PingFederate®

Java Integration Kit

Version 2.5.1

User Guide

PingIdentity®

PingFederate Java Integration Kit *User Guide*
Version 2.5.1
December, 2012

Ping Identity Corporation
1001 17th Street, Suite 100
Denver, CO 80202
U.S.A.

Phone: 877.898.2905 (+1 303.468.2882 outside North America)
Fax: 303.468.2909
Web Site: www.pingidentity.com

Trademarks

Disclaimer

Document Lifetime

Ping Identity may occasionally update online documentation between releases of the related software. Consequently, if this PDF was not downloaded recently, it may not contain the most up-to-date information. Please refer to documentation.pingidentity.com for the most current information.

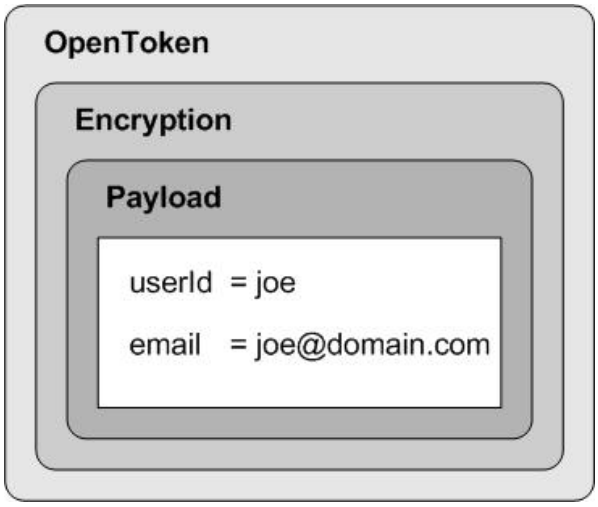
From the Web site, you may also download and refresh this PDF if it has been updated, as indicated by a change in this date: **December 3, 2012**

Contents

Introduction

The Java Integration Kit includes the OpenToken Adapter and a Java agent, which allows developers to integrate their Java applications with a PingFederate server acting as either an Identity Provider (IdP) or a Service Provider (SP). The kit allows an IdP server to receive user attributes from a Java IdP application. On the SP side, the kit allows a Java SP application to receive user attributes from the SP server.

The Java Integration Kit uses an open-standard, secure token called OpenToken to pass user information between an application and PingFederate. The OpenToken is passed through the user's browser as a URL query parameter, an HTTP cookie, or a form POST. The data within the OpenToken is a set of key/value pairs, and the data is encrypted using common encryption algorithms, as illustrated below:



The Integration Kit distribution also contains sample IdP and SP applications. The applications may be installed quickly for testing OpenToken processing and to provide a working demonstration of end-to-end single sign-on (SSO) and single logout (SLO). Source code and supporting files are included in the distribution and may be modified or used as a reference for application developers.

Intended Audience

This document is intended for PingFederate administrators and Web-application developers who will customize one or more Java applications to communicate with PingFederate.

We recommend that you review the PingFederate *Administrator's Manual*—specifically the information on adapters and integration kits. You should have an understanding of how PingFederate uses adapters and how they are configured. After initial installation steps are followed in this Guide, it would also be helpful to complete the tasks in the Java Sample Application Startup Guide to have a working example of a completed configuration.

ZIP Manifest

The distribution ZIP file for the Java Integration Kit contains the following:

- `ReadMeFirst.pdf` – contains links to this online documentation
- `/legal` – contains this document:

  - `Legal.pdf` – copyright and license information
- `/dist` – contains libraries needed to run the adapter:
  - `opentoken-adapter-2.5.1.jar` – OpenToken Adapter JAR file
  - `opentoken-agent-2.5.1.jar` – OpenToken Agent JAR file
  - `commons-collections-3.2.jar` – Apache Commons Collections library

Note: Version 3.2 or higher of the Apache Commons Collection is required and provided as a convenience if not already installed.

  - `commons-beanutils.jar` – Apache Commons Bean Utility library
  - `commons-logging.jar` – Apache Commons Logging library

Note: The last two libraries, `beanutils` and `logging`, are also provided as a convenience and should be installed only if they are not already contained in the application `CLASSPATH`.

