# Oracle Single Sign-On

Application Developer's Guide

Release 3.0.6

November 21, 2000

Part No. A86782-03

ORACLE

Oracle Single Sign-On Application Developer's Guide, Release 3.0.6

Part No. A86782-03



Primary Author: Ted Burroughs

Contributing Authors: Cindee Kibbe, Richard Smith

Contributors: Gaurav Bhatia, Kamalendu Biswas, Paul Encarnacion, Naresh Kumar, Arun Swaminathan

# Contents

## 3 PL/SQL Single Sign-On Application Programming Interface



## 4 Java Oracle Single Sign-On Application Programming Interface

## 5 Examples in PL/SQL and Java

## Index

# Send Us Your Comments

**Oracle Single Sign-On Application Developer's Guide, Release 3.0.6**

**Part No. A86782-01**

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) 506-7228 Attn: Server Technologies Documentation Manager
- Postal service:
  Oracle Corporation
  Server Technologies Documentation
  500 Oracle Parkway, Mailstop 4op12
  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 Single Sign-On Application Developer's Guide* provides the information you need to understand and use the Single Sign-On product and its related applications.

This preface contains these topics:

## Audience

*Oracle Single Sign-On Application Developer's Guide* is intended primarily for application developers responsible for integrating Single Sign-On with partner applications.

*Oracle Single Sign-On Application Developer's Guide* is also provided for anyone who wants to understand how Single Sign-On works.

## Organization

This document contains:

Chapter 1, "Introduction" explains how Single Sign-On solves the problems associated with using and administering user names and passwords for multiple applications in an enterprise.

Table 2, "Concepts and Architecture" discusses the significance of Single Sign-On to users and administrators in an enterprise. It describes the components of Single Sign-On, the application types, and the authentication methods Single Sign-On uses. It also explains the process and architecture through which Single Sign-On authenticates users to applications.

Chapter 3, "PL/SQL Single Sign-On Application Programming Interface" explains how to use the PL/SQL Single Sign-On Application Programming Interface (API).

Chapter 4, "Java Oracle Single Sign-On Application Programming Interface" explains how to use the Java Single Sign-On Application Programming Interface.

Chapter 5, "Examples in PL/SQL and Java" explains how to install Application Programming Interfaces for PL/SQL and Java and gives examples of installation code.

## Related Documentation

For more information about development related issues, refer to the Readme file included in the Software Development Kit (SDK).

For additional information, see the online help and related documentation for Oracle Portal.

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://technet.oracle.com/membership/index.htm`

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://technet.oracle.com/docs/index.htm`

# 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. | The C datatypes such as **ub4**, **sword**, or **OCINumber** are valid.<br>When you specify this clause, you create an **index-organized table**. |
| *Italics* | Italic typeface indicates book titles, emphasis, syntax clauses, or placeholders. | *Oracle8i Concepts*<br>You can specify the *parallel_clause*.<br>Run `U`*`old_release`*`.SQL` where *old_release* refers to the release you installed prior to upgrading. |

| Convention | Meaning | Example |
|---|---|---|
| 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, user names, and roles. | You can specify this clause only for a `NUMBER` column.<br>You can back up the database using the `BACKUP` command.<br>Query the `TABLE_NAME` column in the `USER_TABLES` data dictionary view.<br>Specify the `ROLLBACK_SEGMENTS` parameter.<br>Use the `DBMS_STATS.GENERATE_STATS` procedure. |
| lowercase monospace (fixed-width font) | Lowercase monospace typeface indicates executables 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, user names and roles, program units, and parameter values. | Enter `sqlplus` to open SQL*Plus.<br>The `department_id`, `department_name`, and `location_id` columns are in the `hr.departments` table.<br>Set the `QUERY_REWRITE_ENABLED` initialization parameter to `true`.<br>Connect as `oe` user. |

## 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}`<br>`[COMPRESS \| NOCOMPRESS]` |

| Convention | Meaning | Example |
|---|---|---|
| . . . | Horizontal ellipsis points indicate either:<br>■ That we have omitted parts of the code that are not directly related to the example<br>■ That you can repeat a portion of the code | `CREATE TABLE ... AS subquery;`<br>`SELECT col1, col2, ... , col`*`n`* `FROM employees;` |
| .<br>.<br>. | 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 it is shown. | `acctbal NUMBER(11,2);`<br>`acct CONSTANT NUMBER(4) := 3;` |
| *Italics* | Italicized text indicates variables for which you must supply particular values. | `CONNECT SYSTEM/`*`system_password`* |
| 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;`<br>`SELECT * FROM USER_TABLES;`<br>`DROP TABLE hr.employees;` |
| lowercase | Lowercase typeface indicates programmatic elements that you supply. For example, lowercase indicates names of tables, columns, or files. | `SELECT last_name, employee_id FROM employees;`<br>`sqlplus hr/hr` |

# 1

# Introduction

This chapter explains how Single Sign-On solves the problems associated with using and administering user names and passwords for multiple applications in an enterprise.

This chapter contains these topics:

# What Is Single Sign-On?

Single Sign-On is a service of the Oracle9i Application Server that enables:

- Authentication to all appropriate applications in an enterprise by entering a user name and password only once
- Centralized administration of user name and password combinations for all users in an enterprise

This section contains these topics:



## The Problem of Too Many Passwords

Within any given enterprise, a typical user accesses several applications: one, for example, to create expense reports, another to use email, and still another to schedule appointments. Each application requires the user to enter a valid user name and password, which presents three major difficulties:

- Inconvenience:

  A user must enter a user name and password to access each and every application. Moreover, it can be difficult to remember the user name and password combinations for multiple applications.

- Poor security:

  To remember so many user name and password combinations, users often use one of two strategies:

  - They use the same combination for all applications. This makes it possible for a thief who steals that combination to access all of the user's applications.
  - They use multiple combinations, writing them on pieces of paper that can be lost, stolen, or observed. The more user name and password combinations, the greater the risk that one or more of them may be lost or stolen.

- Difficulty of administration:

  It can be costly and difficult to administer password stores for multiple applications. To create or delete a user, or change a password, an administrator must tediously make changes in each application.

## The Single Sign-On Solution

With Single Sign-On, users typically sign on to a centrally administered Login Server through a central Web portal. Once it authenticates a particular user, the Login Server displays links to all the applications for that user.

Using a central Web portal with a centrally administered Login Server has these advantages:

- Convenience:

  The user enters the user name and password only once, at a central corporate Web portal, to access all the needed applications. From the user's perspective, authentication to each application happens transparently.

- Increased security:

  Fewer user name and password combinations lowers the risk of a thief stealing them and gaining access to a user's restricted information.

- Ease of administration:

  Single Sign-On provides centralized provisioning of user accounts, so that administrators can easily create new user accounts.

  Centralizing the authentication process also makes it possible to support additional authentication mechanisms in a localized manner. For example, you can implement an LDAP-based authentication, or digital certificate-based authentication, and the change would be localized to the Login Server.

# 2

# Concepts and Architecture

This chapter describes the components of Single Sign-On, the kinds of applications to which it can provide access, and the authentication methods it uses. It explains the process and architecture through which Single Sign-On authenticates users to applications.

This chapter contains these topics:

# Components of Single Sign-On

Single Sign-On has two components:

- Login Server
- Single Sign-On Application Programming Interface (API)

## Login Server

The first time that a user seeks access to an application, the Login Server:

- Authenticates the user by means of user name and password
- Passes the client's identity to the various applications
- Marks the client being authenticated with an encrypted login cookie

In subsequent user logins, this login cookie provides the Login Server with the user's identity, and indicates that authentication has already been performed. If there is no login cookie, then the Login Server presents the user with a login challenge.

To guard against sniffing, the Login Server can send the login cookie to the client browser over an encrypted SSL channel.

The login cookie expires with the session, either at the end of a time interval specified by the administrator, or when the user exits the browser. It is never written to disk.

A partner application can expire its session through its own explicit logout.

---

**Note:** To logout of a partner application and log in as another user, you must also log out of the Login Server session. Otherwise, the authentication request returns the partner application to the logged in state of the previous user.

---

## Single Sign-On Application Programming Interface (API)

The Single Sign-On API enables:

- Applications to communicate with the Login Server and to accept a user's identity as validated by the Login Server
- Administrators to manage the application's association to the Login Server

# Single Sign-On Application Types

There are two kinds of applications to which Single Sign-On provides access:

- Partner Applications
- External Applications

## Partner Applications

Partner applications are integrated with the Login Server. They contain a Single Sign-On API that enables them to accept a user's identity as validated by the Login Server.

## External Applications

External applications are web-based applications that retain their authentication logic. They do not delegate authentication to the Login Server and, as such, require a user name and password to provide access. Currently, these applications are limited to those which employ an HTML form for accepting the user name and password. The user name may be different from the SSO user name, and the Login Server provides the necessary mapping.

# Single Sign-On Authentication Methods

Single Sign-On can use one of these authentication methods:

***Table 2-1 Single Sign-On Authentication Methods***

| | |
|---|---|
| Local user authentication | Uses a lookup table within the Login Server schema. This table contains user name, password, Login Server privilege level, and other auditing fields for the user. The incoming password is one-way hashed and compared to the entry in the table. |
| External repository authentication | Typically relies on an LDAP-compliant directory. In this case, the Login Server binds to the LDAP-compliant directory, then looks up the user credentials stored there. External Authentication includes LDAP and Database Authentication and any others that may be custom-developed. |

# How **Single Sign-On Works**

Whenever a user accesses either a partner application or an external application, the Login Server first authenticates that user.

