PingFederate®

Software Development Kit Developer's Guide

PingIdentity®

PingFederate SDK *Developer's Guide*
Version 1.3
April, 2010

Ping Identity Corporation
1099 18th Street, Suite 2950
Denver, CO 80202
U.S.A.

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

Trademarks

Disclaimer

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 Javadoc reference material, to extend the functionality of the PingFederate server (for versions 4.0 and higher).

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 *Integration Overview*, described under "Additional Documentation," below)
- WS-Trust Security Token Service (STS) token translators, including token processors needed to integrate with Web Service Clients at an Identity Provider (IdP) site, or token generators for Web Service Providers at a Service Provider (SP) site.

  Note: Token-translator interfaces are applicable only to PingFederate versions 6.x and higher.

- Custom data-source drivers
- Interfaces to modify certain runtime-processing behavior

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

Additional Documentation

API Javadocs provides detailed reference information for developers. The Javadoc `index.html` file is located in the `<PF_install_dir>/pingfederate/sdk/doc` directory.

The PingFederate *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 wish to review this document before building your own adapter to see if your needs have already been met. The *Integration Overview* is located in `<PF_install_dir>/pingfederate/docs`.

The PingFederate *Administrator's Manual* provides background information and user-interface (UI) configuration details needed to integrate implementation(s) of PingFederate interfaces. The manual is also located in the `<PF_install_dir>/pingfederate/docs` directory.

The PingFederate Java/.NET Integration Kit *Developer's Guide* describes how PingFederate 4.x, the built-in Standard Adapter, and either the Java or .NET Integration Kits handle common federated-identity transactions. The guide includes code snippets demonstrating the implementation of each scenario. This document is available on the Ping Identity Web site.

Note: For PingFederate 5.x and higher, information contained in the Integration Kit *Developer's Guide* has been updated and migrated into the Integration Kit *User Guides* for Java., NET, and PHP.

Introduction

The PingFederate Java SDK consists of:

- Adapter and STS Token-Translator Interfaces
- Custom Data-Source Interfaces
- Service Interfaces

Adapter and STS Token-Translator Interfaces

The adapter and token-translator SDK is a set of Java interfaces and APIs that enable PingFederate integration with external applications and services. This framework provides a means to develop, compile, and deploy custom adapters, token processors (for an IdP), or token generators (for an SP). The SDK allows developers to build their own custom interfaces for communicating authentication and security information between PingFederate and the enterprise environment.

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.

A number of example adapter and token-translator implementations are included in the PingFederate package for reference. The example projects are located in the `<PF_install_dir>/sdk/*-src` directories.

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 Ping Identity Web site for more information about currently available implementations.

Custom Data-Source Interfaces

The custom data-source SDK is a set of Java interfaces and APIs that enable PingFederate to integrate with data stores not covered by existing LDAP or JDBC drivers. Modeled after the adapter SDK, this SDK provides much of the same dynamic UI functionality.

Service Customization Interfaces

The service interfaces define the behavior of a number of services that the PingFederate server needs to function properly. Implementations of service interfaces support, for example, a cluster of runtime engines and the exchange of persistent data between them.

PingFederate provides default implementations of these interfaces, and most deployments will not require any customization.

Note: This *Guide* includes build and deployment information for the service interfaces but does not provide SDK implementation guidelines. If you need to alter a service implementation, please refer to the Javadocs in the `sdk/doc` directory.

Ping Identity Global Client Services

If you need assistance in using the SDK, contact Ping Identity at support.pingidentity.com to see how we can help you with your application.

Getting Started

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_dir>/pingfederate/sdk`) contains the following:

- `adapters-src/` – The directory where you place your custom adapter project. This directory also contains example adapter implementations showing a wide range of functionality. You may use these examples for developing your own implementations.
- `tokens-src/` – The directory where you place your custom token-translator project. This directory also contains example token-translator implementations showing both token processing and generation. 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 properties specific to your environment and override properties set in the `build.properties` file. The main use of this file is declaring the project you wish to build.
- `build.xml` – The Ant build script used to compile, build, and deploy your component. This file should not need modification.

Setting Up a Development Environment

To start developing your own adapter, token-translator, data-source driver, or custom service implementation:

1. For an adapter or data-source driver, create a new project directory in the `<PF_install_dir>/pingfederate/sdk/adapters-src` directory.

   For a token translator, create a new project directory in the `<PF_install_dir>/pingfederate/sdk/tokens-src` directory.