- `/docs`
  - `/javavdoc` – Agent Toolkit API Javadoc
- `/sample`
  - `data.zip` – pre-configured PingFederate configuration archive for the sample applications
  - `/IdpSample.war` – extracted WAR directory for the IdP sample application
  - `/SpSample.war` – extracted WAR directory for the SP sample application
- `/sample_src`
  - `/SpSample` – source code and supporting files for the SP sample application
  - `/IdpSample` – source code and supporting files for the IdP sample application

System Requirements

The following software must be installed in order to implement the Java Integration Kit:

- PingFederate 5.x (or higher)
- J2SE Java Runtime Environment 1.4 or later on the agent side

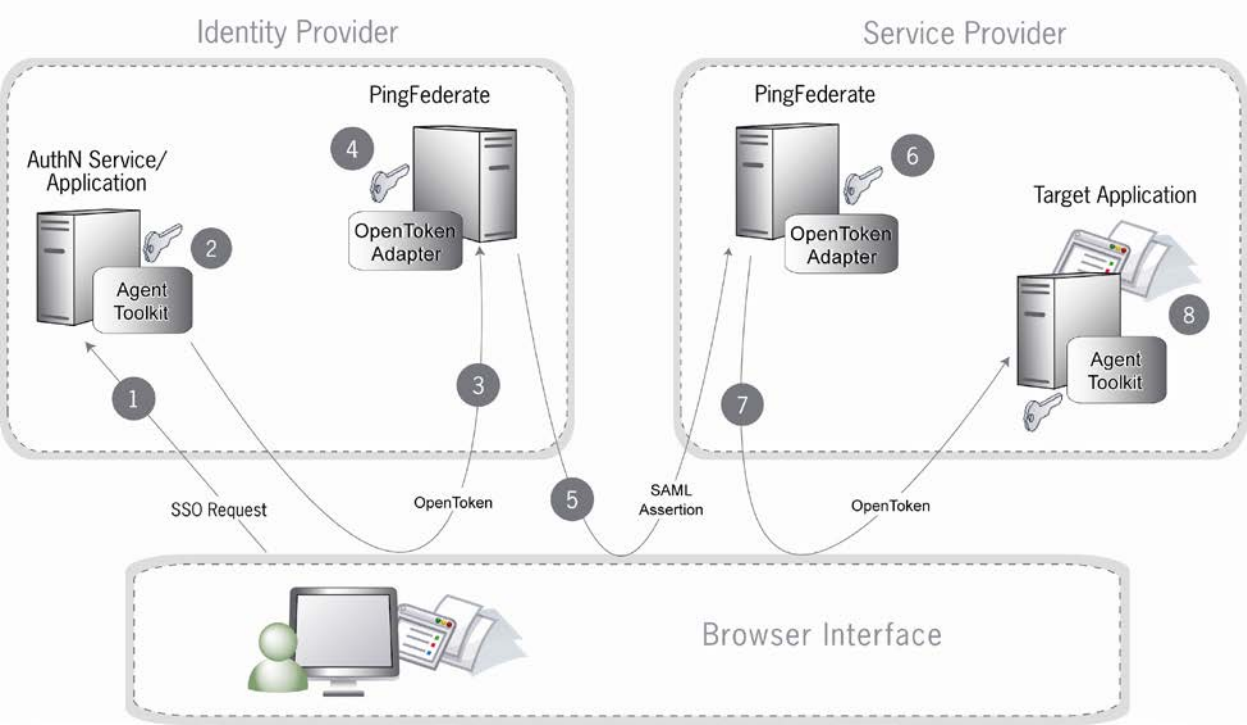
  To use strong Advanced Encryption Standard (AES) encryption with a key size of more than 128 bits, the *Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files* must be installed in your JDK.

