.
6. Mod_osso creates its own cookie for the user in the browser and redirects the user to the requested URL.

If, during the same session, the user again seeks access to the same or to a different application, the user is not prompted for a user name and password; the application uses an HTTP header to obtain this information from the mod_osso session cookie.

Mod_osso redirects the user to the Single Sign-On server only if the requested URL is protected. Some protected applications may have public URLs. These require no redirection to the Single Sign-On server.

About Dynamic Directives

Dynamic directives are HTTP response headers that have special error codes that enable an application to request granular functionality from the Single Sign-On system without having to implement the intricacies of the Single Sign-On protocol. Upon receiving a directive as part of a simple HTTP response from the application, mod_osso creates the appropriate Single Sign-On protocol message and communicates it to the Single Sign-On server.

Oracle9*i*AS, Release 2, supports dynamic directives for Java servlets and JSPs. The Oracle stack for supporting these applications consists of the OC4J engine and a corresponding HTTP module, mod_oc4j, which communicates Web requests to the OC4J engine.

How Dynamic Directives Work

When an application protected by mod_osso wants the Single Sign-On server to perform some action—for example, authentication or single sign-off—it creates an HTTP response whose header contains a dynamic directive status code understood by mod_osso. In addition, the application might want to provide values that are relevant to that directive. The status code for the login directive, for example, is 401. In addition to this code, the directive might contain the header "Osso_Paranoid", "false", indicating a simple authentication request. The response header might look like this:

```
HTTP/1.1   499  Oracle SSO
Osso_Paranoid: false
```

When mod_oc4j looks at an HTTP response and determines that it is a dynamic directive, it passes this directive to mod_osso. A simple response, on the other hand, is passed directly to the HTTP core. The directive handler function for mod_osso constructs the appropriate Single Sign-On protocol message indicating to the Single Sign-On server whether the application is requesting simple dynamic authentication, forced authentication, global sign-off, and so on. The handler then constructs a redirect response and passes it to the HTTP core, which passes it to the Web client. The process is illustrated in Figure 2–2. The black flow lines denote dynamic directives; the white lines simple directives.

Figure 2–2 HTTP Response Path for Dynamic Directives

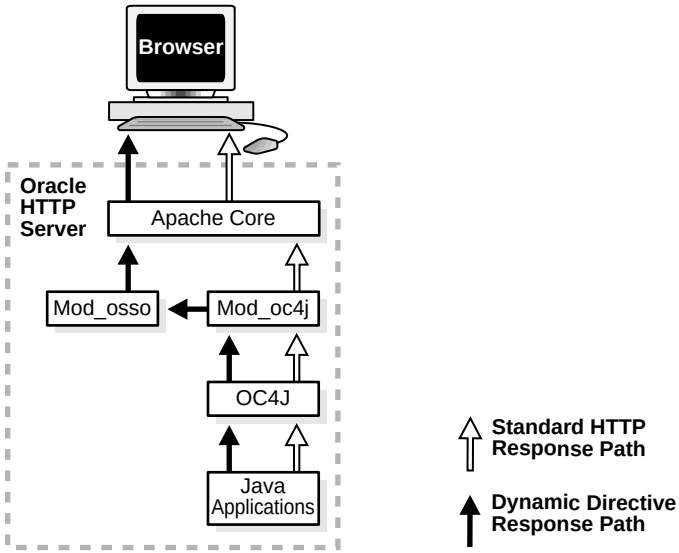


Table 2–1 on page 2-6 lists commonly requested dynamic directives.

Table 2–1 Commonly Requested Dynamic Directives

Directive	Status Code	Headers
Request Authentication	401, 499	`Osso-Paranoid: false` Note that this header need not be set at all
Request Forced Authentication	499	`Osso-Paranoid: true`
Single Sign-Off	470	`Done-url` This is the URL to return to after single sign-off is complete

Developing Applications Using Mod_osso

This section explains how to write and enable applications using mod_osso. The section covers the following topics:

- PL/SQL Applications
- Java Applications
- Java Applications That Use Dynamic Directives

PL/SQL Applications

Use the following steps to write and enable a PL/SQL application using mod_osso:

1. Create a database access descriptor (DAD) for the application in the dads.conf file, as follows:

```
<Location /pls/DAD_name>
  SetHandler pls_handler
  Order deny,allow
  AllowOverride None
  PlsqlDatabaseConnectString     hostname:port:SID
  PlsqlDatabasePassword          schema_password
  PlsqlDatabaseUsername          schema_name
  PlsqlDefaultPage               schema.home
  PlsqlDocumentTablename         schema.wwdoc_document
  PlsqlDocumentPath              docs
  PlsqlDocumentProcedure         schema.wwdoc_process.process_download
  PlsqlAuthenticationMode        Basic
  PlsqlPathAlias                 url
  PlsqlPathAliasProcedure        schema.wwpth_api_alias.process_download
  PlsqlSessionCookieName         schema
  PlsqlCGIEnvironmentList   OSSO-USER-DN,OSSO-USER-GUID,OSSO-SUBSCRIBER,OSSO
-SUBSCRIBER-DN,OSSO-SUBSCRIBER-GUID
</Location>
```

2. Protect the application DAD by entering the following lines in the mod_osso.conf file:

```
<Location /pls/DAD>
  require valid-user
  authType Basic
</Location>
```

Note: The assumption here is that mod_osso is already configured for Single Sign-On. This step can be performed when *i*AS is installed.

3. Write application functions and procedures in the application schema associated with the DAD.

   What follows is an example of a simple mod_osso-protected application. This application logs the user in to the Single Sign-On server, displays user

information, and then logs the user out of both the application and Single Sign-On.

```
create or replace procedure show_user_info
 is
 begin
    begin
        htp.init;
     exception
         when others then null;
     end;
     htp.htmlOpen;
     htp.bodyOpen;
     htp.print('<h2>Welcome to Oracle Single Sign-On</h2>');
     htp.print('<pre>');
     htp.print('Remote user: '
        || owa_util.get_cgi_env('REMOTE_USER'));
     htp.print('User DN: '
        || owa_util.get_cgi_env('Osso-User-Dn'));
     htp.print('User Guid: '
        || owa_util.get_cgi_env('Osso-User-Guid'));
     htp.print('Subscriber: '
        || owa_util.get_cgi_env('Osso-Subscriber'));
     htp.print('Subscriber DN: '
        || owa_util.get_cgi_env('Osso-Subscriber-Dn'));
     htp.print('Subscriber Guid: '
        || owa_util.get_cgi_env('Osso-Subscriber-Guid'));
     htp.print('</pre>');
     htp.print('<a href=/osso_logout?'
        ||'p_done_url=http://my.oracle.com>Logout</a>');

     htp.bodyClose;
     htp.htmlClose;
end show_user_info;
/
show errors;

grant execute on show_user_info to public;
```

4. To test whether the newly created functions and procedures are protected by mod_osso, try to access them from a browser:

   `http://apache_host:port/pls/DAD. [`*Function* | *Procedure*`]`

Selecting the URL should invoke the Single Sign-On login page if the mod_osso.conf file has been configured properly and mod_osso is registered with the Single Sign-On server.

