HP OpenView Select Federation

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

Software Version: 6.5

Web Services Developer's Guide

Document Release Date: April 2006
Software Release Date: April 2006

Documentation Updates

This manual's title page contains the following identifying information:

- Software version number, which indicates the software version
- Document release date, which changes each time the document is updated
- Software release date, which indicates the release date of this version of the software

To check for recent updates, or to verify that you are using the most recent edition of a document, go to:

http://ovweb.external.hp.com/lpe/doc_serv/

You will also receive updated or new editions if you subscribe to the appropriate product support service. Contact your HP sales representative for details.

Support

Please visit the HP OpenView support web site at:

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

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

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 valued support customer, you can benefit by using the support site to:

- Search for knowledge documents of interest
- Submit enhancement requests online
- Download software patches
- 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://www.hp.com/managementsoftware/access_level

To register for an HP Passport ID, go to:

http://www.managementsoftware.hp.com/passport-registration.html

Contents

1 Introducing the HP OpenView Select Federation Web Services Developer's Guide

This *HP OpenView Select Federation Web Services Developer's Guide* describes how to use the Select Federation Software Developer's Kit (SDK) to develop and deploy new web services. This guide also includes samples that you can build and use as a basis for your own development efforts.

➤ This release provides a trial Software Developers Kit (SDK) for developing and deploying new web services. Therefore, the APIs and parameters described in this guide are subject to change.

The Select Federation SDK consists of the following:

- This *OpenView Select Federation Web Services Developer's Guide*.
- *HP OpenView Select Federation Web Application Developer's Guide*, which describes how to add federation functionality to web applications, how to use the SDK to integrate an Authority (IDP) installation of Select Federation with back-end data sources and various authentication systems, and how to build the API samples and what the samples demonstrate.
- API documentation in the `<`*`cd-base-directory`*`>/docs/api/index.html` file.
- Web Application API samples in the `<`*`cd-base-directory`*`>/api/samples/` directory.
- Web Services samples in the `<`*`cd-base-directory`*`>/web-services/filters/samples/` and `<`*`cd-base-directory`*`>/web-services/api/samples/` directories

Audience

This guide is intended for developers that want to extend applications that are integrated with Select Federation with identity-based web service functionality.

Prerequisites

This guide assumes a working knowledge of:

- Identity Management
- Federated Identity
- HP *OpenView Select Federation Web Application Developer's Guide* and the various APIs that ship with Select Federation

System Requirements

HP OpenView Select Federation is designed to work with a number of hardware and operating systems configurations. The flexibility inherent in Select Federation extends to the third-party applications that it supports, namely the application servers, database servers, and LDAP servers. See “System Requirements” in Chapter 3 of the *HP OpenView Select Federation Configuration and Administration Guide* for the specific hardware and software system requirements.

Overview

This chapter briefly discussed the key concepts of Identity-based Web Services, and explains the major steps in defining such services. This chapter is not a replacement for the large body of material that is available in the form of specifications and books on the topic of Web Services.

The *HP OpenView Select Federation Web Application Developer's Guide* introduces the concept of *roles* that a particular deployment can take on, which are *Applications* and *Authorities*. Web Services functionality adds two more roles:

- **Web Service Provider (WSP)** — A party that offers some service through a well defined HTTP/SOAP/XML interface.
- **Web Service Consumer (WSC)** — A party that uses such service, by sending messages to it that are in accordance with the specification of the service.

The service is *Identity-Based* if within the scope of a single transaction the service is *about* a single principal. Examples of identity-based services include the following:

- Service to obtain the geo-location of a user
- Calendar service that allows the WSC to add an entry to the calendar of a user
- Service that knows if a particular user is available for chat.

A WSC that wishes to invoke a specific WSP (such as a calendar) service of a particular user needs various pieces of information, such as:

- Service available for the user
- Network address of the service
- Interface of the service
- Credentials for the service
- ID of the user at that service

It is important to realize that many of these pieces are not specific to the service. Similar information is required to invoke a geo-location service, for example. The combination of auxiliary services and various methods that the WSC can use to get these pieces of information can be viewed as a *Service Invocation Framework* (SIF). The HP OpenView Select Federation Web Service APIs essentially hide the complexity of Service Invocation Frameworks from developers, who can concentrate on service-specific functionality.

Currently there is only one SIF for interoperable Identity-Based Web Services, the Liberty Alliance ID-WSF set of specifications. HP OpenView Select Federation and its SDK support the ID-WSF 1.0 and 1.1 specifications. However it is foreseeable that other SIFs will emerge, possibly with a slightly different focus. Future versions of the Select Federation Web Service APIs should be able to accommodate such SIFs with minimal impact.

Chapters Summary

The following table provides an overview of this guide's contents..

