PingFederate®

SDK Developer's Guide

PingIdentity®

PingFederate SDK *Developer's Guide*
Version 6.10
October, 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

Contents

Preface

This document provides technical guidance for using the Java Software Development Kit (SDK) for PingFederate. Developers can use this *Guide*, in conjunction with the installed Javadocs, to extend the functionality of the PingFederate server.

Intended Audience

The *Guide* is intended for application developers and system administrators responsible for extending PingFederate, including development of:

- Authentication adapters needed to integrate Web applications or identity-management systems (when not already available: see the PingFederate *SSO Integration Overview*, described under Additional Documentation, below)
- Adapter Selectors used to direct SSO authentication to instances of authentication adapters based on specified conditions
- WS-Trust Security Token Service (STS) token translators, including token processors needed to consume and validate security tokens and token generators needed to create security tokens
- Custom data source drivers
- Password credential validators

The reader should be familiar with Java software-development principles and practices.

Additional Documentation

- Javadocs provide detailed reference information for developers. The Javadocs are located in the `<PF_install>/pingfederate/sdk/doc` directory.
- The PingFederate *SSO Integration Overview* describes the types of prebuilt authentication adapters Ping Identity provides for integrating Web applications and identity-management systems with PingFederate. Since these adapters are based on the SDK, you may want to review this document before building your own adapter to see if your needs have already been met.
- The PingFederate *Administrator's Manual* provides background information and user-interface (UI) configuration details needed to integrate implementation(s) of PingFederate interfaces.
- Integration Kit User Guides for Java, .NET, and PHP show examples of SDK implementations.

SDK Introduction

The PingFederate Java SDK consists of several Application Programming Interfaces (APIs), including:

- Adapter and STS Token-Translator Interfaces
- Adapter Selector Interfaces

- Custom Data Source Interfaces
- Password Credential Validator Interfaces

Each of these interfaces allows users to create their own plug-ins, customizing certain behaviors of PingFederate to suit an organization's needs. This SDK provides a means to develop, compile, and deploy custom plug-ins to PingFederate.

A number of example plug-ins are included in the PingFederate package for reference. The example projects are located in the `<PF_install>/sdk/plugin-src` directory.

Adapter and STS Token-Translator Interfaces

The adapter and token-translator APIs enable PingFederate integration with IdPs or SPs. The APIs allow developers to build their own custom implementations for communicating authentication and security information between PingFederate and the enterprise environment.

Note: Token-translator interfaces are applicable only to PingFederate versions 6.0 and higher.

In addition to providing requisite runtime integration, an adapter or token translator also describes its configuration parameters to PingFederate; this enables the administrative console to render configuration screens with extensible validation.

Note: Suitable adapter or token-translator implementations for your deployment may already exist, or new implementations may be under development. Before developing your own custom solution, see the Downloads page (`www.pingidentity.com/support-and-downloads`) for more information about currently available implementations.

Adapter Selector Interfaces

Adapter selectors provide a mechanism to choose among multiple adapters and to direct a user to use a particular adapter, depending on the specified conditions. For example, an adapter selector may map internal corporate users to use one adapter, while it maps external noncorporate users to a different adapter.

Note: Adapter selector interfaces are applicable only to PingFederate versions 6.6 and higher.

Similar to the adapter API, adapter selectors are configurable UI plug-ins, allowing you to render custom configuration screens.

Custom Data Source Interfaces

The custom data source API is a set of Java interfaces that enable PingFederate to integrate with data stores not covered by existing LDAP or JDBC drivers. This allows developers to retrieve attributes from a data source of their choice during attribute fulfillment for various use cases. Similar to the adapter API, custom data source plug-ins also provide much of the same UI configuration functionality.

Password Credential Validator Interfaces

The password credential validator interfaces allow developers to define credential validators that are used to verify a given username and password in various contexts throughout the system. For example, credential validators are used to configure OAuth Resource Owner authorization grants and the HTML Form IdP Adapter.

> **Note**: Credential validator interfaces are applicable only to PingFederate versions 6.5 and higher.

Ping Identity Global Client Services

If you need assistance in using the SDK, visit the Ping Identity Support Center (`www.pingidentity.com/support`) to see how we can help you with your application.