Java Applications

Use the following steps to write and enable a servlet or JSP application using mod_osso:

1. Write the JSP or servlet. Like the PL/SQL application example immediately preceding, the simple servlet that follows logs the user in, displays user information, and then logs the user out.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

/**
 * Example servlet showing how to get the SSO User information
 */

public class SSOProtected extends HttpServlet
{

    public void service(HttpServletRequest request,
                        HttpServletResponse response)
        throws IOException, ServletException
    {
        response.setContentType("text/html");

        // Show authenticated user information
        PrintWriter out = response.getWriter();
        out.println("<h2>Welcome to Oracle Single Sign-On</h2>");
        out.println("<pre>");
        out.println("Remote user: "
            + request.getRemoteUser());
        out.println("Osso-User-Dn: "
            +  request.getHeader("Osso-User-Dn"));
        out.println("Osso-User-Guid: "
            +  request.getHeader("Osso-User-Guid"));
        out.println("Osso-Subscriber: "
            +  request.getHeader("Osso-Subscriber"));
        out.println("Osso-User-Dn: "
            +  request.getHeader("Osso-User-Dn"));
        out.println("Osso-Subscriber-Dn: "
```

```
        +  request.getHeader("Osso-Subscriber-Dn"));
    out.println("Osso-Subscriber-Guid: "
        +  request.getHeader("Osso-Subscriber-Guid"));
    out.println("Lang/Territory: "
        + request.getHeader("Accept-Language"));
    out.println("</pre>");
    out.println("<a href=/osso_logout?"
        +"p_done_url=http://my.oracle.com>Logout</a>");
```

2. Protect the servlet by entering the following lines in the mod_osso.conf file:

```
<Location/servlet>
   require valid-user
   authType Basic
</Location>
```

3. Test the servlet by trying to access it from the browser. As in the PL/SQL example, selecting the URL should invoke the Single Sign-On Login page.

   The process is this: when users try to access the servlet from the browser, they are redirected to the Single Sign-On server for authentication. Next they are redirected back to the servlet, which displays user information. Users may then select the logout link to log out of the application as well as the Single Sign-On server.

Java Applications That Use Dynamic Directives

Applications that use dynamic directives require no entry in the mod_osso.conf file because mod_osso protection is written directly into the application as one or more dynamic directives. The servlets that follow show how such directives are incorporated. Like their "static" counterparts, these "dynamic" applications generate user information.

This section covers the following topics:

- Java Example #1: Simple Authentication
- Java Example #2: Single Sign-Off
- Java Example #3: Forced Authentication

Java Example #1: Simple Authentication

This servlet uses the function `request.getRemoteUser()` to check the mod_osso cookie for the user name. If the user name is absent. It issues dynamic directive 499, a request for simple authentication. The key lines are in boldface.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

/**
 * Example servlet showing how to use
 * Dynamic Directive for login
 */

public class SSODynLogin extends HttpServlet
{

    public void service(HttpServletRequest request,
                        HttpServletResponse response)
        throws IOException, ServletException
    {
        String l_user     = null;

        // Try to get the authenticate user name
        try
        {
            l_user = request.getRemoteUser();
        }
        catch(Exception e)
        {
            l_user = null;
        }

        // If user is not authenticated then generate
        // dynamic directive for authentication
        if((l_user == null) || (l_user.length() <= 0) )
        {
            response.setHeader( "Osso-Paranoid", "false" );
            response.sendError(499, "Oracle SSO");
        }
        else
        {
            // Show authenticated user information
            response.setContentType("text/html");
            PrintWriter out = response.getWriter();
```

```
        out.println("<h2>Welcome to Oracle Single Sign-On</h2>");
        out.println("<pre>");
        out.println("Remote user: "
            + request.getRemoteUser());
        out.println("Osso-User-Dn: "
            +  request.getHeader("Osso-User-Dn"));
        out.println("Osso-User-Guid: "
            +  request.getHeader("Osso-User-Guid"));
        out.println("Osso-Subscriber: "
            +  request.getHeader("Osso-Subscriber"));
        out.println("Osso-User-Dn: "
            +  request.getHeader("Osso-User-Dn"));
        out.println("Osso-Subscriber-Dn: "
            +  request.getHeader("Osso-Subscriber-Dn"));
        out.println("Osso-Subscriber-Guid: "
            +  request.getHeader("Osso-Subscriber-Guid"));
        out.println("Lang/Territory: "
            + request.getHeader("Accept-Language"));
        out.println("</pre>");
    }
}
```

Note: If Oracle JAAS Provider is used, the directive code 401 may be substituted for 499.

Java Example #2: Single Sign-Off

This servlet is invoked when users select the login link within an application. The application sets the URL to return to when sign-off is complete; then it issues a directive that sends users to the Single Sign-off page. The key lines are in boldface.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

/**
 * Example servlet showing how to use
 * Dynamic Directive for logout
 */

public class SSODynLogout extends HttpServlet
{
    public void service (HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException
    {
        // Set the return URL
        response.setHeader("Osso-Return-Url",
               "http://my.oracle.com" );
        // Send Dynamic Directive for logout
        response.sendError(470, "Oracle SSO");
    }
}
```

Java Example #3: Forced Authentication

If logged-in users have exceeded the global user inactivity timeout for a given Single Sign-On session, an application can force them to reauthenticate. The directive for reauthentication is written into the servlet that follows. If the user requesting access to this application is an administrator, the directive is not invoked. The key lines are in boldface.

Note: Before testing this application, make sure that the Single Sign-On server and mod_osso are configured for the global user inactivity timeout. To learn how to configure this timeout, see "Configuring the Global User Inactivity Timeout", in Chapter 2 of *Oracle9iAS Single Sign-On Administrator's Guide*.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

/**
 * Example servlet showing how to use
 * Dynamic Directive for forced login
 */

public class SSODynForcedLogin extends HttpServlet
{

    public void service(HttpServletRequest request,
                        HttpServletResponse response)
        throws IOException, ServletException
    {
        String l_user = null;
        // Try to get the authenticate user name
        try
        {
            l_user = request.getRemoteUser();
        }
        catch(Exception e)
        {
            l_user = null;
        }

        // If the user is authenticated then show
        // user information; otherwise generate Dynamic
        // Directive for forced login
        if(l_user != null)
        {
            // Show authenticated user information
            PrintWriter out = response.getWriter();
            response.setContentType("text/html");
            out.println("<h2>Welcome to Oracle Single Sign-On.</h2>");
            out.println("<pre>");
            out.println("Remote user: "
                + request.getRemoteUser());
            out.println("Osso-User-Dn: "
                +  request.getHeader("Osso-User-Dn"));
            out.println("Osso-User-Guid: "
                +  request.getHeader("Osso-User-Guid"));
            out.println("Osso-Subscriber: "
                +  request.getHeader("Osso-Subscriber"));
```

```
                out.println("Osso-User-Dn: "
                    +  request.getHeader("Osso-User-Dn"));
                out.println("Osso-Subscriber-Dn: "
                    +  request.getHeader("Osso-Subscriber-Dn"));
                out.println("Osso-Subscriber-Guid: "
                    +  request.getHeader("Osso-Subscriber-Guid"));
                out.println("Lang/Territory: "
                    + request.getHeader("Accept-Language"));
                out.println("</pre>");
            }
            else
            {
                response.setHeader( "Osso-Paranoid", "true" );
                response.sendError(499, "Oracle SSO");
            }
        }
    }
```

Enabling IP Checking for Mod_osso

Mod_osso uses the directive OssoIPCheck to verify that the user who authenticates to the Single Sign-On server is the same user who is accessing a mod_osso-protected application.