Table 1 Chapters Summary

Chapter	**Description**
Chapter 1, Introducing the HP OpenView Select Federation Web Services Developer's Guide	This chapter provides a brief overview of the content of this Developer's Guide.
Chapter 2, Adding Web Service Consumer Functionality to an Application	This chapter provides detailed information about using the Select Federation SDK to add WSC (Web Service Consumer) functionality to an application.
Chapter 3, Developing an Identity-Based Web Service	This chapter provides detailed information about using the Select Federation SDK to develop and deploy new ID-WSF compliant web services.
Chapter 4, Support for LUAD-WSC Implementations	This chapter provides information about the services that Select Federation offers to support LUAD-WSC implementations.
Chapter 5, Samples	This chapter provides descriptions of the web services samples included on the Select Federation CD to help you understand the capabilities of Select Federation.

2 Adding Web Service Consumer Functionality to an Application

The HP OpenView Select Federation SDK makes adding WSC functionality to an application surprisingly simple. The main developer responsibility is to write code that deals with the service-specific part of the service interface. For example, adding an entry to a user's calendar or retrieving the geo-location of a user.

However, the Liberty Alliance ID-WSF framework requires that a user must be authenticated by an Identity Provider to the WSC (which at that moment acts as Service Provider) at some point prior to invocation of the WSP. Therefore, it is good practice to enable an Application for a federation first, and then add WSC functionality to it. See the *HP OpenView Select Federation Web Application Developer's Guide* on how to enable an application for a federation.

If the application is going to act as a WSC it is useful to consider the following:

- The Select Federation SDK does **not** provide WSC functionality to applications that are enabled only through integration with HP OpenView Select Access.
- WSC functionality can be added to a federated application irrespective of the type of federation. One-time identifiers can work as well as persistent identifiers and account-linking is **not** a prerequisite.

The technique to add the WSC functionality is dependent on the way the application is enabled for a federation. Applications that make use of the SDK IIS or Apache filters can add code that uses the HP OpenView Select Federation WSC Proxy service (described in the next section). Applications that rely on the Java APIs of the SDK use the SPWSCAPI.

WSC Proxy Service

The HP OpenView Select Federation SDK contains a web application (a `war` file) that can be added to an existing Select Federation 6.5 installation and adds support for the IIS and Apache filters. See Chapter 4 "Using Filters to Protect Web Applications" in the *HP OpenView Select Federation Web Application Developer's Guide* for details on the IIS and Apaches filters.

This `war` web application also adds a WSC Proxy that can be used by applications that are deployed on IIS or Apache filters. The WSC Proxy provides a simple XML interface towards such applications, with a subset of the functionality offered by the Java SPWSCAPI.

To use the `WSCProxy` the application should construct the XML message that should be sent in the `soap:Body` part of the message to the WSP. The application wraps this XML message in another element that is a request to the `WSCProxy`. This wrapper element contains in essence a description of the service. In one variation this description can be seen as a set of criteria for lookup of the actual WSP. These criteria consist of a reference to the *contract* (also known as *service type*) of the WSP and a reference to the federated user. The application gets this user reference from the HTTP request (the filter will have added this to the request). The second variation of the service description is a complete, but encoded, description that was obtained earlier. In either case the application finally sends the XML in an HTTP POST to the `WSCProxy`.

When the WSCProxy receives this XML, the WSCProxy searches for the specified service for the user, and upon success, sends the service-specific message as a properly enveloped ID-WSF request to the found WSP. Once the WSCProxy receives a response from the WSP, it responds to the application with the service-specific part of the message.

An application may also ask the WSCProxy for a storable service description. In this case, when the WSCProxy searches for the service, it responds to the application with an XML description of the found service. The application may store this description and use it for future invocations of the service.

See filter-spwsc-sample on page 26 for a description of how to use the WSCProxy in combination with the Select Federation Filters.

SP WSC API

The Select Federation SP WSC API is the interface that Java-based (Service Provider) web applications use to invoke identity-based web services associated with federation sessions. To use this API, the web application must be deployed in the same application server as Select Federation and initialized with the tfsconfig.properties configuration file.