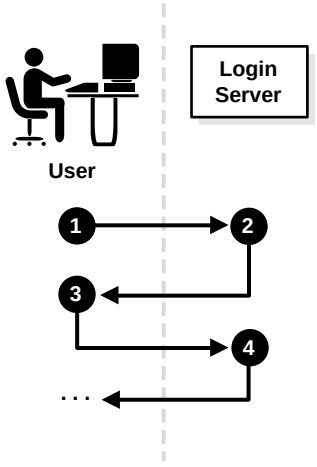
This section contains these topics:

- Authenticating to the Login Server
- Accessing a Partner Application
- Accessing an External Application

## Authenticating to the Login Server

The Login Server authenticates a user in this way:

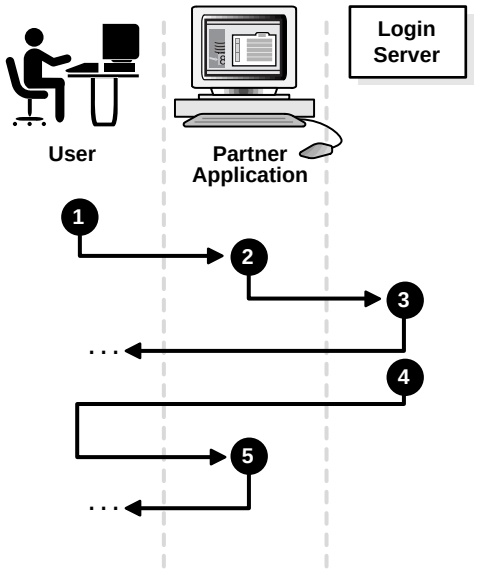
1. The Login Server checks for a login cookie. If one is present, the Login Server identifies the user from the encrypted information in the login cookie.
2. If a login cookie is not present, the Login Server prompts the user for the user's credentials.
3. The user provides the user name and password.
4. The Login Server authenticates the user by passing the provided name and password to the configured authentication routine—either the local routine or one provided by an external authentication module for an external repository. If the authentication is successful, the Login Server establishes a login cookie on the client browser to facilitate Single Sign-On for future authentication requests.




**See Also:** "Login Server" for information on the login cookie

## Accessing a Partner Application

When a user seeks access to a partner application, the following steps occur:




1. The user seeks access to the partner application directly.

2. If this is the first time during a session that the user is accessing this partner application, then the partner application transparently directs the user to the Login Server to obtain authentication credentials.

3. The Login Server authenticates the user as described in "Authenticating to the Login Server."

4. The Login Server transparently directs the user to the partner application. It does this by using a URL with an encrypted parameter containing the user's identity.

5. The partner application:

   —Decrypts the parameter

   —Identifies the user

   —Establishes its own session management

**Note:** In Step 2 of this process, the partner application directs the user to the Login Server only if the application requires it based on the URL requested. Some URLs may be public and no redirection to the Login Server is necessary. When it is necessary, the partner application must protect itself from unauthenticated access by using its own session management.

If, during the same session, the user again seeks access to the same or to a different partner application, then the Login Server does not prompt the user for user name and password. Instead, the Login Server obtains that information from the login cookie on the client browser.

## Partner Application Development Requirement

- To implement an authentication check:

  1. Protected URLs need to check for an application session cookie for authorization.

2. If no application session cookie exists, then the browser redirects the user to the Single Sign-On server.
3. If the URL is publicly accessible, then no authorization check is implemented.

- To implement a sign-on URL:
  1. This URL must establish an application session cookie using the identity information sent by the Single Sign-On server.
  2. The browser then redirects the user to the requested URL

## Accessing an External Application

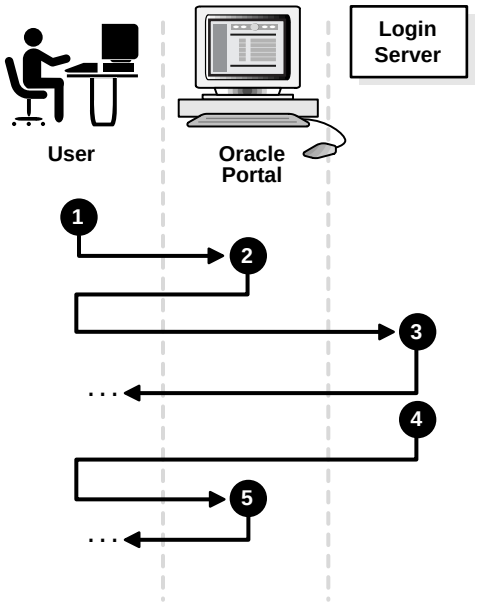
You can accessing an external application through Oracle Portal. In this scenario, Oracle Portal functions as a partner application.

This section contains these topics:



### Authenticating to Oracle Portal

When a user seeks access to an external application by way of Oracle Portal, Single Sign-On authenticates the user to Oracle Portal through this process:




1. The user seeks access to the Oracle Portal site.

2. If this is the first time during a session that the user is accessing Oracle Portal, then Oracle Portal transparently directs the user to the Login Server to obtain authentication credentials.

3. The Login Server authenticates the user as described in "Authenticating to the Login Server."

4. The Login Server transparently directs the user to Oracle Portal. It does this by using a URL with an encrypted parameter containing the user's identity.

5. Oracle Portal:

   —Decrypts the parameter

   —Identifies the user

   —Establishes its own session management

   —Presents the user with links to the external applications

If, during the same session, the user again seeks access to Oracle Portal, then the Login Server does not prompt the user for user name and password. Instead, it obtains that information from the login cookie on the client browser.

### Authenticating to an External Application for the First Time

Single Sign-On uses the process described in the next figure under these conditions:

- The user has authenticated to the Oracle Portal
- The user is accessing an external application for the first time through Oracle Portal

1. Oracle Portal presents to the user links to external applications. These links invoke a routine on the Login Server.

2. A user clicks one of the links.

3. The user's clicking a link invokes on the Login Server the external application login procedure. This procedure checks the Login Server password store for the user's credentials for the requested external application. If it finds that the user has no such credentials, then the Login Server prompts the user for them.

4. The user enters the user name and password. The user can also indicate whether to save these credentials in the Login Server password store.

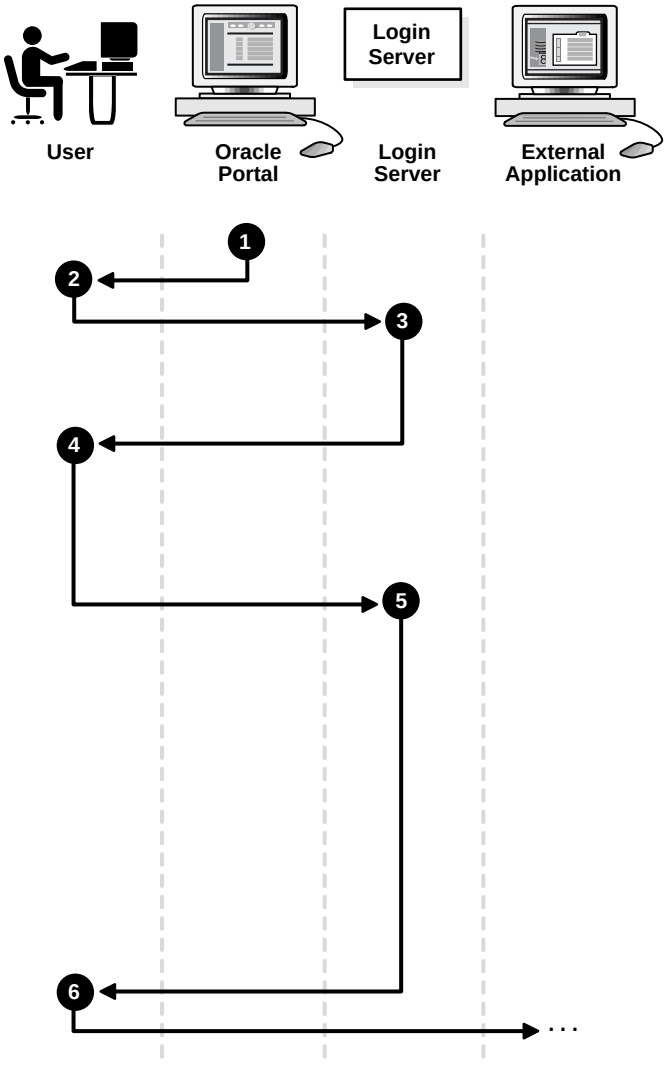
5. If the user chooses to save the credentials in the Login Server password store, then the Login Server saves them. The Login Server performs the following tasks:

—Constructs a login page using the user's credentials

—Formulates the form to post to the external application login processing routine. This routine has been preconfigured by the Login Server administrator and associated with the requested application.

—Sends the form to the client browser, with a directive to post it immediately to the external application

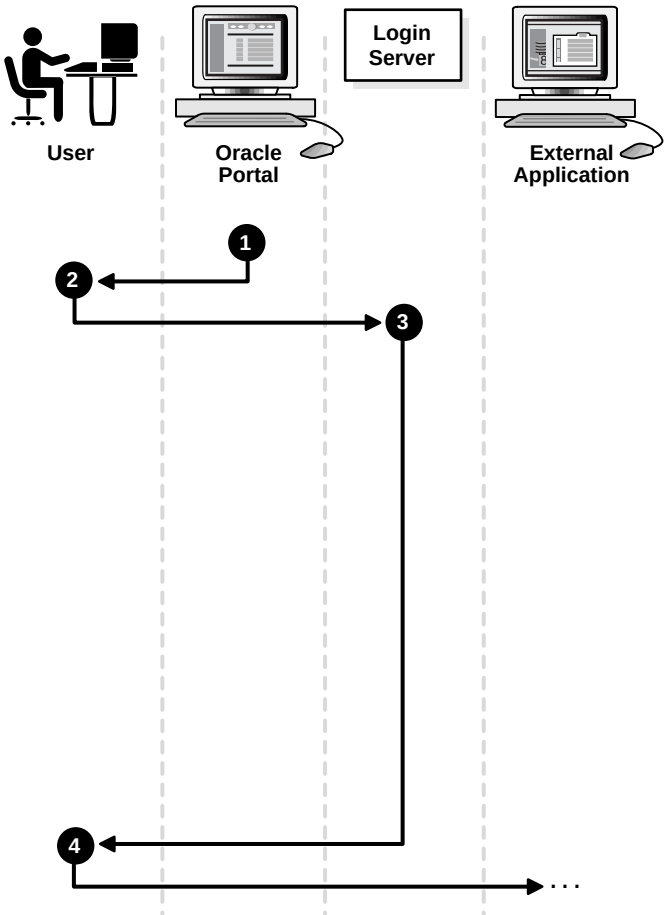
6. The client posts the form to the external application and logs in.



