Oracle® Application Server Single Sign-On

Application Developer's Guide

10*g* (9.0.4)

Part No. B10852-01

September 2003

Oracle Application Server Single Sign-On Application Developer’s Guide, 10*g* (9.0.4)

Part No. B10852-01

Primary Author: Henry Abrecht

Contributors: Kamalendu Biswas, Ganesh Kirti, Arun Swaminathan

Contents

List of Tables

Send Us Your Comments

Oracle Application Server Single Sign-On Application Developer’s Guide, 10*g* (9.0.4)

Part No. B10852-01

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: appserverdocs_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

Oracle Application Server Single Sign-On Application Developer's Guide is written for developers who modify applications for OracleAS Single Sign-On. This modification is effected using either mod_osso, an authentication module on the Oracle HTTP Server, or the single sign-on SDK. The material presented in this book applies to UNIX and Windows NT/2000 platforms.

Note: The chapters in this book use UNIX notation to direct the reader to single sign-on files. With the exception of the `ssocfg` script, UNIX and Windows share the same file names and locations. Use the following format to access the Windows versions:

%ORACLE_HOME%\directory_path\\

This preface contains these topics:

- Audience
- Organization
- Related Documentation
- Conventions
- Documentation Accessibility

Audience

This document assumes the following knowledge or capabilities:

- Access to a working copy of OracleAS or the ability to install one
- An understanding of OracleAS concepts
- Proficiency in the PL/SQL or Java programming language

Organization

Oracle Application Server Single Sign-On Application Developer's Guide focuses on how the Oracle HTTP authentication module mod_osso is used to enable applications for single sign-on. Text that explains how to use the single sign-on SDK for the same purpose is in the appendixes.

Chapter 1, "Introduction"

Introduces mod_osso and the Single Sign-On SDK. Provides a brief description of other single sign-on components.

Chapter 2, "Developing Applications for Single Sign-On"

Explains how the HTTP authentication module mod_osso protects applications enabled by OracleAS Single Sign-On. Provides code that demonstrates how applications are integrated with mod_osso.

Appendix A, "Single Sign-On Software Development Kit"

Lists and describes the PL/SQL APIs for single-sign-on-enabling applications. The SDK also contains Java APIs.

Appendix B, "Using the PL/SQL and Java APIs"

Explains how to write partner applications using PL/SQL and Java. Provides code examples for both languages.

Appendix C, "Adding and Editing SDK-Enabled Applications"

Explains how to add, or register, an SDK-integrated application with the single sign-on server. Explains how to edit the registry of an existing application.

Appendix D, "User Attributes Passed to Partner Applications"

Lists and describes the user attributes that the single sign-on server verifies in or retrieves from Oracle Internet Directory. These attributes are used to construct the URLC token, which is passed to partner applications.

Glossary

Defines terms used in the book.

Related Documentation

For more information, see these Oracle resources:

- *Oracle Application Server Single Sign-On Administrator's Guide*
- *Oracle Application Server Single Sign-On API Reference*

Printed documentation is available for sale in the Oracle Store at

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

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/membership/`

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/documentation/`

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 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.	SQL> SELECT NAME FROM V$DATAFILE; NAME ------------------------------------ /fsl/dbs/tbs_01.dbf /fs1/dbs/tbs_02.dbf . . . /fsl/dbs/tbs_09.dbf 9 rows selected.

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*

Convention	Meaning	Example

`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;`

`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 Windows Operating Systems

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

Convention	Meaning	Example
Choose Start >	How to start a program.	To start the 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`

Convention	Meaning	Example
`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>`

Special characters	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. For Windows NT, the default location was `C:\orant`. 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 the latest Oracle release on a computer with no other Oracle software installed, then the default setting for the first Oracle home directory is `C:\oracle\ora`*nn*, where *nn* is the latest release number. 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 Started 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.