Getting Started With the SDK

This section describes the directories and build components that comprise the SDK and provides instructions for setting up a development environment.

Directory Structure

The PingFederate SDK directory (`<PF_install>/pingfederate/sdk`) contains the following:

- `plugin-src/` – The directory where you place your custom plug-in projects. This directory also contains example plug-in implementations showing a wide range of functionality. You may use these examples for developing your own implementations.
- `doc/` – Contains the SDK Javadocs. Open `index.html` to get started.
- `lib/` – Contains libraries used for compiling and deploying custom components into PingFederate.
- `build.properties` – This file contains properties used by the Ant build script, `build.xml`, to compile and deploy your custom components. Do not modify this file; use `build.local.properties` to override any properties, if needed.
- `build.local.properties` – Allows you to specify which project you want to build and define properties specific to your environment. The main use of this file is declaring the project you want to build.
- `build.xml` – The Ant build script used to compile, build, and deploy your component. This file should not need modification.

Setting Up Your Project

To start developing your own plug-in:

1. Before you start, ensure you have the Java SDK and Apache Ant installed.

2. To create a new plug-in, create a new project directory in the `<PF_install_dir>/pingfederate/sdk/plugin-src` directory.
3. In the new project directory, create a subdirectory named `java`.

   This is where you place the Java source code for your implementation(s).

   Follow standard Java package and directory structure layout.
4. If your project depends on third-party libraries, create another subdirectory called `lib` and place the necessary JAR files in it.
5. The build script builds only one project at a time. Edit the `build.local.properties` file and set `target-plugin-name` to specify the name of the directory (under `<PF_install>/pingfederate/sdk/plugin-src`) that contains your project.
6. In `<PF_install>/pingfederate/sdk` run `ant` to display a list of available build targets:

```
[java] Main targets:
[java]
[java]  clean-plugin   Clean the plug-in build directory
[java]  deploy-plugin  Deploy the plug-in jar and libs to PingFederate
[java]  jar-plugin     Package the plug-in jar
[java]
[java] Default target: help
```

   Run the appropriate target to clean, build, or deploy your plug-in.

Note: Building the project with the `build.xml` included in the SDK is recommended since it packages the jars with additional metadata to make it discoverable by PingFederate. For detailed information, see Building and Deploying Your Project on page 17.

Implementation Guidelines

The following sections provide specific programming guidance for developing custom interfaces. Note that the information is not exhaustive—consult the Javadocs to find more details about interfaces discussed here as well as additional functionality.

Shared Interfaces

All plug-in implementations generally invoke methods discussed in the following sections.

Configurable Plug-in

Any custom plug-in that requires UI settings is considered *configurable* and hence implements the `ConfigurablePlugin` interface. This ensures that PingFederate loads the plug-in instance with the correct configuration settings.

All plug-in types implement the `ConfigurablePlugin` interface and must define the following to enable configuration loading:

```
void configure(Configuration configuration)
```

During processing of a configurable plug-in instance, PingFederate calls the `ConfigurablePlugin.configure` method and passes in a `Configuration` object. The `Configuration` object provides the plug-in adapter-instance configuration set by an administrator in the PingFederate UI.

The `sp-adapter-example` provided with the SDK shows how to use this method to initialize an adapter-instance from a saved configuration. Once your implementation loads the configuration values, the plug-in instance can use them in other method calls.

Describable Plug-in

Any plug-in that requires configuration screens in the PingFederate administrative console is considered a *describable* plug-in. Most plug-ins implement the `DescribablePlugin` interface to ensure that PingFederate renders the correct UI components based on the returned `PluginDescriptor`.

Adapter and custom data source plug-ins are a special case and do not implement the `DescribablePlugin` interface. However, they still return a plug-in descriptor (`AuthnAdapterDescriptor` and `SourceDescriptor` respectively) and are still considered describable plug-ins.