Processing Overview

The Java Integration Kit consists of two parts:

- The OpenToken Adapter, which runs within the PingFederate server
- The Agent Toolkit for Java, which resides within the Java application

The following figure shows a basic IdP-initiated SSO scenario in which PingFederate federation servers using the Java Integration Kit exist on both sides of the identity federation:



Sequence

1. A user initiates an SSO transaction.
2. The IdP application inserts user attributes into the Agent Toolkit for Java, which encrypts the data internally and generates an `OpenToken`.
3. A request containing the `OpenToken` is redirected to the PingFederate IdP server.
4. The server invokes the OpenToken IdP Adapter, which retrieves the `OpenToken`, decrypts, parses, and passes the user attributes to the PingFederate IdP server. The PingFederate IdP server then generates a Security Assertion Markup Language (SAML) assertion.
5. The SAML assertion is sent to the SP site.
6. The PingFederate SP server parses the SAML assertion and passes the user attributes to the OpenToken SP Adapter. The Adapter encrypts the data internally and generates an `OpenToken`.
7. A request containing the `OpenToken` is redirected to the SP application.
8. The Agent Toolkit for Java decrypts and parses the `OpenToken` and makes the user attributes available to the SP Application.

Note: PingFederate can be configured to look up additional attributes from either an IdP or SP data store. See Data Stores in the PingFederate *Administrator's Manual* for more information.

Installation and Setup

The following section describes how to install and configure the OpenToken Adapter for both an IdP and an SP as well as deploy the Java agent.

Installing the OpenToken Adapter and Configuring PingFederate

Note: If you have already deployed version 2.5.1 (or higher) of the OpenToken Adapter skip steps 1 through 4 in the following procedure.

1. Stop the PingFederate server if it is running.
2. Remove any existing OpenToken Adapter files (`opentoken*.jar`) from the directory:

   `<PF_install>\pingfederate\server\default\deploy`

   The adapter JAR file is `opentoken-adapter-<version>.jar`.

   Note: If the adapter JAR filename indicates version 2.1 or less, also delete the supporting library `opentoken-java-1.x.jar` from the same directory.

   Important: If you are running PingFederate 5.1.0, delete the file `opentoken-adapter.jar` from the directory: `<PF_install>/pingfederate/server/default/lib`.
3. Unzip the integration-kit distribution file and copy `opentoken-adapter-2.5.1.jar` from the `/dist` directory to the PingFederate directory:

   `<PF_install>\pingfederate\server\default\deploy`
4. Start or restart the PingFederate server.
5. Configure an instance of the OpenToken Adapter.

   Tip: You may skip this and subsequent steps in this setup if you want to install and deploy the sample applications first, before configuring the Adapter instance for your own application. The sample distribution (in the sample directory) contains a configuration archive that includes preconfigured OpenToken Adapter instances for both the IdP and SP sample applications.

   For detailed instructions, see OpenToken Adapter Configuration in the PingFederate *Administrator's Manual*.

   On the Actions screen in the adapter setup, click the **Invoke Download** link and then click **Export** to download the `agent-config.txt` properties to a directory that is readable by the Agent Toolkit for Java.
6. Once the adapter is configured, create a connection to your partner using that adapter instance. (For more information, see Identity Provider SSO Configuration or Service Provider SSO Configuration in the PingFederate *Administrator's Manual*.)

Deploying the Java Agent

Note: If this is a first-time installation of the Java Integration Kit, proceed directly to step 2 in the following procedure.

If you are upgrading this integration, we strongly recommend reinstalling the OpenToken Agent in all existing applications (IdP or SP).

1. If you are upgrading this integration:
   a. Temporarily stop your Web application if it is running.
   b. Remove the existing OpenToken Agent file (`opentoken-agent-x.x.x.jar`) from wherever it currently exists within the `CLASSPATH`. The file is typically deployed within an application's `/lib` directory.

Note: No code changes are required within applications when upgrading.

2. From the integration-kit `/dist` directory, copy the `opentoken-agent-2.5.1.jar` into a location in the `CLASSPATH` of the Java application.
3. Ensure the following Apache Commons libraries are also available in the `CLASSPATH` of the Java application:
   - `commons-collections-3.2.jar`

Note: This version or higher is required.

   - `commons-beanutils.jar`
   - `commons-logging.jar`

   If the libraries are not already installed, you can add them from the integration-kit `/dist` directory.
4. If you are upgrading, restart your Web application.
5. Repeat these steps as needed for additional custom applications.

