Developer's Guide

/ OpenAM 12

Latest update: 12.0.4

Mark Craig
David Goldsmith
Gene Hirayama
Mike Jang
Chris Lee
Peter Major

ForgeRock AS
33 New Montgomery St., Suite 1500
San Francisco, CA 94105, USA
+1 415-599-1100 (US)
www.forgerock.com

Abstract

Guide to developing OpenAM client applications and service providers. OpenAM provides open source Authentication, Authorization, Entitlement and Federation software.

Table of Contents

Preface

This guide demonstrates how to handle sessions to permit single sign on and single log out in OpenAM client applications. This guide further demonstrates how to use the OpenAM APIs including both APIs for client applications, and also SPIs for authentication, policy, service management, delegation, and identity storage. Finally, this guide demonstrates how to write your own web policy agent.

1. Who Should Use this Guide

This guide is written for developers who adapt client applications to use OpenAM access management capabilities. It is also written for designers and developers extending and integrating OpenAM services for their organizations.

You do not need to be an OpenAM wizard to learn something from this guide, though a background in access management and developing web applications or developing for web and application servers can help. You can nevertheless get started with this guide, and then learn more as you go along.

2. Formatting Conventions

Most examples in the documentation are created in GNU/Linux or Mac OS X operating environments. If distinctions are necessary between operating environments, examples are labeled with the operating environment name in parentheses. To avoid repetition file system directory names are often given only in UNIX format as in `/path/to/server`, even if the text applies to `C:\path\to\server` as well.

Absolute path names usually begin with the placeholder `/path/to/`. This path might translate to `/opt/`, `C:\Program Files\`, or somewhere else on your system.

Command-line, terminal sessions are formatted as follows:

```
$ echo $JAVA_HOME
/path/to/jdk
```

Command output is sometimes formatted for narrower, more readable output even though formatting parameters are not shown in the command.

Program listings are formatted as follows:

```
class Test {
    public static void main(String [] args)  {
        System.out.println("This is a program listing.");
    }
}
```

3. Accessing Documentation Online

ForgeRock publishes comprehensive documentation online:

- The ForgeRock Knowledge Base offers a large and increasing number of up-to-date, practical articles that help you deploy and manage ForgeRock software.

  While many articles are visible to community members, ForgeRock customers have access to much more, including advanced information for customers using ForgeRock software in a mission-critical capacity.

- ForgeRock product documentation, such as this document, aims to be technically accurate and complete with respect to the software documented. It is visible to everyone and covers all product features and examples of how to use them.

4. Using the ForgeRock.org Site

The ForgeRock.org site has links to source code for ForgeRock open source software, as well as links to the ForgeRock forums and technical blogs.

If you are a *ForgeRock customer*, raise a support ticket instead of using the forums. ForgeRock support professionals will get in touch to help you.

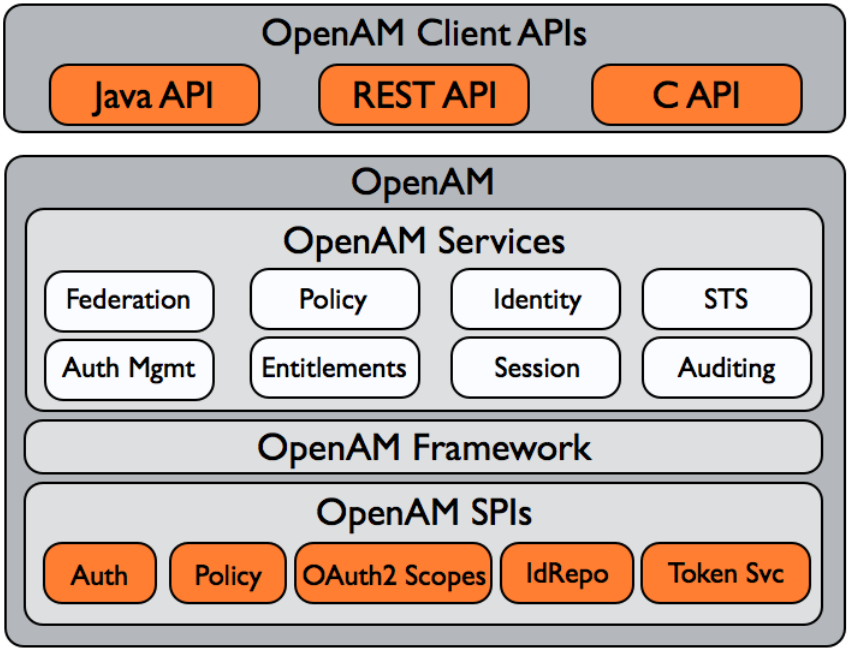
Chapter 1

OpenAM APIs and Protocols

Although policy agents and standards support make it possible for applications to use OpenAM for access management without changing your code, some deployments require tighter integration, or direct use of supported protocols and OpenAM APIs.

OpenAM supports a range of protocols and APIs that allow you not only to define specifically how access is managed in your client applications, but also to extend OpenAM capabilities to meet even those deployment requirements not yet covered in OpenAM.

This short chapter presents an overview of the APIs and protocols that OpenAM supports.



This guide primarily covers the OpenAM client APIs and SPIs, with emphasis on the Java APIs.

1.1. OpenAM APIs

OpenAM provides client application programming interfaces for a variety of needs.

- The OpenAM Java APIs provided through the OpenAM Java SDK let your Java and Java EE applications call on OpenAM for authentication, and authorization in both OpenAM and federated environments.

  Detailed reference information is provided in the *OpenAM Java SDK API Specification*.

- The C SDK also provides APIs for native applications, such as new web server policy agents. The C SDK is delivered with OpenAM for Linux, Solaris, and Windows platforms.

- OpenAM exposes a RESTful API that can return JSON or XML over HTTP, allowing you to access authentication, authorization, and identity services from your web applications using REST clients in the language of your choice.

1.2. OpenAM SPIs

OpenAM provides Java based service provider interfaces to let you extend services for the requirements of your particular deployment.

Some examples of the plugins you can write follow in the list below. This guide demonstrates how to implement such plugins.

- Custom OAuth 2.0 scopes plugins define how OpenAM playing the role of authorization server handles scopes, including what token information to return regarding scopes set when authorization was granted.

- Custom authentication plugins let OpenAM authenticate users against a new authentication service or an authentication service specific to your deployment

- Post authentication plugins perform additional processing at the end of the authentication process, but before the subject is authenticated. Post authentication plugins can for example store information about the authentication in the user's profile, or call another system for audit logging purposes.

- Policy evaluation plugins implement new policy conditions, send attributes from the user profile as part of a policy response, extend the definition of the subjects to whom the policy applies, or customize how policy management is delegated.

- Identity repository plugins let OpenAM employ a new or custom user data store, other than a directory server or JDBC-accessible database.

1.3. OpenAM, IPv4, and IPv6

OpenAM provides functionality for IPv4, IPv6, and a hybrid of the two. While the majority of the interaction is done on the backend, there are a few places where the GUI requires some inputs, such as setting up policy conditions. These areas follow the same standard that applies to IPv4 and IPv6. IPv4 uses a 32-bit integer value, with a dot-decimal system. IPv6 uses a hexadecimal system, and the eight groups of hexadecimal digits are separated by a colon.

Chapter 2

Developing Client Applications

Client applications access OpenAM services for authentication, authorization, and single sign on/single log out through the use of sessions. Client applications can also be allowed to manage authorization policies.

Client application integration with OpenAM can be coupled loosely, as in the case of an application running in a web server with an OpenAM policy agent to handle interaction with OpenAM service, more directly, as in the case where the client interacts with OpenAM over protocol, or tightly, as in the case of an application using the OpenAM Java or C API to interact with OpenAM services.

This part of the guide covers client interaction with OpenAM over supported protocols and using OpenAM APIs.

Chapter 3

RESTful Web Services

This chapter shows how to use the OpenAM RESTful interfaces for direct integration between web client applications and OpenAM.

3.1. About the RESTful APIs

Representational State Transfer(REST) is an architectural style that sets certain constraints for designing and building large-scale distributed hypermedia systems. As an architectural style, REST has very broad application. The designs of both HTTP 1.1 and also URIs follow RESTful principles. The World Wide Web is no doubt the largest and best known REST application. Many other web services also follow the REST architectural style. Examples include OAuth 2.0 and OpenID Connect 1.0.

ForgeRock Common REST (CREST) applies RESTful principles to define common verbs for HTTP-based APIs that access web resources and collections of web resources.

Native OpenAM REST APIs in version 11.0.0 and later use the CREST verbs. (In contrast, OAuth 2.0 and OpenID Connect 1.0 APIs follow the standards.) APIs covered in sections labelled (Legacy API) predate CREST, and do not use the CREST verbs.

When using a CREST API, you use the common verbs as query string parameters in resource and resource collection URIs.

CREST APIs use these verbs.

_create

Add a new resource.

Create maps to HTTP PUT (or POST).

_read

Retrieve a single resource.

Read maps to HTTP GET.

_update

Replace an existing resource.

Update maps to HTTP PUT.

_delete

Remove an existing resource.

Delete maps to HTTP DELETE.

_patch

Modify part of an existing resource

Patch maps to HTTP PATCH.

_action

Perform a predefined action.

Action maps to HTTP POST.

The generic _action verb extends what the API can do where none of the other standard CREST verbs fit, as in `_action=logout`.

_query

Search a collection of resources.

Query maps to HTTP GET.

CRUDPAQ is an acronym for the verbs. Notice that reserved words in CREST, such as the verbs, start with underscores (_).

In CREST, you can address resources in collections of resources by their unique identifiers, their IDs. IDs are exposed in the resource URIs as in `/users/id` and `/groups/id`. The ID is also in the "_id" field of the resource.

In CREST, resources are versioned using revision numbers. A revision is specified in the resource's "_rev" field. Revisions make it possible to figure out whether to apply changes without resource locking and without distributed transactions.

In CREST, you can explicitly request API versions. This means that OpenAM can continue to support older API versions as well as newer API versions as developers migrate their applications to take advantage of capabilities provided by newer APIs.

Interface Stability: Evolving in the *Administration Guide*

OpenAM offers RESTful APIs for these access and identity management operations:

- Authentication (login)
- Logout

- Cookie information
- Token attribute retrieval
- Token validation
- Logging
- Authorization
- OAuth 2.0 Authorization
- OpenID Connect 1.0
- User Self-Registration
- Resetting Forgotten Passwords
- Dashboard Service
- Identity Management (creating, reading, updating, deleting identities)
- Realm Management (creating, reading, updating, deleting realms)
- Secure Token Service (STS)

This chapter also includes a section on REST Status Codes.

In this chapter, long URLs are wrapped to fit the printed page, as some of the output is formatted for easier reading.

3.2. REST API Versioning

In OpenAM 12.0.0 and later, REST API features are assigned version numbers.

Providing version numbers in the REST API helps ensure compatibility between OpenAM releases. The version number of a feature increases when OpenAM introduces a non-backwards-compatible change that affects clients making use of the feature.

OpenAM provides versions for the following aspects of the REST API.

resource

Any changes to the structure or syntax of a returned response will incur a *resource* version change. For example changing `"errorMessage"` to `"message"` in a JSON response.

protocol

Any changes to the methods used to make REST API calls will incur a *protocol* version change. For example changing `_action` to `$action` in the required parameters of an API feature.

3.2.1. Supported REST API Versions

The REST API version numbers supported in OpenAM 12.0.4 are as follows:

Supported protocol versions

The *protocol* versions supported in OpenAM 12.0.4 are:

```
1.0
```

Supported resource versions

The *resource* versions supported in OpenAM 12.0.4 are shown in the following table.

Supported resource Versions

Base	End Point	Supported Versions
/json	/authenticate	1.1, 2.0
	/users	1.1, 2.0
	/groups	1.1, 2.0
	/agents	1.1, 2.0
	/realms	1.0
	/dashboard	1.0
	/sessions	1.1
	/serverinfo/*	1.1
	/users/*{user-id}*/devices/trusted	1.0
	/applications	1.0
	/policies	1.0
	/applicationtypes	1.0
	/conditiontypes	1.0
	/subjecttypes	1.0
	/subjectattributes	1.0
	/decisioncombiners	1.0
	/subjectattributes	1.0
/xacml	/policies	1.0
/frrest	/token	1.0
	/client	1.0

The *OpenAM Release Notes* section, *OpenAM Changes & Deprecated Functionality* in the *Release Notes* describes the differences between API versions.

3.2.2. Specifying an Explicit REST API Version

You can specify which version of the REST API to use by adding an `Accept-API-Version` header to the request, as in the following example, which is requesting *resource* version 2.0 and *protocol* version 1.0.

```
$ curl \
 --request POST \
 --header "Content-Type: application/json" \
 --header "X-OpenAM-Username: demo" \
 --header "X-OpenAM-Password: changeit" \
 --header "Accept-API-Version: resource=2.0, protocol=1.0" \
 https://openam.example.com:8443/openam/json/authenticate
```

You can configure the default behavior OpenAM will take when a REST call does not specify explicit version information. For more information, see *Configuring REST APIs* in the *Administration Guide* in the *OpenAM Administration Guide*.

3.2.3. REST API Versioning Messages

OpenAM provides REST API version messages in the JSON response to a REST API call. You can also configure OpenAM to return version messages in the response headers. See *Configuring REST APIs* in the *Administration Guide* in the *OpenAM Administration Guide*

Messages include:

- Details of the REST API versions used to service a REST API call.
- Warning messages if REST API version information is not specified or is incorrect in a REST API call.

The `resource` and `protocol` version used to service a REST API call are returned in the `Content-API-Version` header, as shown below.

```
$ curl \
 -i \
 --request POST \
 --header "Content-Type: application/json" \
 --header "X-OpenAM-Username: demo" \
 --header "X-OpenAM-Password: changeit" \
 --header "Accept-API-Version: resource=2.0, protocol=1.0" \
 https://openam.example.com:8443/openam/json/authenticate

HTTP/1.1 200 OK
Content-API-Version: protocol=1.0,resource=2.0
Server: Restlet-Framework/2.1.7
Content-Type: application/json;charset=UTF-8

{
 "tokenId":"AQIC5wM...TU3OQ*",
 "successUrl":"/openam/console"
}
```

If the default REST API version behavior is set to `None`, and a REST API call does not include the `Accept-API-Version` header, or does not specify a `resource` version, then a `400 Bad Request` status code is returned, as shown below.

```
$ curl \
 --header "Content-Type: application/json" \
 --header "Accept-API-Version: protocol=1.0" \
 https://openam.example.com:8443/openam/json/serverinfo/*

{
 "code":400,
 "reason":"Bad Request",
 "message":"No requested version specified and behavior set to NONE."
}
```

If a REST API call does include the `Accept-API-Version` header, but the specified `resource` or `protocol` version does not exist in OpenAM, then a `404 Not Found` status code is returned, as shown below.

```
$ curl \
 --header "Content-Type: application/json" \
 --header "Accept-API-Version: protocol=1.0, resource=999.0" \
 https://openam.example.com:8443/openam/json/serverinfo/*

{
 "code":404,
 "reason":"Not Found",
 "message":"Accept-API-Version: Requested version \"999.0\" does not match any routes."
}
```

Tip

For more information on setting the default REST API version behavior, see *Configuring REST APIs* in the *Administration Guide* in the *OpenAM Administration Guide*.

3.3. Token Encoding

Valid tokens in OpenAM requires configuration either in percent encoding or in *C66Encode* format. C66Encode format is encouraged. It is the default token format for OpenAM, and is used in this chapter. The following is an example token that has not been encoded:

```
AQIC5wM2LY4SfcztBbXvEAOuECbqMY3J4NW3byH6xwgkGE=@AAJTSQACMDE=#
```

This token includes reserved characters such as `+`, `/`, and `=` (The `@`, `#`, and `*` are not reserved characters per se, but substitutions are still required). To c66encode this token, you would substitute certain characters for others, as follows:

+ is replaced with **-**
/ is replaced with **_**
= is replaced with **.**

@ is replaced with *
# is replaced with *
* (first instance) is replaced with **@**
* (subsequent instances) is replaced with **#**

In this case, the translated token would appear as shown here:

```
AQIC5wM2LY4SfcztBbXvEAOuECbqMY3J4NW3byH6xwgkGE.*AAJTSQACMDE.*
```

3.4. Authentication and Logout

OpenAM provides REST APIs for authentication and for logout.

- Under `/json/authenticate` and `/json/sessions`, you find the newer JSON-based APIs.

  See "REST APIs for Authentication & Logout" below.

- Under `/identity/authenticate` and `/identity/logout`, you find the backwards-compatible, legacy API.

  See "Authentication & Logout (Legacy API)" below.

Tip

When authentication depends on the client IP address, and OpenAM lies behind a load balancer or proxy layer, configure the load balancer or proxy to send the address by using the `X-Forwarded-For` header, and configure OpenAM to consume and forward the header as necessary. For details, see the *Installation Guide* section, *Handling HTTP Request Headers* in the *Installation Guide*.

3.4.1. REST APIs for Authentication & Logout

The simplest user name/password authentication returns a `tokenId` that applications can present as a cookie value for other operations that require authentication. In this case use HTTP POST to prevent the web container from logging the credentials. Pass the user name in an `X-OpenAM-Username` header, and the password in an `X-OpenAM-Password` header.

```
$ curl \
 --request POST \
 --header "X-OpenAM-Username: demo" \
 --header "X-OpenAM-Password: changeit" \
 --header "Content-Type: application/json" \
 --data "{}" \
 https://openam.example.com:8443/openam/json/authenticate
{ "tokenId": "AQIC5w...NTcy*", "successUrl": "/openam/console" }
```

This "zero page login" mechanism works only for name/password authentication. If you include a POST body with the request, it must be an empty JSON string as shown in the example. Alternatively,

you can leave the POST body empty. Otherwise, OpenAM interprets the body as a continuation of an existing authentication attempt, one that uses a supported callback mechanism.

The authentication service at `/json/authenticate` supports callback mechanisms that make it possible to perform other types of authentication in addition to simple user name/password login.

Callbacks that are not completed based on the content of the client HTTP request are returned in JSON as a response to the request. Each callback has an array of output suitable for displaying to the end user, and input which is what the client must complete and send back to OpenAM. The default is still user name/password authentication.

```
$ curl \
 --request POST \
 --header "Content-Type: application/json" \
 https://openam.example.com:8443/openam/json/authenticate

{
    "authId": "...jwt-value...",
    "template": "",
    "stage": "DataStore1",
    "callbacks": [
        {
            "type": "NameCallback",
            "output": [
                {
                    "name": "prompt",
                    "value": " User Name: "
                }
            ],
            "input": [
                {
                    "name": "IDToken1",
                    "value": ""
                }
            ]
        },
        {
            "type": "PasswordCallback",
            "output": [
                {
                    "name": "prompt",
                    "value": " Password: "
                }
            ],
            "input": [
                {
                    "name": "IDToken2",
                    "value": ""
                }
            ]
        }
    ]
}
```

The "authId" value is a JSON Web Token (JWT) that uniquely identifies the authentication context to OpenAM, and so must also be sent back with the requests.

To respond to the callback, send back the JSON object with the missing values filled, as in this case where the user name is `demo` and the password is `changeit`.

```
$ curl \
 --request POST \
 --header "Content-Type: application/json" \
 --data '{ "authId": "...jwt-value...", "template": "", "stage": "DataStore1",
   "callbacks": [ { "type": "NameCallback", "output": [ { "name": "prompt",
   "value": " User Name: " } ], "input": [ { "name": "IDToken1", "value": "demo" } ] },
   { "type": "PasswordCallback", "output": [ { "name": "prompt", "value": " Password: " } ],
   "input": [ { "name": "IDToken2", "value": "changeit" } ] } ] }' \
 https://openam.example.com:8443/openam/json/authenticate

{ "tokenId": "AQIC5wM2...U3MTE4NA..*", "successUrl": "/openam/console" }
```

The response is a token ID holding the SSO Token value.

Alternatively, you can authenticate without requesting a session using the `noSession` query string parameter.

```
$ curl \
 --request POST \
 --header "Content-Type: application/json" \
 --data '{ "authId": "...jwt-value...", "template": "", "stage": "DataStore1",
   "callbacks": [ { "type": "NameCallback", "output": [ { "name": "prompt",
   "value": " User Name: " } ], "input": [ { "name": "IDToken1", "value": "demo" } ] },
   { "type": "PasswordCallback", "output": [ { "name": "prompt", "value": " Password: " } ],
   "input": [ { "name": "IDToken2", "value": "changeit" } ] } ] }' \
 https://openam.example.com:8443/openam/json/authenticate?noSession=true

{ "message": "Authentication Successful", "successUrl": "/openam/console" }
```

OpenAM can be configured to return a failure URL value when authentication fails. No failure URL is configured by default. The Default Failure Login URL can be configured for the Core in the *Administration Guide* authentication module. Alternatively, failure URLs can be configured per authentication chain, which your client can specify using the `service` parameter described below. On failure OpenAM then returns HTTP status code 401 Unauthorized, and the JSON in the reply indicates the failure URL.

```
$ curl \
 --request POST \
 --header "Content-Type: application/json" \
 --header "X-OpenAM-Username: demo" \
 --header "X-OpenAM-Password: badpassword" \
 https://openam.example.com:8443/openam/json/authenticate
{
  "code":401,
  "reason":"Unauthorized",
  "message":"Invalid Password!!",
  "failureUrl": "http://www.example.com/401.html"
}
```

To specify a realm in your request, first make sure that the name of your realm does not match an endpoint name to avoid any potential routing errors. Then, specify the realm in one of two ways. For example, if you have a realm titled "`myRealm`," you can use it in your request as follows:

- Using the realm in the URI to the endpoint (preferred method):

  ```
  https://openam.example.com:8443/openam/json/myRealm/authenticate
  ```

- Using the realm query string parameter:

  ```
  https://openam.example.com:8443/openam/json/authenticate?realm=myRealm
  ```

You can use the `authIndexType` and `authIndexValue` query string parameters as a pair to provide additional information about how you are authenticating. The `authIndexType` can be one of the following types:

composite

Set the value to a composite advice string.

level

Set the value to the authentication level.

module

Set the value to the name of an authentication module.

resource

Set the value to a URL protected by an OpenAM policy.

role

Set the value to an OpenAM role.

service

Set the value to the name of an authentication chain.

user

Set the value to an OpenAM user ID.

You can use the query string parameter, `sessionUpgradeSSOTokenId=tokenId`, to request session upgrade. For an explanation of session upgrade, see the *Administration Guide* section on, *Authentication Levels & Session Upgrade* in the *Administration Guide*.

OpenAM uses the following callback types depending on the authentication module in use.

- `ChoiceCallback`: Used to display a list of choices and retrieve the selected choice
- `ConfirmationCallback`: Used to ask for a confirmation such as Yes, No, or Cancel and retrieve the selection
- `HiddenValueCallback`: Used to return form values that are not visually rendered to the end user
- `HttpCallback`: Used for HTTP handshake negotiations
- `LanguageCallback`: Used to retrieve the locale for localizing text presented to the end user
- `NameCallback`: Used to retrieve a name string
- `PasswordCallback`: Used to retrieve a password value
- `RedirectCallback`: Used to redirect the client user-agent
- `ScriptTextOutputCallback`: Used to insert a script into the page presented to the end user. The script can, for example, collect data about the user's environment.
- `TextInputCallback`: Used to retrieve text input from the end user
- `TextOutputCallback`: Used to display a message to the end user
- `X509CertificateCallback`: Used to retrieve the content of an x.509 certificate

Authenticated users can log out with the token cookie value and an HTTP POST to `/json/sessions/?_action=logout`.

```
$ curl \
 --request POST \
 --header "iplanetDirectoryPro: AQIC5wM2...U3MTE4NA..*" \
 --header "Content-Type: application/json" \
 "https://openam.example.com:8443/openam/json/sessions/?_action=logout"

{"result":"Successfully logged out"}
```

3.4.2. Authentication & Logout (Legacy API)

Interface Stability: Deprecated in the *Administration Guide*

Simple authentication with a user name and password returns a token.

```
$ curl \
 --request POST \
 --data "username=bjensen&password=hifalutin" \
 https://openam.example.com:8443/openam/identity/authenticate

token.id=AQIC5wM2LY4SfcxvdvHOXjtC_eWSs2RB54tgvgK8SuYi7aQ.*AAJTSQACMDE.*
```

If you must specify parameters as when authenticating to `/UI/Login`, you provide a percent encoded string of the parameters as the value of the `uri` parameter. The `/UI/Login` parameter deals with the `realm`, `module`, and `service` parameters. Setting the `client` parameter sets the user's IP address as part of the token following successful authentication. The default for the `client` parameter is the IP of the machine making the REST request.

```
$ curl \
 --request POST \
 --data "username=bjensen&password=hifalutin&uri=realm%3D%2F%26module%3DDataStore\
&client=192.168.1.1" \
 https://openam.example.com:8443/openam/identity/authenticate

token.id=AQIC5wM2LY4SfcxvdvHOXjtC_eWSs2RB54tgvgK8SuYi7aQ.*AAJTSQACMDE.*
```

You log out using the token to end the user session.

```
$ curl \
 --request POST \
 --data "subjectid=AQIC5w...*AAJTSQACMDE.*" \
 https://openam.example.com:8443/openam/identity/logout
```

3.5. Server Information

You can retrieve OpenAM server information by using HTTP GET on `/json/serverinfo/*`.

```
$ curl https://openam.example.com:8443/openam/json/serverinfo/*
{
    "domains": [
        ".example.com"
    ],
    "protectedUserAttributes": [],
    "cookieName": "iPlanetDirectoryPro",
    "forgotPassword": "false",
    "selfRegistration": "false",
    "lang": "en",
    "successfulUserRegistrationDestination": "default",
    "socialImplementations": [
        {
            "iconPath": "XUI/images/dashboard/facebook.png",
            "authnChain": "FacebookSocialAuthenticationService",
            "displayName": "Facebook",
            "valid": true
        }
    ],
    "referralsEnabled": "false",
    "zeroPageLogin": {
        "enabled": false,
        "refererWhitelist": [
            ""
        ],
        "allowedWithoutReferer": true
    },
    "FQDN": "openam.example.com"
}
```

3.6. Cookie Information

You can retrieve the cookie domains that OpenAM supports by HTTP GET on `/json/serverinfo/cookieDomains`.

```
$ curl https://openam.example.com:8443/openam/json/serverinfo/cookieDomains
{"domains":[".example.com"]}
```

You can retrieve the name of the cookie used for storing the session token. By default it is `iPlanetDirectoryPro`.

```
$ curl https://openam.example.com:8443/openam/identity/getCookieNameForToken
string=iPlanetDirectoryPro
```

You can also retrieve the name of the cookie used for storing the session token and the names of the cookies to forward with requests.

```
$ curl https://openam.example.com:8443/openam/identity/getCookieNamesToForward
string=iPlanetDirectoryPro
string=amlbcookie
```

3.7. Token Validation and Session Information

OpenAM provides REST APIs for validating SSO tokens and getting information about active sessions.

- Under `/json/sessions` you find the newer JSON-based API.

  See "Token Validation" and "Session Information" below.

- Under `/identity/isTokenValid` you find the backwards compatible, legacy APIs.

  See "Token Validation, Attribute Retrieval (Legacy API)" below.

3.7.1. Token Validation

To check over REST whether a token is valid, perform an HTTP POST to the resource URL, `/json/sessions/tokenId`, using the `validate` action as shown in the following example:

```
$ curl \
 --request POST \
 --header "Content-Type: application/json" \
 http://openam.example.com:8080/openam/json/sessions/AQIC5...?_action=validate
{"valid":true,"uid":"demo","realm":"/myRealm"}
```

An invalid token returns only information about the validity.

```
$ curl \
 --request POST \
 --header "Content-Type: application/json" \
 http://openam.example.com:8080/openam/json/sessions/AQIC5...?_action=validate
{"valid":false}
```

3.7.2. Session Information

You can use REST API calls to:

- Identify whether a session is active
- Check the maximum remaining amount of time a session has left before the user is required to reauthenticate

- Determine the length of time a session has been idle
- Reset a session's idle time to 0

For these REST endpoints, specify two SSO tokens. Provide the SSO token for the current authenticated user as the value of a header whose name is the name of the SSO token cookie, by default `iPlanetDirectoryPro`. Specify the SSO token about which you want information as the `tokenId` query string parameter of the REST URL. In the examples in this section, `AQIC5w...NTcy*` is the SSO token for the current authenticated user, while `BXCCq...NX*1*` is the token being queried.

To determine whether a session is active, perform an HTTP POST to the resource URL, `/json/sessions/`, using the `isActive` action as shown in the following example:

```
$ curl \
 --request POST \
 --header "Content-Type: application/json" \
 --header "iplanetDirectoryPro: AQIC5w...NTcy*" \
 http://openam.example.com:8080/openam/json/sessions/?_action=isActive&tokenId=BXCCq...NX*1*

{"active":true}
```

To check the maximum remaining time (in seconds) of a session, perform an HTTP POST to the resource URL, `/json/sessions/`, using the `getMaxTime` action as shown in the following example:

```
$ curl \
 --request POST \
 --header "Content-Type: application/json" \
 --header "iplanetDirectoryPro: AQIC5w...NTcy*" \
 http://openam.example.com:8080/openam/json/sessions/?_action=getMaxTime&tokenId=BXCCq...NX*1*

{"maxtime":7022}
```

To check the amount of time (in seconds) that a session has been idle, perform an HTTP POST to the resource URL, `/json/sessions/`, using the `getIdle` action as shown in the following example:

```
$ curl \
 --request POST \
 --header "Content-Type: application/json" \
 --header "iplanetDirectoryPro: AQIC5w...NTcy*" \
 http://openam.example.com:8080/openam/json/sessions/?_action=getIdle&tokenId=BXCCq...NX*1*

{"idletime":355}
```

To reset a session's idle time, perform an HTTP POST to the resource URL, `/json/sessions/`, using the `isActive` action with the `refresh=true` option as shown in the following example:

```
$ curl \
 --request POST \
 --header "Content-Type: application/json" \
 --header "iplanetDirectoryPro: AQIC5w...NTcy*" \
 http://openam.example.com:8080/openam/json/sessions/?_action=isActive&refresh=true&tokenId=BXCCq...NX*1*

{"active":true}
```

3.7.3. Token Validation, Attribute Retrieval (Legacy API)

Interface Stability: Deprecated in the *Administration Guide*

You check whether a token is valid as follows:

```
$ curl \
 --request POST \
 --data "tokenid=AQIC5w...*AAJTSQACMDE.*" \
 https://openam.example.com:8443/openam/identity/isTokenValid
boolean=true
```

An invalid token returns `boolean=false`.

```
$ curl \
 --request POST \
 --data "tokenid=INVALID" \
 https://openam.example.com:8443/openam/identity/isTokenValid
boolean=false
```

With a valid token, you can retrieve attributes about the subject. OpenAM returns a series of *name, value* pairs.

The newer API for retrieving user information is demonstrated in *Reading Identities*. What follows describes the legacy API.

```
$ curl \
 --request POST \
 --data "subjectid=AQIC5w...*AAJTSQACMDE.*" \
 https://openam.example.com:8443/openam/identity/attributes
userdetails.token.id=
 AQIC5wM2LY4SfcxuxIP0VnP2lVjs7ypEM6VDx6srk56CN1Q.*AAJTSQACMDE.*
userdetails.attribute.name=uid
userdetails.attribute.value=bjensen
userdetails.attribute.name=mail
userdetails.attribute.value=bjensen@example.com
userdetails.attribute.name=sn
userdetails.attribute.value=Jensen
userdetails.attribute.name=userpassword
userdetails.attribute.value={SSHA}rhusOfYpkapDWEHcfT2Y7y83LMuC++F4Abqvig==
userdetails.attribute.name=cn
userdetails.attribute.value=Babs Jensen
userdetails.attribute.value=Barbara Jensen
userdetails.attribute.name=givenname
userdetails.attribute.value=Barbara
userdetails.attribute.name=dn
userdetails.attribute.value=uid=bjensen,ou=people,dc=example,dc=com
userdetails.attribute.name=telephonenumber
userdetails.attribute.value=+1 408 555 1862
userdetails.attribute.name=objectclass
userdetails.attribute.value=organizationalPerson
```

```
userdetails.attribute.value=person
userdetails.attribute.value=posixAccount
userdetails.attribute.value=inetOrgPerson
userdetails.attribute.value=krbprincipalaux
userdetails.attribute.value=krbTicketPolicyAux
userdetails.attribute.value=top
```

You can specify attributes to limit what you retrieve.

```
$ curl "https://openam.example.com:8443/openam/identity/attributes?\
subjectid=AQIC5wM2LY4SfcxuxIP0VnP2lVjs7ypEM6VDx6srk56CN1Q.*AAJTSQACMDE.*\
&attributenames=mail\
&attributenames=uid"
userdetails.token.id=
 AQIC5wM2LY4SfcxuxIP0VnP2lVjs7ypEM6VDx6srk56CN1Q.*AAJTSQACMDE.*
userdetails.attribute.name=uid
userdetails.attribute.value=bjensen
userdetails.attribute.name=mail
userdetails.attribute.value=bjensen@example.com
```

When retrieving attributes, you can refresh the session thus setting the idle time to 0, by adding the boolean parameter `refresh=true` to the query string.

```
$ curl "https://openam.example.com:8443/openam/identity/attributes?\
subjectid=AQIC5wM2LY4SfcxuxIP0VnP2lVjs7ypEM6VDx6srk56CN1Q.*AAJTSQACMDE.*\
&attributenames=cn\
&refresh=true"
userdetails.token.id=
 AQIC5wM2LY4SfcxuxIP0VnP2lVjs7ypEM6VDx6srk56CN1Q.*AAJTSQACMDE.*
userdetails.attribute.name=cn
userdetails.attribute.value=Babs Jensen
userdetails.attribute.value=Barbara Jensen
```

You can specify the following attributes to retrieve information about the user's session time limits and current session: `maxsessiontime` (maximum length of a session), `maxidletime` (maximum idle time allowed during a session), `idletime` (actual idle time during the current session), `timeleft` (actual time left in session). The unit for maximum times is minutes. The unit for actual times is seconds.

Also use the parameter `refresh=false` to avoid changing the `idletime` with your request.

```
$ curl \
--data "subjectid=AQIC5w....*AAJTSQACMDE.*\
 &attributenames=idletime\
 &attributenames=maxidletime\
 &attributenames=timeleft\
 &attributenames=maxsessiontime\
 &refresh=false" \
 https://openam.example.com:8443/openam/identity/attributes

userdetails.token.id=AQIC5w....*AAJTSQACMDE.*
userdetails.attribute.name=maxsessiontime
userdetails.attribute.value=120
userdetails.attribute.name=maxidletime
userdetails.attribute.value=30
userdetails.attribute.name=idletime
userdetails.attribute.value=664
userdetails.attribute.name=timeleft
userdetails.attribute.value=6319
```

3.8. REST Goto URL Validation

You can set valid goto URLs using the OpenAM console by following the instructions in "Configuring Valid goto URL Resources" in the *Administration Guide*.

To validate a goto URL over REST, use the endpoint: `/json/user?_action=validateGoto`.

```
   $ curl \
--request POST --header "Content-Type: application/json"
 \
--header "iPlanetDirectoryPro: AQIC5...ACMDE.*"
 \
--data "'{"goto":"http://www.example.com/"}' \
http://openam.example.com:8080/openam/json/users?_action=validateGoto
{"successURL":"http://www.example.com/"}
```

3.9. Logging

OpenAM supports access and audit logging for any request going to a CREST endpoint and writes the data to two files:

- **amRest.access**. Records accesses to a CREST endpoint, regardless of whether the request successfully reached the endpoint through policy authorization.

  An `amRest.access` example is as follows:

```
$ cat openam/openam/log/amRest.access

#Version: 1.0
#Fields: time  Data  LoginID  ContextID  IPAddr  LogLevel  Domain  LoggedBy  MessageID  ModuleName
NameID  HostName
"2011-09-14 16:38:17"   /home/user/openam/openam/log/ "cn=dsameuser,ou=DSAME Users,o=openam"
aa307b2dcb721d4201 "Not Available" INFO  o=openam   "cn=dsameuser,ou=DSAME Users,o=openam"
LOG-1  amRest.access  "Not Available"  192.168.56.2
"2011-09-14 16:38:17"  "Hello World"  id=bjensen,ou=user,o=openam 8a4025a2b3af291d01  "Not Available"
INFO  o=openam id=amadmin,ou=user,o=openam "Not Available" amRest.access "Not Available"
192.168.56.2
```

- **amRest.authz**. Records all CREST authorization results regardless of success. If a request has an entry in the `amRest.access` log, but no corresponding entry in `amRest.authz`, then that endpoint was not protected by an authorization filter and therefore the request was granted access to the resource.

  The `amRest.authz` file has a key field, the `Data` field, which specifies the authorization decision, resource, and type of action performed on that resource. The `Data` field has the following syntax:

```
("GRANT"||"DENY") > "RESOURCE | ACTION"

where
  "GRANT > " is prepended to the entry if the request was allowed
  "DENY  > " is prepended to the entry if the request was not allowed
  "RESOURCE" is "ResourceLocation | ResourceParameter"
     where
       "ResourceLocation" is the endpoint location (e.g., subrealm/applicationtypes)
       "ResourceParameter" is the ID of the resource being touched
        (e.g., myApplicationType) if applicable. Otherwise, this field is empty
        if touching the resource itself, such as in a query.

  "ACTION" is "ActionType | ActionParameter"
     where
       "ActionType" is "CREATE||READ||UPDATE||DELETE||PATCH||ACTION||QUERY"
       "ActionParameter" is one of the following depending on the ActionType:
          For CREATE: the new resource ID
          For READ: empty
          For UPDATE: the revision of the resource to update
          For DELETE: the revision of the resource to delete
          For PATCH: the revision of the resource to patch
          For ACTION: the actual action performed (e.g., "forgotPassword")
          For QUERY: the query ID if any
```

```
$ cat openam/openam/log/amRest.authz

#Version: 1.0
#Fields: time   Data  ContextID  LoginID  IPAddr  LogLevel  Domain  MessageID  LoggedBy  NameID
ModuleName    HostName
"2014-09-16 14:17:28"   /var/root/openam/openam/log/   7d3af9e799b6393301
"cn=dsameuser,ou=DSAME Users,dc=openam,dc=forgerock,dc=org" "Not Available" INFO
dc=openam,dc=forgerock,dc=org  LOG-1  "cn=dsameuser,ou=DSAME Users,dc=openam,dc=forgerock,dc=org"
"Not Available" amRest.authz    10.0.1.5
"2014-09-16 15:56:12"  "GRANT > sessions|ACTION|logout|AdminOnlyFilter"  d3977a55a2ee18c201
id=amadmin,ou=user,dc=openam,dc=forgerock,dc=org "Not Available" INFO  dc=openam,dc=forgerock,dc=org
OAuth2Provider-2  "cn=dsameuser,ou=DSAME Users,dc=openam,dc=forgerock,dc=org"  "Not Available"
amRest.authz    127.0.0.1
"2014-09-16 15:56:40"   "GRANT > sessions|ACTION|logout|AdminOnlyFilter"  eedbc205bf51780001
id=amadmin,ou=user,dc=openam,dc=forgerock,dc=org  "Not Available" INFO dc=openam,dc=forgerock,dc=org
OAuth2Provider-2  "cn=dsameuser,ou=DSAME Users,dc=openam,dc=forgerock,dc=org"  "Not Available"
amRest.authz    127.0.0.1
```

OpenAM also provides additional information in its debug notifications for accesses to any endpoint, depending on the message type (error, warning or message) including realm, user, and result of the operation.

You can also send OpenAM messages to log, specifying the message content and the log file in which to write your message. Logging takes a valid `appid` token for the subject with access to log the message, and also a `subjectid` token for the user whom the message concerns. If the tokens are valid and the access rights correct, your message ends up in the log specified.

```
$ curl "https://openam.example.com:8443/openam/identity/log?\
appid=AQIC5wM2LY4SfcwyCZkk-1JXzx6q1EzgagabHfBjMidb5jI.*AAJTSQACMDE.*\
&subjectid=AQIC5wM2LY4SfcxuxIP0VnP2lVjs7ypEM6VDx6srk56CN1Q.*AAJTSQACMDE.*\
&logname=amRest.access\
&message=Hello%20World"
```

3.10. REST Status Codes

OpenAM REST APIs respond to successful requests with HTTP status codes in the 2xx range. OpenAM REST APIs respond to error conditions with HTTP status codes in the 4xx and 5xx range. Status codes used are described in the following list.

200 OK

The request was successful and a resource returned, depending on the request. For example, a successful HTTP GET on `/users/myUser` returns a user profile and status code 200, whereas a successful HTTP DELETE returns `{"success","true"}` and status code 200.

201 Created

The request succeeded and the resource was created.

400 Bad Request

The request was malformed. Either parameters required by the action were missing, or as in the following example incorrect data was sent in the payload for the action.

```
$ curl \
 --request POST \
 --header "Content-Type: application/json" \
 --data '{"bad":"data"}' \
 https://openam.example.com:8443/openam/json/users?_action=forgotPassword

{
 "code":400,
 "reason":"Bad Request",
 "message":"Username or email not provided in request"
}
```

401 Unauthorized

The request requires user authentication as in the following example, which is missing an SSO Token value.

```
$ curl \
 --request POST \
 https://openam.example.com:8443/openam/json/sessions?_action=logout

{
 "code": 401,
 "reason": "Unauthorized",
 "message": "Access denied"
}
```

403 Forbidden

Access was forbidden during an operation on a resource as in the following example, which has a regular user trying to read the OpenAM administrator profile.

```
$ curl \
 --request POST \
 --header "X-OpenAM-Username: demo" \
 --header "X-OpenAM-Password: changeit" \
 https://openam.example.com:8443/openam/json/authenticate

{ "tokenId": "AQIC5w...YyMA..*" }

$ curl \
 --header "iplanetDirectoryPro: AQIC5w...YyMA..*" \
 https://openam.example.com:8443/openam/json/users/amadmin

{
 "code": 403,
 "reason": "Forbidden",
 "message": "Permission to perform the read operation denied to
             id=demo,ou=user,dc=openam,dc=forgerock,dc=org"
}
```

404 Not Found

The specified resource could not be found as in the following example, which is attempting to read a nonexistent user's profile.

```
$ curl \
 --header "iplanetDirectoryPro: AQIC5w...NTcy*" \
 https://openam.example.com:8443/openam/json/users/missing

{
 "code":404,
 "reason":"Not Found",
 "message":"Resource cannot be found."
}
```

405 Method Not Allowed

The HTTP method is not allowed for the requested resource.

406 Not Acceptable

The request contains parameters that are not acceptable as in the following example, which specifies an API version parameter that is not supported by OpenAM.

```
$ curl \
 --request POST \
 --header "Content-Type: application/json" \
 --header "X-OpenAM-Username: demo" \
 --header "X-OpenAM-Password: changeit" \
 --header "Accept-API-Version: protocol=1.0, resource=999.0" \
 https://openam.example.com:8443/openam/json/authenticate

{
 "code":406,
 "reason":"Not Acceptable",
 "message":"Accept-API-Version: Requested version \"999.0\" does not match any routes."
}
```

409 Conflict

The request would have resulted in a conflict with the current state of the resource. For example using the Forgot Password feature and specifying the user's email address as in the following example, where multiple users have the same email address.

```
$ curl \
 --request POST \
 --header "Content-Type: application/json" \
 --data '{"email":"demo@example.com"}' \
 https://openam.example.com:8443/openam/json/users?_action=forgotPassword

{
 "code":409,
 "reason":"Conflict",
 "message":"Multiple users found"
}
```

410 Gone

The requested resource is no longer available, and will not become available again. The URI returned for resetting a password may have expired as in the following example:

```
$ curl \
 --request POST \
 --header "Content-Type: application/json" \
 --data '{"username": "demo"}' \
 https://openam.example.com:8443/openam/json/users/?_action=forgotPassword

{
 "code":410,
 "reason":"Gone",
 "message":"Token not found"
}
```

415 Unsupported Media Type

The request is in a format not supported by the requested resource for the requested method as in the following example, which is attempting to pass basic authentication credentials as form-encoded data rather than query string parameters.

```
$ curl \
 --request POST \
 --data "username=demo&password=changeit" \
 https://openam.example.com:8443/openam/json/authenticate

...
HTTP Status 415
...
The server refused this request because the request entity is in a
format not supported by the requested resource for the requested method
...
```

500 Internal Server Error

The server encountered an unexpected condition which prevented it from fulfilling the request. This could be caused by an invalid configuration in the Email Service, or as in the following example the specified user account not having an associated email address to send the Forgot Password URI to.

```
$ curl \
 --request POST \
 --header "Content-Type: application/json" \
 --data '{"username": "demo"}' \
 https://openam.example.com:8443/openam/json/users/?_action=forgotPassword

{
 "code":500,
 "reason":"Internal Server Error",
 "message":"No email provided in profile."
}
```

501 Not Implemented

The resource does not support the functionality required to fulfill the request as in the following example, which is attempting to delete an entry as a delete action instead of using an HTTP DELETE request.

```
$ curl \
 --request POST \
 --header "iplanetDirectoryPro: AQIC5w...NTcy*" \
 https://openam.example.com:8443/openam/json/users/demo?_action=delete

{
 "code": 501,
 "reason": "Not Implemented",
 "message": "Actions are not supported for resource instances"
}
```

503 Service Unavailable

The requested resource was temporarily unavailable. The service may have been disabled, as in the following example, where the Forgot Password functionality has been disabled.

```
$ curl \
 --request POST \
 --header "Content-Type: application/json" \
 --data '{"username": "demo"}' \
 https://openam.example.com:8443/openam/json/users/?_action=forgotPassword

{
 "code":503,
 "reason":"Service Unavailable",
 "message":"Forgot password is not accessible."
}
```

Chapter 4

RESTful Authorization and Policy Management Services

This chapter shows how to use the OpenAM RESTful interfaces for authorization and policy management.