All describable plug-ins must define a UI descriptor. Use one of the following methods to implement a UI descriptor, depending on the type of plug-in:

- For `DescribablePlugin`:

  ```
  PluginDescriptor getPluginDescriptor()
  ```

- For adapter plug-ins:

  ```
  AuthnAdapterDescriptor getAdapterDescriptor()
  ```

- For custom data source plug-ins:

  ```
  SourceDescriptor getSourceDescriptor()
  ```

In many cases, describable plug-ins return a subclass of `PluginDescriptor`, so the return type of the plug-in descriptor getters might be slightly different among plug-in implementations. Your plug-in implementation populates `PluginDescriptor` with `FieldDescriptors`, `FieldValidators`, and `Actions` and is presented as a set of UI components in the PingFederate administrative console.

> **Tip**: Some plug-in types offer concrete descriptor implementations for developers. The Javadocs and examples provided with the SDK show which descriptor classes are available for each plug-in type. The examples also show you how to use `FieldDescriptors`, `FieldValidators`, and `Actions` directly to define your plug-in descriptor.

IdP Adapter Implementation

You create an IdP adapter by implementing the `IdpAuthenticationAdapter` or the `IdpAuthenticationAdapterV2` interface. The following Java packages are needed, at a minimum, for implementing this interface:

- `org.sourceid.saml20.adapter.idp.authn`
- `org.sourceid.saml20.adapter.gui`

- `org.sourceid.saml20.adapter.conf`

For each IdP adapter implementation, in addition to the methods described under Shared Interfaces, you must define the following:

- Session Lookup
- Session Logout

IdP Adapter Session Lookup

```
java.util.Map lookupAuthN(javax.servlet.http.HttpServletRequest req,
   javax.servlet.http.HttpServletResponse resp,
   java.lang.String partnerSpEntityId,
   AuthnPolicy authnPolicy,
   java.lang.String resumePath)
   throws AuthnAdapterException, java.io.IOException
```

PingFederate invokes the `lookupAuthN` method of your IdP adapter to look up user-session information to handle a request. This method is invoked regardless of whether the request is for IdP- or SP-initiated SSO, an OAuth transaction, or direct IdP-to-SP adapter processing.

> **Note**: The `IdpAuthenticationAdapterV2` interface provides an overloaded version of `lookupAuthN` applicable to PingFederate versions 6.4 and higher. Use this interface if your adapter requires additional parameters from PingFederate. Refer to the `IdpAuthenticationAdapterV2` interface in the Javadocs for a complete list of available parameters.

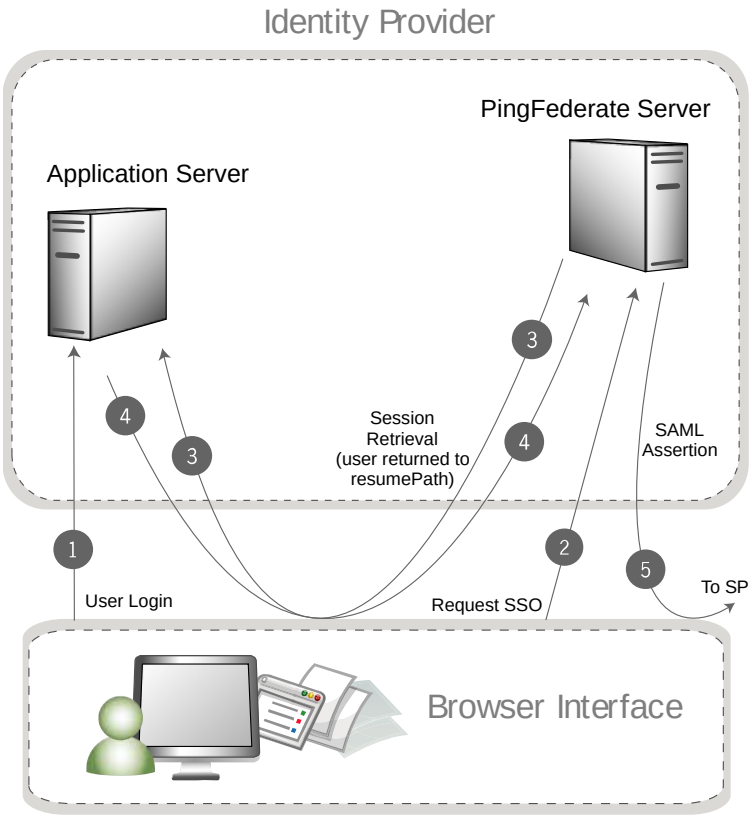
In most implementations, a user's session information or a reference to it is communicated to PingFederate via the `HttpServletRequest`, which is passed to the `lookupAuthN` method. For example, the user's session information can be passed in by the IdP application as a cookie or query parameter.