   For a first-time installation, complete the integration as described in Instantiating the Agent API (next) and in either Integrating with an IdP PingFederate Server on page 9 or Integrating with an SP PingFederate Server on page 12.

Instantiating the Agent API

Use the Agent API to read and write an `OpenToken`. The API provides access to functionality for writing an `OpenToken` to a given HTTP response and reading an `OpenToken` from a given HTTP request.

Sample Code

Instantiating the agent object is done simply by invoking a constructor and loading the configuration, as in the example below:

```
import java.io.IOException;
```

```
import java.io.File;
import java.io.InputStream;
import java.util.Map;
import java.util.HashMap;
import java.util.List;

import org.apache.commons.collections.MultiMap;
import org.apache.commons.collections.map.MultiValueMap;

import com.pingidentity.opentoken.Agent;
import com.pingidentity.opentoken.TokenException;
import com.pingidentity.opentoken.util.UrlHelper;
. . . .
Agent agent = new Agent("<PATH_TO_FILE>/agent-config.txt");
```

When the agent object is instantiated, it uses the `agent-config.txt` file to find the configuration data generated when the PingFederate adapter was configured. This configuration data includes the name of the cookie that the agent object will write, as well as the key to use when encrypting a new `OpenToken`. If the `agent-config.txt` file is not found, the agent constructor will throw an exception.

Integrating with an IdP PingFederate Server

This section provides implementation guidelines and code examples for Java developers, covering the following types of IdP SAML 2.0 implementation profiles:

- IdP Single Sign-On (SSO)
- IdP Single Logout (SLO)

IdP Single Sign-On (SSO)

When PingFederate is configured as an IdP, it needs to be able to identify a user prior to issuing a SAML assertion for that user. When using the OpenToken Adapter with PingFederate, this means that the PingFederate server attempts to read a cookie or query parameter containing an `OpenToken` and then use the values within to identify the user. The application that starts the SSO must include an `OpenToken` so that PingFederate can identify the user. Use the Agent API to write an `OpenToken`. The API is a Java object that provides access to functionality for writing an `OpenToken` to a given HTTP response.

Sample Code

The `writeToken` method of the Agent class takes an `org.apache.commons.collections.MultiMap` collection of attributes and encodes them into an `OpenToken`, which is then written to the HTTP response.

> **Note:** The collection of attributes *must* contain a key named "subject" for a valid token to be generated.

If any errors are encountered while creating the token or writing the token out to the response, a `TokenException` is thrown.

The code snippet below demonstrates the use of the `writeToken` method:

```
String username = (String)request.getSession().getAttribute("username");
Map<String, String> userInfo = new HashMap<String, String>();
userInfo.put(Agent.TOKEN_SUBJECT, username);
String returnUrl = "https://<PingFederate-DNS>:9031" +
   request.getParameter("resume");
. . . .
try {
   UrlHelper urlHelper = new UrlHelper(returnUrl);
   //see "Using the Agent API" section for sample code
   //that instantiates and configures an Agent instance
   agent.writeToken(userInfo,response,urlHelper,false);
   returnUrl = (String)urlHelper.toString();
}
catch(TokenException e) {
// Handle exception
}
```

Passing Multi-Value Attributes

The Agent Toolkit for Java supports passing multi-value attributes to PingFederate that will each appear in its own discrete `<AttributeValue>` element in the SAML 2.0 assertion. Multi-value attributes are passed using the `org.apache.commons.collections.map.MultiValueMap` collection.