### Authenticating to an External Application After the First Time

Single Sign-On uses the process described in the next figure if the user:

- Has authenticated to the Oracle Portal
- Has a user name and password in the Login Server password store
- Is accessing an external application after the first time




1. Oracle Portal presents to the user links to external applications. These links invoke a routine on the Login Server.

2. A user clicks one of the links.

3.The user's clicking a link invokes on the Login Server the external application login procedure. This procedure checks the password store for any credentials the user has for the requested external application.

The Login Server then:

—Constructs a login page using the user's credentials

—Formulates the form to post to the external application's login processing routine. This routine has been preconfigured by the Login Server administrator and associated with the requested application.

—Sends the form to the client browser, with a directive to post it immediately to the external application

4. The client posts the form to the external application and logs in.

If the user has not stored a user name and password in the Login Server password store, then Single Sign-On follows the process described in "Authenticating to an External Application for the First Time".

# 3

# PL/SQL Single Sign-On Application Programming Interface

This chapter explains how to use the PL/SQL Single Sign-On Application Programming Interface.

This chapter contains these topics:

# Developing Partner Applications

The information in this section allows you to enable applications to participate in Single Sign-On by becoming partner applications. It discusses the application restructuring required. It also describes the basic architecture and explains where the API calls in this package are to be used.

This section contains these topics:



## How a Partner Application Works

Partner applications delegate user authentication to the Login Server. When the application determines that this delegation is needed, it uses `WWSEC_SSO_ENABLER_PRIVATE.GENERATE_REDIRECT` to obtain the URL to which it performs the redirect.

As a result of this redirect, the Login Server:

- Authenticates the user.
- Calls a procedure that it is configured to call in the partner application. It makes this call by redirecting the browser to the specified procedure.

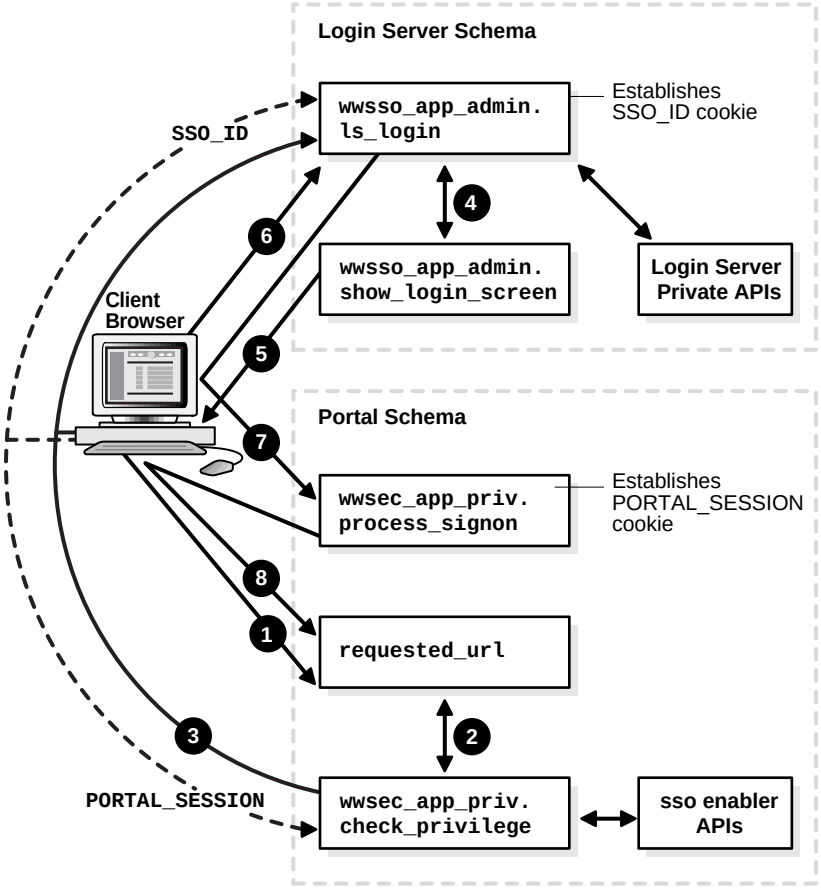
The procedure that the Login Server calls has a single `VARCHAR2` parameter that has the default name `URLC` (an abbreviation for URL Cookie). This procedure should parse the encrypted value that is passed in this parameter by using the `WWSEC_SSO_ENABLER_PRIVATE.PARSE_URL_COOKIE` procedure. This parsing enables the partner application to obtain the `ssousername` and the `urlrequested` from the parameter.

The partner application should then establish an application session for the `ssousername` obtained from the parameter. Typically, this means that the application establishes a cookie for its own use, and may set up some session information internally. The application can then redirect to the `urlrequested` that is typically the protected URL that the user seeks to access.

Each application must develop its own convention for protecting pages that need authentication. Ideally, a well-defined procedure or function is established that all components of the application can use for access control. When it is determined that the user needs to be authenticated, then the application can perform the redirect to the Login Server from the single centralized place.

### The Single Sign-On Process for a Partner Application

The figure that follows illustrates the Login Server authentication sequence used by Oracle Portal as a partner application.



1. The browser requests a URL. This URL could be a PL/SQL procedure in Oracle Portal. It also could be indirectly a PL/SQL procedure of the `WWDOC_PROCESS.PROCESS_DOWNLOAD` procedure that is invoked when requesting a document resident in Oracle Portal.

   If the procedure has no security, then it simply returns HTML, resulting in the display of the page.

2. If the procedure applies access restrictions, then it calls internal Oracle Portal security APIs to check whether the current user has privileges on the current procedure. Typically, this is accomplished by calling an Oracle Portal security function, `WWSEC_APP_PRIV.CHECK_PRIVILEGE`. This function in turn checks whether the user is logged on by inspecting the current session information obtained from the `portal_session` cookie. If the user is not logged in, and, as a public user, has insufficient privileges to execute the procedure, then `WWSEC_APP_PRIV.CHECK_PRIVILEGE` must invoke the login sequence. Note that there will always be a `portal_session` cookie, because the Oracle Portal gateway establishes a public session if it cannot find an existing cookie. The cookie name is specified in the DAD, and if not specified, defaults to the DAD name.

3. To initiate the login sequence, Oracle Portal generates a redirect request to the Login Server, using enabler APIs provided by the Login Server to generate the token that is passed as a URL parameter. The token contains the name of the partner application, the URL that was requested in the partner application, and, optionally, another URL to return to if authentication is canceled by the user. The Login Server `LS_LOGIN` procedure checks for an SSO_ID cookie, referred to as the login cookie. It checks whether this specific browser has already performed a Login Server authentication within this session. If it has, then the Login Server uses the information in the login cookie and does not provide the user with another authentication challenge.

4. If the login cookie is not present, then the Login Server calls the `SHOW_LOGIN_SCREEN` procedure.

5. The Login Server presents a login page to the user, prompting the user for a user name and password.

6. The user enters a user name and password and submits the form.

   The Login Server then authenticates the user's user name and password, using the configured authentication mechanism.

   If the authentication fails, then the login page is displayed again with an error message.

If the user clicks Cancel on the login page, then the Login Server redirects the page to the cancel URL provided in the initial request (in Step 3).

If the authentication is successful, then the Login Server establishes a login cookie. The default name for this cookie is `SSO_ID`. It keeps track of the user name of the user that logged in, and the session expiry time.

7. The Login Server constructs a URL with an encrypted parameter containing the user's identity for processing by the partner application. The Login Server sends this URL to the partner application with the purpose of:

   - Informing the partner application that the user has successfully authenticated
   - Providing the user name with which the user authenticated
   - Returning the URL that is being requested by the user

   The URL to which this parameter is passed is stored in the Login Server configuration table. The Partner Application entry specifies:

   - The URL that will process this parameter
   - The name of the parameter itself

   In Oracle Portal 3.0, the name of the procedure that processes this is `WWSEC_APP_PRIV.PROCESS_SIGNON`. The parameter name is `URLC`. This procedure uses the `WWSEC_SSO_ENABLER_PRIVATE.PARSE_URL_COOKIE` API to get the SSO user name and the requested URL. When this procedure is invoked, Oracle Portal converts the `portal_session` cookie to an authenticated cookie, updating the user name with the logged in user's name, and, if necessary, updating the associated `db_user`. Also, the `WWCTX_SSO_SESSION$` table is updated with the updated session information. The session is then flagged as logged on.

8. Finally, the `PROCESS_SIGNON` procedure redirects the browser to the URL initially requested by the user. When invoking this page, the `CHECK_PRIVILEGE` function is again invoked, and, because the user is now logged in, it is possible to check whether the user has sufficient privilege, by using the APIs that query the Oracle Portal `WWSEC_SYS_PRIV$` table, to invoke the procedure.

   If the user has sufficient privileges, then the procedure executes. If the user does not have sufficient privileges, then an error page indicating insufficient privileges is displayed.

When a user later seeks access to secured pages, the `CHECK_PRIVILEGE` procedure sees the authenticated `portal_session` cookie, and does not need to interact with the Login Server. Instead, it uses the privilege APIs to determine whether the user has sufficient access privileges.

## Functions and Procedures

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

This section contains these topics:

### PAPP_SHOW_CONFIG Procedure

This procedure returns enabler configuration information for a partner application.

**Syntax**

```
PROCEDURE PAPP_SHOW_CONFIG
(
    P_LSNR_TOKEN IN VARCHAR2
ENABLER_CONFIG IN OUT sec_enabler_config_type
);
```

***Table 3–1 PAPP_SHOW_CONFIG Procedure Parameters***

| Parameter | Description |
|---|---|
| P_LSNR_TOKEN | Listener token to get the necessary partner application registration configuration |
| ENABLER_CONFIG | Enabler configuration type |

#### Example

```
wwsec_sso_enabler_private.papp_show_config
(
   p_lsnr_token =>  listener token
   enabler_config => enabler configuration type
);
```

### GENERATE_REDIRECT Function (URL Cookie Version V1.0)

This function generates a redirect URL along with SITE2PSTORETOKEN that the Login Server will parse.