1 Introduction

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

OracleAS Single Sign-On consists of the following components:

- OracleAS Single Sign-On Server

  Program logic that enables users to log in securely to single-sign-on-enabled applications such as expense reports, mail, and benefits

- Partner Applications

  OracleAS 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, "Components and Processes: an Overview" *in Oracle Application Server Single Sign-On Administrator's Guide*

In OracleAS release 9.0.4 you use mod_osso, an authentication module on the Oracle HTTP Server, to enable applications for single sign-on. mod_osso is a simple alternative to the single sign-on SDK, used in earlier releases to integrate partner applications. mod_osso simplifies the authentication process by serving as the sole partner application to the single sign-on server. By doing so, it renders authentication transparent for OracleAS applications. The administrator for these applications is spared the burden of integrating them with the single sign-on SDK.

Note that the SDK has been deprecated. If you have built applications using the release 9.0.2 SDK, Oracle Corporation recommends modifying these with mod_osso. Nevertheless, 9.0.2 applications will continue to work in 9.0.4.

The main body of this book explains how to integrate applications with mod_osso. If you want to know more about the SDK, see the appendixes.

2 Developing Applications for Single Sign-On

This chapter explains how to develop applications to work with mod_osso. The chapter contains the following topics:

- Protecting Applications Using mod_osso: Two Methods
- Developing Applications Using mod_osso
- Security Issues: Single Sign-Off and Application Logout

Protecting Applications Using mod_osso: Two Methods

mod_osso redirects the user to the single sign-on server only if the URL you request is configured to be protected. You can secure URLs in one of two ways: statically or dynamically. Static directives simply protect the application, ceding control over user interaction to mod_osso. Dynamic directives not only protect the application, they also enable it to regulate user access.

This section contains the following topics:

- Protecting URLs Statically
- Protecting URLs with Dynamic Directives

Protecting URLs Statically

You can statically protect URLs with mod_osso by applying directives to the mod_osso.conf file. In the example that follows, a directory named `/private`, located just below the Oracle HTTP Server document root, is protected by this directive:

```
<IfModule mod_osso.c>

  <Location /private>
    AuthType Basic
    require valid-user
  </Location>

</IfModule>
```

After making the entry, populate the directory with pages and then test them. For example:

```
http://host:port/private/helloworld.html
```

Finally, restart the Oracle HTTP Server:

```
$ORACLE_HOME/opmn/bin/opmnctl restartproc type=ohs
```

Protecting URLs with 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.

OracleAS release 9.0.4 supports dynamic directives for Java servlets and JSPs. The product does not currently support dynamic directives for PL/SQL applications.

Table 2–1 lists commonly requested dynamic directives.

Table 2–1 Commonly Requested Dynamic Directives

Directive	Status Code	Headers
Request Authentication	401, 499	-
Request Forced Authentication	499	`Osso-Paranoid: true`
Single Sign-Off	470	`Osso-Return-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 contains the following topics:

- Developing Statically Protected PL/SQL Applications
- Developing Statically Protected Java Applications
- Developing Java Applications That Use Dynamic Directives
- A Word About Non-GET Authentication

Developing Statically Protected PL/SQL Applications

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 the single sign-on server.

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

1. Create the schema where application procedure will be loaded.

   ```
   sqlplus sys/sys_password as sysdba
   create user schema_name identified by schema_password;
   grant connect,  resource to schema_name;
   ```
2. Load the following procedure into the schema and grant the public access to the procedure:

```
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;
```

3. Create a database access descriptor (DAD) for the application in the dads.conf file, located at $ORACLE_HOME/Apache/modplsql/conf:

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

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

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

Note: The assumption here is that mod_osso is already configured for single sign-on. This step is performed when OracleAS is installed.

5. Restart the Oracle HTTP Server that will be used by this application:

```
$ORACLE_HOME/opmn/bin/opmnctl restartproc type=ohs
```

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

```
http://host:port/pls/DAD/schema_name.show_user_info
```

Selecting the URL should invoke the single sign-on login page if mod_osso.conf has been configured properly and mod_osso is registered with the single sign-on server.