Sample Code

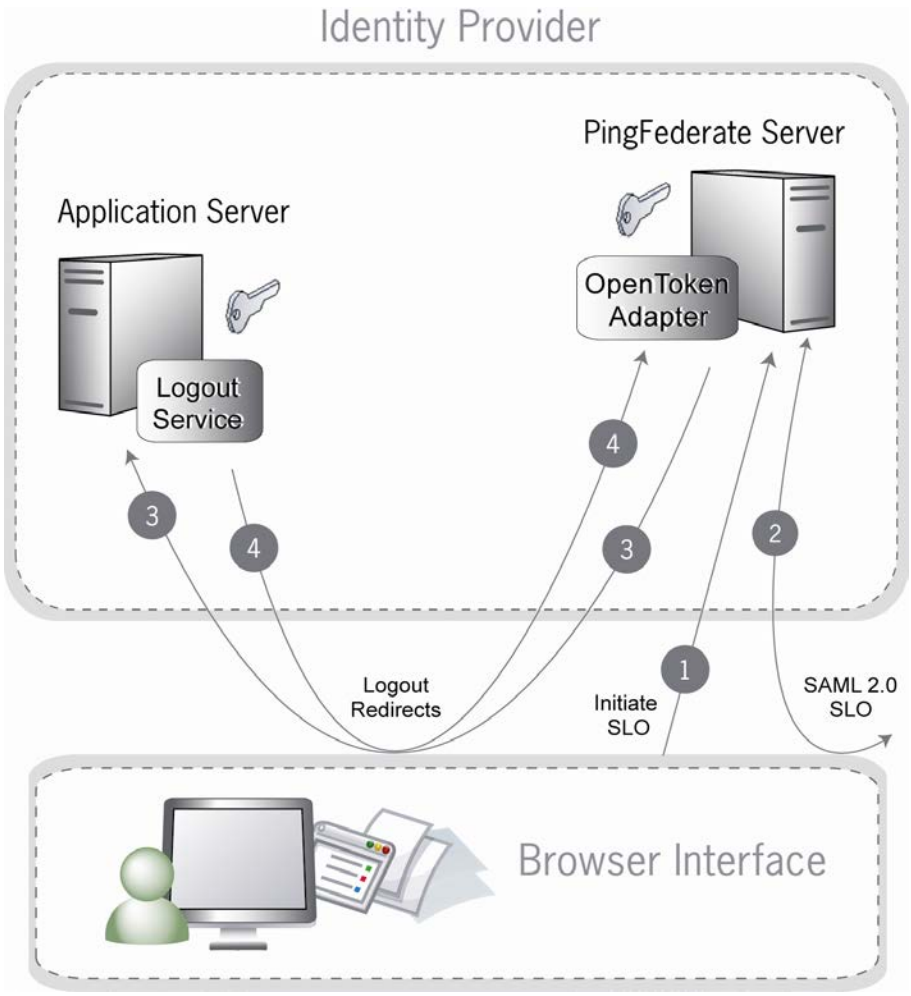
The following code snippet demonstrates how to pass multi-value attributes using the Agent Toolkit:

```
String username = (String)request.getSession().getAttribute("username");
MultiMap userInfo = new MultiValueMap();
userInfo.put(Agent.TOKEN_SUBJECT, username);
// Add an attribute GROUP with multiple values
userInfo.put("GROUP", "Administrators");
userInfo.put("GROUP", "Users");
String returnUrl = "https://<PingFederate_DNS>:9031" +
   request.getParameter("resume");
. . . .
try {
   UrlHelper urlHelper = new UrlHelper(returnUrl);
   //see "Using the Agent API" section for sample code
   //that instantiates and configures an Agent instance
   agent.writeToken(userInfo,response,urlHelper,false);
   returnUrl = (String)urlHelper.toString();
}
catch(TokenException e) {
// Handle exception
}
```

IdP Single Logout (SLO)

When an IdP PingFederate server receives a request for SLO, it redirects the user's browser to the Logout Service defined in the IdP OpenToken Adapter configuration. The redirect URL includes an `OpenToken` containing the user attributes defined in the IdP OpenToken Adapter instance for the partner connection. The Logout Service should remove the user's session on the application server and redirect the user's browser back to the IdP PingFederate server.

The following diagram shows the flow of IdP-initiated SLO, but the architecture would also support SP-initiated SLO:



Sequence

1. User initiates a single logout request. The request targets the PingFederate server's `/idp/startSLO.ping` endpoint.
2. PingFederate sends a logout requests and receives responses for all SPs registered for the current SSO session.
3. PingFederate redirects the request to the IdP Web application's Logout Service, which identifies and removes the user's session locally.
4. The application Logout Service redirects back to PingFederate to display a logout-success page.