4.1. About the REST Policy Endpoints

OpenAM provides REST APIs both for requesting policy decisions, and also for administering policy definitions.

- Under `/json[/realm]/policies`, you find the newer JSON-based APIs for policy management and evaluation.
- Under `/json[/realm]/applications` and `/json/applicationtypes` you find JSON-based APIs for administering applications and reading application types.
- Under `/json/conditiontypes` you find a JSON-based API for viewing what types of conditions you can use when defining policies.
- Under `/json/subjecttypes` you find a JSON-based API for viewing what types of subjects you can use when defining policies.
- Under `/json/subjectattributes` you find a JSON-based API for viewing subjects' attributes you can use when defining response attributes in policies.
- Under `/json/decisioncombiners` you find a JSON-based API for viewing implementations you can use when defining policies to specify how to combine results when multiple policies apply.
- Under `/json[/realm]/referrals` you find a JSON-based API for managing policy referrals.
- Under `/identity/authorize` and `/ws/1/entitlement/`, you find the backwards-compatible, legacy APIs. See "Authorization (Legacy API)" below.

4.2. Requesting Policy Decisions

You can request policy decisions from OpenAM by using the REST APIs described in this section. OpenAM evaluates requests based on the context and the policies configured, and returns decisions

that indicate what actions are allowed or denied, as well as any attributes or advice for the resources specified.

To request decisions for specific resources, see "Requesting Policy Decisions For Specific Resources".

To request decisions for a resource and all resources beneath it, see "Requesting Policy Decisions For a Tree of Resources".

4.2.1. Requesting Policy Decisions For Specific Resources

This section shows how you can request a policy decision over REST for specific resources.

To request policy decisions for specific resources, perform an HTTP POST using the evaluation action to the appropriate path under the URI where OpenAM is deployed, `/json[/realm][/ subrealm]/policies?_action=evaluate`, where *realm* and *subrealm* optionally specifies the realm. The payload for the HTTP POST is a JSON object that specifies at least the resources, and takes the following form.

```
{
    "resources": [
        "resource1",
        "resource2",
        ...,
        "resourceN"
    ],
    "application": "defaults to iPlanetAMWebAgentService if not specified",
    "subject": {
        "ssoToken": "SSO token ID string",
        "jwt": "JSON Web Token string",
        "claims": {
            "key": "value",
            ...
        }
    },
    "environment": {
        "optional key1": [
            "value",
            "another value",
            ...
        ],
        "optional key2": [
            "value",
            "another value",
            ...
        ],
        ...
    }
}
```

The values for the fields shown above are explained below:

`"resources"`

This required field specifies the list of resources for which to return decisions.

For example, when using the default application, `"iPlanetAMWebAgentService"`, you can request decisions for resource URLs.

```
{
    "resources": [
        "http://www.example.com/index.html",
        "http://www.example.com/do?action=run"
    ]
}
```

`"application"`

This field holds the name of the application, and defaults to `"iPlanetAMWebAgentService"` if not specified.

For more on applications, see "Defining Applications".

`"subject"`

This optional field holds an object that represents the subject. You can specify one or more of the following keys. If you specify multiple keys, the subject can have multiple associated principals, and you can use subject conditions corresponding to any type in the request.

"ssoToken"

The value is the SSO token ID string for the subject, returned for example on successful authentication as described in *REST APIs for Authentication & Logout*.

"jwt"

The value is a JWT string.

"claims"

The value is an object (map) of JWT claims to their values.

If you do not specify the subject, OpenAM uses the SSO token ID of the subject making the request.

`"environment"`

This optional field holds a map of keys to lists of values.

If you do not specify the environment, the default is an empty map.

The example below requests policy decisions for two URL resources. The `iPlanetDirectoryPro` header sets the SSO token for a user who has access to perform the operation.

```
$ curl \
 --request POST \
 --header "iPlanetDirectoryPro: AQIC5..." \
 --header "Content-Type: application/json" \
```

```
 --data '{
    "resources": [
        "http://www.example.com/index.html",
        "http://www.example.com/do?action=run"
    ],
    "application": "iPlanetAMWebAgentService"
 }' \
 https://openam.example.com:8443/openam/json/policies?_action
    =evaluate
[ {
  "resource" : "http://www.example.com/do?action=run",
  "actions" : {
  },
  "attributes" : {
  },
  "advices" : {
    "AuthLevelConditionAdvice" : [ "3" ]
  }
}, {
  "resource" : "http://www.example.com/index.html",
  "actions" : {
    "POST" : false,
    "GET" : true
  },
  "attributes" : {
    "cn" : [ "demo" ]
  },
  "advices" : {
  }
} ]
```

In the JSON list of decisions returned for each resource, OpenAM includes these fields.

`"resource"`

A resource specified in the request.

The decisions returned are not guaranteed to be in the same order as the resources were requested.

`"actions"`

A map of action name keys to Boolean values that indicate whether the action is allowed (`true`) or denied (`false`) for the specified resource.

In the example, for resource `http://www.example.com:80/index.html` HTTP GET is allowed, whereas HTTP POST is denied.

`"attributes"`

A map of attribute names to their values, if any response attributes are returned according to applicable policies.

In the example, the policy that applies to `http://www.example.com:80/index.html` causes that the value of the subject's "cn" profile attribute to be returned.

`"advices"`

A map of advice names to their values, if any advice is returned according to applicable policies.

The `"advices"` field can provide hints regarding what OpenAM needs to take the authorization decision.

In the example, the policy that applies to `http://www.example.com:80/do?action=run` requests that the subject be authenticated at an authentication level of at least 3.

```
{
    "advices": {
        "AuthLevelConditionAdvice": [
            "3"
        ]
    }
}
```

See "Policy Decision Advice" for details.

You can use the query string parameters `_prettyPrint=true` to make the output easier to read, and `_fields=field-name[,field-name...]` to limit the fields returned in the output.

4.2.2. Policy Decision Advice

When OpenAM returns a policy decision, the JSON for the decision can include an "advices" field. This field contains hints for the policy enforcement point.

```
{
    "advices": {
        "type": [
            "advice"
        ]
    }
}
```

The "advices" returned depend on policy conditions. For more information about OpenAM policy conditions, see "Managing Policies".

This section shows examples of the different types of policy decision advice and the conditions that cause OpenAM to return the advice.

`"AuthLevel"` and `"LEAuthLevel"` condition failures can result in advice showing the expected or maximum possible authentication level. For example, failure against the following condition:

```
{
    "type": "AuthLevel",
    "authLevel": 2
}
```

Leads to this advice:

```
{
    "AuthLevelConditionAdvice": [
        "2"
    ]
}
```

An `"AuthScheme"` condition failure can result in advice showing one or more required authentication modules. For example, failure against the following condition:

```
{
    "type": "AuthScheme",
    "authScheme": [
        "HOTP"
    ],
    "applicationName": "iPlanetAMWebAgentService",
    "applicationIdleTimeout": 10
}
```

Leads to this advice:

```
{
    "AuthSchemeConditionAdvice": [
        "HOTP"
    ]
}
```

An `"AuthenticateToRealm"` condition failure can result in advice showing the name of the realm to which authentication is required. For example, failure against the following condition:

```
{
    "type": "AuthenticateToRealm",
    "authenticateToRealm": "MyRealm"
}
```

Leads to this advice:

```
{
    "AuthenticateToRealmConditionAdvice": [
        "/myRealm"
    ]
}
```

An `"AuthenticateToService"` condition failure can result in advice showing the name of the required authentication chain. For example, failure against the following condition:

```
{
    "type": "AuthenticateToService",
    "authenticateToService": "MyAuthnChain"
}
```

Leads to this advice:

```
{
    "AuthenticateToServiceConditionAdvice": [
        "MyAuthnChain"
    ]
}
```

A `"ResourceEnvIP"` condition failure can result in advice showing that indicates corrective action to be taken to resolve the problem. The advice varies, depending on what the condition tests. For example, failure against the following condition:

```
{
    "type": "ResourceEnvIP",
    "resourceEnvIPConditionValue": [
        "IF IP=[127.0.0.12] THEN authlevel=4"
    ]
}
```

Leads to this advice:

```
{
    "AuthLevelConditionAdvice": [
        "4"
    ]
}
```

Failure against a different type of `"ResourceEnvIP"` condition such as the following:

```
{
    "type": "ResourceEnvIP",
    "resourceEnvIPConditionValue": [
        "IF IP=[127.0.0.11] THEN service=MyAuthnChain"
    ]
}
```

Leads to this advice:

```
{
    "AuthenticateToServiceConditionAdvice": [
        "MyAuthnChain"
    ]
}
```

A `"Session"` condition failure can result in advice showing that access has been denied because the user's session has been active longer than allowed by the condition. The advice will also show if the the user's session was terminated and reauthentication is required. For example, failure against the following condition:

```
{
    "type": "Session",
    "maxSessionTime": "10",
    "terminateSession": false
}
```

Leads to this advice:

```
{
    "SessionConditionAdvice": [
        "deny"
    ]
}
```

When policy evaluation denials occur against the following conditions, OpenAM does not return any advice:

- `IPv4`
- `IPv6`
- `LDAPFilter`
- `OAuth2Scope`
- `SessionProperty`
- `SimpleTime`

4.2.3. Requesting Policy Decisions For a Tree of Resources

This section shows how you can request policy decisions over REST for a resource and all other resources in the subtree beneath it.

To request policy decisions for a tree of resources, perform an HTTP POST using the evaluation action to the appropriate path under the URI where OpenAM is deployed, `/json[/realm]/policies?`

`_action=evaluateTree`, where *realm* optionally specifies the realm. The payload for the HTTP POST is a JSON object that specifies at least the root resource, and takes the following form.

```
{
    "resource": "resource string",
    "application": "defaults to iPlanetAMWebAgentService if not specified",
    "subject": {
        "ssoToken": "SSO token ID string",
        "jwt": "JSON Web Token string",
        "claims": {
            "key": "value",
            ...
        }
    },
    "environment": {
        "optional key1": [
            "value",
            "another value",
            ...
        ],
        "optional key2": [
            "value",
            "another value",
            ...
        ],
        ...
    }
}
```

The values for the fields shown above are explained below:

`"resource"`

This required field specifies the root resource for the decisions to return.

For example, when using the default application, `"iPlanetAMWebAgentService"`, you can request decisions for resource URLs.

```
{
    "resource": "http://www.example.com/"
}
```

`"application"`

This field holds the name of the application, and defaults to `"iPlanetAMWebAgentService"` if not specified.

For more on applications, see "Defining Applications".

`"subject"`

This optional field holds an object that represents the subject. You can specify one or more of the following keys. If you specify multiple keys, the subject can have multiple associated principals, and you can use subject conditions corresponding to any type in the request.

"ssoToken"

The value is the SSO token ID string for the subject, returned for example on successful authentication as described in, *REST APIs for Authentication & Logout*.

"jwt"

The value is a JWT string.

"claims"

The value is an object (map) of JWT claims to their values.

If you do not specify the subject, OpenAM uses the SSO token ID of the subject making the request.

`"environment"`

This optional field holds a map of keys to lists of values.

If you do not specify the environment, the default is an empty map.

The example below requests policy decisions for `http://www.example.com/`. The `iPlanetDirectoryPro` header sets the SSO token for a user who has access to perform the operation, and the subject takes the SSO token of the user who wants to access a resource.

```
$ curl \
 --request POST \
 --header "iPlanetDirectoryPro: AQIC5...NDU1*" \
 --header "Content-Type: application/json" \
 --data '{
    "resource": "http://www.example.com/",
    "subject": { "ssoToken": "AQIC5...zE4*" }
 }' \
 https://openam.example.com:8443/openam/json/policies?_action=evaluateTree
[ {
  "resource" : "http://www.example.com/",
  "actions" : {
    "GET" : true,
    "OPTIONS" : true,
    "HEAD" : true
  },
  "attributes" : {
  },
  "advices" : {
  }
}, {
  "resource" : "http://www.example.com/*",
  "actions" : {
    "POST" : false,
```

```
    "PATCH" : false,
    "GET" : true,
    "DELETE" : true,
    "OPTIONS" : true,
    "HEAD" : true,
    "PUT" : true
  },
  "attributes" : {
    "myStaticAttr" : [ "myStaticValue" ]
  },
  "advices" : {
  }
}, {
  "resource" : "http://www.example.com/*?*",
  "actions" : {
    "POST" : false,
    "PATCH" : false,
    "GET" : false,
    "DELETE" : false,
    "OPTIONS" : true,
    "HEAD" : false,
    "PUT" : false
  },
  "attributes" : {
  },
  "advices" : {
    "AuthLevelConditionAdvice" : [ "3" ]
  }
} ]
```

Notice that OpenAM returns decisions not only for the specified resource, but also for matching resource names in the tree whose root is the specified resource.

In the JSON list of decisions returned for each resource, OpenAM includes these fields.

`"resource"`

A resource name whose root is the resource specified in the request.

The decisions returned are not guaranteed to be in the same order as the resources were requested.

`"actions"`

A map of action name keys to Boolean values that indicate whether the action is allowed (`true`) or denied (`false`) for the specified resource.

In the example, for matching resources with a query string only HTTP OPTIONS is allowed according to the policies configured.

`"attributes"`

A map of attribute names to their values, if any response attributes are returned according to applicable policies.

In the example, the policy that applies to `http://www.example.com:80/*` causes a static attribute to be returned.

`"advices"`

A map of advice names to their values, if any advice is returned according to applicable policies.

The `"advices"` field can provide hints regarding what OpenAM needs to take the authorization decision.

In the example, the policy that applies to resources with a query string requests that the subject be authenticated at an authentication level of at least 3.

Notice that with the `"advices"` field present, no `"advices"` appear in the JSON response.

```
{
    "advices": {
        "AuthLevelConditionAdvice": [ "3" ]
    }
}
```

You can use the query string parameters `_prettyPrint=true` to make the output easier to read, and `_fields=field-name[,field-name...]` to limit the fields returned in the output.

4.3. Managing Policies

Policy resources are represented in JSON and take the following form. Policy resources are built from standard JSON objects and values (strings, numbers, objects, arrays, `true`, `false`, and `null`).

```
{
   "name": "test",
   "active": true,
   "description": "A test policy",
   "resources": [
           "http://www.example.com:80/*"
   ],
   "applicationName": "application name",
   "actionValues": {
       "read": true,
       "write": false
   },
   "subject": {
       "a subject or": "a composite of subjects"
   },
   "condition": {
       "a condition or": "a composite of conditions"
   },
   "resourceAttributes": [
       {
```

```
            "type": "Static",
            "propertyName": "name",
            "propertyValues": [
                "value"
            ]
        },
        {
            "type": "User",
            "propertyName": "profile attribute",
        },
    ]
}
```

The values for the fields shown in the example are explained below:

`"name"`

String matching the name in the URL used when creating the policy by HTTP PUT or in the body when creating the policy by HTTP POST.

`"active"`

Boolean indicating whether OpenAM considers the policy active for evaluation purposes, defaults to `false`.

`"description"`

String describing the policy.

`"resources"`

List of the resource name pattern strings to which the policy applies.

`"applicationName"`

String application name, such as `"iPlanetAMWebAgentService"`, `"crestPolicyService"`, or some other application name.

`"actionValues"`

Set of string action names, each set to a boolean indicating whether the action is allowed.

Action values can also be expressed as numeric values. When using numeric values, use the value `0` for `false` and use any non-zero numeric value for `true`.

`"subject"`

Specifies the subject conditions to which the policy applies, where subjects can be combined by using the built-in types `"AND"`, `"OR"`, and `"NOT"`, and where subject implementations are pluggable.

Subjects are shown as JSON objects with `"type"` set to the name of the implementation (using a short name for all registered subject implementations), and also other fields depending on the implementation. The subject types registered by default include the following:

- `"AuthenticatedUsers"`, meaning any user that has successfully authenticated to OpenAM.

```
{
    "type": "AuthenticatedUsers"
}
```

- `"Identity"` to specify one or more users from an OpenAM identity repository:

```
{
   "type": "Identity",
   "subjectValues": [
       "uid=scarter,ou=People,dc=example,dc=com",
       "uid=ahall,ou=People,dc=example,dc=com"
   ]
}
```

  You can also use the `"Identity"` subject type to specify one or more groups from an identity repository:

```
{
    "type": "Identity",
    "subjectValues": [
        "cn=HR Managers,ou=Groups,dc=example,dc=com"
    ]
}
```

- `"JwtClaim"` to specify a claim in a user's JSON web token (JWT).

```
{
    "type": "JwtClaim",
    "claimName": "sub",
    "claimValue": "scarter"
}
```

- `"NONE"`, meaning never match any subject. The result is not that access is denied, but rather that the policy itself does not match and therefore cannot be evaluated in order to allow access.

The following example defines the subject either as the user Sam Carter from an OpenAM identity repository, or as a user with a JWT claim with a subject claim with the value scarter:

```
"subject": {
    "type": "OR",
    "subjects": [
       {
           "type": "Identity",
           "subjectValues": [
               "uid=scarter,ou=People,dc=example,dc=com"
           ]
       },
       {
           "type": "JwtClaim",
           "claimName": "sub",
           "claimValue": "scarter"
       }
   ]
}
```

To read a single subject type description, or to list all the available subject types, see "Viewing Subject Condition Types".

`"condition"`

Specifies environment conditions, where conditions can be combined by using the built-in types `"AND"`, `"OR"`, and `"NOT"`, and where condition implementations are pluggable.

Conditions are shown as JSON objects with `"type"` set to the name of the implementation (using a short name for all registered condition implementations), and also other fields depending on the implementation. The condition types registered by default include the following.

- `"AMIdentityMembership"` to specify a list of OpenAM users and groups.

```
{
    "type": "AMIdentityMembership",
    "amIdentityName": [
        "id=scarter,ou=People,dc=example,dc=com"
    ]
}
```

- `"AuthLevel"` to specify the authentication level.

```
{
    "type": "AuthLevel",
    "authLevel": 2
}
```

- `"AuthScheme"` to specify the authentication module used to authenticate and the application name, and to set a timeout for application authentication.

```
{
    "type": "AuthScheme",
    "authScheme": [
        "DataStore"
    ],
    "applicationName": "iPlanetAMWebAgentService",
    "applicationIdleTimeout": 10
}
```

- `"AuthenticateToRealm"` to specify the realm to which the user authenticated.

```
{
    "type": "AuthenticateToRealm",
    "authenticateToRealm": "MyRealm"
}
```

- `"AuthenticateToService"` to specify the authentication chain that was used to authenticate.

```
{
    "type": "AuthenticateToService",
    "authenticateToService": "MyAuthnChain"
}
```

- `"IPv4"` or `"IPv6"` to specify an IP address range from which the request originated.

```
{
   "type": "IPv4",
   "startIp": "127.0.0.1",
   "endIp": "127.0.0.255"
}
```

  You can also use the `"IPv4"` and `"IPv6"` conditions with the `"dnsName"` field to specify domain names from which the request originated. Omit `"startIp"` and `"endIp"` when using `"dnsName"`.

```
{
    "type": "IPv4",
    "dnsName": [
        "*.example.com"
    ]
}
```

- `"LDAPFilter"` to specify an LDAP search filter. The user's entry is tested against the search filter in the directory configured in the Policy Configuration Service.

```
{
    "type": "LDAPFilter",
    "ldapFilter": "(&(c=US)(preferredLanguage=en-us))"
}
```

- `"LEAuthLevel"` to specify a maximum acceptable authentication level.

```
{
    "type": "LEAuthLevel",
    "authLevel": 2
}
```

- `"OAuth2Scope"` to specify a list of attributes that must be present in the user profile.

```
{
    "type": "OAuth2Scope",
    "requiredScopes": [
        "name",
        "address",
        "email"
    ]
}
```

- `"ResourceEnvIP"` to specify a complex condition such as whether the user is making a request from a given host and has authenticated with a given authentication level. For example:

```
{
    "type": "ResourceEnvIP",
    "resourceEnvIPConditionValue": [
        "IF IP=[127.168.10.*] THEN authlevel=4"
    ]
}
```

Entries must take the form of one or more IF...ELSE statements. If the IF statement is true, the THEN statement must also be true for the condition to be fulfilled. The IF statement can specify either IP to match the user's IP address, or dnsName to match their DNS name. The IP address can be IPv4 or IPv6 format, or a hybrid of the two, and can include wildcard characters.

The available parameters for the THEN statement are as follows:

`module`

The module that was used to authenticate the user, for example DataStore.

`service`

The authentication chain that was used to authenticate the user.

`authlevel`

The minimum required authentication level.

`role`

The role of the authenticated user.

`user`

The name of the authenticated user.

`redirectURL`

The URL from which the user was redirected.

`realm`

The realm to which the user authenticated.

- `"Session"` to specify how long the user's session has been active, and to terminate the session if deemed too old, such that the user must authenticate again.

```
{
    "type": "Session",
    "maxSessionTime": "10",
    "terminateSession": false
}
```

- `"SessionProperty"` to specify attributes set in the user's session.

```
{
    "type": "SessionProperty",
    "ignoreValueCase": true,
    "properties": {
        "CharSet": [
            "UTF-8"
        ],
        "clientType": [
            "genericHTML"
        ]
    }
}
```

- `"SimpleTime"` to specify a time range, where `"type"` is the only required field.

```
{
    "type": "SimpleTime",
    "startTime": "07:00",
    "endTime": "19:00",
    "startDay": "mon",
    "endDay": "fri",
    "startDate": "2015:01:01",
    "endDate": "2015:12:31",
    "enforcementTimeZone": "GMT+0:00"
}
```

The following example defines the condition as neither Saturday or Sunday, nor certain client IP addresses.

```
{
    "type": "NOT",
    "condition": {
        "type": "OR",
        "conditions": [
            {
                "type": "SimpleTime",
                "startDay": "sat",
                "endDay": "sun",
                "enforcementTimeZone": "GMT+8:00"
            },
            {
                "type": "IPv4",
                "startIp": "192.168.0.1",
                "endIp": "192.168.0.255"
            }
        ]
    }
}
```

To read a single condition type description, or to list all the available condition types, see "Viewing Environment Condition Types".

`"resourceAttributes"`

List of attributes to return with decisions. These attributes are known as *response attributes*.

The response attribute provider is pluggable. The default implementation provides for statically defined attributes and for attributes retrieved from user profiles.

Attributes are shown as JSON objects with `"type"` set to the name of the implementation (by default either `"Static"` for statically defined attributes or `"User"` for attributes from the user profile), `"propertyName"` set to the attribute names. For static attributes, `"propertyValues"` holds the attribute values. For user attributes, `"propertyValues"` is not used; the property values are determined at evaluation time.

The examples above do not show the fields added to a policy by OpenAM to indicate when the policy was created and last updated, and by whom. Those field are `"createdBy"` and `"lastModifiedBy"`, which take strings holding universal identifier DNs as their values, and `"creationDate"` and `"lastModified"`, which take strings holding ISO-8601 timestamps.

4.3.1. Creating Policies

To create a policy, either perform an HTTP PUT indicating the full path to the resource and the name in the resource matching the name in the path, or perform an HTTP POST with the name to use specified in the resource.

The HTTP PUT form includes the policy definition as the JSON resource data, with the header `Content-Type: application/json` and uses the `If-None-Match: *` header.

The `iPlanetDirectoryPro` header sets the SSO token for a user who has access to perform the operation.

Do not use special characters within policy, application or referral names (for example, "my +referral") using the Policy Editor or REST endpoints as OpenAM returns a 400 Bad Request error. The special characters are: double quotes ("), plus sign (+), comma (,), less than (<), equals (=), greater than (>), backslash (\), and null (\u0000).

```
$ curl \
 --request PUT \
 --header "iPlanetDirectoryPro: AQIC5w..." \
 --header "If-None-Match: *" \
 --header "Content-Type: application/json" \
 --data '{
    "name": "example",
    "active": true,
    "description": "Example Policy",
    "resources": [
        "http://www.example.com:80/*",
        "http://www.example.com:80/*?*"
    ],
    "actionValues": {
        "POST": false,
        "GET": true
    },
    "subject": {
        "type": "Identity",
        "subjectValues": [
            "uid=scarter,ou=People,dc=example,dc=com"
        ]
    }
}' \
 https://openam.example.com:8443/openam/json/policies/example

{
  "name" : "example",
  "active" : true,
  "description" : "Example Policy",
  "applicationName" : "iPlanetAMWebAgentService",
  "actionValues" : {
    "POST" : false,
```

```
    "GET" : true
  },
  "resources" : [
    "http://www.example.com:80/*",
    "http://www.example.com:80/*?*"
  ],
  "subject" : {
   "type": "Identity",
    "subjectValues": [
      "uid=scarter,ou=People,dc=example,dc=com"
    ]
  },
  "lastModifiedBy" :
  "id=demo,ou=user,o=realm,ou=services,dc=openam,dc=forgerock,dc=org",
  "lastModified" : "2014-04-24T16:23:34Z",
  "createdBy" :
  "id=demo,ou=user,o=realm,ou=services,dc=openam,dc=forgerock,dc=org",
  "creationDate" : "2014-04-24T16:23:34Z"
}
```

You can use the query string parameters `_prettyPrint=true` to make the output easier to read, and `_fields=field-name[,field-name...]` to limit the fields returned in the output.

The HTTP POST form includes the policy definition as the JSON resource data, with the header `Content-Type: application/json` and uses the `_action=create` operation.

The `iPlanetDirectoryPro` header sets the SSO token for a user who has access to perform the operation.

```
$ curl \
 --request POST \
 --header "iPlanetDirectoryPro: AQIC5..." \
 --header "Content-Type: application/json" \
 --data '{
    "name": "example",
    "active": true,
    "description": "Example Policy",
    "resources": [
        "http://www.example.com:80/*",
        "http://www.example.com:80/*?*"
    ],
    "actionValues": {
        "POST": false,
        "GET": true
    },
    "subject": {
        "type": "Identity",
        "subjectValues": [
            "uid=scarter,ou=People,dc=example,dc=com"
        ]
    }
}' \
 https://openam.example.com:8443/openam/json/policies?_action=create

{
  "name" : "example",
  "active" : true,
```

```
  "description" : "Example Policy",
  "applicationName" : "iPlanetAMWebAgentService",
  "actionValues" : {
    "POST" : false,
    "GET" : true
  },
  "resources" : [
    "http://www.example.com:80/*",
    "http://www.example.com:80/*?*"
  ],
  "subject" : {
    "type": "Identity",
    "subjectValues": [
      "uid=scarter,ou=People,dc=example,dc=com"
    ]
  },
  "lastModifiedBy" :
   "id=demo,ou=user,o=realm,ou=services,dc=openam,dc=forgerock,dc=org",
  "lastModified" : "2014-04-29T07:33:54Z",
  "createdBy" :
   "id=demo,ou=user,o=realm,ou=services,dc=openam,dc=forgerock,dc=org",
  "creationDate" : "2014-04-29T07:33:54Z"
}
```

4.3.2. Reading Policies

To read a policy definition, perform an HTTP GET specifying the resource name.

The `iPlanetDirectoryPro` header sets the SSO token for a user who has access to perform the operation.

```
$ curl \
 --header "iPlanetDirectoryPro: AQIC5..." \
 https://openam.example.com:8443/openam/json/policies/example

{
  "name" : "example",
  "active" : true,
  "description" : "Example Policy",
  "applicationName" : "iPlanetAMWebAgentService",
  "actionValues" : {
    "POST" : false,
    "GET" : true
  },
  "resources" : [
    "http://www.example.com:80/*",
    "http://www.example.com:80/*?*"
  ],
  "subject" : {
    "type": "Identity",
    "subjectValues": [
      "uid=scarter,ou=People,dc=example,dc=com"
    ]
  },
  "lastModifiedBy" :
  "id=demo,ou=user,o=realm,ou=services,dc=openam,dc=forgerock,dc=org",
```

```
  "lastModified" : "2014-04-24T16:23:34Z",
  "createdBy" :
  "id=demo,ou=user,o=realm,ou=services,dc=openam,dc=forgerock,dc=org",
  "creationDate" : "2014-04-24T16:23:34Z"
}
```

You can use the query string parameters `_prettyPrint=true` to make the output easier to read, and `_fields=field-name[,field-name...]` to limit the fields returned in the output.

4.3.3. Updating Policies

To update a policy definition, perform an HTTP PUT specifying the resource name with the policy definition as the JSON resource data, and with the header `Content-Type: application/json`. This is essentially the same as creating a policy, but without the `If-None-Match: *` header.

The `iPlanetDirectoryPro` header sets the SSO token for a user who has access to perform the operation.

```
$ curl \
 --request PUT \
 --header "iPlanetDirectoryPro: AQIC5w..." \
 --header "Content-Type: application/json" \
 --data '{
    "name": "example",
    "active": true,
    "description": "Updated example policy",
    "resources": [
        "http://www.example.com:80/*",
        "http://www.example.com:80/*?*"
    ],
    "actionValues": {
        "POST": true,
        "GET": true
    },
    "subject": {
        "type": "Identity",
        "subjectValues": [
            "uid=scarter,ou=People,dc=example,dc=com"
        ]
    }
}' \
 https://openam.example.com:8443/openam/json/policies/example

{
  "name" : "example",
  "active" : true,
  "description" : "Updated example policy",
  "applicationName" : "iPlanetAMWebAgentService",
  "actionValues" : {
    "POST" : true,
    "GET" : true
  },
  "resources" : [
    "http://www.example.com:80/*",
    "http://www.example.com:80/*?*"
```

```
  ],
  "subject" : {
    "type": "Identity",
    "subjectValues": [
      "uid=scarter,ou=People,dc=example,dc=com"
    ]
  },
  "lastModifiedBy" :
  "id=demo,ou=user,o=realm,ou=services,dc=openam,dc=forgerock,dc=org",
  "lastModified" : "2014-04-24T16:44:01Z",
  "createdBy" :
  "id=demo,ou=user,o=realm,ou=services,dc=openam,dc=forgerock,dc=org",
  "creationDate" : "2014-04-24T16:23:34Z"
}
```

Tip

You can rename a policy by sending a new value in the *name* attribute of the JSON body, as shown below.

```
$ curl \
 --request PUT \
 --header "iPlanetDirectoryPro: AQIC5w..." \
 --header "Content-Type: application/json" \
 --data '{
     "name": "new-example",
     "active": true,
     "description": "Renamed example policy",
     "resources": [
         "http://www.example.com:80/*",
         "http://www.example.com:80/*?*"
     ],
     "actionValues": {
         "POST": true,
         "GET": true
     },
     "subject": {
         "type": "Identity",
         "subjectValues": [
             "uid=scarter,ou=People,dc=example,dc=com"
         ]
     }
 }' \
 https://openam.example.com:8443/openam/json/policies/example

{
    "name" : "new-example",
    "active" : true,
    "description" : "Renamed example policy",
    "applicationName" : "iPlanetAMWebAgentService",
    "actionValues" : {
        "POST" : true,
        "GET" : true
    },
    "resources" : [
        "http://www.example.com:80/*",
        "http://www.example.com:80/*?*"
```

```
    ],
    "subject" : {
        "type": "Identity",
        "subjectValues": [
            "uid=scarter,ou=People,dc=example,dc=com"
        ]
    },
    "lastModifiedBy" :
    "id=demo,ou=user,o=realm,ou=services,dc=openam,dc=forgerock,dc=org",
    "lastModified" : "2014-04-24T16:48:01Z",
    "createdBy" :
    "id=demo,ou=user,o=realm,ou=services,dc=openam,dc=forgerock,dc=org",
    "creationDate" : "2014-04-24T16:23:34Z"
}
```

You can use the query string parameters `_prettyPrint=true` to make the output easier to read, and `_fields=field-name[,field-name...]` to limit the fields returned in the output.

4.3.4. Deleting Policies

To delete a policy definition, perform an HTTP DELETE specifying the resource name.

The `iPlanetDirectoryPro` header sets the SSO token for a user who has access to perform the operation.

```
$ curl \
 --header "iPlanetDirectoryPro: AQIC5w..." \
 --request DELETE \
 https://openam.example.com:8443/openam/json/policies/myPolicy
{}
```

4.3.5. Listing Policies

To list policy definitions, perform an HTTP GET on the endpoint, setting the `_queryFilter` query string parameter.

```
$ curl \
 --header "iPlanetDirectoryPro: AQIC5w..." \
 https://openam.example.com:8443/openam/json/policies?_queryFilter=true

{
  "result" : [ ...policies... ],
  "resultCount" : 0,
  "pagedResultsCookie" : null,
  "remainingPagedResults" : -1
}
```

The `_queryFilter` parameter can take `true` to match every policy, `false` to match no policies, or a filter of the following form to match field values: `field operator value` where *field* represents the field name,

operator is the operator code, *value* is the value to match, and the entire filter is URL-encoded. Supported operators are as follows:

- `eq`: equals
- `ge`: greater than or equal to
- `gt`: greater than
- `le`: less than or equal to
- `lt`: less than

The *`field`* value can take the following values:

- `"name"` (string equality only)
- `"description"` (string equality only)
- `"applicationName"` (string equality only)
- `"createdBy"` (string equality only)
- `"lastModifiedBy"` (string equality only)
- `"creationDate"` (all comparisons are supported; the date is either an ISO-8601 string, or a integer number of seconds from the UNIX epoch)
- `"lastModified"` (all comparisons are supported; the date is either an ISO-8601 string, or a integer number of seconds from the UNIX epoch)

Filters can be composed of multiple expressions by a using boolean operator `AND`, and by using parentheses, `(expression)`, to group expressions. You must URL encode the filter expression in `_queryFilter=filter`.

You can use the query string parameters `_prettyPrint=true` to make the output easier to read, `_fields=field-name[,field-name...]` to limit the fields returned in the output.

You can use `_pageSize=integer` to limit the number of results returned, as shown in the following example that returns only the first of three policies.

```
$ curl \
 --header "iPlanetDirectoryPro: AQIC5w..." \
 https://openam.example.com:8443/openam/json/policies?_queryFilter=true\&_fields=name\&_pageSize=1
{
  "result" : [ {
    "name" : "My Other Policy"
  } ],
  "resultCount" : 1,
  "pagedResultsCookie" : null,
  "remainingPagedResults" : 2
}
```

You can use `_sortKeys=[+-]field[,field...]` to sort the results returned, where *field* represents a field in the JSON policy objects returned. Optionally use the `+` prefix to sort in ascending order (the default), or `-` to sort in descending order. The following example sorts the policy objects by their names.

```
$ curl \
  --header "iPlanetDirectoryPro: AQIC5w..." \
 https://openam.example.com:8443/openam/json/policies?_queryFilter=true\&_sortKeys=name
  {
  "result" : [ {
    "name" : "Another Example Policy",
    ...
  }, {
    "name" : "My Other Policy",
    ...
  }, {
    "name" : "Sample Policy",
    ...
  } ],
  "resultCount" : 3,
  "pagedResultsCookie" : null,
  "remainingPagedResults" : 0
}
```

4.4. XACML 3.0 Export and Import

OpenAM supports the ability to export policies to eXtensible Access Control Markup Language (XACML) 3.0-based formatted policy sets through its `/xacml/policies` REST endpoint. You can also import XACML 3.0 policy sets back into OpenAM by using the same endpoint. The endpoint's functionally is identical to that of the `ssoadm import-xacml` and `export-xacml` commands. For more information, see Importing and Exporting Policies in the *Administration Guide*

Note

OpenAM can only import XACML 3.0 policy sets that were either created by an OpenAM instance, or that have had minor manual modifications, due to the reuse of some XACML 3.0 parameters for non-standard information.

When exporting OpenAM policies to XACML 3.0 policy sets, OpenAM maps its policies to XACML 3.0 policy elements. The mappings are as follows:

OpenAM Policies to XACML Mappings

OpenAM Policy	XACML Policy
Policy Name	Policy ID
Description	Description
Current Time (yyyy.MM.dd.HH.mm.ss.SSS)	Version

OpenAM Policy	XACML Policy
xacml rule target	entitlement excluded resource names
Rule Deny Overrides	Rule Combining Algorithm ID
Any of: • Entitlement Subject • Resource Names • Application Names • Action Values	Target
Any of: • Application Name • Entitlement Name • Privilege Created By • Privilege Modified By • Privilege Creation Date • Privilege Last Modification Date	Variable Definitions
Single Level Permit/Deny Actions converted to Policy Rules	Rules

Note

XACML obligation is not supported. Also, only one XACML match is defined for each privilege action, and only one XACML rule for each privilege action value.

4.4.1. Exporting from OpenAM to XACML

OpenAM supports exporting policies into XACML 3.0 format. OpenAM only exports a policy set that contains policy definitions. No other types can be included in the policy set, such as sub-policy sets or rules. The policy set mapping is as follows:

Policy Set Mappings

OpenAM	XACML
Realm:<timestamp>(yyyy.MM.dd.HH.mm.ss.SSS)	PolicySetID
Current Time (yyyy.MM.dd.HH.mm.ss.SSS)	Version
Deny Overrides	Policy Combining Algorithm ID

OpenAM	XACML
No targets defined	Target

The export service is accessible at the `/xacml/policies` endpoint using a HTTP GET request at the following endpoint for the root realm or a specific realm:

```
http://openam.example.com:8080/openam/xacml/policies
http://openam.example.com:8080/openam/xacml/{realm}/policies

  where {realm} is the name of a specific realm
```

You can filter your XACML exports using query search filters. Note the following points about the search filters:

- **LDAP-based Searches**. The search filters follow the standard guidelines for LDAP searches as they are applied to the entitlements index in the LDAP configuration backend, located at: `ou=default,ou=OrganizationalConfig,ou=1.0,ou=sunEntitlementIndexes, ou=services,dc=openam,dc=forgerock,dc=org`.
- **Search Filter Format**. You can specify a single search filter or multiple filters in the HTTP URL parameters. The format for the search filter is as follows:

  ```
  [attribute name][operator][attribute value]
  ```

  If you specify multiple search filters, they are logically ANDed: the search results meet the criteria specified in all the search filters.

XACML Export Search Filter Format

Element	Description
Attribute Name	The name of the attribute to be searched for. The only permissible values are: `application`, `createdby`, `lastmodifiedby`, `creationdate`, `lastmodifieddate`, `name`, `description`.
Operator	The type of comparison operation to perform. • = Equals (text) • < Less Than or Equal To (numerical) • > Greater Than or Equal To (numerical)
Attribute Value	The matching value. Asterisk wildcards are supported.

To Export Policies

- Use the `/xacml/policies` endpoint to export the OpenAM entitlement policies into XACML 3.0 format. The following curl command exports the policies and returns the XACML response (truncated for display purposes).

```
$ curl \
  --request GET \
  --header "iPlanetDirectoryPro: AQIC5..." \
  --header "Content-Type: application/json" \
  http://openam.example.com:8080/openam/xacml/policies


<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<PolicySet xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
 PolicyCombiningAlgId="urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:deny-overrides"
 Version="2014.10.08.21.59.39.231" PolicySetId="/:2014.10.08.21.59.39.231">
 <Target/>
 <Policy RuleCombiningAlgId="urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:deny-overrides"
  Version="2014.10.08.18.01.03.626"
  PolicyId="Rockshop_Checkout_https://forgerock-rockshop.openrock.org:443/wp-login.php*?*">
  ...
```

To Export Policies with Search Filters

1. Use the `/xacml/policies` endpoint to export the policies into XACML 3.0 format with a search filter. This command only exports policies that were created by "amadmin".

```
$ curl \
  --request GET \
  --header "iPlanetDirectoryPro: AQIC5..." \
  --header "Content-Type: application/json" \
  http://openam.example.com:8080/openam/xacml/policies?filter=createdby=amadmin
```

2. You can also specify more than one search filter by logically ANDing the filters as follows:

```
$ curl \
  --request GET \
  --header "iPlanetDirectoryPro: AQIC5..." \
  --header "Content-Type: application/json" \
  http://openam.example.com:8080/openam/xacml/policies?filter=createdby=amadmin&
  filter=creationdate=135563832
```

4.4.2. Importing from XACML to OpenAM

OpenAM supports the import of XACML 3.0-based policy sets into OpenAM policies using the REST `/xacml/policies` endpoint. To test an import, OpenAM provides a dry-run feature that runs an import without saving the changes to the database. The dry-run feature provides a summary of the import so that you can troubleshoot any potential mismatches prior to the actual import.

You can import a XACML policy using an HTTP POST request for the root realm or a specific realm at the following endpoints:

```
http://openam.example.com:8080/openam/xacml/policies
http://openam.example.com:8080/openam/xacml/{realm}/policies

where {realm} is the name of a specific realm
```

To Import a XACML 3.0 Policy

1. You can do a dry run using the `dryrun=true` query to test the import. The dry-run option outputs in JSON format and displays the status of each import policy, where "U" indicates "Updated"; "A" for "Added". The dry-run does not actually update to the database. When you are ready for an actual import, you need to re-run the command without the `dryrun=true` query.

```
$ curl \
  --request POST \
  --header "iPlanetDirectoryPro: AQIC5..." \
  --header "Content-Type: application/xml" \
  --data @xacml-policy.xml
  http://openam.example.com:8080/openam/xacml/policies?dryrun=true
[
 {
   "status":"U",
   "name":"testHelpDeskreferralpolicy"
 },
 {
   "status":"U",
   "name":"test_Referral"
 },
 {
   "status":"U",
   "name":"testexternalreferralpolicy"
 },
 {
   "status":"U",
   "name":"testregexternalreferralpolicy"
 }
]
```

2. Use the `/xacml/policies` endpoint to import a XACML policy:

```
$ curl \
  --request POST \
  --header "iPlanetDirectoryPro: AQIC5..." \
  --header "Content-Type: application/xml" \
  --data @xacml-policy.xml
  http://openam.example.com:8080/openam/xacml/policies
```

Tip

You can import a XACML policy into a realm as follows:

```
$ curl \
  --request POST \
  --header "iPlanetDirectoryPro: AQIC5..." \
  --header "Content-Type: application/xml" \
  --data @xacml-policy.xml
  http://openam.example.com:8080/openam/xacml/{realm}/policies
```

4.5. Defining Applications

Application definitions set constraints for the policies that can be defined for a particular application. The default built-in application is the `iPlanetAMWebAgentService`, which OpenAM policy agents use to allow policy management through the console.

Application resources are represented in JSON and take the following form. Application resources are built from standard JSON objects and values (strings, numbers, objects, arrays, `true`, `false`, and `null`).

Do not use special characters within policy, application or referral names (for example, "my +referral") using the Policy Editor or REST endpoints as OpenAM returns a 400 Bad Request error. The special characters are: double quotes ("), plus sign (+), comma (,), less than (<), equals (=), greater than (>), backslash (\), and null (\u0000).

```
{
  "name": "application name string",
  "resources": [
     "resource name pattern",
     ...
  ],
  "actions": {
     "action name string": true,
     "other action name string": false,
     ...
  },
  "conditions": [
     "condition type",
     ...
  ],
  "realm": "the realm in which the application is defined",
  "applicationType": "application type name string",
  "description": "string describing application",
  "resourceComparator": "resource comparator class name",
  "subjects": [
     "subject type",
     ...
  ],
  "entitlementCombiner": "decision combiner",
  "saveIndex": "save index class name",
  "searchIndex": "search index class name",
  "attributeNames": [
     "attribute implementation class name",
     ...
```

```
  ]
}
```

The values for the fields shown in the description are explained below:

`"name"`

String matching the name in the URL used when creating the application by HTTP PUT or in the body when creating the application by HTTP POST.

`"resources"`

Strings specifying resource name patterns as in the following example:

```
{
    "resources": [
        "http://www.example.com:8080/*",
        "http://www.example.com:8080/*?*"
    ]
}
```

`"actions"`

Set of string action names, each set to a boolean indicating whether the action is allowed in the context of this application as in the following example:

```
{
    "actions": {
        "UPDATE": true,
        "PATCH": true,
        "QUERY": true,
        "CREATE": true,
        "DELETE": true,
        "READ": true,
        "ACTION": true
    }
}
```

`"conditions"`

Condition types allowed in the context of this application.

The following condition types are available:

`"AND"`
`"OR"`
`"NOT"`
`"AMIdentityMembership"`

```
"AuthLevel"
"AuthScheme"
"AuthenticateToRealm"
"AuthenticateToService"
"IPv4"
"IPv6"
"LDAPFilter"
"LEAuthLevel"
"OAuth2Scope"
"ResourceEnvIP"
"Session"
"SessionProperty"
"SimpleTime"
```

For more on condition types, see "Managing Policies" and "Viewing Environment Condition Types".

