HP OpenView Select Access

For the Windows®, HP-UX, Linux, and Solaris Operating Systems

Software Version: 6.1

Integration Paper for IBM WebSphere 6.x

August 30, 2005 10:18 am

Legal Notices

Warranty

Hewlett-Packard makes no warranty of any kind with regard to this document, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Hewlett-Packard shall not be held liable for errors contained herein or direct, indirect, special, incidental or consequential damages in connection with the furnishing, performance, or use of this material.

A copy of the specific warranty terms applicable to your Hewlett-Packard product can be obtained from your local Sales and Service Office.

Restricted Rights Legend

Use, duplication, or disclosure by the U.S. Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause in DFARS 252.227-7013.

Hewlett-Packard Company
United States of America

Rights for non-DOD U.S. Government Departments and Agencies are as set forth in FAR 52.227-19(c)(1,2).

Copyright Notices

Trademark Notices

HP OpenView Select Access includes software developed by third parties. The software HP OpenView Select Access uses includes:

- The OpenSSL Project for use in the OpenSSL Toolkit.
- Cryptographic software written by Eric Young.
- Cryptographic software developed by The Cryptix Foundation Limited.
- JavaService software from Alexandria Software Consulting.
- Software developed by Claymore Systems, Inc.
- Software developed by the Apache Software Foundation.
- JavaBeans Activation Framework version 1.0.1 © Sun Microsystems, Inc.
- JavaMail, version 1.2 © Sun Microsystems, Inc.
- SoapRMI, Copyright © 2001 Extreme! Lab, Indiana University.
- cURL, Copyright © 2000 Daniel Stenberg.
- Protomatter Syslog, Copyright © 1998-2000 Nate Sammons.
- JClass LiveTable, Copyright © 2002 Sitraka Inc.

For expanded copyright notices, see HP OpenView Select Access `<install_path>/3rd_party_license` directory.

All other product names are the property of their respective trademark or service mark holders and are hereby acknowledged.

Support

Please visit the HP OpenView web site at:

http://www.managementsoftware.hp.com/

This web site provides contact information and details about the products, services, and support that HP OpenView offers.

You can also go directly to the support web site at:

http://support.openview.hp.com/

HP OpenView online software support provides customer self-solve capabilities. It provides a fast and efficient way to access interactive technical support tools needed to manage your business. As a valuable support customer, you can benefit by using the support site to:

- Search for knowledge documents of interest
- Submit and track progress on support cases
- Manage a support contract
- Look up HP support contacts
- Review information about available services
- Enter discussions with other software customers
- Research and register for software training

Most of the support areas require that you register as an HP Passport user and log in. Many also require a support contract.

To find more information about access levels, go to:

http://support.openview.hp.com/access_level.jsp

To register for an HP Passport ID, go to:

https://passport2.hp.com/hpp/newuser.do

Contents

1 Understanding Your Select Access Integration

Select Access is an integral part HP's comprehensive Identity Management suite. It delivers a full solution for complex access management across the enterprise. Select Access:

- Automates access control and user life-cycle management.
- Extends the enterprise through federation.
- Delegates management to business owners and the end users themselves.

Along with robust workflow, user self-service, reporting, and delegated administration capabilities, Select Access is the most comprehensive access control system available. Select Access simplifies your ability to secure user access to IBM WebSphere 6.x Server™ resources.

Assumptions in this Document

This document assumes the following:

- That you have IBM WebSphere 6.x installed and running on your network.
- That you understand the features and functions of Select Access.
- That you have installed Select Access 6.1.
- That you have a working knowledge of Select Access and LDAP 3.0-compliant directory servers.

Integrating the Select Access and WebSphere

In a typical Select Access and EJB application server configuration, all requests for network resources must sent to the Web server. That way, when a request for an EJB application is made via the Web server, the Enforcer plugin queries the Policy Validator to determine whether or not the user is allowed to access the Web server and its resources. If the resource/application request is authorized according to the access policies set in Select Access, network requests proceed with the application server plugin which proxies a query to the servlet engine.

IBM WebSphere Application Server supports the IIS and Sun ONE Web servers, and the IBM HTTP server distributed with WebSphere. Depending on which Web server you use, you need to configure IBM WebSphere Application Server differently.

Understanding the Key Components of this Integration

When integrating Select Access with version 6.x of WebSphere, access controls are tied to three components:

- Select Access' user registry API
- Select Access' Trust Association Interface (TAI) adapter API
- Java Authorization Contract for Containers (JACC)

Identity Data Storage: The Custom User Registry API

Via Select Access' proprietary API for user registries, this integration allows you to choose where identity data is stored an retrieved. Custom user registries allows WebSphere to access data stored date from:

- The same identity data location as Select Access (that is, an LDAP v3.0 compliant directory).
- Another data store.

However, only the Select Access identity data locations supports role and group features native to the Select Access solution.