The following sections show two examples of using the SPWSCAPI (the code is from the <*cd-base-directory*>/web-services/api/samples/spwsc-sample included on the CD):

- Example 1: Instantiate and Use SPWSCAPI to Call a Sample Web Service
- Example 2: Use SPWSCAPI to Handle ID-WSF User Interaction Requests

Example 1: Instantiate and Use SPWSCAPI to Call a Sample Web Service

This example shows how to instantiate and use the SPWSCAPI to call a sample web service:

```
    // Construct request body xml
     String req = "<SomeRequest xmlns=\"http://schemas.example.com/
sampleservice\" />";
     // Prepare to receive a successful result or an error
     String res = null;
     String err = null;
     try {
         // Instantiate the SPWSCAPI using the path to tfsconfig.properties,
         // the type of the service to call, and the federated session to use
         // in locating the service.
         // If the service can't be located, an exception will be thrown.
           SPWSCAPI spwscAPI = new SPWSCAPI(propfile, "http://
schemas.example.com/sampleservice", tfsSessionInfo);
         // Call the service and save the result
           res = spwscAPI.send(req);
     } catch (Exception e) {
         // Save the error description
         err = e.getMessage();
     }
```

Example 2: Use SPWSCAPI to Handle ID-WSF User Interaction Requests

This is a more complex example, which shows how ID-WSF user interaction requests can be handled:

```
   // Construct request body xml
     String req = "<SomeOtherRequest xmlns=\"http://schemas.example.com/
sampleservice\" />";
   // Prepare to receive a successful result or an error
     String res = null;
     String err = null;
     try {
         // Prepare to instantiate the SPWSCAPI
         SPWSCAPI spwscAPI;
         // First check to see if we have a saved reference in this session
         ServiceDescription serviceRef = (ServiceDescription)session.
getAttribute("serviceRef");
         if (serviceRef != null) {
            // Instantiate the SPWSCAPI using the path to tfsconfig.properties
             // and a saved reference
             spwscAPI = new SPWSCAPI(propfile, serviceRef);
         } else {
             // Instantiate the SPWSCAPI using the path to
             // tfsconfig.properties,
             // the type of the service to call, and the federated session to
             // use in locating the service.
             // If the service can't be located, an exception will be thrown.
             spwscAPI = new SPWSCAPI(propfile, "http://schemas.example.com/
sampleservice", tfsSessionInfo);
             // Save a reference for future use
             session.setAttribute("serviceRef", spwscAPI.getReference());
         }
         // Indicate that we are prepared to handle user agent redirects
         spwscAPI.setAllowUIRedirect(true);
         // Check to see if we have a previous call to retry (in which case
         // we are returning from handling a user agent redirect)
         String retryRefToMessageId =
(String)request.getParameter("RetryRefToMsgId");
         if (retryRefToMessageId != null)
             // Indicate that we are retrying a previous call
             spwscAPI.setRetry(retryRefToMessageId);
         // Call the service and save the result
         res = spwscAPI.send(req);
     } catch (TFSUIRedirectException e) {
         // Construct a return url that will trigger us to retry this call
         String returnToURL = request.getRequestURL().toString() +
"?RetryRefToMsgId=" + e.getRetryRefToMessageId();
         // Redirect to the requested URL, passing in our returnURL
         e.sendRedirect(response, returnToURL);
         return;
     } catch (Exception e) {
         // Save the error description
         err = e.getMessage();
     }
```

For more information, see the following:

- Chapter 4, “Select Federation API Overview” in the *HP OpenView Select Federation Web Application Developer’s Guide* for an overview of the Select Federation SP APIs.
- `<cd-base-directory>/docs/api/index.html` file on the Select Federation SDK CD for detailed API documentation.
- spwsc-sample on page 27, for a description of how to use the SPWSCAPI in combination with the SPAPI or J2EE Access Filter.

3 Developing an Identity-Based Web Service

This chapter describes how to develop new identity-based web services. When these services are deployed, HP OpenView Select Federation exposes these services in accordance with the Liberty Alliance ID-WSF specifications.

The development of a new service consists of the following steps, which are described in detail in each step:

- Step 1: Define the Functionality of the Service
- Step 2: Define the XML Interface of the Service
- Step 3: Develop and Deploy the Service
- Step 4: Register the Service

Step 1: Define the Functionality of the Service

This step is not specific to HP OpenView Select Federation. Most importantly this step is to ensure that the service offers some clearly defined functionality that can be used in a variety of settings. A typical mistake is to develop the service around a single usage scenario. It is good to work from one or more usage scenarios, but it is important to find a small common set of general functions that can be used in all known, and in many unknown, scenarios. At the same time avoid defining a service that tries to work for too many scenarios. For example, a `geolocation` service should offer location information, not other personal information.

An important consideration for ID-WSF based services is the relationship of the service and identity. Broadly, services can be divided into ***identity providing*** and **identity consuming** services.

- Identity providing services are those that inform service consumers about some identity (such as a person). In general, such services can be thought of as `my... service` such as `myCalendar`, `myWallet`, and so on. An identity providing service can only be found through a specific SIF. For example, prospective web service consumers find such services through the ID-WSF Discovery Service (which is an identity providing service).
- Identity consuming services can be thought of as services that need to get an identity in service requests. The service may need to be able to operate on behalf of the identity to do accounting, and so on. Typically, the service consumer is pre-configured with the address (and policy) of such a service. An instant messaging service is perhaps an example of such an identity consuming service. The service expects some identity in requests but does not really inform service consumers about end-users (whereas a Presence service would).