"realm"

Name of the realm where this application is defined. You must specify the realm in the application resource JSON even though it can be derived from the URL that is used when creating the application using an HTTP PUT call.

"applicationType"

Name of the application type used as a template for this application.

"description"

String describing the application.

"resourceComparator"

Class name of the resource comparator implementation used in the context of this application.

The following implementations are available:

```
"com.sun.identity.entitlement.ExactMatchResourceName"
"com.sun.identity.entitlement.PrefixResourceName"
"com.sun.identity.entitlement.RegExResourceName"
"com.sun.identity.entitlement.URLResourceName"
```

"subjects"

Subject types allowed in the context of this application.

The following subject types are available:

```
"AND"
```

"OR"
"NOT"
"AuthenticatedUsers"
"Identity"
"JwtClaim"
"NONE"

For more on subject types, see "Managing Policies" and "Viewing Subject Condition Types".

"entitlementCombiner"

Name of the decision combiner, such as "DenyOverride".

For more on decision combiners, see "Viewing Decision Combiners".

"saveIndex"

Class name of the implementation for creating indexes for resource names, such as "com.sun.identity.entitlement.util.ResourceNameIndexGenerator" for URL resource names.

"searchIndex"

Class name of the implementation for searching indexes for resource names, such as "com.sun.identity.entitlement.util.ResourceNameSplitter" for URL resource names.

"attributeNames"

A list of attribute names such as cn. The list is used to aid policy indexing and lookup.

The examples above do not show the fields added by OpenAM to indicate when the application was created and last updated, and by whom. Those field are "createdBy" and "lastModifiedBy", which take strings holding universal identifier DNs as their values, and "creationDate" and "lastModifiedDate", which an integer number of seconds since the Unix epoch.

4.5.1. Creating Applications

To create an application definition, either perform an HTTP PUT indicating the full path to the resource and the name in the resource matching the name in the path, or perform an HTTP POST with the name to use specified in the resource.

The HTTP PUT form includes the application definition as the JSON resource data, with the header Content-Type: application/json and uses the If-None-Match: * header.

The iPlanetDirectoryPro header sets the SSO token for a user who has access to perform the operation.

Do not use special characters within policy, application or referral names (for example, "my +referral") using the Policy Editor or REST endpoints as OpenAM returns a 400 Bad Request error.

The special characters are: double quotes ("), plus sign (+), comma (,), less than (<), equals (=), greater than (>), backslash (\), and null (\u0000).

```
$ curl \
 --request PUT \
 --header "If-None-Match: *" \
 --header "iPlanetDirectoryPro: AQIC5..." \
 --header "Content-Type: application/json" \
 --data '{
    "name": "myApplication",
    "resources": [
        "http://www.example.com:8080/*",
        "http://www.example.com:8080/*?*"
    ],
    "actions": {
        "UPDATE": true,
        "PATCH": true,
        "QUERY": true,
        "CREATE": true,
        "DELETE": true,
        "READ": true,
        "ACTION": true
    },
    "conditions": [
        "AND",
        "OR",
        "NOT",
        "AMIdentityMembership",
        "AuthLevel",
        "AuthScheme",
        "AuthenticateToRealm",
        "AuthenticateToService",
        "IPv4",
        "IPv6",
        "LDAPFilter",
        "LEAuthLevel",
        "OAuth2Scope",
        "ResourceEnvIP",
        "Session",
        "SessionProperty",
        "SimpleTime"
    ],
    "realm": "/",
    "applicationType": "iPlanetAMWebAgentService",
    "description": "An example application",
    "resourceComparator": "com.sun.identity.entitlement.URLResourceName",
    "subjects": [
        "AND",
        "OR",
        "NOT",
        "AuthenticatedUsers",
        "Identity",
        "JwtClaim"
    ],
    "entitlementCombiner": "DenyOverride",
    "saveIndex": null,
    "searchIndex": null,
    "attributeNames": []
```

```
}
' \
 https://openam.example.com:8443/openam/json/applications/myApplication
{
  "name" : "myApplication",
  "resources" : [
    "http://www.example.com:8080/*",
    "http://www.example.com:8080/*?*"
  ],
  "actions" : {
    "UPDATE" : true,
    "PATCH" : true,
    "QUERY" : true,
    "CREATE" : true,
    "DELETE" : true,
    "READ" : true,
    "ACTION" : true
  },
  "conditions": [
    "AND",
    "OR",
    "NOT",
    "AMIdentityMembership",
    "AuthLevel",
    "AuthScheme",
    "AuthenticateToRealm",
    "AuthenticateToService",
    "IPv4",
    "IPv6",
    "LDAPFilter",
    "LEAuthLevel",
    "OAuth2Scope",
    "ResourceEnvIP",
    "Session",
    "SessionProperty",
    "SimpleTime"
  ],
  "realm" : "/",
  "creationDate" : 1398761708295,
  "lastModifiedDate" : 1398761708295,
  "createdBy" : "id=amadmin,ou=user,dc=openam,dc=forgerock,dc=org",
  "lastModifiedBy" : "id=amadmin,ou=user,dc=openam,dc=forgerock,dc=org",
  "applicationType" : "iPlanetAMWebAgentService",
  "description" : "An example application",
  "resourceComparator" : "com.sun.identity.entitlement.URLResourceName",
  "subjects" : [
    "AND",
    "OR",
    "NOT",
    "AuthenticatedUsers",
    "Identity",
    "JwtClaim"
  ],
  "entitlementCombiner" : "DenyOverride",
  "saveIndex" : null,
  "searchIndex" : null,
  "attributeNames" : [ ]
}
```

You can use the query string parameters `_prettyPrint=true` to make the output easier to read, and `_fields=field-name[,field-name...]` to limit the fields returned in the output.

The HTTP POST form includes the application definition as the JSON resource data, with the header `Content-Type: application/json` and uses the `_action=create` operation.

The `iPlanetDirectoryPro` header sets the SSO token for a user who has access to perform the operation.

```
$ curl \
 --request POST \
 --header "iPlanetDirectoryPro: AQIC5..." \
 --header "Content-Type: application/json" \
 --data '{
    "name": "myApplication",
    "resources": [
        "http://www.example.com:8080/*",
        "http://www.example.com:8080/*?*"
    ],
    "actions": {
        "UPDATE": true,
        "PATCH": true,
        "QUERY": true,
        "CREATE": true,
        "DELETE": true,
        "READ": true,
        "ACTION": true
    },
    "conditions": [
        "AND",
        "OR",
        "NOT",
        "AMIdentityMembership",
        "AuthLevel",
        "AuthScheme",
        "AuthenticateToRealm",
        "AuthenticateToService",
        "IPv4",
        "IPv6",
        "LDAPFilter",
        "LEAuthLevel",
        "OAuth2Scope",
        "ResourceEnvIP",
        "Session",
        "SessionProperty",
        "SimpleTime"
    ],
    "realm": "/",
    "applicationType": "iPlanetAMWebAgentService",
    "description": "An example application",
    "resourceComparator": "com.sun.identity.entitlement.URLResourceName",
    "subjects": [
        "AND",
        "OR",
        "NOT",
        "AuthenticatedUsers",
```

```
        "Identity",
        "JwtClaim"
    ],
    "entitlementCombiner": "DenyOverride",
    "saveIndex": null,
    "searchIndex": null,
    "attributeNames": []
}' \
 https://openam.example.com:8443/openam/json/applications/?_action=create
{
  "name" : "myApplication",
  "resources" : [
    "http://www.example.com:8080/*",
    "http://www.example.com:8080/*?*"
  ],
  "actions" : {
    "UPDATE" : true,
    "PATCH" : true,
    "QUERY" : true,
    "CREATE" : true,
    "DELETE" : true,
    "READ" : true,
    "ACTION" : true
  },
  "conditions": [
    "AND",
    "OR",
    "NOT",
    "AMIdentityMembership",
    "AuthLevel",
    "AuthScheme",
    "AuthenticateToRealm",
    "AuthenticateToService",
    "IPv4",
    "IPv6",
    "LDAPFilter",
    "LEAuthLevel",
    "OAuth2Scope",
    "ResourceEnvIP",
    "Session",
    "SessionProperty",
    "SimpleTime"
  ],
  "realm" : "/",
  "creationDate" : 1398762452667,
  "lastModifiedDate" : 1398762452667,
  "createdBy" : "id=amadmin,ou=user,dc=openam,dc=forgerock,dc=org",
  "lastModifiedBy" : "id=amadmin,ou=user,dc=openam,dc=forgerock,dc=org",
  "applicationType" : "iPlanetAMWebAgentService",
  "description" : "An example application",
  "resourceComparator" : "com.sun.identity.entitlement.URLResourceName",
  "subjects" : [
    "AND",
    "OR",
    "NOT",
    "AuthenticatedUsers",
    "Identity",
    "JwtClaim"
  ],
```

```
  "entitlementCombiner" : "DenyOverride",
  "saveIndex" : null,
  "searchIndex" : null,
  "attributeNames" : [ ]
}
```

4.5.2. Reading Applications

To read an application definition, perform an HTTP GET specifying the resource name.

The `iPlanetDirectoryPro` header sets the SSO token for a user who has access to perform the operation.

```
$ curl \
 --header "iPlanetDirectoryPro: AQIC5..." \
 https://openam.example.com:8443/openam/json/applications/myApplication
{
  "name" : "myApplication",
  "resources" : [
    "http://www.example.com:8080/*",
    "http://www.example.com:8080/*?*"
  ],
  "actions" : {
    "POST" : true,
    "PATCH" : true,
    "GET" : true,
    "DELETE" : true,
    "OPTIONS" : true,
    "PUT" : true,
    "HEAD" : true
  },
  "conditions": [
    "AND",
    "OR",
    "NOT",
    "AMIdentityMembership",
    "AuthLevel",
    "AuthScheme",
    "AuthenticateToRealm",
    "AuthenticateToService",
    "IPv4",
    "IPv6",
    "LDAPFilter",
    "LEAuthLevel",
    "OAuth2Scope",
    "ResourceEnvIP",
    "Session",
    "SessionProperty",
    "SimpleTime"
  ],
  "realm" : "/",
  "creationDate" : 1398760362341,
  "lastModifiedDate" : 1398760362341,
  "createdBy" : "id=dsameuser,ou=user,dc=openam,dc=forgerock,dc=org",
  "lastModifiedBy" : "id=dsameuser,ou=user,dc=openam,dc=forgerock,dc=org",
  "applicationType" : "iPlanetAMWebAgentService",
```

```
  "description" : null,
  "resourceComparator" : "com.sun.identity.entitlement.URLResourceName",
  "subjects" : [
    "AND",
    "OR",
    "NOT",
    "AuthenticatedUsers",
    "Identity",
    "JwtClaim"
  ],
  "entitlementCombiner" : "DenyOverride",
  "saveIndex" : null,
  "searchIndex" : null,
  "attributeNames" : [ ]
}
```

You can use the query string parameters `_prettyPrint=true` to make the output easier to read, and `_fields=field-name[,field-name...]` to limit the fields returned in the output.

4.5.3. Updating Applications

To update an application definition, perform an HTTP PUT specifying the resource name with the application definition as the JSON resource data, and with the header `Content-Type: application/json`. This is essentially the same as creating an application definition, but without the `If-None-Match: *` header.

The `iPlanetDirectoryPro` header sets the SSO token for a user who has access to perform the operation.

```
$ curl \
 --request PUT \
 --header "iPlanetDirectoryPro: AQIC5..." \
 --header "Content-Type: application/json" \
 --data '{
    "name": "myApplication",
    "resources": [
        "http://www.example.com:8080/*",
        "http://www.example.com:8080/*?*"
    ],
    "actions": {
        "UPDATE": false,
        "PATCH": false,
        "QUERY": true,
        "CREATE": false,
        "DELETE": false,
        "READ": true,
        "ACTION": false
    },
    "conditions": [
        "SimpleTime"
    ],
    "realm": "/",
    "applicationType": "iPlanetAMWebAgentService",
    "description": "Updated application with fewer conditions and subjects",
```

```
    "resourceComparator": "com.sun.identity.entitlement.URLResourceName",
    "subjects": [
        "AuthenticatedUsers",
        "JwtClaim"
    ],
    "entitlementCombiner": "DenyOverride",
    "saveIndex": null,
    "searchIndex": null,
    "attributeNames": []
}' \
 https://openam.example.com:8443/openam/json/applications/myApplication
{
  "name" : "myApplication",
  "resources" : [
    "http://www.example.com:8080/*",
    "http://www.example.com:8080/*?*"
  ],
  "actions" : {
    "UPDATE" : false,
    "PATCH" : false,
    "QUERY" : true,
    "CREATE" : false,
    "DELETE" : false,
    "READ" : true,
    "ACTION" : false
  },
  "conditions" : [
    "SimpleTime"
  ],
  "realm" : "/",
  "creationDate" : 1398762194628,
  "lastModifiedDate" : 1398762194628,
  "createdBy" : "id=amadmin,ou=user,dc=openam,dc=forgerock,dc=org",
  "lastModifiedBy" : "id=amadmin,ou=user,dc=openam,dc=forgerock,dc=org",
  "applicationType" : "iPlanetAMWebAgentService",
  "description" : "Updated application with fewer conditions and subjects",
  "resourceComparator" : "com.sun.identity.entitlement.URLResourceName",
  "subjects" : [
    "AuthenticatedUsers",
    "JwtClaim"
  ],
  "entitlementCombiner" : "DenyOverride",
  "saveIndex" : null,
  "searchIndex" : null,
  "attributeNames" : [ ]
}
```

You can use the query string parameters `_prettyPrint=true` to make the output easier to read, and `_fields=field-name[,field-name...]` to limit the fields returned in the output.

4.5.4. Deleting Applications

To delete an application definition, perform an HTTP DELETE specifying the resource name.

The `iPlanetDirectoryPro` header sets the SSO token for a user who has access to perform the operation.

```
$ curl \
 --request DELETE \
 --header "iPlanetDirectoryPro: AQIC5..." \
 https://openam.example.com:8443/openam/json/applications/myApplication
{}
```

4.5.5. Listing Applications

To list application definitions, perform an HTTP GET on the endpoint, setting the `_queryFilter` query string parameter.

```
$ curl \
 --header "iPlanetDirectoryPro: AQIC5..." \
 https://openam.example.com:8443/openam/json/applications?_queryFilter=true

{
  "result" : [ {
    "name" : "myApplication",
    "resources" : [
      "http://www.example.com:8080/*",
      "http://www.example.com:8080/*?*"
    ],
    "actions" : {
      "POST" : true,
      "PATCH" : true,
      "GET" : true,
      "DELETE" : true,
      "OPTIONS" : true,
      "PUT" : true,
      "HEAD" : true
    },
    "conditions" : [
      "SimpleTime"
    ],
    "attributeNames" : [
    ],
    "realm":"/",
    "creationDate":1416428109115,
    "lastModifiedDate":1416428109115,
    "lastModifiedBy":"id=dsameuser,ou=user,dc=openam,dc=forgerock,dc=org",
    "subjects":[
      "JwtClaim",
      "AuthenticateUsers"
    ],
    "description":"Updated application with fewer conditions and subjects",
    "createdBy":id=dsameuser,ou=user,dc=openam,dc=forgerock,dc=org",
    "applicationType":"iPlanetAMWebServices",
    "entitlementCombiner":"DenyOverride",
    },
  ],
  "resultCount":1,
  "pagedResultsCookie":null,
  "remainingPagedResults":0
}
```

The `_queryFilter` parameter can take `true` to match every policy, `false` to match no policies, or a filter of the following form to match field values: *`field operator value`* where *field* represents the field name, *operator* is the operator code, *value* is the value to match, and the entire filter is URL-encoded. Supported operators are as follows:

- `eq`: equals (for matching strings)
- `ge`: greater than or equal to (for matching integers)
- `gt`: greater than (for matching integers)
- `le`: less than or equal to (for matching integers)
- `lt`: less than (for matching integers)

The *`field`* value can take the following values:

- `"name"` (string)
- `"description"` (string)
- `"createdBy"` (string)
- `"creationDate"` (date)
- `"lastModifiedBy"` (string)
- `"lastModified"` (date)
- `"lastModified"`

The String fields only support the `eq` operator, which is implemented using regular expression pattern matching.

The Date files support the `eq`, `ge`, `gt`, `le`, and `lt` operators. The implementation of `eq` for the Date fields do not use regular expression pattern matching.

Filters can be composed of multiple expressions by a using boolean operator `AND`, and by using parentheses, `(expression)`, to group expressions. You must URL encode the filter expression in `_queryFilter=filter`.

Since regular expressions are implemented for some operators, you can specify a filter that can include or exclude specified elements. For example, the following query searches for application names containing 'iPlanet'. The `queryFilter` string is URL-encoded.

```
$ curl --header "iPlanetDirectoryPro" AQIC5..." \
https://openam.example.com:8443/openam/json/applications?_queryFilter=name%20eq%20%22iPlanet.*%22
```

Note

While some operators use regular expressions as its underlying mechanism, CREST endpoints do not use regular expressions.

You can use the query string parameters `_prettyPrint=true` to make the output easier to read, `_fields=field-name[,field-name...]` to limit the fields returned in the output.

You can use `_pageSize=integer` to limit the number of results returned.

4.6. Viewing Application Types

Application types act as templates for creating applications.

The default application type that represents web resources is `iPlanetAMWebAgentService`, which defines resources as URL patterns and actions as HTTP methods. OpenAM policy agents use a default application type based on this type, which is called `iPlanetAMWebAgentService`. This is the application type for policies that you edit through OpenAM console.

OpenAM supports other application types as well, such as CREST types, that you can manage over the policy REST endpoints.

Applications types are server-wide, and do not differ by realm. Hence the URI for the application types API does not contain a realm component, but is `/json/applicationtypes`.

Application type resources are represented in JSON and take the following form. Application type resources are built from standard JSON objects and values (strings, numbers, objects, arrays, `true`, `false`, and `null`).

```
{
  "name": "application type name string",
  "actions": {
     "action name string": true,
     "other action name string": false,
     ...
  },
  "resourceComparator": "resource comparator class name",
  "saveIndex": "save index class name",
  "searchIndex": "search index class name",
  "applicationClassName": "com.sun.identity.entitlement.Application"
}
```

The values for the fields shown in the description are explained below:

`"name"`

String matching the name in the URL used when creating the application type by HTTP PUT or in the body when creating the application type by HTTP POST.

`"actions"`

Set of string action names, each set to a boolean indicating whether the action is allowed as in the following example:

```
{
    "actions": {
        "UPDATE": true,
        "PATCH": true,
        "QUERY": true,
        "CREATE": true,
        "DELETE": true,
        "READ": true,
        "ACTION": true
    }
}
```

`"resourceComparator"`

Class name of the resource comparator implementation used in the context of this application.

The following implementations are available:

`"com.sun.identity.entitlement.ExactMatchResourceName"`
`"com.sun.identity.entitlement.PrefixResourceName"`
`"com.sun.identity.entitlement.RegExResourceName"`
`"com.sun.identity.entitlement.URLResourceName"`

`"saveIndex"`

Class name of the implementation for creating indexes for resource names, such as `"com.sun.identity.entitlement.util.ResourceNameIndexGenerator"` for URL resource names.

`"searchIndex"`

Class name of the implementation for searching indexes for resource names, such as `"com.sun.identity.entitlement.util.ResourceNameSplitter"` for URL resource names.

`"applicationClassName"`

Class name of the application implementation, such as `"com.sun.identity.entitlement.Application"`.

4.6.1. Reading Application Types

To read an application type, perform an HTTP GET specifying the resource name.

The `iPlanetDirectoryPro` header sets the SSO token for a user who has access to perform the operation.

```
$ curl \
 --header "iPlanetDirectoryPro: AQIC5..." \
 https://openam.example.com:8443/openam/json/applicationtypes/crestPolicyService
{
  "name" : "crestPolicyService",
  "actions" : {
    "UPDATE" : true,
    "PATCH" : true,
    "QUERY" : true,
    "CREATE" : true,
    "DELETE" : true,
    "READ" : true,
    "ACTION" : true
  },
  "resourceComparator" : "com.sun.identity.entitlement.URLResourceName",
  "saveIndex" : "org.forgerock.openam.entitlement.indextree.TreeSaveIndex",
  "searchIndex" : "org.forgerock.openam.entitlement.indextree.TreeSearchIndex",
  "applicationClassName" : "com.sun.identity.entitlement.Application"
}
```

You can use the query string parameters `_prettyPrint=true` to make the output easier to read, and `_fields=field-name[,field-name...]` to limit the fields returned in the output.

4.6.2. Listing Application Types

To list application types, perform an HTTP GET on the endpoint, setting the `_queryFilter` query string parameter as in the following example:

```
$ curl \
 --header "iPlanetDirectoryPro: AQIC5..." \
 https://openam.example.com:8443/openam/json/applicationtypes?_queryFilter=true
{
  "result" : [ ... application types ... ],
  "resultCount" : 8,
  "pagedResultsCookie" : null,
  "remainingPagedResults" : -1
}
```

4.7. Viewing Environment Condition Types

Environment condition types describe the JSON representation of environment conditions that you can use in policy definitions.

Environment condition types are server-wide, and do not differ by realm. Hence the URI for the condition types API does not contain a realm component, but is `/json/conditiontypes`.

4.7.1. Reading Environment Condition Types

To read a environment condition type, perform an HTTP GET specifying the resource name.

The `iPlanetDirectoryPro` header sets the SSO token for a user who has access to perform the operation.

You can use the query string parameter `_prettyPrint=true` to make the output easier to read.

```
$ curl \
 --header "iPlanetDirectoryPro: AQIC5..." \
 https://openam.example.com:8443/openam/json/conditiontypes/IPv4
{
  "title" : "IPv4",
  "logical" : false,
  "config" : {
    "type" : "object",
    "properties" : {
      "startIp" : {
        "type" : "string"
      },
      "endIp" : {
        "type" : "string"
      },
      "dnsName" : {
        "type" : "array",
        "items" : {
          "type" : "string"
        }
      }
    }
  }
}
```

Notice that the environment condition type has a title, a "logical" field that indicates whether the type is a logical operator or takes a predicate, and a configuration specification. The configuration specification in this case indicates that an IPv4 environment condition has two properties, "startIp" and "endIp", that each take a single string value, and a third property, "dnsName," that takes an array of string values. In other words, a concrete IP environment condition specification without a DNS name constraint could be represented in a policy definition as in the following example:

```
{
   "type": "IPv4",
   "startIp": "127.0.0.1",
   "endIp": "127.0.0.255"
}
```

The configuration is what differs the most across environment condition types. The NOT condition, for example, takes a single condition object as the body of its configuration.

```
$ curl \
 --header "iPlanetDirectoryPro: AQIC5..." \
 https://openam.example.com:8443/openam/json/conditiontypes
    /NOT
{
  "title" : "NOT",
  "logical" : true,
  "config" : {
    "type" : "object",
    "properties" : {
      "condition" : {
        "type" : "object",
        "properties" : {
        }
      }
    }
  }
}
```

The concrete NOT condition therefore takes the following form.

```
{
    "type": "NOT",
    "condition": {
        ...
    }
}
```

The OR condition takes an array of conditions.

```
$ curl \
 --header "iPlanetDirectoryPro: AQIC5..." \
 https://openam.example.com:8443/openam/json/conditiontypes/OR
{
  "title" : "OR",
  "logical" : true,
  "config" : {
    "type" : "object",
    "properties" : {
      "conditions" : {
        "type" : "array",
        "items" : {
          "type" : "any"
        }
      }
    }
  }
}
```

A corresponding concrete OR condition thus takes the following form.

```
{
    "type": "OR",
    "conditions": [
        {
            ...
        },
        {
            ...
        },
        ...
    ]
}
```

4.7.2. Listing Environment Condition Types

To list all environment condition types, perform an HTTP GET on the endpoint, setting the query string parameter, `_queryFilter=true`, as in the following example:

```
$ curl \
 --header "iPlanetDirectoryPro: AQIC5..." \
 https://openam.example.com:8443/openam/json/conditiontypes?_queryFilter=true
{
  "result" : [ ... condition types ... ],
  "resultCount" : 18,
  "pagedResultsCookie" : null,
  "remainingPagedResults" : 0
}
```

You can use the query string parameter `_prettyPrint=true` to make the output easier to read.

4.8. Viewing Subject Condition Types

Subject types describe the JSON representation of subject conditions that you can use in policy definitions.

Subject condition types are server-wide, and do not differ by realm. Hence the URI for the subject types API does not contain a realm component, but is `/json/subjecttypes`.

4.8.1. Reading Subject Condition Types

To read a subject condition type, perform an HTTP GET specifying the resource name.

The `iPlanetDirectoryPro` header sets the SSO token for a user who has access to perform the operation.

You can use the query string parameter `_prettyPrint=true` to make the output easier to read.

```
$ curl \
 --header "iPlanetDirectoryPro: AQIC5..." \
 https://openam.example.com:8443/openam/json/subjecttypes/Identity
{
  "title" : "Identity",
  "logical" : false,
  "config" : {
    "type" : "object",
    "properties" : {
      "subjectValues" : {
        "type" : "array",
        "items" : {
          "type" : "string"
        }
      }
    }
  }
}
```

Notice that the subject type has a title, a "logical" field that indicates whether the type is a logical operator or takes a predicate, and a configuration specification. The configuration specification in this case indicates that an Identity subject condition has one property, "subjectValues", which takes an array of string values. In other words, a concrete Identity subject condition specification is represented in a policy definition as in the following example:

```
{
   "type": "Identity",
   "subjectValues": [
       "uid=scarter,ou=People,dc=example,dc=com"
   ]
}
```

The configuration is what differs the most across subject condition types. The AND condition, for example, takes an array of subject condition objects as the body of its configuration.

```
$ curl \
 --header "iPlanetDirectoryPro: AQIC5..." \
 https://openam.example.com:8443/openam/json/subjecttypes/AND
{
  "title" : "AND",
  "logical" : true,
  "config" : {
    "type" : "object",
    "properties" : {
      "subjects" : {
        "type" : "array",
        "items" : {
          "type" : "any"
        }
      }
    }
  }
}
```

The concrete AND subject condition therefore takes the following form.

```
{
    "type": "AND",
    "subject": [
      {
        ...
      },
      ...
    ]
}
```

4.8.2. Listing Subject Condition Types

To list all subject condition types, perform an HTTP GET on the endpoint, setting the query string parameter, `_queryFilter=true`, as in the following example:

```
$ curl \
 --header "iPlanetDirectoryPro: AQIC5..." \
 https://openam.example.com:8443/openam/json/subjecttypes/?_queryFilter=true
{
  "result" : [ ... subject types ... ],
  "resultCount" : 8,
  "pagedResultsCookie" : null,
  "remainingPagedResults" : 0
}
```

You can use the query string parameter `_prettyPrint=true` to make the output easier to read.

4.9. Viewing Subject Attributes

When you define a policy subject condition, the condition can depend on values of subject attributes stored in a user's profile. The list of possible subject attributes that you can use depends on the LDAP User Attributes configured for the Identity data store where OpenAM looks up the user's profile.

Subject attributes derive from the list of LDAP user attributes configured for the Identity data store. For more information, see Configuring Data Stores in the *Administration Guide*.

You can get a listing of all subject attribute names using the `/json/subjectattributes/?_queryFilter=true` command on the endpoint. There are no restrictions on the search and no pagination cookie is set. The subject attribute names are all returned as one in a "result" array.

```
$ curl \
  --request GET \
  --header "iPlanetDirectoryPro: AQIC5..." \
  --header "Content-Type: application/json" \
  https://openam.example.com:8443/openam/json/subjectattributes/?_queryFilter=true

{
 "result" : [ ... subject attribute types ... ],
 "resultCount": 87,
 "pagedResultsCookie": null,
 "remainingPagedResults": 0
}
```

4.10. Viewing Decision Combiners

Decision combiners describe how to resolve policy decisions when multiple policies apply.

Decision combiners are server-wide, and do not differ by realm. Hence the URI for the condition types API does not contain a realm component, but is `/json/decisioncombiners`.

To list all decision combiners, perform an HTTP GET on the endpoint, setting the query string parameter, `_queryFilter=true`, as in the following example:

```
$ curl \
 --header "iPlanetDirectoryPro: AQIC5..." \
 https://openam.example.com:8443/openam/json/decisioncombiners/?_queryFilter=true
{
  "result" : [ {
    "title" : "DenyOverride"
  } ],
  "resultCount" : 1,
  "pagedResultsCookie" : null,
  "remainingPagedResults" : 0
}
```

You can use the query string parameter `_prettyPrint=true` to make the output easier to read.

To view an individual decision combiner, perform an HTTP GET on its resource.

```
$ curl \
 --header "iPlanetDirectoryPro: AQIC5..." \
 https://openam.example.com:8443/openam/json/decisioncombiners/DenyOverride
{
  "title" : "DenyOverride"
}
```

4.11. Managing Referrals

Referrals are represented in JSON. They use standard JSON objects and values (strings, numbers, objects, arrays, `true`, `false`, and `null`) as follows:

```
{
  "name": "MyReferral",
  "description": "Look for policies in two additional subrealms to protect HR pages",
  "resources": {
    "iPlanetAMWebAgentService" : [ "http://example.com/hr/*"://example.com/hr/*"]
  },
  "realms" : ["/MySubrealm", "/MySubrealm/ChildSubrealm"]
}
```

The values for the fields shown in the example are explained below:

`"name"`

String matching the name in the URL used when creating the referral by HTTP PUT or in the body when creating the referral by HTTP POST.

`"description"`

String describing the referral.

`"resources"`

The application and resource pattern strings to which the referral applies.

`"realms"`

One or more realms to search for policies that might protect the resource.

4.11.1. Creating Referrals

To create a referral, either perform an HTTP PUT indicating the full path to the resource and the name in the resource matching the name in the path, or perform an HTTP POST with the name to use specified in the resource.

Do not use special characters within policy, application or referral names (for example, "my +referral") using the Policy Editor or REST endpoints as OpenAM returns a 400 Bad Request error. The special characters are: double quotes ("), plus sign (+), comma (,), less than (<), equals (=), greater than (>), backslash (\), and null (\u0000).

The HTTP PUT form includes the referral definition as the JSON resource data, with the header `Content-Type: application/json` and uses the `If-None-Match: *` header.

The `iPlanetDirectoryPro` header sets the SSO token for a user who has access to perform the operation.

The following example creates a referral named `MyReferral` in the top-level realm. The referral causes OpenAM policy evaluation to search for policies defined in the `/MySubrealm` and `/MySubrealm/ChildSubrealms` realms that have application type `iPlanetAMWebAgentService` and protect pages under `http://example.com/hr/`.

```
$ curl \
--request PUT \
--header "iPlanetDirectoryPro: AQIC5w..." \
--header "If-None-Match: *" \
--header "Content-Type: application/json" \
--data '{
  "name": "MyReferral",
  "description": "Look for policies in two additional subrealms to protect HR pages",
  "resources": {
    "iPlanetAMWebAgentService" : [ "http://example.com/hr/*"://example.com/hr/*"]
  },
  "realms" : ["/MySubrealm", "/MySubrealm/ChildSubrealm"]
}' \
https://openam.example.com:8443/openam/json/referrals/MyReferral

{
 "creationDate" : 1416956012949,
 "lastModifiedDate" : 1416956012949,
 "realms" : [ "/MySubrealm", "/MySubrealm/ChildSubrealm" ],
 "createdBy" : "id=amadmin,ou=user,dc=example,dc=com",
 "lastModifiedBy" : "id=amadmin,ou=user,dc=example,dc=com",
 "resources" : {
   "iPlanetAMWebAgentService" : [ "http://example.com/hr/*" ://example.com/hr/*" ]
 },
 "name" : "MyReferral"
}
```

You can use the query string parameters `_prettyPrint=true` to make the output easier to read, and `_fields=field-name [,field-name...]` to limit the fields returned in the output.

The HTTP POST form includes the referral definition as the JSON resource data, with the header `Content-Type: application/json` and uses the `_action=create` operation.

The `iPlanetDirectoryPro` header sets the SSO token for a user who has access to perform the operation.

```
$ curl \
--request POST \
--header "iPlanetDirectoryPro: AQIC5..." \
--header "Content-Type: application/json" \
--data '{
  "name": "MyReferralTwo",
  "description": "Look for policies in two additional subrealms to protect IT pages",
  "resources": {
    "iPlanetAMWebAgentService" : [ "http://example.com/it/*"://example.com/it/*"]
  },
  "realms" : ["/MySubrealm", "/MySubrealm/ChildSubrealm"]
}' \
https://openam.example.com:8443/openam/json/referrals?_action=create


{
  "creationDate" : 1416956884965,
  "lastModifiedDate" : 1416956884965,
  "realms" : [ "/MySubrealm", "/MySubrealm/ChildSubrealm" ],
  "createdBy" : "id=amadmin,ou=user,dc=example,dc=com",
  "lastModifiedBy" : "id=amadmin,ou=user,dc=example,dc=com",
  "resources" : {
    "iPlanetAMWebAgentService" : [ "http://example.com/it/*" ://example.com/it/*" ]
  },
  "name" : "MyReferralTwo"
}
```

4.11.2. Reading Referrals

To read a referral definition, perform an HTTP GET specifying the resource name.

The `iPlanetDirectoryPro` header sets the SSO token for a user who has access to perform the operation.

```
$ curl \
--header "iPlanetDirectoryPro: AQIC5..." \
https://openam.example.com:8443/openam/json/referrals/MyReferral


{
  "creationDate" : 1416956012949,
  "lastModifiedDate" : 1416956012949,
  "realms" : [ "/MySubrealm", "/MySubrealm/ChildSubrealm" ],
  "createdBy" : "id=amadmin,ou=user,dc=example,dc=com",
  "lastModifiedBy" : "id=amadmin,ou=user,dc=example,dc=com",
  "resources" : {
    "iPlanetAMWebAgentService" : [ "http://example.com/hr/*" ://example.com/hr/*" ]
  },
  "name" : "MyReferral"
}
```

You can use the query string parameters `_prettyPrint=true` to make the output easier to read, and `_fields=field-name [,field-name...]` to limit the fields returned in the output.

4.11.3. Updating Referrals

To update a referral, perform an HTTP PUT specifying the resource name with the referral definition as the JSON resource data, and with the header `Content-Type: application/json`. This is essentially the same as creating a referral, but without the `If-None-Match: *` header.

The `iPlanetDirectoryPro` header sets the SSO token for a user who has access to perform the operation.

```
 $ curl \
 --request PUT \
 --header "iPlanetDirectoryPro: AQIC5w..." \
 --header "Content-Type: application/json" \
 --data '{
   "name": "MyReferral",
   "description": "Update referral to protect HR US pages only",
   "resources": {
     "iPlanetAMWebAgentService" : [ "http://example.com/hr/us/*"://example.com/hr/us/*"]
   },
   "realms" : ["/MySubrealm", "/MySubrealm/ChildSubrealm"]
 }' \
 https://openam.example.com:8443/openam/json/referrals/MyReferral


{
  "creationDate" : 1416959024216,
  "lastModifiedDate" : 1416959024216,
  "realms" : [ "/MySubrealm", "/MySubrealm/ChildSubrealm" ],
  "createdBy" : "id=amadmin,ou=user,dc=example,dc=com",
  "lastModifiedBy" : "id=amadmin,ou=user,dc=example,dc=com",
  "resources" : {
    "iPlanetAMWebAgentService" : [ "http://example.com/hr/us/*" ://example.com/hr/us/*" ]
  },
  "name" : "MyReferral"
}
```

Tip

You can rename a referral by sending a new value in the *name* attribute of the JSON body, as shown below.

```
$ curl \
--request PUT \
--header "iPlanetDirectoryPro: AQIC5w..." \
--header "Content-Type: application/json" \
--data '{
  "name": "MyReferralHRUS",
  "description": "Update referral to protect HR US pages only",
  "resources": {
    "iPlanetAMWebAgentService" : [ "http://example.com/hr/us/*"://example.com/hr/us/*"]
  },
  "realms" : ["/MySubrealm", "/MySubrealm/ChildSubrealm"]
}' \
https://openam.example.com:8443/openam/json/referrals/MyReferral


{
  "creationDate" : 1416959240491,
  "lastModifiedDate" : 1416959240491,
  "realms" : [ "/MySubrealm", "/MySubrealm/ChildSubrealm" ],
  "createdBy" : "id=amadmin,ou=user,dc=example,dc=com",
  "lastModifiedBy" : "id=amadmin,ou=user,dc=example,dc=com",
  "resources" : {
    "iPlanetAMWebAgentService" : [ "http://example.com/hr/us/*" ://example.com/hr/us/*" ]
  },
  "name" : "MyReferralHRUS"
}
```

You can use the query string parameters `_prettyPrint=true` to make the output easier to read, and `_fields=field-name [,field-name...]` to limit the fields returned in the output.

4.11.4. Deleting Referrals

To delete a referral, perform an HTTP DELETE specifying the resource name.

The `iPlanetDirectoryPro` header sets the SSO token for a user who has access to perform the operation.

```
$ curl \
--header "iPlanetDirectoryPro: AQIC5w..." \
--request DELETE \
https://openam.example.com:8443/openam/json/referrals/MyReferralHRUS
{}
```

4.11.5. Listing Referrals

To list referrals, perform an HTTP GET on the endpoint, setting the `_queryFilter` query string parameter.

```
 $ curl \
 --header "iPlanetDirectoryPro: AQIC5w..." \
 https://openam.example.com:8443/openam/json/referrals?_queryFilter=true


{
  "result" : [ {
    "creationDate" : 1416956689230,
    "lastModifiedDate" : 1416956689230,
    "realms" : [ "/MySubrealm", "/MySubrealm/ChildSubrealm" ],
    "createdBy" : "id=amadmin,ou=user,dc=example,dc=com",
    "lastModifiedBy" : "id=amadmin,ou=user,dc=example,dc=com",
    "resources" : {
      "iPlanetAMWebAgentService" : [ "http://example.com/it/*" ://example.com/it/*" ]
    },
    "name" : "MyReferralOne"
    }, {
    "creationDate" : 1416956884965,
    "lastModifiedDate" : 1416956884965,
    "realms" : [ "/MySubrealm", "/MySubrealm/ChildSubrealm" ],
    "createdBy" : "id=amadmin,ou=user,dc=example,dc=com",
    "lastModifiedBy" : "id=amadmin,ou=user,dc=example,dc=com",
    "resources" : {
      "iPlanetAMWebAgentService" : [ "http://example.com/finance/*" ://example.com/finance/*" ]
    },
    "name" : "MyReferralTwo"
  } ],
  "resultCount" : 2,
  "pagedResultsCookie" : null,
  "remainingPagedResults" : 2
}
```

The `_queryFilter` parameter can take `true` to match every referral, `false` to match no referrals, or a filter of the following form to match field values: *`field operator value`* where *field* represents the field name, *operator* is the operator code, *value* is the value to match, and the entire filter is URL-encoded. Supported operators are as follows:

- `eq`: equals
- `ge`: greater than or equal to
- `gt`: greater than
- `le`: less than or equal to
- `lt`: less than

The *`field`* value can take the following values:

- `"name"` (string equality only)
- `"description"` (string equality only)

- `"applicationName"` (string equality only)
- `"createdBy"` (string equality only)
- `"lastModifiedBy"` (string equality only)
- `"creationDate"` (all comparisons are supported; the date is either an ISO-8601 string, or a integer number of seconds from the UNIX epoch)
- `"lastModified"` (all comparisons are supported; the date is either an ISO-8601 string, or a integer number of seconds from the UNIX epoch)

Filters can be composed of multiple expressions by a using boolean operator `AND`, and by using parentheses, `(expression)`, to group expressions. You must URL encode the filter expression in `_queryFilter=filter`.

You can use the query string parameters `_prettyPrint=true` to make the output easier to read, `_fields=field-name[,field-name...]` to limit the fields returned in the output.

You can use `_pageSize=integer` to limit the number of results returned.

You can use `_sortKeys=[+-]field[,field...]` to sort the results returned, where *field* represents a field in the JSON referral objects returned. Optionally use the `+` prefix to sort in ascending order (the default), or `-` to sort in descending order. The following example sorts the referrals by their names in descending order.

```
$ curl \
--header "iPlanetDirectoryPro: AQIC5w..." \
https://openam.example.com:8443/openam/json/referrals?_queryFilter=true\&_sortKeys=-name

{
  "result" : [ {
    "name" : "MyReferralTwo"
   }, {
     "name" : "MyReferralOne"
   } ],
   "resultCount" : 2,
   "pagedResultsCookie" : null,
   "remainingPagedResults" : 2
 }
```

4.12. Authorization (Legacy API)

You can call on OpenAM to decide whether to authorize access to a protected resource based on a valid token. Of course, you must percent encode the resource URI.

Interface Stability: Deprecated in the *Administration Guide*

```
$ curl "https://openam.example.com:8443/openam/identity/authorize?\
uri=http%3A%2F%2Fwww.example.com%3A8080%2Fexamples%2Findex.html\
&subjectid=AQIC5wM2LY4SfcxuxIP0VnP2lVjs7ypEM6VDx6srk56CN1Q.*AAJTSQACMDE.*"
boolean=true
```

To indicate access denied, OpenAM returns `boolean=false`.

4.12.1. Requesting Policy Decisions (Legacy API)

OpenAM provides additional REST APIs for requesting policy decisions.

The policy decision interfaces use the following path suffixes and query string parameters.

Path suffixes for policy decision requests include the following:

- `ws/1/entitlement/decision`. Request a decision pertaining to a single resource.
- `ws/1/entitlement/decisions`. Request decisions pertaining to multiple resources.
- `ws/1/entitlement/entitlement`. Request decisions for a specified resource URL.
- `ws/1/entitlement/entitlements`. Request decisions for a specified resource URL and all resources underneath.

Query string parameters for policy decision requests include the following.

- `subject=`*`encoded-token`*, where the token is encoded using the method implemented in `Encoder.java`.

  In the examples for this section, the token ID obtained during authentication for `amadmin` is abbreviated as `AQIC5...DU3*` and the encoded token ID for the subject is `MJ3QFTr4ZV2QrtlJvXlg0Q2dMRM=`.

- `action=get`, or `action=post`, which identifies the user agent action when requesting a decision.
- `application=iPlanetAMWebAgentService` or `application=crestPolicyService`
- `resource=`*`resource-url`*, or multiple `resources=`*`resource-url`* parameters for multiple decisions.
- `env=`*`requestDnsName%3Dfqdn`*, `env=`*`requestIP%3Ddotted-quads`*, `env=`*`requestTime%3Dseconds-since-epoch`*, and `env=`*`requestDnsName%3Dtime-zone`* where *time-zone* is from Java `TimeZone.getTimeZone().getID()`. The `env` parameters thus express conditions.

  In order to express a condition that specifies OAuth 2.0 scopes, set the value of the parameter to `scope=`*`scopes`*. To set scopes to `openid` and `profile`, use `env=scope%3Dopenid%20profile` for example.

Authentication for these interfaces uses cookies, so if your application is not running in a browser, first authenticate as described in REST APIs for Authentication & Logout.

To request a decision for a single resource, use an HTTP GET on `/ws/1/entitlement/decision` as in the following example:

```
$ curl \
 --request GET \
 --cookie "iPlanetDirectoryPro=AQIC5...DU3*" \
 "https://openam.example.com:8443/openam/ws/1/entitlement/decision\
?subject=MJ3QFTr4ZV2QrtlJvXlg0Q2dMRM=&action=GET\
&application=iPlanetAMWebAgentService\
&resource=http%3A%2F%2Fwww.example.com%2Findex.html"
allow
```

If access is denied, the result is `deny`.