Identity Authentication: The Select Access TAI Adapter API

To authenticate users with Select Access as WebSphere's external security provider requires that the Select Access TAI adapter API in order to establish a trust association between Select Access and WebSphere. This trust-driven relationship gives Select Access the same privileges as authentication via WebSphere's internal mechanisms.

The authentication mechanism in this integration works as follows:

1. Resource requests are routed through the TAI interface.
2. Select Access' authentication servers authenticate the identity using the authentication method required (for example, password, tokens, and so on).
3. Once the identity has been authenticated the TAI adapter creates a Java `Subject` is created using the `SAPrincipalImpl` class (a child of `java.security.Principal` class).
4. Select Access creates a nonce and adds it to a hashtable as part of the `Subject`'s public credential element.
5. If any personalization data exists for the identity, a `SAP13n` object is created for this purpose.

Retrieving Identity Data

To retrieve identity data, call the `com.ibm.websphere.security.auth.WSSubject.getCallerSubject()` method in the current session. I think we should give more details. To retrieve the UID variable programmatically from the Subject on page 17 shows how to use this method so it yields the authenticated subject and all principal and credential data.

Identity Authorization: The JACC

Select Access uses JACC is used as the mechanism for authorizing access to a specific requested J2EE resource on WebSphere. Select Access has implemented all but the following mechanisms of the JACC:

- Pre-loading of configurations or policies
- JACC authorizations

JACC authorizations can work in conjunction with JAAS-based and TAI-based authentication processes, however. For details, see **http://java.sun.com/j2ee/javaacc/**.

This is due to JACC containers overlapping with Select Access' internal mechanisms.

2 Integration Tasks

Configuring WebSphere for External Authentication and Authorization

Before we can install TAI and JACC, reconfigure WebSphere to use these external security mechanisms. This involves the following tasks:

1. Download the Sun JSSE and install it on your WebSphere server. For details, see **http://www-128.ibm.com/developerworks/websphere/library/techarticles/0403_yu/0403_yu.html?ca=dnp-314**. HP recommends that you specifically refer to these topics: *Using the Sun JSSE implementation in the WebSphere run time* and *Static registration*.
2. Copy the following JAR files into the `<WS_install_path>/classes` folder:

   ```
   bcprov-jdk13-125.jar
   castor-0.9.3.19-xml.jar
   EnforcerAPI.jar
   jkarta-oro-2_0.jar
   jdom.jar
   ldapjdk.jar
   msgresources.jar
   protomatter.jar
   servletenforcer.jar
   shared.jar
   sslsupport.jar
   xalan.jar
   xercesImpl.jar
   xml.jar
   xmlsec.jar
   saint_util.jar
   saint_wassec.jar
   ```
3. Create a new properties file to capture Select Access' Enforcer plugin parameters that will be required by WebSphere. For details, see The Select Access Enforcer Properties File for WebSphere on page 7.
4. Configure WebSphere to use external mechanisms via its Administration Console. For details, see The Administration Console Configuration Parameters on page 9.
5. Configure Select Access to act as WebSphere's security provider. For details, see The WebSphere-specific Settings for Select Access on page 11.

The Select Access Enforcer Properties File for WebSphere

The file you are about to create contains vital information that the WebSphere server requires. It is a typical properties file that contains parameters similar to the following:

```
LogLevel=0
EnforcerAPIConfigFile=c:/Program Files/HP OpenView/Select Access/bin/
enforcer_websphere.xml
Service=http://MyWebsphereService.com:9080
Resource=/
SecurityRealm=MyOrg
```

To create your own properties file

1. Create a properties file named `<WS_install_path>/properties directory/sa_enforcer.properties`.
2. Add the following parameters to this file and save your changes.

Table 1 Properties to Configure

Parameter	Value	Description

`LogLevel`	Any of Select Access' supported log levels:
	`0=Debug`
	`4=Info`
	`8=Warning`
	`12=Error`
	`16=Fatal`
		Sets the level of logging output. The number you assign reflects the amount of detail you want to capture.

`EnforcerAPIConfigFile`	The path and filename of your WebSphere `enforcer.xml` configuration file.	Created by the setup tool when using the Generic Enforcer setup. It contains information on how to connect to the Policy Validator from the Enforcer plugin.
`Service`	URL For example, `http://localhost:9080/webspherestartup`	Sets the URL to WebSphere's default authentication service. This URL is required for two purposes • For startup and shutdown of the WebSphere server. • For default Select Access authentication, which requires a service/resource as well as identity credentials. Since this URL is required for both, ensure you set permissions in the Policy Builder, deny access to all but the most important administrators. Use either Password or NTLM authentication for this service/resource combination.
`Resource`	URI to Resource Name	Sets the URI to WebSphere's default authentication resource page.
`SecurityRealm`	Realm or domain of Server	Sets the domain name or realm on which the service is available on.