Step 2: Define the XML Interface of the Service

It is recommended that you define the interface of the service at the XML level. In general we advise against using WSDL/schema generators. Many developers may not have access to high quality WSDL compilers, and/or need to work within constrained environments. Simple, well structured XML eases the work of these developers.

It is best to start with example request and response messages. Be sure that these have the correct information as well as an acceptable structure to generalize the examples into an XML schema. Another consideration is to think about the likely data model on the service consumer side. If it can be expected that the service consumer maintains some sort of database it may be useful to define the interface of the service such that it returns messages that can effectively be used to update that database.

It is important to define the XML interface in terms of the "abstract WSDL", that is as the content of `soap:Body` elements. This way the interface does not change if and when the service is deployed using a framework other then ID-WSF. This is also important if the service will be used by applications that use the Select Federation IIS or Apache filters. Unfortunately ID-WSF ResourceIDs break this nice separation. However, note that the Select Federation Web Service Java APIs hide this complexity, but that is based on a convention that a ResourceID, if present, will be the first child element of the first element in the `soap:Body`.

It is worth considering the nature of the functionality that the service will offer. If the service is mainly about obtaining user information it may be worthwhile to define the XML interface according to the ID-WSF *Data Service Template* specification. The advantage of defining a DST-based service is that the specification work can be minimal. A DST-based specification essentially defines the schema of a virtual XML document that the service will "expose". The schema should have a namespace, for example:

http://wsp.company.com/geoloc/1.0

The DST-based specification should also define one or more `Select` statements that the service will support against the virtual XML document. For example, a simple geolocation service could support only one such statement, which gets the value of the Location element:

```
//geo:Location
```

Step 3: Develop and Deploy the Service

HP OpenView Select Federation 6.5 and its SDK offer two ways to develop and deploy new services:

- Configure additional attributes for a DST-based attribute service
- Implement the ServicePlugin interface of the SPWSPAPI

The following sections describe these methods.

Configure a New DST-Based Service

HP OpenView Select Federation Premium Edition has a built-in module that can offer user attributes through Liberty Alliance ID-WSF DST-based services. It is possible to add a new DST-based service by simply editing the system configuration `tfsconfig.properties` file and then allocating existing or new attributes to the new service. This method has the advantage of being very simple, but the restriction is that the new service is read-only. For details on editing the `tfsconfig.propertie` file, see Appendix A, "Configuration Parameters" in the *HP OpenView Select Federation Configuration and Administration Guide.*

To add a new DST service to an Authority, perform the following steps:

1 Declare a new service name and associate it with a namespace, which also serves as the service type. For example:

```
geo.dstNS=http://wsp.company.com/geoloc/1.0
```

2 Configure the new attributes that provide the actual data.

See Chapter 10, "Configuring Attributes" in the *HP OpenView Select Federation Configuration and Administration Guide* for details. For each new attribute, the service name and `Select` statement must be declared. For example:

```
location.dstSvc=geo
location.dstSelect=/geo:Location
```

In addition, the Authority Select Federation installation must be able to find the actual attribute values. Therefore, the attribute needs to be configured with the information required by the Directory Plugin that will provide the attribute. See the "IDP-SampleDirPlugin" sample description in the *HP OpenView Select Federation Web Application Developer's Guide.*

3 Set which attributes are allowed to be queried by the Applications in the admin console of the Authority.

If the Application is a Select Federation installation, it is possible to add the same service and attributes there, and then you can set which attributes are to be queried from the Authority in the admin console. Alternatively, the application may make use of the `DSTAPI`, the `WSCAPI` or the `WSCProxy`.

Implement the ServicePlugin Interface

If the previous method is too restrictive, you need to develop new code to deal with service requests. The best way to do this is to implement the ServicePlugin interface in the SPWSPAPI and deploy your service through the Select Federation `tfs-wsp.war` file. This is an additional `war` file, which needs to be added to your Select Federation Premium Edition installation. See How to Install the tfs-wsp.war File in Select Federation Premium Edition on page 19 for instructions.

A ServicePlugin is an implementation of a web service that is largely independent of the service invocation framework that is used to discover and invoke web services. A ServicePlugin is plugged into a container (`tfs-wsp.war`) that takes care of security, authentication, identity, and so on. The container processes incoming (SOAP) messages and prepares a ServiceRequest that contains the service specific message, such as the content of the `soap:Body`, as sent by the service consumer. In addition the ServiceRequest contains information about that consumer as well as the target user on whose behalf the ServicePlugin is expected to serve the request.

A ServicePlugin may require consent or other information from the end-user before being able to serve a request. To this end the plugin can construct and throw an InteractionRequest. Any user response to such a request arrives in a new ServiceRequest that contains UserInput.