#### Syntax

```
FUNCTION GENERATE_REDIRECT
(
   P_LSNR_TOKEN IN VARCHAR2,
   URLREQUESTED IN VARCHAR2,
   URLONCANCEL IN VARCHAR2
) RETURN VARCHAR2;
```

***Table 3–2 GENERATE_REDIRECT Function Parameters***

| 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. |
| URLREQUESTED | URL requested by the client for which authentication is needed |
| URLONCANCEL | URL to go to if client clicks cancel on the login page |

***Table 3–3 GENERATE_REDIRECT Function Return Values***

| Return Value | Description |
|---|---|
| REDIRECTURL | URL to which the partner application must direct the browser to delegate authentication to the Login Server. This URL contains the request for authentication. |

**Example**

```
WWSEC_SSO_ENABLER_PRIVATE.GENERATE_REDIRECT
(
    p_lsnr_token => listener token
   ,urlrequested => URL requested by the client for which authentication is
                    needed
   ,urloncancel => URL to go to if client clicks cancel on the login page
);
```

**Note:** Depending on the architecture of the system, it may be necessary for an application to be accessible through multiple web addresses. The partner application establishes an application session cookie to keep track of authenticated sessions. Since cookies have scoping properties, the session cookie needs to be scoped to the appropriate web address.

When a partner application requests authentication, the `GENERATE_REDIRECT` function creates the `site2pstoretoken` parameter, containing the ID of the partner application (`site_id`, `site_token`). This is used to look up the appropriate partner configuration on the Login Server. Also in the Login Server's partner configuration data is the URL that should be called on a successful authentication to establish the partner application's session. The URL for this 'Success URL' must have the same cookie scope (since it will be generating the cookie from this URL) as the requested URL. For this reason, each entry in the partner's configuration table must have a corresponding entry in the Login Server partner configuration file. The `p_lsnr_token` is what is used by the partner application to look up the appropriate configuration entry based on the current request. To establish the correct cookie scope, it needs to use a `p_lsnr_token`, which will retrieve the appropriate enabler entry. Typically, the `p_lsnr_token` should be the `hostname.domain:port` of the current request if the cookie path is scoped to the root "/." (without quotes) Otherwise, if the cookie is scoped down to a path, then the `p_lsnr_token` should include a path as well.

## PARSE_URL_COOKIE Function (URL Cookie Version V1.0)

This function parses the URL cookie that is generated by the `GENERATE_REDIRECT` function on the Login Server side.

### Syntax

```
PROCEDURE parse_url_cookie
(
   P_LSNR_TOKEN        IN VARCHAR2,
   ENCRYPTED_URLCOOKIE IN VARCHAR2,
   SSOUSERNAME      IN OUT VARCHAR2,
   IPADD            IN OUT VARCHAR2,
   SSOTIMEREMAINING IN OUT NUMBER,
   SITETIMESTAMP    IN OUT DATE,
   URLREQUESTED     IN OUT VARCHAR2,
   NEWSITEKEY       IN OUT VARCHAR2
);
```

***Table 3–4 PARSE_URL_COOKIE Function Parameters***

| Parameter | Description |
|---|---|
| P_LSNR_TOKEN | Listener token to get the necessary partner application registration configuration |
| ENCRYPTED_URLCOOKIE | URL cookie |
| SSOUSERNAME | SSO user name |
| IPADD | IP Address of user |
| SSOTIMEREMAINING | Time remaining on SSO session |
| SITETIMESTAMP | Timestamp at cookie generation |
| URLREQUESTED | URL that the client is attempting to access |
| NEWSITEKEY | Reserved for future use |

#### Example

```
WWSEC_SSO_ENABLER_PRIVATE.PARSE_URL_COOKIE
(
    p_LSNR_TOKEN     => listener token
   ,ENCRYPTED_URLCOOKIE => URL cookie
   ,SSOUSERNAME      => ssousername
   ,IPADD            => IP Address of user
   ,SSOTIMEREMAINING => time remaining on SSO session
   ,SITETIMESTAMP    => timestamp at cookie generation
   ,URLREQUESTED     => URL that the client is authenticated to access
   ,NEWSITEKEY       => reserved for future use
);
```

### GET_ENABLER_CONFIG Function

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

#### Syntax

```
PROCEDURE get_enabler_config
(
    P_LSNR_TOKEN IN VARCHAR2
) RETURN sec_enabler_config_type;
```

***Table 3–5 GET_ENABLER_CONFIG Function Parameters***

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

#### Example

```
WWSEC_SSO_ENABLER_PRIVATE.GET_ENABLER_CONFIG
(
   p_lsnr_token => listener token
)
```

### CREATE_ENABLER_CONFIG Procedure

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

### Syntax

```
PROCEDURE create_enabler_config
(
    P_CONFIG IN sec_enabler_config_type
);
```

***Table 3–6 CREATE_ENABLER_CONFIG Procedure Parameters***

| Parameter | Description |
|---|---|
| P_CONFIG | sec_enabler_config_type object which contains partner application registration information |

### Example

```
WWSEC_SSO_ENABLER_PRIVATE.CREATE_ENABLER_CONFIG
(
   p_config => sec_enabler_config_type object
)
```

## UPDATE_ENABLER_CONFIG Procedure

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

### Syntax

```
PROCEDURE update_enabler_config
(
    P_LSNR_TOKEN IN VARCHAR2,
    P_CONFIG     IN sec_enabler_config_type
);
```

***Table 3–7 UPDATE_ENABLER_CONFIG Procedure Parameters***

| Parameter | Description |
|---|---|
| P_LSNR_TOKEN | Listener token to get the necessary partner application registration configuration |
| P_CONFIG | sec_enabler_config_type object which contains partner application registration information |

#### Example

```
WWSEC_SSO_ENABLER_PRIVATE.UPDATE_ENABLER_CONFIG
(
   p_lsnr_token   => listener token
   ,p_config      => sec_enabler_config_type object
)
```

## 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 3–8 DELETE_ENABLER_CONFIG Procedure Parameters***

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

#### Example

```
WWSEC_SSO_ENABLER_PRIVATE.DELETE_ENABLER_CONFIG
(
   p_lsnr_token =>  listener token
)
```

# Exceptions

This section lists and describes the exceptions raised by the procedures and functions in this chapter.

***Table 3–9 Exceptions***

| Exception | Description |
|---|---|
| COOKIE_DECRYPTION_FAILED | Decryption of URL parameter failed. |
| COOKIE_ENCRYPTION_FAILED | Creation of site2pstore token failed. |
| DUP_ENABLER_EXCEPTION | Site with the same name already exists. |
| ENABLER_CONFIG_NOT_FOUND | Could not find configuration information for partner site. |
| MANDATORY_ATTRIBUTE_IS_NULL | Login Server registration information is not correct. |
| INVALID_IP_ADDRESS | IP address contained in the URLC does not match the client's IP address. |
| UNSUPPORTED_COOKIE_VERSION | Cookie version is not supported. |
| URL_COOKIE_EXPIRED | URL cookie is timed out. The URLC parameter will time out if the user takes more than about 5 minutes to log in. |

## Datatype and Table Definitions

### SEC_ENABLER_CONFIG_TYPE

This is the object type for partner application configuration.

```
CREATE OR replace TYPE sec_enabler_config_type AS object
  (
  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_ENABLER_CONFIG_INFO$

This table stores partner application configuration information.

```
create table wwsec_enabler_config_info$ OF sec_enabler_config_type
  (
  lsnr_token               constraint wwsec_seci_pk primary key
  , site_token             constraint wwsec_seci_uk1 UNIQUE
  , site_id                constraint wwsec_seci_uk2 UNIQUE
  , ls_login_url          NOT NULL
  , urlcookie_version      NOT NULL
  , encryption_key         NOT NULL
  , encryption_mask_pre    NOT NULL
  , encryption_mask_post   NOT NULL
  , CHECK (url_cookie_ip_check IN ('Y','N'))
  );
```

### WWSEC_SSO_LOG

This table stores debug information when debug is enabled.

```
CREATE TABLE wwsec_sso_log$
(
    id  NUMBER
  , msg  VARCHAR2(1000)
  , log_date DATE
 );
```

# 4

# Java Oracle Single Sign-On Application Programming Interface

The Java package, `oracle.security.sso`, contains information about how application developers can use Java classes and methods to enable web users to access partner applications by means of Oracle Single Sign-On. This chapter should be used as a reference and assumes that the reader is familiar with PL/SQL functions and procedures for using Oracle Single Sign-On.

This chapter contains these topics:



**See Also:** Chapter 3, "PL/SQL Single Sign-On Application Programming Interface"

# Package

**oracle.security.sso.enabler Description**

***Table 4–1 oracle.security.sso.enabler***

| Class Summary | |
|---|---|
| Classes | |
| SSOEnabler | This class implements the enabler stack of the Oracle Single Sign-On service for partner application development. |
| SSOEnablerConfig | This class is used with `SSOEnabler` class for configuration parameters setup. |
| SSOEnablerConfigMgr | This class implements the enabler stack of the Oracle Single Sign-On service for partner application development. |
| SSOEnablerUtil | |
| SSOUserInfo | This class is used for returning user information after parsing the redirect URL from `SSOEnabler` class. |
| Exceptions | |
| SSOEnablerException | This class is subclass of `java.lang.Exception`. |

# oracle.security.sso.enabler

## SSOEnabler

### Syntax