The Administration Console Configuration Parameters

Open the Administration Console to configure or reconfigure you WebSphere server to use external authentication and authorization mechanisms. Unique properties need to be configured for:

- General Global Security Settings. For details, see To enable external security mechanisms on page 9.
- Custom User Registries. For details, see To configure Custom User Registries parameters on page 9.
- Trust Associations. For details, see To configure Trust Association parameters on page 10.
- JACC Provider for Select Access properties. For details, see To configure JACC provider settings on page 10.
- Stopping the WebSphere server. For details see To stop and restart the WebSphere server on page 10.

To enable external security mechanisms

1. In the WebSphere administrative console's navigation panel, click **Security→Global Security**.
2. In the **Configuration** tab, enable the following mechanisms by:
   - Checking the **Enable global security** box
   - Checking the **Enforce Java 2 security** box
   - Checking the **Issue permission warning** box
   - In the **Active authentication mechanism** droplist box, selecting **Lightweight Third Party Authentication (LTPA**
   - In the **Active user registry** droplist box, selecting **Custom user registry**.
3. Click **Ok** and save these settings.

To configure Custom User Registries parameters

1. In the WebSphere Global Security menu at the top of the page, click **Custom User Registries**.
2. In the **Configuration** tab, configure the following values:

   Server user ID=<myID>
   Server user password=<myPwd>

   Remember your ID and password you set here. You need to use these credentials when stopping and restarting the server, as described in To stop and restart the WebSphere server on page 10.

   Custom registry class name=
   com.hp.ov.selectaccess.solutions.websphere.registry.SelectAccessRegistry

   Ignore case for authorization=checked
3. Click **Custom Properties**.
4. Add the following property name and value:

Name =**SA_ENFORCER**

Value =**sa_enforcer.properties**

5 Click **Ok** and save these settings.

To configure Trust Association parameters

1 In the WebSphere Global Security menu at the top of the page, click **Trust Association**.

2 In the **Configuration** tab, click the **Enable trust association** box.

3 Click **Ok** and save these settings.

4 Create a new interceptor for Select Access by:

a In the menu at the top of the page, click **Interceptors**.

b Create a new interceptor named **com.hp.ov.selectaccess.solutions.websphere.tai.SelectAccessTAIv6**.

c Click **Custom Properties**.

d Add the following property name and value:

Name =**SA_ENFORCER**

Value =**sa_enforcer.properties**

5 Click **Ok** and save these settings.

To configure JACC provider settings

1 In the WebSphere Global Security menu at the top of the page, click **Authorization providers→External JACC provider**.

2 In the **Configuration** tab, configure the following values:

Name=Select Access

Policy class name=com.hp.ov.selectaccess.solutions.websphere.jacc.SAPolicyImpl

Policy configuration class=com.hp.ov.selectaccess.websphere.jacc.SAPolicyConfigurationFactoryImpl

Select Access does not support external configuration with this version. However, this integration does include a configuration factory so Select Access can accept the JACC parameters.

3 Click **Custom Properties**.

4 Add the following property name and value:

Name =**SA_ENFORCER**

Value =**sa_enforcer.properties**

5 Click **Ok** and save these settings.

To stop and restart the WebSphere server

1 Ensure the Select Access Policy Validator is running. You cannot stop WebSphere without a running Policy Validator.

2 Change to the `<WS_install_path>\bin` folder.

3 Using the credentials you set in To configure Custom User Registries parameters on page 9, at the command prompt and type:

```
stopserver server1 -username <MyID> -password <MyPwd>
```

4 To restart the server, at the command prompt type:

```
startserver server1
```

The WebSphere-specific Settings for Select Access

To prepare Select Access to act as WebSphere's Security Provider you must:

- Configure the appropriate kind of personalization parameter so that identity information can be shared by the two systems. You must add a personalization variable named UID for the uid attribute for every authentication server used to authenticate identities for WebSphere. For details see To pass identity information to WebSphere on page 11.

  Configuring the personalization parameters for your authentication services gives you the ability to retrieve custom attributes data from Select Access' identity data store programmatically from the WebSphere Subject. For details on how to retrieve attributes programatically see Personalization Data Retrieval on page 17.

- Select Access-protect specific WebSphere server resource by adding these resources to the Policy Builder Resource Tree and set the appropriate level of permissions for it. For details, see To register and secure WebSphere resources on page 11.

To pass identity information to WebSphere

1 In the Policy Builder, click **Tools→Authentication Services**.

- To add a new authentication service and configure the personalization properties for that service, click the **Add** button in the **Authentication Services** dialog box. Choose the authentication method you require.
- To modify an existing authentication service and modify the personalization properties for that service, select a service entry and click the **Properties** button in the **Authentication Services** dialog box.

2 In the resulting **Authentication Properties** dialog box, click the **Personalization** tab.

3 In the **Identity Data** tab, check the **Store identity attribute in** box.

4 Click the **Add** button to create a new row.

5 Type **`uid, userid`** in the new **Directory Attribute Name** column.

6 Type **`UID`** in the **Environment Variable Name** column.

To register and secure WebSphere resources

1 At minimum, add two resource services: one for WebSphere Administration assets and one for corporate assets served to end-users by the WebSphere server. This intelligently separates WebSphere resources so you can set different authentication and access policies according to the types of identities who will be requesting access to these resources. For details, see Chapter 3, Building your Identities and Resources Trees, in the *HP OpenView Select Access 6.1 Policy Builder Guide*.