Developing Statically Protected 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 informationsingle sign-on
        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. Deploy the servlet; then restart the Oracle HTTP Server and OC4J:

```
$ORACLE_HOME/opmn/bin/opmnctl restartproc type=ohs
$ORACLE_HOME/opmn/bin/opmnctl stopproc type=oc4j
$ORACLE_HOME/opmn/bin/opmnctl startproc type=oc4j
```

4. Test the servlet by trying to access it from the browser. Selecting the URL should invoke the login page.

   The process is this: when you try to access the servlet from the browser, you are redirected to the single sign-on server for authentication. Next you are redirected back to the servlet, which displays user information. You may then select the logout link to log out of the application as well as the single sign-on server.

Developing Java Applications That Use Dynamic Directives

Applications that use dynamic directives require no entry in mod_osso.conf 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 sample "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 `request.getRemoteUser()` method 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.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");
        }
    }
```

Note: Alternatively, you can redirect to the `osso_logout` URL on that computer.

Java Example #3: Forced Authentication

If logged-in users have exceeded a timeout, an application can force them to reauthenticate. The directive for reauthentication is written into the servlet that follows. 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 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");
               }
           }
       }
```

A Word About Non-GET Authentication

The first page of a mod_osso-protected application must be a URL that uses the GET authentication method. If the POST method is used, the data that the user provides when logging in is lost during redirection to the single sign-on server.

When deciding whether to enable the global user inactivity timeout, please note that users are redirected after timing out and logging in again.

Security Issues: Single Sign-Off and Application Logout

If you build custom applications using OracleAS release 9.0.4, note the following: when global logout, or single sign-off, is invoked, only the single sign-on and mod_osso cookies are cleared. This means that an OracleAS application must be coded to store single sign-on user and realm names in either the OC4J session or in the application session. The application must then compare these values to those passed by mod_osso. If a match occurs, the application must show personalized content. If no match occurs, which means that the mod_osso cookie is absent, the application must clear the application session and force the user to log in.

This section covers the following topics:

- Application Login: Code Examples
- Application Logout: Recommended Code

Application Login: Code Examples

The first two code examples in this section do not incorporate the logic prescribed in the section immediately preceding. The third example does incorporate this logic. Although these are Java examples, they could be examples written in other languages such as Perl, PL/SQL, and CGI.

Bad Code Example #1

```
// Get user name from application session. This session was
// established by the application cookie or OC4J session cookie
String username = request.getSession().getAttribute('USER_NAME');

// Get subscriber name from application session. This session was
// established by the application cookie or OC4J session cookie.
String subscriber = request.getSession().getAttribute('SUBSCRIBER_NAME');

// Get user security information from application session. This session was
established  by the application cookie or OC4J session cookie
String user_sec_info = request.getSession().getAttribute('USER_APP_SEC');

if((username != null) && (subscriber!= null))
{
  // Show personalized user content
  show_personalized_page(username, subscriber, user_sec_info);
}
else
{
  // Send Dynamic Directive for login
  response.sendError( 499, "Oracle SSO" );
```

Bad Code Example #2

```
// Get SSO username from http header
String username = request.getRemoteUser();

// Get subscriber name from SSO http header
String subscriber = request.getHeader(’OSSO-SUBSCRIBER’);

// Get user security information from application session.  This session // was
established by the application or OC4J session
String user_sec_info =request.getSession().getAttribute(’USER_APP_SEC’);

if((ssousername != null)&&(subscriber!= null))
{
  // Show personalized user content
show_personalized_page(username, subscriber, user_sec_info);
}
else
{
  // Send Dynamic Directive for login
response.sendError( 499, "Oracle SSO" );
}
```