   For a customized service, create a new directory called:
   `<PF_install_dir>/pingfederate/sdk/services-src`

2. In the new project directory, create a subdirectory named `java`.

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

   Follow standard Java package and directory structure layout.

3. If your project depends on third-party libraries, create another subdirectory called `lib` and place the necessary JAR files in it.

Implementation Guidelines

The following sections provide specific programming guidance for developing custom interfaces. Each section covers uses of the SDK for typical sets of requirements, discussing the primary Java packages required. Note, however, that the information is not exhaustive—consult the Javadocs to find more details about interfaces discussed here as well as additional functionality.

IdP Adapter Implementation

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

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

For each IdP adapter implementation, you must define the following:

- Adapter UI Descriptor
- Configuration Retrieval
- Session Lookup
- Session Logout

Adapter UI Descriptor

```
IdpAuthnAdapterDescriptor getAdapterDescriptor()
```

Deploying your adapter requires configuration of an adapter instance in the PingFederate administrative console. The `IdpAuthenticationAdapter` interface includes a `getAdapterDescriptor()` method which returns an `IdpAuthnAdapterDescriptor`. Your adapter implementation populates the `IdpAuthnAdapterDescriptor` with `FieldDescriptors` presented as UI controls to the administrator through the PingFederate administrative console.

The `api-usage-example` for an SP adapter provided with the SDK shows how to use most of the configuration `FieldDescriptors`, data `Validators`, and `Actions`. These implementations generally apply to IdP implementations as well.

Configuration Retrieval

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

During processing of a single sign-on (SSO) or single logout (SLO) transaction, an adapter instance must reference its configuration as set by the administrator in the PingFederate UI. The

`IdpAuthenticationAdapter.configure()` method (inherited from `ConfigurableAuthAdapter`) provides access to this data. During transaction processing, PingFederate calls this method and passes in a `Configuration` object. The `Configuration` object provides access to the configuration values.

The `api-usage-example` SP adapter provided with the SDK shows how to use the `configure()` method to retrieve an adapter-instance configuration. Once your implementation captures the configuration values, the adapter instance can use them during session lookup or session logout.

Session Lookup

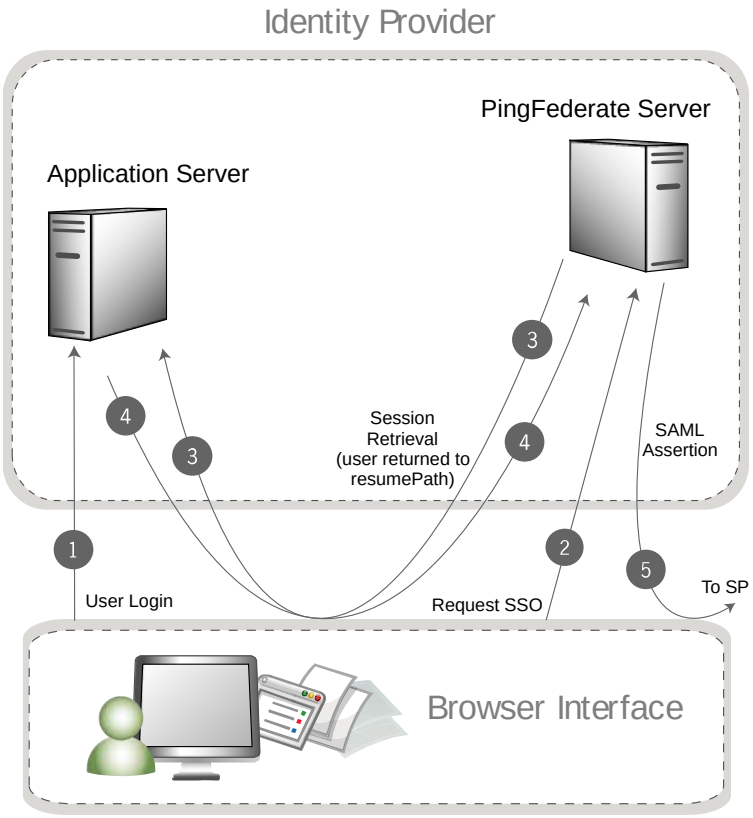
```
java.util.Map lookupAuthN(javax.servlet.http.HttpServletRequest req,
    javax.servlet.http.HttpServletResponse resp,
    java.lang.String partnerSpEntityId,
    AuthnPolicy authnPolicy,
    java.lang.String resumePath)
    throws AuthnAdapterException, java.io.IOException
```

PingFederate invokes the `lookupAuthN()` method of your IdP adapter to look up user-session information to handle an SSO request. This method is invoked regardless of whether the SSO request is IdP- or SP-initiated.