2 Set your authentication and authorization policies for each service you have created:

- Enable SelectAuth and use at least one authentication service to authenticate identities for WebSphere. For details, see Chapter 5, Authentication Basics: Select Auth & Personalization, in the *HP OpenView Select Access 6.1 Policy Builder Guide*.
- Set a Deny access policy to authorize identities for WebSphere. This prevents users from directly accessing the WebSphere server. For details, see Chapter 7, Controlling Network Access, in the *HP OpenView Select Access 6.1 Policy Builder Guide*.

3 Under each service you have created, add the appropriate number of resources served to identities by that service. For details, see Chapter 3, Building your Identities and Resources Trees, in the *HP OpenView Select Access 6.1 Policy Builder Guide*.

4 Set your authentication and authorization policies for each resource if applicable.

- For each resource you add, the SelectAuth policy is inherited from the parent service. You do not need to explicitly set this policy.
- If you want to set a different access policy other than the Deny policy that is inherited from the parent service, choose a different policy for each Identity/resource combination as required.

5 Test the Enforcer plugin on your Web server by trying to access the server directly via the "snoop" servlet. For example, `http://<server name>:9080/snoop`. If you've set the appropriate SelectAuth policy on this resource, Select Access prompts you to enter your credentials.

If you are accessing the WebSphere content via the WebSphere server, you may have a problem with your WebSphere and/or its proxy configuration. For details on how to setup WebSphere with a proxy, see the *Select Access 6.0 Integration Paper for WebSphere 5.x*.

Configuring WebSphere for the JAAS Login

Chapter 3, Programmatic Integration Tasks, introduces you to the JAAS login module. If you require the JAAS login module because WebSphere includes applications that do not use a standard Web interface, you have to configure WebSphere to include an `SALogin` reference.

The SALogin Setup

The `SALogin` reference can be set in the WebSphere administrative console by configuring parameters in the Application Login and Configuration page.

To configure an alias for the SALogin reference

1. In the Administration Console, click **Security→Global security→JAAS Configuration→Application Login Configuration**.
2. In the **Application Login Configuration** page, click the **Create** button.
3. Create a new alias named **`SALogin`** using the following class:

`com.hp.ov.selectaccess.solutions.websphere.jaas.SAApplicationLoginModule`

4. Click **Custom Properties** to create a new property named **`SA_ENFORCER`** with a value of **`sa_enforcer`**.

Ensure the **`sa_enforcer`** value is appropriate for your JAAS LoginModule authentication strategy. See the JAAS documentation for more details on different authentication strategies.

Protecting a Web Service

A JAAS default systems login must be setup with the following JAAS class to authenticate a web service resource:

```
com.hp.ov.selectaccess.solutions.websphere.jaas.
SASystemDefaultLoginModule.
```

This login module only uses password authentication. You have to configure the resource to use the password or NTLM authentication method. The login module will not be able to retrieve location for authorization therefore it will look for the default authorization in `sa_enforcer.properties` and authenticate against it. See Configuring WebSphere for the JAAS Login on page 13 for configuration information.

Web Service Resource Authorization

Web service resource authorization is a two step process. The authorization is checked against the SOAPAction in the HTTP Headers (`wsdl targetNamespace tag`) and the actual URL. The SOAPAction is defined in the SOAP 1.1 specifications but removed from SOAP 1.2.

SOAP1.1

If your SOAP message is still using the 1.1 standard, but you do not want to use SOAPAction, you need to create a policy in the Policy Builder with the `targetNamespace` as the resource and allow unknown users to access it.

Example

```
<wsdl:definitions name="AddressBook" targetNamespace="http://
addressbook.com/AddressBook/" xmlns:tns="http://addressbook.com/
AddressBook/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:wsdl="http:/
/schemas.xmlsoap.org/wsdl/">
```

To use the SOAP1.1 standard without SOAPAction

1. Create a policy named `http://addressbook.com/AddressBook` with unknown user rights.
2. Set the permissions on the specific opertations, for example, `http://addressbook.com/AddressBook/SaveAddress` to protect these resources.
3. Place a policy on the URL resource for the HTTP access to the SOAP object. For an example, if the URL to access the SOAP is `http://websphere.can.hp.com:9080/AddressBook/services/AddressBookPort`, this is also the URL to protect in the Policy Builder. The URL is global for all operations on this particular SOAP service.

3 Programmatic Integration Tasks

With this integration, there are three instances that require some extra programmatic integration steps:

- When using the JAAS login module for applications that do not use a standard Web interface. For details, see The JAAS Login Module on page 15.
- When an EJB requires authentication. For details, see EJB Authentication on page 17.
- When you want to retrieve personalization information from the Subject. This personalization is forwarded by Select Access as described in Chapter 2, Integration Tasks. For details, see Personalization Data Retrieval on page 17.