If the request from the user's browser does not contain the necessary information to identify the user, the `HttpServletResponse` can be used in various ways to retrieve the user's session data—for example, by creating a 302 redirect or presenting a Web page asking for credentials. If your adapter implementation uses the `HttpServletResponse` to retrieve the user's session information, you must return the user's browser to the URL in the `resumePath` parameter set by the PingFederate runtime server and passed to this method. The `resumePath` is a relative URL signaling PingFederate that a user is continuing an SSO transaction that has already been initiated.

If your adapter implementation writes to the `HttpServletResponse` to retrieve the user's session data, we recommend that the browser return to the `resumePath` URL at all times, whether the retrieval succeeds or fails. Doing so ensures the adapter does not interrupt the "adapter chain" if it is used with the Composite Adapter. The Composite Adapter allows an administrator to "chain" together a selection of available adapter instances for a connection. At runtime, adapter chaining means that SSO requests are passed sequentially through each adapter instance until one or more authentication results are found for the user. If the browser is unable to return to the `resumePath` URL at all times, then it could interrupt the adapter chain causing unexpected results for the Composite Adapter.

For some authentication mechanisms, not all adapters can return the browser to the `resumePath` URL. Such adapters should not be used with the Composite Adapter's "Sufficient" chaining policy (see Composite Adapter Configuration in the PingFederate *Administrator's Manual*).

The following diagram illustrates the request sequence of an IdP-initiated SSO scenario that uses the `resumePath`:



Processing Steps

1. User logs in to a local application or domain through an identity-management system or some other authentication mechanism.
2. User clicks a link or otherwise requests access to a protected resource located in the SP domain. The link or other mechanism invokes the PingFederate SSO service.
3. PingFederate invokes the designated adapter's lookup method, including the `resumePath` parameter. In this example, the adapter determines there isn't enough information and redirects the browser to the application server to fetch additional session information.
4. The application server returns the session information and redirects the browser along with the returned information to `resumePath` URL.
5. PingFederate generates a SAML assertion and sends the browser with the SAML assertion to the SP's SAML gateway.

IdP Adapter Session Logout

```
boolean logoutAuthN(java.util.Map authnIdentifiers,
   javax.servlet.http.HttpServletRequest req,
   javax.servlet.http.HttpServletResponse resp,
   java.lang.String resumePath)
   throws AuthnAdapterException, java.io.IOException
```

During SLO request processing, PingFederate invokes your IdP adapter's `logoutAuthN()` method to terminate a user's session. This method is invoked during IdP- or SP-initiated SLO requests.

Like the `lookupAuthN()` method, the `logoutAuthN()` method has access to the user's `HttpServletRequest` and `HttpServletResponse` objects. Use these objects to retrieve data about the user's session as well as to redirect the browser to an endpoint used to terminate the session at the application. Again, the `resumePath` parameter contains the URL to which the user is redirected to complete the SLO process.

SP Adapter Implementation

You create an SP adapter by implementing the `SPAuthenticationAdapter` interface. The Java packages required are, at a minimum:

- `org.sourceid.saml20.adapter.sp.authn`
- `org.sourceid.saml20.adapter.gui`
- `org.sourceid.saml20.adapter.conf`

At a high level, in addition to the methods described under Shared Interfaces, you must define the following:

- Session Creation
- Session Logout
- Account Linking (if configured in PingFederate for an IdP partner)

SP Session Creation

```
java.io.Serializable createAuthN(SsoContext ssoContext,
   javax.servlet.http.HttpServletRequest req,
   javax.servlet.http.HttpServletResponse resp,
   java.lang.String resumePath)
```

PingFederate invokes the `createAuthN()` method during the processing of an SSO request to establish a security context in the external application for the user. This method is similar to the `IdpAuthenticationAdapter.lookupAuthN()` method in terms of the objects passed to it and its support for asynchronous requests via the `HttpServletResponse` and `resumePath` parameters. This method also accepts an `SsoContext` object, which has access to information such as user attributes and the target destination URL.