To request decisions for multiple resources, use an HTTP GET on `/ws/1/entitlement/decisions` as in the following example:

```
$ curl \
 --request GET \
 --cookie "iPlanetDirectoryPro=AQIC5...DU3*" \
 "https://openam.example.com:8443/openam/ws/1/entitlement/decisions\
?subject=MJ3QFTr4ZV2QrtlJvXlg0Q2dMRM=&action=GET\
&application=iPlanetAMWebAgentService\
&resources=http%3A%2F%2Fwww.example.com%2Findex.html\
&resources=http%3A%2F%2Fwww.example.com%2Ffavicon.ico"
{
    "statusCode": 200,
    "body": {
        "results": [
            {
                "actionsValues": {
                    "POST": true,
                    "GET": true
                },
                "attributes": {},
                "advices": {},
                "resourceName": "http://www.example.com:80/index.html"
            },
            {
                "actionsValues": {
                    "POST": true,
                    "GET": true
                },
                "attributes": {},
                "advices": {},
                "resourceName": "http://www.example.com:80/favicon.ico"
            }
        ]
    },
    "statusMessage": "OK"
}
```

To request decisions for a given resource, use an HTTP GET on `/ws/1/entitlement/entitlement` as in the following example:

```
$ curl \
 --request GET \
 --cookie "iPlanetDirectoryPro=AQIC5...DU3*" \
 "https://openam.example.com:8443/openam/ws/1/entitlement/entitlement\
?subject=MJ3QFTr4ZV2QrtlJvXlg0Q2dMRM=\
&application=iPlanetAMWebAgentService\
&resource=http%3A%2F%2Fwww.example.com%2F*"
{
    "statusCode": 200,
    "body": {
        "actionsValues": {
            "POST": true,
            "GET": true
        },
        "attributes": {},
        "advices": {},
        "resourceName": "http://www.example.com:80/*"
    },
    "statusMessage": "OK"
}
```

To request decisions for all resources underneath a given resource, use an HTTP GET on `/ws/1/entitlement/entitlements` as in the following example:

```
$ curl \
 --request GET \
 --cookie "iPlanetDirectoryPro=AQIC5...DU3*" \
 "https://openam.example.com:8443/openam/ws/1/entitlement/entitlements\
?subject=MJ3QFTr4ZV2QrtlJvXlg0Q2dMRM=\
&application=iPlanetAMWebAgentService\
&resource=http%3A%2F%2Fwww.example.com%2F"
{
    "statusCode": 200,
    "body": {
        "results": [
            {
                "actionsValues": {},
                "resourceName": "http://www.example.com:80/"
            },
            {
                "actionsValues": {
                    "POST": true,
                    "GET": true
                },
                "advices": {},
                "resourceName": "http://www.example.com:80/*"
            },
            {
                "actionsValues": {
                    "POST": true,
                    "GET": true
                },
                "attributes": {},
                "advices": {},
                "resourceName": "http://www.example.com:80/*?*"
```

```
            }
        ]
    },
    "statusMessage": "OK"
}
```

4.13. Managing Policies (Legacy API)

OpenAM exposes a REST API through the `/ws/1/entitlement/privilege` endpoint under the deployment URI. The API lets you create, read, update, delete, and query policies.

Authentication for these interfaces uses cookies, so if your application is not running in a browser, first authenticate as described in REST APIs for Authentication & Logout.

4.14. Creating Policies (Legacy API)

You create a policy by using an HTTP POST of the JSON representation to the endpoint. You must URL encode the JSON before passing it to OpenAM.

```
$ cat entitlement.json
{
    "name": "Example HTTP",
    "eSubject": {
        "state": {
            "className": "com.sun.identity.policy.plugins.AuthenticatedUsers",
            "exclusive": false,
            "name": "All Authenticated Users",
            "values": []
        },
        "className": "com.sun.identity.entitlement.opensso.PolicySubject"
    },
    "entitlement": {
        "actionsValues": {
            "POST": true,
            "GET": true
        },
        "applicationName": "iPlanetAMWebAgentService",
        "name": "authorize",
        "resourceNames": [
            "http://www.example.com:80/*"
        ]
    }
}
$ curl \
 --request POST \
 --cookie "iPlanetDirectoryPro=AQIC5...DU3*" \
 --data-urlencode "privilege.json@entitlement.json" \
 https://openam.example.com:8443/openam/ws/1/entitlement/privilege
{"statusCode":201,"body":"Created","statusMessage":"Created"}

$ cat entitlement2.json
```

```
{
    "name": "Example HTTPS",
    "eSubject": {
        "state": {
            "className": "com.sun.identity.policy.plugins.AuthenticatedUsers",
            "exclusive": false,
            "name": "All Authenticated Users",
            "values": []
        },
        "className": "com.sun.identity.entitlement.opensso.PolicySubject"
    },
    "entitlement": {
        "actionsValues": {
            "POST": false,
            "GET": true
        },
        "applicationName": "iPlanetAMWebAgentService",
        "name": "authorize",
        "resourceNames": [
            "https://www.example.com:443/*?*"
        ]
    }
}

$ curl \
 --request POST \
 --cookie "iPlanetDirectoryPro=AQIC5...DU3*" \
 --data-urlencode "privilege.json@entitlement2.json" \
 https://openam.example.com:8443/openam/ws/1/entitlement/privilege
{"statusCode":201,"body":"Created","statusMessage":"Created"}
```

4.15. Reading Policies (Legacy API)

To read a policy, use an HTTP GET on the endpoint followed by the URL-encoded name of the policy.

Notice that the "state" is returned as a long string, and so is not shown here in full.

```
$ curl \
 --request GET \
 --cookie "iPlanetDirectoryPro=AQIC5...DU3*" \
 https://openam.example.com:8443/openam/ws/1/entitlement/privilege/Example%20HTTP
{
    "statusCode": 200,
    "body": {
        "results": {
            "name": "Example HTTP",
            "eSubject": {
                "state": "{\n  \"className\": \"com.sun.identity.policy...}",
                "className": "com.sun.identity.entitlement.opensso.PolicySubject"
            },
            "entitlement": {
                "actionsValues": {
                    "POST": true,
                    "GET": true
```

```
                },
                "applicationName": "iPlanetAMWebAgentService",
                "name": "authorize",
                "resourceNames": [
                    "http://www.example.com:80/*"
                ]
            }
        }
    },
    "statusMessage": "OK"
}

$ curl \
 --request GET \
 --cookie "iPlanetDirectoryPro=AQIC5...DU3*" \
 https://openam.example.com:8443/openam/ws/1/entitlement/privilege/Example%20HTTPS
{
    "statusCode": 200,
    "body": {
        "results": {
            "name": "Example HTTPS",
            "eSubject": {
                "state": "{\n  \"className\": \"com.sun.identity.policy...}",
                "className": "com.sun.identity.entitlement.opensso.PolicySubject"
            },
            "entitlement": {
                "actionsValues": {
                    "POST": false,
                    "GET": true
                },
                "applicationName": "iPlanetAMWebAgentService",
                "name": "authorize",
                "resourceNames": [
                    "https://www.example.com:443/*?*"
                ]
            }
        }
    },
    "statusMessage": "OK"
}
```

4.16. Updating Policies (Legacy API)

To update a policy, use an HTTP PUT on the endpoint followed by the URL-encoded name of the policy.

```
$ cat update.json
{
    "name": "Example HTTP",
    "eSubject": {
        "state": {
            "className": "com.sun.identity.policy.plugins.AuthenticatedUsers",
            "exclusive": false,
            "name": "All Authenticated Users",
```

```
            "values": []
        },
        "className": "com.sun.identity.entitlement.opensso.PolicySubject"
    },
    "entitlement": {
        "actionsValues": {
            "POST": false,
            "GET": true
        },
        "applicationName": "iPlanetAMWebAgentService",
        "name": "authorize",
        "resourceNames": [
            "http://www.example.com:80/*?*"
        ]
    }
}

$ curl \
 --request PUT \
 --cookie "iPlanetDirectoryPro=AQIC5...DU3*" \
 --data-urlencode "privilege.json@update.json" \
 https://openam.example.com:8443/openam/ws/1/entitlement/privilege/Example%20HTTP
{"statusCode":200,"body":"OK","statusMessage":"OK"}
```

4.17. Deleting Policies (Legacy API)

To delete a policy, use an HTTP DELETE on the endpoint followed by the URL-encoded name of the policy.

```
$ curl \
 --request DELETE \
 --cookie "iPlanetDirectoryPro=AQIC5...DU3*" \
 https://openam.example.com:8443/openam/ws/1/entitlement/privilege/Example%20HTTPS
{"statusCode":200,"body":"OK","statusMessage":"OK"}
```

4.18. Querying Policies (Legacy API)

To get the names of policies, use an HTTP GET on the endpoint.

```
$ curl \
 --request GET \
 --cookie "iPlanetDirectoryPro=AQIC5...DU3*" \
 https://openam.example.com:8443/openam/ws/1/entitlement/privilege
{
    "statusCode": 200,
    "body": {
        "results": [
            "Example HTTPS",
            "Example HTTP"
        ]
    },
    "statusMessage": "OK"
}
```

You can pass a filter query parameter to get only policies that match the filter. Make sure you URL encode the filter value.

```
$ curl \
 --request GET \
 --cookie "iPlanetDirectoryPro=AQIC5...DU3*" \
 "https://openam.example.com:8443/openam/ws/1/entitlement/privilege\
?subject=MJ3QFTr4ZV2QrtlJvXlg0Q2dMRM=&filter=name%3D*HTTP"
{
    "statusCode": 200,
    "body": {
        "results": [
            "Example HTTP"
        ]
    },
    "statusMessage": "OK"
}
```

Chapter 5

RESTful OAuth 2.0 and OpenID Connect 1.0 Services

This chapter shows how to use the OpenAM RESTful interfaces for OAuth 2.0 and OpenID Connect 1.0.

In this chapter, long URLs are wrapped to fit the printed page, as some of the output is formatted for easier reading.

5.1. OAuth 2.0 Authorization

OpenAM exposes the following REST endpoints for different OAuth 2.0 purposes.

- Endpoints for OAuth 2.0 clients and resource servers, mostly defined in RFC 6749, *The OAuth 2.0 Authorization Framework*, with an additional `tokeninfo` endpoint useful to resource servers.
- An endpoint for OAuth 2.0 token administration. This is specific to OpenAM.
- An endpoint for OAuth 2.0 client administration. This is specific to OpenAM.

When accessing the APIs, browser-based REST clients can rely on OpenAM to handle the session as usual. First authenticate with OpenAM. Then perform the operations in the browser session.

Clients not running in a browser can authenticate as described in *Authorization & Policy Management*, whereby OpenAM returns a `token.id` value. Clients pass the `token.id` value in a header named after the authentication cookie, by default `iplanetDirectoryPro`.

5.1.1. OAuth 2.0 Client & Resource Server Endpoints

As described in the *Administration Guide* chapter on *Managing OAuth 2.0 Authorization* in the *Administration Guide*, OpenAM exposes REST endpoints for making calls to OpenAM acting as an authorization server.

In addition to the standard authorization and token endpoints described in RFC 6749, OpenAM also exposes a token information endpoint for resource servers to get information about access tokens so they can determine how to respond to requests for protected resources.

OpenAM as authorization server exposes the following endpoints for clients and resource servers:

`/oauth2/authorize`

Authorization endpoint defined in RFC 6749, used to obtain an authorization grant from the resource owner

The `/oauth2/authorize` endpoint is protected by the policy created during OAuth 2.0 authorization server configuration, which grants all authenticated users access.

```
$ curl \
  https://openam.example.com:8443/openam/oauth2/authorize \
  ?client_id=myClient \
  &response_type=code \
  &scope=profile \
  &redirect_uri=https://www.example.com
```

If creating your own consent page, you can create a POST request to the endpoint with the following additional parameters:

`decision`

Whether the resource owner consents to the requested access, or denies consent.

Valid values are `allow` or `deny`.

`save_consent`

Updates the resource owner's profile to avoid having to prompt the resource owner to grant authorization when the client issues subsequent authorization requests.

To save consent, set the `save_consent` property to `on`.

You must provide the *Saved Consent Attribute Name* property with a profile attribute in which to store the resource owner's consent decision.

For more information on setting this property in the OAuth2 Provider service, see OAuth2 Provider Configuration in the *Reference*.

`csrf`

Duplicates the contents of the `iPlanetDirectoryPro` cookie, which contains the SSO token of the resource owner giving consent.

Duplicating the cookie value helps prevent against Cross-Site Request Forgery (CSRF) attacks.

Important

The `csrf` parameter is required by the `/oauth2/authorize` endpoint.

Example:

```
$ curl \
 --request POST \
 --header  "Content-Type: application/x-www-form-urlencoded" \
 --Cookie "iPlanetDirectoryPro=AQIC5w...*" \
 --data "redirect_uri=http://www.example.net" \
 --data "scope=profile" \
 --data "response_type=code" \
 --data "client_id=myClient" \
 --data "csrf=AQIC5w...*" \
 --data "decision=allow" \
 --data "save_consent=on" \
 "https://openam.example.com:8443/openam/oauth2/authorize?response_type=code&client_id=myClient"\
 "&realm=/&scope=profile&redirect_uri=http://www.example.net"
```

You must specify the realm if the OpenAM OAuth 2.0 provider is configured for a subrealm rather than the top-level realm. For example, if the OAuth 2.0 provider is configured for the `/customers` realm, then use `/oauth2/customers/authorize`.

The `/oauth2/authorize` endpoint can take additional parameters, such as:

- `module` and `service`. Use either as described in "Authenticating To OpenAM" in the *Administration Guide*, where `module` specifies the authentication module instance to use or `service` specifies the authentication chain to use when authenticating the resource owner.
- `response_mode=form_post`. Use this parameter to return a self-submitting form that contains the code instead of redirecting to the redirect URL with the code as a string parameter. For more information, see the OAuth 2.0 Form Post Response Mode spec.

`/oauth2/access_token`

Token endpoint defined in RFC 6749, used to obtain an access token from the authorization server

Example: `https://openam.example.com:8443/openam/oauth2/access_token`

`/oauth2/tokeninfo`

Endpoint not defined in RFC 6749, used to validate tokens, and to retrieve information such as scopes

Given an access token, a resource server can perform an HTTP GET on `/oauth2/tokeninfo?access_token=`*`token-id`* to retrieve a JSON object indicating `token_type`, `expires_in`, `scope`, and the `access_token` ID.

Example: `https://openam.example.com:8443/openam/oauth2/tokeninfo`

The `/oauth2/authorize`, and `/oauth2/access_token` endpoints function as described in RFC 6749.

The `/oauth2/authorize` endpoint is protected by the policy created during OAuth 2.0 authorization server configuration, which grants all authenticated users access.

The `/oauth2/tokeninfo` endpoint takes an HTTP GET on `/oauth2/tokeninfo?access_token=`*`token-id`*, and returns information about the token.

Resource servers — or any party having the token ID — can get token information through this endpoint without authenticating. This means any application or user can validate the token without having to be registered with OpenAM.

The following example shows OpenAM issuing an access token, and then returning token information:

```
$ curl \
 --request POST \
 --user "myClientID:password" \
 --data "grant_type=password&username=demo&password=changeit&scope=cn%20mail" \
 https://openam.example.com:8443/openam/oauth2/access_token
{
    "expires_in": 599,
    "token_type": "Bearer",
    "refresh_token": "f6dcf133-f00b-4943-a8d4-ee939fc1bf29",
    "access_token": "f9063e26-3a29-41ec-86de-1d0d68aa85e9"
}

$ curl https://openam.example.com:8443/openam/oauth2/tokeninfo
\
?access_token=f9063e26-3a29-41ec-86de-1d0d68aa85e9
{
    "mail": "demo@example.com",
    "scope": [
        "mail",
        "cn"
    ],
    "cn": "demo",
    "realm": "/",
    "token_type": "Bearer",
    "expires_in": 577,
    "access_token": "f9063e26-3a29-41ec-86de-1d0d68aa85e9"
}
```

The resource server making decisions about whether the token is valid can thus use the `/oauth2/tokeninfo` endpoint to retrieve expiration information about the token. Depending on the scopes implementation, the JSON response about the token can also contain scope information. As described in the *Administration Guide*, the default scopes implementation in OpenAM considers scopes to be names of attributes in the resource owner's user profile. Notice that the JSON response contains the values for those attributes from the user's profile, as in the preceding example, with scopes set to `mail` and `cn`.

Both the `/oauth2/authorize` and `/oauth2/access_token` endpoints can take additional parameters. In particular you must specify the realm using the `realm=realm-name` parameter if the OpenAM OAuth 2.0 provider is configured for a subrealm rather than / (Top-Level Realm). For example, if the OAuth 2.0 provider is configured for the `/customers` realm, then use `/oauth2/authorize?realm=/customers` and `/oauth2/access_token?realm=/customers`. The realm can be passed either as a URL query string parameter as shown here, or as parameters in the POST data.

The `/oauth2/authorize` endpoint can also take `module` and `service` parameters. Use either as described in *Authenticating To OpenAM* in the *Administration Guide*, where `module` specifies the authentication

module instance to use or `service` specifies the authentication chain to use when authenticating the resource owner.

5.1.2. OAuth 2.0 Token Administration Endpoint

The OpenAM-specific OAuth 2.0 token administration endpoint lets administrators read, list, and delete OAuth 2.0 tokens. OAuth 2.0 clients can also manage their own tokens.

OpenAM exposes the token administration endpoint at `/frrest/oauth2/token`, such as `https://openam.example.com:8443/openam/frrest/oauth2/token`.

Note

This endpoint location is likely to change in the future.

To get a token, perform an HTTP GET on `/frrest/oauth2/token/token-id`, as in the following example:

```
$ curl \
 --request POST \
 --user "myClientID:password" \
 --data "grant_type=password&username=demo&password=changeit&scope=cn" \
 https://openam.example.com:8443/openam/oauth2/access_token
{
    "scope": "cn",
    "expires_in": 60,
    "token_type": "Bearer",
    "access_token": "f5fb4833-ba3d-41c8-bba4-833b49c3fe2c"
}

$ curl \
 --header "iplanetDirectoryPro: AQIC5wM2LY4Sfcxs...EwNDU2NjE0*" \
 https://openam.example.com:8443/openam/frrest/oauth2/token/9c6a48fc...fba468b0f
{
    "expireTime": [
        "1418818601396"
    ],
    "tokenName": [
        "access_token"
    ],
    "scope": [
        "cn"
    ],
    "grant_type": [
        "password"
    ],
    "clientID": [
        "myClientID"
    ],
    "parent": [],
    "id": [
        "f5fb4833-ba3d-41c8-bba4-833b49c3fe2c"
    ],
    "tokenType": [
        "Bearer"
```

```
    ],
    "redirectURI": [],
    "nonce": [],
    "realm": [
        "/"
    ],
    "userName": [
        "demo"
    ]
}
```

To list tokens, perform an HTTP GET on `/frrest/oauth2/token/?_queryId=string`, where *string* is either `access_token` to request the list of access tokens for the current user or all users if requested by `amAdmin`, or any other string to request the list of access tokens and refresh tokens.

The following example shows a search for the demo user's access tokens.

```
$ curl \
 --header "iplanetDirectoryPro: AQIC5wM2LY4Sfcw..." \
 https://openam.example.com:8443/openam/frrest/oauth2/token/?_queryId=access_token
{
    "result": [
        {
            "expireTime": "Dec 17, 2014 1:02 PM",
            "tokenName": [
                "access_token"
            ],
            "scope": [
                "cn"
            ],
            "grant_type": [
                "password"
            ],
            "clientID": [
                "myClientID"
            ],
            "parent": [],
            "id": [
                "d2bbd4f9-955f-4683-bb83-bc0d7523b0f9"
            ],
            "tokenType": [
                "Bearer"
            ],
            "redirectURI": [],
            "nonce": [],
            "realm": [
                "/"
            ],
            "userName": [
                "demo"
            ],
            "display_name": "",
            "scopes": "cn"
        },
        {
            "expireTime": "Dec 17, 2014 1:02 PM",
```

```
            "tokenName": [
                "access_token"
            ],
            "scope": [
                "cn"
            ],
            "grant_type": [
                "password"
            ],
            "clientID": [
                "myClientID"
            ],
            "parent": [],
            "id": [
                "12f9907f-dbfb-4c4c-a231-2b24964fee8b"
            ],
            "tokenType": [
                "Bearer"
            ],
            "redirectURI": [],
            "nonce": [],
            "realm": [
                "/"
            ],
            "userName": [
                "demo"
            ],
            "display_name": "",
            "scopes": "cn"
        }
    ],
    "resultCount": 2,
    "pagedResultsCookie": null,
    "remainingPagedResults": -1
}
```


To delete a token, perform an HTTP DELETE on `/frrest/oauth2/token/token-id`, as in the following example:

```
$ curl \
 --request POST \
 --data "grant_type=client_credentials&username=demo&password=changeit\
&client_id=myClientID&client_secret=password&scope=cn%20mail" \
 https://openam.example.com:8443/openam/oauth2/access_token
{
    "expires_in": 599,
    "token_type": "Bearer",
    "access_token": "867aaab2-61d7-4b78-9b80-4f9098034540"
}

$ curl \
 --request DELETE \
 --header "iplanetDirectoryPro: AQIC5wM2LY4Sfcxs...EwNDU2NjE0*" \
 https://openam.example.com:8443/openam/frrest/oauth2/token/867aaab2..098034540
{
    "success": "true"
}
```

5.1.3. OAuth 2.0 Client Administration Endpoint

The OAuth 2.0 administration endpoint lets OpenAM administrators and agent administrators create (that is, register) and delete OAuth 2.0 clients.

OpenAM exposes this endpoint at `/frrest/oauth2/client`, such as `https://openam.example.com:8443/openam/frrest/oauth2/client`.

Note

This endpoint location is likely to change in the future.

To create an OAuth 2.0 client, perform an HTTP POST to `/frrest/oauth2/client/?_action=create` with a JSON object fully specifying the client, as in the following example.

```
$ curl \
 --request POST \
 --header "iplanetDirectoryPro: AQIC5wM...3MTYxOA..*" \
 --header "Content-Type: application/json" \
 --data \
 '{"client_id":["testClient"],
   "realm":["/"],
   "userpassword":["secret12"],
   "com.forgerock.openam.oauth2provider.clientType":["Confidential"],
   "com.forgerock.openam.oauth2provider.redirectionURIs":
     ["www.client.com","www.example.com"],
   "com.forgerock.openam.oauth2provider.scopes":["cn","sn"],
   "com.forgerock.openam.oauth2provider.defaultScopes":["cn"],
   "com.forgerock.openam.oauth2provider.name":["My Test Client"],
   "com.forgerock.openam.oauth2provider.description":["OAuth 2.0 Client"]
  }' \
 https://openam.example.com:8443/openam/frrest/oauth2/client/?_action=create
{"success":"true"}
```

When creating an OAuth 2.0 client, use the following fields in your JSON object.

`"client_id"`

(Required) This field takes an array containing the client identifier as defined in RFC 6749.

`"realm"`

(Required) This field takes an array containing the OpenAM realm in which to create the client as defined in RFC 6749.

`"userpassword"`

(Required) This field takes an array containing the client secret as defined in RFC 6749.

`"com.forgerock.openam.oauth2provider.clientType"`

(Required) This field takes an array containing the client type, either `"Confidential"` or `"Public"` as defined in RFC 6749.

`"com.forgerock.openam.oauth2provider.redirectionURIs"`

(Optional for confidential clients) This field takes an array of client redirection endpoints as defined in RFC 6749.

`"com.forgerock.openam.oauth2provider.scopes"`

(Optional) This field takes an array of scopes as defined in RFC 6749. The default scopes implementation takes scopes to be names of attributes in the resource owner profile.

Specify localized scopes in *`scope|locale|localized description`* format.

`"com.forgerock.openam.oauth2provider.defaultScopes"`

(Optional) This field takes an array of default scopes set automatically when tokens are issued.

`"com.forgerock.openam.oauth2provider.name"`

(Optional) This field takes an array containing the client name to display to the resource owner when the resource owner must authorize client access to protected resources.

Specify localized names in *`locale|localized name`* format.

`"com.forgerock.openam.oauth2provider.description"`

(Optional) This field takes an array containing the description to display to the resource owner when the resource owner must authorize client access to protected resources.

Specify localized descriptions in *`locale|localized description`* format.

To delete an OAuth 2.0 client, perform an HTTP DELETE on `/frrest/oauth2/client/client-id`, as in the following example:

```
$ curl \
 --request DELETE \
 --header "iplanetDirectoryPro: AQIC5wM...3MTYxOA..*" \
 https://openam.example.com:8443/openam/frrest/oauth2/client/myClient
{"success":"true"}
```

Tip

To delete an OAuth 2.0 client from a subrealm, add the name of the subrealm in a query parameter named `realm`, as in the following example:

```
$ curl \
 --request DELETE \
 --header "iplanetDirectoryPro: AQIC5wM...3MTYxOA..*" \
 https://openam.example.com:8443/openam/frrest/oauth2/client/myClient?realm=myRealm
{"success":"true"}
```

5.2. OpenID Connect 1.0

OpenID Connect 1.0 extends OAuth 2.0 so the client can verify claims about the identity of the end user, get profile information about the end user, and log the user out at the end of the OpenAM session.

OpenAM exposes the following REST endpoints for OpenID Connect 1.0 purposes.

- Endpoints for discovering information.

- An endpoint for registering client applications.
- Endpoints for client authorization.
- Endpoints for session management.

5.2.1. Discovering OpenID Connect 1.0 Configuration

OpenAM exposes endpoints for discovering information about the provider configuration, and about the provider for a given end user.

- `/.well-known/openid-configuration` allows clients to retrieve OpenID Provider configuration by HTTP GET as specified by OpenID Connect Discovery 1.0.
- `/.well-known/webfinger` allows clients to retrieve the provider URL for an end user by HTTP GET as specified by OpenID Connect Discovery 1.0.

For examples, see *Configuring OpenAM For OpenID Connect Discovery* in the *Administration Guide*.

5.2.2. Registering OpenID Connect 1.0 Clients

OpenAM allows both static and dynamic registration of OpenID Connect client applications. For dynamic registration according to the OpenID Connect Dynamic Client Registration 1.0 specification, the endpoint is `/oauth2/connect/register`. See *To Register a Client Dynamically* in the *Administration Guide* for details.

5.2.3. Performing OpenID Connect 1.0 Client Authorization

Registered clients can request authorization through OpenAM.

OpenID Connect 1.0 supports both a Basic Client Profile using the OAuth 2.0 authorization code grant, and an Implicit Client Profile using the OAuth 2.0 implicit grant. These client profiles rely on the OAuth 2.0 endpoints for authorization. Those endpoints are described in "OAuth 2.0 Client & Resource Server Endpoints".

In addition, authorized clients can access end user information through the OpenID Connect 1.0 specific endpoint `/oauth2/userinfo`.

For examples, see *Client Examples* in the *Administration Guide*.

5.2.4. Managing OpenID Connect 1.0 Sessions

Registered clients can use OpenID Connect Session Management 1.0 to handle end user logout actions.

- `/oauth2/connect/checkSession` allows clients to retrieve session status notifications.

- `/oauth2/connect/endSession` allows clients to terminate end user sessions.

For an example, see *Managing User Sessions* in the *Administration Guide*.

Chapter 6
RESTful User Self-Service

This chapter shows how to use the OpenAM RESTful interfaces for user self-service functionality: user self-registration and forgotten password reset.

In this chapter, long URLs are wrapped to fit the printed page, as some of the output is formatted for easier reading.

6.1. User Self-Registration

The OpenAM REST API for users provides an action for self-registration. The feature works by sending an email to the user in response to RESTful HTTP POST requesting registration with an email address. When the user clicks the link received by mail, an application intercepts the HTTP GET, transforms the query string values into an HTTP POST to confirm the operation. OpenAM responds to the application with a JSON object that the application can further use to request creation of the user account to complete the transaction.

To Set Up User Self-Registration

1. Configure the Email Service.

   You must configure the Email Service to send mail notifications to users who self-register. You can configure these globally in OpenAM console at Configuration > Global > Email Service.

   Alternatively, you can configure them for an individual realm under Access Control > *Realm Name* > Services.

2. Configure User Self Service.

   You must enable self-registration in the User Self Service service. You can configure these globally in OpenAM console at Configure > Global > User Self Service. On the User Self Service page, click the `Enabled` checkbox next to Self-Registration for Users, and then click Save.

   At this point users can self-register. The starting screen for self-registration is at `/XUI/#register/` under the base URL where OpenAM is installed. The default confirmation URI is `/XUI/confirm.html`.

3. Perform an HTTP POST on `/json/users?_action=register` with the new user's mail.

   To use a subject and message other than those configured in the Email Service, you can optionally set the mail subject and message content by including "subject" and "message"

strings in the JSON data. For example, the following POST results in a mail with subject `Confirm registration with OpenAM` and content `Follow this link to confirm your registration` in addition to the confirmation link.

Notice that authentication is not required.

```
$ curl \
 --request POST \
 --header "Content-Type: application/json" \
 --data \
 '{
   "email": "newuser@example.com",
   "subject": "Confirm registration with OpenAM",
   "message": "Follow this link to confirm your registration"
 }' \
 https://openam.example.com:8443/openam/json/users?_action=register
{}
```

On success, the response is an empty JSON object `{}` as shown in the example.

4. The user receives an email message that includes a URL similar to the following example, but all on one line. The user has self-registered in the root realm:

```
https://openam.example.com:8443/openam/XUI/confirm.html?
     confirmationId=f4x0Dh6iZCXtX8nhiSb3xahNxrg=
     &email=newuser@example.com
     &tokenId=yA26LZ6SxFEgNuF86/SIXfimGlg=
     &realm=/
```

5. Intercept the HTTP GET request to this URL when the user clicks the link.

   Your application must use the confirmation link to construct an HTTP POST to `/json/users?_action=confirm` from the query string parameters as shown in the following example:

```
$ curl \
 --request POST \
 --header "Content-Type: application/json" \
 --data \
 '{
   "email": "newuser@example.com",
   "tokenId": "yA26LZ6SxFEgNuF86/SIXfimGlg=",
   "confirmationId": "f4x0Dh6iZCXtX8nhiSb3xahNxrg="
 }' \
 https://openam.example.com:8443/openam/json/users?_action=confirm
{
    "email": "newuser@example.com",
    "tokenId": "yA26LZ6SxFEgNuF86/SIXfimGlg=",
    "confirmationId": "f4x0Dh6iZCXtX8nhiSb3xahNxrg="
}
```

   The response is a further confirmation that the account can be created.

6. Using the confirmation, your application must make an authenticated HTTP POST to `/json/users?_action=anonymousCreate` to create the user as shown in the following example:

```
$ curl \
 --request POST \
 --header "Content-Type: application/json" \
 --data \
 '{
   "email": "newuser@example.com",
   "tokenId": "yA26LZ6SxFEgNuF86/SIXfimGlg=",
   "confirmationId": "f4x0Dh6iZCXtX8nhiSb3xahNxrg=",
   "username": "newuser",
   "userpassword": "password"
 }' \
 https://openam.example.com:8443/openam/json/users?_action=anonymousCreate
{
    "username": "newuser",
    "realm": "/",
    "uid": [
        "newuser"
    ],
    "mail": [
        "newuser@example.com"
    ],
    "sn": [
        "newuser"
    ],
    "userPassword": [
        "{SSHA}dAiONYMxqFiNilXeLXUQoDpHlePYtiJcjYw8Dw=="
    ],
    "cn": [
        "newuser"
    ],
    "inetUserStatus": [
        "Active"
    ],
    "dn": [
        "uid=newuser,ou=people,dc=openam,dc=forgerock,dc=org"
    ],
    "objectClass": [
        "devicePrintProfilesContainer",
        "person",
        "sunIdentityServerLibertyPPService",
        "inetorgperson",
        "sunFederationManagerDataStore",
        "iPlanetPreferences",
        "iplanet-am-auth-configuration-service",
        "organizationalperson",
        "sunFMSAML2NameIdentifier",
        "inetuser",
        "forgerock-am-dashboard-service",
        "iplanet-am-managed-person",
        "iplanet-am-user-service",
        "sunAMAuthAccountLockout",
        "top"
    ],
    "universalid": [
        "id=newuser,ou=user,dc=openam,dc=forgerock,dc=org"
```

```
    ]
}
```

At this point, the user is registered, active, and can authenticate with OpenAM.

6.2. Resetting Forgotten Passwords

The OpenAM REST API provides an action for handling forgotten passwords as long as the user has a valid email address in their profile. This is an alternative to the password reset capability described in the *Administration Guide* chapter, *Configuring Password Reset* in the *Administration Guide*.

Tip

If the current password is known, use the *Change Password* feature to change a password.

The option is disabled by default. You can enable it in the OpenAM Console globally by using Configuration > Global > User Self Service.

Alternatively, you can enable it for an individual realm under Access Control > *Realm Name* > Services > Add > User Self Service.

An example follows, showing the steps in more detail.

1. Configure the Email Service.

   In particular, you must configure the Email Service to send mail allowing the user to reset the forgotten password.

   You can configure the service globally in the OpenAM Console via Configuration > Global > Email Service.

   Alternatively, you can configure it for an individual realm under Access Control > *Realm Name* > Services.

   At this point users with mail addresses can reset their forgotten passwords. The starting screen for forgotten password reset is at `/XUI/#forgotPassword/` under the base URL where OpenAM is installed. The default confirmation URI is `/XUI/confirm.html`.

   The steps that follow show how to use the REST API directly.

2. Perform an HTTP POST on `/json/users?_action=forgotPassword` with the user's ID.

   To use a subject and message other than those configured in the Email Service, you can optionally set the mail subject and message content by including "subject" and "message" strings in the JSON data. For example, the following POST results in a mail with subject `Reset your forgotten password with OpenAM` and content `Follow this link to reset your password` in addition to the confirmation link.

Notice that authentication is not required.

```
$ curl \
 --request POST \
 --header "Content-Type: application/json" \
 --data '{
   "username": "demo",
   "subject": "Reset your forgotten password with OpenAM",
   "message": "Follow this link to reset your password"
 }' \
 https://openam.example.com:8443/openam/json/users/?_action=forgotPassword
{}
```

Note that you can also use the `email` attribute to locate the user. If both `username` and `mail` attributes are used, then a request error is issued. If more than one account has been registered with the same email address, the password reset process does not start.

```
$ curl \
 --request POST \
 --header "Content-Type: application/json" \
 --data '{
   "email": "demo@example.com",
   "subject": "Reset your forgotten password with OpenAM",
   "message": "Follow this link to reset your password"
 }' \
 https://openam.example.com:8443/openam/json/users/?_action=forgotPassword
{}
```

On success, the response is an empty JSON object `{}` as shown in the example.

3. OpenAM looks up the email address in the user profile, and sends an email message that includes a URL as in the following example, but all on one line.

```
https://openam.example.com:8443/openam/json/XUI/confirm.html
     ?confirmationId=sdfsfeM+URcWVQ7vvCDnx4N5Vut7SBIY=
     &tokenId=vkm+5v58cTs1yQcQl5HCQGOsuQk=
     &username=demo&realm=/
```

4. Intercept the HTTP GET request to this URL when the user clicks the link.

   Your application must use the confirmation link to construct an HTTP POST to `/json/users?_action=confirm` from the query string parameters as shown in the following example:

```
$ curl \
--request POST \
--header "Content-Type: application/json" \
--data \
'{
"username":"demo",
"tokenId":"vkm+5v58cTs1yQcQl5HCQGOsuQk=",
"confirmationId":"sdfsfeM+URcWVQ7vvCDnx4N5Vut7SBIY="
}' \
https://openam.example.com:8443/openam/json/users?_action=confirm
{
 "username": "demo",
 "tokenId": "vkm+5v58cTs1yQcQl5HCQGOsuQk=",
 "confirmationId": "sdfsfeM+URcWVQ7vvCDnx4N5Vut7SBIY="
 }
```

The response is a further confirmation that the request is valid, has not expired, and the password can be reset.

5. Using the confirmation, your application must construct an HTTP POST to `/json/users?_action=forgotPasswordReset` to reset the password as shown in the following example.

   Your POST includes the new password as the value of the "userpassword" field in the JSON payload. You can also use the `email` attribute instead of `username`.

```
$ curl \
 --request POST \
 --header "Content-Type: application/json" \
 --data '{
 "username":"demo",
 "userpassword":"password",
 "tokenId":"vkm+5v58cTs1yQcQl5HCQGOsuQk=",
 "confirmationId":"sdfsfeM+URcWVQ7vvCDnx4N5Vut7SBIY="
 }' \
 https://openam.example.com:8443/openam/json/users?_action=forgotPasswordReset
{}
```

   On success or failure, the REST call returns an empty message, so that information is not leaked.

At this point the user can authenticate with the new password.

6.3. Displaying Dashboard Applications

OpenAm lets administrators configure online applications to display applications on user Dashboards. You can used exposed REST API to display information about the online applications.

`/dashboard/assigned`

This endpoint retrieves the list of applications assigned to the authenticated user.

```
$ curl \
 --header "iplanetDirectoryPro: AQIC5w...2NzEz*" \
 https://openam.example.com:8443/openam/json/dashboard/assigned
{
  "google": {
      "dashboardIcon": [
          "Google.gif"
      ],
      "dashboardName": [
          "Google"
      ],
      "dashboardLogin": [
          "http://www.google.com"
      ],
      "ICFIdentifier": [
          ""
      ],
      "dashboardDisplayName": [
          "Google"
      ],
      "dashboardClassName": [
          "SAML2ApplicationClass"
      ]
   }
}
```

/dashboard/available

This endpoint retrieves the list of applications available in the authenticated user's realm. The example is based on two of the default Dashboard applications: Google and Salesforce.

```
$ curl \
 --header "iplanetDirectoryPro: AQIC5w...2NzEz*" \
 https://openam.example.com:8443/openam/json/dashboard/available
{
  "google": {
      "dashboardIcon": [
          "Google.gif"
      ],
      "dashboardName": [
          "Google"
      ],
      "dashboardLogin": [
          "http://www.google.com"
      ],
      "ICFIdentifier": [
          ""
      ],
      "dashboardDisplayName": [
          "Google"
      ],
      "dashboardClassName": [
          "SAML2ApplicationClass"
      ]
  }
```

```
  "salesforce": {
      "dashboardIcon": [
          "salesforce.gif"
      ],
      "dashboardName": [
          "Salesforce"
      ],
      "dashboardLogin": [
          "http://salesforce.com"
      ],
      "ICFIdentifier": [
          ""
      ],
      "dashboardDisplayName": [
          "Salesforce"
      ],
      "dashboardClassName": [
          "SAML2ApplicationClass"
      ]
  }
}
```

/dashboard/defined

This endpoint retrieves the list of all applications available defined for the OpenAM Dashboard service. The example is based on the three default Dashboard applications: Google, Salesforce, and Zendesk.

```
$ curl \
 --header "iplanetDirectoryPro: AQIC5w...2NzEz*" \
 https://openam.example.com:8443/openam/json/dashboard/defined
{
    "google": {
        "dashboardIcon": [
            "Google.gif"
        ],
        "dashboardName": [
            "Google"
        ],
        "dashboardLogin": [
            "http://www.google.com"
        ],
        "ICFIdentifier": [
            "idm magic 34"
        ],
        "dashboardDisplayName": [
            "Google"
        ],
        "dashboardClassName": [
            "SAML2ApplicationClass"
        ]
    },
    "salesforce": {
        "dashboardIcon": [
            "salesforce.gif"
        ],
```

```
        "dashboardName": [
            "SalesForce"
        ],
        "dashboardLogin": [
            "http://www.salesforce.com"
        ],
        "ICFIdentifier": [
            "idm magic 12"
        ],
        "dashboardDisplayName": [
            "Salesforce"
        ],
        "dashboardClassName": [
            "SAML2ApplicationClass"
        ]
    },
    "zendesk": {
        "dashboardIcon": [
            "ZenDesk.gif"
        ],
        "dashboardName": [
            "ZenDesk"
        ],
        "dashboardLogin": [
            "http://www.ZenDesk.com"
        ],
        "ICFIdentifier": [
            "idm magic 56"
        ],
        "dashboardDisplayName": [
            "ZenDesk"
        ],
        "dashboardClassName": [
            "SAML2ApplicationClass"
        ]
    }
}
```

If your application runs in a user-agent such as a browser, you can rely on OpenAM to handle authentication.

Chapter 7

RESTful Identity and Realm Management Services

This chapter shows how to use the OpenAM RESTful interfaces for identity and realm management.

In this chapter, long URLs are wrapped to fit the printed page, as some of the output is formatted for easier reading.

7.1. Identity Management

This section shows how to create, read, update, delete, and list identities using the RESTful APIs.

Important

OpenAM is not primarily an identity data store, nor is it provisioning software. For storing identity data, consider OpenDJ. For provisioning, consider OpenIDM. Both of these products provide REST APIs as well.

OpenAM has two REST APIs for managing identities.

- Under the `/json/agents`, `/json/groups`, and `/json/users`, you find the newer JSON-based APIs. The newer APIs follow the ForgeRock common REST pattern creating, reading, updating, deleting, and querying resources.

  Examples in this section do not repeat the authentication shown in Authorization & Policy Management. For browser-based clients, you can rely on OpenAM cookies rather than construct the header in your application. Managing agent profiles, groups, realms, and users with these APIs of course require authorization. The examples shown in this section were performed with the token ID gained after authenticating as OpenAM administrator.

  Although the examples here show user management, you can use `/json/agents`, `/json/groups`, `/json/realms` in similar fashion. See "Realm Management" for examples related to realms.

  The following sections cover this JSON-based API.

  - "Creating Identities"
  - "Reading Identities"
  - "Updating Identities"

- "Deleting Identities"
- "Listing Identities"
- "Changing Passwords"

- Under the `/identity` endpoint, you find the backwards-compatible, legacy API.

  The following sections cover this backwards-compatible API.

  - "Creating Identities (Legacy API)"
  - "Reading & Searching for Identities (Legacy API)"
  - "Updating Identities (Legacy API)"
  - "Deleting Identities (Legacy API)"

7.1.1. Creating Identities

OpenAM lets administrators create a user profile by making an HTTP POST of the JSON representation of the profile to `/json/subrealm/users/?_action=create`. To add a user to the Top Level Realm, you do not need to specify the realm.

The following example shows an administrator creating a new user. The only required fields are `username` and `userpassword`. If no other name is provided, the entry you make for `username` defaults to both the user id and the user's last name.

```
$ curl \
 --request POST \
 --header "iplanetDirectoryPro: AQIC5w...2NzEz*" \
 --header "Content-Type: application/json" \
 --data \
 '{
   "username": "bjensen",
   "userpassword": "secret12",
   "mail": "bjensen@example.com"
 }' \
 https://openam.example.com:8443/openam/json/users/?_action=create
{
    "username": "bjensen",
    "realm": "/",
    "uid": [
        "bjensen"
    ],
    "mail": [
        "bjensen@example.com"
    ],
    "sn": [
        "bjensen"
    ],
    "userpassword": [
```

```
        "{SSHA}0pXpKLPRKCGY7g3YqZygJmKMW6IC2BLJimmlwg=="
    ],
    "cn": [
        "bjensen"
    ],
    "inetuserstatus": [
        "Active"
    ],
    "dn": [
        "uid=bjensen,ou=people,dc=openam,dc=forgerock,dc=org"
    ],
    "objectclass": [
        "person",
        "sunIdentityServerLibertyPPService",
        "sunFederationManagerDataStore",
        "inetorgperson",
        "iPlanetPreferences",
        "iplanet-am-auth-configuration-service",
        "organizationalperson",
        "sunFMSAML2NameIdentifier",
        "inetuser",
        "iplanet-am-managed-person",
        "sunAMAuthAccountLockout",
        "iplanet-am-user-service",
        "top"
    ],
    "universalid": [
        "id=bjensen,ou=user,dc=openam,dc=forgerock,dc=org"
    ]
}
```

Alternatively, administrators can create user profiles with specific user IDs by doing an HTTP PUT of the JSON representation of the changes to `/json/users/user-id`, as shown in the following example:

```
$ curl \
 --request PUT \
 --header "iplanetDirectoryPro: AQIC5w...2NzEz*" \
 --header "Content-Type: application/json" \
 --header "If-None-Match: *" \
 --data \
 '{
   "username": "janedoe",
   "userpassword": "secret12",
   "mail": "janedoe@example.com"
 }' \
 https://openam.example.com:8443/openam/json/users/janedoe
{
    "username": "janedoe",
    "realm": "/",
    "uid": [
        "janedoe"
    ],
    "mail": [
        "janedoe@example.com"
    ],
    "sn": [
        "janedoe"
```

```
    ],
    "userpassword": [
        "{SSHA}e4DJoxvYVW/nsp62XJf29ZADE16YQgrxK+XuKA=="
    ],
    "cn": [
        "janedoe"
    ],
    "inetuserstatus": [
        "Active"
    ],
    "dn": [
        "uid=janedoe,ou=people,dc=openam,dc=forgerock,dc=org"
    ],
    "objectclass": [
        "devicePrintProfilesContainer",
        "person",
        "sunIdentityServerLibertyPPService",
        "inetorgperson",
        "sunFederationManagerDataStore",
        "iPlanetPreferences",
        "iplanet-am-auth-configuration-service",
        "organizationalperson",
        "sunFMSAML2NameIdentifier",
        "inetuser",
        "forgerock-am-dashboard-service",
        "iplanet-am-managed-person",
        "iplanet-am-user-service",
        "sunAMAuthAccountLockout",
        "top"
    ],
    "universalid": [
        "id=janedoe,ou=user,dc=openam,dc=forgerock,dc=org"
    ]
}
```

As shown in the examples, OpenAM returns the JSON representation of the profile on successful creation. On failure, OpenAM returns a JSON representation of the error including the HTTP status code. For example, version 2.0 of the CREST `/json/users`, `/json/groups`, and `/json/agents` endpoints return 403 if the user making the request is not authorized to do so.