```
public class SSOEnabler extends java.lang.Object

java.lang.Object
  |
  +--oracle.security.sso.enabler.SSOEnabler
```

### Description

This class implements the enabler stack of the Oracle Single Sign-On service for partner application development.

### Since:

1.0

### See Also:

SSOUserInfo

***Table 4–2 SSOEnabler Member Summary***

| Member Summary | |
|---|---|
| Constructors | |
| SSOEnabler | Creates an Oracle Single Sign-On enabler object, with no database connection. |
| SSOEnabler(Connection) | Creates an Oracle Single Sign-On enabler object, with database connection. |
| Methods | |
| generateRedirect(String, String, String) | Generates a redirect URL from requested URL and cancel URL. |
| getSSOUserInfo(String, String, InetAddress) | Parses a redirect URL from Oracle Single Sign-On server which contains user information. |
| setDbConnection(Connection) | Initializes Oracle Single Sign-On enabler object, with a database connection. |

***Table 4–3 SSOEnabler Inherited Member Summary***

| Inherited Member Summary |
|---|
| Methods inherited from `class java.lang.Object` |
| `equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait` |

# Constructors

## SSOEnabler()

```
public SSOEnabler()
```

Creates an Oracle Single Sign-On enabler object, with no database connection.

## SSOEnabler(Connection)

```
public SSOEnabler(java.sql.Connection p_db_conn)
```

Creates an Oracle Single Sign-On enabler object, with a database connection.

**See Also:**

`java.sql.Connection`

# Methods

## generateRedirect(String, String, String)

```
public java.lang.String generateRedirect(java.lang.String p_listenerToken,
java.lang.String p_requestedUrl, java.lang.String p_onCancelUrl)
```

Generates a redirect URL from the requested URL and the cancel URL. When a partner application wants to authenticate a user, it redirects the user (browser) to the Oracle Single Sign-On server with this URL. The requested URL string is the URL which the user wants to access. The cancel URL string is the URL to which the Oracle Single Sign-On server redirects the user when the user does not wish to authenticate at that moment.

**Returns:**

Redirect url

**Throws:**

SSOEnablerException—whenthere is an error in constructing redirect url

## getSSOUserInfo(String, String, InetAddress)

```
public SSOUserInfo getSSOUserInfo(java.lang.String p_listenerToken,
java.lang.String p_cookieStr, java.net.InetAddress p_clientIp)
```

Parses a redirect URL from the Oracle Single Sign-On server which contains user information.

**Returns:**

SSOUserInfo object which contains user information

**Throws:**

SSOEnablerException—whenthere is an error in parsing

**See Also:**

SSOUserInfo

## setDbConnection(Connection)

```
public void setDbConnection(java.sql.Connection p_db_conn)
```

Initializes the Oracle Single Sign-On enabler object with a database connection.

**Throws:**

SSOEnablerException—when the database connection is lost

**See Also:**

java.sql.Connection

# oracle.security.sso.enabler

## SSOEnablerConfig

### Syntax

```
public class SSOEnablerConfig extends java.lang.Object

java.lang.Object
  |
  +--oracle.security.sso.enabler.SSOEnablerConfig
```

### Description

This class is used with SSOEnabler class for configuration parameters setup.

### Since:

1.0

### See Also:

SSOEnabler

***Table 4–4 SSOEnablerConfig Member Summary***

| Member Summary | |
|---|---|
| Constructors | |
| SSOEnablerConfig() | Sets none of the properties. |
| SSOEnablerConfig(String, String, String, String, String, String, String, String, String) | Sets all of the properties. |
| Methods | |
| getEncryptionKey() | Returns the encryption key. |
| getEncryptionMaskPost() | Returns the encryption mask post. |
| getEncryptionMaskPre() | Returns the encryption mask pre. |
| getListnerToken() | Returns the listener token. |
| getLoginUrl() | Returns the login URL. |

| Member Summary | |
|---|---|
| getSiteID() | Returns the site identifier. |
| getSiteToken() | Returns the site token. |
| getUrlCookieIPCheck() | Returns the URL cookie IP check. |
| getUrlCookieVersion() | Returns the URL cookie version. |
| setEncryptionKey(String) | Sets the encryption key. |
| setEncryptionMaskPost(String) | Sets the encryption mask post. |
| setEncryptionMaskPre(String) | Sets the encryption mask pre. |
| setListnerToken(String) | Sets the listener token. |
| setLoginUrl(String) | Sets the login URL. |
| setSiteID(String) | Sets the site identifier. |
| setSiteToken(String) | Sets the site token. |
| setUrlCookieIPCheck(String) | Sets the URL cookie IP check. |
| setUrlCookieVersion(String) | Sets the URL cookie version. |

***Table 4–5 SSOEnablerConfig Inherited Member Summary***

| Inherited Member Summary |
|---|
| Methods inherited from class java.lang.Object |
| equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait |

# Constructors

## SSOEnablerConfig()

```
public SSOEnablerConfig()
```

Sets no properties.

### SSOEnablerConfig(String, String, String, String, String, String, String, String, String)

```
public  SSOEnablerConfig(java.lang.String p_listenerToken, java.lang.String p_
siteToken, java.lang.String p_siteID, java.lang.String p_loginURL,
java.lang.String p_urlCookieVersion, java.lang.String p_encryptionKey,
java.lang.String p_encryptionMaskPre, java.lang.String p_encryptionMaskPost,
java.lang.String p_urlCookieIPCheck)
```

Sets all properties.

## Methods

### getEncryptionKey()

```
public java.lang.String getEncryptionKey()
```

Returns the encryption key.

**Returns:**

The encryption key.

### getEncryptionMaskPost()

```
public java.lang.String getEncryptionMaskPost()
```

Returns the encryption mask post.

**Returns:**

The encryption mask post.

### getEncryptionMaskPre()

```
public java.lang.String getEncryptionMaskPre()
```

Returns the encryption mask pre.

**Returns:**

The encryption mask pre.

## getListnerToken()

```
public java.lang.String getListnerToken()
```

Returns the listener token.

**Returns:**

The listener token.

## getLoginUrl()

```
public java.lang.String getLoginUrl()
```

Returns the login URL.

**Returns:**

The login URL.

## getSiteID()

```
public java.lang.String getSiteID()
```

Returns the site identifier.

**Returns:**

The site identifier.

## getSiteToken()

```
public java.lang.String getSiteToken()
```

Returns the site token.

**Returns:**

The site token.

## getUrlCookieIPCheck()

```
public java.lang.String getUrlCookieIPCheck()
```

Returns the URL cookie IP check.

**Returns:**

The URL cookie IP check.

## getUrlCookieVersion()

```
public java.lang.String getUrlCookieVersion()
```

Returns the URL cookie version.

**Returns:**

The URL cookie version.

## setEncryptionKey(String)

```
public void setEncryptionKey(java.lang.String p_encryptionKey)
```

Sets the encryption key.

**Parameter:**

`encryptionKey`—The encryption key.

## setEncryptionMaskPost(String)

```
public void setEncryptionMaskPost(java.lang.String p_encryptionMaskPost)
```

Sets the encryption mask post.

**Parameter:**

`encryptionMaskPost`—The encryption mask post.

## setEncryptionMaskPre(String)

```
public void setEncryptionMaskPre(java.lang.String p_encryptionMaskPre)
```

Sets the encryption mask pre.

**Parameter:**

`encryptionMaskPre`—The encryption mask pre.

## setListnerToken(String)

```
public void setListnerToken(java.lang.String p_listnerToken)
```

Sets the listener token.

**Parameter:**

`listnerToken`—The listener token.

## setLoginUrl(String)

```
public void setLoginUrl(java.lang.String p_loginURL)
```

Sets the login URL.

**Parameter:**

`loginURL`—The login URL

## setSiteID(String)

```
public void setSiteID(java.lang.String p_siteID)
```

Sets the site identifier.

**Parameter:**

`siteID`—The site identifier

## setSiteToken(String)

```
public void setSiteToken(java.lang.String p_siteToken)
```

Sets the site token.

**Parameters**

`siteToken`—The site token

## setUrlCookieIPCheck(String)

```
public void setUrlCookieIPCheck(java.lang.String p_urlCookieIPCheck)
```

Sets the URL cookie IP check.

**Parameter:**

`urlCookieIPCheck`—The URL cookie IP check

## setUrlCookieVersion(String)

```
public void setUrlCookieVersion(java.lang.String p_urlCookieVersion)
```

Sets the URL cookie version.

**Parameter:**

`urlCookieVersion`—The URL cookie version

# oracle.security.sso.enabler

# SSOEnablerConfigMgr

**Syntax**

```
public class SSOEnablerConfigMgr extends java.lang.Object

java.lang.Object
  |
  +--oracle.security.sso.enabler.SSOEnablerConfigMgr
```

**Description**

Implements the enabler stack of the Oracle Single Sign-On service for partner application development.

**Since:**

1.0

**See Also:**

SSOEnablerConfig

***Table 4–6 SSOEnablerConfigMgr Member Summary***

| Member Summary | |
|---|---|
| Constructors | |
| SSOEnablerConfigMgr() | Creates an Oracle Single Sign-On enabler object, with no database connection. |
| SSOEnablerConfigMgr(Connection) | Creates an Oracle Single Sign-On enabler object, with database connection. |
| Methods | |
| createEnablerConfig(SSOEnablerConfig) | Creates configuration parameters of the SSO enabler specified by the listener token. |
| deleteEnablerConfig(String) | Deletes the configuration parameters of the SSO enabler specified by the listener token. |

| Member Summary | |
|---|---|
| getEnablerConfig(String) | Returns the configuration parameters of the SSO enabler specified by the listener token. |
| setDbConnection(Connection) | Initializes Oracle Single Sign-On enabler object, with a database connection. |
| setEnablerConfig(String, SSOEnablerConfig) | Updates the configuration parameters of the SSO enabler specified by the listener token. |

***Table 4–7 SSOEnablerConfigMgr Inheirited Member Summary***

| Inherited Member Summary |
|---|
| Methods inherited from `class java.lang.Object` |
| `equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait` |

# Constructors

## SSOEnablerConfigMgr()

```
public SSOEnablerConfigMgr()
```

Creates an Oracle Single Sign-On enabler object, with no database connection.

## SSOEnablerConfigMgr(Connection)

```
public SSOEnablerConfigMgr(java.sql.Connection p_db_conn)
```

Creates an Oracle Single Sign-On enabler object, with database connection.

**See Also:**

`java.sql.Connection`

# Methods

## createEnablerConfig(SSOEnablerConfig)