The following section shows a simple ServicePlugin implementation.

Example of a Simple Service Plugin Implementation Using the SPWSPAPI

This example shows how to use the SPWSPAPI to implement a service plugin (this code is from the `<`*`cd-base-directory`*`>/web-services/api/samples/spwsp-sample` included on the CD).

```
public class SampleService implements ServicePlugin {

  public SampleService(Config conf) {
         // Nothing to configure for this service
     }

     public Map initService(Map containerConfig) throws ServiceException {
         Map serviceConfig = new HashMap();
         // Indicate that this service requires a target user in order
         // to process a request. In ID-WSF, this corresponds to having
         // a ResourceID that maps to a valid local user
         serviceConfig.put("requiresUser", "1");
         serviceConfig.put("userAttributes", "name_firstname");
         return serviceConfig;
     }

     public ServiceResponse getResponseFor(ServiceRequest request) throws
Servic\eException {
         // Get the target user for this request
         ServiceUser user = request.getUser();
         if (user == null)
             throw new ServiceException("no user");
         String userId = user.getLocalUserId();
         Map profile = user.getProfile();

         // Get the target user's first name (default to user id)
         String name = (String)profile.get("name_firstname");
         if (name == null)
             name = userId;

         // Dispatch based on the request element
         String req = request.getBodyElement().getLocalName();
         if ("SomeRequest".equals(req)) {

             // Return a simple response based on the identity of the
             // target user. In a real service, this would perform some
             // action associated with the user.
             return request.newResponse("<SomeResponse xmlns=\"http://
schemas.example.com/sampleservice\">this is " + name + "'s sample
service</SomeResponse>");
         } if ("SomeOtherRequest".equals(req)) {

             // Demonstrate the use of user interaction in handling a
             // request.

             // First, check to see if we have received the user input
             // that we asked for.
             UserInput userInput = request.getUserInput();
             if (userInput == null) {
                 // If not, construct and throw an interaction request
                 InteractionRequest ir;
                 try {
                     // Construct a simple interaction request that asks
                     // a simple yes/no question
                     ir = InteractionRequest.confirm("answer", "What is your
answer?", false);
                 } catch (TFSException e) {
                   throw new ServiceException("error constructing interaction
request");
                 }
                 throw ir;
             }
             // Return a simple response based on the identity of the
             // target user and their response to the interaction request.
             // In a real service, this would perform some action associated
             // with the user.
             String answer = userInput.getParameter("answer");
             return request.newResponse("<SomeOtherResponse
xmlns=\"http://schemas.example.com/sampleservice\">" + name + " says the
answer is " + answer + "</SomeOtherResponse>");

         } else {
             throw new ServiceException("unknown request " + req);
         }
     }
```

How to Install the tfs-wsp.war File in Select Federation Premium Edition

Perform the following steps to install the `<`*`cd-base-directory`*`>/filters/support/tfs-ws.war` file:

1 Copy the files in the `<`*`cd-base-directory`*`>/web-services/hpsf-pe-additions/stylesheets/` directory to the stylesheets sub-folder of the configuration folder for your Select Federation instance, such as `<`*`SF-installation-dir`*`>/conf/stylesheets.`

The new `tfs-fs.war` file requires these additional stylesheets to support user interaction during web service calls.

2 Deploy the `<`*`cd-base-directory`*`>/web-services/hpsf-pe-additions/tfs-fs.war` file to your Application Server that hosts your Select Federation `war` files.

- If your Select Federation install uses the built-in application server, deploy the `tfs-fs.war` file by placing it in the `<`*`SF-installation-dir`*`>/webapps/` directory.
- If your Select Federation install uses WebLogic or WebSphere, deploy the `tfs-fs.war` file through the administrative console. See the respective application server documentation for details.

You are now ready to register new services (see Step 4: Register the Service on page 20 for instructions).

For more information, see the following:

- Chapter 4, “Select Federation API Overview” in the *HP OpenView Select Federation Web Application Developer’s Guide* for an overview of the Select Federation SP APIs.
- `<cd-base-directory>/docs/api/index.html` file on the Select Federation SDK CD for detailed API documentation.

- spwsp-sample on page 28, for a description of how to use the SPWSPAPI in combination with the SPAPI or J2EE Access Filter.

Step 4: Register the Service

For each user, the service (if “identity providing”) should be registered with the Discovery Service. In the case where the service is deployed on a Select Federation installation and acts as the Authority for the user, you can register the service by adding the new service to the system configuration. This will virtually register the service for each user.

If the installation is an Application that relies on other Authorities for authentication, registration of the new service should be done through using the SPWSPAPI within a page that the user will visit. For example, the user might visit the SP side of the provider that acts as the WSP.