To enable mod_osso for IP checks, the directive OssoIPCheck in the httpd.conf file must be set to on. Note, however, that OssoIPCheck should be enabled only if a browser can communicate with both the Single Sign-On server and the HTTP server without using proxy servers. If proxies are used, and the directive OssoIpCheck is enabled, an error message might be displayed. This is because proxies might not use static IP addresses.

3

Single Sign-On Software Development Kit

The Oracle9*i*AS Single Sign-On Software Development Kit (SDK) consists of application programming interfaces (APIs) for PL/SQL and Java. These APIs are used to create partner applications—that is, applications enabled for Single Sign-On. Chapter 4, "Using the PL/SQL and Java APIs", provides code that shows how the APIs might be implemented.

The chapter covers the following topics:

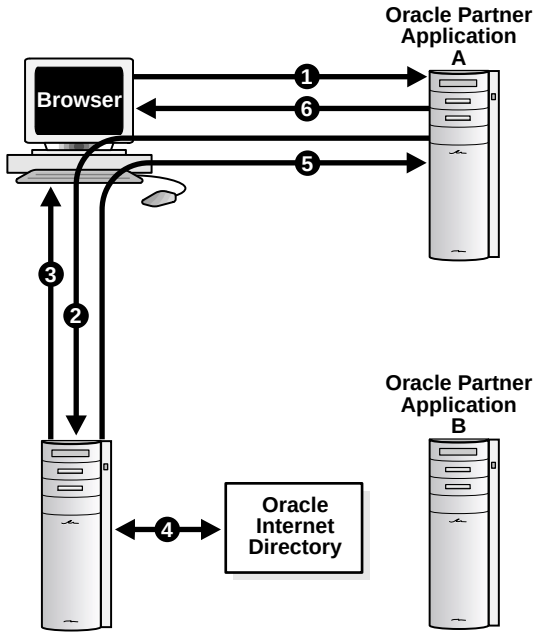
- Authentication Flow in SDK-Enabled Single Sign-On
- PL/SQL APIs
- Java APIs

Authentication Flow in SDK-Enabled Single Sign-On

Applications that use the SDK to enable Single Sign-On are required to implement the functions and procedures in the package `WWSEC_SSO_ENABLER` if PL/SQL is the language of implementation. The classes in `oracle.security.sso.enabler` are used if Java is the language of choice. Applications that are integrated with the SDK are called partner applications. They incorporate logic that enables them to delegate authentication, but not authorization, to the Single Sign-On server.

Figure 3–1 illustrates the authentication steps in SDK-enabled Single Sign-On. To understand how the process differs from mod_osso authentication, see "How Mod_osso Works" in Chapter 2, "Developing Applications Using Mod_osso."

Figure 3–1 Authentication Flow for SDK-Enabled Single Sign-On



1. The user requests a URL for Partner Application A.
2. The user is redirected to the Single Sign-On server.
3. The Single Sign-On server presents the user with the Single Sign-On login form. He uses this form to submit his user name and password.

4. The Single Sign-On server validates the user's password by checking it against Oracle Internet Directory. It also retrieves user attributes—specifically, the user's distinguished name and globally unique user ID.
5. The user is redirected to the success URL. This URL passes the URLC token to the application. The URLC token contains the user's information.
6. The application success URL sets the application session cookie and then redirects the user to the requested URL.

If the authenticated user requests Partner Application B, he need not authenticate again. The Single Sign-On server retrieves his credentials from the Single Sign-On cookie upon redirect (Step 2).

PL/SQL APIs

This section covers the following topics:

- Functions and Procedures
- Table Definitions
- Exceptions

Functions and Procedures

The functions and procedures in this section are part of the `WWSEC_SSO_ENABLER` package. This package is used to enable a PL/SQL application to become a partner application.

The section covers the following functions and procedures:

- GENERATE_REDIRECT Function
- PARSE_URL_COOKIE Procedure
- GET_ENABLER_CONFIG Procedure
- CREATE_ENABLER_CONFIG Procedure
- MODIFY_ENABLER_CONFIG Procedure
- DELETE_ENABLER_CONFIG Procedure
- ENCRYPT_COOKIE Function
- DECRYPT_COOKIE Function

GENERATE_REDIRECT Function

This function generates a redirect URL, along with SITE2PSTORETOKEN, that the Single Sign-On server parses.

Syntax

```
FUNCTION GENERATE_REDIRECT
(
     P_LSNR_TOKEN    IN  VARCHAR2
   , P_URL_REQUESTED IN  VARCHAR2
   , P_URL_CANCEL    IN  VARCHAR2
   , P_FORCED_AUTH   IN  NUMBER DEFAULT SIMPLE_AUTH
) RETURN VARCHAR2;
```

Table 3–1 Parameters for GENERATE_REDIRECT

Parameter	Description
P_LSNR_TOKEN	Listener token to get the necessary partner application registration configuration. The listener token is the host name and port used on the URL for the current request. This is used to select the appropriate configuration entry in the WWSEC_ENABLER_CONFIG_INFO$ table.
P_URL_REQUESTED	URL requested by the client.
P_URL_CANCEL	URL that users are directed to if they select cancel on the login page.
P_FORCED_AUTH	Forced authentication flag.
REDIRECTURL	URL to which the partner application must direct the browser to delegate authentication to the Single Sign-On server. This URL contains the request for authentication.

Example

```
WWSEC_SSO_ENABLER.GENERATE_REDIRECT
(
  p_lsnr_token    => listener token
  p_url_requested => requested url
  p_url_cancel    => cancel url
  p_forced_auth   => forced authentication flag
  redirecturl     => redirect url
);
```

PARSE_URL_COOKIE Procedure

This procedure parses the URL cookie that is generated by the GENERATE_REDIRECT function on the Single Sign-On server side.

Syntax

```
PROCEDURE parse_url_cookie
(
     P_LSNR_TOKEN            IN   VARCHAR2
   , P_ENC_URL_COOKIE        IN   VARCHAR2
   , P_URL_REQUESTED         OUT  VARCHAR2
   , P_SSO_USERNAME          OUT  VARCHAR2
   , P_SSO_USER_DN           OUT  VARCHAR2
   , P_SSO_USER_GUID         OUT  VARCHAR2
   , P_SUBSCRIBER_NAME       OUT  VARCHAR2
   , P_SUBSCRIBER_DN         OUT  VARCHAR2
   , P_SUBSCRIBER_GUID       OUT  VARCHAR2
   , P_USER_IPADDRESS        OUT  VARCHAR2
   , P_SSO_TIMEREMAINING     OUT  NUMBER
   , P_NLS_LANGUAGE          OUT  VARCHAR2
   , P_NLS_TERRITORY         OUT  VARCHAR2
);
```