SP Adapter Session Logout

```
boolean logoutAuthN(java.io.Serializable authnBean,
   javax.servlet.http.HttpServletRequest req,
   javax.servlet.http.HttpServletResponse resp,
   java.lang.String resumePath)
   throws AuthnAdapterException, java.io.IOException
```

PingFederate invokes the `logoutAuthN()` method during an SLO request to terminate a user's session with the external application. The `HttpServletResponse` and `resumePath` objects are available to support scenarios where redirection of the user's browser is needed to an additional service to clean up any remaining sessions.

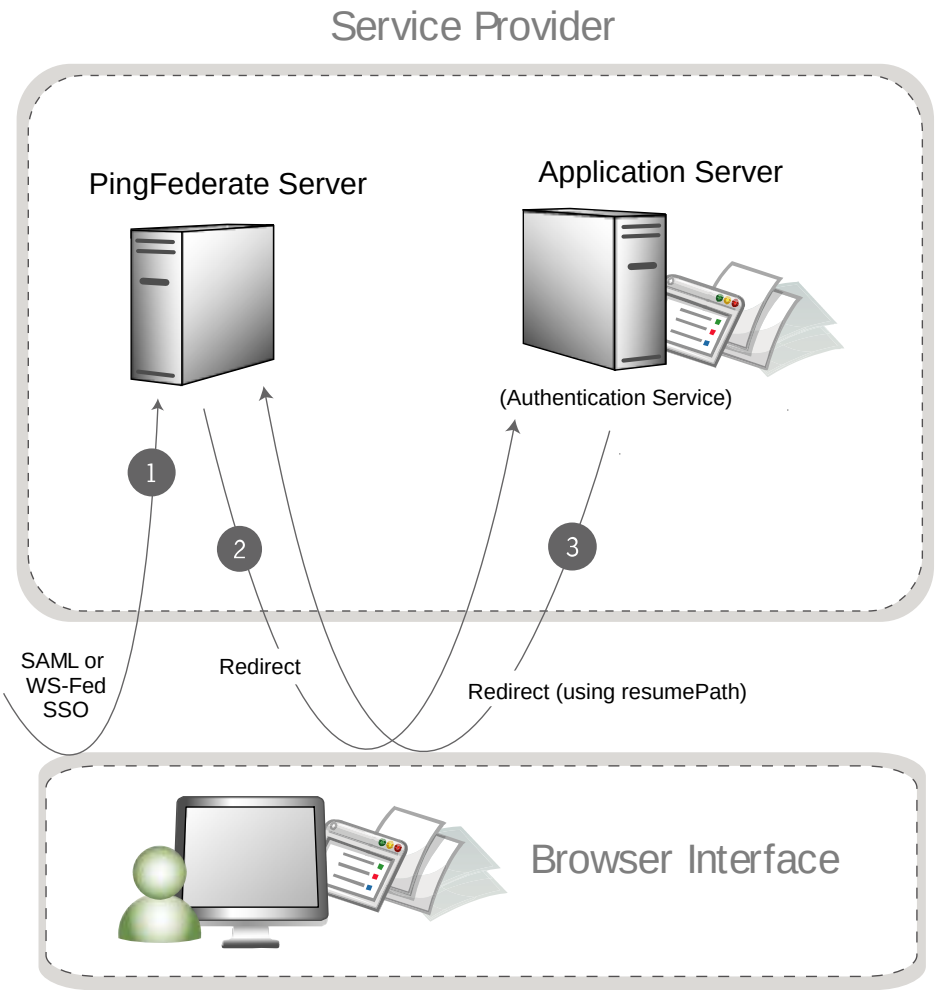
SP Account Linking

```
java.lang.String lookupLocalUserId(
   javax.servlet.http.HttpServletRequest req,
   javax.servlet.http.HttpServletResponse resp,
   java.lang.String partnerIdpEntityId,
   java.lang.String resumePath)
   throws AuthnAdapterException, java.io.IOException
```

PingFederate invokes the `lookupLocalUserId()` method during an SSO request when the IdP connection is configured to use account linking but no account link for this user is yet established. Once the account link is set, PingFederate maintains this information until the user "defederates." Defederation occurs when the user clicks a link redirecting him/her to the `/sp/defederate.ping` PingFederate endpoint.