Registration as an IDP Service

Select Federation has a concept of IDP-hosted services, which are services that are advertised for all of an IDP's users without having been explicitly registered with the built-in Discovery Service. New services can be added as IDP services by adding the following type of structure to the `tfsconfig.properties` file on an IDP:

```
#space separated shorthand for local services exposed by this IDP

idpServices=sample

sample.class=SampleService
sample.jar=/hpsf-sdk-cd/web-services/samples/spwsp-sample/dist/
sampleservice.jar
sample.roType=http://schemas.example.com/sampleservice
sample.roURL=https://youridp.com/tfs-wsp/SPWSP_IDWSF11/sample
```

This structure provides one step to both deploy and register the service for all users. When deployed this way, no registration with the DS is needed. The service is available for all users and all federated sites (WSCs).

Notice that the URL is constructed by appending the path of the deployed `tfs-wsp.war` file with `/SPWSP_IDWSF11/` and then the service alias is used in the configuration entries.

Registration as an SP Service

Registering an SP service requires the following two steps:

1. Deploy the SP service by adding the following entries to the `tfsconfig.properties` file.

   ```
   spwspServices=sample

   sample.class=SampleService
   sample.jar=/hpsf-sdk-cd/web-services/samples/spwsp-sample/dist/
   sampleservice.jar
   sample.roType=http://schemas.example.com/sampleservice
   ```

2 Register the SP service one user at a time using the SPWSPAPI, described in the next section Registration with the SPWSPAPI.

Registration with the SPWSPAPI

Following is a code snippet from the `index.jsp` page of `sp-sample` that was enhanced to enable registration:

```
String spwsppropfile =
                application.getInitParameter("com.trustgenix.tfs.propFile");
// Instantiate the SPWSPAPI
SPWSPAPI spwspAPI = new SPWSPAPI(tfspropfile, tfsSessionInfo);

String action = request.getParameter("action");
if (action != null && !action.equals("")) {
                // an alternative is to register the service during activation
                if (action.equals("reg")) {
                  spwspAPI.register("sample");
                } else if (action.equals("dereg")) {
                  spwspAPI.deregister("sample");
                }
                response.sendRedirect("index.jsp");
                return;
}
```

The application using SPWSPAPI must be deployed on the server as the Select Federation hosting the service, and the API must be initialized with the server's `tfsconfig.properties` file.

The Authority needs an entry in its `tfsconfig.properties` file that lists those partners that are allowed to register entries in its Discovery Service. For example:

```
#To allow updates to the DS from SPs
#idwsfDSAllowUpdatesFrom=providerid1 providerid2 ...
idwsfDSAllowUpdatesFrom=http://company.com:8080/tfs
```

Other, but not required configuration parameters for the Authority include the following:

```
# enable DS
idwsfSupportDS=1
#DS generated tokens expire after 30 mins
idwsfDSTokenTimeout=30m
```

Note that registration may be triggered by an explicit user action or it might happen silently and automatically during an activation procedure. Registration happens within the context of a federation with respect to a particular Authority, which knows about a particular Discovery Service.

4 Support for LUAD-WSC Implementations

A LUAD-WSC is a Liberty-enabled User-Agent or Device that acts as a WSC. As a LUAD-WSC is not an Application that acts as a partner known to the Authority (or IDP) it uses a slightly different service invocation flow. An HP Select Federation is never a LUAD, but an Authority installation can serve LUAD-WSCs. This requires additional configuration that is described in this chapter.

LUAD-WSC implementations must receive information about the DS from an Authentication Service. HP OpenView Select Federation offers this service. By default the Authentication Service is not exposed. But, you can enable the Authentication Service in the `tfsconfig.properties` file, which exposes the Authentication Service on a URL. For example:

http://company.com/tfs/IDPSSO_IDWSF10

The ID-WSF Authentication Service specification defines the use of SASL authentication mechanisms to actually authenticate the LUAD. HP Select Federation offers an IDPSASLAuthnPlugin interface and two built-in implementations of it. The IDPSASLAuthnPlugin**_Dir** is simpler and does not require configuration beyond declaring the plugin. The IDPSASLAuthnPlugin**_File** supports mechanism selection and password transformation.