Table 3–2 Parameters for PARSE_URL_COOKIE

Parameter	Description
P_LSNR_TOKEN	Listener token
P_ENC_URL_COOKIE	Encrypted URL cookie
P_URL_REQUESTED	Requested URL
P_SSO_USERNAME	Authenticated Single Sign-On user name
P_SSO_USER_DN	Authenticated Single Sign-On user DN
P_SSO_USER_GUID	Authenticated Single Sign-On user GUID
P_SUBSCRIBER_NAME	Subscriber name
P_SUBSCRIBER_DN	Subscriber DN
P_SUBSCRIBER_GUID	Subscriber GUID
P_USER_IPADDRESS	IP address of the Single Sign-On user's machine
P_SSO_TIMEREMAINING	Remaining Single Sign-On session duration
P_NLS_LANGUAGE	language selection of Single Sign-On user

Table 3–2 Parameters for PARSE_URL_COOKIE

Parameter	Description
P_NLS_TERRITORY	territory selection of Single Sign-On user

Example

```
WWSEC_SSO_ENABLER.PARSE_URL_COOKIE
(
   p_lsnr_token        => listener token
   p_enc_url_cookie    => encrypted URL cookie
   p_url_requested     => requested URL
   p_sso_username      => authenticated SSO username
   p_sso_user_dn       => authenticated SSO user DN
   p_sso_user_guid     => authenticated SSO user GUID
   p_subscriber_name   => subscriber name
   p_subscriber_dn     => subscriber DN
   p_subscriber_guid   => subscriber GUID
   p_user_ipaddress    => ipaddress of the sso user's machine
   p_sso_timeremaining => remaining Single Sign-On session duration
   p_nls_language      => language selection of sso user
   p_nls_territory     => territory selection of sso user
 );
```

GET_ENABLER_CONFIG Procedure

This function returns the partner application registration information specified by the listener token.

Syntax

```
PROCEDURE GET_ENABLER_CONFIG
(
    P_LSNR_TOKEN            IN   VARCHAR2,
    P_SITE_TOKEN            OUT  VARCHAR2,
    P_SITE_ID               OUT  VARCHAR2,
    P_LS_LOGIN_URL          OUT  VARCHAR2,
    P_LS_LOGOUT_URL         OUT  VARCHAR2,
    P_URL_COOKIE_VERSION    OUT  VARCHAR2,
    P_ENCRYPTION_KEY        OUT  VARCHAR2,
    P_IPADDR_CHECK          OUT  VARCHAR2
);
```

Table 3–3 Parameters for GET_ENABLER_CONFIG

Parameter	Description
P_LSNR_TOKEN	Listener token
P_SITE_TOKEN	Site token
P_SITE_ID	Site token
P_LS_LOGIN_URL	Login url of SSO Server
P_LS_LOGOUT_URL	Single Sign-Off URL of SSO Server
P_URL_COOKIE_VERSION	URL cookie version
P_ENCRYPTION_KEY	Encryption key
P_IPADDR_CHECK	If IP address should be verified

Example

```
WWSEC_SSO_ENABLER_PRIVATE.GET_ENABLER_CONFIG
(
   p_lsnr_token          =>  listener token
   p_site_token          =>  site token
   p_site_id             =>  site token
   p_ls_login_url        =>  login url of SSO Server
   p_ls_logout_url       =>  Single Sign-Off URL of SSO Server
   p_url_cookie_version  =>  url cookie version
   p_encryption_key      =>  encryption key
   p_ipaddr_check        =>  if ip address should be verified
```

CREATE_ENABLER_CONFIG Procedure

This procedure stores the partner application registration information, specified by the listener token, in the enabler configuration table.

Syntax

```
PROCEDURE CREATE_ENABLER_CONFIG
(
    P_LSNR_TOKEN           IN  VARCHAR2,
    P_SITE_TOKEN           IN  VARCHAR2,
    P_SITE_ID              IN  VARCHAR2,
    P_LS_LOGIN_URL         IN  VARCHAR2,
    P_LS_LOGOUT_URL        IN  VARCHAR2,
    P_URL_COOKIE_VERSION   IN  VARCHAR2,
    P_ENCRYPTION_KEY       IN  VARCHAR2,
    P_IPADDR_CHECK         IN  VARCHAR2
);
```

Table 3–4 Parameters for CREATE_ENABLER_CONFIG

Parameter	Description
P_LSNR_TOKEN	Listener token
P_SITE_TOKEN	Site token
P_SITE_ID	Site token
P_LS_LOGIN_URL	Login URL of the Single Sign-On server
P_LS_LOGOUT_URL	Single Sign-Off URL of the Single Sign-On server
P_URL_COOKIE_VERSION	URL cookie version
P_ENCRYPTION_KEY	Encryption key
P_IPADDR_CHECK	If IP address should be verified or not

Example

```
WWSEC_SSO_ENABLER.CREATE_ENABLER_CONFIG
(
   p_lsnr_token          =>  listener token
   p_site_token          =>  site token
   p_site_id             =>  site token
   p_ls_login_url        =>  login url of SSO Server
   p_ls_logout_url       =>  Single Sign-Off URL of the Single Sign-On server
   p_url_cookie_version  =>  URL cookie version
   p_encryption_key      =>  Encryption key
   p_ipaddr_check        =>  If IP address should be verified
)
```

MODIFY_ENABLER_CONFIG Procedure

This procedure modifies the partner application registration information specified by the listener token.

Syntax

```
PROCEDURE MODIFY_ENABLER_CONFIG
(
    P_LSNR_TOKEN           IN  VARCHAR2,
    P_SITE_TOKEN           IN  VARCHAR2,
    P_SITE_ID              IN  VARCHAR2,
    P_LS_LOGIN_URL         IN  VARCHAR2,
    P_LS_LOGOUT_URL        IN  VARCHAR2,
    P_URL_COOKIE_VERSION   IN  VARCHAR2,
    P_ENCRYPTION_KEY       IN  VARCHAR2,
    P_IPADDR_CHECK         IN  VARCHAR2
);
```

Table 3–5 Parameters for UPDATE_ENABLER_CONFIG

Parameter	Description
P_LSNR_TOKEN	Listener token
P_SITE_TOKEN	Site token
P_SITE_ID	Site token
P_LS_LOGIN_URL	Login url of SSO Server
P_LS_LOGOUT_URL	Single Sign-Off URL of SSO Server
P_URL_COOKIE_VERSION	URL cookie version

Table 3–5 Parameters for UPDATE_ENABLER_CONFIG

Parameter	Description
P_ENCRYPTION_KEY	Encryption key
P_IPADDR_CHECK	If IP address should be verified or not

Example

```
WWSEC_SSO_ENABLER.MODIFY_ENABLER_CONFIG
(
   p_lsnr_token          =>  listener token
   p_site_token          =>  site token
   p_site_id             =>  site token
   p_ls_login_url        =>  login url of SSO Server
   p_ls_logout_url       =>  Single Sign-Off URL of SSO Server
   p_url_cookie_version  =>  url cookie version
   p_encryption_key      =>  encryption key
   p_ipaddr_check        =>  if IP address should be verified or not
)
```

DELETE_ENABLER_CONFIG Procedure

This procedure deletes the partner application registration information specified by the listener token.

Syntax

```
PROCEDURE DELETE_ENABLER_CONFIG
(
    P_LSNR_TOKEN  IN VARCHAR2
);
```

Table 3–6 Parameters for DELETE_ENABLER_CONFIG

Parameter	Description
P_LSNR_TOKEN	Listener token to get the necessary partner application registration configuration

Example

```
WWSEC_SSO_ENABLER.DELETE_ENABLER_CONFIG
(
   p_lsnr_token  =>  listener token
);
```

ENCRYPT_COOKIE Function

This function returns the encrypted cookie body.