The `HttpServletResponse` and `resumePath` objects are used to send the user to a local service where the user authenticates. After authentication, the user is redirected to the URL specified in the `resumePath` parameter and PingFederate completes the account link.

The following diagram illustrates a typical account-link sequence:



Use the `HttpServletRequest` to read a local session token. The `String` object returned from the `lookupLocalUserId()` method should be a local user identifier.

Token Processor Implementation

You create a token-processor implementation (for PingFederate 6.0 and higher) by implementing the `TokenProcessor` interface. The following Java packages are needed, at a minimum, for implementing this interface:

- `org.sourceid.saml20.adapter.attribute`
- `org.sourceid.saml20.adapter.idp.authn`
- `org.sourceid.saml20.adapter.gui`
- `org.sourceid.saml20.adapter.conf`
- `org.sourceid.wstrust.model`
- `org.sourceid.wstrust.plugin`
- `org.sourceid.wstrust.plugin.process`
- `com.pingidentity.sdk`

For each token-processor implementation, in addition to the methods described under Shared Interfaces, you must define the method:

```
TokenContext processToken(T token)
```

PingFederate invokes the `processToken` method during the processing of an STS request to perform necessary operations for determining the validity of a token. Type `T` must extend, at a minimum, the type `SecurityToken`. The type `BinarySecurityToken` is also available and may be used to represent custom security tokens that can be transported as Base64-encoded data.

Token Generator Implementation

You create a token-generator implementation (for PingFederate 6.0 and higher) by implementing the `TokenGenerator` interface. The following Java packages needed, at a minimum, for implementing this interface:

- `org.sourceid.saml20.adapter.sp.authn`
- `org.sourceid.saml20.adapter.gui`
- `org.sourceid.saml20.adapter.conf`
- `org.sourceid.wstrust.model`
- `org.sourceid.wstrust.plugin`
- `org.sourceid.wstrust.plugin.process`
- `com.pingidentity.sdk`

For each token-generator implementation, described under Shared Interfaces, you must define the method:

```
SecurityToken generateToken(TokenContext attributeContext)
```

PingFederate invokes the `generateToken()` method during the processing of an STS request to perform necessary operations for generation of a security token. The type `BinarySecurityToken` is available and may be used to represent custom security tokens that can be transported as Base64-encoded data. The `TokenContext` contains subject data available for insertion into the generated security token.

Adapter Selector Implementation

Adapter selectors allow PingFederate (version 6.6 and higher) to choose an appropriate IdP adapter based on criteria defined in the adapter selector instance.

When creating an adapter selector, the following are the primary Java packages used:

- `org.sourceid.saml20.adapter.gui`
- `org.sourceid.saml20.adapter.conf`
- `com.pingidentity.sdk`

For each adapter selector implementation, in addition to the methods described under Shared Interfaces, you must define the following at a minimum:

- Context Selection
- Adapter Callback

Context Selection

```
AdapterSelectorContext selectContext(HttpServletRequest req,
   HttpServletResponse resp,
   Map<String, String> mappedAdapterIdsNames,
   Map<String, Object> extraParameters,
   String resumePath)
```

PingFederate calls the `selectContext()` method to determine which adapter to select. The `mappedAdapterIdsNames` contains the list of adapter IDs that are available for the selector to reference. The `HttpServletRequest` is available to evaluate cookies, parameters, headers, etc. to help determine which adapter should be selected. The `HttpServletResponse` is also available if the adapter selector requires user interaction to help determine the appropriate adapter to select. If the `resp` object is written to, it is considered a committed response and returned to the user's browser. The `resumePath` is a relative URL that should be used in conjunction with the `resp` object, such that the user's browser can be sent to this URL to resume the SSO workflow.

Once an adapter is selected, an `AdapterSelectorContext` can be created to denote which adapter to use. The selected adapter can be referenced by its adapter-instance ID or by its context. The context is a name that decouples adapter selectors from the configured adapter-instance IDs.