In most implementations, a user's session information or a reference to it can be communicated to PingFederate via the `HttpServletRequest`, which is passed to the `lookupAuthN()` method. The parameter is usually in the form of a cookie or query parameter set by the IdP application.

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.

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.
4. The adapter returns session information along with the `resumePath` parameter.
5. PingFederate generates a SAML assertion and sends it to the SP's SAML gateway.

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, you must define the following:

- Adapter UI Descriptor
- Configuration Retrieval
- Session Creation
- Session Logout
- Account Linking (if configured in PingFederate for an IdP partner)

Adapter UI Descriptor

```
AuthnAdapterDescriptor getAdapterDescriptor()
```

As with an IdP adapter, an SP adapter presents UI controls in the PingFederate administrative console at deployment time (see "Adapter UI Descriptor" on page 7). There is only one difference between the IdP and SP methods: the IdP-adapter option to configure a SAML Authentication Context is not applicable to an SP adapter.

Configuration Retrieval

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

An SP adapter receives the configuration set in the PingFederate administrative console in the same way as an IdP adapter (see "Configuration Retrieval" on page 7).

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 raw assertion.

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.

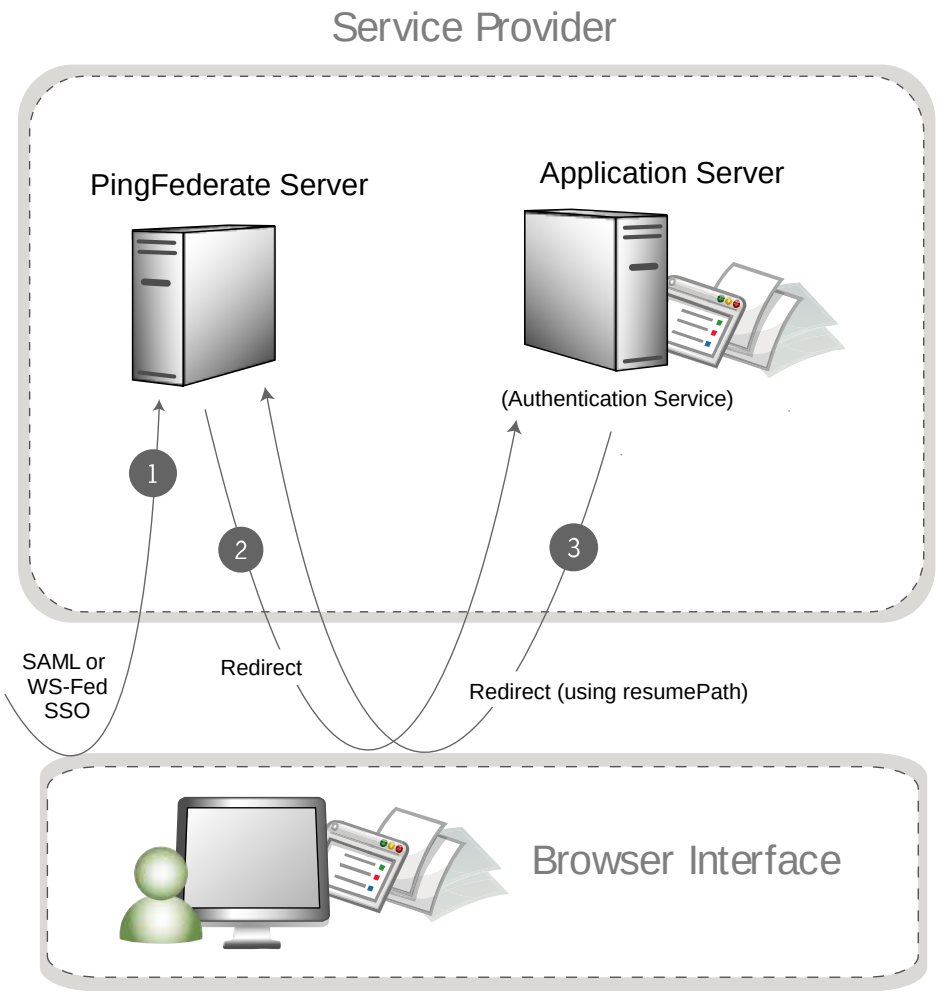
Account Linking

```
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, usually from a cookie or query parameter. The `String` object returned from the `lookupLocalUserId()` method should be a local user identifier.

Token Processor Implementation

You create a token-processor implementation by implementing the `TokenProcessor` interface. The Java packages needed, at a minimum, for implementing this interface are:

- `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, you must define the following:

- UI Configuration Descriptor
- Configuration Retrieval
- Token Processing

UI Configuration Descriptor