Syntax

```
FUNCTION ENCRYPT_COOKIE
(
    p_lsnr_token  in  varchar2,
    p_cookie      in  varchar2
 )  return varchar2;
```

Table 3–7 Parameters for ENCRYPT_COOKIE

Parameter	Description
P_LSNR_TOKEN	Listener token to get the necessary partner application registration configuration

Example

```
WWSEC_SSO_ENABLER.ENCRYPT_COOKIE
(
  p_lsnr_token  =>  listener token
  p_enc_cookie  =>  cookie value to be encrypted
)
```

DECRYPT_COOKIE Function

This function returns the decrypted cookie value from the encrypted cookie.

Syntax

```
(
    P_LSNR_TOKEN  IN  VARCHAR2,
    P_ENC_COOKIE  IN  VARCHAR2
 )  RETURN VARCHAR2;
```

Table 3–8 Parameters for DECRYPT_COOKIE

Parameter	Description
P_LSNR_TOKEN	Listener token to get the necessary partner application registration configuration
P_ENC_COOKIE	Cookie value to be encrypted

Example

```
WWSEC_SSO_ENABLER.DECRYPT_COOKIE
(
   p_lsnr_token  =>  listener token
   p_enc_cookie  =>  cookie value to be encrypted
)
```

Table Definitions

The Single Sign-On SDK contains two tables for partner applications: SEC_WWSEC_ENABLER_CONFIG_INFO$ and WWSEC_SSO_LOG$. The first stores configuration information that enables the application to determine which Single Sign-On server to connect to. The second stores client-side debug information, which can be accessed when debugging is enabled.

WWSEC_ENABLER_CONFIG_INFO$

```
CREATE TABLE wwsec_enabler_config_info$
(
  lsnr_token              VARCHAR2(255)
  , site_token            VARCHAR2(255)
  , site_id               VARCHAR2(255)
  , ls_login_url          VARCHAR2(1000)
  , urlcookie_version     VARCHAR2(80)
  , encryption_key        VARCHAR2(1000)
  , encryption_mask_pre   VARCHAR2(1000)
  , encryption_mask_post  VARCHAR2(1000)
  , url_cookie_ip_check   VARCHAR2(1)
  );
```

WWSEC_SSO_LOG$

```
CREATE TABLE wwsec_sso_log$
(
  , SUBSCRIBER_ID  NUMBER  NOT NULL
  , id           NUMBER
  , msg          VARCHAR2(1000)
  , log_date     DATE
 );
```

Exceptions

Table 3–9 lists and describes the exceptions raised by PL/SQL functions and procedures.

Table 3–9 Exceptions

Exception	Description
UNKNOWN_ERROR_EXCEPTION	Generic error
CONFIG_MISSING_EXCEPTION	SDK configuration table is unpopulated, or its contents are invalid
DUP_CONFIG_EXCEPTION	Partner configuration with same listener token already exists
ENCRYPTION_FAILED_EXCEPTION	Wrong key or bad input data
DECRYPTION_FAILED_EXCEPTION	Wrong key or bad input data
UNSUPPORTED_VERSION_EXCEPTION	SDK version and Single Sign-On server version are incompatible
IPADDR_ERROR_EXCEPTION	Pre- and post-authentication addresses do not match. User might be accessing applications through a proxy, or there might be a security attack, or user's computer might not use fixed IP addresses
COOKIE_EXPIRED_EXCEPTION	Authentication token sent by Single Sign-On server timed out
NULL_ATTRIBUTE_EXCEPTION	Wrong input data

Java APIs

Java APIs can be used in place of PL/SQL APIs to create partner applications. To learn how to use the Java APIs, see *Oracle9iAS Single-On API Reference (Javadoc)*.

4

Using the PL/SQL and Java APIs

This chapter provides sample programs that illustrate how to enable partner applications for Single Sign-On.

The chapter covers the following topics:

- Writing Partner Applications Using PL/SQL APIs
- Writing Partner Applications Using Java APIs

Writing Partner Applications Using PL/SQL APIs

The example that follows shows how to develop a partner application using PL/SQL APIs. The example incorporates three procedures: SAMPLE_SSO_PAPP.SSOAPP, SAMPLE_SSO_PAPP.SIGN_ON, and SAMPLE_SSO_PAPP.LOGOUT.

SAMPLE_SSO_PAPP.SSOAPP

This procedure constructs the application URL. It requires authentication to access it. The procedure checks to see if the application cookie exists and user information can be retrieved; otherwise it redirects the user to the Single Sign-On server by generating a redirect URL.

SAMPLE_SSO_PAPP.SIGN_ON

This procedure gets the URLC token from the Single Sign-On server, decrypts it, and retrieves user information and the requested URL. It sets the application cookie and redirects the browser to the partner application URL.