Following is a commented section of the `tfsconfig.properties` file that controls the behavior:

```
# enable and configure the AS
# (this assumes that you have configured a dirPlugin)
idwsfSupportAS=1
idpSASLAuthnPlugin=com.trustgenix.tfsIDP.util.IDPSASLAuthnPlugin_Dir


# AS tokens expire after 30 minutes
authTimeout=30m


## The various settings below require that the IDPSASLAuthnPlugin_File is used
## like this:
#idpSASLAuthnPlugin=com.trustgenix.tfsIDP.util.IDPSASLAuthnPlugin_File
#IDPSASLAuthnPlugin_File.acctFilePath=properties/users.properties


## So everything from here on requires the _File plugin!
# this line defines the SASL mechanisms that the server will choose.
# the server chooses the first one out of this list that the client supports
# if you only want CRAM-MD5 then simply make it the only entry in the list
here.
#IDPSASLAuthnPlugin_File.initialMechs=PLAIN CRAM-MD5


# this line defines the password transforms that the service
# will ask the client to perform
#IDPSASLAuthnPlugin_File.passwordTransforms=lc san uc t8
```

```
# the particular set defined here (and below) tests all transforms
# if e.g. the password entered by the user on the device is 'AB34**cö90defG'
# as after all transforms you should get: 'AB34C90D'

# the following entries define the actual transforms
# each transform is a single line of XML
#passwordTransform.t8=<sa:Transform xmlns:sa="urn:liberty:sa:2004-04"
name="urn:liberty:sa:pw:truncate"><sa:Parameter
name="length">8</sa:Parameter></sa:Transform>

#passwordTransform.lc=<sa:Transform xmlns:sa="urn:liberty:sa:2004-04"
name="urn:liberty:sa:pw:lowercase" />

#passwordTransform.uc=<sa:Transform xmlns:sa="urn:liberty:sa:2004-04"
name="urn:liberty:sa:pw:uppercase" />

#passwordTransform.san=<sa:Transform xmlns:sa="urn:liberty:sa:2004-04"
name="urn:liberty:sa:pw:select"><sa:Parameter
name="allowed">0123456789abcdefghijklmnopqrstyvwxyz</sa:Parameter></sa:
Transform>
```

When you use the IDPSASLAuthnPlugin_Dir plugin, the normal directory plugin authenticates the LUAD.

5 Samples

This chapter provides descriptions of the web services samples provided on the HP OpenView Select Federation CD, to help you understand the capabilities of Select Federation.

Samples List

The web services samples included on the Select Federation CD are in the `<cd-base-directory>/web-services/filters/samples/` and the `<cd-base-directory>/web-services/api/samples/` directories.

The following web services samples are included on the CD:

- **filter-spwsc-sample**: Demonstrates how to use the WSCProxy in combination with the Select Federation Filters.
- **spwsc-sample**: Demonstrates how to use the SPWSCAPI in combination with the SPAPI or J2EE Access Filter.
- **spwsp-sample**: Demonstrates how to use the SPWSPAPI in combination with the SPAPI or J2EE Access Filter.

Building the Samples

You can build the samples by copying them from the Select Federation CD to a location on your hard disk.

Required Software

- **Ant**: Ant tool from the apache Jakarta project, version 1.5 is desirable, though earlier versions may also work.
- **JDK**: JDK version 1.4.2 or later.
- **J2EE Servlet Engine**: Since all the samples involve servlets or JSPs, you need a J2EE servlet engine (such as Tomcat, IBM WebSphere, BEA WebLogic, and so on). The sample applications can reside on the same server as Select Federation.

Build Process

To build a sample, change your current working directory to the top-level directory of the sample. For example:

```
cd samples/idp-authnplugin
```

At the top level directory enter the following command:

```
ant clean package
```

The output of the compilation command appears in the `dist` directory.

Samples Descriptions

filter-spwsc-sample

This sample demonstrates the use of the WSCProxy from an application protected by a Select Federation Filter, with examples for both the IIS and Apache filters.

Sources

`spwsc-sample.php`: This demonstrates a simple web service invocation.

> This sample assumes knowledge of writing PHP scripts.

Running the filter-spwsc-sample Sample

> To run the `filter-spwsc-sample`, you must deploy it on a partner installation.

To run the `filter-spwsc-sample` sample, perform the following steps:

1. Ensure the following:
   - Select Federation installation supports filters.
   - There is a directory on the web server that is properly protected by one of the filters. See "How to Configure the IIS Filter" and "How to Configure the Apache Filter" in the *HP OpenView Select Federation Web Application Developer's Guide* for information on protecting directories.
   - Deploy the sample Web Service Provider (WSP) on another system.
2. Deploy the `spwsc-sample.php` file in a directory, which is enabled for PHP scripts and is protected by a Select Federation filter.

   Ensure that the Web Service Consumer (WSC) and WSP Select Federation installations are set up as partners for each other.
3. Point your web browser to the location where you deployed the `spwsc-sample.php` file.
4. Navigate to the `spwsc-sample.php` page.

   This page displays a login prompt and links for logging in through the configured IDPs. If you do not see any links to login through the IDPs, you have not configured Authority sites in your circle of trust.
5. Click on the login through the IDP link to navigate to the IDP.
6. Login as user on this page.

   You are redirected back to the SP. The index page opens with two samples called **Example 1** and **Example 2**.