Adapter Callback

```
void callback(HttpServletRequest req,
   HttpServletResponse resp,
   Map authnIdentifiers,
   String adapterInstanceId,
   AdapterSelectorContext adapterSelectorContext)
```

PingFederate calls the `callback()` method after a selected adapter is authenticated against. The `callback()` method allows adapter selectors to update resulting attributes, set cookies, or perform other custom functions.

Note: Writing content to the `resp` object in the `callback()` method is not supported, and doing so may result in unexpected behavior. Setting cookies is acceptable.

Custom Data Source Implementation

Out of the box, PingFederate provides the capability of querying data sources for a variety of purposes using LDAP or JDBC interfaces. You can use the PingFederate SDK to build data source connectors to query additional data source types. Examples of other data sources include a Web service, a flat file, or perhaps a different way of using a JDBC or LDAP connection than what is supplied by PingFederate.

The following are the primary Java packages used to build a custom data source:

- `com.pingidentity.sources`
- `com.pingidentity.sources.gui`

For each implementation, described under Shared Interfaces, you must define the following at a minimum:

- Connection Testing
- Available Fields Retrieval
- Data Source Query Handling

Data Source Connection Testing

```
boolean testConnection()
```

When associating a custom data source with an IdP or SP connection, PingFederate tests connectivity to the data source by calling the `testConnection()` method. Your implementation of this method should perform the necessary steps to demonstrate a successful connection and return `true`. Return `false` if your implementation cannot communicate with the data store. A `false` result prevents an administrator from continuing with the data source configuration.

Data Source Available Fields Retrieval

```
java.util.List<java.lang.String> getAvailableFields()
```

PingFederate calls the `getAvailableFields()` method to determine the available fields that could be returned from a query of this data source. These fields are displayed to the PingFederate administrator during the configuration of a data source lookup. The administrator can then select the attributes from the data source and map them to the adapter or attribute contract. PingFederate requires at least one field returned from this method.

Data Source Query Handling

```
java.util.Map<java.lang.String,java.lang.Object> retrieveValues(
   java.util.Collection<java.lang.String> attributeNamesToFill,
   SimpleFieldList filterConfiguration)
```

When processing a connection using a custom data source, PingFederate calls the `retrieveValues()` method to perform the actual query for user attributes. This method receives a list of attribute names that should be populated with data. The method may also receive a `filterConfiguration` object populated with a list of fields. Each field contains a name/value pair that is determined at runtime and collectively used as the criteria for selecting a specific record. In most cases, the criteria is used to locate additional user attributes.

You create the filter criteria selections needed for this lookup by passing back a `CustomDataSourceDriverDescriptor`, an implementation of `SourceDescriptor`, from the `getSourceDescriptor()` method. A `CustomDataSourceDriverDescriptor` can include a `FilterFieldDataDescriptor` composed of a list of fields that can be used as the query criteria. This list of fields is displayed similarly to the other UI-descriptor display fields.

Note: The `filterConfiguration` object is set and populated with a list of fields only if the data source was defined with a `CustomDataSourceDriverDescriptor`. If the `CustomDataSourceDriverDescriptor` was not used in the definition of the data source, the `filterConfiguration` object is set to `null`.

Important: To pass runtime attribute values to the filter, an administrator must reference the attributes using the `${attribute name}` format when defining a filter in the PingFederate administrative console.

Once all the relevant attributes are retrieved from the data source, they must be returned as a map of name/value pairs, where the names correspond to the initial collection of attribute names that was passed into the method and the values are the attributes.

Password Credential Validator Implementation

Password credential validators allow PingFederate administrators to define a centralized location for username/password validation, allowing validator instances to be referenced by various PingFederate configurations..