SAMPLE_SSO_PAPP.LOGOUT

This procedure implements the logout URL for the application.

```
/* papp.pks */

CREATE OR REPLACE PACKAGE sample_sso_papp
IS
    PROCEDURE ssoapp;
    PROCEDURE sign_on(urlc IN VARCHAR2);
    PROCEDURE logout;
END sample_sso_papp;
/
show errors package sample_sso_papp;

/* papp.pkb */

set define on;
set verify off;

CREATE OR REPLACE PACKAGE BODY sample_sso_papp
IS
    g_listener_token VARCHAR2(1000);
    g_requested_url  VARCHAR2(1000);
    g_cancel_url     VARCHAR2(1000);
    g_cookie_domain  VARCHAR2(1000);
    p_html_str       VARCHAR2(32000);
```

```
    g_cookie_path    VARCHAR2(1000) := '/';
    g_dad_name       VARCHAR2(100)  := '&partner_app_dad_name';
    g_cookie_name    VARCHAR2(1000) := g_dad_name;
    g_schema_name    VARCHAR2(100)  := user;

PROCEDURE init_params
AS
   l_host_name        VARCHAR2(256);
   l_server_port      VARCHAR2(256);
   l_protocol         VARCHAR2(256);
BEGIN
   begin
     htp.init;
   exception
     when others then null;
   end;

   l_host_name := owa_util.get_cgi_env('SERVER_NAME');
   l_server_port := owa_util.get_cgi_env('SERVER_PORT');
   -- the mod_plsql gateway will pass in the protocol in
   -- a new environment variable REQUEST_PROTOCOL.
   -- The SERVER_PROTOCOL, which the Apache Listener sets,
   -- and currently always sets to HTTP/1.0, will not be
   -- modified by the gateway.
   l_protocol := owa_util.get_cgi_env('REQUEST_PROTOCOL');

   g_listener_token := l_host_name || ':' || l_server_port;
   if(l_protocol is null) or (length(l_protocol) = 0) then
     l_protocol := 'http';
   end if;
   l_protocol := lower(l_protocol);
   g_requested_url := l_protocol || '://' || g_listener_token
    || '/pls/' || g_dad_name || '/' ||g_schema_name ||'.sample_sso_papp.ssoapp';

   g_cancel_url := l_protocol || '://' || g_listener_token;
   g_cookie_domain := l_host_name;
EXCEPTION
  when others then
    htp.p(SQLERRM);htp.nl;
END init_params;


/* Get user information */
FUNCTION get_user_info
```

```
    RETURN VARCHAR2
IS
    l_user_info  VARCHAR2(1000);
    l_app_cookie owa_cookie.cookie;
BEGIN

    l_app_cookie := owa_cookie.get(g_cookie_name);
    if (l_app_cookie.num_vals > 0)
    then
      l_user_info  := l_app_cookie.vals(1);
      if l_user_info is not null then
        l_user_info := wwsec_sso_enabler.decrypt_cookie(
            g_listener_token, l_user_info);
      end if;
    else
      l_user_info  := NULL;
    end if;
    return l_user_info;
EXCEPTION
    WHEN OTHERS THEN
      htp.p('get_user_info: '||SQLERRM);htp.nl;
END get_user_info;

function gen_html_post_str
(
    l_gen_url IN VARCHAR2
)
   RETURN VARCHAR2
IS
   l_htmlstr  varchar2(1000);
   l_ls_url   varchar2(1000);
   l_tname    varchar2(100);
   l_tvalue   varchar2(1000);
   l_len      number;
   l_qindex   number;
   l_eq_index number;
BEGIN
   l_len      := length(l_gen_url);
   l_qindex   := instr(l_gen_url, '?');
   l_eq_index := instr(l_gen_url, '=');

   l_ls_url := substr(l_gen_url, 0,  l_qindex-1);
   l_tname  := substr(l_gen_url, l_qindex+1, l_eq_index-l_qindex-1);
   l_tvalue := substr(l_gen_url, l_eq_index+1);
```

```
   l_htmlstr :=
      '<HTML><BODY onLoad="document.LoginForm.submit();">'
   || '<FORM ACTION="' || l_ls_url || '" METHOD="POST" NAME="LoginForm">'
   || '<INPUT TYPE="HIDDEN" NAME="' || l_tname
        || '" VALUE="' || l_tvalue  || '">'
   || '</FORM></BODY></HTML>';
   return l_htmlstr;
EXCEPTION
        l_gen_redirect_url :=
        wwsec_sso_enabler.generate_redirect
        (
            p_lsnr_token    => g_listener_token,
            p_url_requested => g_requested_url,
            p_url_cancel    => g_cancel_url,
            p_forced_auth   => wwsec_sso_enabler.SIMPLE_AUTH
        );
       htp.p('Redirecting to the SSO Server for authentication...');
       --
       -- The l_gen_redirect_url is usually large url which might
       -- get truncated by the browser.
       -- Instead of using owa_util.redirect_url, we will use
       -- HTTP POST for sending redirect.
       -- For moblie application etc. it may not be possible to use HTTP
       -- POST since it may not support html hidden form parameter.
       -- owa_util.redirect_url(l_gen_redirect_url);
       --
       l_html_str := gen_html_post_str(l_gen_redirect_url);
       htp.p(l_html_str);
    ELSE
        owa_util.mime_header('text/html', FALSE);
        htp.p('Expires: ' 'Thu, 29 Oct 2000 17:04:19 GMT');
        htp.p('Pragma: ' 'no-cache');
        htp.p('Cache-Control: ' || 'no-cache');
        owa_util.http_header_close;

        htp.htmlOpen;
        htp.headOpen;
        htp.title('PL/SQL based SSO Partner Application');
        htp.headCLose;
        htp.bodyOpen;
        htp.p('Congratulations! It is working!<br>');
        htp.p('User Information: ');
        htp.p(l_user_info || '<br>');

        wwsec_sso_enabler.get_enabler_config
```

```
            (
                p_lsnr_token          => g_listener_token,
                p_site_token          => l_unused1,
                p_site_id             => l_unused2,
                p_ls_login_url        => l_unused3,
                p_ls_logout_url       => l_sso_logout_url,
                p_url_cookie_version  => l_unused4,
                p_encryption_key      => l_unused5,
                p_ipaddr_check        => l_unused6
             );

        htp.p('<A HREF="' || l_sso_logout_url || '?p_done_url=">Logout</A>');

        htp.bodyClose;
        htp.htmlClose;
    END IF;
EXCEPTION
    WHEN no_data_found OR wwsec_sso_enabler.CONFIG_MISSING_EXCEPTION THEN
       htp.p('Error in application: missing application registration
       information');
       htp.p('<br>');
      htp.p('Please register this application as described in the installation'
       || 'guide');
       htp.nl;
   WHEN others THEN
       htp.p('Error in application:' || sqlerrm);
       htp.nl;
END ssoapp;

PROCEDURE sign_on
(
    urlc IN VARCHAR2
)
IS
    l_url_requested       VARCHAR2(2000);
    l_sso_user_name       VARCHAR2(1000);
    l_user_dn             VARCHAR2(1000);
    l_user_guid           VARCHAR2(1000);
    l_subs_name           VARCHAR2(1000);
    l_subs_dn             VARCHAR2(1000);
    l_subs_guid           VARCHAR2(1000);
    l_ip_address          VARCHAR2(100);
    l_sso_time_remaining  NUMBER;
    l_nls_lang            VARCHAR2(100);
    l_nls_territory       VARCHAR2(100);
```

```
    l_cookie              VARCHAR2(5000);
BEGIN
    init_params;
    -- Process URLC token
    wwsec_sso_enabler.parse_url_cookie
    (
       p_lsnr_token        => g_listener_token
     , p_enc_url_cookie    => urlc
     , p_url_requested     => l_url_requested
     , p_sso_username      => l_sso_user_name
     , p_sso_user_dn       => l_user_dn
     , p_sso_user_guid     => l_user_guid
     , p_subscriber_name   => l_subs_name
     , p_subscriber_dn     => l_subs_dn
     , p_subscriber_guid   => l_subs_guid
     , p_user_ipaddress    => l_ip_address
     , p_sso_timeremaining => l_sso_time_remaining
     , p_nls_language      => l_nls_lang
     , p_nls_territory     => l_nls_territory
    );

    -- Set application cookie
    -- This program will set few variables for displaying
    -- This cookie must be encrypted for production application
    owa_util.mime_header('text/html', FALSE);
    htp.p('Expires: ' || 'Thu, 29 Oct 2000 17:04:19 GMT');
    htp.p('Pragma: ' || 'no-cache');
    htp.p('Cache-Control: ' || 'no-cache');

    l_cookie := l_sso_user_name || '/' || l_subs_name;
    owa_cookie.send
    (
        name    => g_cookie_name,
        value   => wwsec_sso_enabler.encrypt_cookie(
                     g_listener_token, l_cookie),
        path    => g_cookie_path,
        domain  => g_cookie_domain
    );
    owa_util.redirect_url(l_url_requested);
    owa_util.http_header_close;
    -- Redirect user to the requested application url
    htp.htmlOpen;
    htp.headOpen;
    htp.p('');
```

```
    htp.headClose;
    htp.htmlClose;
EXCEPTION
  WHEN OTHERS THEN
    htp.p(sqlerrm);
END sign_on;

PROCEDURE logout
AS
  l_img_blob blob;
BEGIN
   init_params;

   owa_util.mime_header('image/gif', FALSE);
   owa_cookie.send
    (
        name    => g_cookie_name,
        value   => '',
        path    => g_cookie_path,
        domain  => g_cookie_domain
    );
   select image_blob into l_img_blob from wwsec_image_config$;
   htp.p('Content-Length: ' || dbms_lob.getlength(l_img_blob));
   htp.p('Expires: Thu, 29 Oct 2000 17:04:19 GMT');
   htp.p('Pragma: no-cache');
   htp.p('Cache-Control: no-cache');
   owa_util.http_header_close;
   wpg_docload.download_file(l_img_blob);
EXCEPTION
  WHEN OTHERS THEN
   htp.p(sqlerrm);
END logout;

END sample_sso_papp;
/
show errors package body sample_sso_papp
```

Writing Partner Applications Using Java APIs

Initially, the partner application redirects the user to the Single Sign-On server for authentication and, after successful authentication, sets its own application session cookie. Any future request first attempts to validate the application session cookie. If the application session cookie is not found, the partner application redirects the user to the Single Sign-On server. To avoid contacting the Single Sign-On server for authentication verification of every user request, all partner applications should maintain their own application session.