```
public void createEnablerConfig(SSOEnablerConfig p_configuration)
```

Creates configuration parameters of the SSO enabler specified by the listener token.

**Parameter:**

`p_configuration`—The configuration for the SSO enabler to be added. All the members of this class must be filled in except for `encryptionMaskPre` and `encryptionMaskPost`, which must be empty strings ("").

**Throws:**

`SSOEnablerException`—Raised when the database connection is lost, the database is not configured properly, or invalid data is passed into this procedure.

**See Also:**

`SSOEnablerConfig`

## deleteEnablerConfig(String)

```
public void deleteEnablerConfig(java.lang.String p_listenerToken)
```

Deletes the configuration parameters of the SSO enabler specified by the listener token.

**Parameter:**

`p_listenerToken`—The listener token of the SSO enabler `p_configuration` that is to be deleted

**Throws:**

`SSOEnablerException`—Raised when the database connection is lost, the database is not configured properly, or invalid data is passed into this procedure.

## getEnablerConfig(String)

```
public SSOEnablerConfig getEnablerConfig(java.lang.String p_listenerToken)
```

Returns the configuration parameters of the SSO enabler specified by the listener token.

**Parameter:**

`p_listenerToken`—The listener token of the SSO enabler `p_configuration` that is to be selected

**Returns:**

An instance of SSOEnablerConfig containing the `p_configuration` of the SSO enabler specified by the listener token

**Throws:**

`SSOEnablerException`—Raised when the database connection is lost, the database is not configured properly, or the listener token is invalid.

**See Also:**

`SSOEnablerConfig`

## setDbConnection(Connection)

```
public void setDbConnection(java.sql.Connection p_db_conn)
```

Initializes the Oracle Single Sign-On enabler object, with a database connection.

**Throws:**

`SSOEnablerException`—whenthe database connection lost

**See Also:**

`java.sql.Connection`

## setEnablerConfig(String, SSOEnablerConfig)

```
public void setEnablerConfig(java.lang.String p_listenerToken,
SSOEnablerConfig p_configuration)
```

Updates the configuration parameters of the SSO enabler specified by the listener token.

**Parameter:**

p_listenerToken—The listener token of the SSO enabler p_configuration that is to be updated

p_configuration—The configuration for the SSO enabler to be updated. (All the members of this class must be filled in.)

**Throws:**

SSOEnablerException—Raised when the database connection is lost, the database is not configured properly, or invalid data is passed into this procedure.

**See Also:**

SSOEnablerConfig

# oracle.security.sso.enabler

## SSOEnablerException

**Syntax**

```
public class SSOEnablerException extends java.lang.Exception

java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--oracle.security.sso.enabler.SSOEnablerException
```

**All Implemented Interfaces:**

java.io.Serializable

**Description**

This class is a subclass of java.lang.Exception

**Since:**

1.0

*Table 4–8 SSOEnablerException Member Summary*

| Member Summary | |
|---|---|
| Constructors | |
| SSOEnablerException() | Constructs an SSOEnablerException object with no specified detail message. |
| SSOEnablerException(String) | Constructs an SSOEnablerException object with specified detail message. |

*Table 4–9 SSOEnablerException Inheirited Member Summary*

| Inherited Member Summary |
|---|
| Methods inherited from class java.lang.Throwable |
| fillInStackTrace, getLocalizedMessage, getMessage, printStackTrace, printStackTrace, printStackTrace, toString |
| Methods inherited from class java.lang.Object |
| equals, getClass, hashCode, notify, notifyAll, wait, wait, wait |

# Constructors

## SSOEnablerException()

```
public  SSOEnablerException()
```

Constructs an SSOEnablerException object with no specified detail message.

## SSOEnablerException(String)

```
public  SSOEnablerException(java.lang.String p_str)
```

Constructs an SSOEnablerException object with specified detail message.

# oracle.security.sso.enabler

## SSOEnablerUtil

### Syntax

```
public class SSOEnablerUtil extends java.lang.Object

java.lang.Object
  |
  +--oracle.security.sso.enabler.SSOEnablerUtil
```

### Description

*Table 4–10 SSOEnablerUtil Member Summary*

| Member Summary | |
|---|---|
| Constructors | |
| SSOEnablerUtil() | Creates a utility object for application cookie baking / unbaking, with no database connection. |
| SSOEnablerUtil(Connection) | Creates a utility object for application cookie baking / unbaking, with database connection. |
| Methods | |
| bakeAppCookie(String, String) | Bakes the input application cookie for encryption and hashing The return string is encrypted along with hashed application cookie. |
| genHtmlPostForm(String) | Generates a HTML post form to the login server URL from generate redirect url. |
| genRedirect(String) | Generate a HTML redirect to the specified url. |
| setDbConnection(Connection) | Initializes utility object for application cookie baking / unbaking, with a database connection. |
| unbakeAppCookie(String, String) | Unbakes the input baked application cookie. The return string is the decrypted application cookie. |

*Table 4–11 SSOEnablerUtil Inheirited Member Summary*

| Inherited Member Summary |
|---|
| Methods inherited from class java.lang.Object |
| equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait |

# Constructors

## SSOEnablerUtil()

```
public SSOEnablerUtil()
```

Creates a utility object for application cookie baking / unbaking, with no database connection.

## SSOEnablerUtil(Connection)

```
public SSOEnablerUtil(java.sql.Connection p_db_conn)
```

Creates a utility object for application cookie baking / unbaking, with a database connection.

**See Also:**

`java.sql.Connection`

# Methods

## bakeAppCookie(String, String)

```
public java.lang.String bakeAppCookie(java.lang.String p_listenerToken,
java.lang.String p_appCookie)
```

Bakes the input application cookie for encryption and hashing. The return string is encrypted along with the hashed application cookie.

**Parameters:**

`p_listenerToken`—Listener token for the specific login server

**`p_appCookie`**—Application cookie

**Returns:**

Baked application cookie

**Throws:**

`SSOEnablerException`—whenthe database connection is lost or any other error occurs

## genHtmlPostForm(String)

```
public static java.lang.String genHtmlPostForm(java.lang.String p_
genRedirectUrl)
```

Generates a HTML post form to the login server URL from the generate redirect url.

**Parameter:**

`p_genRedirectUrl`—generate redirect URL

**Returns:**

html redirect url

**Throws:**

`IllegalArgumentException`—whenthe input URL is incorrect

## genRedirect(String)

```
public static java.lang.String genRedirect(java.lang.String p_redirectUrl)
```

Generates a HTML redirect to the specified url.

**Parameter:**

`p_redirectUrl`—generate redirect url

**Returns:**

html post form for login server

**Throws:**

`IllegalArgumentException`—whenthe input URL is incorrect

## setDbConnection(Connection)

```
public void setDbConnection(java.sql.Connection p_db_conn)
```

Initializes utility object for application cookie baking / unbaking, with a database connection.

**Throws:**

`SSOEnablerException`—whenthe database connection is lost

**See Also:**

`java.sql.Connection`

## unbakeAppCookie(String, String)

```
public java.lang.String unbakeAppCookie(java.lang.String p_listenerToken,
java.lang.String p_bakedAppCookie)
```

Unbakes the input baked application cookie. The return string is the decrypted application cookie.

**Parameter:**

`p_listenerToken`—Listener token for the specific login server

`p_bakedAppCookie`—Unbaked application cookie

**Returns:**

Unbaked application cookie

**Throws:**

`SSOEnablerException`—whenthe database connection is lost or any other error occurs

# oracle.security.sso.enabler

## SSOUserInfo

**Syntax**

```
public class SSOUserInfo extends java.lang.Object

java.lang.Object
  |
  +--oracle.security.sso.enabler.SSOUserInfo
```

**Description**

Returns user information after parsing redirect URL from `SSOEnabler` class.

**Since:**

1.0

**See Also:**

`SSOEnabler`

***Table 4–12 SSOUserInfo Member Summary***

| Member Summary | |
|---|---|
| Methods | |
| `getIPAddress()` | Returns IP Address. |
| `getSiteTimeStamp()` | Returns the site time stamp. |
| `getSSOTimeRemaining()` | Returns remaining Single Sign-On time in hours. |
| `getSSOUserName()` | Returns Single Sign-On user name. |
| `getUrlRequested()` | Returns URL requested by the user. |

***Table 4–13 SSOUserInfo InheiritedMember Summary***

| Inherited Member Summary |
|---|
| Methods inherited from class java.lang.Object |

**Inherited Member Summary**

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

# Methods

## getIPAddress()

```
public java.lang.String getIPAddress()
```

Returns the IP Address.

## getSiteTimeStamp()

```
public java.sql.Date getSiteTimeStamp()
```

Returns the site time stamp.

## getSSOTimeRemaining()

```
public int getSSOTimeRemaining()
```

Returns remaining Single Sign-On time in hours.

## getSSOUserName()

```
public java.lang.String getSSOUserName()
```

Returns Single Sign-On user name.

## getUrlRequested()

```
public java.lang.String getUrlRequested()
```

Returns URL requested by the user.

# 5

# Examples in PL/SQL and Java

This chapter provides some sample programs and examples of code to illustrate for developers how to implement partner applications.

This chapter contains the following topics:

- Writing Partner Application using PL/SQL SSO APIs
- Writing Partner Application Using Java SSO APIs

> **Note:** For additional information about development related issues, refer to the Readme file included in the Software Development Kit (SDK).

# Writing Partner Application using PL/SQL SSO APIs

Writing a partner application using PL/SQL requires Oracle Web Agent packages for web related functionality and requires that two procedures be implemented. In the following code example, these two public procedures perform all redirection and parsing functionality. The public procedures are as follows:

- SAMPLE_SSO_PAPP.SSOAPP
- SAMPLE_SSO_PAPP.SIGN_ON

### SAMPLE_SSO_PAPP.SSOAPP

This procedure constructs the application URL and it requires authentication to access it. This procedure checks to see if the application cookie exists and user information can be retrieved. Otherwise it redirects the user to the SSO server by generating redirect url.

### SAMPLE_SSO_PAPP.SIGN_ON

This procedure gets the URLC token from the SSO 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 ( i.e. SSOAPP URL).