7 Click on **Example 1** or **Example 2**.

For a better understanding, it is recommended to take a trace of the network traffic around the Select Federation installation of the WSCProxy.

spwsc-sample

This sample demonstrates the use of the SPWSCAPI in a simple application scenario. It uses the SPAPI to authenticate a user and then the SPWSCAPI to invoke a sample web service for the authenticated user.

Sources

- **`web/index.jsp`**: This is the index page and the login page.
- **web/example1.jsp**: This demonstrates a simple web service invocation, not handling user interaction.
- **web/example2.jsp**: This demonstrates a more complex web service invocation that handles user interaction exceptions and retries the operation.

Running the spwsc-sample Sample

➤ To run the `spwsc-sample`, you must deploy it on a partner installation.

To run the `spwc-sample` sample, perform the following steps:

1 Deploy the file `dist/spwsc-sample.war` to your J2EE server.

2 Navigate to the `index.jsp` page.

This page displays a login prompt and links for logging in through the configured IDPs. If you do not see any links to login through the IDPs, you have not configured Authority sites in your circle of trust.

3 Click on the login through the IDP link to navigate to the IDP.

4 Login as user on this page.

You are redirected back to the SP. If the user logged in at the IDP, but is not federated with the SP, you will be navigated to the activation page.

5 On the activation page, you can do one of the following:

- Associate the user with a local account.
- Assign a new user ID to the user at the SP site.

The index page opens with two samples called **Example 1** and **Example 2**.

6 Invoke the sample web service using **Example 1** or **Example 2**:

- Click on **Example 1** to test a simple web service invocation. On the result page, click on **back** to return to the index page.
- Click on **Example 2** to test a more complex web service invocation involving user interaction. Click **yes** or **no** when prompted for user interaction. On the result page, click **back** to return to the index page.

spwsp-sample

This sample demonstrates the use of the ServicePlugin interface and the SPWSPAPI in a simple web service scenario. It uses the ServicePlugin interface to implement a web service and then, optionally, the SPAPI to authenticate a user and the SPWSPAPI to register the service for the authenticated user.

➤ If the sample is deployed on an IDP then the registration application is not needed (the service can be implicitly registered for all users). If it is deployed on an SP then the registration application is needed.

Sources

- **`src/SampleService.java`**: The ServicePlugin implementation for the sample service.
- **`web/index.jsp`**: This is the optional login and registration page.

Running the spwsp-sample Sample on an IDP

To run the `spwsp-sample` sample, perform the following steps:

1. Add the following lines to the `tfsconfig.properties` file of the Select Federation IDP on which the sample service is to be deployed (adjusting the path to the `sampleservice.jar` file and the URL to your Select Federation installation):

   ```
   idpServices=sample

   sample.class=SampleService

   sample.jar=/hpsf-sdk-cd/web-services/samples/spwsp-sample/dist/
   sampleservice.jar

   sample.roType=http://schemas.example.com/sampleservice

   sample.roURL=https://youridp.com/tfs-wsp/SPWSP_IDWSF11/sample
   ```

   There is no need to deploy the `spwsp-sample.war` file in this case, as the service is automatically registered for all users of your IDP.

2. Follow the instructions for Running the spwsc-sample Sample on page 27, or Running the filter-spwsc-sample Sample on page 26 to invoke the service.

Running the spwsp-sample Sample on an SP

To run the spwsp-sample on an SP, perform the following steps:

1. Add the following lines to the `tfsconfig.properties` file of the Select Federation SP on which the sample service is to be deployed (adjusting the path to the `sampleservice.jar`):

   ```
   spwspServices=sample

   sample.class=SampleService

   sample.jar=/hpsf-sdk-cd/web-services/samples/spwsp-sample/dist/
   sampleservice.jar

   sample.roType=http://schemas.example.com/sampleservice
   ```

2. Deploy the `spwsp-sample.war` file to load the `index.jsp` page in a web browser.
3. Deploy the file `dist/spwsp-sample.war` to your J2EE server.
4. Navigate to the `index.jsp` page.

   This page displays a login prompt and links for logging in through the configured IDPs. If you do not see any links to login through IDPs, it means you have not configured Authority sites in your circle of trust.
5. Click on the login through the IDP link to navigate to the IDP.
6. Login as user on this page.

   You will be redirected back to the SP. If the user logged in at the IDP, but is not federated with the SP, you will be navigated to the activation page.
7. On the activation page, you can do one of the following:
   - Associate the user with a local account
   - Assign a new user ID to the user at the SP site.

   You will then be navigated to the index page.
8. On the index page, you can register and de-register the sample web service.
9. Follow the instructions for running the `spwsc-sample` (or `filter-spwsc-sample`) to invoke the service after registering the sample service.

Index