This section covers the following topics

- Implementing the Partner Application in Java
- Servlet Partner Application
- JSP Partner Application

Implementing the Partner Application in Java

The following program implements a generic bean that can be used in servlets and JavaServer pages (JSPs).

Servlet Partner Application

A servlet partner application can be implemented using one bean and three servlets.

1. The user goes to the `SSOPartnerServlet` application URL. This servlet will get the user information with the help of `SSOEnablerServletBean`. If the user information is found, it is used inside the application; otherwise, the browser redirects the user to the Single Sign-On server.
2. After authentication, the Single Sign-On server does the following:
   - redirects the user to the `SSOSignOnServlet` URL
   - sets the application cookie
   - redirects the user to the requested application URL using `SSOEnablerServletBean`

SSOEnablerServletBean

This bean is derived from `SSOEnablerBean`. It implements the necessary methods for servlet applications.

```
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import oracle.security.sso.enabler.SSOEnablerException;

public class SSOEnablerServletBean
{
   /** Start configuration parameters
    *  For production quality application, you should read these
    *  parameters from database instead of harcoding them here.
    */

    // Partner application  session cookie name
    private static String m_cookieName     = "SSO_PAPP_SERVLET_ID";

    // Host name of the database
    private static String m_dbHostName     = "wwssosvr.us.oracle.com";
    // Port for database
    private static int    m_dbPort         = 9521;
    // Sehema name
    private static String m_dbSchemaName   = "papp";
    // Schema password
    private static String m_dbSchemaPasswd = "papp";
    // Database SID name
    private static String m_dbSID          = "orcl9i";
    // Database connection pool size
    private static int    m_dbPoolSize     = 3;

    /* End of configuration parameters */

    // Enabler object (Don't change)
    private SSOEnablerBean  m_enablerBean = null;

   /**
    *  Default constructor
    */
```

```
    public SSOEnablerServletBean()
    {
        m_enablerBean = new SSOEnablerBean();
        m_enablerBean.setAppCookieInfo(m_cookieName);
        m_enablerBean.setDbConnectionInfo(m_dbSchemaName, m_dbSchemaPasswd,
            m_dbHostName, m_dbPort, m_dbSID, m_dbPoolSize);
    }

    public String getSSOUserInfo(HttpServletRequest p_request,
    HttpServletResponse p_response)
        throws SSOEnablerException
    {
        return m_enablerBean.getSSOUserInfo(p_request, p_response);
    }

    public void setPartnerAppCookie(HttpServletRequest p_request,
    HttpServletResponse p_response)
        throws SSOEnablerException
    {
        m_enablerBean.setPartnerAppCookie(p_request, p_response);
    }

    public void removeServletAppCookie(HttpServletResponse p_response)
        throws SSOEnablerException
    {
        m_enablerBean.removeAppCookie(p_response);
    }

    public String getSingleSignOffUrl(HttpServletRequest p_request)
        throws SSOEnablerException
    {
        return m_enablerBean.getSingleSignOffUrl(p_request);
    }

    public void destroy()
    {
        m_enablerBean.destroy();
    }
}
```

SSOPartnerServlet

This servlet is the main partner application servlet. To access it, the user must authenticate to the Single Sign-On server. This servlet redirects the unauthenticated user to the Single Sign-On server.

```
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpServlet;

import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import javax.servlet.UnavailableException;

import java.io.PrintWriter;

public class SSOPartnerServlet extends HttpServlet
{
   SSOEnablerServletBean m_ssobean = null;

   public void init(ServletConfig p_config)
     throws ServletException
   {
        try
        {
           super.init(p_config);
           m_ssobean = new SSOEnablerServletBean();
        }
        catch(Exception e)
        {
           throw new UnavailableException (
                    "Could not create SSOEnablerServletBean object");
        }
   }
   /**
    * The HTTP GET request will show the application content of the user if
      he/she is already authenticated, otherwise he/she will be redirected to
      the Single Sign-On server
    */
   public void doGet(HttpServletRequest p_request, HttpServletResponse
   p_response)
        throws ServletException
   {
       p_response.setContentType("text/html");

       if(p_request == null || p_response == null)
```

```
{
     throw new ServletException("Http objects are null");
}

try
{
    PrintWriter l_out   = p_response.getWriter();
    String l_userInfo   = m_ssobean.getSSOUserInfo(p_request,
    p_response);

    if(l_userInfo != null)
    {
        String l_sso_logout_url = m_ssobean.getSingleSignOffUrl(p_
        request);
        // Display some application content for the SSO user
        l_out.println("<HTML><HEAD><TITLE>Servlet based SSO Partner" +
        "Application</TITLE></HEAD><BODY>");
        l_out.println("<H3><center>Servlet based SSO Partner" +
        "Application</center></H3>");
        l_out.println("<P><center>User Information: " + l_userInfo +
        "<center><BR>");
        // You may specify a URL value in p_done_url.
        // User can be redirected to this URL after Single Sign-Off
        // (Global Logout)
        l_out.println("<P><center><A HREF='"+l_sso_logout_url+
        "?p_done_url='>Logout</A><center></P>");
        l_out.println("</BODY></HTML>");
    }
    else
    {
        // Display redirection to SSO server message
        l_out.println("<HTML><HEAD><TITLE>Servlet based SSO Partner" +
        "Application</TITLE></HEAD><BODY>");
        l_out.println("<center>Please wait while redirecting to the SSO"
        + "Server...</center>");
        l_out.println("</BODY></HTML>");
    }
}
catch(Exception e)
{
     try
     {
         p_response.getWriter().println("Error " + e.toString());
     }
     catch(Exception e1)
```

```
            {
                throw new ServletException(e1.toString());
            }
        }
    }

    public void destroy()
    {
       m_ssobean.destroy();
    }
}
```