Recommended Code

```
// Get user name from application session. This session was
// established by the application or OC4J session
String username = request.getSession().getAttribute(’USER_NAME’);

// Get subscriber name from application session. This session was
// established by the application or OC4J session
String subscriber = request.getSession().getAttribute(’SUBSCRIBER_NAME’);

// Get user security information from application session. This session // was
established by the application or OC4J session
String user_sec_info = request.getSession().getAttribute(’USER_APP_SEC’);

// Get username and subscriber name from JAZN API */
```

```
JAZNUserAdaptor jaznuser  = (JAZNUserAdaptor)requset.getUserPrincipal();
     String ssousername   = jaznuser.getName();
     String ssosubscriber = jaznuser.getRealm().getName();

// If you are not using JAZN api then you can also get the username and
// subscriber name from mod_osso headers
String ssousername   = request.getRemoteUser();
String ssosubscriber = request.getHeader('OSSO-SUBSCRIBER');

// Check for application session. Create it if necessary.
if((username == null) || (subscriber == null) )
  {
    ...Code to create application session. Get the user information from
    JAZN api(or mod_osso headers if you are not using JAZN api) and     populate
the application session with user, subscriber and user     security info...
  }

if((username != null)&&(subscriber != null)
  &&(ssousername != null)&&(ssosubscriber != null)
  &&(username.equalsIgnoreCase(ssousername) == 0 )
  &&(subscriber.equalsIgnoreCase(ssosubscriber) == 0))
{
  // Show personalized user content
show_personalized_page(username, subscriber, user_sec_info);
}
else
{
     ...Code to Wipe-out application session, followed by...

// Send Dynamic Directive for login
// If you are using JAZN then you should use following code
// response.sendError( 401);

// If you are not using JAZN api then you should use following code
// response.sendError( 499, "Oracle SSO" );
}
```

Application Logout: Recommended Code

Most applications that authenticate users have a logout link. In a single-sign-on-enabled application, the user invokes the dynamic directive for logout in addition to other code in the logout handler of the application. Invoking the logout directive initiates single sign-off, or global logout. The example that follows shows what single sign-off code should look like in Java.

```
// Clear application session, if any
String l_return_url := return url to your application e.g. home page
response.setHeader( "Osso-Return-Url", l_return_url);
response.sendError( 470, "Oracle SSO" );
```

A Single Sign-On Software Development Kit

The single sign-on SDK consists of APIs for PL/SQL and Java. You can use these APIs to create partner applications. Appendix B, "Using the PL/SQL and Java APIs", provides code that shows how the APIs are implemented.

This appendix contains the following topics:

- PL/SQL APIs
- Java APIs

Note: The SDK has been deprecated. If you have built applications using the release 9.0.2 SDK, Oracle recommends modifying these for mod_osso. Nevertheless, 9.0.2 applications will continue to work in 9.0.4.

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 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 A–1 Parameters for GENERATE_REDIRECT

Parameter	Description
P_LSNR_TOKEN	Listener token that retrieves registration information about the partner application. The listener token is the host name and port used on the URL for the current request. This token is used to select the appropriate configuration entry in the WWSEC_ENABLER_CONFIG_INFO$ table.
P_URL_REQUESTED	URL requested by the client. Must be URL encoded if it contains a URL parameter. For example: `http://`*`host`*`:`*`port`*`/jsp/order.jsp?itemid=1234&type=purchase`
P_URL_CANCEL	URL that users are directed to if they click **Cancel** on the login page. Must be URL encoded if it contains a URL parameter. For example: `http://`*`host`*`:`*`port`*`/jsp/order.jsp?itemid=1234&type=purchase`
P_FORCED_AUTH	Forced authentication flag.
REDIRECTURL	URL to which the partner application must direct the browser. 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 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 A–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 user name.
P_SSO_USER_DN	Authenticated user DN.
P_SSO_USER_GUID	Authenticated user GUID.
P_SUBSCRIBER_NAME	Subscriber name.
P_SUBSCRIBER_DN	Subscriber DN.
P_SUBSCRIBER_GUID	Subscriber GUID.
P_USER_IPADDRESS	IP address of the user’s machine.
P_SSO_TIMEREMAINING	Remaining session duration.
P_NLS_LANGUAGE	Language selection of the user.
P_NLS_TERRITORY	Territory selection of the 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 A–3 Parameters for GET_ENABLER_CONFIG

Parameter	Description
P_LSNR_TOKEN	Listener token.
P_SITE_TOKEN	Site token.
P_SITE_ID	Site token.