The same HTTP POST and PUT mechanisms also work for other objects such as policy agent profiles and groups.

```
$ curl \
 --request POST \
 --header "iPlanetDirectoryPro: AQIC5w...2NzEz*" \
 --header "Content-Type: application/json" \
 --data \
 '{
   "name":"myWebAgent","realm":"/",
   "com.sun.identity.agents.config.fqdn.default":["www.example.com"],
   "sunidentityserverdevicekeyvalue":
     ["agentRootURL=http://www.example.com:80/"],
   "com.sun.identity.agents.config.remote.logfile":
     ["amAgent_www_example_com_80.log"],
   "com.sun.identity.agents.config.repository.location":["centralized"],
```

```
   "agenttype":["WebAgent"],
    "com.sun.identity.agents.config.cdsso.cdcservlet.url":
     ["[0]=https://openam.example.com:8443/openam/cdcservlet"],
   "com.sun.identity.client.notification.url":
     ["http://www.example.com:80/UpdateAgentCacheServlet?shortcircuit=false"],
   "com.sun.identity.agents.config.agenturi.prefix":
     ["http://www.example.com:80/amagent"],
   "userpassword":["password"],
   "com.sun.identity.agents.config.login.url":
     ["[0]=https://openam.example.com:8443/openam/UI/Login"],
   "com.sun.identity.agents.config.logout.url":
     ["[0]=https://openam.example.com:8443/openam/UI/Logout"],
   "sunidentityserverdevicestatus":["Active"]
 }' \
 https://openam.example.com:8443/openam/json/agents/?_action=create
{
    "name": "myWebAgent",
    "realm": "/",
    "com.sun.identity.agents.config.cdsso.enable": [
        "false"
    ],
    "com.sun.identity.agents.config.cdsso.cookie.domain": [
        "[0]="
    ],
    "com.sun.identity.agents.config.get.client.host.name": [
        "false"
    ],
    "com.sun.identity.agents.config.profile.attribute.fetch.mode": [
        "NONE"
    ],
    "com.sun.identity.agents.config.notenforced.ip": [
        "[0]="
    ],
    "com.sun.identity.agents.config.fqdn.check.enable": [
        "true"
    ],
    "com.sun.identity.agents.config.cleanup.interval": [
        "30"
    ],
    "com.sun.identity.agents.config.notenforced.url.attributes.enable": [
        "false"
    ],
    "com.sun.identity.agents.config.ignore.preferred.naming.url": [
        "true"
    ],
    "com.sun.identity.agents.config.client.ip.header": [],
    "com.sun.identity.agents.config.session.attribute.mapping": [
        "[]="
    ],
    "com.sun.identity.agents.config.audit.accesstype": [
        "LOG_NONE"
    ],
    "com.sun.identity.agents.config.proxy.override.host.port": [
        "false"
    ],
    "com.sun.identity.agents.config.load.balancer.enable": [
        "false"
    ],
    "com.sun.identity.agents.config.encode.url.special.chars.enable": [
```

```
        "false"
    ],
    "com.sun.identity.agents.config.convert.mbyte.enable": [
        "false"
    ],
    "com.sun.identity.agents.config.domino.check.name.database": [
        "false"
    ],
    "com.sun.identity.agents.config.iis.owa.enable": [
        "false"
    ],
    "com.sun.identity.agents.config.override.port": [
        "false"
    ],
    "com.sun.identity.agents.config.policy.clock.skew": [
        "0"
    ],
    "com.sun.identity.agents.config.sso.only": [
        "false"
    ],
    "com.sun.identity.agents.config.iis.owa.enable.session.timeout.url": [],
    "com.sun.identity.agents.config.domino.ltpa.config.name": [
        "LtpaToken"
    ],
    "com.sun.identity.agents.config.cookie.reset": [
        "[0]="
    ],
    "com.sun.identity.agents.config.fqdn.default": [
        "www.example.com"
    ],
    "sunIdentityServerDeviceKeyValue": [
        "agentRootURL=http://www.example.com:80/"
    ],
    "com.sun.identity.agents.config.domino.ltpa.cookie.name": [
        "LtpaToken"
    ],
    "com.sun.identity.agents.config.iis.password.header": [
        "false"
    ],
    "com.sun.identity.agents.config.response.attribute.mapping": [
        "[]="
    ],
    "com.sun.identity.agents.config.userid.param.type": [
        "session"
    ],
    "com.sun.identity.agents.config.url.comparison.case.ignore": [
        "true"
    ],
    "com.sun.identity.agents.config.profile.attribute.cookie.maxage": [
        "300"
    ],
    "com.sun.identity.agents.config.remote.logfile": [
        "amAgent_www_example_com_80.log"
    ],
    "com.sun.identity.agents.config.domino.ltpa.enable": [
        "false"
    ],
    "com.sun.identity.agents.config.notenforced.url": [
        "[0]="
```

```
    ],
    "com.sun.identity.agents.config.notification.enable": [
        "true"
    ],
    "com.sun.identity.agents.config.profile.attribute.cookie.prefix": [
        "HTTP_"
    ],
    "com.sun.identity.agents.config.logout.cookie.reset": [
        "[0]="
    ],
    "com.sun.identity.agents.config.polling.interval": [
        "60"
    ],
    "com.sun.identity.agents.config.attribute.multi.value.separator": [
        "|"
    ],
    "com.sun.identity.agents.config.debug.file.rotate": [
        "true"
    ],
    "com.sun.identity.agents.config.debug.level": [
        "Error"
    ],
    "com.sun.identity.agents.config.local.log.rotate": [
        "false"
    ],
    "com.sun.identity.agents.config.repository.location": [
        "centralized"
    ],
    "com.sun.identity.agents.config.client.ip.validation.enable": [
        "false"
    ],
    "com.sun.identity.agents.config.override.protocol": [
        "false"
    ],
    "AgentType": [
        "WebAgent"
    ],
    "com.sun.identity.agents.config.logout.redirect.url": [],
    "com.sun.identity.agents.config.ignore.path.info": [
        "false"
    ],
    "com.sun.identity.agents.config.override.notification.url": [
        "false"
    ],
    "com.sun.identity.agents.config.session.attribute.fetch.mode": [
        "NONE"
    ],
    "com.sun.identity.agents.config.policy.cache.polling.interval": [
        "3"
    ],
    "com.sun.identity.agents.config.cdsso.cdcservlet.url": [
        "[0]=https://openam.example.com:8443/openam/cdcservlet"
    ],
    "com.sun.identity.agents.config.cookie.name": [
        "iPlanetDirectoryPro"
    ],
    "com.sun.identity.agents.config.profile.attribute.mapping": [
        "[]="
    ],
```

```
    "com.sun.identity.agents.config.iis.filter.priority": [
        "HIGH"
    ],
    "com.sun.identity.agents.config.iis.auth.type": [],
    "com.sun.identity.client.notification.url": [
        "http://www.example.com:80/UpdateAgentCacheServlet?shortcircuit=false"
    ],
    "com.sun.identity.agents.config.cookie.secure": [
        "false"
    ],
    "com.sun.identity.agents.config.ignore.path.info.for.not.enforced.list": [
        "true"
    ],
    "com.sun.identity.agents.config.remote.log.interval": [
        "5"
    ],
    "com.sun.identity.agents.config.notenforced.url.invert": [
        "false"
    ],
    "universalid": [
        "id=myWebAgent,ou=agent,dc=openam,dc=forgerock,dc=org"
    ],
    "com.sun.identity.agents.config.replaypasswd.key": [],
    "com.sun.identity.agents.config.iis.owa.enable.change.protocol": [
        "false"
    ],
    "com.sun.identity.agents.config.userid.param": [
        "UserToken"
    ],
    "userpassword": [
        "{SHA-1}W6ph5Mm5Pz8GgiULbPgzG37mj9g="
    ],
    "com.sun.identity.agents.config.response.attribute.fetch.mode": [
        "NONE"
    ],
    "com.sun.identity.agents.config.freeformproperties": [
        "sunidentityserverdevicestatus=Active",
        "sunidentityserverdevicekeyvalue=agentRootURL=http://www.example.com:80/",
        "realm=/",
        "name=myWebAgent"
    ],
    "com.sun.identity.agents.config.postdata.preserve.enable": [
        "false"
    ],
    "com.sun.identity.agents.config.log.disposition": [
        "REMOTE"
    ],
    "com.sun.identity.agents.config.agenturi.prefix": [
        "http://www.example.com:80/amagent"
    ],
    "com.sun.identity.agents.config.override.host": [
        "false"
    ],
    "com.sun.identity.agents.config.cookie.reset.enable": [
        "false"
    ],
    "com.sun.identity.agents.config.local.log.size": [
        "52428800"
    ],
```

```
    "com.sun.identity.agents.config.access.denied.url": [],
    "com.sun.identity.agents.config.debug.file.size": [
        "10000000"
    ],
    "com.sun.identity.agents.config.change.notification.enable": [
        "true"
    ],
    "com.sun.identity.agents.config.anonymous.user.enable": [
        "false"
    ],
    "com.sun.identity.agents.config.domino.ltpa.org.name": [],
    "com.sun.identity.agents.config.agent.logout.url": [
        "[0]="
    ],
    "com.sun.identity.agents.config.poll.primary.server": [
        "5"
    ],
    "com.sun.identity.agents.config.fqdn.mapping": [
        "[]="
    ],
    "com.sun.identity.agents.config.auth.connection.timeout": [
        "2"
    ],
    "com.sun.identity.agents.config.client.hostname.header": [],
    "com.sun.identity.agents.config.iis.logonuser": [
        "false"
    ],
    "com.sun.identity.agents.config.ignore.server.check": [
        "false"
    ],
    "com.sun.identity.agents.config.fetch.from.root.resource": [
        "false"
    ],
    "com.sun.identity.agents.config.login.url": [
        "[0]=https://openam.example.com:8443/openam/UI/Login"
    ],
    "com.sun.identity.agents.config.redirect.param": [
        "goto"
    ],
    "com.sun.identity.agents.config.logout.url": [
        "[0]=https://openam.example.com:8443/openam/UI/Logout"
    ],
    "sunIdentityServerDeviceStatus": [
        "Active"
    ],
    "com.sun.identity.agents.config.sso.cache.polling.interval": [
        "3"
    ],
    "com.sun.identity.agents.config.anonymous.user.id": [
        "anonymous"
    ],
    "com.sun.identity.agents.config.encode.cookie.special.chars.enable": [
        "false"
    ],
    "com.sun.identity.agents.config.locale": [
        "en_US"
    ],
    "com.sun.identity.agents.config.postcache.entry.lifetime": [
        "10"
```

```
    ]
}
```

```
$ curl \
 --request POST \
 --header "iPlanetDirectoryPro: AQIC5w...2NzEz*" \
 --header "Content-Type: application/json" \
 --data '{
   "username":"newGroup",
   "uniquemember":["uid=demo,ou=people,dc=openam,dc=forgerock,dc=org"]
 }' \
 https://openam.example.com:8443/openam/json/groups?_action=create
{
    "username": "newGroup",
    "realm": "/",
    "uniqueMember": [
        "uid=demo,ou=people,dc=openam,dc=forgerock,dc=org"
    ],
    "cn": [
        "newGroup"
    ],
    "dn": [
        "cn=newGroup,ou=groups,dc=openam,dc=forgerock,dc=org"
    ],
    "objectclass": [
        "groupofuniquenames",
        "top"
    ],
    "universalid": [
        "id=newGroup,ou=group,dc=openam,dc=forgerock,dc=org"
    ]
}

$ curl \
 --request PUT \
 --header "If-None-Match: *" \
 --header "iPlanetDirectoryPro: AQIC5w...2NzEz*" \
 --header "Content-Type: application/json" \
 --data '{
   "username":"anotherGroup",
   "uniquemember":["uid=demo,ou=people,dc=openam,dc=forgerock,dc=org"]
 }' \
 https://openam.example.com:8443/openam/json/groups/anotherGroup
{
    "username": "anotherGroup",
    "realm": "/",
    "uniqueMember": [
        "uid=demo,ou=people,dc=openam,dc=forgerock,dc=org"
    ],
    "cn": [
        "anotherGroup"
    ],
    "dn": [
        "cn=anotherGroup,ou=groups,dc=openam,dc=forgerock,dc=org"
    ],
    "objectclass": [
        "groupofuniquenames",
```

```
        "top"
    ],
    "universalid": [
        "id=anotherGroup,ou=group,dc=openam,dc=forgerock,dc=org"
    ]
}
```

7.1.2. Reading Identities

OpenAM lets users and administrators read profiles by requesting an HTTP GET on `/json/subrealm/users/user-id`. This allows users and administrators to verify user data, status, and directory. If users or administrators see missing or incorrect information, they can write down the correct information and add it using "Updating Identities". To read a profile on the Top Level Realm, you do not need to specify the realm.

Users can review the data associated with their accounts and administrators can read other user's profiles. The following example shows an administrator accessing user data. Users can view their information by changing `username=amadmin` to *`user-id`*.

```
$ curl \
 --header "iplanetDirectoryPro: AQIC5w...2NzEz*" \
 https://openam.example.com:8443/openam/json/users/demo
{
    "username": "demo",
    "realm": "dc=openam,dc=forgerock,dc=org",
    "uid": [
        "demo"
    ],
    "userpassword": [
        "{SSHA}BKPAKRS3QKkvQRw25MfXbVC4VEuVNUf+yCaejg=="
    ],
    "sn": [
        "demo"
    ],
    "cn": [
        "demo"
    ],
    "inetuserstatus": [
        "Active"
    ],
    "dn": [
        "uid=demo,ou=people,dc=openam,dc=forgerock,dc=org"
    ],
    "objectclass": [
        "devicePrintProfilesContainer",
        "person",
        "sunIdentityServerLibertyPPService",
        "inetorgperson",
        "sunFederationManagerDataStore",
        "iPlanetPreferences",
        "iplanet-am-auth-configuration-service",
        "organizationalperson",
        "sunFMSAML2NameIdentifier",
```

```
        "inetuser",
        "forgerock-am-dashboard-service",
        "iplanet-am-managed-person",
        "iplanet-am-user-service",
        "sunAMAuthAccountLockout",
        "top"
    ],
    "universalid": [
        "id=demo,ou=user,dc=openam,dc=forgerock,dc=org"
    ]
}
```

Use the `_fields` query string parameter to restrict the list of attributes returned. This parameter takes a comma-separated list of JSON object fields to include in the result.

```
$ curl \
 --header "iPlanetDirectoryPro: AQIC5w...2NzEz*" \
 https://openam.example.com:8443/openam/json/users/demo?_fields=username,uid
{"username":"demo","uid":["demo"]}
```

As shown in the examples, OpenAM returns the JSON representation of the profile on success. On failure, OpenAM returns a JSON representation of the error including the HTTP status code.

Using HTTP GET to read also works for other objects such as agent profiles and groups.

```
$ curl \
 --header "iplanetDirectoryPro: AQIC5w...2NzEz*" \
 https://openam.example.com:8443/openam/json/agents/myClientID
{
    "username": "myClientID",
    "realm": "dc=openam,dc=forgerock,dc=org",
    "com.forgerock.openam.oauth2provider.accesstoken": [],
    "com.forgerock.openam.oauth2provider.clientsessionuri": [],
    "com.forgerock.openam.oauth2provider.defaultscopes": [
        "[0]="
    ],
    "com.forgerock.openam.oauth2provider.clientname": [],
    "com.forgerock.openam.oauth2provider.clienttype": [
        "Confidential"
    ],
    "universalid": [
        "id=myClientID,ou=agent,dc=openam,dc=forgerock,dc=org"
    ],
    "com.forgerock.openam.oauth2provider.responsetypes": [
        "[6]=code token id_token",
        "[0]=code",
        "[2]=id_token",
        "[4]=token id_token",
        "[3]=code token",
        "[1]=token",
        "[5]=code id_token"
    ],
    "userpassword": [
```

```
        "{SHA-1}W6ph5Mm5Pz8GgiULbPgzG37mj9g="
    ],
    "com.forgerock.openam.oauth2provider.name": [
        "[0]="
    ],
    "com.forgerock.openam.oauth2provider.redirectionuris": [
        "[0]="
    ],
    "com.forgerock.openam.oauth2provider.idtokensignedresponsealg": [
        "HS256"
    ],
    "com.forgerock.openam.oauth2provider.scopes": [
        "[0]="
    ],
    "com.forgerock.openam.oauth2provider.postlogoutredirecturi": [],
    "sunidentityserverdevicestatus": [
        "Active"
    ],
    "agenttype": [
        "OAuth2Client"
    ],
    "com.forgerock.openam.oauth2provider.description": [
        "[0]="
    ]
}
```

The `_prettyPrint` query string parameter can make the resulting JSON easier to read when you are viewing the resulting JSON directly.

```
$ curl \
 --header "iPlanetDirectoryPro: AQIC5w...2NzEz*" \
 https://openam.example.com:8443/openam/json/groups/myGroup?_prettyPrint=true
{
    "username": "myGroup",
    "realm": "dc=openam,dc=forgerock,dc=org",
    "uniquemember": [
        "uid=demo,ou=people,dc=openam,dc=forgerock,dc=org"
    ],
    "cn": [
        "myGroup"
    ],
    "dn": [
        "cn=myGroup,ou=groups,dc=openam,dc=forgerock,dc=org"
    ],
    "objectclass": [
        "groupofuniquenames",
        "top"
    ],
    "universalid": [
        "id=myGroup,ou=group,dc=openam,dc=forgerock,dc=org"
    ]
}
```

7.1.3. Updating Identities

OpenAM lets users update their own profiles, and lets administrators update other users' profiles. To update an identity do an HTTP PUT of the JSON representation of the changes to `/json/subrealm/users/user-id`. To update a profile on the Top Level Realm, you do not need to specify the realm.

The following example shows how users can update their own profiles.

```
$ curl \
 --request PUT \
 --header "iplanetDirectoryPro: AQIC5...Y3MTAx*" \
 --header "Content-Type: application/json" \
 --data '{ "mail": "demo@example.com" }' \
 https://openam.example.com:8443/openam/json/users/demo
{
    "username": "demo",
    "realm": "/",
    "uid": [
        "demo"
    ],
    "mail": [
        "demo@example.com"
    ],
    "sn": [
        "demo"
    ],
    "userpassword": [
        "{SSHA}S14oR2gusLWtiDkAS4twj63slXNNaMKpwrOWdw=="
    ],
    "cn": [
        "demo"
    ],
    "inetuserstatus": [
        "Active"
    ],
    "dn": [
        "uid=demo,ou=people,dc=openam,dc=forgerock,dc=org"
    ],
    "objectclass": [
        "person",
        "sunIdentityServerLibertyPPService",
        "sunFederationManagerDataStore",
        "inetorgperson",
        "iPlanetPreferences",
        "iplanet-am-auth-configuration-service",
        "organizationalperson",
        "sunFMSAML2NameIdentifier",
        "inetuser",
        "iplanet-am-managed-person",
        "sunAMAuthAccountLockout",
        "iplanet-am-user-service",
        "top"
    ],
    "universalid": [
        "id=demo,ou=user,dc=openam,dc=forgerock,dc=org"
    ]
}
```

As shown in the example, OpenAM returns the JSON representation of the profile on success. On failure, OpenAM returns a JSON representation of the error including the HTTP status code.

You can use HTTP PUT to update other objects as well, such as policy agent profiles and groups.

The following example creates a web policy agent profile.

```
$ curl \
 --request PUT \
 --header "iPlanetDirectoryPro: AQIC5...Y3MTAx*" \
 --header "Content-Type: application/json" \
 --header "If-None-Match: *" \
 --data '{
  "username" : "myWebAgent",
  "realm" : "/",
  "com.sun.identity.agents.config.fqdn.default" : [ "www.example.com" ],
  "sunIdentityServerDeviceKeyValue" :
   [ "agentRootURL=http://www.example.com:80/" ],
  "com.sun.identity.agents.config.remote.logfile" :
   [ "amAgent_www_example_com_80.log" ],
  "com.sun.identity.agents.config.repository.location" : [ "centralized" ],
  "AgentType" : [ "WebAgent" ],
  "com.sun.identity.agents.config.cdsso.cdcservlet.url" :
   [ "[0]=https://openam.example.com:8443/openam/cdcservlet" ],
  "com.sun.identity.client.notification.url" :
   [ "http://www.example.com:80/UpdateAgentCacheServlet?shortcircuit=false" ],
  "universalid" : [ "id=myWebAgent,ou=agent,dc=openam,dc=forgerock,dc=org" ],
  "userpassword" : [ "changeit" ],
  "com.sun.identity.agents.config.agenturi.prefix" :
   [ "http://www.example.com:80/amagent" ],
  "com.sun.identity.agents.config.login.url" :
   [ "[0]=https://openam.example.com:8443/openam/UI/Login" ],
  "com.sun.identity.agents.config.logout.url" :
   [ "[0]=https://openam.example.com:8443/openam/UI/Logout" ],
  "sunIdentityServerDeviceStatus" : [ "Active" ]
 }' \
 https://openam.example.com:8443/openam/json/agents/myWebAgent?_prettyPrint=true
```

When you create a policy agent profile, OpenAM returns the full profile in JSON format.

Notice in the following example that updates `myGroup` the object class value is not included in the JSON sent to OpenAM.

```
$ curl \
 --request PUT \
 --header "iPlanetDirectoryPro: AQIC5...Y3MTAx*" \
 --header "Content-Type: application/json" \
 --data '{
   "name":"myGroup",
   "realm":"/",
   "uniquemember":["uid=demo,ou=people,dc=openam,dc=forgerock,dc=org"],
   "cn":["myGroup"],
```

```
   "description":["Updated the group"]
 }' \
 https://openam.example.com:8443/openam/json/groups/myGroup
{
    "name": "myGroup",
    "realm": "/",
    "uniqueMember": [
        "uid=demo,ou=people,dc=openam,dc=forgerock,dc=org"
    ],
    "cn": [
        "myGroup"
    ],
    "dn": [
        "cn=myGroup,ou=groups,dc=openam,dc=forgerock,dc=org"
    ],
    "objectclass": [
        "groupofuniquenames",
        "top"
    ],
    "universalid": [
        "id=myGroup,ou=group,dc=openam,dc=forgerock,dc=org"
    ]
}
```

7.1.4. Deleting Identities

OpenAM lets administrators delete a user profile by making an HTTP DELETE call to `/json/subrealm/users/user-id`. To delete a user from the Top Level Realm, you do not need to specify the realm.

The following example removes a user from the top level realm. Only administrators should delete users. The user id is the only field required to delete a user.

```
$ curl \
 --request DELETE \
 --header "iplanetDirectoryPro: AQIC5w...2NzEz*" \
 https://openam.example.com:8443/openam/json/users/bjensen
{"success":"true"}
```

On success, OpenAM returns a JSON object indicating success. On failure, OpenAM returns a JSON representation of the error including the HTTP status code.

You can use this same logic for other resources such as performing an HTTP DELETE of an agent profile or of a group.

```
$ curl \
 --request DELETE \
 --header "iplanetDirectoryPro: AQIC5w...2NzEz*" \
 https://openam.example.com:8443/openam/json/agents/myOAuth2ClientAgent
{"success":"true"}
```

```
$ curl \
 --request DELETE \
 --header "iPlanetDirectoryPro: AQIC5w...2NzEz*" \
 https://openam.example.com:8443/openam/json/groups/myGroup
{"success":"true"}
```

Note

Deleting a user does not automatically remove any of the user's sessions. After deleting a user, check for any sessions for the user and remove them under the Console's Sessions tab.

7.1.5. Listing Identities

OpenAM lets administrators list identities by making an HTTP GET call to `/json/subrealm/users/?_queryId=*`. To query the Top Level Realm, you do not need to specify the realm.

```
$ curl \
 --header "iPlanetDirectoryPro: AQIC5w...2NzEz*" \
 "https://openam.example.com:8443/openam/json/users?_queryID=*"
{
  "result" : [ "amAdmin", "demo", "anonymous" ],
  "resultCount" : 3,
  "pagedResultsCookie" : null,
  "remainingPagedResults" : -1
}
```

This also works for other types of objects, such as agent profiles and groups.

```
$ curl \
 --header "iPlanetDirectoryPro: AQIC5w...2NzEz*" \
 "https://openam.example.com:8443/openam/json/agents?_queryID=*"
{
  "result" : [ "wsp", "wsc", "agentAuth", "SecurityTokenService" ],
  "resultCount" : 4,
  "pagedResultsCookie" : null,
  "remainingPagedResults" : -1
}
```

```
$ curl \
 --header "iPlanetDirectoryPro: AQIC5w...2NzEz*" \
 "https://openam.example.com:8443/openam/json/groups?_queryID=*"
{
  "result" : [ "myOtherGroup", "myGroup" ],
  "resultCount" : 2,
  "pagedResultsCookie" : null,
  "remainingPagedResults" : -1
}
```

As the result lists include all objects, this capability to list identity names is mainly useful in testing.

As shown in the examples, OpenAM returns the JSON representation of the resource list if successful. On failure, OpenAM returns a JSON representation of the error including the HTTP status code.

7.1.6. Retrieving Identities Using the Session Cookie

If you only have access to the `iPlanetDirectoryPro` session cookie, you can retrieve the user ID by performing an HTTP POST operation on the `/json/users` endpoint using the `idFromSession` action.

```
$ curl \
 --verbose \
 --request POST \
 --header "iplanetdirectorypro: AQIC5wM2LY4SfczUFNs-TJwFrCVAKgR0NulIAyNaIkQmjis.*AAJTSQACMDEA
 AlNLABQtNTQ3NDE2Njc5ODk4MjYzMzA2MQ..*" \
 --header "Content-Type: application/json" http://openam.example.com:8080/openam/json/users?
_action=idFromSession

{
  "id":"demo",
  "realm":"/",
  "dn":"id=demo,ou=user,dc=openam,dc=forgerock,dc=org",
  "successURL":"/openam/console",
  "fullLoginURL":null
}
```

7.1.7. Changing Passwords

A user can change their own password with an HTTP POST to `/json/subrealm/users/username?_action=changePassword` including the new password as the value of `userpassword` in the request data.

A user must provide the current password, which is set in the request as the value of the `currentpassword`.

For the case where the user has forgotten their password, see Resetting Forgotten Passwords instead.

The following example shows a successful request to change the *demo* user's password to *password*.

```
$ curl \
 --request POST \
 --header "iPlanetDirectoryPro: AQIC5w...NTcy*" \
 --header "Content-Type: application/json" \
 --data '{
     "currentpassword":"changeit",
     "userpassword":"password"
 }' \
 https://openam.example.com:8443/openam/json/users/demo?_action=changePassword
{}
```

On success, the response is an empty JSON object {} as shown in the example.

On failure, OpenAM returns a JSON representation of the error including the HTTP status code. See also REST Status Codes for more information.

7.1.8. Creating Identities (Legacy API)

Interface Stability: Deprecated in the *Administration Guide*

OpenAM lets you create user profiles, and also create web and J2EE policy agent profiles. When you create an entry, you must provide the following parameters.

admin

Valid token for the user with permissions to add the identity

identity_name

A unique name for the identity to create

identity_attribute_names

LDAP attribute names for attributes to create

identity_attribute_values_*name*

LDAP attribute values for the identity to create. For example, `identity_attribute_names=sn&identity_attribute_values_sn=Jensen`.

identity_realm

The realm in which to create the identity

identity_type

Either `user` or `AgentOnly`

```
$ curl "https://openam.example.com:8443/openam/identity/create?\
admin=AQIC5wM2LY4SfcxSYA8eG-vrNHb_W7nG8XkfAGyRyuaebDY.*AAJTSQACMDE.*\
&identity_name=testuser\
&identity_attribute_names=cn\
&identity_attribute_values_cn=Test%20User\
&identity_attribute_names=sn\
&identity_attribute_values_sn=User\
&identity_attribute_names=userpassword\
&identity_attribute_values_userpassword=secret12\
&identity_realm=%2F\
&identity_type=user"
```

7.1.9. Reading & Searching for Identities (Legacy API)

Interface Stability: Deprecated in the *Administration Guide*

Reading is similar to attribute retrieval, as described in Token Validation, but obtained using the token of a user with permissions to perform the search, as shown in the following example.

```
$ curl "https://openam.example.com:8443/openam/identity/read?\
admin=AQIC5wM2LY4SfcxSYA8eG-vrNHb_W7nG8XkfAGyRyuaebDY.*AAJTSQACMDE.*\
&name=testuser\
&attributes_names=realm\
&attributes_values_realm=%2F"
identitydetails.name=testuser
identitydetails.type=user
identitydetails.realm=o=openam
identitydetails.attribute=
identitydetails.attribute.name=uid
identitydetails.attribute.value=testuser
identitydetails.attribute=
identitydetails.attribute.name=sn
identitydetails.attribute.value=User
identitydetails.attribute=
identitydetails.attribute.name=userpassword
identitydetails.attribute.value={SSHA}AzpT+N1sjrQhL1wfX2ETWh/Aqbd+lH9LOlhDqg==
identitydetails.attribute=
identitydetails.attribute.name=cn
identitydetails.attribute.value=Test User
identitydetails.attribute=
identitydetails.attribute.name=inetuserstatus
identitydetails.attribute.value=Active
identitydetails.attribute=
identitydetails.attribute.name=dn
identitydetails.attribute.value=uid=testuser,ou=people,dc=example,dc=com
identitydetails.attribute=
identitydetails.attribute.name=objectclass
identitydetails.attribute.value=person
identitydetails.attribute.value=sunIdentityServerLibertyPPService
identitydetails.attribute.value=inetorgperson
identitydetails.attribute.value=sunFederationManagerDataStore
identitydetails.attribute.value=iPlanetPreferences
identitydetails.attribute.value=iplanet-am-auth-configuration-service
identitydetails.attribute.value=organizationalperson
identitydetails.attribute.value=sunFMSAML2NameIdentifier
identitydetails.attribute.value=inetuser
identitydetails.attribute.value=iplanet-am-managed-person
identitydetails.attribute.value=iplanet-am-user-service
identitydetails.attribute.value=sunAMAuthAccountLockout
identitydetails.attribute.value=top
identitydetails.attribute=
identitydetails.attribute.name=universalid
identitydetails.attribute.value=id=testuser,ou=user,o=openam
```

You can search for user IDs by providing the following parameters.

admin

Valid token for the user with access to perform the search

attributes_names

LDAP attribute names for attributes to search

attributes_values_*name*

LDAP attribute values for the identity to search. For example, `attribute_names=sn&attribute_values_sn=Jensen`.

filter

Additional LDAP filter component to limit the search results returned

```
$ curl "https://openam.example.com:8443/openam/identity/search?\
admin=AQIC5wM2LY4SfcxSYA8eG-vrNHb_W7nG8XkfAGyRyuaebDY.*AAJTSQACMDE.*\
&attributes_names=sn\
&attributes_values_sn=Jensen\
&attributes_names=mail\
&attributes_values_mail=bjensen*\
&attributes_names=realm\
&attributes_values_realm=%2F"
string=bjensen
```

7.1.10. Updating Identities (Legacy API)

Interface Stability: Deprecated in the *Administration Guide*

You can update an identity with the same parameters used to create identities, provided the token corresponds to a user with access to update the identity.

```
$ curl "https://openam.example.com:8443/openam/identity/update?\
admin=AQIC5wM2LY4SfcxSYA8eG-vrNHb_W7nG8XkfAGyRyuaebDY.*AAJTSQACMDE.*\
&identity_name=testuser\
&identity_attribute_names=mail\
&identity_attribute_values_mail=testuser%40example.com\
&identity_realm=%2F\
&identity_type=user"
```

7.1.11. Deleting Identities (Legacy API)

Interface Stability: Deprecated in the *Administration Guide*

You can also delete an identity.

```
$ curl "https://openam.example.com:8443/openam/identity/delete?\
admin=AQIC5wM2LY4SfcxSYA8eG-vrNHb_W7nG8XkfAGyRyuaebDY.*AAJTSQACMDE.*\
&identity_name=testuser\
&identity_realm=%2F\
&identity_type=user"
```

7.2. Realm Management

This section shows how to create, read, update, and delete realms using the RESTful APIs.

- Under the `/json/realms` endpoint, you find the newer JSON-based API.

  The following sections cover this JSON-based API.

  - "Default Parameters for Realms"
  - "Creating Realms"
  - "Reading Realms"
  - "Listing Realms"
  - "Updating Realms"
  - "Deleting Realms"

7.2.1. Default Parameters for Realms

Realms have a number of fields entered with the default loading. The following table provides information on what the default realm settings are, and these settings can be updated, added, or deleted when updating a realm.

Realm Parameters for JSON-based API

Realm Parameter	Default	Purpose
realm	None - the only required field to add a realm	The name of the realm Example: `myRealm`
sunOrganizationStatus	Active	The status of the realm `Active` or `Inactive`
sunOrganizationAliases	None	Any applicable aliases associated with the realm. Be aware that an alias can only be used once. Entering an alias used by another

Realm Parameter	Default	Purpose
		realm will remove the alias from that realm and you will lose configuration. Example: `opensso.example.com`

serviceNames	`sunAMAuthHOTPService`
	`iPlanetAMAuthConfiguration`
	`sunAMAuthFederationService`
	`sunIdentityRepositoryService`
	`iPlanetAMPolicyConfigService`
	`iPlanetAMAuthService`
	`iPlanetAMAuthLDAPService`
	`sunAMAuthDataStoreService`
	`sunAMAuthSAEService`
	`sunAMDelegationService`
	`sunAMAuthWSSAuthModuleService`
	`iPlanetAMAuthOATHService`
		Services needed for the realm, including authentication modules

7.2.2. Creating Realms

OpenAM lets administrators create a realm by making an HTTP POST of the JSON representation of the profile to `/json/realms/?_action=create`.

You can create realms using an HTTP POST of the JSON representation of the profile to `/json/realms/?_action=create`, as shown in the following example. The only required field is `realm`, but the realm will not be active if the status is not set.

```
$ curl \
 --request POST \
 --header "iplanetDirectoryPro: AQIC5w...2NzEz*" \
 --header "Content-Type: application/json" \
 --data '{ "realm": "myRealm" }' \
 https://openam.example.com:8443/openam/json/realms/?_action=create
{"realmCreated":"/myRealm"}
```

Note

Do not use the names of OpenAM REST endpoints as the name of a realm. The OpenAM REST endpoint names that should not be used includes: "users", "groups", "realms", "policies" and "applications".

You can also set the `sunOrganizationAliases` parameter, but it can only be assigned to one realm (usually the top level realm). Before setting this parameter, make sure it is not already assigned elsewhere. If you replace remove it from another realm, you will lose your configuration.

Alternatively, administrators can create realms by the specific realm name using the HTTP PUT of the JSON representation of the changes to `/json/realms/realm-id`, as shown in the following example:

```
$ curl \
 --request PUT \
 --header "iplanetDirectoryPro: AQIC5w...2NzEz*" \
 --header "Content-Type: application/json" \
 --data '{ }' \
 https://openam.example.com:8443/openam/json/realms/myRealm
```

OpenAM returns the JSON representation of the profile on success. On failure, OpenAM returns a JSON representation of the error including the HTTP status code.

7.2.3. Reading Realms

OpenAM lets administrators read realms by requesting an HTTP GET on `/json/realms/realm-id`. This allows administrators to review all active realm services for the realm, like policy configuration and modules. If users or administrators see missing information (such as Active status) or incorrect information, they can write down the correct information and add it using "Updating Realms"

The following example shows an administrator receiving information about a realm called `myRealm`.

```
$ curl \
 --header "iplanetDirectoryPro: AQIC5w...2NzEz*" \
 https://openam.example.com:8443/openam/json/realms/myRealm
{
    "serviceNames":[
      "sunAMAuthHOTPService",
      "iPlanetAMAuthConfiguration",
      "sunAMAuthFederationService",
      "sunIdentityRepositoryService",
      "iPlanetAMPolicyConfigService",
      "iPlanetAMAuthService",
      "iPlanetAMAuthLDAPService",
      "sunAMAuthDataStoreService",
      "sunAMAuthSAEService",
      "sunAMDelegationService",
      "sunAMAuthWSSAuthModuleService",
      "iPlanetAMAuthOATHService"
    ]
}
```

As shown in the example, OpenAM returns the JSON representation of the profile on success. On failure, OpenAM returns a JSON representation of the error including the HTTP status code.

To read the top-level realm, use `toplevelrealm` with the `realms` endpoint.

```
$ curl \
 --header "iplanetDirectoryPro: AQIC5w...2NzEz*" \
 https://openam.example.com:8443/openam/json/realms/toplevelrealm
{
      "serviceNames" : [
       "sunAMAuthFederationService",
       "sunEntitlementIndexes",
       "iPlanetAMAuthService",
       "sunAMAuthDataStoreService",
       "sunAMAuthWSSAuthModuleService",
       "sunAMDelegationService",
       "iPlanetAMAuthOATHService",
       "iPlanetAMAuthConfiguration",
       "sunAMAuthHOTPService",
       "sunIdentityRepositoryService",
       "iPlanetAMPolicyConfigService",
       "iPlanetAMAuthLDAPService",
       "sunEntitlementService",
       "iPlanetAMPolicyService",
       "sunAMAuthSAEService",
       "AgentService" ]
}
```

If the realm you want to read is not an immediate subrealm of the top-level realm, specify its parent realm to the left of `realms` in the URL, and specify the realm's final qualifier to the right of `realms`. For example, to read the `/myRealm/myRealmsChildRealm` realm:

```
$ curl \
 --header "iplanetDirectoryPro: AQIC5w...2NzEz*" \
 https://openam.example.com:8443/openam/json/myRealm/realms/myRealmsChildRealm
{
      "serviceNames" : [
       "sunAMAuthHOTPService",
       "iPlanetAMAuthConfiguration",
       "sunAMAuthFederationService",
       "sunIdentityRepositoryService",
       "iPlanetAMPolicyConfigService",
       "iPlanetAMAuthService",
       "iPlanetAMAuthLDAPService",
       "sunAMAuthDataStoreService",
       "sunAMAuthSAEService",
       "sunAMDelegationService",
       "sunAMAuthWSSAuthModuleService",
       "iPlanetAMAuthOATHService"
      ]
}
```

7.2.4. Listing Realms

To list a realm and its subrealms, perform an HTTP GET on the endpoint, setting the `_queryFilter` query string parameter as in the following example, which lists the top-level realm and all of its subrealms:

```
$ curl \
--header "iPlanetDirectoryPro: AQIC5..." \
https://openam.example.com:8443/openam/json/realms?_queryFilter=true
{
    "result" : [ "/", "/myRealm", "/myRealm/myRealmsChildRealm" ],
    "resultCount" : 3,
    "pagedResultsCookie" : null,
    "remainingPagedResults" : -1
}
```

You can start listing realms from below the top-level realm by placing the starting realm name in the URL. The following example lists the realm `myRealm` and all of its subrealms.

```
$ curl \
 --header "iPlanetDirectoryPro: AQIC5..." \
 https://openam.example.com:8443/openam/json/myRealm/realms?_queryFilter=true
 {
     "result" : [ "/myRealm", "/myRealm/myRealmsChildRealm" ],
     "resultCount" : 2,
     "pagedResultsCookie" : null,
     "remainingPagedResults" : -1
 }
```

7.2.5. Updating Realms

OpenAM lets administrators update realms. To update a realm, do an HTTP PUT of the JSON representation of the changes to `/json/realms/realm-id`.

The following example shows how to update a realm called `myRealm`. The example command sets the realm's status to Inactive.

```
$ curl \
 --request PUT \
 --header "iplanetDirectoryPro: AQIC5...Y3MTAx*" \
 --header "Content-Type: application/json" \
 --data '{ "sunOrganizationStatus": "Inactive" }' \
 https://openam.example.com:8443/openam/json/realms/myRealm
```

OpenAM returns the JSON representation of the profile on success. On failure, OpenAM returns a JSON representation of the error including the HTTP status code.

7.2.6. Deleting Realms

OpenAM lets administrators delete a realm by making an HTTP DELETE call to `/json/realms/realm-id`.

The following example deletes a realm called `myRealm`. The top level realm cannot be deleted. Only administrators should delete realms. The name of the realm is the only field required to delete the realm.

Make sure that you do not have any information you need within a realm before deleting it. Once a realm is deleted, the only way to restore it is to return to a backed up deployment of OpenAM.

```
$ curl \
 --request DELETE \
 --header "iplanetDirectoryPro: AQIC5w...2NzEz*" \
 https://openam.example.com:8443/openam/json/realms/myRealm
{"success":"true"}
```

On success, OpenAM returns a JSON object indicating success. On failure, OpenAM returns a JSON representation of the error including the HTTP status code.

If the realm you want to delete is not an immediate subrealm of the top-level realm, specify its parent realm to the left of `realms` in the URL, and specify the realm's final qualifier to the right of `realms`. For example, to delete the `/myRealm/myRealmsChildRealm` realm:

```
$ curl \
--request DELETE
--header "iplanetDirectoryPro: AQIC5w...2NzEz*" \
https://openam.example.com:8443/openam/json/myRealm/realms/myRealmsChildRealm
{ "success":"true" }
```

Chapter 8

RESTful Secure Token Service

OpenAM provides a RESTful Security Token Service (STS) that allows OpenAM to bridge identities across web and enterprise identity access management (IAM) systems through its token transformation process.

If you are unfamiliar with STS, you can read the specification, which is part of the WS-Trust extensions to the Oasis WS-Security standard, governing the management of security tokens in secure message exchanges.

8.1. Introduction

OpenAM's REST STS framework provides a programmatic means to create STS instances, extending single sign-on (SSO) and other features across token boundaries and systems. You can also create STS instances using the Admin console.

How does it work? In simple terms, the claims in an x509 or OpenID Connect token are used to map to a subject in the OpenAM datastore, and the state associated with this principal is used to create the claims in the issued SAML2 assertion. Specifically, the process of authenticating the input token type involves mapping an input token state to a principal (for example, for a <username, password> combination). The issued SAML2 assertion asserts the identity of this subject, which can contain AttributeStatements that assert claims related to LDAP attribute state associated with this subject.

For example, you can create a REST STS instance programmatically or through the Admin console to transform a validated OpenID Connect (OIDC) 1.0 access token, whose principal is mapped to a new SAML 2.0 token, asserting claims mapped in the OIDC access token's scope.

The following token transformations are currently supported:

- OpenAM->SAML2
- UsernameToken->SAML2
- OpenID Connect ID Token->SAML2
- X509->SAML2

The generated SAML 2.0 token supports assertion signing and encryption as well as with the Bearer, Holder of Key, or Sender Vouches Subject confirmation:

- **Holder of Key**. Specifies that the subject must contain one or more `KeyInfo` elements within the `subjectConfirmationData` element so that service provider can validate that the requesting entity is in the possession of the key(s).
- **Bearer**. Specifies that the subject does not contain key material but the subject is considered an acceptable attesting entity. The service provider can accept the assertion or not, and optionally invoke other means or constraints specified in the `subjectConfirmationData` element.
- **Sender Vouches**. Specifies that the subject does not contain key material. The service provider can optionally invoke other methods to accept the assertion or not.

You can publish REST STS instances in a given realm, each with different configuration characteristics, or update an existing STS instance using the Admin Console (click Access Control > *Realm Name* > STS) or programmatically using the Client SDK.

8.2. REST STS Configuration

The following sections presents points about the REST STS configuration.

8.2.1. Instance Configuration

Each REST STS instance issues SAML 2.0 assertions for a single ServiceProvider. As a result, each REST STS instance is configured with the following elements:

- **Issuer**. Corresponds to the Identity Provider (IdP) EntityID.
- **Service Provider (SP) EntityID**. Used in the `AudienceRestriction` element of the `Conditions` of the issued assertion.
- **SP Assertion Consumer Service URL**. Used as the `Recipient` attribute of the `SubjectConfirmation` element in `Subject` elements, as required for Bearer assertions according to the Web SSO profile.

8.2.2. Signing and Encryption

For signing and encryption support, each REST STS instance has a configuration state specifying the keystore location containing the signing and encryption keys.

If you configure assertion signing, you must specify the keystore path and password, as well as the alias and password corresponding to the private key used to sign the assertion.

If you configure assertion encryption, you must specify the keystore path and password, as well as the alias corresponding to the SP's X509 certificate, which encapsulates the public key used to encrypt the symmetric key that encrypted the generated assertion.

Note

You can specify the keystore location using an absolute path on the local filesystem or a path relative to the OpenAM classpath.

You can encrypt the entire assertion, or the `NameID` and/or the `AttributeStatement` attributes.

8.2.3. Custom Plug-ins

All statements constituting a SAML 2.0 assertion can be fully customized. For each REST STS instance, you can write custom plug-ins for the `Conditions`, `Subject`, `AuthenticationStatements`, `AttributeStatements`, and `AuthorizationDecisionStatements` classes. If you specify the classes in the configuration of the published REST STS instance, these custom classes are consulted to provide the specific statements. See the interfaces in the `org.forgerock.openam.sts.tokengeneration.saml2.statements` package for details.

8.2.4. AuthnContext

Each REST STS instance must specify the `AuthnContext` in the `AuthenticationStatements` of the generated assertion. This `AuthnContext` allows the generated SAML 2.0 assertion to specify the manner in which the assertion's subject was authenticated. For a token transformation, this `AuthnContext` is a function of the input token type.

By default, the following `AuthnContext` string is included in the SAML2 assertion, generated as part of the transformation of the following input token types:

- OpenAM: `urn:oasis:names:tc:SAML:2.0:ac:classes:PreviousSession`
- UsernameToken and OpenID Connect ID Token: `urn:oasis:names:tc:SAML:2.0:ac:classes:PasswordProtectedTransport`
- X509 Token: `urn:oasis:names:tc:SAML:2.0:ac:classes:X509`

Note

You can override these default mappings by implementing the `org.forgerock.openam.sts.token.provider.AuthnContextMapper` interface, and specifying the name of this implementation in the configuration of the published REST STS instance.