SSOSignOnServlet

This servlet parses the URLC token received from Single Sign-On server, sets the application cookie, and redirects the user to the requested web application URL (i.e. `SSOPartnerServlet`).

```
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpServlet;

import javax.servlet.UnavailableException;
import javax.servlet.ServletException;
import javax.servlet.ServletConfig;

import java.io.PrintWriter;

public class SSOSignOnServlet extends HttpServlet
{
  SSOEnablerServletBean m_ssobean = null;

   public void init(ServletConfig p_config)
     throws ServletException
   {
        try
        {
           super.init(p_config);
           m_ssobean = new SSOEnablerServletBean();
        }
```

```
            catch(Exception e)
            {
               throw new UnavailableException (
                        "Could not create SSOEnablerServletBean object");
            }
        }

        /**
         * The HTTP Post will set application cookie from SSO server token and then
           redirect user to the Servlet based partner application
         */
        public void doPost(HttpServletRequest p_request, HttpServletResponse
        p_response)
            throws ServletException
        {
            p_response.setContentType("text/html");

            if(p_request == null || p_response == null)
            {
                throw new ServletException("Http objects are null");
            }
            try
            {
                m_ssobean.setPartnerAppCookie(p_request, p_response);
            }
            catch(Exception e)
            {
                try
                {
                    p_response.getWriter().println("Error " + e.toString());
                }
                catch(Exception e1)
                {
                    throw new ServletException(e1.toString());
                }
            }
        }

        public void destroy()
        {
           m_ssobean.destroy();
        }
    }
```

SSOPartnerLogoutServlet

This servlet removes the application session of the partner application.

```
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpServlet;

import javax.servlet.UnavailableException;
import javax.servlet.ServletException;
import javax.servlet.ServletConfig;

import java.io.PrintWriter;

public class SSOPartnerLogoutServlet extends HttpServlet
{
   SSOEnablerServletBean m_ssobean = null;

   public void init(ServletConfig p_config)
     throws ServletException
   {
        try
        {
           super.init(p_config);
           m_ssobean = new SSOEnablerServletBean();
        }
        catch(Exception e)
        {
           throw new UnavailableException (
                    "Could not create SSOEnablerServletBean object");
        }
   }
   public void doGet(HttpServletRequest p_request, HttpServletResponse
   p_response)
     throws ServletException
   {
       try
       {
           m_ssobean.removeServletAppCookie(p_response);
       }
       catch(Exception e)
       {
          throw new ServletException(e.toString());
       }
   }
```

```
   public void destroy()
   {
      m_ssobean.destroy();
   }
}
```

JSP Partner Application

The JSP partner application can be implemented using a Java bean for generating a redirection URL and processing the redirected URL parameter from the Single Sign-On server. A JSP page should embed this bean, which can be included in all JSP based applications that require Single Sign-On functionality.

1. The user goes to the `papp.jsp` page.
2. This page gets the user information with the help of the `ssoinclude.jsp` page. If the user information can be found, then it is used by the application. Otherwise, the browser redirects the user to the Single Sign-On server using `SSOEnablerJspBean`.
3. After authentication, the Single Sign-On server redirects the user to the `ssosignon.jsp` page. This page sets the application cookie and redirects the user to the requested application URL using `SSOEnablerJspBean`.

A JSP application can be implemented with the following bean and JSP pages:

SSOEnablerJspBean.java

This bean uses the `getSSOUserInfo` method, which returns the user information when the application cookie is already set; otherwise, it redirects the user to the Single Sign-On server for authentication.

```
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import oracle.security.sso.enabler.SSOEnablerException;

public class SSOEnablerJspBean
{
   /** Start configuration parameters
    *  For production quality application, you should read these
    *  parameters from database instead of harcoding them here.
    */

    // Partner application  session cookie name
    private static String m_cookieName     = "SSO_PAPP_JSP_ID";
```

```
// Host name of the database
private static String m_dbHostName     = "wwssosvr.us.oracle.com";
// Port for database
private static int    m_dbPort         = 9521;
// Sehema name
private static String m_dbSchemaName   = "papp";
// Schema password
private static String m_dbSchemaPasswd = "papp";
// Database SID name
private static String m_dbSID          = "orcl9i";
// Database connection pool size
private static int    m_dbPoolSize     =  5;

/* End of configuration parameters */

// Enabler object (Don't change)
private SSOEnablerBean  m_enablerBean = null;

/**
 *  Default constructor
 */
public SSOEnablerJspBean()
{
    m_enablerBean = new SSOEnablerBean();
    m_enablerBean.setAppCookieInfo(m_cookieName);
    m_enablerBean.setDbConnectionInfo(m_dbSchemaName, m_dbSchemaPasswd,
    m_dbHostName , m_dbPort , m_dbSID, m_dbPoolSize);
}

public String getSSOUserInfo(HttpServletRequest p_request,
HttpServletResponse p_response)
    throws SSOEnablerException
{
    return m_enablerBean.getSSOUserInfo(p_request, p_response);
}

public void setPartnerAppCookie(HttpServletRequest p_request,
HttpServletResponse p_response)
    throws SSOEnablerException
{
    m_enablerBean.setPartnerAppCookie(p_request, p_response);
}

public void removeJspAppCookie(HttpServletResponse p_response)
```

```
        throws SSOEnablerException
    {
        m_enablerBean.removeAppCookie(p_response);
    }

    public String getSingleSignOffUrl(HttpServletRequest p_request)
        throws SSOEnablerException
    {
        return m_enablerBean.getSingleSignOffUrl(p_request);
    }

    public void destroy()
    {
        m_enablerBean.destroy();
    }
}
```

ssoinclude.jsp

This page embeds the `SSOEnablerJsp` bean. It should be included in all JSP pages where Single Sign-On functionality is required.

```
<%@ page language="java" import="oracle.security.sso.enabler.*" %>
<jsp:useBean id="ssoObj" scope="page" class="SSOEnablerJspBean" />
<%
    String usrInfo    = ssoObj.getSSOUserInfo(request, response);
    String logoutURL  = ssoObj.getSingleSignOffUrl(request);
    if(usrInfo == null)
    {
%>
    <center>Please wait while redirecting to the SSO Server...</center>
<%
    }
%>
```

ssosignon.jsp

This page embeds the `SSOEnablerJspBean` for generating the redirection URL and processing the redirected URL parameter received from the Single Sign-On server.

```
<%@ page language="java" import="oracle.security.sso.enabler.*" %>
<jsp:useBean id="ssoObj" scope="page" class="SSOEnablerJspBean" />
<%
    ssoObj.setPartnerAppCookie(request, response);
%>
```

papp.jsp

This page is the main application page and requires Single Sign-On functionality. To retrieve user information, the page must include the `ssoinclude.jsp` page.

```
<%@ page buffer="5" autoFlush="true" %>

<%@ include file="ssoinclude.jsp" %>
<%
    if(usrInfo != null)
    {
        response.getWriter().println("<center><h2>Sample JSP Partner
        Application" + "</FONT></h2></center>");
        response.getWriter().println("<center>User information :" + usrInfo
        +"</center>");
        response.getWriter().println("<center><A HREF='" + logoutURL +
        "?p_done_url='>Logout</A><center></P>");
    }
    else
    {
       response.getWriter().println("<center>User information not"
       + "found</center>
       ");
    }
%>
```

papplogoff.jsp

This JSP page removes the application session.

```
<%@ page language="java" import="oracle.security.sso.enabler.*" %>
<jsp:useBean id="ssoObj" scope="page" class="SSOEnablerJspBean" />
<%
        try
        {
            ssoObj.removeJspAppCookie(response);
        }
        catch(Exception e)
        {
%>
           <center>
             Error in ending JSP application session.
             Please quit your all browser windows.
           </center>
<%
           return;
        }
%>
         <center>
           You are logged off from JSP application session
           <br>
           <a href="papp.jsp">Login</a>
         </center>
```

Index

S

W