```
TokenProcessorDescriptor getPluginDescriptor()
```

Deploying your token processor requires configuration of a token-processor instance in the PingFederate administrative console. The `TokenProcessor` interface includes a `getPluginDescriptor()` method which returns an `TokenProcessorDescriptor`. Your token-processor implementation populates the `TokenProcessorDescriptor` with `FieldDescriptors` presented as UI controls to the administrator through the PingFederate administrative console.

The `api-usage-example` for an SP adapter provided with the SDK shows how to use most of the configuration `FieldDescriptors`, data `Validators`, and `Actions`. These implementations generally apply to token-processor implementations as well.

Configuration Retrieval

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

During processing of a STS transaction, a token-processor instance must reference its configuration as set by the administrator in the PingFederate UI. The `TokenProcessor.configure()` method (inherited from `ConfigurablePlugin`) provides access to this data. During transaction processing, PingFederate calls this method and passes in a `Configuration` object. The `Configuration` object provides access to the configuration values.

The `api-usage-example` SP adapter provided with the SDK shows how to use the `configure()` method to retrieve an adapter instance configuration. Once your implementation captures the configuration values, the token-translator instance can use them during token processing or generation.

Token Processing

```
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 at a minimum extend type `SecurityToken`. Type Binary`SecurityToken` is 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 by implementing the `TokenGenerator` interface. The Java packages needed, at a minimum, for implementing this interface are:

- `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, you must define the following:

- UI Configuration Descriptor
- Configuration Retrieval
- Token Generation

UI Configuration Descriptor

`PluginDescriptor getPluginDescriptor()`

Deploying your token generator requires configuration of a token-generator instance in the PingFederate administrative console. The `TokenGenerator` interface includes a `getPluginDescriptor()` method (inherited from `DescribablePlugin`) which returns an `PluginDescriptor`. Your token-generator implementation populates the `PluginDescriptor` with `FieldDescriptors` presented as UI controls to the administrator through the PingFederate administrative console.

The `api-usage-example` for an SP adapter provided with the SDK shows how to use most of the configuration `FieldDescriptors`, data `Validators`, and `Actions`. These implementations generally apply to token-generator implementations as well.

Configuration Retrieval

`void configure(Configuration configuration)`

During processing of a STS transaction, a token-generator instance must reference its configuration as set by the administrator in the PingFederate UI. The `TokenGenerator.configure()` method (inherited from `ConfigurablePlugin`) provides access to this data. During transaction processing, PingFederate calls this method and passes in a `Configuration` object. The `Configuration` object provides access to the configuration values.

The `api-usage-example` SP adapter provided with the SDK shows how to use the `configure()` method to retrieve an adapter-instance configuration. Once your implementation captures the configuration values, the token instance can use them during token processing or generation.

Token Generation

`SecurityToken generateToken(TokenContext attributeContext)`

PingFederate optionally invokes the `generateToken()` method during the processing of an STS request to perform necessary operations for generation of a security token. 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.

Custom Data-Source Implementation

Out of the box, PingFederate provides the capability of querying data sources 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.

Data sources are typically used to retrieve additional user attributes using parameters about the user who invokes an SSO transaction at runtime. An IdP may also use data sources in response to an attribute-query request to return user attributes to an SP.

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

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

For each implementation, you must define the following at a minimum:

- A Data-Source UI descriptor
- Configuration Retrieval
- Connection Testing
- Available Fields Retrieval
- Data-Source Query Handling

Data-Source UI Descriptor

`SourceDescriptor getSourceDescriptor()`

An adapter requires configuration fields in the PingFederate administrative console. The UI descriptor for custom data-source implementations works in much the same way as the same method does for adapters (see "Adapter UI Descriptor" on page 7). An `AdapterConfigurationGuiDescriptor` is populated with objects that will appear as controls in the PingFederate administrative console when a custom data source is deployed. The `AdapterConfigurationGuiDescriptor` is passed into the constructor of the `SourceDescriptor` that is later returned from the `getSourceDescriptor()` method.

Filters provide the necessary information to locate a user record at runtime. You create the filter criteria selections needed for this lookup by passing back a `CustomDataSourceDriverDescriptor`, as a subclass 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 just as the other UI descriptors display fields. The PingFederate administrator can include runtime data by setting the value of the field descriptor using the `${<attribute name>}` format. The definition of the filter criteria and the values set at runtime are submitted to the `retrieveValues()` method discussed later (see "Data-Source" on page 16).