Sample Code

Below is an example code snippet for processing a logout request and sending it back to PingFederate through the user's browser:

```
request.getSession().invalidate();
String returnUrl = "https://<PingFederate DNS>:9031" +
   request.getParameter("resume");
response.sendRedirect(returnUrl);
```

Integrating with an SP PingFederate Server

This section provides implementation guidelines and code examples for Java developers, covering the following types of SP SAML 2.0 implementation profiles:

- SP Single Sign-On (SSO)
- SP Single Sign-On (Using Account Linking)
- SP Single Logout (SLO)

SP Single Sign-On (SSO)

When PingFederate is configured as an SP, it takes inbound SAML assertions and converts them to some local format (cookie or otherwise) that can be used by an application to create a user's session. For an `OpenToken`, the PingFederate adapter takes the attributes and values from the SAML assertion and stores them in an `OpenToken` cookie or query parameter in the user's browser. The user is then redirected to the target application, which can then identify the user from the `OpenToken`, using the Agent API.

As with the IdP, you can use the Agent API to read tokens directly. The Agent API is a Java object that provides access to functionality for reading an `OpenToken` from a given HTTP request.

Sample Code

The `readToken` method inspects the cookie (or query parameters, depending on the configuration of the agent instance) and decodes the `OpenToken`, returning a `Collection` of attributes or `null` if no token is found or an error is encountered. In the case of an error, a `TokenException` is thrown. The following code demonstrates the use of this method:

```
try {
   //see "Using the Agent API" section for sample code
   //that instantiates and configures an Agent instance

   Map userInfo = agent.readToken(request);
   if(userInfo != null) {
      String username = (String)userInfo.get(Agent.TOKEN_SUBJECT);
   }
}
```

```
catch(TokenException e) {
   // Handle exception
}
```

Receiving Multi-Value Attributes

The Agent Toolkit for Java receives multi-value attributes passed in the SAML assertion from PingFederate as an `org.apache.commons.collections.MultiMap` collection of attributes.

Sample Code

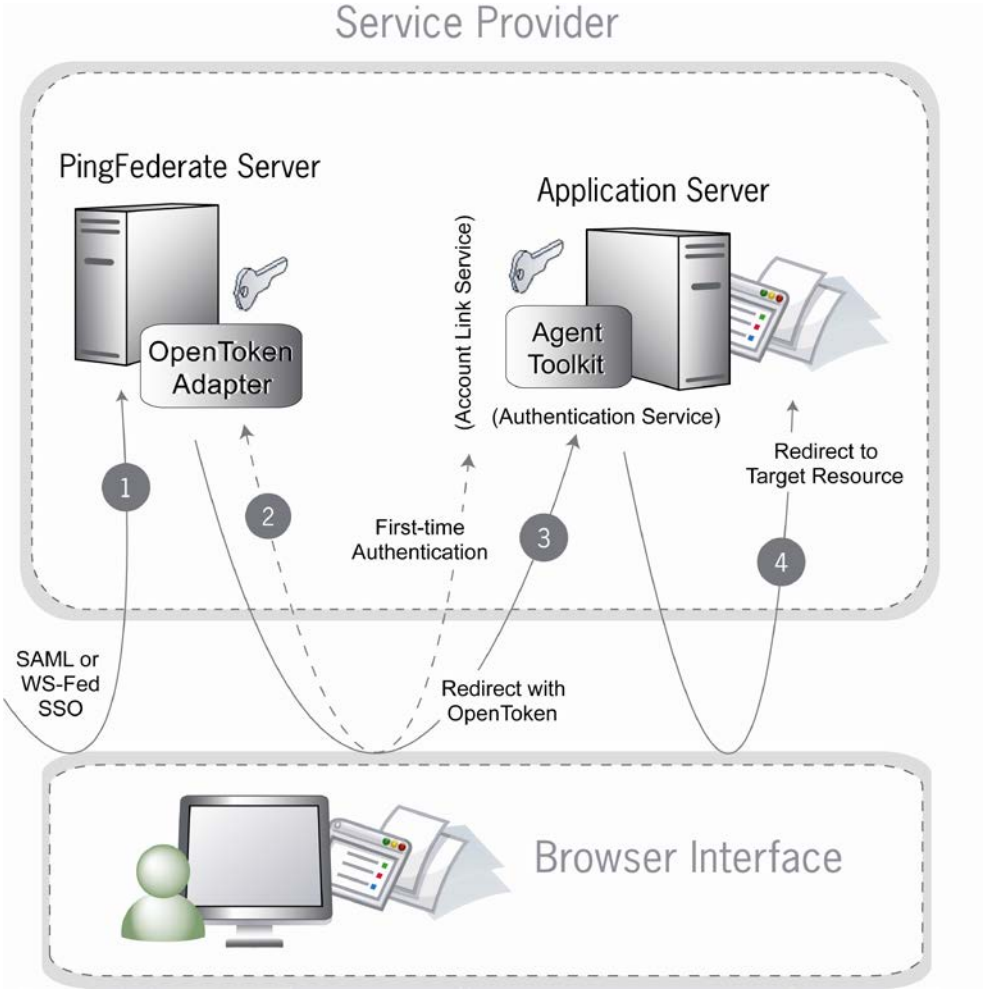
The following code snippet demonstrates how to get the multi-value attributes in the `org.apache.commons.collections.MultiMap` collection using the Agent Toolkit:

```
try {
   //see "Using the Agent API" section for sample code
   //that instantiates and configures an Agent instance
   MultiMap userInfo = agent.readTokenToMultiMap(request);
   if(userInfo != null) {
      String username =
(String)((List)userInfo.get(Agent.TOKEN_SUBJECT)).get(0);
      List groups = (List)userInfo.get("GROUP");
   }
}
catch(TokenException e) {
   // Handle exception
}
```