8.2.5. SAML Attribute Mapping

You can set the mapping of SAML attribute names (Map keys) to local OpenAM attributes (Map values) using the `org.forgerock.openam.sts.tokengeneration.saml2.statements.DefaultAttributeMapper` class, which looks at profile attributes in the data stores or in the session properties for each published REST STS instance. The keys define the name of the attributes in the Assertion Attribute statements,

and the values define these attribute names. REST STS pulls the data from the subject's directory entry or from the session state that corresponds to the map value.

The keys can have the following format:

```
[NameFomatURI|]SAML ATTRIBUTE NAME
```

If the attribute value is enclosed in quotes, that quoted value is included in the attribute without mapping. Binary attributes should be followed by `;binary` as seen in the following example:

```
EmailAddress=mail,
Address=postaladdress,
urn:oasis:names:tc:SAML:2.0:attrname-format:uri|urn:mace:dir:attribute-def:cn=cn,
partnerID="staticPartnerIDValue",
urn:oasis:names:tc:SAML:2.0:attrname-format:uri|nameID="staticNameIDValue",
photo=photo;binary,
urn:oasis:names:tc:SAML:2.0:attrname-format:uri|photo=photo;binary
```

If this attribute mapping is insufficient, implement the `org.forgerock.openam.sts.tokengeneration.saml2.statements.AttributeMapper` interface and specify the name of the custom `AttributeMapper` in the configuration corresponding to your published REST STS instance.

8.3. Token Transformations

You can configure each REST STS instance with a distinct set of token transformations prgrammatically using the Client SDK or using the Admin Console. You select the transformations under the Supported Token Transforms under General Configuration, where you select a token transformation from a supported list.

Each transformation has the format:

```
X->Y;(don’t) invalidate interim OpenAM session
  where
    "X" refers to the input token type
    "Y" refers to the output token type (currently limited to SAML 2.0)
    "(don't) invalidate interim OpenAM session" specifies whether the interim
     OpenAM session, created during the authentication of the presented input token
     type, should be invalidated following the creation of the output token type.
     This setting allows a REST STS instance to support token transformations
     with or without a residual OpenAM session remaining after the token
     transformation.
```

8.3.1. Authentication

The input tokens in REST STS token transformations are authenticated via OpenAM’s restful authentication context. This context needs to be configured for every input token type in the set of

token transformations. For example, if you select the transformation `OPENIDCONNECT->SAML2`, the STS instance must be configured with information specifying which elements of the OpenAM restful authentication context needs to be consumed to validate the `OPENIDCONNECT` token.

The elements of the configuration tuple are separated by '|'.

- The first element is the input token type in the token transform: i.e. X509, OPENIDCONNECT, USERNAME, or OPENAM.
- The second element is the authentication target, either `module` or `service`.
- The third element is the name of the authentication module or service.
- The fourth element (optional) provides the STS authentication context information about the to-be-consumed authentication context.

8.3.2. Transforming OpenID Connect Tokens

When transforming OpenID Connect ID tokens, the OpenID Connect authentication module must be consumed, and thus a deployed REST STS instance must be configured with the name of the header/cookie element where the OpenID Connect ID token is placed. For example, the following Admin Console string defines these configurations:

```
OPENIDCONNECT|module|oidc|oidc_id_token_auth_target_header_key=oidc_id_token.
```

In this case, `oidc` is the name of the OpenID Connect authentication module created to authenticate OpenID Connect tokens. REST STS looks for the name of the header key in the string `oidc_id_token_auth_target_header_key`, which is the header where the `oidc` module expects to find the OpenID Connect ID Token.

Note

You can create these configurations in a more intuitive, fluent way by using the builders supporting the programmatic publishing of REST STS instances.

8.3.3. Transforming X509 Certificate

When transforming a X509 Certificate, the Certificate module must be consumed, and the published REST STS instance must be configured with the name of the Certificate module (or the service containing the module), and the header name configured for the Certificate module corresponding to where the Certificate module can expect to find the to-be-validated Certificate.

The following Admin Console string would define these configurations (note that these configurations can be created in a more intuitive, fluent way using the builders supporting the programmatic publishing of REST STS instances):

```
X509|module|cert_module|x509_token_token_auth_target_header_key=client_cert
```

In this case `cert_module` is the name of the Certificate module, and `client_cert` is the header name where Certificate module has been configured to find the client's Certificate. REST STS looks for the name of the header key in the string `x509_token_token_auth_target_header_keyy`, which is the header x509 ID Token.

8.3.4. Authentication x509

An x509 Certificate is a representation of subject identity only when ownership of the associated private key is demonstrated, as performed by the TLS-handshake, and described in the AsymmetricBinding examples 2.2.1 and 2.2.2 in WS-SecurityPolicy.

Because the REST STS is not protected by WS-SecurityPolicy bindings (this is the domain of the SOAP STS), a REST STS token transformation, which defines an x509 Certificate as an input token, can only be consumed by two-way TLS. Here the client must present their certificate, with the TLS handshake certifying that the private key corresponding to the presented certificate is in the possession of the certificate presenter. In such a deployment, REST STS obtains the client's certificate from the `javax.servlet.request.X509Certificate` attribute in the `ServletRequest`.

However, if OpenAM is deployed in a TLS-offloaded context, REST STS obtains the client's certificate from the header key defined in the published REST STS instance. The header key is specified in the "Client Certificate Header Key" under the "Deployment Configuration" section of a REST STS instance in the Admin Console, or in the `org.forgerock.openam.sts.rest.config.user. RestDeploymentConfig .RestDeploymentConfigBuilder#offloadedTwoWayTLSHeaderKey` method.

If a header key referencing a client's certificate is defined, then only this header is examined for the client's certificate. If this header is undefined, only the `javax.servlet.request.X509Certificate` ServletRequest attribute is examined. Note that the token transformation definition specifying an x509 input token does not actually encapsulate the caller's certificate. This token state only triggers REST STS to consult one of the two locations described above.

If the REST STS is configured to pull the client’s certificate out of a specified header, the set of hosts trusted to set this header must also be specified. See the "Trusted Remote Hosts" list in the "Deployment Configuration" section of the Admin Console or the `org.forgerock.openam.sts.rest.config .user.RestDeploymentConfig. RestDeploymentConfigBuilder#tlsOffloadEngineHostIpAddrs` method.

Note

An entry of ‘any’ in this list causes any caller to be trusted.

Authentication of the client-presented certificate is performed by the Certificate module. This module optionally performs certificate revocation list (CRL) or Online Certificate Status Protocol (OCSP) checking, and optionally checks to see that the specified certificate is in a LDAP datastore.

The Certificate module also expects to reference the caller's certificate in a specified header value, and that the IP addresses of hosts trusted to provide this information must also be specified. See the "Trusted Remote Hosts" and the "Http Header Name for Client Certificate" in the Certificate module configuration at Hints for the Certificate Authentication module in the *Administration Guide*. This header must be specified in the Authentication Target Mapping definition for X509 tokens for the published REST STS instance, so that the REST STS instance can know which header should reference the client's x509 certificate when consuming the Certificate module.

For example, given the following authentication target mapping:

```
X509|module|cert_module|x509_token_token_auth_target_header_key=client_cert
```

X509 specifies that for X509 input tokens, a module named `cert_module` should be invoked, and that this module expects to find the client's X509 Certificate in a header named `client_cert`. Thus the string `client_cert` must appear in the HTTP Header Name for Client Certificate configuration field for the Certificate module named `cert_module`. Also, the IP address of the OpenAM deployment must be added to the Trusted Remote Hosts configuration of the targeted Certificate module.

For more information on mapping x509 Certificate state to a subject in the OpenAM datastore, see Hints for the Certificate Authentication module in the *Administration Guide*.

8.3.5. Authentication: Username

The Username token passed to the REST STS represents the <username, password> combination in clear text. This means that REST STS instances that support Username Token-based token transformations should only be deployed on TLS.

8.3.6. Authentication: OpenID Connect

REST STS instances must be configured to consume an appropriately-configured OpenID Connect authentication module for OpenID Connect-based token transformations. The OIDC authentication module expects to reference the `id` token in a header value. Thus, the `AuthTargetMapping` of the published REST STS instance must be configured with the name of the header configured for the referenced OIDC authentication module.

The OIDC authentication module must be configured with either the discovery URL, the jwk URL or the client secret corresponding to the issuer, and the issuer name must be set (for OpenAM issued OIDC tokens, the issuer name corresponds to the OpenAM deployment url).

The OIDC authentication module also checks the authorized party (`azp`) and audience (`aud`) claims. The OIDC token `aud` claim must match that configured for the OIDC authentication module (that is, set to the OAuth2 client's ID). If the OIDC token has a `azp` claim, it must match that configured for the OIDC authentication module (again, set to the OAuth2 client ID). You must configure the attribute mappings, so that jwk claim state is mapped to an entry in the OpenAM user data store.

For additional details on the OpenID Connect authentication module, Hints for the OpenID Connect Module in the *Administration Guide*.

8.4. The Publish Service

REST STS instances configured via the Admin Console or programmatically, are published via the REST STS publish service, exposed at `<server-root>/rest-sts-publish/publish`. This restful interface can only be consumed by administrators.

Performing a GET on the `/rest-sts-publish/publish` URL returns the json representation of the configuration state corresponding to all published REST STS instances. Only administrators can perform this GET operation, that is, admin users with the iPDP cookie can receive the return state.

A particular REST STS instance is identified by:

```
Realm Name/{deployment_url_element},
  where
    Realm Name is the realm in which the REST STS instance is published.
    {deployment_url_element} is the "Deployment Url Element" set in the Admin
      Console or via the
      RestSTSDeploymentConfig.RestDeploymentConfigBuilder#uriElement(String uriElement
      method.
```

All published REST STS instances are exposed relative to `/rest-sts`, so if OpenAM is exposed at `https://host.com:443/openam`, and the REST STS instance is published in a realm called "accounting" with a Deployment URl element of `accountingsts`, then the URL would be:

```
https://host.com:443/openam/rest-sts/accounting/accountingsts
```

A json representation of this instance's configuration state could be examined by a GET method to:

```
https://host.com:443/openam/rest-sts-publish/publish/accounting/accountingsts
```

This instance could be deleted by performing a DELETE on `https://host.com:443/openam/rest-sts-publish/publish/accounting/accountingsts`.

Note

The REST STS Publish and Consume examples are contained within their own maven module (`openam-client-sts`), and the contents of this module are added to the `client-sdk`. To get the examples to work, there are a number of maven dependencies, specified in the `META-INF/maven/org.forgerock.openam/openam/client-sts/pom.xml` file in the `client-sdk.jar`. You should review these dependencies prior to setting up the examples.

8.5. REST STS Code Examples

The code example shows how to programmatically publish REST STS instances.

```
package org.forgerock.openam.sts.rest;
```

```
import org.forgerock.json.fluent.JsonException;
import org.forgerock.json.fluent.JsonValue;
import org.forgerock.openam.shared.sts.SharedSTSConstants;
import org.forgerock.openam.sts.AMSTSConstants;
import org.forgerock.openam.sts.config.user.AuthTargetMapping;
import org.forgerock.openam.sts.TokenType;
import org.forgerock.openam.sts.config.user.SAML2Config;
import org.forgerock.openam.sts.rest.config.user.RestDeploymentConfig;
import org.forgerock.openam.sts.rest.config.user.RestSTSInstanceConfig;
import org.forgerock.openam.sts.token.UrlConstituentCatenatorImpl;
import org.forgerock.openam.utils.IOUtils;
import org.forgerock.openam.utils.JsonValueBuilder;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStreamWriter;
import java.io.UnsupportedEncodingException;
import java.net.HttpURLConnection;
import java.net.InetAddress;
import java.net.MalformedURLException;
import java.net.URISyntaxException;
import java.net.URL;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.HashSet;
import java.util.List;
import java.util.Map;
import java.util.Set;
import static org.forgerock.json.fluent.JsonValue.field;
import static org.forgerock.json.fluent.JsonValue.json;
import static org.forgerock.json.fluent.JsonValue.object;

/**
 * This class provides an example of how to programatically publish REST STS
 * instances. It does not provide an exhaustive enumeration of configuration options,
 * but rather provides an indication of how to consume the various classes used to
 * build the configuration state corresponding to a published REST STS instance.
 */

public class RestSTSInstancePublisher {

    private static final String COOKIE = "Cookie";
    private static final String EQUALS = "=";
    private final String publishEndpoint;
    private final String spEntityId;
    private final String idpEntityId;
    private final String stsClientCertHeaderName;
    private final String spACSEndpoint;
    private final String adminSessionId;
    private final String createAction;

 /**
  *
  * @param publishEndpoint         The url of the publish service. For example,
  *                                http://myhost.com:8080/openam/rest-sts-publish/publish
  * @param spEntityId              The entity id of the Service Provider consuming the
  *                                created SAML2 assertions
  * @param idpEntityId             The entity id of the Identity Provider for which the
  *                                REST STS instance is issuing assertions
  * @param stsClientCertHeaderName If x509->SAML2 token transformations are being
```

```
 *                                provided in a tls-offloaded context, this value
 *                                specifies the header name that these offload
 *                                engines places the client's certificate.
 *                                Can be null.
 * @param spACSEndpoint           The url of the Service Provider's Assertion Consumer
 *                                Service. Required for Bearer assertions to conform to the
 *                                web-sso profile.
 * @param adminSessionId          The valid session id (iPlanetDirectoryPro cookie value)
 *                                for an OpenAM administrator. Required as only admins
 *                                can consume the REST STS publish service.
 * @param createAction            Used to append onto the end of the publishEndpoint string,
 *                                to specify a publish action. The value is ?_action=create
 */

public RestSTSInstancePublisher(String publishEndpoint, String spEntityId,
                                String idpEntityId, String stsClientCertHeaderName,
                                String spACSEndpoint, String adminSessionId,
                                String createAction) {
      this.publishEndpoint = publishEndpoint;
      this.spEntityId = spEntityId;
      this.idpEntityId = idpEntityId;
      this.stsClientCertHeaderName = stsClientCertHeaderName;
      this.spACSEndpoint = spACSEndpoint;
      this.adminSessionId = adminSessionId;
      this.createAction = createAction;
}

/**
An example of the json posted as part of this method invocation:
{
   "invocation_context": "invocation_context_client_sdk",
   "instance_state": {
      "issuer-name": "http://macbook.dirk.internal.forgerock.com:8080/openam",
      "saml2-config": {
         "saml2-name-id-format": "urn:oasis:names:tc:SAML:2.0:nameid-format:persistent",
         "saml2-token-lifetime-seconds": "600",
         "saml2-custom-conditions-provider-class-name": null,
         "saml2-custom-subject-provider-class-name": null,
         "saml2-custom-attribute-statements-provider-class-name": null,
         "saml2-custom-attribute-mapper-class-name": null,
         "saml2-custom-authn-context-mapper-class-name": null,
         "saml2-custom-authentication-statements-provider-class-name": null,
         "saml2-custom-authz-decision-statements-provider-class-name": null,
         "saml2-sign-assertion": "true",
         "saml2-encrypt-assertion": "true",
         "saml2-encrypt-attributes": "false",
         "saml2-encrypt-nameid": "false",
         "saml2-encryption-algorithm": "http://www.w3.org/2001/04/xmlenc#aes128-cbc",
         "saml2-encryption-algorithm-strength": "128",
         "saml2-attribute-map": {
            "email": "mail"
         },
         "saml2-keystore-filename": "/Users/DirkHogan/openam/openam/keystore.jks",
         "saml2-keystore-password": "changeit",
         "saml2-sp-acs-url": "http://macbook.dirk.internal.forgerock.com:18080/
                             openam/Consumer/metaAlias/sp",
         "saml2-sp-entity-id": "http://macbook.dirk.internal.forgerock.com:18080/
                               openam",
         "saml2-signature-key-alias": "test",
```

```
            "saml2-signature-key-password": "changeit",
            "saml2-encryption-key-alias": "test"
         },
         "deployment-config": {
            "deployment-url-element": "inst458259670",
            "deployment-realm": "/",
            "deployment-auth-target-mappings": {
               "[Ljava.security.cert.X509Certificate;": {
                  "mapping-auth-index-type": "module",
                  "mapping-auth-index-value": "cert_module",
                  "mapping-context": {
                     "x509_token_token_auth_target_header_key": "client_cert"
               }
            },
            "org.forgerock.openam.sts.token.model.OpenIdConnectIdToken": {
               "mapping-auth-index-type": "module",
               "mapping-auth-index-value": "oidc", "mapping-context": {
                  "oidc_id_token_auth_target_header_key": "oidc_id_token"
               }
            },
            "org.apache.ws.security.message.token.UsernameToken": {
               "mapping-auth-index-type": "service",
               "mapping-auth-index-value": "ldapService"
            }
         },
         "deployment-offloaded-two-way-tls-header-key": "client_cert",
         "deployment-tls-offload-engine-hosts": [
            "192.168.1.2",
            "127.0.0.1"
         ]
      },
      "supported-token-transforms": [
         {
            "inputTokenType": "OPENIDCONNECT",
            "outputTokenType": "SAML2",
            "invalidateInterimOpenAMSession": true
         },
         {
            "inputTokenType": "USERNAME",
            "outputTokenType": "SAML2",
            "invalidateInterimOpenAMSession": true
         },
         {
            "inputTokenType": "OPENAM",
            "outputTokenType": "SAML2",
            "invalidateInterimOpenAMSession": false
         },
         {
            "inputTokenType": "X509",
            "outputTokenType": "SAML2",
            "invalidateInterimOpenAMSession": true
         }
      ]
   }
}
*/

public String publishInstance(final String urlElement, String realm) throws Exception {
    RestSTSInstanceConfig instanceConfig = createRestSTSInstanceConfig(urlElement, realm);
```

```
    String jsonString = buildPublishInvocationJsonValue(instanceConfig).toString();
    String response = invokeRestSTSInstancePublish(jsonString);
    return parseInstanceUrl(response);
}

/**
  The fullSTSId should be the string returned by publishInstance.
*/

public String removeInstance(final String fullSTSId) throws Exception {
    return invokeRestSTSInstanceDeletion(getPublishRemoveInstanceUri(fullSTSId));
}

public List<RestSTSInstanceConfig> getPublishedInstances() throws Exception {
    String response = getPublishedRestSTSInstancesConfigContent(publishEndpoint);
    List<RestSTSInstanceConfig> instanceConfigs = new ArrayList<RestSTSInstanceConfig>();
    JsonValue json;
    try {
        json = JsonValueBuilder.toJsonValue(response);
    } catch (JsonException e) {
        System.out.println("JsonException parsing response: " + response + "\n Exception: "
         + e);
        throw e;
    }
    for (String key : json.asMap().keySet()) {
        JsonValue value = json.get(key);
        JsonValue jsonInstanceConfig = null;
        try {
            jsonInstanceConfig = JsonValueBuilder.toJsonValue(value.asString());
        } catch (JsonException e) {
            System.out.println("JsonException caught parsing json for instance.
             The value: " + value.asString() + "\n Exception: " + e);
        }
        RestSTSInstanceConfig instanceConfig
         = RestSTSInstanceConfig.fromJson(jsonInstanceConfig);
        instanceConfigs.add(instanceConfig);
    }
    return instanceConfigs;
}

private String getPublishedRestSTSInstancesConfigContent(String publishEndpoint)
    throws Exception {
    HttpURLConnection connection
     = (HttpURLConnection)new URL(publishEndpoint).openConnection();
    connection.setRequestMethod("GET");
    connection.connect();
    return readInputStream(connection.getInputStream());
}

private String invokeRestSTSInstancePublish(String invocationPayload)
    throws IOException {
    HttpURLConnection connection
     = (HttpURLConnection)getPublishAddInstanceUri().openConnection();
    connection.setDoOutput(true);
    connection.setRequestMethod("POST");
    connection.setRequestProperty(SharedSTSConstants.CONTENT_TYPE,
                                  SharedSTSConstants.APPLICATION_JSON);
    connection.setRequestProperty(COOKIE, getAdminSessionTokenCookie());
    OutputStreamWriter writer = new OutputStreamWriter(connection.getOutputStream());
```

```
    writer.write(invocationPayload);
    writer.close();

    if (connection.getResponseCode() == HttpURLConnection.HTTP_CREATED) {
        return getSuccessMessage(connection);
    } else {
        return getErrorMessage(connection);
    }
}

private String invokeRestSTSInstanceDeletion(String deletionUrl)
    throws IOException {
    HttpURLConnection connection
     = (HttpURLConnection)new URL(deletionUrl).openConnection();
    connection.setDoOutput(true);
    connection.setRequestMethod("DELETE");
    connection.setRequestProperty(SharedSTSConstants.CONTENT_TYPE,
                                  SharedSTSConstants.APPLICATION_JSON);
    connection.setRequestProperty(COOKIE, getAdminSessionTokenCookie());
    connection.connect();

    if (connection.getResponseCode() == HttpURLConnection.HTTP_OK) {
        return getSuccessMessage(connection);
    } else {
        return getErrorMessage(connection);
    }
}

private String getSuccessMessage(HttpURLConnection connection) throws IOException {
    return readInputStream(connection.getInputStream());
}

private String getErrorMessage(HttpURLConnection connection) throws IOException {
    if (connection.getErrorStream() != null) {
        return readInputStream(connection.getErrorStream());
    } else {
        return readInputStream(connection.getInputStream());
    }
}

private String getAdminSessionTokenCookie() {
    return "iPlanetDirectoryPro" + EQUALS + adminSessionId;
}

private String readInputStream(InputStream inputStream) throws IOException {
    if (inputStream == null) {
        return "Empty error stream";
    } else {
        return IOUtils.readStream(inputStream);
    }
}

private String parseInstanceUrl(String publishResponse) throws Exception {
    Object responseContent;
    try {
        org.codehaus.jackson.JsonParser parser =
         new org.codehaus.jackson.map.ObjectMapper()
             .getJsonFactory()
             .createJsonParser(publishResponse);
```

```
            responseContent = parser.readValueAs(Object.class);
        } catch (IOException e) {
            throw new Exception("Could not map the response from the PublishService
            to a json object. The response: " + publishResponse + "; The exception: " + e);
        }
        return new JsonValue(responseContent).get("url_element").asString();
    }

    private URL getPublishAddInstanceUri() throws MalformedURLException {
        return new URL(publishEndpoint + createAction);
    }

    private String getPublishRemoveInstanceUri(String stsId) throws URISyntaxException {
        return new UrlConstituentCatenatorImpl().catenateUrlConstituents(publishEndpoint,
                                                                         stsId);
    }

    private JsonValue buildPublishInvocationJsonValue(RestSTSInstanceConfig instanceConfig) {
       return json(object(field(AMSTSConstants.REST_STS_PUBLISH_INVOCATION_CONTEXT,
                                AMSTSConstants.REST_STS_PUBLISH_INVOCATION_CONTEXT_CLIENT_SDK),
                          field(AMSTSConstants.REST_STS_PUBLISH_INSTANCE_STATE,
                                instanceConfig.toJson())));
    }

    /**
     * This method creates the RestSTSInstanceConfig instance which determines the
     * nature of the published rest sts instance. Note that this method does not
     * take parameters corresponding to all options. It is there only to demonstrate
     * some of the options which could be set.
     * @param urlElement  The deployment url of the REST STS instance.
     * @param realm       The realm in which the rests-sts instance is deployed.
     *                    Note that the url of the published REST STS instance is
     *                    composed of the OpenAM deploymentUrl + realm + /urlElement.
     * @return            A RestSTSInstanceConfig configuring the published REST STS instance
     * @throws Exception  If something goes wrong
     */

     private RestSTSInstanceConfig createRestSTSInstanceConfig(String urlElement,
                                                               String realm) throws Exception {

     /**
         If you want to target a specific module or service for a particular token
         type, add it via the addMapping call below.
         See org.forgerock.openam.forgerockrest.authn.core.AuthIndexType for the
         list of valid authIndexType values (the second parameter in addMapping).
         The third parameter is simply the name of the specified module, service, etc.
         If you want to target the default service, don't add a mapping, or add a
         mapping corresponding to the default service, as below.
     */
     /**
         Build the context necessary for the OIDC token validation. The value in
         the map must correspond to the header which the OIDC module consults
         to obtain the oidc id token.
     */

     Map<String, String> oidcContext = new HashMap<String, String>();
     oidcContext.put(AMSTSConstants.OPEN_ID_CONNECT_ID_TOKEN_AUTH_TARGET_HEADER_KEY,
                     "oidc_id_token");
     /**
```

```
    Build the context necessary for Cert validation. The value in the map must
    correspond to the header which the Cert module consults to obtain the
    client's Certificate.
*/

Map<String, String> certContext = new HashMap<String, String>();
certContext.put(AMSTSConstants.X509_TOKEN_AUTH_TARGET_HEADER_KEY, "client_cert");
AuthTargetMapping mapping = AuthTargetMapping.builder()
    .addMapping(TokenType.USERNAME, "service", "ldapService")
    .addMapping(TokenType.OPENIDCONNECT, "module", "oidc", oidcContext)
    .addMapping(TokenType.X509, "module", "cert_module", certContext)
    .build();
RestDeploymentConfig.RestDeploymentConfigBuilder deploymentConfigBuilder =
 RestDeploymentConfig.builder()
    .uriElement(urlElement)
    .authTargetMapping(mapping)
    .realm(realm);

/**
If the clientCertHeader field is specified, this implies that the client cert
for x509 transformations should be specified in a header field. If this is the
case, then the set of IP addrs corresponding to the TLS offload engines must
also be specified. Simply specify the ip addr of this client, as it is
the one invoking token transformation functionality. If all hosts should be
trusted, add 'any' to the list.
*/
if (stsClientCertHeaderName != null) {
    Set<String> offloadHostsSet = new HashSet<String>();
    offloadHostsSet.add(InetAddress.getLocalHost().getHostAddress());
    offloadHostsSet.add("127.0.0.1");
    deploymentConfigBuilder
        .offloadedTwoWayTLSHeaderKey(stsClientCertHeaderName)
        .tlsOffloadEngineHostIpAddrs(offloadHostsSet);
}
RestDeploymentConfig deploymentConfig = deploymentConfigBuilder.build();
SAML2Config saml2Config;
try {
    Map<String, String> attributeMapping = new HashMap<String, String>();
    attributeMapping.put("email", "mail");
    saml2Config =
        SAML2Config.builder()
          .keystoreFile("/Users/DirkHogan/openam/openam/keystore.jks")  //Mac
        //.keystoreFile("/home/dhogan/openam/openam/keystore.jks") //linux
          .keystorePassword("changeit".getBytes(AMSTSConstants.UTF_8_CHARSET_ID))
          .encryptionKeyAlias("test")
          .signatureKeyAlias("test")
          .signatureKeyPassword("changeit".getBytes(AMSTSConstants.UTF_8_CHARSET_ID))
          .signAssertion(true)
          .encryptAssertion(true)
          .encryptionAlgorithm("http://www.w3.org/2001/04/xmlenc#aes128-cbc")
          .encryptionAlgorithmStrength(128)
          .attributeMap(attributeMapping)
          .nameIdFormat("urn:oasis:names:tc:SAML:2.0:nameid-format:persistent")
          .spEntityId(spEntityId)
          .spAcsUrl(spACSEndpoint)
          //custom statement providers could also be specified.
          .build();
} catch (UnsupportedEncodingException e) {
    throw new Exception(e);
```

```
    }
    return RestSTSInstanceConfig.builder()
        .deploymentConfig(deploymentConfig)
        .saml2Config(saml2Config)
        .issuerName(idpEntityId)
        .addSupportedTokenTranslation(
            TokenType.USERNAME,
            TokenType.SAML2,
            AMSTSConstants.INVALIDATE_INTERIM_OPENAM_SESSION)
        .addSupportedTokenTranslation(
            TokenType.OPENAM,
            TokenType.SAML2,
            !AMSTSConstants.INVALIDATE_INTERIM_OPENAM_SESSION)
        .addSupportedTokenTranslation(
            TokenType.OPENIDCONNECT,
            TokenType.SAML2,
            AMSTSConstants.INVALIDATE_INTERIM_OPENAM_SESSION)
        .addSupportedTokenTranslation(
            TokenType.X509,
            TokenType.SAML2,
            AMSTSConstants.INVALIDATE_INTERIM_OPENAM_SESSION)
        .build();
    }
}
```

The following code example shows how to consume REST STS token transformations.

```
package org.forgerock.openam.sts.rest;
import org.forgerock.json.fluent.JsonValue;
import org.forgerock.openam.shared.sts.SharedSTSConstants;
import org.forgerock.openam.sts.AMSTSConstants;
import org.forgerock.openam.sts.service.invocation.OpenAMTokenState;
import org.forgerock.openam.sts.service.invocation.OpenIdConnectTokenState;
import org.forgerock.openam.sts.service.invocation.ProofTokenState;
import org.forgerock.openam.sts.service.invocation.RestSTSServiceInvocationState;
import org.forgerock.openam.sts.service.invocation.SAML2TokenState;
import org.forgerock.openam.sts.service.invocation.UsernameTokenState;
import org.forgerock.openam.sts.service.invocation.X509TokenState;
import org.forgerock.openam.sts.token.SAML2SubjectConfirmation;
import org.forgerock.openam.utils.IOUtils;
import org.forgerock.util.encode.Base64;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStreamWriter;
import java.net.HttpURLConnection;
import java.net.MalformedURLException;
import java.net.URL;
import java.security.cert.CertificateException;
import java.security.cert.CertificateFactory;
import java.security.cert.X509Certificate;

/**
 * This class demonstrates consumption of REST STS token transformations.
 */

public class RestSTSConsumer {
    private static final boolean INSERT_X509_CERT_IN_HEADER = true;
```

```
    private final URL restSTSInstanceUrl;
    private final String stsClientCertHeaderName;
    /**
     *
     * @param restSTSInstanceTranslateUrl The full url of the REST STS instance,
     *        with the translate action specified. For example, for a REST STS
     *        instance with a deployment url of instanceId, published to the root
     *        realm, the url would be:
     *        http://amhost.com:8080/openam/rest-sts/instanceId?_action=translate
     *        A REST STS instance published to realm fred with a deployment
     *        url of instanceId2, the url would be:
     *        http://amhost.com:8080/openam/rest-sts/fred/instanceId2?_action=translate
     *        A REST STS instance published to realm bobo, a subrealm of fred
     *        with a deployment url of instanceId3, the url would be:
     *        http://amhost.com:8080/openam/rest-sts/fred/bobo/instanceId3?_action=translate
     * @param stsClientCertHeaderName The header name, configured for the REST STS instance,
     *                                where this instance expects to find the client's cert.
     *                                Necessary for x509->SAML2 token transformations
     * @throws MalformedURLException  In case the specified restSTSInstanceTranslateUrl
     *                                is mal-formed.
     */
    public RestSTSConsumer(String restSTSInstanceTranslateUrl,
                           String stsClientCertHeaderName) throws MalformedURLException {
        this.stsClientCertHeaderName = stsClientCertHeaderName;
        restSTSInstanceUrl = new URL(restSTSInstanceTranslateUrl);
        System.out.println("RestSTSConsumer consumes the REST STS at url: "
         + restSTSInstanceUrl.toString());
    }
    /**
    * Invokes a UsernameToken->SAML2 token transformation.
    *
    * Sample json posted at the REST STS instance in this method:
    * {
    *    "input_token_state": {
    *       "token_type": "USERNAME",
    *       "username": "user",
    *       "password": "userpassword"
    *    },
    *    "output_token_state": {
    *       "token_type": "SAML2",
    *       "subject_confirmation": "BEARER"
    *    }
    * }
    *
    * @param username The username in the UsernameToken
    * @param password The password in the UsernameToken
    * @param subjectConfirmation The SAML2 SubjectConfirmation.
    *                            For HoK, the certificate in the file /cert.jks
    *                            on the classpath is included.
    * @return The string representation of the SAML2 Assertion
    * @throws Exception If transformation fails
    */

    public String transformUntToSAML2(String username, String password,
                                      SAML2SubjectConfirmation subjectConfirmation)
                                      throws Exception {
        UsernameTokenState untState = UsernameTokenState.builder()
            .username(username.getBytes(AMSTSConstants.UTF_8_CHARSET_ID))
            .password(password.getBytes(AMSTSConstants.UTF_8_CHARSET_ID))
```

```
        .build();
RestSTSServiceInvocationState invocationState = RestSTSServiceInvocationState.builder()
    .inputTokenState(untState.toJson())
    .outputTokenState(buildSAML2TokenState(subjectConfirmation).toJson())
    .build();
final String response = invokeTokenTranslation(invocationState.toJson().toString(),
                                               !INSERT_X509_CERT_IN_HEADER);
return parseTokenResponse(response);
}

/**
* Invokes a OpenAMToken->SAML2 token transformation.
*
* Sample json posted at the REST STS instance in this method:
* {
*    "input_token_state": {
*       "token_type": "OPENAM",
*       "session_id": "AQI...NDI4NTUxMTQ3NDY5Ng..*"
*    },
*    "output_token_state": {
*       "token_type": "SAML2",
*       "subject_confirmation": "BEARER"
*    }
* }
*
* @param sessionId the OpenAM session ID. Corresponds to the iPlanetDirectoryPro
*                                        (or equivalent) cookie.
* @param subjectConfirmation The SAML2 SubjectConfirmation. For HoK, the certificate
*                            in the file /cert.jks on the classpath is included.
* @return The string representation of the SAML2 Assertion
* @throws Exception If transformation fails
*/

public String transformOpenAMToSAML2(String sessionId,
                                     SAML2SubjectConfirmation subjectConfirmation)
                                     throws Exception {
    OpenAMTokenState sessionState = OpenAMTokenState.builder()
        .sessionId(sessionId)
        .build();
    RestSTSServiceInvocationState invocationState = RestSTSServiceInvocationState
        .builder()
        .inputTokenState(sessionState.toJson())
        .outputTokenState(buildSAML2TokenState(subjectConfirmation).toJson())
        .build();
final String response = invokeTokenTranslation(invocationState.toJson().toString(),
                                               !INSERT_X509_CERT_IN_HEADER);
return parseTokenResponse(response);
}

/**
* Invokes a OIDCToken->SAML2 token transformation
* Sample json posted at the REST STS instance in this method (HoK SubjectConfirmation,
* with token elements truncated):
* {
*    "input_token_state": {
*       "token_type": "OPENIDCONNECT",
*       "oidc_id_token": "eyAiYWxQ.euTNnNDExNTkyMjEyIH0.kuNlKwyvZJqaC8EYpDyPJMiEcII"
*    },
*    "output_token_state": {
```

```
     *       "token_type": "SAML2",
     *       "subject_confirmation": "HOLDER_OF_KEY",
     *       "proof_token_state": {
     *        "base64EncodedCertificate": "MIMbFAAOBjQAwgYkCgYEArSQc/U75GB2AtKhbGS5pimrW0Y0Q=="
     *       }
     *    }
     * }
     * @param oidcTokenValue The OpenIdConnect ID token. Note that the targeted
     *                       REST STS instance has to be deployed with a
     *                       AuthTargetMapping which references an instance of the
     *                       OIDC module with configuration state necessary to
     *                       validate an OIDC token from a specific issuer.
     * @param subjectConfirmation The SAML2 SubjectConfirmation. For HoK, the
     *                            certificate in the file /cert.jks on the
     *                            classpath is included.
     * @return The string representation of the SAML2 Assertion
     * @throws Exception If transformation fails
     */
    public String transformOpenIdConnectToSAML2(SAML2SubjectConfirmation subjectConfirmation,
                                                String oidcTokenValue)
                                                throws Exception {
        OpenIdConnectTokenState tokenState = OpenIdConnectTokenState
            .builder()
            .tokenValue(oidcTokenValue)
            .build();
        RestSTSServiceInvocationState invocationState = RestSTSServiceInvocationState
            .builder()
            .inputTokenState(tokenState.toJson())
            .outputTokenState(buildSAML2TokenState(subjectConfirmation).toJson())
            .build();
        final String response = invokeTokenTranslation(invocationState.toJson().toString(),
                                                       !INSERT_X509_CERT_IN_HEADER);
        return parseTokenResponse(response);
    }

    /**
     * Invokes a X509->SAML2 token transformation
     *
     * Sample json posted at the REST STS instance in this method:
     * {
     *    "input_token_state": {
     *       "token_type": "X509"
     *    },
     *    "output_token_state": {
     *       "token_type": "SAML2",
     *       "subject_confirmation": "SENDER_VOUCHES"
     *    }
     * }
     *
     * Note that the targeted REST STS module has to be deployed with an
     * AuthTargetMapping that references an instance of the Certificate module
     * configured to reference the client's certificate from the header specified
     * in the AuthTargetMapping, and configured to trust the local OpenAM instance.
     * In addition, the "Certificate Field Used to Access User Profile" should be set
     * to subject CN. The CN used in the test cert deployed with OpenAM, and used
     * in this integration test, is 'test', so a subject with a uid of "test" has
     * to be created for account mapping to work. Likewise the published REST STS
     * instance must also be configured to trust the host running this test, and must
     * be configured to reference the client's certificate in the header specified
```

```
    * by stsClientCertHeaderName (unless the REST STS is being consumed via
    * two-way-TLS, in which case the stsClientCertHeaderName is irrelevant,
    * as the REST STS references the client's certificate via the
    * javax.servlet.request.X509Certificate ServletRequest attribute.
    *
    * @param subjectConfirmation The SAML2 SubjectConfirmation. For HoK, the certificate
    *                            in the file /cert.jks on the classpath is included.
    * @return The string representation of the SAML2 Assertion
    * @throws Exception If transformation fails
    */

    public String transformX509ToSAML2(SAML2SubjectConfirmation subjectConfirmation)
     throws Exception {
        X509TokenState tokenState = new X509TokenState();
        RestSTSServiceInvocationState invocationState = RestSTSServiceInvocationState
            .builder()
            .inputTokenState(tokenState.toJson())
            .outputTokenState(buildSAML2TokenState(subjectConfirmation).toJson())
            .build();
        final String response = invokeTokenTranslation(invocationState.toJson().toString(),
                                                       INSERT_X509_CERT_IN_HEADER);
        return parseTokenResponse(response);
    }
    private SAML2TokenState buildSAML2TokenState(SAML2SubjectConfirmation subjectConfirmation)
     throws Exception {
        if (SAML2SubjectConfirmation.HOLDER_OF_KEY.equals(subjectConfirmation)) {
            ProofTokenState proofTokenState = ProofTokenState
                .builder()
                .x509Certificate(getCertificate())
                .build();
            return SAML2TokenState.builder()
                .saml2SubjectConfirmation(subjectConfirmation)
                .proofTokenState(proofTokenState)
                .build();
        } else {
            return SAML2TokenState.builder()
                .saml2SubjectConfirmation(subjectConfirmation)
                .build();
        }
    }

    private String getEncodedCertificate() throws IOException, CertificateException {
        return Base64.encode(getCertificate().getEncoded());
    }

    private X509Certificate getCertificate() throws IOException, CertificateException {
        return (X509Certificate) CertificateFactory.getInstance("X.509")
                                    .generateCertificate(getClass()
                                    .getResourceAsStream("/cert.jks"));
    }

    private String invokeTokenTranslation(String invocationPayload,
                                          boolean insertX509CertInHeader)
     throws IOException, CertificateException {
        HttpURLConnection connection = (HttpURLConnection)restSTSInstanceUrl.openConnection();
        connection.setDoOutput(true);
        connection.setRequestMethod("POST");
        connection.setRequestProperty(SharedSTSConstants.CONTENT_TYPE,
                                      SharedSTSConstants.APPLICATION_JSON);
```

```
        if (insertX509CertInHeader) {
            connection.setRequestProperty(stsClientCertHeaderName, getEncodedCertificate());
        }
        OutputStreamWriter writer = new OutputStreamWriter(connection.getOutputStream());
        writer.write(invocationPayload);
        writer.close();
        if (connection.getResponseCode() == HttpURLConnection.HTTP_CREATED) {
            return getSuccessMessage(connection);
        } else {
            return getErrorMessage(connection);
        }
    }

    private String getSuccessMessage(HttpURLConnection connection) throws IOException {
        return readInputStream(connection.getInputStream());
    }

    private String getErrorMessage(HttpURLConnection connection) throws IOException {
        if (connection.getErrorStream() != null) {
            return readInputStream(connection.getErrorStream());
        } else {
            return readInputStream(connection.getInputStream());
        }
    }

    private String readInputStream(InputStream inputStream) throws IOException {
        if (inputStream == null) {
            return "Empty error stream";
        } else {
            return IOUtils.readStream(inputStream);
        }
    }

    private String parseTokenResponse(String response) throws Exception {
        Object responseContent;
        try {
            org.codehaus.jackson.JsonParser parser =
             new org.codehaus.jackson.map.ObjectMapper()
                 .getJsonFactory()
                 .createJsonParser(response);
            responseContent = parser.readValueAs(Object.class);
        } catch (IOException e) {
            throw new Exception("Could not map the response from the rest-sts instance at url "
               + restSTSInstanceUrl + " to a json object. The response: "
               + response + "; The exception: " + e);
        }
        JsonValue assertionJson =
         new JsonValue(responseContent).get(AMSTSConstants.ISSUED_TOKEN);
        if (assertionJson.isNull() || !assertionJson.isString()) {
            throw new Exception("The json response returned from the rest-sts instance at url "
               + restSTSInstanceUrl + " did not have a non-null string  element for the "
               + AMSTSConstants.ISSUED_TOKEN + " key. The json: "
               + responseContent.toString());
        }
        return assertionJson.asString();
    }
}
```

Chapter 9

Using the OpenAM Java SDK

This chapter introduces OpenAM Java SDK. OpenAM Java SDK is delivered with the full version of OpenAM, `OpenAM-12.0.0.zip`.

9.1. Installing OpenAM Client SDK Samples

The full OpenAM download, `OpenAM-12.0.0.zip`, contains the Java Client SDK library, `ClientSDK-12.0.0.jar`, as well as samples for use on the command line in `ExampleClientSDK-CLI-12.0.0.zip`, and samples in a web application, `ExampleClientSDK-WAR-12.0.0.war`. The *OpenAM Java SDK API Specification* provides a reference to the public APIs.

To Deploy the Sample Web Application

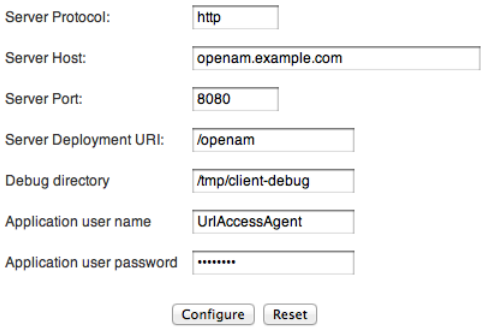
The sample web application deploys in your container to show you the client SDK samples in action.

1. Deploy the .war in your Java web application container such as Apache Tomcat or JBoss.

   ```
   $ cp ExampleClientSDK-WAR-12.0.0.war /path/to/tomcat/webapps/client.war
   ```

2. If you have run this procedure before, make sure to deploy a fresh copy of the .war file to a different location, such as `/path/to/tomcat/webapps/client1.war`

3. Browse to the location where you deployed the client, and configure the application to access OpenAM using the application user name, `UrlAccessAgent`, and password configured when you set up OpenAM.



Use the following hints to complete the configuration.

Server Protocol

Protocol to access OpenAM (`http` or `https`)

Server Host

Fully qualified domain name for OpenAM, such as `openam.example.com`

Server Port

OpenAM port number such as 8080 or 8443

Server Deployment URI

URI entry point to OpenAM such as `/openam`

Debug directory

Where to write the debug messages for the client samples

Application user name

An user agent configured to access OpenAM, such as `UrlAccessAgent` set up when OpenAM was installed

Application user password

The user agent password

The sample client writes configuration information under `$HOME/OpenAMClient/`, where $HOME is that of the user running the web application container.

4. Verify that you have properly configured the sample web application.

a. In another browser tab page of the same browser instance, login to OpenAM as the OpenAM Administrator, `amadmin`.

   This signs you into OpenAM, storing the cookie in your browser.

b. On the Samples tab page, click the link under Single Sign On Token Verification Servlet.

   If the sample web application is properly configured, you should see something like the following text in your browser.

```
SSOToken host name: 127.0.0.1
SSOToken Principal name: id=amadmin,ou=user,dc=openam,dc=forgerock,dc=org
Authentication type used: DataStore
IPAddress of the host: 127.0.0.1
SSO Token validation test succeeded
The token id is AQIC5...CMDEAAlNLABQtODY0Mjc5MDUwNDQzOTA2MzYxNg..*
...
User Attributes: {... givenName=[amAdmin], ...roles=[Top-level Admin Role], ...}
```

To Build the Command-Line Sample Applications

Follow these steps to set up the command-line examples.