Configuration Retrieval

`void configure(Configuration configuration)`

A custom data source receives the configuration set in the PingFederate administrative console in the same way that an IdP adapter does—for more information, see "Configuration Retrieval" on page 7 and the `api-usage-example`.

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.

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 data-store 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 containing criteria to use for selecting a specific record based on data during runtime.

Important: The `filterConfiguration` object is set and populated with a list of fields only if the data source was defined with a `CustomDataSourceDriverDescriptor` (see "Data-Source UI Descriptor" on page 15). Each field contains a name/value pair that is set at runtime based on that field's configuration in the PingFederate administrative console. If the `CustomDataSourceDriverDescriptor` was not used in the definition of the data source, the `filterConfiguration` object is set to `NULL`.

This method returns a map of name-value pairs. This map contains the collection of attribute names passed into the method and their corresponding values retrieved from the query.

Building and Deploying Your Project

For an adapter, or custom data-source project:

1. Edit the `build.local.properties` file and set the `target-adapter.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-adapter.name` property as needed to build and deploy other projects.

---

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

To compile the project and create a JAR, enter:

```
ant jar-adapter
```

The SDK creates a deployment descriptor and places it in a JAR. The descriptor tells PingFederate what adapter implementations are contained in the JAR.

The descriptor(s) are placed in the `PF_INF` directory. If an IdP adapter is created, the adapter class name is placed in the `idp-authn-adapters` file. If an SP adapter is created, its class name is placed in the `sp-authn-adapters` file. If a custom data-source driver is created, the class name is placed in the `custom-drivers` file.

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

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

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

```
ant deploy-adapter
```

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

For a token-translator project:

3. Edit the `build.local.properties` file and set the `target-token.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-token.name` property as needed to build and deploy other projects.

---

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

   To compile the project and create a JAR, enter:

   ```
   ant jar-token
   ```

   The SDK creates a deployment descriptor and places it in a JAR. The descriptor tells PingFederate what token-translator implementations are contained in the JAR.

   The descriptor(s) are placed in the `PF_INF` directory. If a token processor is created, its class name is placed in the `token-processors` file. If a token generator is created, the token generator class name is placed in the `token-generator` file.

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

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

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

```
ant deploy-token
```

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 adapter or token-translator project 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.

For a service project:

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

   To compile the project and create a JAR, enter:

   ```
   ant jar-services
   ```

   Note: The jar is named `pf-services-extensions.jar` by default. You can change the name by adding a property named `svcs.jar.name` to the `build.local.properties` file.

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

   ```
   ant deploy-services
   ```

   This build target also deploys any JAR files in the `lib` directory of your subproject.

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

2. Edit the file `hivemodule.xml` in the `/server/default/conf/META-INF` directory of your PingFederate installation.

   Each `service-point` element specifies the implementation for a particular interface as indicated by the `interface` attribute. Set the value of the `class` attribute on the `create-instance` sub-element to the fully qualified class name of your custom implementation.

3. Restart the PingFederate server.

Hot-Deploying a Project

While developing an adapter, token translator, or custom data source, you may want to redeploy your project iteratively—after initial deployment and setup—to test added logic or configuration options. You can do this without restarting the server, if you wish, by following the applicable procedure below.

Note: Although in most situations hot-deploying your project should work, the behavior can be unpredictable under some circumstances. If in doubt, restart PingFederate and retest.

To hot-deploy an adapter or token translator:

1. Build and redeploy the project (see previous section).
2. On the Manage […] screen, click **Save**.

   (The full screen name, for example "Manage SP Adapter Instances", depends on what plug-in you are building.)

   This will re-initialize instances that you have previously configured. You can then navigate to the configuration pages to change or add configuration as necessary. Finally, you can exercise your new implementation by executing an SSO, SLO, or STS transaction.

To hot-deploy a custom data-source driver:

3. Build and redeploy the data-source project (see the previous section).
4. On the Manage Data Stores screen, click **Save**.

   This will re-initialize the custom data source (assuming you have already configured it in the administrative console).

5. Access each IdP or SP connection that you have configured to use the data source; click **Save** on each Summary screen.

   You can now navigate to the various data-store configuration screens to change or add configuration, or test any new processing logic by exercising the data store through an SSO request.

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-store driver. The SP adapter contained in the directory `sdk/adapters-src/api-usage-example` shows how to use a logger for your adapter.