To implement a custom password credential validator, the following Java packages need to be imported:

- `org.sourceid.saml20.adapter.gui`
- `org.sourceid.saml20.adapter.conf`
- `org.sourceid.util.log`
- `com.pingidentity.sdk`
- `com.pingidentity.sdk.password`

For each implementation, in addition to the methods described under Shared Interfaces, you must define the following at a minimum:

```
AttributeMap processPasswordCredential(String username,
   String password)
   throws PasswordValidationException
```

This method takes a `username` and `password` and verifies the credential against an external source. If the credentials are valid, then an `AttributeMap` is returned containing at least one entry representing the principal. If the credentials are invalid, then `null` or an empty map is returned. A

`PasswordValidationException` is thrown if the plug-in was unable to validate the credentials (for example, due to an offline host or network problems).

Building and Deploying Your Project

To build and deploy your project, you can choose to use the provided Apache Ant script or another build utility.

Building and Deploying With Ant

The PingFederate Java SDK comes with an Apache Ant build script that makes building and deploying your project simple.

1. Edit the `build.local.properties` file and set the `target-plugin.name` property to the name of your project subdirectory (see Directory Structure on page 6).

   Note: You can develop source code for multiple projects simultaneously, but you can build and deploy only one at a time. Change the value of the `target-plugin.name` property as needed to build and deploy other projects.

2. If your project depends on any third-party jars, place them into your project's `lib` directory.

   If the directory does not exist, create a new directory called `lib`, directly under your project's directory, For example, `pingfederate/sdk/plugin-src/<subproject-name>/lib`

3. On the command line in the `sdk` directory, use `ant` to clean, build, and package or to build, package, and deploy your project.

   To clean the project, enter:

   ```
   ant clean-plugin
   ```

   To compile the project, enter:

   ```
   ant compile-plugin
   ```

   To compile the project and create a JAR, enter:

   ```
   ant jar-plugin
   ```

   The SDK creates deployment descriptor(s) in the `PF_INF` directory and places it in a JAR. The descriptor tells PingFederate what plug-in implementations are contained in the JAR.

   The compiled class files and the deployment descriptor(s) are placed in the `pingfederate/sdk/plugin-src/<subproject-name>/build/classes` directory.

   The `pf.plugins.<subproject-name>.jar` file is placed in the `pingfederate/sdk/plugin-src/<subproject-name>/build/jar` directory.

   To compile, create a JAR, and deploy the project to PingFederate, enter:

   ```
   ant deploy-plugin
   ```

This build target performs the steps described above as well as deploying any JAR files found in the `lib` directory of your subproject.

Note: To deploy your plug-in manually to an installation of the PingFederate server, copy the JAR file and any third-party JAR files into the `/server/default/deploy/` directory of that PingFederate installation.

4. Restart the PingFederate server.

Manually Building and Deploying

To build your project with another build utility, you must take some prerequisite steps to create the deployment descriptors for each of your plug-ins. The deployment descriptor files allow PingFederate to discover your plug-ins.

Creating Deployment Descriptors:

1. In your project, create a new directory called `PF-INF`. This directory must be at the root of your JAR file, similar to `META-INF`.
2. Inside `PF-INF` create the appropriate text file(s) for each type of plug-ins you created:

Plug-in Type	Filename
IdP Adapter	idp-authn-adapters
SP Adapter	sp-authn-adapters
Custom Data Source	custom-drivers
Token Processor	token-processors
Token Generator	token-generators
Adapter Selector	adapter-selectors
Password Credential Validator	password-credential-validators

3. In each text file created, specify the fully qualified class name of each plug-in that implements the corresponding plug-in interface. Place each class name on a separate line.

Manually Building Your Project:

To compile your project, you need to have the following directories on your classpath:

- `pingfederate/server/default/lib`
- `pingfederate/lib`
- `pingfederate/sdk/lib`
- `pingfederate/sdk/plugin-src/<subproject-name>/lib`

To create a JAR, simply archive the compiled class files along with the deployment descriptor(s) using your build tool. The deployment descriptors must be in the `PF-INF` directory, located at the root of the JAR.

Deploying Your Project:

To deploy your plug-in, simply copy the JAR file and any third-party JAR files into the `pingfederate/server/default/deploy` directory of the PingFederate installation.

Logging

You can use a typical logging pattern based on the Apache Commons logging framework to log messages from your adapter, token translator, or custom data source driver. The SP adapter contained in the directory `sdk/adapters-src/sp-adapter-example` shows how to use a logger for your adapter.