1. Unpack the sample applications and related libraries.

```
$ mkdir sdk && cd sdk
$ unzip ~/Downloads/ExampleClientSDK-CLI-12.0.0.zip
```

2. Configure the samples to access OpenAM.

```
$ sh scripts/setup.sh
Debug directory (make sure this directory exists): /Users/me/openam/openam/debug
Application user (e.g. URLAccessAgent) password: secret12
Protocol of the server: http
Host name of the server: openam.example.com
Port of the server: 8080
Server's deployment URI: openam
Naming URL (hit enter to accept default value,
 http://openam.example.com:8080/openam/namingservice):
$
```

3. Verify that you have properly configured the samples.

```
$ sh scripts/Login.sh
Realm (e.g. /): /
Login module name (e.g. DataStore or LDAP): DataStore
Login locale (e.g. en_US or fr_FR): fr_FR
DataStore: Obtained login context
Nom d'utilisateur :demo
Mot de passe :changeit
Login succeeded.
Logged Out!!
```

9.2. About the OpenAM Java SDK

After installing the Java SDK command line samples, you see the following content.

- `lib/`: SDK and other libraries
- `resources/`: properties configuration files for the SDK and samples
- `scripts/`: scripts to run the samples
- `source/`: sample code

After deploying the Java SDK web application archive, you find the following content where the .war file was unpacked.

- `META-INF/`: build information
- `WEB-INF/`: sample classes and libraries
- `console/`: images for sample UI
- `index.html`: sample home page
- `keystore.jks`: OpenAM test certificate, alias: `test`, key store password: `changeit`
- `policy/`: Policy Evaluator Client Sample page
- `saml2/`: Secure Attribute Exchange example
- `sample.css`: sample styles
- `sm/`: Service Configuration sample
- `um/`: User Profile sample

Registering Your Java SDK Client to Shut Down Gracefully

When writing a client using the OpenAM Java SDK, make sure you register hooks to make sure the application can be shut down gracefully. How you register for shutdown depends on the type of application.

- For Java EE applications, make sure the OpenAM client SDK shuts down successfully by including the following context listener in your application's `web.xml` file.

```
<listener>
 <listener-class>
  com.sun.identity.common.ShutdownServletContextListener
 </listener-class>
</listener>
```

- For standalone applications, set the following JVM property.

```
-Dopenam.runtime.shutdown.hook.enabled=true
```

Chapter 10

Authenticating Using OpenAM Java SDK

This chapter looks at authentication with the OpenAM Java SDK and at the sample client, `Login.java`, which demonstrates authenticating to OpenAM from a client application, provided a realm, user name, and password. This is the sample you ran to test installation of the command-line SDK samples. The class shown in this chapter is `com.sun.identity.samples.authentication.Login`.

Before you continue, make sure that the packages described in the Using the OpenAM Java SDK chapter are installed.

With OpenAM, your client application performs the following steps to handle authentication.

1. Sets up an `AuthContext`, based on the realm in which the user authenticates.
2. Starts the login process by calling the `AuthContext login()` method.
3. Handling authentication callbacks to retrieve credentials from the user who is authenticating.

   Your application loops through the authentication callbacks by using the `AuthContext getRequirements()` and `hasMoreRequirements()` methods. Each time it finishes populating a callback with the credentials retrieved, your application calls `submitRequirements()` to send the credentials to OpenAM's Authentication Service.
4. After handling all authentication callbacks, your application calls the `AuthContext getStatus()` method.

   On login success, OpenAM sets up an `SSOToken` that holds information about the authentication, and also about the user's environment and session.
5. When the user logs out, your application can end the session by calling the `AuthContext logout()` method.

The `AuthContext` class is provided by the `com.sun.identity.authentication` package, part of the OpenAM client API. Callback classes are provided by the `javax.security.auth.callback` package, which provides callbacks for choices, confirmations, locales, names, passwords, text input, and text output.

See the *OpenAM Public API JavaDoc* for reference.

As the sample client gets the realm (called organization in the sample), locale, and authentication module to set up the authentication context, there is not need for a language callback to get the local

afterwards. The `Login.java` example does, however, show simple ways of handling callbacks for the command-line context. The implementation of the sample client follows.

```
package com.sun.identity.samples.authentication;

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.IOException;
import javax.security.auth.callback.Callback;
import javax.security.auth.callback.ChoiceCallback;
import javax.security.auth.callback.NameCallback;
import javax.security.auth.callback.PasswordCallback;
import javax.security.auth.callback.TextInputCallback;
import javax.security.auth.callback.TextOutputCallback;
import javax.security.auth.callback.UnsupportedCallbackException;
import com.sun.identity.authentication.AuthContext;
import com.sun.identity.authentication.spi.AuthLoginException;
import com.sun.identity.shared.debug.Debug;

public class Login {
    private String loginIndexName;
    private String orgName;
    private String locale;

    private Login(String loginIndexName, String orgName) {
        this.loginIndexName = loginIndexName;
        this.orgName = orgName;
    }

    private Login(String loginIndexName, String orgName, String locale) {
        this.loginIndexName = loginIndexName;
        this.orgName = orgName;
        this.locale = locale;
    }

    protected AuthContext getAuthContext()
        throws AuthLoginException {
        AuthContext lc = new AuthContext(orgName);
        AuthContext.IndexType indexType = AuthContext.IndexType.MODULE_INSTANCE;
        if (locale == null || locale.length() == 0) {
            lc.login(indexType, loginIndexName);
        } else {
            lc.login(indexType, loginIndexName, locale);
        }
        debugMessage(loginIndexName + ": Obtained login context");
        return lc;
    }

    private void addLoginCallbackMessage(Callback[] callbacks)
    throws UnsupportedCallbackException {
        int i = 0;
        try {
            for (i = 0; i < callbacks.length; i++) {
                if (callbacks[i] instanceof TextOutputCallback) {
                    handleTextOutputCallback((TextOutputCallback)callbacks[i]);
                } else if (callbacks[i] instanceof NameCallback) {
                    handleNameCallback((NameCallback)callbacks[i]);
                } else if (callbacks[i] instanceof PasswordCallback) {
                    handlePasswordCallback((PasswordCallback)callbacks[i]);
```

```
                } else if (callbacks[i] instanceof TextInputCallback) {
                    handleTextInputCallback((TextInputCallback)callbacks[i]);
                } else if (callbacks[i] instanceof ChoiceCallback) {
                    handleChoiceCallback((ChoiceCallback)callbacks[i]);
                }
            }
        } catch (IOException e) {
            e.printStackTrace();
            throw new UnsupportedCallbackException(callbacks[i],e.getMessage());
        }
    }

    private void handleTextOutputCallback(TextOutputCallback toc) {
        debugMessage("Got TextOutputCallback");
        // display the message according to the specified type

        switch (toc.getMessageType()) {
            case TextOutputCallback.INFORMATION:
                debugMessage(toc.getMessage());
                break;
            case TextOutputCallback.ERROR:
                debugMessage("ERROR: " + toc.getMessage());
                break;
            case TextOutputCallback.WARNING:
                debugMessage("WARNING: " + toc.getMessage());
                break;
            default:
                debugMessage("Unsupported message type: " +
                    toc.getMessageType());
        }
    }

    private void handleNameCallback(NameCallback nc)
        throws IOException {
        // prompt the user for a username
        System.out.print(nc.getPrompt());
        System.out.flush();
        nc.setName((new BufferedReader
            (new InputStreamReader(System.in))).readLine());
    }

    private void handleTextInputCallback(TextInputCallback tic)
        throws IOException {
        // prompt for text input
        System.out.print(tic.getPrompt());
        System.out.flush();
        tic.setText((new BufferedReader
            (new InputStreamReader(System.in))).readLine());
    }

    private void handlePasswordCallback(PasswordCallback pc)
        throws IOException {
        // prompt the user for sensitive information
        System.out.print(pc.getPrompt());
        System.out.flush();
        String passwd = (new BufferedReader(new InputStreamReader(System.in))).
            readLine();
        pc.setPassword(passwd.toCharArray());
    }
```

```
    private void handleChoiceCallback(ChoiceCallback cc)
        throws IOException {
        // ignore the provided defaultValue
        System.out.print(cc.getPrompt());

        String[] strChoices = cc.getChoices();
        for (int j = 0; j < strChoices.length; j++) {
            System.out.print("choice[" + j + "] : " + strChoices[j]);
        }
        System.out.flush();
        cc.setSelectedIndex(Integer.parseInt((new BufferedReader
            (new InputStreamReader(System.in))).readLine()));
    }

    protected boolean login(AuthContext lc)
        throws UnsupportedCallbackException {
        boolean succeed = false;
        Callback[] callbacks = null;

        // get information requested from module
        while (lc.hasMoreRequirements()) {
            callbacks = lc.getRequirements();
            if (callbacks != null) {
                addLoginCallbackMessage(callbacks);
                lc.submitRequirements(callbacks);
            }
        }

        if (lc.getStatus() == AuthContext.Status.SUCCESS) {
            System.out.println("Login succeeded.");
            succeed = true;
        } else if (lc.getStatus() == AuthContext.Status.FAILED) {
            System.out.println("Login failed.");
        } else {
            System.out.println("Unknown status: " + lc.getStatus());
        }

        return succeed;
    }

    protected void logout(AuthContext lc)
        throws AuthLoginException {
        lc.logout();
        System.out.println("Logged Out!!");
    }

    static void debugMessage(String msg) {
        System.out.println(msg);
    }

    public static void main(String[] args) {
        try {
            System.out.print("Realm (e.g. /): ");
            String orgName = (new BufferedReader(
                new InputStreamReader(System.in))).readLine();

            System.out.print("Login module name (e.g. DataStore or LDAP): ");
            String moduleName = (new BufferedReader(
```

```
                new InputStreamReader(System.in))).readLine();

            System.out.print("Login locale (e.g. en_US or fr_FR): ");
            String locale = (new BufferedReader(
                new InputStreamReader(System.in))).readLine();

            Login login = new Login(moduleName, orgName, locale);
            AuthContext lc = login.getAuthContext();
            if (login.login(lc)) {
                login.logout(lc);
            }
        } catch (IOException e) {
            e.printStackTrace();
        } catch (AuthLoginException e) {
            e.printStackTrace();
        } catch (UnsupportedCallbackException e) {
            e.printStackTrace();
        }
        System.exit(0);
    }
}
```

Chapter 11

Handling Single Sign-On Using OpenAM Java SDK

This chapter looks at handling session tokens with the OpenAM Java SDK. The class shown in this chapter is `com.sun.identity.samples.sso.SSOTokenSample`.

When a user authenticates successfully, OpenAM sets up a single sign-on (SSO) session for the user. The session is associated with an SSO token that holds information about the authentication, and also about the user's environment and session. OpenAM deletes the session when the authentication context `logout()` method is called, or when a session timeout is reached. At that point the SSO token is no longer valid.

Before you continue, make sure that the packages described in the Using the OpenAM Java SDK chapter are installed.

When your application has an `AuthContext` after successful authentication, you can retrieve the SSO token from the context. You also can get the token as shown in the sample client by passing an SSO token ID from OpenAM to an `SSOTokenManager`.

If your application needs to be notified of changes, you can register an `SSOTokenListener` on the token by using the token's `addSSOTokenListener()` method. OpenAM then calls your `SSOTokenListener ssoTokenChanged()` method when the session times out, is disposed of, or has a property that changes.

The sample client takes an SSO token ID to get the token from OpenAM, and then displays some information from the SSO token. The implementation of the sample client follows.

```
package com.sun.identity.samples.sso;

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.IOException;
import java.net.InetAddress;
import com.iplanet.sso.SSOException;
import com.iplanet.sso.SSOToken;
import com.iplanet.sso.SSOTokenID;
import com.iplanet.sso.SSOTokenManager;

public class SSOTokenSample {
    private SSOTokenManager manager;
    private SSOToken token;

    private SSOTokenSample(String tokenID)
        throws SSOException
    {
```

```
        if (validateToken(tokenID)) {
            setGetProperties(token);
        }
    }

    private boolean validateToken(String tokenID)
        throws SSOException
    {
        boolean validated = false;
        manager = SSOTokenManager.getInstance();
        token = manager.createSSOToken(tokenID);

        // isValid method returns true for valid token.
        if (manager.isValidToken(token)) {
                // let us get all the values from the token
            String host = token.getHostName();
            java.security.Principal principal = token.getPrincipal();
            String authType = token.getAuthType();
            int level = token.getAuthLevel();
            InetAddress ipAddress = token.getIPAddress();
            long maxTime = token.getMaxSessionTime();
            long idleTime = token.getIdleTime();
            long maxIdleTime = token.getMaxIdleTime();

            System.out.println("SSOToken host name: " + host);
            System.out.println("SSOToken Principal name: " +
                principal.getName());
            System.out.println("Authentication type used: " + authType);
            System.out.println("IPAddress of the host: " +
                ipAddress.getHostAddress());
            validated = true;
        }

        return validated;
    }

    private void setGetProperties(SSOToken token)
        throws SSOException
    {
        /*
         * Validate the token again, with another method
         * if token is invalid, this method throws an exception
         */
        manager.validateToken(token);
        System.out.println("SSO Token validation test Succeeded.");

        // Get the SSOTokenID associated with the token and print it.
        SSOTokenID id = token.getTokenID();
        String tokenId = id.toString();
        System.out.println("Token ID: " + tokenId);

        // Set and get properties in the token.
        token.setProperty("TimeZone", "PST");
        token.setProperty("County", "SantaClara");
        String tZone = token.getProperty("TimeZone");
        String county = token.getProperty("County");

        System.out.println("Property: TimeZone: " + tZone);
        System.out.println("Property: County: " + county);
```

```
    }

    public static void main(String[] args) {
        try {
            System.out.print("Enter SSOToken ID: ");
            String ssoTokenID = (new BufferedReader(
                new InputStreamReader(System.in))).readLine();
            new SSOTokenSample(ssoTokenID.trim());
        } catch (SSOException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
        System.exit(0);
    }

}
```

Before you run the script that calls the sample, authenticate to OpenAM in order to have OpenAM generate the SSO token ID. To see the SSO token ID, use the RESTful `authenticate` command as shown in the following example, or alternatively run the `SSOTokenSampleServlet` web-based sample.

```
$ curl \
 --request POST \
 --data "username=demo&password=changeit" \
 http://openam.example.com:8080/openam/identity/authenticate
token.id=AQIC5wM2LY4Sfcyy10grl...AlNLABQtNjI4OTkyNTUxNTc4MDQ3NzEzOQ..*
$ sh scripts/SSOTokenSample.sh
Enter SSOToken ID: AQIC5wM2LY4Sfcyy10grl...AlNLABQtNjI4OTkyNTUxNTc4MDQ3NzEzOQ..*
SSOToken host name: 172.16.203.239
SSOToken Principal name: id=demo,ou=user,dc=openam,dc=forgerock,dc=org
Authentication type used: DataStore
IPAddress of the host: 172.16.203.239
SSO Token validation test Succeeded.
Token ID: AQIC5wM2LY4Sfcyy10grl...AlNLABQtNjI4OTkyNTUxNTc4MDQ3NzEzOQ..*
Property: TimeZone: PST
Property: County: SantaClara
```

Notice both the properties populated by OpenAM, and also the two properties, `TimeZone` and `County`, that are set by the sample client.

11.1. Receiving Notifications

If your application implements a listener for change notification, such as a `SessionListener` to handle notification when a session is invalidated, then you must configure the following settings in the `AMConfig.properties` configuration file for your application.

com.iplanet.am.notification.url

Set this parameter to `http://host:port/context/notificationservice`.

com.iplanet.am.sdk.caching.enabled

Set this parameter to `true`.

com.iplanet.am.serverMode

Set this parameter to `false`.

com.sun.identity.client.notification.url

Set this parameter to `http://host:port/context/notificationservice`.

com.sun.identity.idm.cache.enabled

Set this parameter to `true`.

com.sun.identity.idm.remote.notification.enabled

Set this parameter to `true`.

com.sun.identity.sm.cache.enabled

Set this parameter to `true`.

com.sun.identity.sm.enableDataStoreNotification

Set this parameter to `true`.

The above configuration to access the notification service also applies for other types of listeners, such as `ServiceListener`, and `IdEventListener` implementations. See the *OpenAM Java SDK API Specification* for details on the available listener interfaces.

Chapter 12

Requesting Policy Decisions Using OpenAM Java SDK

This chapter shows how to request policy decision by using OpenAM Java SDK. The chapter focuses on the sample client, `source/samples/policy/PolicyEvaluationSample.java`, which demonstrates making a request to OpenAM for a policy decision about access to a web resource.

Before you continue, make sure that the packages described in the Using the OpenAM Java SDK chapter are installed.

OpenAM centralizes policy administration, policy evaluation, and policy decision making so that your applications do not have to do so. In many deployments, OpenAM policy agents and the Open Identity gateway can handle policy enforcement independently from your application code.

If your application does need to request a policy decision from OpenAM, then your application can retrieve a `PolicyEvaluator` from a client-side `PolicyEvaluatorFactory`, and then call the `PolicyEvaluator getPolicyDecision()` method. For boolean decisions such as allow or deny, your application can also call the `isAllowed()` method.

To make a policy decision, OpenAM needs an `SSOToken`, the resource to access, the action the user wants to perform on the resource such as HTTP `GET` or `POST`, and a `Map` of environment settings you can use to specify conditions and attributes in the session or can pass back as an empty `Map` if your policy does not include conditions and response attributes.

The `PolicyEvaluationSample` class takes as its configuration the user credentials, service name, resource, and action that you provide in a Java properties file. It then authenticates the user to get an `SSOToken` using the `TokenUtils.java` helper methods. At that point it has sufficient information to request a policy decision.

The implementation of the sample client follows.

```
package samples.policy;

import com.iplanet.sso.SSOToken;
import com.iplanet.sso.SSOTokenManager;

import com.sun.identity.policy.PolicyDecision;
import com.sun.identity.policy.client.PolicyEvaluator;
import com.sun.identity.policy.client.PolicyEvaluatorFactory;

import samples.policy.TokenUtils;

import java.util.Enumeration;
import java.util.HashMap;
```

```
import java.util.Map;
import java.util.HashSet;
import java.util.Properties;
import java.util.MissingResourceException;
import java.util.ResourceBundle;
import java.util.Set;

public class PolicyEvaluationSample {


    public PolicyEvaluationSample() {
    }

    public static void main(String[] args) throws Exception {
        PolicyEvaluationSample clientSample = new PolicyEvaluationSample();
        clientSample.runSample(args);
        System.exit(0);
    }

    public void runSample(String[] args) throws Exception {
        if (args.length == 0 || args.length > 1) {
            System.out.println("Missing argument:"
                    + "properties file name not specified");
        } else {
            System.out.println("Using properties file:" + args[0]);
            Properties sampleProperties = getProperties(args[0]);
            SSOToken ssoToken = getSSOToken(
                (String)sampleProperties.get("user.name"),
                (String)sampleProperties.get("user.password")
            );
            getPolicyDecision(
                ssoToken,
                (String)sampleProperties.get("service.name"),
                (String)sampleProperties.get("resource.name"),
                (String)sampleProperties.get("action.name")
            );
        }
    }

    private SSOToken getSSOToken(
            String userName, String password) throws Exception {
        System.out.println("Entering getSSOToken():"
                + "userName=" + userName + ","
                + "password=" + password);
        SSOToken ssoToken = TokenUtils.getSessionToken("/",
                userName, password);
        System.out.println("TokenID:" + ssoToken.getTokenID().toString());
        System.out.println("returning from getSSOToken()");
        return ssoToken;
    }

    private void getPolicyDecision(
            SSOToken ssoToken,
            String serviceName,
            String resourceName,
            String actionName)
            throws Exception {

        System.out.println("Entering getPolicyDecision():"
```

```
                + "resourceName=" + resourceName + ","
                + "serviceName=" + serviceName + ","
                + "actionName=" + actionName);
        PolicyEvaluator pe = PolicyEvaluatorFactory.getInstance().
                    getPolicyEvaluator(serviceName);

        Map env = new HashMap();
        Set attrSet = new HashSet();
        Set actions = new HashSet();
        actions.add(actionName);
        PolicyDecision pd = pe.getPolicyDecision(ssoToken, resourceName,
                actions, env);
        System.out.println("policyDecision:" + pd.toXML());

        System.out.println("returning from getPolicyDecision()");
    }

    private Properties getProperties(String file)
      throws MissingResourceException {
        Properties properties = new Properties();
        ResourceBundle bundle = ResourceBundle.getBundle(file);
        Enumeration e = bundle.getKeys();
        System.out.println("sample properties:");
        while (e.hasMoreElements()) {
            String key = (String) e.nextElement();
            String value = bundle.getString(key);
            properties.put(key, value);
            System.out.println(key + ":" + value);
        }
        return properties;
    }
}
```

Before you run the script that calls the sample, edit the properties file, `resources/policyEvaluationSample.properties`, to indicate the user credentials, resource to access, and HTTP method to use. You can use a resource that might not exist for the purposes of this example, but you will need to set up a policy for that resource to get meaningful results.

```
user.name=demo
user.password=changeit
service.name=iPlanetAMWebAgentService
resource.name=http://www.example.com:80/banner.html
action.name=GET
```

Also, set up a policy in OpenAM that corresponds to the resource in question. You can set up the policy in OpenAM console under Access Control > *Realm Name* > Policies. Concerning the *Realm Name*, notice that unless you change the code, the sample uses the top-level realm, `/` to authenticate the user.

With the properties configured and policy in place, get the decision from OpenAM using the script, `scripts/run-policy-evaluation-sample.sh`.

```
$ sh scripts/run-policy-evaluation-sample.sh
Using properties file:policyEvaluationSample
sample properties:
user.password:changeit
service.name:iPlanetAMWebAgentService
user.name:demo
resource.name:http://www.example.com:80/banner.html
action
.name:GET
---------------------------------------------------------------------------:
Entering getSSOToken():userName=demo,password=changeit
TokenID:AQIC5wM2LY4Sfcx3aQGFRKu5-r1a-Vfyjb...5ODM4NDY0MzE0ODYzODQ1*
returning from getSSOToken()
Entering getPolicyDecision():resourceName=http://www.example.com:80/banner.html,
 serviceName=iPlanetAMWebAgentService,actionName=GET
policyDecision:<PolicyDecision>
<ResponseAttributes>
</ResponseAttributes>
<ActionDecision timeToLive="9223372036854775807">
<AttributeValuePair>
<Attribute name="GET"/>
<Value>allow</Value>
</AttributeValuePair>
<Advices>
</Advices>
</ActionDecision>
</PolicyDecision>

returning from getPolicyDecision()
```

As you see, the policy decision response is formatted here as an XML document.[1] Notice here the line showing that OpenAM has allowed access to the resource.

```
<Value>allow</Value>
```

[1]The `PolicyDecision` element is defined in *`openam`*`/WEB-INF/remoteInterface.dtd` where *openam* is the location where the OpenAM web application is deployed.

Chapter 13

Requesting a XACML Policy Decision Using OpenAM Java SDK

This chapter shows how to request a XACML policy decision with OpenAM Java SDK, using the sample client, `source/samples/xacml/XACMLClientSample.java`. The sample client relies on an OpenAM server acting as a policy decision point and another OpenAM server acting as a policy enforcement point.

Before you continue, make sure that the packages described in the Using the OpenAM Java SDK chapter are installed.

The sample client uses the XACML `ContextFactory` to create the XACML request. It then uses the `XACMLRequestProcessor` to get a decision as XACML `Response` from OpenAM. Most of the work in the sample is done setting up the request.

The implementation of the `XACMLClientSample` class follows.

```
package samples.xacml;

import com.sun.identity.saml2.common.SAML2Exception;

import com.sun.identity.xacml.client.XACMLRequestProcessor;
import com.sun.identity.xacml.common.XACMLConstants;
import com.sun.identity.xacml.common.XACMLException;
import com.sun.identity.xacml.context.ContextFactory;
import com.sun.identity.xacml.context.Action;
import com.sun.identity.xacml.context.Attribute;
import com.sun.identity.xacml.context.Environment;
import com.sun.identity.xacml.context.Request;
import com.sun.identity.xacml.context.Resource;
import com.sun.identity.xacml.context.Response;
import com.sun.identity.xacml.context.Subject;

import java.net.URI;
import java.net.URISyntaxException;

import java.io.PrintWriter;

import java.util.ArrayList;
import java.util.Enumeration;
import java.util.List;
import java.util.MissingResourceException;
import java.util.Properties;
import java.util.ResourceBundle;

public class XACMLClientSample {
```

```
public XACMLClientSample() {
}

public static void main(String[] args) throws Exception {
    XACMLClientSample clientSample = new XACMLClientSample();
    clientSample.runSample(args);
    System.exit(0);
}

public void runSample(String[] args) throws Exception {
    if (args.length == 0 || args.length > 1) {
        System.out.println("Missing argument:"
                + "properties file name not specified");
    } else {
        System.out.println("Using properties file:" + args[0]);
        Properties sampleProperties = getProperties(args[0]);
        testProcessRequest(
            (String)sampleProperties.get("pdp.entityId"),
            (String)sampleProperties.get("pep.entityId"),
            (String)sampleProperties.get("subject.id"),
            (String)sampleProperties.get("subject.id.datatype"),
            (String)sampleProperties.get("subject.category"),
            (String)sampleProperties.get("resource.id"),
            (String)sampleProperties.get("resource.id.datatype"),
            (String)sampleProperties.get("resource.servicename"),
            (String)sampleProperties.get("resource.servicename.datatype"),
            (String)sampleProperties.get("action.id"),
            (String)sampleProperties.get("action.id.datatype")
        );
    }
}

private void testProcessRequest(
        String pdpEntityId, String pepEntityId,
        String subjectId, String subjectIdType,
        String subjectCategory,
        String resourceId, String resourceIdType,
        String serviceName, String serviceNameType,
        String actionId, String actionIdType)
        throws XACMLException, SAML2Exception,
        URISyntaxException, Exception {

    Request xacmlRequest = createSampleXacmlRequest(
        subjectId, subjectIdType,
        subjectCategory,
        resourceId, resourceIdType,
        serviceName, serviceNameType,
        actionId, actionIdType);

    System.out.println("\ntestProcessRequest():xacmlRequest:\n"
            + xacmlRequest.toXMLString(true, true));

    Response xacmlResponse = XACMLRequestProcessor.getInstance()
            .processRequest(xacmlRequest, pdpEntityId, pepEntityId);

    System.out.println("testProcessRequest():xacmlResponse:\n"
            + xacmlResponse.toXMLString(true, true));
}
```

```
    private Request createSampleXacmlRequest(
            String subjectId, String subjectIdType,
            String subjectCategory,
            String resourceId, String resourceIdType,
            String serviceName, String serviceNameType,
            String actionId, String actionIdType)
            throws XACMLException, URISyntaxException {

        Request request = ContextFactory.getInstance().createRequest();

        //Subject
        Subject subject = ContextFactory.getInstance().createSubject();
        subject.setSubjectCategory(new URI(subjectCategory));

        //set subject id
        Attribute attribute = ContextFactory.getInstance().createAttribute();
        attribute.setAttributeId(new URI(XACMLConstants.SUBJECT_ID));
        attribute.setDataType(new URI(subjectIdType));
        List valueList = new ArrayList();
        valueList.add(subjectId);
        attribute.setAttributeStringValues(valueList);
        List attributeList = new ArrayList();
        attributeList.add(attribute);
        subject.setAttributes(attributeList);


        //set Subject in Request
        List subjectList = new ArrayList();
        subjectList.add(subject);
        request.setSubjects(subjectList);

        //Resource
        Resource resource = ContextFactory.getInstance().createResource();

        //set resource id
        attribute = ContextFactory.getInstance().createAttribute();
        attribute.setAttributeId(new URI(XACMLConstants.RESOURCE_ID));
        attribute.setDataType( new URI(resourceIdType));
        valueList = new ArrayList();
        valueList.add(resourceId);
        attribute.setAttributeStringValues(valueList);
        attributeList = new ArrayList();
        attributeList.add(attribute);

        //set serviceName
        attribute = ContextFactory.getInstance().createAttribute();
        attribute.setAttributeId(new URI(XACMLConstants.TARGET_SERVICE));
        attribute.setDataType(new URI(serviceNameType));
        valueList = new ArrayList();
        valueList.add(serviceName);
        attribute.setAttributeStringValues(valueList);
        attributeList.add(attribute);
        resource.setAttributes(attributeList);

        //set Resource in Request
        List resourceList = new ArrayList();
        resourceList.add(resource);
        request.setResources(resourceList);
```

```
        //Action
        Action action = ContextFactory.getInstance().createAction();
        attribute = ContextFactory.getInstance().createAttribute();
        attribute.setAttributeId(new URI(XACMLConstants.ACTION_ID));
        attribute.setDataType(new URI(actionIdType));

        //set actionId
        valueList = new ArrayList();
        valueList.add(actionId);
        attribute.setAttributeStringValues(valueList);
        attributeList = new ArrayList();
        attributeList.add(attribute);
        action.setAttributes(attributeList);

        //set Action in Request
        request.setAction(action);

        //Environment, our PDP does not use environment now
        Environment environment = ContextFactory.getInstance()
            .createEnvironment();
        request.setEnvironment(environment);
        return request;
    }

    private Properties getProperties(String file)
        throws MissingResourceException {
        Properties properties = new Properties();
        ResourceBundle bundle = ResourceBundle.getBundle(file);
        Enumeration e = bundle.getKeys();
        System.out.println("sample properties:");
        while (e.hasMoreElements()) {
            String key = (String) e.nextElement();
            String value = bundle.getString(key);
            properties.put(key, value);
            System.out.println(key + ":" + value);
        }
        return properties;
    }
}
```

Before running the sample client, you must set up the configuration as described in the comments at the outset of the `scripts/run-xacml-client-sample.sh` script.

- Check `resources/AMConfig.properties` to see which OpenAM server the SDK is configured to use.

  The relevant settings from `resources/AMConfig.properties` specify the server protocol, host, port and deployment URI.

  ```
  com.iplanet.am.server.protocol=http
  com.iplanet.am.server.host=openam.example.com
  com.iplanet.am.server.port=8080
  com.iplanet.am.services.deploymentDescriptor=openam
  ```

  For the purpose of this example, the XACML policy decision point (PDP) and the XACML policy enforcement point (PEP) are configured on this server.

- Edit `resources/xacmlClientSample.properties` and `resources/policyEvaluationSample.properties` to set up the configuration for the sample client.

  The relevant settings from `resources/xacmlClientSample.properties` are the following.

  ```
  pdp.entityId=xacmlPdpEntity
  pep.entityId=xacmlPepEntity
  subject.id=id=demo,ou=user,dc=openam,dc=forgerock,dc=org
  subject.id.datatype=urn:oasis:names:tc:xacml:1.0:data-type:x500Name
  subject.category=urn:oasis:names:tc:xacml:1.0:subject-category:access-subject
  resource.id=http://www.example.com:80/banner.html
  resource.id.datatype=http://www.w3.org/2001/XMLSchema#string
  resource.servicename=iPlanetAMWebAgentService
  resource.servicename.datatype=http://www.w3.org/2001/XMLSchema#string
  action.id=GET
  action.id.datatype=http://www.w3.org/2001/XMLSchema#string
  ```

  The relevant settings from `resources/policyEvaluationSample.properties` are the following.

  ```
  user.name=demo
  user.password=changeit
  service.name=iPlanetAMWebAgentService
  resource.name=http://www.example.com:80/banner.html
  action.name=GET
  ```

  These settings use the default `demo` user as the subject, who has ID `id=demo,ou=user,dc=openam,dc=forgerock,dc=org`, and password `changeit`. If you choose a different subject, then change the `subject.id` value in `resources/xacmlClientSample.properties`, and the `user.name` and `user.password` values in `resources/policyEvaluationSample.properties`.

- The client accesses an OpenAM server acting as the policy enforcement point, configured in a circle of trust with the OpenAM server acting as the policy decision point. When you set up the sample clients, you pointed them to an OpenAM server. For this example, configure that server to function as a policy enforcement point and also as a policy decision point.

  1. In OpenAM console, browse to Configuration > Global > SAMLv2 SOAP Binding, and configure a new request handler with Key `/xacmlPdpEntity` and Class `com.sun.identity.xacml.plugins.XACMLAuthzDecisionQueryHandler`.

  2. Set up the circle of trust, and then create and import the metadata for the policy enforcement point and the policy decision point. In the following simplified example, both the policy enforcement point and policy decision point are hosted on the same OpenAM server. You could also set up the policy enforcement point and policy decision point on separate servers, as long as the circles of trust on both servers each include both the policy enforcement point and the policy decision point. You can set up the trust relationship between the two entities either by using the **ssoadm** command as shown below, or by using the `ssoadm.jsp` page, which you can activate as described in *OpenAM ssoadm.jsp* in the *Administration Guide*.

     ```
     $ ssoadm \
      create-cot \
      --adminid amadmin \
      --password-file /tmp/pwd.txt \
     ```

```
 --cot cot

Circle of trust, cot was created.
$ ssoadm \
 create-metadata-templ \
 --adminid amadmin \
 --password-file /tmp/pwd.txt \
 --entityid xacmlPepEntity \
 --xacmlpep /xacmlPepEntity \
 --meta-data-file xacmlPep.xml \
 --extended-data-file xacmlPep-extended.xml

Hosted entity configuration was written to xacmlPep-extended.xml.
Hosted entity descriptor was written to xacmlPep.xml.
$ ssoadm \
 import-entity \
 --adminid amadmin \
 --password-file /tmp/pwd.txt \
 --cot cot \
 --meta-data-file xacmlPep.xml \
 --extended-data-file xacmlPep-extended.xml

Import file, xacmlPep.xml.
Import file, xacmlPep-extended.xml.
$ ssoadm \
 create-metadata-templ \
 --adminid amadmin \
 --password-file /tmp/pwd.txt \
 --entityid xacmlPdpEntity \
 --xacmlpdp /xacmlPdpEntity \
 --meta-data-file xacmlPdp.xml \
 --extended-data-file xacmlPdp-extended.xml

Hosted entity configuration was written to xacmlPdp-extended.xml.
Hosted entity descriptor was written to xacmlPdp.xml.
$ ssoadm \
 import-entity \
 --adminid amadmin \
 --password-file /tmp/pwd.txt \
 --cot cot \
 --meta-data-file xacmlPdp.xml \
 --extended-data-file xacmlPdp-extended.xml

Import file, xacmlPdp.xml.
Import file, xacmlPdp-extended.xml.
```

- Create a policy that allows authenticated users to perform an HTTP `GET` on the sample `resource.id` URL you configured, such as `http://www.example.com:80/banner.html`.

  See "Defining Applications" for details.

After you have configured OpenAM and the properties files, run the sample client script, and observe the XACML request and response.

```
$ sh scripts/run-xacml-client-sample.sh
Using properties file:xacmlClientSample
```

```
sample properties:
subject.id.datatype:urn:oasis:names:tc:xacml:1.0:data-type:x500Name
pdp.entityId:xacmlPdpEntity
resource.servicename.datatype:http://www.w3.org/2001/XMLSchema#string
resource.id:http://www.example.com:80/banner.html
resource.servicename:iPlanetAMWebAgentService
action.id.datatype:http://www.w3.org/2001/XMLSchema#string
resource.id.datatype:http://www.w3.org/2001/XMLSchema#string
action.id:GET
subject.category:urn:oasis:names:tc:xacml:1.0:subject-category:access-subject
pep.entityId:xacmlPepEntity
subject.id:id=demo,ou=user,dc=openam,dc=forgerock,dc=org

testProcessRequest():xacmlRequest:

<xacml-context:Request
 xmlns:xacml-context="urn:oasis:names:tc:xacml:2.0:context:schema:os"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="urn:oasis:names:tc:xacml:2.0:context:schema:os
  http://docs.oasis-open.org/xacml/access_control-xacml-2.0-context-schema-os.xsd">
<Subject SubjectCategory=
 "urn:oasis:names:tc:xacml:1.0:subject-category:access-subject">
<Attribute
 AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
 DataType="urn:oasis:names:tc:xacml:1.0:data-type:x500Name" >
<AttributeValue
 >id=demo,ou=user,dc=openam,dc=forgerock,dc=org</AttributeValue>
</Attribute>
</Subject>
<xacml-context:Resource>
<Attribute
 AttributeId="ResourceId"
 DataType="http://www.w3.org/2001/XMLSchema#string" >
<AttributeValue>http://www.example.com:80/banner.html</AttributeValue>
</Attribute>
<Attribute
  AttributeId="urn:sun:names:xacml:2.0:resource:target-service"
  DataType="http://www.w3.org/2001/XMLSchema#string" >
<AttributeValue>iPlanetAMWebAgentService</AttributeValue>
</Attribute>
</xacml-context:Resource>
<xacml-context:Action>
<Attribute
 AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
 DataType="http://www.w3.org/2001/XMLSchema#string" >
<AttributeValue>GET</AttributeValue>
</Attribute>
</xacml-context:Action>
<xacml-context:Environment></xacml-context:Environment>
</xacml-context:Request>

testProcessRequest():xacmlResponse:
<xacml-context:Response
 xmlns:xacml-context="urn:oasis:names:tc:xacml:2.0:context:schema:os" >
<xacml-context:Result ResourceId="http://www.example.com:80/banner.html">
<xacml-context:Decision>Permit</xacml-context:Decision>
<xacml-context:Status>
<xacml-context:StatusCode
 Value="urn:oasis:names:tc:xacml:1.0:status:ok">
```

```
</xacml-context:StatusCode>
<xacml-context:StatusMessage>ok</xacml-context:StatusMessage>
<xacml-context:StatusDetail
 xmlns:xacml-context="urn:oasis:names:tc:xacml:2.0:context:schema:cd:04">
<xacml-context:StatusDetail/></xacml-context:StatusDetail>
</xacml-context:Status>
</xacml-context:Result>
</xacml-context:Response>
```

Chapter 14

Using Fedlets in Java Web Applications

This chapter introduces OpenAM Fedlets, and shows how to use the Fedlet as part of your Java web application.

14.1. Creating & Installing a Java Fedlet

An OpenAM *Fedlet* is a small web application that can do federation in your service provider application with OpenAM acting as the identity provider. The Fedlet does not require an entire OpenAM installation alongside your application, but instead can redirect to OpenAM for single sign on, and to retrieve SAML assertions.

The process for creating and installing a Java Fedlet is divided into several procedures.

- "To Create a Fedlet"
- "To Install the Fedlet as a Demo Application"
- "To Try the Fedlet Attribute Query"
- "To Try the Fedlet XACML Query"

To Create a Fedlet

The OpenAM administrator running the identity provider server creates a `Fedlet.zip` file for your service provider application, and then sends you the .zip.

1. Before creating the Fedlet, create a Hosted Identity Provider if you have not already done so, using the test certificate for the signing key.

   The single sign-on and single logout features that the Java Fedlet demonstrates do work with the Hosted Identity Provider you create starting from the Common Tasks page. The Java Fedlet Attribute Query and XACML Query tests require additional configuration, however.

   See "To Try the Fedlet Attribute Query" and "To Try the Fedlet XACML Query" for details.

2. On the Common Tasks page of the OpenAM console, click Create Fedlet.

3. Note that the Circle of Trust includes your hosted identity provider, and that Identity Provider is set to your to hosted identity provider.

4. Name the Fedlet, and also set the Destination URL.

   You can use the deployment URL, such as `http://openam.example.com:8080/fedlet` as both the name and the destination URL.

5. Click create to generate the Fedlet.zip file, such as `$HOME/openam/myfedlets/httpopenamexamplecom8080fedlet/Fedlet.zip`.

6. Provide the Fedlet to the service provider, or install it yourself to demonstrate the Fedlet's features.

To Install the Fedlet as a Demo Application

`Fedlet.zip` includes the `fedlet.war` archive corresponding to the identity provider, and a `README` file.

- The `fedlet.war` archive contains both the Fedlet as a demo web application, and also the files you use to include the Fedlet in your service provider application.
- The README file describes how to use the Fedlet.

Perform the following steps to try single sign-on and single logout tests. For the attribute query and XACML query tests, additional configuration is required.

1. Deploy the Fedlet in your web container.

```
$ unzip Fedlet.zip
$ mv fedlet.war /path/to/tomcat/webapps
```

2. Browse to the Fedlet URL, and then click the links to set up the configuration directory in `$HOME/fedlet`, where $HOME corresponds to the user running the web application container.

3. Try one or more examples from the Fedlet home page to validate Fedlet setup.

   After setting up OpenAM with the default subjects, you can login on the identity provider with user name `demo` and password `changeit`.

To Try the Fedlet Attribute Query

To try the Fedlet Attribute Query test, the Identity Provider must be configured with the Attribute Authority (`AttrAuth`) type and should sign responses. The Fedlet must be configured to deal with signed responses. Furthermore, map the attributes to request in both the Identity Provider and the Fedlet configurations.

1. Add the Attribute Authority type to the hosted Identity Provider.

   a. On OpenAM where the Identity Provider is hosted, log in to OpenAM console as `amadmin`.

   b. Under Federation > Entity Providers, select the Identity Provider, and then click New..., even though you plan to change the configuration rather than create a new provider.

   c. Select the protocol of the provider: SAMLv2.

   d. In the Create SAMLv2 Entity Provider page, do the following.

      - Set the Realm.
      - Set the Entity Identifier to the same entity identifier you selected in the previous screen.
      - In the Attribute Authority section, set the Meta Alias for example to `/attra`, and set the Signing certificate alias and Encryption certificate alias values to `test` (to use the test certificate).
      - Click Create to save your changes.

        Disregard any error messages stating that the entity already exists.

   `AttrAuth` now appears in the list of Types for your Identity Provider.

2. Under Federation > Entity Providers, click the Identity Provider link to open the provider's configuration.

3. Make sure attributes for the query are mapped on the Identity Provider.

   Under IDP > Attribute Mapper, add the following values to the Attribute Map if they are not yet present.

   - `cn=cn`
   - `sn=sn`
   - `uid=uid`

Note

Make sure to use thread-safe code if you implement the AttributeAuthorityMapper. You can use the attributes on the HttpRequest instead of synchronizing them. The default AttributeAuthorityMapper uses an attribute on the HttpServletRequest to pass information to the AttributeQueryUtil.

Click Save to save your changes.

4. Create the Fedlet as described in "To Create a Fedlet", making sure you map the attributes.

   - `cn=cn`
   - `sn=sn`
   - `uid=uid`

   This step creates a Fedlet with updated Identity Provider metadata. If you already created a Fedlet, either use a different name, or delete the existing Fedlet.

5. Deploy the new Fedlet .war as described in "To Install the Fedlet as a Demo Application".

6. Edit the new Fedlet configuration to request signing and encryption, and replace the existing configuration in OpenAM with the edited configuration.

   a. Copy the test key store from OpenAM, and prepare password files.

   ```
   $ scp user@openam:/home/user/openam/openam/keystore.jks ~/fedlet/
   ```

   The Fedlet uses password files when accessing the keystore. These password files contain encoded passwords, where the encoding is specific to the Fedlet.

   To encode the password, use `fedletEncode.jsp`. `fedletEncode.jsp` is in the deployed Fedlet, for example `http://openam.example.com:8080/fedlet/fedletEncode.jsp`. The only password to encode for OpenAM's test keystore is `changeit`, because the key store and private key passwords are both the same.

   Use the encoded value to create the password files as in the following example.

   ```
   $ echo AQIC5BHNSjLwT303GqndmHbyYvzP9Tz7OAnK > ~/fedlet/.storepass
   $ echo AQIC5BHNSjLwT303GqndmHbyYvzP9Tz7OAnK > ~/fedlet/.keypass
   ```

   b. Edit `~/fedlet/sp.xml`.

   To use the test certificate for the attribute query feature, add a `RoleDescriptor` to the `EntityDescriptor` after the `SSODescriptor`. The `RoleDescriptor` describes the certificates that are

used for signing and encryption. The attribute authority encrypts the response with the Fedlet's public key, and the Fedlet decrypts the response with its private key.