The JAAS Login Module

The code sample shown in To login with JAAS programmatically on page 16, uses the `HttpRequest` and `HttpResponse` servlet object in a JAAS `LoginModule`. Important things to note are:

- The callstore variable: is an array of `SACallbackStore`. Its purpose is to store callback variables used in the validation process. In this example, we use the `SAWebCallbackStore` that takes the `HttpServletRequest` and `HttpServletResponse` as input parameters.

  Valid callback stores include:

  — `SACertCallbackStore`—holds a X509 certificate.

  — `SALocationCallbackStore`—holds a URL which is needed by all except `SAWebCallbackStore`.

  — `SABasicAuthCallbackStore`—holds a user ID and password.

  — `SANonceCallbackStore`—holds the Select Access nonce.

  If you are using a callback store other then the `SAWebCallbackStore`, you must include the `SALocationCallbackStore`. This parameter describes where and which resource to authenticate against.

- The Select Access login name reference: `SALogin` is a variable that you must set according to the details described in The SALogin Setup on page 13. With this information, the `SAApplicationCallbackHandler` takes the `SACallbackStore` array runs validations against the data entered.

  If there is more than one callback store, the login module performs authentication similar to the following until it can successfully authenticate an identity. All subsequent authentication steps are then aborted.

```
SAWebCallbackStore
SANonceCallbackStore
SACertCallbackStore
SABasicAuthCallbackStore
```

You can also add an external `CallbackHandler` as part of the constructor for GUI-based login validation with the IBM user ID and password (for example, `com.ibm.wsspi.wssecurity.auth.callback.GUIPromptCallbackHandler`).

- The Principal object: The `SAPrincipalImpl` contains user ID, nonce and personalization data that is returned after the Select Access authentication. Sometimes, depending on how you have configured WebSphere, there can be more than one Principal object in the Subject so you may need to search for it.

To login with JAAS programmatically

```
import com.hp.ov.selectaccess.solutions.jaas.*;

import com.hp.ov.selectaccess.solutions.websphere.jaas.*;

public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
        throws IOException, ServletException {

HttpServletRequest httpReq = (HttpServletRequest) request;
HttpServletResponse httpRes = (HttpServletResponse) response;

try {
        1) SACallbackStore [] callstore = new SACallbackStore[1];
        callstore[0] = new SAWebCallbackStore(httpReq, httpRes);

        2) LoginContext lc = new LoginContext("SALogin", new
SAApplicationCallbackHandler(callstore));

        lc.login();

        javax.security.auth.Subject caller_subject = lc.getSubject();

        if (caller_subject != null) {
                java.util.Iterator iter = caller_subject.getPrincipals().iterator();

                while (iter.hasNext()) {
                        java.security.Principal principal =
(java.security.Principal)iter.next();

                        if (principal instanceof SAPrincipalImpl) {
                             3) SAPrincipalImpl sa_principal = (SAPrincipalImpl)principal;

                               String nonce = sa_principal.getNonce();
                        }
                }
        }
} catch (Exception le) {
        System.out.println("Can not login. error: " + le);
        return;
}
}
```

EJB Authentication

EJB authentication requires the same SALogin alias required by the JAAS login module (for details, see Configuring WebSphere for the JAAS Login on page 13). If you did not configure JACC with this alias, your EJB applications cannot retrieve the authenticated Subject either from a JAAS login. Nor can it get the default one if you are using a Web login.

The following code sample shows how to get the default Subject for the session from TAI. It uses the `com.ibm.websphere.security.auth.WSSubject.getCallerSubject()` method. Include the appropriate credentials or Principal; otherwise you won't be able to create the EJB.

To authenticate EJBs programmatically

```
Subject caller_subject = com.ibm.websphere.security.auth.WSSubject.getCallerSubject();

Hashtable h = new Hashtable();

h.put(Context.INITIAL_CONTEXT_FACTORY,"com.ibm.websphere.naming.WsnInitialContextFactory")
;
h.put(Context.PROVIDER_URL, "iiop://localhost:2809");
h.put(Context.SECURITY_PRINCIPAL, caller_subject.getPrincipals());
h.put(Context.SECURITY_CREDENTIALS,caller_subject.getPublicCredentials());

Context ctx = new InitialContext(h);

Object home = ctx.lookup("ejb/ejbs/SATestHome");

SATestHome saHome = (SATestHome)PortableRemoteObject.narrow(home, SATestHome.class);

SATest simpleMath = (SATest)PortableRemoteObject.narrow(saHome.create(), SATest.class);

System.out.println("Call simpleMath.add(2,4) = " + simpleMath.add(2,4));

simpleMath.remove();
```

Personalization Data Retrieval

You can retrieve any variable configured by the Policy Builder from the Subject. In the case of the integration requirements for these two products, UID is the variable required.