SP Single Sign-On (Using Account Linking)

If an SP's SSO implementation employs account linking, the flow of events is slightly different since a user must authenticate to the SP application the first time SSO is initiated. (For more information, see Key Concepts in the PingFederate *Administrator's Manual*). In this case, PingFederate and the OpenToken Adapter support an integration mechanism to redirect the user to an Account Link Service to which a user can initially authenticate. Upon successful authentication, the account link service must redirect the user back to PingFederate with an `OpenToken`, which PingFederate uses to create an account link for the user. For subsequent SSO requests, PingFederate uses the account link established in the first SSO request to identify the user. It then creates an `OpenToken` and sends it to the Authentication Service associated with the application.

The following diagram shows the flow of SP SSO using account linking:



Sequence

1. PingFederate receives an assertion under either the SAML 2.0 or WS-Federation protocol.
2. If this is the first time the user has initiated SSO to this SP, PingFederate redirects the browser to the Application Server's Account Link Service, where the user must authenticate. Upon successful authentication, an `OpenToken` is returned to PingFederate, and an account link is established for this user within PingFederate. This account link is used on subsequent SSO transactions.
3. PingFederate retrieves the local user ID from its account link data store. PingFederate's OpenToken Adapter generates an `OpenToken` based on the assertion and account link. PingFederate then redirects the user's browser to the Web application's SSO Authentication Service, passing the `OpenToken` in the redirect.
4. The Authentication Service extracts the contents of the `OpenToken`, establishes a session for the user, and redirects the user's browser to the Target Resource (the `resumePath` URL is sent as a query parameter).

Sample Code

In an Account Linking event, the user's browser is redirected to the configured Account Linking Service in the SP OpenToken Adapter instance. The application should capture the `resumePath` upon a `GET` request to this URL with something similar to the following:

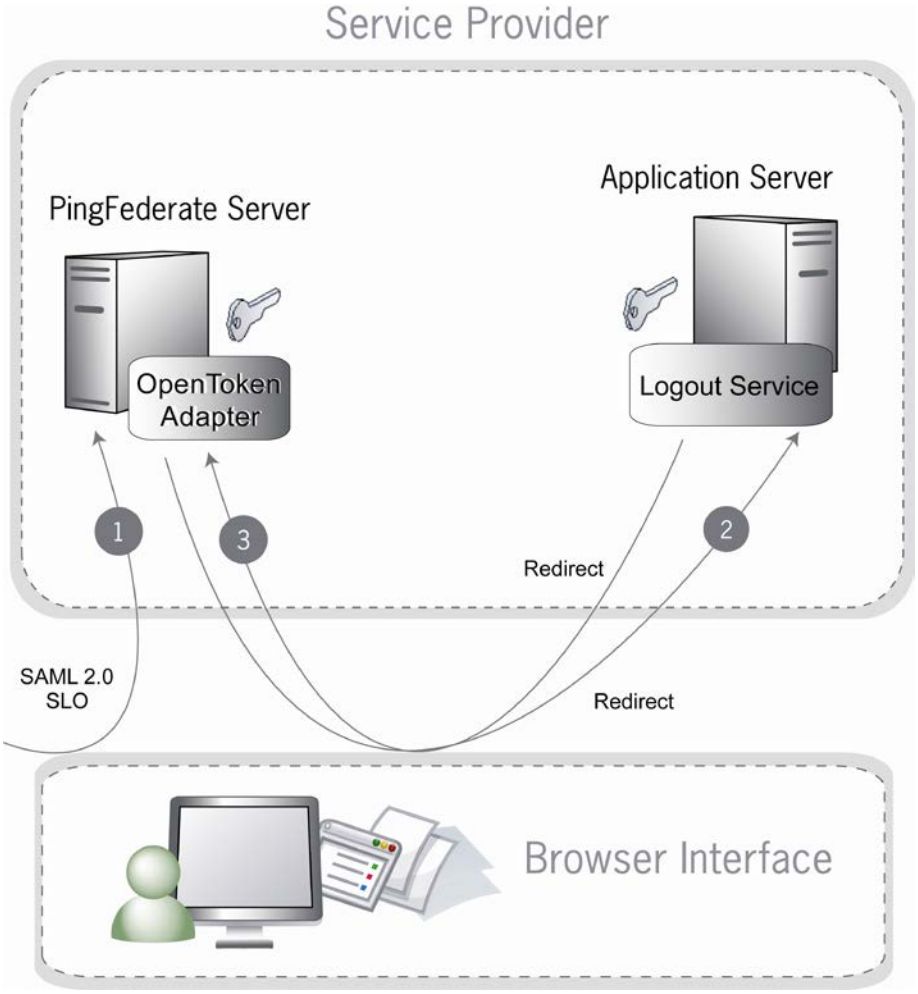
```
String username = (String)request.getSession().getAttribute("username");
Map<String, String> userInfo = new HashMap<String, String>();
userInfo.put(Agent.TOKEN_SUBJECT, username);
String returnUrl = "https://<PingFederate DNS>:9031" +
   request.getParameter("resume");
. . . .
try {
   UrlHelper urlHelper = new UrlHelper(returnUrl);
   //see "Using the Agent API" section for sample code
   //that instantiates and configures an Agent instance
   agent.writeToken(userInfo,response,urlHelper,false);
   returnUrl = (String)urlHelper.toString();
}
catch(TokenException e) {
   // Handle exception
}
response.sendRedirect(returnUrl);
```

SP Single Logout (SLO)

When an SP PingFederate server receives a request for SLO, it redirects the user's browser to the Logout Service as configured in the SP OpenToken Adapter instance. As part of the redirect, PingFederate and the OpenToken Adapter include both an `OpenToken` and a `resumePath` query parameter.

- The `OpenToken` includes attributes about the user.
- The `resumePath` query parameter provides the SP with the target URL where the user's browser must return after the application completes the local logout.

A user can have multiple sessions. This logout sequence, as shown in the following diagram, will occur for each of the user's sessions controlled by the SP PingFederate server.



Sequence

1. PingFederate receives an SLO request under the SAML 2.0 protocol.
2. PingFederate, via the OpenToken Adapter, redirects the browser to the Application Server's Logout Service.
3. The Logout Service returns to PingFederate, indicating that the logout was successful.

The code needed to perform an SP SLO is identical to that required for an IdP SLO (see page 12).

Testing

You can test the Java Integration Kit using one of the sample applications bundled with this distribution. (See the Java Sample Application Startup Guide.)