Table A–3 Parameters for GET_ENABLER_CONFIG

Parameter	Description
P_LS_LOGIN_URL	Login URL.
P_LS_LOGOUT_URL	Single sign-off URL.
P_URL_COOKIE_VERSION	URL cookie version.
P_ENCRYPTION_KEY	Encryption key.
P_IPADDR_CHECK	Indicates whether the 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 A–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.
`P_LS_LOGOUT_URL`	Single sign-off URL.
`P_URL_COOKIE_VERSION`	URL cookie version.
`P_ENCRYPTION_KEY`	Encryption key.
`P_IPADDR_CHECK`	Indicates whether the IP address should be verified.

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 A–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.
`P_LS_LOGOUT_URL`	Single sign-off URL.
`P_URL_COOKIE_VERSION`	URL cookie version.
`P_ENCRYPTION_KEY`	Encryption key.
`P_IPADDR_CHECK`	indicates whether the IP address should be verified.

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 A–6 Parameters for DELETE_ENABLER_CONFIG

Parameter	Description
P_LSNR_TOKEN	Listener token. Retrieves registration information about the partner application.

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 A–7 Parameters for ENCRYPT_COOKIE

Parameter	Description
P_LSNR_TOKEN	Listener token. Retrieves registration information about the partner application.

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 A–8   Parameters for DECRYPT_COOKIE

Parameter	Description
P_LSNR_TOKEN	Listener token. Retrieves registration information about the partner application.
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: 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 A–9 lists and describes the exceptions raised by PL/SQL functions and procedures.

Table A–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 *Oracle Application Server Single Sign-On API Reference.*

B

Using the PL/SQL and Java APIs

This appendix provides sample programs that illustrate how to enable partner applications for single sign-on.

The appendix contains the following topics:

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

Before Using the APIs

Before you begin writing partner applications with the single sign-on SDK, make sure that you have correctly installed and configured it. Follow the instructions included with it. The SDK is available at $ORACLE_HOME/sso/lib/ssosdk902.zip.

Writing Partner Applications Using PL/SQL APIs

The example that follows shows how to develop a partner application using PL/SQL APIs. If you need help creating a database access descriptor, see *Oracle Application Server 10G mod_plsql User's Guide*. If you need help writing PL/SQL applications, see *Oracle Application Server 10G PL/SQL Web Toolkit Reference*. 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. 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. The procedure 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.

The sample code for the package papp.pks and papp.pkb is in the file ssosdk902.zip, which is located in demo/plsql.

Note: The request URL and the cancel URL must be URL encoded if these URLs contain a URL parameter. For example:

```
http://host:port/dad/schema.procedure?itemid=1234&type=purchase
```

In PL/SQL, the `wwutl_htf.encode` procedure can be used to encode the URL.

Writing Partner Applications Using Java APIs

Initially, the partner application redirects you to the single sign-on server for authentication and, after successful authentication, sets its own application session cookie. Any subsequent 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 spare the server from having to verify every user request, all partner applications should maintain their own application session.

This section shows how to implement a generic bean that can be used in servlets and JavaServer pages (JSPs). The section contains the following topics

- Servlet Partner Application
- JSP Partner Application

Note: The request URL and the cancel URL must be URL encoded if these URLs contain a URL parameter. For example:

```
http://host:port/jsp/order.jsp?itemid=1234&type=purchase
```

In Java, the java.net.URLEncoder class can be used to encode the URL.

Servlet Partner Application

The example Java servlet provided consists of files that are located in ssosdk902.zip. These are the files:

- Bean classes

  The files SSOEnablerBean.java and SSOEnablerServletBean.java are located in demo/java/beans. Edit these files to suit your deployment.

- Servlet classes

  The files SSOPartnerServlet.java, SSOSignOnServlet.java and SSOPartnerLogoutServlet.java are in demo/java/servlet.

You must compile the bean and servlet files and deploy them in OC4J before you can access your application.

The authentication flow for this application is as follows:

1. The user goes to the SSOPartnerServlet application URL. This servlet retrieves 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 to set the application cookie
   - Redirects the user to the requested application URL using SSOEnablerServletBean. The servlet uses the application cookie to shows user information.