To retrieve the UID variable programmatically from the Subject

```
Subject caller_subject = com.ibm.websphere.security.auth.WSSubject.getCallerSubject();

java.util.Iterator iter = caller_subject.getPrincipals().iterator();

while (iter.hasNext()) {
        java.security.Principal principal = (java.security.Principal)iter.next();

        if (principal instanceof SAPrincipalImpl) {
                SAPrincipalImpl sa_principal = (SAPrincipalImpl)principal;

                SAP13n p13n = sa_principal.getP13n();

                System.out.println("This is the uid of the user: " +
                        p13n.getAttribute("UID"));
         }
```

4 Troubleshooting

Q--->Why does my user name or user ID come up as something long such as cn=Joe Doe,dc=can,dc=hp,dc=com?

A--->You need to configure a personalization parameter named UID with the LDAP attribute `uid`.

Q--->Why doesn't WebSphere call the Validator for every Web resource access?

A--->Turn SSO off in WebSphere for LTPA because we are using Select Access for this instead of the internal WebSphere mechanism. In the administrative console, click on Security->Global security->Authentication mechanisms->LPTA->Single sign-on (SSO) make sure the Enable checkmark is not checked.

Q--->I can't log into the WebSphere administrative console after I've setup security. Why is this now happening?

A--->You need to do the following:

- Create a new resource for `https://localhost:9043`.
- Ensure you have a WebSphere identity/user with the same profile data as you had setup in Select Access.

Once you've checked these two items, you should test authentication with Password service. This is the easiest authentication service to use testing this authentication processes.

Q--->How can I log out of the WebSphere administrative console *and* Select Access at the same time. When I log out of the administrative console, the Select Access session seems to still be active.

A--->You need to do the following in Select Access:

- Create a logout policy. Set this conditional access policy logs against the following WebSphere resource for all Known Identities:
  `https://localhost:9043/ibm/console/logout.do`
- Create a redirect policy to the URL described above. Set this conditional policy against the following WebSphere resource for all WebSphere administration identities:
  `https://localhost:9043/ibm/console/logon.jsp`

Q--->I'm getting abnormal results in WebSphere. Sometimes resources are not being authenticated properly. Why is this happening?

A--->Don't use internal WebSphere security mechanisms. You must delegate all identity and access management to Select Access. Otherwise, when you introduce a WebSphere security infrastructure such as roles or SSO using LTPA, it will conflict with Select Access' mechanisms and you may not get the proper results.

5 API Reference

This API reference documents the modules, exception-handling methods, and classes used with programmatic integration tasks documented in Chapter 3, Programmatic Integration Tasks.

The Login Modules

There are two login modules:

- `com.hp.ov.selectaccess.solutions.websphere.jaas.SAApplicationLoginModule`–For programming a JAAS LoginModule authentication in WebSphere for application logins. To create a JAAS application login alias, see To configure an alias for the SALogin reference on page 13 for details.
- `com.hp.ov.selectaccess.solutions.websphere.jaas.SASystemWebLoginModule`—Not recommended for use; can be used as a Web system-wide JAAS LoginModule for authentication in the WebSphere application server.

The Exception Handling Classes

There are two ways to handle exceptions:

- `com.hp.ov.selectaccess.solutions.util.SAP13nNotFoundException`–When a personalization object cannot be found by the Principal, this exception will be raised.
- `com.hp.ov.selectaccess.solutions.util.SAAuthException`—The general Select Access authentication exception that can be raised when an exception situation occurs.

Other Classes and Their Methods

The functions upon which the integration framework for Select Access and WebSphere is built are defined in classes listed below. Methods are listed alphabetically in the sections that follow.

Class Name	Class Contents		
com.hp.ov.selectaccess.solutions.jaas.SABasicAuthCallbackStore	Contains the		
	Method Name	**Page**	**Description**
	getAttribute(String key)	page 25	Gets the string of the attribute with the given key.
com.hp.ov.selectaccess.solutions.jaas.SACertCallbackStore	Contains the		
	Method Name	**Page**	**Description**
	SACertCallbackStore(X509Certificate [ ] certificates, Integer keySize)	page 29	Stores the X509 certificate that will be used in the JAAS LoginModule for authentication.
com.hp.ov.selectaccess.solutions.jaas.SALocationCallbackStore	Contains the		
	Method Names	**Page**	**Description**
	SALocationCallbackStore(URL location, String sourceHost)	page 30	Stores the URL or string path location that is be used in the JAAS LoginModule for authentication.
	SALocationCallbackStore(String protocol, String host, String path, String port, String sourceHost)	page 30	
com.hp.ov.selectaccess.solutions.jaas.SAWebCallbackStore	Contains the		
	Method Name	**Page**	**Description**
	SANonceCallbackStore(String nonce)	page 31	Stores the nonce that is used in the JAAS LoginModule for authentication.