```
CREATE OR REPLACE PACKAGE sample_sso_papp
IS
    PROCEDURE ssoapp;

    PROCEDURE sign_on
    (
        urlc IN VARCHAR2
    )
END sample_sso_papp;
/
show errors package sample_sso_papp

CREATE OR REPLACE PACKAGE BODY sample_sso_papp
IS
    l_listener_token VARCHAR2(1000) := 'app1:ssosvr:443';
    l_requested_url  VARCHAR2(1000) :=
       'http://www.ssopapp.com/pls/ssodad/schamaname/sample_sso_papp.sso_papp';
    l_cancel_url     VARCHAR2(1000) := 'http://www.ssopapp.com';
    l_cookie_name    VARCHAR2(1000) := 'SSO_SQLPAPP_ID';
    l_cookie_path    VARCHAR2(1000) := '/';
    l_cookie_domain  VARCHAR2(1000) := 'www.ssopapp.com';
```

```
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(l_cookie_name);
    l_user_info  := l_app_cookie.vals(1);
    return l_user_info;
END get_user_info;

PROCEDURE ssoapp
IS
    l_user_info        VARCHAR2(1000);
    l_gen_redirect_url VARCHAR2(32000);
BEGIN
    l_user_info := get_user_info;
    IF l_user_info IS NULL THEN
        l_gen_redirect_url :=
        wwsec_sso_enabler_private.generate_redirect
        (
            p_lsnr_token => l_listener_token,
            urlrequested => l_requested_url,
            urloncancel  => l_cancel_url
        );
        owa_util.redirect_url(l_gen_redirect_url);
    ELSE
        begin
            htp.init;
        exception
            when others then null;
        end;

        htp.htmlOpen;
        htp.headOpen;
        htp.print('<meta http-equiv="Content-Type"'
            || ' content="text/html; charset=iso-8859-1">');
        htp.title('PL/SQL based SSO Partner Application');
        htp.headCLose;
        htp.bodyOpen;
        htp.print('User Information:' || l_user_info || '<br>');
        htp.bodyClose;
```

```
        htp.htmlClose;
    END IF;
EXCEPTION
    WHEN OTHERS THEN
       RAISE;
END ssoapp;

PROCEDURE sign_on
(
    urlc IN VARCHAR2
)
IS
    l_urlc                VARCHAR2(32000);
    l_sso_user_name       VARCHAR2(1000);
    l_ip_address          VARCHAR2(1000);
    l_sso_time_remaining  VARCHAR2(1000);
    l_site_time_stamp     VARCHAR2(1000);
    l_url_requested       VARCHAR2(1000);
    l_unused_param        VARCHAR2(1000);
BEGIN
    -- Process URLC token
    wwsec_sso_enabler_private.parse_url_cookie
    (
        p_lsnr_token => l_listener_token,
        encrypted_urlcookie => urlc,
        ssousername => l_sso_user_name,
        ipadd => l_ip_address,
        ssotimeremaining => l_sso_time_remaining,
        sitetimestamp => l_site_time_stamp,
        urlrequested => l_url_requested,
        newsitekey => l_unused_param
    );
    -- Set application cookie
    begin
        htp.init;
    exception
        when others then null;
    end;

    owa_util.mime_header('text/html', FALSE);
    owa_cookie.send
    (
        name => l_cookie_name,
        value => l_sso_user_name,
        expires => null,
```

```
        path => l_cookie_path,
        domain => l_cookie_domain
    );
    -- Redirect user to the requested application url
    htp.htmlOpen;
    htp.headOpen;
    htp.meta(chttp_equiv=>'Refresh',
        cname=> 'Requested URL',
        ccontent=>'0 URL=' || l_url_requested);
    htp.headClose;
    htp.htmlClose;
EXCEPTION
  WHEN OTHERS THEN
    RAISE;
END sign_on;

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

# Writing Partner Application Using Java SSO APIs

Initially, the partner application redirects the user to the Login 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, then the partner application redirects the user to the Login Server. To avoid contacting Login Server for authentication verification of every user request, all partner applications should maintain their own application session.

This section contains the following topics

- Implementing the Partner Application in Java
- Servlet Based Partner Application
- JSP based partner application

## Implementing the Partner Application in Java

To implement the partner application in Java, we will implement a generic bean which will be used in Servlet as well as JSP based applications.

```
// SSOEnablerBean.java
```

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

import java.net.URL;
import java.net.InetAddress;

import java.sql.DriverManager;
import java.sql.Connection;

import oracle.security.sso.enabler.SSOEnabler;
import oracle.security.sso.enabler.SSOUserInfo;
import oracle.security.sso.enabler.SSOEnablerUtil;
import oracle.security.sso.enabler.SSOEnablerException;

public class SSOEnablerBean
{
    private  String m_listenerToken    = null;
    private  String m_requestedUrl     = null;
    private  String m_onCancelUrl      = null;

    private  String m_pappCookieName   = null;
    private  String m_pappCookieDomain = null;
    private  String m_pappCookieScope  = null;
    private  String m_pappCookieDesc   = null;

    private  String m_dbCred           = null;
    private  String m_dbhost           = null;
    private  String m_dbport           = null;
    private  String m_dbsid            = null;

   /**
    *  Default constructor
    */
    public SSOEnablerBean()
    {
    }

   /**
    *  Set listener token
    */
    public void setListenerToken(String p_listenerToken)
    {
        m_listenerToken = p_listenerToken;
    }
```

```
/**
 *  Set requested and cancel url
 */
 public void setUrls(String p_requestedUrl, String p_cancelUrl)
 {
     m_requestedUrl = p_requestedUrl;
     m_onCancelUrl  = p_cancelUrl;
 }

 /**
  * Set application cookie information
  */
 public void setAppCookieInfo(String p_name, String p_domain, String p_path)
 {
     m_pappCookieName        = p_name;
     m_pappCookieDomain      = p_domain;
     m_pappCookieScope       = p_path;
     m_pappCookieDesc        = "SSO application cookie";
 }

 public void setDbConnectionInfo(String p_schema , String p_password,
     String p_hostname, String p_port, String p_sid)
 {
     m_dbCred   = p_schema + "/"+ p_password;
     m_dbhost   = p_hostname;
     m_dbport   = p_port;
     m_dbsid    = p_sid;
 }

/**
 *  This method will return SSO user information. If the user is not
 *  authenticated against  SSO server then it will redirect user to the SSO
 *  Server for authentication
 */
 public String getSSOUserInfo(HttpServletRequest p_request,
      HttpServletResponse p_response)
     throws SSOEnablerException
 {
    String l_userName = null;

    if(p_response == null || p_response == null)
    {
         throw new SSOEnablerException("Http objects are null");
    }
```

```
if(m_listenerToken == null)
{
     throw new SSOEnablerException("Listener token is null");
}

if(m_requestedUrl == null || m_onCancelUrl == null)
{
     throw new SSOEnablerException(
          "Requested URL and cancel URL must be set");
}

try
{
     // Get database connection
     Connection l_db_con = getDbConnection();

     // Try to get user information from application cookie
     l_userName = getUserInfo(p_request);

     if(l_userName == null)
     {
         // Create SSOEnabler object
         SSOEnabler l_ssoEnabler = new SSOEnabler(l_db_con);
         // Create redirect URL to the SSO server for user authentication
         String l_redirectUrl =
             l_ssoEnabler.generateRedirect(m_listenerToken,
                         m_requestedUrl, m_onCancelUrl);
         // close database connection
         closeDbConnection(l_db_con);

         // p_response.sendRedirect(l_redirectUrl);

         // Since the redirect URL is usually large so send the
         // redirect URL input parameters using HTTP post method instead
         // of usual GET method of
         // HttpServletResponse.sendRedirect
         String htmlPostForm
             = SSOEnablerUtil.genHtmlPostForm(l_redirectUrl);
         p_response.getWriter().println(htmlPostForm);

         return null;
     }
     else
     {
```

```
            // We got this user information from application cookie
            SSOEnablerUtil l_ssoAppUtil = new SSOEnablerUtil(l_db_con);
            return l_ssoAppUtil.unbakeAppCookie(m_listenerToken,
                                                    l_userName);
         }
        }
    }
    catch(Exception e)
    {
       throw new SSOEnablerException(e.toString());
    }
 }

/**
 *  Get  user information from application cookie
 */
 private String getUserInfo(HttpServletRequest p_request)
     throws SSOEnablerException
 {
     boolean l_gotPappCookie = false;
     String  l_userInfo      = null;

     if(m_pappCookieName == null)
         throw new SSOEnablerException("Cookie name is null");
     try
     {
         Cookie[] l_cookies = p_request.getCookies();
         for(int i=0; i < l_cookies.length; i++)
         {
             Cookie l_pappCookie = l_cookies[i];
             if (l_pappCookie.getName().equals(m_pappCookieName))
             {
                 l_gotPappCookie = true;
                 l_userInfo      = l_pappCookie.getValue();
                 break;
             }
         }
     }
     catch(Exception e)
     {
        return null;
     }

     if( (l_userInfo != null) && (l_userInfo.length() > 0) )
     {
```

```
            return l_userInfo;
        }
        else
        {
            return null;
        }
    }

   /**
    *  This method will set application cookie from SSO server token and
    * then redirect user to the application url
    */
    public void setPartnerAppCookie(HttpServletRequest p_request,
            HttpServletResponse p_response)
        throws SSOEnablerException
    {
       if(p_response == null || p_response == null)
       {
            throw new SSOEnablerException("Http objects are null");
       }

       if(m_listenerToken == null)
       {
            throw new SSOEnablerException("Listener token is null");
       }

       if( m_pappCookieName  == null
            || m_pappCookieDomain == null
            || m_pappCookieScope  == null
            || m_pappCookieDesc   == null)
       {
            throw new SSOEnablerException(
                  "Application cookie information is not available");
       }

       SSOUserInfo l_ssoUserInfo = null;
       try
       {
          String l_urlParam = p_request.getParameterValues("urlc")[0];
          if(l_urlParam != null)
          {
              // Get database connection
             Connection l_db_con = getDbConnection();