JSP Partner Application

The example JSP partner application also consists of files that are located in ssosdk902.zip. These are the files:

- Bean classes

  The files SSOEnablerJspBean.java and SSOEnablerServletBean.java are located in demo/java/beans. Edit these files to suit your deployment.

- JSP files

  The files ssoinclude.jsp, ssosignon.jsp, papp.jsp, and papplogoff.jsp are located in demo/java/jsp.

You must compile bean java files and then deploy them with JSP files in OC4J before you can access your application. For detailed information about compilation, see ssosdk902.zip.

The authentication flow for this application is as follows:

1. The user goes to the papp.jsp page.
2. papp.jsp retrieves 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.

C

Adding and Editing SDK-Enabled Applications

The Administer Partner Applications page, accessible as a link on the SSO Server Administration page, is used to register and edit applications integrated with the single sign-on SDK. This page is also useful for viewing mod_osso configuration information. Starting with release 9.0.2, mod_osso is registered automatically by the installer.

If you use the UI to manually register an SDK-enabled partner application, bear in mind that you are registering the application only with the single sign-on server. To register the application in the partner application database, you must manually copy the registration information from the single sign-on UI.

This appendix contains the following topics:

- Add Partner Application Page
- Edit Partner Application Page

Add Partner Application Page

Selecting the Add Partner Applications link on the administration pages takes you to the Create Partner Applications page. Use the fields on this page, described in the tables immediately following, to register an application with the single sign-on server. Once you add the application, it appears with existing applications in a list sorted by date.

Table C–1 Partner Application Login

Field	Description
Name	Enter a unique name for the partner application.
Home URL	Enter the URL of the home page for the application.
Success URL	Enter the URL to the routine responsible for establishing the partner application's session and session cookies. This routine should redirect the browser to the URL that the user originally requested. The URL must point to a procedure that processes the user identification information from the single sign-on server. Include the `http://` prefix in the URL. The following example shows a success URL for OracleAS Portal: `http://server.domain.com:5000/pls/DAD/portal.wwsec_app_priv.process_signon`
Logout URL	Enter the URL for the logout routine of the application. The single sign-off page invokes this URL in parallel with others, enabling users to simultaneously log out of the server and active partner applications.

Table C–2 Valid Login Timeframes

Field	Description
Start Date	Enter the date when users are first able to access the partner application through the single sign-on server. Use the format shown next to the field label.
End Date	Enter the date when users are last able to access the partner application through the single sign-on server. Use the format shown next to the field label. **Note:** If you leave this field blank, this partner application is valid for an indefinite period.

Table C–3 Application Administrator

Field	Description
Administrator E-mail	Enter the e-mail address of the administrator of the partner application.
Administrator Information	Enter any additional information that you want to include about the administrator of the partner application.

Edit Partner Application Page

The Edit Partner Application page contains all of the fields that are on the Create Partner Application page, plus five additional fields in the **Partner Application Login** section. Table C–4 on page C-3 describes the additional fields.

Table C–4 Fields on the Edit Partner Application Page

Field	Description
ID	The ID value is automatically set when a partner application is added. It is used by the single sign-on server to identify the partner application.
Token	The token is automatically set when a partner application is added. It is used by the single sign-on server to identify the partner application. The partner application must use the application token to identify itself to the single sign-on server when requesting authentication.
Encryption Key	The encryption key for the partner application.
Login URL	This is the same as the success URL, which sets the application session and cookies.
Single Sign-Off URL	This is the same as the URL for application logout.

Use the following steps to edit a partner application:

1. From the Administer Partner Applications page, choose an application from the list that appears under the **Edit/Delete Partner Application** heading.
2. Click the **Edit** link for that application. This link appears as a pencil icon.
3. On the Edit Partner Application page, edit the field values in Table C–1. These are the only values that can be edited.
4. Click **Apply** to store changes for the current screen and to display the updated screen. Click **OK** to store all changes and to return to the previous screen.

D User Attributes Passed to Partner Applications

Table D–1 lists all of the user attributes that mod_osso passes to applications. The table also recommends attributes to use as keys, or handles, to retrieve additional user attributes from Oracle Internet Directory.