Class Name	Class Contents		
com.hp.ov.selectaccess.solutions.jaas.SAPrincipalImpl	The SelectAccess Principal is stored in a Subject that has been authenticated by the SelectAccess Validator. In the Principal, you will be able to retrieve the user name, current service name, nonce and the personalization object. The nonce can be used in subsequent authentication or authorization calls.		
	Method Names	**Page**	**Description**
	getName()	page 25	Returns the name of the current object.
	getService()	page 25	Returns the URL of the current service.
	isPerimeterAuth()	page 26	Determines whether or not perimeter authentication is used.
	getNonce()	page 27	Returns the nonce of the current Subject.
	getP13n()	page 27	Returns the personalization data of the current Subject.
	getSignature()	page 27	Not used at this time.
	setSignature(String signature)	page 31	Not used at this time.
com.hp.ov.selectaccess.solutions.util.SAP13n	Retrieves personalization data that is set by the administrator in the Policy Builder. You can retrieve this data from this object.		
	Method Name	**Page**	**Description**
	getAttribute(String key)	page 25	Gets the string of the attribute with the given key.
	keySet()	page 26	Not used at this time.
	isEmpty()	page 26	Determines whether there is any personalization data in the object.

Class Name	Class Contents		
com.hp.ov.selectaccess.solutions.websphere.jaas.SAApplicationCallbackHandler	The callback handler to use for use with all SelectAccess JAAS LoginModules. If you want to use user ID and password as the last resort for authentication and would like to use a different callback handler, (such as the IBM GUI callback handler) then use the second constructor. The external callback handler will not be used until all roads are exhausted.		
	Method Name	**Page**	**Description**
	SANonceCallbackStore(String nonce)	page 31	Used to handle callbacks for use with all SelectAccess JAAS LoginModules.
	SAApplicationCallbackHandler(CallbackHandler callbackHandler, SACallbackStore [ ] callbackStores)	page 28	Used to handle callbacks for use with all SelectAccess JAAS LoginModules. Use this constructor if you want to use user ID and password as an authentication fall-back mechanism with a different callback handler (such as the IBM GUI callback handler).
com.hp.ov.selectaccess.solutions.websphere.jaas.SANonceCallbackStore	Used to store the nonce that will be used in the JAAS LoginModule for authentication.		
	Method Name	**Page**	**Description**
	SANonceCallbackStore(String nonce)	page 31	Stores the nonce that is used in the JAAS LoginModule for authentication.

getAttribute(String key)

Description Gets the string of the attribute with the given key.

Class `com.hp.ov.selectaccess.solutions.util.SAP13n`

Syntax
```
String getAttribute(String key)
```

Parameters
- `String key` — An object containing the cache keys for the current identity's LDAP record. In Select Access, keys are based on DN and UIDs (possibly several).

Return Values

Remarks

Example The following code sample.

getName()

Description Returns the name of the current object.

Class `com.hp.ov.selectaccess.solutions.jaas.SAPrincipalImpl`

Syntax
```
String getName(
```

Parameters None.

Return Values The name of the current object.

Remarks

Example The following code sample.

getService()

Description Returns the URL of the current service.

Class `com.hp.ov.selectaccess.solutions.jaas.SAPrincipalImpl`

Syntax
```
String getURL()
```

Parameters None.

Return Values The name of the current service.

Remarks

Example The following code sample.

keySet()

Description Not used at this time.

Class `com.hp.ov.selectaccess.solutions.util.SAP13n`

Syntax

```
Set keySet()
```

Parameters None.

Return Values

Remarks

Example The following code sample.

isEmpty()

Description Determines whether there is any personalization data in the object.

The retrievable data is set in the Policy Builder. For details, see To pass identity information to WebSphere on page 11.

Class `com.hp.ov.selectaccess.solutions.util.SAP13n`

Syntax

```
boolean isEmpty()
```

Parameters None.

Return Values

- `true` — No information exists in the object.
- `false` — The object contains information.

Remarks

Example The following code sample.

isPerimeterAuth()

Description Determines whether or not perimeter authentication is used.

Class `com.hp.ov.selectaccess.solutions.jaas.SAPrincipalImpl`

Syntax

```
String isPerimeterAuth()
```

Parameters None.

Return Values

- `true` — Perimeter authentication is used.
- `false` — Perimeter authentication is used.

. **Remarks**

Example The following code sample.

getNonce()

Description	Returns the nonce of the current Subject.
Class	`com.hp.ov.selectaccess.solutions.jaas.SAPrincipalImpl`
Syntax	`String getNonce()`
Parameters	None.
Return Values	The nonce of the current subject.
Remarks	The nonce can be used in subsequent authentication or authorization calls.
Example	The following code sample.

getP13n()

Description	Returns the personalization data of the current Subject.
Class	`com.hp.ov.selectaccess.solutions.jaas.SAPrincipalImpl`
Syntax	`SAP13n getP13n()`
Parameters	None.
Return Values	
Remarks	
Example	The following code sample.

getSignature()

Description	Not used at this time.
Class	`com.hp.ov.selectaccess.solutions.jaas.SAPrincipalImpl`
Syntax	`String getSignature()`
Parameters	None.
Return Values	
Remarks	
Example	The following code sample.