             // Create SSOEnabler object
```

```
            SSOEnabler l_ssoEnabler  = new SSOEnabler(l_db_con);

            // Get IP address of the client
            InetAddress l_clientIp
                       = InetAddress.getByName(p_request.getRemoteAddr());
            l_ssoUserInfo = l_ssoEnabler.getSSOUserInfo(m_listenerToken,
                                                 l_urlParam, l_clientIp);

            // Set application cookie
            SSOEnablerUtil l_ssoAppUtil = new SSOEnablerUtil(l_db_con);
            String l_bakedAppCookie =
                l_ssoAppUtil.bakeAppCookie(m_listenerToken,
                                    l_ssoUserInfo.getSSOUserName());
            // Close database connection
            closeDbConnection(l_db_con);

            // Create application cookie and set it
            // ** IMPORTANT **
            // Time stamp **must** be added in this cookie and should implement
            // application cookie time out based on user inactivity etc.
            Cookie l_appCookie = new Cookie(m_pappCookieName,
                l_bakedAppCookie);
            l_appCookie.setDomain(m_pappCookieDomain);
            // In-memory cookie for better security
            l_appCookie.setMaxAge(-1);
            l_appCookie.setPath(m_pappCookieScope);
            l_appCookie.setComment(m_pappCookieDesc);
            p_response.addCookie(l_appCookie);

            String reqRedirHtmlStr
                 = SSOEnablerUtil.genRedirect(l_ssoUserInfo.getUrlRequested());
            p_response.getWriter().println(reqRedirHtmlStr);
          }
          else
          {
              throw new SSOEnablerException(
                 "SSO server returned null user information");
          }
        }
        catch(Exception e)
        {
            throw new SSOEnablerException(e.toString());
        }
    }
```

```
/**
 * Remove application cookie to end user application session
 */
 public void removePartnerAppCookie(HttpServletResponse p_response)
     throws SSOEnablerException
 {
     if(p_response == null)
     {
         throw new SSOEnablerException("HttpServletResponse is null");
     }

     if( m_pappCookieName  == null
         || m_pappCookieDomain == null
         || m_pappCookieScope  == null

     {
         throw new SSOEnablerException(
               "Application cookie information is not available");
     }

     Cookie l_AppCookie = new Cookie(m_pappCookieName,
              "End application sesion");
     l_AppCookie.setDomain(m_pappCookieDomain);
     l_AppCookie.setMaxAge(0);
     l_AppCookie.setPath(m_pappCookieScope);
     l_AppCookie.setComment(m_pappCookieDesc);
     p_response.addCookie(l_AppCookie);
 }

/**
 * Get database connection to the schema where partner application
 * packages are installed and configured
 *
 * ** IMPORTANT **
 *   This method **must** be implemented as pooled database connection for
 *   better performance
 */
 private Connection getDbConnection()
     throws SSOEnablerException
 {
     // Database connection string
     String l_connect_string =
     "jdbc:oracle:thin:" + m_dbCred + "@"
          + "(description="
          + "(address_list="
```

```
                + "(address=(protocol=tcp)(host=" + m_dbhost + ")(port="
                + m_dbport + ")))" + "(source_route=yes)(connect_data=(sid="
                + m_dbsid + ")))";
            try
            {

                // Get database connection
                Class.forName ("oracle.jdbc.driver.OracleDriver");
                Connection l_db_con
                        = DriverManager.getConnection(l_connect_string);
                return l_db_con;
            }
            catch(Exception e)
            {
                throw new SSOEnablerException(e.toString());
            }
        }

       /**
        * Release database connection to the schema where partner application
        * packages are  installed and configured
        *
        * ** IMPORTANT **
        *  This method **must** be implemented as pooled release database
        *  connection for better performance
        */
        private void closeDbConnection(Connection p_dbConn)
            throws SSOEnablerException
        {
            try
            {
                p_dbConn.close();
            }
            catch(Exception e)
            {
                throw new SSOEnablerException(e.toString());
            }
        }

    }
```

## Servlet Based Partner Application

A sample servlet based partner application could 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 can be found, then 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:
   a. It redirects the user to the `SSOSignOnServlet` URL.
   b. It sets the application cookie.
   c. It redirects the user to the requested application URL using `SSOEnablerServletBean`.

### SSOEnablerServletBean

This bean is derived from the `SSOEnablerBean` and implements the necessary methods for servlet based application.

### SSOPartnerServlet

This servlet is the main partner application servlet. To access this servlet, the user must authenticate to the SSO server. This servlet redirects the unauthenticated user to the SSO server.

### SSOSignOnServlet

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

### SSOPartnerLogoutServlet

This servlet removes the application session of the partner application

**SSOEnablerServletBean.java**

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

import oracle.security.sso.enabler.SSOEnablerException;
```

```
public class SSOEnablerServletBean
{
    private SSOEnablerBean  m_enablerBean = null;
   /**
    *  Default constructor
    */
    public SSOEnablerServletBean()
    {
        m_enablerBean = new SSOEnablerBean();

        m_enablerBean.setListenerToken("servletapp2:loginserver:443");
        m_enablerBean.setUrls(
            "http://app2.xyz.com:80/servlet/SSOPartnerServlet",
            "http://app2.xyz.com:80");
        m_enablerBean.setAppCookieInfo("SSO_APP2_SERVLET_ID",
             "app2.xyz.com", "/");
        m_enablerBean.setDbConnectionInfo("papp", "papp",
             "db-hostname", "1521", "db-sid");
    }

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

    public void setPartnerServletAppCookie(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.removePartnerAppCookie(p_response);
    }
}
```

**SSOPartnerServlet.java**

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

```
import javax.servlet.ServletException;

import java.io.PrintWriter;

public class SSOPartnerServlet extends HttpServlet
{
   /**
    * 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();
           SSOEnablerServletBean l_ssobean = new SSOEnablerServletBean();
           String l_userInfo   = l_ssobean.getSSOUserInfo(p_request,
                                          p_response);

           if(l_userInfo != null)
           {
               // 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>");
               l_out.println("<P><center>"
                      +"<A HREF='/servlet/SSOPartnerLogoutServlet'>"
                      +"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());
             }
        }
    }
}
```

**SSOSignOnServlet.java**

```
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpServlet;
import javax.servlet.ServletException;
import java.io.PrintWriter;

public class SSOSignOnServlet extends HttpServlet
{
   /**
    * 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
        {
            SSOEnablerServletBean l_ssobean = new SSOEnablerServletBean();
            l_ssobean.setPartnerServletAppCookie(p_request, p_response);
        }
        catch(Exception e)
        {
             try
             {
                 p_response.getWriter().println("Error " + e.toString());
             }
             catch(Exception e1)
             {
                 throw new ServletException(e1.toString());
             }
        }
    }
}
```

**SSOPartnerLogoutServlet.java**

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

import java.io.PrintWriter;

public class SSOPartnerLogoutServlet extends HttpServlet
{
   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
        {
            SSOEnablerServletBean l_ssobean = new SSOEnablerServletBean();
            l_ssobean.removeServletAppCookie(p_response);
```

```
            PrintWriter l_out = p_response.getWriter();
            l_out.println("<HTML><HEAD><TITLE>"
                 + "Servlet based SSO Partner Application</TITLE></HEAD><BODY>");
            l_out.println("<center><H3>Servlet based SSO Partner"
                 + " Application</H3><center>");
            l_out.println("<P><center>You are logged off from application"
                 + " session<center><BR>");
            l_out.println("<P><center>"
                 +"<A HREF='/servlet/SSOPartnerServlet'>Login</A><center></P>");
            l_out.println("</BODY></HTML>");
        }
        catch(Exception e)
        {
             try
             {
                 p_response.getWriter().println("Error " + e.toString());
             }
             catch(Exception e1)
             {
                 throw new ServletException(e1.toString());
             }
        }
    }
}
```

## JSP based partner application

The JSP based partner application can be implemented using a Java bean for generating a redirection URL and processing the redirected URL parameter from the SSO server. A JSP page should embed this bean, which can be included in all JSP based applications that require SSO 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 sample JSP based application can be implemented by implementing the following bean and JSP pages:

### SSOEnablerJspBean.java

This bean has the `getSSOUserInfo` method which returns the user information when the application cookie is already set. Otherwise, it redirects the user to the SSO server for authentication.

### ssoinclude.jsp

This page embeds the `SSOEnablerJsp` bean and should be included all application JSP pages where SSO functionality is necessary.

### ssosignon.jsp

This page embeds the `SSOEnablerJspBean` for generating redirection URL and processing the redirected URL parameter received from the SSO server.

### papp.jsp

This page is the main application page and requires SSO functionality. This page must include the `ssoinclude.jsp` page to get the user information.

### papplogoff.jsp

This JSP page removes the application session

### SSOEnablerJspBean.java

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

import oracle.security.sso.enabler.SSOEnablerException;

public class SSOEnablerJspBean
{
    private SSOEnablerBean  m_enablerBean = null;

   /**
    *  Default constructor
    */
    public SSOEnablerJspBean()
    {
```

```
        m_enablerBean = new SSOEnablerBean();

        m_enablerBean.setListenerToken("jspapp1:loginserver:443");
        m_enablerBean.setUrls("http://app1.xyz.com:80/jsp/papp.jsp",
            "http://app1.xyz.com:80");
        m_enablerBean.setAppCookieInfo("SSO_APP1_JSP_ID", "app1.xyz.com", "/");
        m_enablerBean.setDbConnectionInfo("papp", "papp", "db-hostname",
                    "1521", "db-sid");
    }

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

    public void setPartnerJspAppCookie(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.removePartnerAppCookie(p_response);
    }
}
```

### ssoinclude.jsp

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

```
    }
%>
```

### ssosignon.jsp

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

### papp.jsp

```
<%@ 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='papplogoff.jsp'>Logoff</a></center>");
    }
    else
    {
       response.getWriter().println("<center>User information "
         + "not found</center>");
    }
%>
```

### papplogoff.jsp

```
<%@ 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

## L



## O



## P

## S

## U



## W