Table D–1 User Attributes Passed to Partner Applications

HTTP Header Name	Description	Source	Use as Key or Handle?
`Osso-User-Guid`	Single sign-on user's globally unique user ID (GUID)	Single sign-on user's globally unique user ID (GUID)	Recommended
`Osso-Subscriber-Guid`	Realm GUID	Realm entry in Oracle Internet Directory	Recommended
`Remote-User`	User nickname as entered by user on the login page	Single sign-on login page	Recommended for pre-9.0.4 applications, but not for 9.0.4 applications.
`Osso-Subscriber`	User-friendly name for a realm	Realm entry in Oracle Internet Directory	Recommended for pre-9.0.4 applications, but not for 9.0.4 applications.

Table D–1 User Attributes Passed to Partner Applications

HTTP Header Name	Description	Source	Use as Key or Handle?
`Accept-Language`	Language and territory in ISO format	Single sign-on server	Not applicable
`Osso-User-Dn`	Single sign-on user's distinguished name (DN)	User entry in Oracle Internet Directory	Not recommended. Use GUID headers to perform user searches in Oracle Internet Directory.[1]
`Osso-Subscriber-DN`	Realm DN	Realm entry in Oracle Internet Directory	Not recommended. Use GUID headers to perform user searches in Oracle Internet Directory.[2]

[1] Deprecated. To be removed in the next release.

[2] Deprecated. To be removed in the next release.

Glossary

dads.conf

File on the Oracle HTTP server that is used to configure a database access descriptor.

database access descriptor

Database connection information for a particular OracleAS component such as the the single sign-on schema.

dynamic directive

HTTP response header that uses special error codes to enable an application to request single sign-on functionality without having to implement the single sign-on protocol.

external application

An application that does not delegate authentication to the single sign-on server. Instead, it displays an HTML login form that asks for an application user name and password. At the first login, you can choose to have the server retrieve these credentials for you. Thereafter, you are logged in to the application transparently.

forced authentication

The act of forcing you to reauthenticate if you have been idle for a preconfigured amount of time. OracleAS Single Sign-On enables you to specify a global user inactivity timeout. This feature is intended for installations that have sensitive applications.

GET

Authentication method whereby login credentials are submitted as part of the login URL.

HTTP response headers

Data embedded in a single sign-on application that instructs the mod-osso-enabled single sign-on server to perform some action such as forced authentication or single sign-off. Absent such headers, applications must use the single sign-on SDK to interact with the single sign-on system.

httpd.conf

File used to configure the Oracle HTTP Server.

mod_oc4j

Oracle HTTP module that communicates Web requests to the OC4J engine.

mod_osso

Module on the Oracle HTTP server that enables applications protected by OracleAS Single Sign-On to accept HTTP headers in lieu of a user name and password once the user has logged into the single sign-on server. The values for these headers are stored in the mod_osso cookie.

mod_osso cookie

User data stored on the HTTP server. The cookie is created when a user authenticates. When the same user requests another application. The Web server uses the information in the mod_osso cookie, not the single sign-on cookie, to log the user in to the application. This feature speeds server response time.

OC4J (Oracle Containers for J2EE)

A lightweight, scalable container for Java2 Enterprise Edition.

Oracle HTTP Server

Software that processes Web transactions that use the Hypertext Transfer Protocol. Oracle uses HTTP software developed by the Apache Group.

partner application

An application that delegates the authentication function to the single sign-on server. This type of application spares you from reauthenticating by accepting mod_osso headers or by redirecting you to the server itself. To redirect users itself, the application must be integrated with the single sign-on SDK.

POST

Authentication method whereby login credentials are submitted within the body of the login form.

single sign-off

Process by which you terminate a single sign-on session and log out of all active partner applications simultaneously. You do this by logging out of the application you are working in.

single sign-on SDK

The APIs that enable partner applications for single sign-on. The SDK consists of PL/SQL and Java APIs as well as sample code that demonstrates how these APIs are implemented.

single sign-on server

Program logic that enables you to log in securely to single sign-on applications such as expense reports, mail, and benefits.

Index