SAApplicationCallbackHandler(SACallbackStore [ ] callbackStores)

Description Used to handle callbacks for use with all SelectAccess JAAS LoginModules.

Class
```
com.hp.ov.selectaccess.solutions.websphere.jaas.
SAApplicationCallbackHandler
```

Syntax
```
SAApplicationCallbackHandler(SACallbackStore [] callbackStores)
```

Parameters
- `SACallbackStore []` — .
- `callbackStores` — .

Return Values

Remarks The external callback handler is not be used until all roads are exhausted.

Example The following code sample.

SAApplicationCallbackHandler(CallbackHandler callbackHandler, SACallbackStore [ ] callbackStores)

Description Used to handle callbacks for use with all SelectAccess JAAS LoginModules. Use this constructor if you want to use user ID and password as an authentication fall-back mechanism with a different callback handler (such as the IBM GUI callback handler).

Class
```
com.hp.ov.selectaccess.solutions.websphere.jaas.
SAApplicationCallbackHandler
```

Syntax
```
SAApplicationCallbackHandler(CallbackHandler callbackHandler,
SACallbackStore [] callbackStores)
```

Parameters
- `SACallbackStore []` — .
- `callbackStores` — .

Return Values

Remarks The external callback handler is not be used until all roads are exhausted.

Example The following code sample.

SABasicAuthCallbackStore(String username, char [ ] password)

Description Stores the user name and password that is used by the JAAS LoginModule for authentication.

Class `com.hp.ov.selectaccess.solutions.jaas.SABasicAuthCallbackStore`

Syntax

```
SABasicAuthCallbackStore(String username, char [] password)
```

Parameters

- `username` — The name of the identity in the Subject.
- `password` — The password of the identity in the Subject.

Return Values

Remarks The password is removed from the object after it is used by `LoginModule`.

Example The following code sample.

SACertCallbackStore(X509Certificate [ ] certificates, Integer keySize)

Description Stores the X509 certificate that will be used in the JAAS LoginModule for authentication.

Class `com.hp.ov.selectaccess.solutions.jaas.SACertCallbackStore`

Syntax

```
SACertCallbackStore(X509Certificate [] certificates, Integer keySize)
```

Parameters

- `X509Certificate [] certificates` — The X509 format certificate in the Subject.
- `keysize` — .

Return Values

Remarks

Example The following code sample.

SALocationCallbackStore(URL location, String sourceHost)

Description Stores the URL or string path location that is be used in the JAAS LoginModule for authentication.

Class `com.hp.ov.selectaccess.solutions.jaas.SACertCallbackStore`

Remarks This callback is required if a `SAWebCallbackStore` is not used.

Syntax

```
SALocationCallbackStore(URL location, String sourceHost)
```

Parameters

- `URL location` — The complete URI to the resource.
- `String sourceHost` — .

Return Values

Remarks

Example The following code sample.

SALocationCallbackStore(String protocol, String host, String path, String port, String sourceHost)

Description Stores the URL or string path location that is be used in the JAAS LoginModule for authentication.

Class `com.hp.ov.selectaccess.solutions.jaas.SACertCallbackStore`

Remarks This callback is required if a `SAWebCallbackStore` is not used.

Syntax

```
SALocationCallbackStore(String protocol, String host, String path,
String port, String sourceHost)
```

Parameters

- `String protocol` — The complete protocol required for the resource. Can be one of:

  — .
- `String host` — .
- `String path` — The complete path and file name to the resource.
- `String sourceHost` — .

Return Values

Remarks

Example The following code sample.

SANonceCallbackStore(String nonce)

Description Stores the nonce that is used in the JAAS LoginModule for authentication.

Class `com.hp.ov.selectaccess.solutions.websphere.jaas.SANonceCallbackStore`

Syntax
```
SANonceCallbackStore(String nonce)
```

Parameters None.

Return Values None.

Remarks The nonce can be used in subsequent authentication or authorization calls.

Example The following code sample.

SAWebCallbackStore(HttpServletRequest request, HttpServletResponse response)

Description Stores the `HttpServletRequest` and `HttpServletResponse` that will be used in the JAAS LoginModule for authentication.

Class `com.hp.ov.selectaccess.solutions.jaas.SAWebCallbackStore`

Syntax
```
SAWebCallbackStore(HttpServletRequest request, HttpServletResponse
response)
```

Parameters

- `HttpServletRequest request` — .
- `HttpServletResponse response` — .

Return Values

Remarks

Example The following code sample.

setSignature(String signature)

Description Not used at this time.

Class `com.hp.ov.selectaccess.solutions.jaas.SAPrincipalImpl`

Syntax
```
void setSignature(String signature)
```

Parameters

- `String signature` — .

Return Values

Remarks

Example The following code sample.