Change the following:

```
<RoleDescriptor xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:query="urn:oasis:names:tc:SAML:metadata:ext:query"
    xsi:type="query:AttributeQueryDescriptorType"
    protocolSupportEnumeration="urn:oasis:names:tc:SAML:2.0:protocol">
 </RoleDescriptor>
```

To:

```
 <RoleDescriptor xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:query="urn:oasis:names:tc:SAML:metadata:ext:query"
    xsi:type="query:AttributeQueryDescriptorType"
    protocolSupportEnumeration="urn:oasis:names:tc:SAML:2.0:protocol">
  <KeyDescriptor use="signing">
   <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
    <ds:X509Data>
     <ds:X509Certificate>
MIICQDCCAakCBEeNB0swDQYJKoZIhvcNAQEEBQAwZzELMAkGA1UEBhMCVVMxEzARBgNVBAgTCkNh
bGlmb3JuaWExFDASBgNVBAcTC1NhbnRhIENsYXJhMQwwCgYDVQQKEwNTdW4xEDAOBgNVBAsTB09w
ZW5TU08xDTALBgNVBAMTBHRlc3QwHhcNMDgwMTE1MTkxOTM5WhcNMTgwMTEyMTkxOTM5WjBnMQsw
CQYDVQQGEwJVUzETMBEGA1UECBMKQ2FsaWZvcm5pYTEUMBIGA1UEBxMLU2FudGEgQ2xhcmExDDAK
BgNVBAoTA1N1bjEQMA4GA1UECxMHT3BlblNTTzENMAsGA1UEAxMEdGVzdDCBnzANBgkqhkiG9w0B
AQEFAAOBjQAwgYkCgYEArSQc/U75GB2AtKhbGS5piiLkmJzqEsp64rDxbMJ+xDrye0EN/q1U5Of+
RkDsaN/igkAvV1cuXEgTL6RlafFPcUX7QxDhZBhsYF9pbwtMzi4A4su9hnxIhURebGEmxKW9qJNY
Js0Vo5+IgjxuEWnjnnVgHTs1+mq5QYTA7E6ZyL8CAwEAATANBgkqhkiG9w0BAQQFAAOBgQB3Pw/U
QzPKTPTYi9upbFXlrAKMwtFf2OW4yvGWWvlcwcNSZJmTJ8ARvVYOMEVNbsT4OFcfu2/PeYoAdiDA
cGy/F2Zuj8XJJpuQRSE6PtQqBuDEHjjmOQJ0rV/r8mO1ZCtHRhpZ5zYRjhRC9eCbjx9VrFax0JDC
/FfwWigmrW0Y0Q==
     </ds:X509Certificate>
    </ds:X509Data>
   </ds:KeyInfo>
  </KeyDescriptor>
  <KeyDescriptor use="encryption">
   <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
    <ds:X509Data>
     <ds:X509Certificate>
MIICQDCCAakCBEeNB0swDQYJKoZIhvcNAQEEBQAwZzELMAkGA1UEBhMCVVMxEzARBgNVBAgTCkNh
bGlmb3JuaWExFDASBgNVBAcTC1NhbnRhIENsYXJhMQwwCgYDVQQKEwNTdW4xEDAOBgNVBAsTB09w
ZW5TU08xDTALBgNVBAMTBHRlc3QwHhcNMDgwMTE1MTkxOTM5WhcNMTgwMTEyMTkxOTM5WjBnMQsw
CQYDVQQGEwJVUzETMBEGA1UECBMKQ2FsaWZvcm5pYTEUMBIGA1UEBxMLU2FudGEgQ2xhcmExDDAK
BgNVBAoTA1N1bjEQMA4GA1UECxMHT3BlblNTTzENMAsGA1UEAxMEdGVzdDCBnzANBgkqhkiG9w0B
AQEFAAOBjQAwgYkCgYEArSQc/U75GB2AtKhbGS5piiLkmJzqEsp64rDxbMJ+xDrye0EN/q1U5Of+
RkDsaN/igkAvV1cuXEgTL6RlafFPcUX7QxDhZBhsYF9pbwtMzi4A4su9hnxIhURebGEmxKW9qJNY
Js0Vo5+IgjxuEWnjnnVgHTs1+mq5QYTA7E6ZyL8CAwEAATANBgkqhkiG9w0BAQQFAAOBgQB3Pw/U
QzPKTPTYi9upbFXlrAKMwtFf2OW4yvGWWvlcwcNSZJmTJ8ARvVYOMEVNbsT4OFcfu2/PeYoAdiDA
cGy/F2Zuj8XJJpuQRSE6PtQqBuDEHjjmOQJ0rV/r8mO1ZCtHRhpZ5zYRjhRC9eCbjx9VrFax0JDC
/FfwWigmrW0Y0Q==
     </ds:X509Certificate>
    </ds:X509Data>
   </ds:KeyInfo>
   <EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#aes128-cbc">
    <xenc:KeySize xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
      >128</xenc:KeySize>
   </EncryptionMethod>
```

```
  </KeyDescriptor>
 </RoleDescriptor>
```

c. Edit `~/fedlet/sp-extended.xml` to use the test certificate for the attribute query.

Change the following, assuming your Circle of Trust is called `cot`:

```
<AttributeQueryConfig metaAlias="/attrQuery">
    <Attribute name="signingCertAlias">
        <Value></Value>
    </Attribute>
    <Attribute name="encryptionCertAlias">
        <Value></Value>
    </Attribute>
    <Attribute name="wantNameIDEncrypted">
        <Value></Value>
    </Attribute>
    <Attribute name="cotlist">
        <Value>cot</Value>
    </Attribute>
</AttributeQueryConfig>
```

To:

```
<AttributeQueryConfig metaAlias="/attrQuery">
    <Attribute name="signingCertAlias">
        <Value>test</Value>
    </Attribute>
    <Attribute name="encryptionCertAlias">
        <Value>test</Value>
    </Attribute>
    <Attribute name="wantNameIDEncrypted">
        <Value>true</Value>
    </Attribute>
    <Attribute name="cotlist">
        <Value>cot</Value>
    </Attribute>
</AttributeQueryConfig>
```

d. In OpenAM Console, under Federation > Entity Providers, delete the existing configuration for your new Fedlet.

e. Make a copy of `sp-extended.xml` called `sp-extended-copy.xml` and set `hosted="0"` in the root element of the copy.

Use the copy, `sp-extended-copy.xml`, when importing the Fedlet configuration into OpenAM. OpenAM must register the Fedlet as a *remote* service provider.

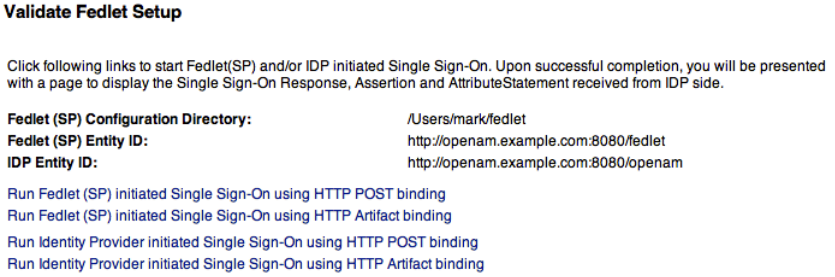
f. Under Federation > Entity Providers, click Import Entity... and import your updated Fedlet configuration.

This ensures OpenAM has the correct service provider configuration for your new Fedlet.

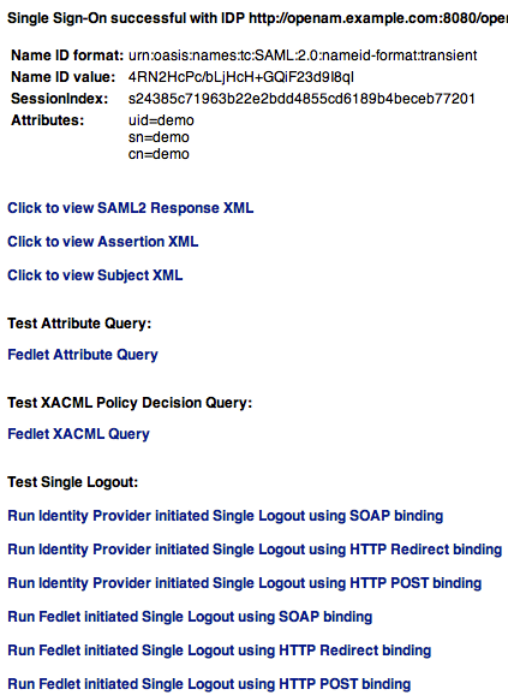
g. Restart the Fedlet or the container where it is deployed.

7. Try the Attribute Query test.

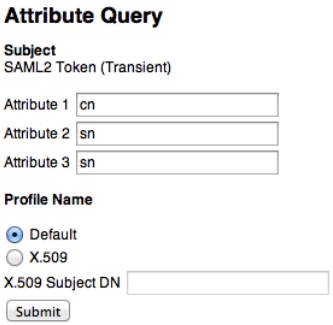
   a. Access the Fedlet.



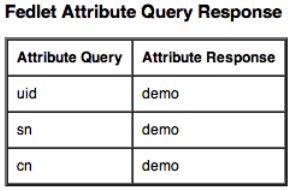
   b. Try SSO with user name `demo`, password `changeit`.



   c. Click Fedlet Attribute Query, set the attributes in the Attribute Query page to match the mapped attributes, and then click Submit.



d. Check that you see the attribute values in the response.



Fedlet Attribute Query Response

Attribute Query	Attribute Response
uid	demo
sn	demo
cn	demo

To Try the Fedlet XACML Query

To try the Fedlet XACML Query test, the Identity Provider must have a policy configured, must be configured with the Policy Decision Point (`XACML PDP`) type, and must have a SAMLv2 SOAP Binding PDP handler configured.

1. Configure a policy on the hosted Identity Provider.

   OpenAM uses the policy to make the decision whether to permit or deny access to a resource. For the purpose of the demonstration, configure a simple policy that allows all authenticated users HTTP GET access on `http://www.example.com/`.

   a. Log in to OpenAM console as `amadmin`.

   b. Access the policy editor under Access Control > *realm-name* > Policies.

   c. Choose an application that allows the resource pattern `http://www.example.com/*`, and HTTP GET as an action.

      If no application exists in the realm, add a new application for the resource pattern `http://www.example.com/*`.

   d. Add a new policy with the following characteristics.

      - Resource pattern: `http://www.example.com/*`

- Actions: allow `GET`
- Subject conditions: `Authenticated Users`

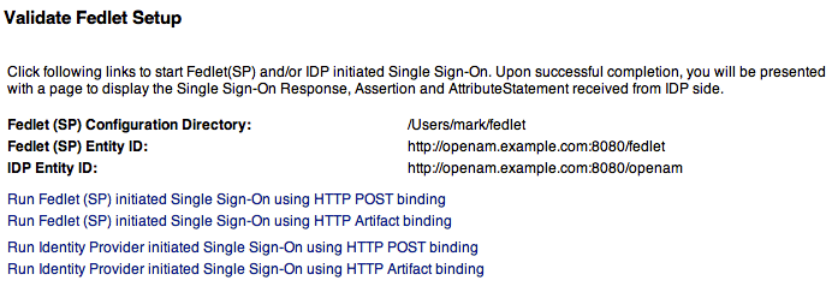
2. Add the Policy Decision Point type to the Identity Provider.
   a. Under Federation > Entity Providers, select the Identity Provider, and then click New..., even though you plan to change the configuration rather than create a new provider.
   b. Select the protocol of the provider: SAMLv2.
   c. In the Create SAMLv2 Entity Provider page, do the following.
      - Set the Realm.
      - Set the Entity Identifier to the entity identifier for the hosted Identity Provider.
      - In the XACML Policy Decision Point section, set the Meta Alias for example to `/pdp`.
      - Click Create to save your changes.

        Disregard any error messages stating that the entity already exists.

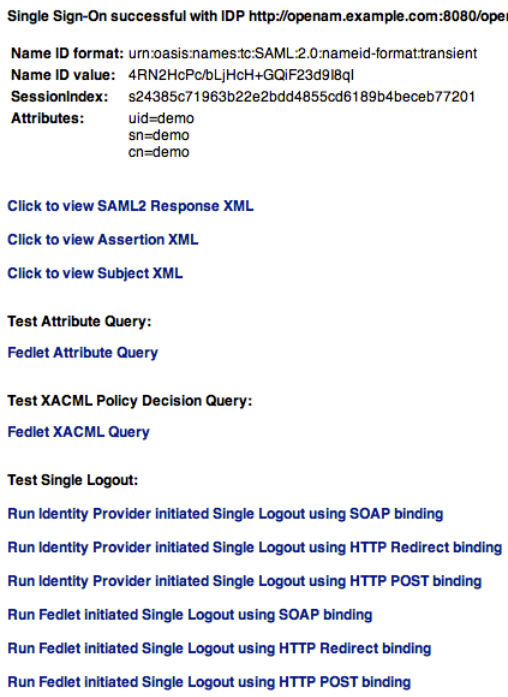
      `XACML PDP` now appears in the list of Types for your identity provider.
3. Add the PDP handler for the SAMLv2 SOAP Binding.
   a. Under Configuration > Global > SAMLv2 SOAP Binding, click New...
   b. Set the new key to match the meta alias you used when adding the XACML PDP type to the Identity Provider configuration, for example `/pdp`.
      - Key: `/pdp`
      - Class: `com.sun.identity.xacml.plugins.XACMLAuthzDecisionQueryHandler`

      Click OK. (Your changes are not saved, yet.)
   c. Click Save to actually save the new Key:Class pair.
4. Create the Fedlet as described in "To Create a Fedlet".

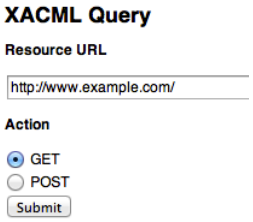
   This step creates a Fedlet with updated identity provider metadata. If you already created a Fedlet, either use a different name, or delete the existing Fedlet.
5. Deploy the new Fedlet .war as described in "To Install the Fedlet as a Demo Application".
6. Try the XACML Query test.
   a. Access the Fedlet.



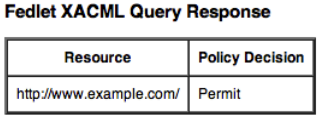
b. Try SSO with user name `demo`, password `changeit`.



c. Click XACML Attribute Query, set the Resource URL in the XACML Query page to `http://www.example.com/`, and then click Submit.



d. Check that you see the permit decision in the response.



14.2. Signing & Encryption For a Fedlet

By default when you create the Java Fedlet, signing and encryption are not configured. You can however set up OpenAM and the fedlet to sign and to verify XML signatures and to encrypt and to decrypt data such as SAML assertions. If you have tried the Attribute Query demonstration, then you have already configured the Fedlet to request signing and encryption using the test keys from the identity provider.

Enable signing and encryption for the Java Fedlet involves the following high level stages.

- Before you create the Fedlet, configure the IDP to sign and encrypt data. See Federation > Entity Providers > *IDP Name* > Signing and Encryption in the OpenAM console.

  For evaluation, you can use the `test` certificate delivered with OpenAM.

- Initially deploy and configure the Fedlet, but do not use the Fedlet until you finish.

- On the Fedlet side set up a JKS keystore used for signing and encryption. For evaluation, you can use copy the `keystore.jks` file delivered with OpenAM. You can find the file under the configuration directory for OpenAM, such as `$HOME/openam/openam/` for a server instance with base URI `openam`. The built-in keystore includes the `test` certificate.

  You must also set up `.storepass` and `.keypass` files using the `fedletEncode.jsp` page, such as `http://openam.example.com:8080/fedlet/fedletEncode.jsp`, to encode passwords on the Fedlet side. The passwords for the test key store and private key are both `changeit`.

- Configure the Fedlet to perform signing and encryption by ensuring the Fedlet has access to the key store, and by updating the SP metadata for the Fedlet.

- Import the updated SP metadata into the IDP to replace the default Fedlet configuration.

- Restart the Fedlet or container in which the Fedlet runs for the changes you made on the Fedlet side to take effect.

To Configure the Fedlet For Signing & Encryption

The `FederationConfig.properties` file specifies the paths to the JKS keystore holding the signing or encryption keys for the Fedlet, the keystore password file, and the private key password file.

1. After setting up your keystore and password files as described above, edit the properties file in the configuration directory, such as `$HOME/fedlet/FederationConfig.properties`, to point to the keystore and password files.

2. Export the certificate to use for signing and encryption purposes.

```
$ keytool -export -rfc -keystore keystore.jks -alias test
Enter keystore password:
-----BEGIN CERTIFICATE-----
MIICQDCCAakCBEeNB0swDQYJKoZIhvcNAQEEBQAwZzELMAkGA1UEBhMCVVMxEzARBgNVBAgTCkNh
bGlmb3JuaWExFDASBgNVBAcTC1NhbnRhIENsYXJhMQwwCgYDVQQKEwNTdW4xEDAOBgNVBAsTB09w
ZW5TU08xDTALBgNVBAMTBHRlc3QwHhcNMDgwMTE1MTkxOTM5WhcNMTgwMTEyMTkxOTM5WjBnMQsw
CQYDVQQGEwJVUzETMBEGA1UECBMKQ2FsaWZvcm5pYTEUMBIGA1UEBxMLU2FudGEgQ2xhcmExDDAK
BgNVBAoTA1N1bjEQMA4GA1UECxMHT3BlblNTTzENMAsGA1UEAxMEdGVzdDCBnzANBgkqhkiG9w0B
AQEFAAOBjQAwgYkCgYEArSQc/U75GB2AtKhbGS5piiLkmJzqEsp64rDxbMJ+xDrye0EN/q1U5Of+
RkDsaN/igkAvV1cuXEgTL6RlafFPcUX7QxDhZBhsYF9pbwtMzi4A4su9hnxIhURebGEmxKW9qJNY
Js0Vo5+IgjxuEWnjnnVgHTs1+mq5QYTA7E6ZyL8CAwEAATANBgkqhkiG9w0BAQQFAAOBgQB3Pw/U
QzPKTPTYi9upbFXlrAKMwtFf2OW4yvGWWvlcwcNSZJmTJ8ARvVYOMEVNbsT4OFcfu2/PeYoAdiDA
cGy/F2Zuj8XJJpuQRSE6PtQqBuDEHjjmOQJ0rV/
r8mO1ZCtHRhpZ5zYRjhRC9eCbjx9VrFax0JDC
/FfwWigmrW0Y0Q==
```

3. Edit the standard metadata file for the Fedlet, such as `$HOME/fedlet/sp.xml`, to include the certificate in KeyDescriptor elements, that are children of the SPSSODescriptor element.

```
<EntityDescriptor
 xmlns="urn:oasis:names:tc:SAML:2.0:metadata"
 entityID="http://www.example.com:8080/fedlet">
 <SPSSODescriptor
  AuthnRequestsSigned="true"
  WantAssertionsSigned="true"
  protocolSupportEnumeration="urn:oasis:names:tc:SAML:2.0:protocol">
  <KeyDescriptor use="signing">
   <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
    <ds:X509Data>
     <ds:X509Certificate>
MIICQDCCAakCBEeNB0swDQYJKoZIhvcNAQEEBQAwZzELMAkGA1UEBhMCVVMxEzARBgNVBAgTCkNh
bGlmb3JuaWExFDASBgNVBAcTC1NhbnRhIENsYXJhMQwwCgYDVQQKEwNTdW4xEDAOBgNVBAsTB09w
ZW5TU08xDTALBgNVBAMTBHRlc3QwHhcNMDgwMTE1MTkxOTM5WhcNMTgwMTEyMTkxOTM5WjBnMQsw
CQYDVQQGEwJVUzETMBEGA1UECBMKQ2FsaWZvcm5pYTEUMBIGA1UEBxMLU2FudGEgQ2xhcmExDDAK
BgNVBAoTA1N1bjEQMA4GA1UECxMHT3BlblNTTzENMAsGA1UEAxMEdGVzdDCBnzANBgkqhkiG9w0B
AQEFAAOBjQAwgYkCgYEArSQc/U75GB2AtKhbGS5piiLkmJzqEsp64rDxbMJ+xDrye0EN/q1U5Of+
RkDsaN/igkAvV1cuXEgTL6RlafFPcUX7QxDhZBhsYF9pbwtMzi4A4su9hnxIhURebGEmxKW9qJNY
Js0Vo5+IgjxuEWnjnnVgHTs1+mq5QYTA7E6ZyL8CAwEAATANBgkqhkiG9w0BAQQFAAOBgQB3Pw/U
QzPKTPTYi9upbFXlrAKMwtFf2OW4yvGWWvlcwcNSZJmTJ8ARvVYOMEVNbsT4OFcfu2/PeYoAdiDA
cGy/F2Zuj8XJJpuQRSE6PtQqBuDEHjjmOQJ0rV/r8mO1ZCtHRhpZ5zYRjhRC9eCbjx9VrFax0JDC
```

```
/FfwWigmrW0Y0Q==
     </ds:X509Certificate>
    </ds:X509Data>
   </ds:KeyInfo>
  </KeyDescriptor>
  <KeyDescriptor use="encryption">
   <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
    <ds:X509Data>
     <ds:X509Certificate>
MIICQDCCAakCBEeNB0swDQYJKoZIhvcNAQEEBQAwZzELMAkGA1UEBhMCVVMxEzARBgNVBAgTCkNh
bGlmb3JuaWExFDASBgNVBAcTC1NhbnRhIENsYXJhMQwwCgYDVQQKEwNTdW4xEDAOBgNVBAsTB09w
ZW5TU08xDTALBgNVBAMTBHRlc3QwHhcNMDgwMTE1MTkxOTM5WhcNMTgwMTEyMTkxOTM5WjBnMQsw
CQYDVQQGEwJVUzETMBEGA1UECBMKQ2FsaWZvcm5pYTEUMBIGA1UEBxMLU2FudGEgQ2xhcmExDDAK
BgNVBAoTA1N1bjEQMA4GA1UECxMHT3BlblNTTzENMAsGA1UEAxMEdGVzdDCBnzANBgkqhkiG9w0B
AQEFAAOBjQAwgYkCgYEArSQc/U75GB2AtKhbGS5piiLkmJzqEsp64rDxbMJ+xDrye0EN/q1U5Of+
RkDsaN/igkAvV1cuXEgTL6RlafFPcUX7QxDhZBhsYF9pbwtMzi4A4su9hnxIhURebGEmxKW9qJNY
Js0Vo5+IgjxuEWnjnnVgHTs1+mq5QYTA7E6ZyL8CAwEAATANBgkqhkiG9w0BAQQFAAOBgQB3Pw/U
QzPKTPTYi9upbFXlrAKMwtFf2OW4yvGWWvlcwcNSZJmTJ8ARvVYOMEVNbsT4OFcfu2/PeYoAdiDA
cGy/F2Zuj8XJJpuQRSE6PtQqBuDEHjjmOQJ0rV/r8mO1ZCtHRhpZ5zYRjhRC9eCbjx9VrFax0JDC
/FfwWigmrW0Y0Q==
     </ds:X509Certificate>
    </ds:X509Data>
   </ds:KeyInfo>
   <EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#aes128-cbc">
    <xenc:KeySize xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
     128
    </xenc:KeySize>
   </EncryptionMethod>
  </KeyDescriptor>
  <SingleLogoutService
   Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Redirect"
   Location="http://www.example.com:8080/fedlet/fedletSloRedirect"
   ResponseLocation="http://www.example.com:8080/fedlet/fedletSloRedirect" />
  <SingleLogoutService
   Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST"
   Location="http://www.example.com:8080/fedlet/fedletSloPOST"
   ResponseLocation="http://www.example.com:8080/fedlet/fedletSloPOST" />
  <SingleLogoutService
   Binding="urn:oasis:names:tc:SAML:2.0:bindings:SOAP"
   Location="http://www.example.com:8080/fedlet/fedletSloSoap" />
  <NameIDFormat>
   urn:oasis:names:tc:SAML:2.0:nameid-format:transient
  </NameIDFormat>
  <AssertionConsumerService
   index="0"
   isDefault="true"
   Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST"
   Location="http://www.example.com:8080/fedlet/fedletapplication" />
  <AssertionConsumerService
   index="1"
   Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Artifact"
   Location="http://www.example.com:8080/fedlet/fedletapplication" />
 </SPSSODescriptor>
 <RoleDescriptor
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:query="urn:oasis:names:tc:SAML:metadata:ext:query"
  xsi:type="query:AttributeQueryDescriptorType"
  protocolSupportEnumeration="urn:oasis:names:tc:SAML:2.0:protocol">
 </RoleDescriptor>
```

```
 <XACMLAuthzDecisionQueryDescriptor
  WantAssertionsSigned="false"
  protocolSupportEnumeration="urn:oasis:names:tc:SAML:2.0:protocol" />
</EntityDescriptor>
```

4. Edit the extended metadata file for the Fedlet, such as `$HOME/fedlet/sp-extended.xml`, to set the certificate alias names to the alias for the Fedlet certificate, and the `want*Signed` and `want*Encrypted` values to `true`.

   If you reformat the file, take care not to add white space around string values in elements.

```
<EntityConfig xmlns="urn:sun:fm:SAML:2.0:entityconfig"
 xmlns:fm="urn:sun:fm:SAML:2.0:entityconfig" hosted="1"
 entityID="http://www.example.com:8080/fedlet">
 <SPSSOConfig metaAlias="/sp">
  <Attribute name="description">
   <Value></Value>
  </Attribute>
  <Attribute name="signingCertAlias">
   <Value>test</Value>
  </Attribute>
  <Attribute name="encryptionCertAlias">
   <Value>test</Value>
  </Attribute>
  <Attribute name="basicAuthOn">
   <Value>false</Value>
  </Attribute>
  <Attribute name="basicAuthUser">
   <Value></Value>
  </Attribute>
  <Attribute name="basicAuthPassword">
   <Value></Value>
  </Attribute>
  <Attribute name="autofedEnabled">
   <Value>false</Value>
  </Attribute>
  <Attribute name="autofedAttribute">
   <Value></Value>
  </Attribute>
  <Attribute name="transientUser">
   <Value>anonymous</Value>
  </Attribute>
  <Attribute name="spAdapter">
   <Value></Value>
  </Attribute>
  <Attribute name="spAdapterEnv">
   <Value></Value>
  </Attribute>
  <Attribute name="fedletAdapter">
   <Value>com.sun.identity.saml2.plugins.DefaultFedletAdapter</Value>
  </Attribute>
  <Attribute name="fedletAdapterEnv">
   <Value></Value>
  </Attribute>
  <Attribute name="spAccountMapper">
   <Value>com.sun.identity.saml2.plugins.DefaultLibrarySPAccountMapper</Value>
  </Attribute>
```

```
<Attribute name="useNameIDAsSPUserID">
 <Value>false</Value>
</Attribute>
<Attribute name="spAttributeMapper">
 <Value>com.sun.identity.saml2.plugins.DefaultSPAttributeMapper</Value>
</Attribute>
<Attribute name="spAuthncontextMapper">
 <Value>com.sun.identity.saml2.plugins.DefaultSPAuthnContextMapper</Value>
</Attribute>
<Attribute name="spAuthncontextClassrefMapping">
 <Value
 >urn:oasis:names:tc:SAML:2.0:ac:classes:PasswordProtectedTransport|0|default</Value>
</Attribute>
<Attribute name="spAuthncontextComparisonType">
 <Value>exact</Value>
</Attribute>
<Attribute name="attributeMap">
 <Value>*=*</Value>
</Attribute>
<Attribute name="saml2AuthModuleName">
 <Value></Value>
</Attribute>
<Attribute name="localAuthURL">
 <Value></Value>
</Attribute>
<Attribute name="intermediateUrl">
 <Value></Value>
</Attribute>
<Attribute name="defaultRelayState">
 <Value></Value>
</Attribute>
<Attribute name="appLogoutUrl">
 <Value>http://www.example.com:8080/fedlet/logout</Value>
</Attribute>
<Attribute name="assertionTimeSkew">
 <Value>300</Value>
</Attribute>
<Attribute name="wantAttributeEncrypted">
 <Value>true</Value>
</Attribute>
<Attribute name="wantAssertionEncrypted">
 <Value>true</Value>
</Attribute>
<Attribute name="wantNameIDEncrypted">
 <Value>true</Value>
</Attribute>
<Attribute name="wantPOSTResponseSigned">
 <Value></Value>
</Attribute>
<Attribute name="wantArtifactResponseSigned">
 <Value></Value>
</Attribute>
<Attribute name="wantLogoutRequestSigned">
 <Value></Value>
</Attribute>
<Attribute name="wantLogoutResponseSigned">
 <Value></Value>
</Attribute>
<Attribute name="wantMNIRequestSigned">
```

```
   <Value></Value>
  </Attribute>
  <Attribute name="wantMNIResponseSigned">
   <Value></Value>
  </Attribute>
  <Attribute name="responseArtifactMessageEncoding">
   <Value>URI</Value>
  </Attribute>
  <Attribute name="cotlist">
   <Value>fedlet-cot</Value>
  </Attribute>
  <Attribute name="saeAppSecretList">
  </Attribute>
  <Attribute name="saeSPUrl">
   <Value></Value>
  </Attribute>
  <Attribute name="saeSPLogoutUrl">
  </Attribute>
  <Attribute name="ECPRequestIDPListFinderImpl">
   <Value>com.sun.identity.saml2.plugins.ECPIDPFinder</Value>
  </Attribute>
  <Attribute name="ECPRequestIDPList">
   <Value></Value>
  </Attribute>
  <Attribute name="ECPRequestIDPListGetComplete">
   <Value></Value>
  </Attribute>
  <Attribute name="enableIDPProxy">
   <Value>false</Value>
  </Attribute>
  <Attribute name="idpProxyList">
   <Value></Value>
  </Attribute>
  <Attribute name="idpProxyCount">
   <Value>0</Value>
  </Attribute>
  <Attribute name="useIntroductionForIDPProxy">
   <Value>false</Value>
  </Attribute>
  <Attribute name="spSessionSyncEnabled">
   <Value>false</Value>
  </Attribute>
  <Attribute name="relayStateUrlList">
  </Attribute>
 </SPSSOConfig>
 <AttributeQueryConfig metaAlias="/attrQuery">
  <Attribute name="signingCertAlias">
   <Value>test</Value>
  </Attribute>
  <Attribute name="encryptionCertAlias">
   <Value>test</Value>
  </Attribute>
  <Attribute name="wantNameIDEncrypted">
   <Value>true</Value>
  </Attribute>
  <Attribute name="cotlist">
   <Value>fedlet-cot</Value>
  </Attribute>
 </AttributeQueryConfig>
```

```
 <XACMLAuthzDecisionQueryConfig metaAlias="/pep">
  <Attribute name="signingCertAlias">
   <Value>test</Value>
  </Attribute>
  <Attribute name="encryptionCertAlias">
   <Value>test</Value>
  </Attribute>
  <Attribute name="basicAuthOn">
   <Value>false</Value>
  </Attribute>
  <Attribute name="basicAuthUser">
   <Value></Value>
  </Attribute>
  <Attribute name="basicAuthPassword">
   <Value></Value>
  </Attribute>
  <Attribute name="wantXACMLAuthzDecisionResponseSigned">
   <Value>false</Value>
  </Attribute>
  <Attribute name="wantAssertionEncrypted">
   <Value>true</Value>
  </Attribute>
  <Attribute name="cotlist">
   <Value>fedlet-cot</Value>
  </Attribute>
 </XACMLAuthzDecisionQueryConfig>
</EntityConfig>
```

5. Make a copy of `sp-extended.xml` called `sp-extended-copy.xml` and set `hosted="0"` in the root element of the copy.

   Use the copy, `sp-extended-copy.xml`, when importing the Fedlet configuration into OpenAM. OpenAM must register the Fedlet as a *remote* service provider.

6. In OpenAM console delete the original SP entity configuration for the Fedlet, and then import the updated metadata for the new configuration into OpenAM on the IDP side.

7. Restart the Fedlet or the container in which it runs in order for the Fedlet to pick up the changes to the configuration properties and the metadata.

14.3. Customizing a Java Fedlet

You can customize the Java Fedlet to perform many of the SAML 2.0 service provider operations. The Java Fedlet has the SAML 2.0 capabilities identified in the table in the *Administration Guide*, *Fedlet Support for SAML 2.0 Features* in the *Administration Guide*.

To Add Your Application

The Fedlet includes the following files that you use when building your own service provider application based on the demo web application, including a set of JavaServer Pages (JSP) examples.

`conf/`

Configuration files copied to `$HOME/fedlet` when you first deploy and configure the Fedlet. When deploying your application, you can move these to an alternate location passed to the Java virtual machine for the web application container at startup. For example, if you store the configuration under `/export/fedlet/`, then you could pass the following property to the JVM.

```
-Dcom.sun.identity.fedlet.home=/export/fedlet/conf
```

You do not need to include these files in your application.

`fedletAttrQuery.jsp`
`fedletAttrResp.jsp`

Sample SAML attribute query and response handlers.

`fedletEncode.jsp`

Utility JSP to encode a password, such as the password used to protect a Java keystore

`fedletSampleApp.jsp`
`index.jsp`

Demo application. You can remove these before deployment to replace them with your application.

`fedletXACMLQuery.jsp`
`fedletXACMLResp.jsp`

Sample SAML XACML query and response handlers.

`logout.jsp`

Utility page to perform single log out

`saml2/jsp/`

JSPs to initiate single sign on and single logout, and to handle errors, and also a JSP for obtaining Fedlet metadata, `saml2/jsp/exportmetadata.jsp`

`WEB-INF/classes/`

Localized Java properties files for strings used in the Fedlet user interface

`WEB-INF/lib/`

Fedlet libraries required by your application

`WEB-INF/web.xml`

Fedlet web application configuration, showing how JSPs map to URLs used in the Fedlet. Add mappings for your application before deployment.

In the `web.xml` mappings, your application must be mapped to `/fedletapplication`, as this is the assertion consumer URL set in the Fedlet metadata.

```
<servlet>
    <servlet-name>yourApp</servlet-name>
    <jsp-file>/fedletSampleApp.jsp</jsp-file>
</servlet>
<servlet-mapping>
    <servlet-name>yourApp</servlet-name>
    <url-pattern>/fedletapplication</url-pattern>
</servlet-mapping>
```

Follow these steps for a very simple demonstration of how to customize the Fedlet.

1. Backup `fedletSampleApp.jsp`.

```
$ cd /path/to/tomcat/webapps/fedlet/
$ cp fedletSampleApp.jsp fedletSampleApp.jsp.orig
```

2. Edit `fedletSampleApp.jsp` to reduce it to a single redirection to `myapp.jsp`. An implementation of the <html> element of the file follows below.

```
<html>
<head>
    <title>Fedlet Sample Application</title>
    <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
</head>

<body>
<%
    // BEGIN : following code is a must for Fedlet (SP) side application
    Map map;
    try {
        // invoke the Fedlet processing logic. this will do all the
        // necessary processing conforming to SAMLv2 specifications,
        // such as XML signature validation, Audience and Recipient
        // validation etc.
        map = SPACSUtils.processResponseForFedlet(request, response);
        response.sendRedirect("myapp.jsp");
    } catch (SAML2Exception sme) {
        SAMLUtils.sendError(request, response,
            response.SC_INTERNAL_SERVER_ERROR, "failedToProcessSSOResponse",
            sme.getMessage());
        return;
    } catch (IOException ioe) {
        SAMLUtils.sendError(request, response,
            response.SC_INTERNAL_SERVER_ERROR, "failedToProcessSSOResponse",
            ioe.getMessage());
        return;
    } catch (SessionException se) {
        SAMLUtils.sendError(request, response,
            response.SC_INTERNAL_SERVER_ERROR, "failedToProcessSSOResponse",
            se.getMessage());
        return;
    } catch (ServletException se) {
        SAMLUtils.sendError(request, response,
```

```
            response.SC_BAD_REQUEST, "failedToProcessSSOResponse",
            se.getMessage());
        return;
    }
    // END : code is a must for Fedlet (SP) side application
%>
</body>
</html>
```

3. Add a `myapp.jsp` page to the Fedlet, such as the following.

```
<html>
<head>
<title>My Application</title>
<meta http-equiv="Content-Type" content="text/html" />
</head>

<body>

<h1>My Application</h1>

<p>After you change the <code>fedletSampleApp.jsp</code>,
   all it does is redirect to this home page after
   successful login.</p>

</body>
</html>
```

4. Browse to the Fedlet URL, such as `http://openam.example.com:8080/fedlet/`, and try one of the login methods.

   After login you are redirected to `myapp.jsp`.

14.3.1. Performing Single Sign-On

The Java Fedlet includes a JSP file, `saml2/jsp/fedletSSOInit.jsp`, that you can call to initiate single sign-on from the Fedlet (SP) side. The Fedlet home page, `index.jsp`, calls this page when the user does Fedlet-initiated single sign-on.

When calling this JSP, the parameters to use are those also used by `saml2/jsp/spSSOInit.jsp` in OpenAM. Those parameters are described in the *Administration Guide* section, *`spSSOInit.jsp` Parameters* in the *Administration Guide*.

For IDP-initiated single sign-on, call the appropriate page on the Identity Provider. OpenAM's page is described in the *Administration Guide* section, *`idpSSOInit.jsp` Parameters* in the *Administration Guide*.

After single sign-on, the user-agent is directed by default to the assertion consumer URI set in the Fedlet metadata, which by default is `/fedletapplication`. Also by default that URI points to the JSP, `fedletSampleApp.jsp`

14.3.2. Performing Single Logout

The Java Fedlet includes a JSP file, `saml2/jsp/spSingleLogoutInit.jsp`, that you can call to initiate single logout from the Fedlet (SP) side. The Fedlet assertion consumer page, `fedletSampleApp.jsp`, calls this when the user does Fedlet-initiated single logout.

When calling this JSP, the parameters to use are those also used by `saml2/jsp/spSingleLogoutInit.jsp` in OpenAM. Those parameters are described in the *Administration Guide* section, *`spSingleLogoutInit.jsp` Parameters* in the *Administration Guide*.

For IDP-initiated single logout, call the appropriate page on the Identity Provider. OpenAM's page is described in the *Administration Guide* section, *`idpSingleLogoutInit.jsp` Parameters* in the *Administration Guide*.

Set the `RelayState` parameter when initiating logout to redirect the user-agent appropriately when the process is complete.

14.3.3. Performing Attribute Queries

As seen in the procedure, "To Try the Fedlet Attribute Query", an attribute query allows the Fedlet to get profile information about a subject from the Attribute Authority. The Fedlet must be configured to deal with responses from the Attribute Authority, including configuration for signing and encryption. Also, an Identity Provider and Attribute Authority is likely to share only those attributes that the Fedlet absolutely requires to provide service, such as, for example, a name to customize a page. The attributes must then be mapped in the Attribute Authority and Fedlet metadata.

The Java Fedlet includes a JSP file, `fedletAttrQuery.jsp`, which is used in the procedure described above to prepare an attribute query using the transient subject identifier obtained during single sign-on. The `fedletAttrQuery.jsp` also supports using the Subject name from an X.509 identity certificate.

Another JSP file, `fedletAttrResp.jsp`, sends the query to the Attribute Authority using `com.sun.identity.saml2.profile.AttributeQueryUtil.html.getAttributesForFedlet()`, and if successful processes the result, which is a `java.util.Map` of the attribute types and their values.

14.3.4. Performing XACML Queries

As seen in the procedure, "To Try the Fedlet XACML Query", a XACML query allows the Fedlet to request a policy decision from a XACML PDP. You can configure OpenAM to respond to such queries as described in that procedure.

The Java Fedlet includes a JSP file, `fedletXACMLQuery.jsp`, which is used in the procedure described above to prepare a XACML query, identifying a resource URL and a type of HTTP operation to perform, and specifying the subject identifier obtained during single sign-on.

Another JSP file, `fedletXACMLResp.jsp`, sends the query to the XACML PDP using `com.sun.identity.saml2.profile.XACMLQueryUtil.getPolicyDecisionForFedlet()`, and if successful processes the result, which is a

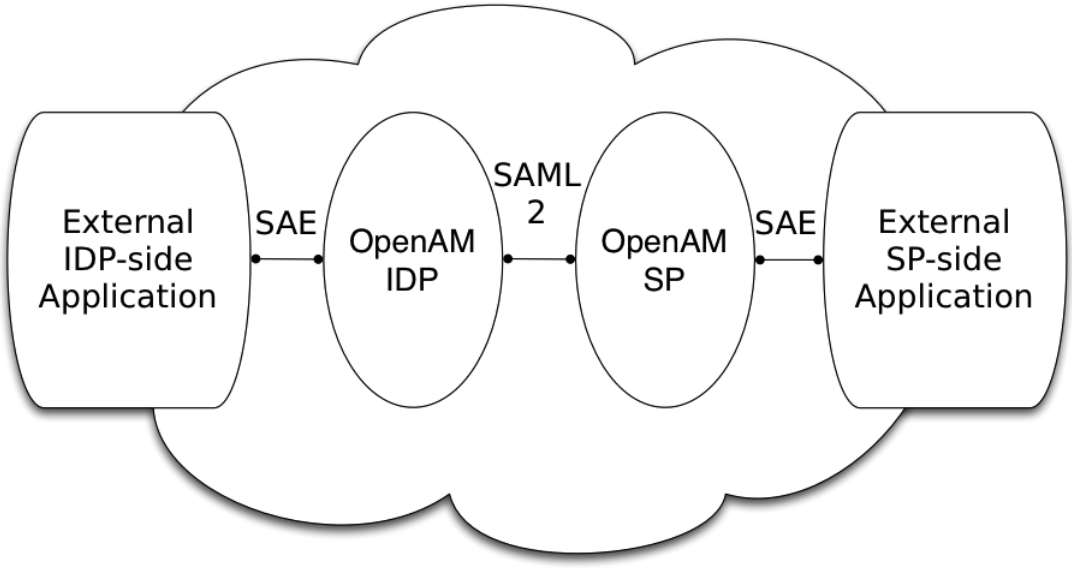
`java.lang.String` representing the decision, such as `Permit` if the decision is to allow access, or `Deny` if the decision is to deny access.

Chapter 15

Using Secure Attribute Exchange

Most deployments can rely on OpenAM to handle authentication and provide identity assertions. Not only does OpenAM support a wide variety of authentication scenarios out of the box, but OpenAM also makes it possible to add custom authentication modules. Furthermore OpenIG lets you integrate legacy systems into your access management deployment.

In a deployment where you need OpenAM to act as a SAML 2.0 gateway to a legacy application that serves as an identity provider, you can use OpenAM Secure Attribute Exchange (SAE). On the identity provider side, SAE lets OpenAM retrieve the information needed to create assertions from an external authentication service, bypassing OpenAM authentication and trusting the external service as the authoritative source of authentication. On the service provider side, SAE lets OpenAM securely provide attributes to an application that makes its own policy decision based on the attributes rather than rely on OpenAM for the policy decision.



When you use SAE on the identity provider side, an external application acts as the authoritative source of authentication. After a user authenticates successfully, the application lets OpenAM know to create a session by sending a secure HTTP GET or POST to OpenAM that asserts the identity of

the user. OpenAM processes the assertion to create a session for the user. If the user is already authenticated and comes back to access the application, the application sends a secure HTTP POST to OpenAM to assert both the user's identity and also any necessary attributes related to the user. OpenAM processes the assertion to create the session for the user and populate the attributes in the user's session. When the user logs out, the external authentication application can initiate single logout from the identity provider OpenAM server by sending the `sun.cmd=logout` attribute to OpenAM using SAE.

On the service provider side OpenAM communicates using SAML 2.0 with OpenAM on the identity provider side. OpenAM can use SAE to transmit attributes to an application through a secure HTTP POST.

SAE relies either on shared keys and symmetric encryption, or on public and private keys and asymmetric encryption to protect attributes communicated between OpenAM and external applications.

OpenAM ships with sample JSPs that demonstrate secure attribute exchange. In order to try the sample, you must set up an OpenAM circle of trust to include an identity provider and a service provider, install the SDK sample web application on each provider and then configure the providers appropriately as described in this chapter to secure communications with the sample SAE applications on both the identity provider and service provider sides.

15.1. Installing the Samples

Set up an OpenAM server as an identity provider, and another as a service provider, connecting the two in a circle of trust called `samplesaml2cot`. Configure both the hosted providers and also the remote providers as described in *Setting Up SAML 2.0 SSO* in the *Administration Guide*. This chapter assumes you set up the hosted identity provider at `http://idp.example.com:8080/openam` and the hosted service provider at `http://sp.example.com:8080/openam`. Use Common Tasks > Test Federation Connectivity in OpenAM console to make sure Federation is working before you add secure attribute exchange applications that rely on functioning SAML 2.0 communications between the providers.

Set up the sample web application as described in *Installing OpenAM Client SDK Samples*, both on the identity provider side and also on the service provider side. The SAE samples are found under `/saml2/sae` where you installed the samples. `saeIDPApp.jsp` is the identity provider side external application. `saeSPApp.jsp` is the service provider side external application.

15.2. Preparing to Secure SAE Communications

In order for SAE to be secure, you must both set up a trust relationship between the application on the identity provider side and the OpenAM server acting as identity provider, and also set up a trust relationship between the application on the service provider side and the OpenAM server acting as the service provider. These trust relationships can be based on a shared secret and symmetric encryption, or on public and private key pairs and asymmetric encryption. The trust relationships on

either side are independent. In other words you can for example use a shared secret on the identity provider side and certificates on the service provider side if you chose.

When using symmetric encryption, you must define a shared secret string used both for the application and the provider. The sample uses `secret12` as the shared secret. To simplify configuration, the sample uses the same shared secret, and thus symmetric encryption, for both trust relationships.

When using symmetric encryption, you must also use the encoded version of your shared secret. To get the encoded version of a shared secret string, use the `encode.jsp` page on the provider, as in `http://idp.example.com:8080/openam/encode.jsp` and `http://sp.example.com:8080/openam/encode.jsp`. An encoded version of `secret12` looks something like `AQICEcFhDWmb6sVmMuCJuVh43306HVacDte9`.

When using asymmetric encryption, you must obtain a public-private key pair for the application, and store the keys in a key store on the application side. Also store the public key from OpenAM which is acting as the provider in the application's key store. Make note of the certificate aliases for your application's private key, and for OpenAM's public key. Also note the path to the key store for your application, the key store password, and the private key password.

15.3. Securing the Identity Provider Side

This configuration uses the default sample settings with a shared secret of `secret12`, without encryption of the attributes.

1. Login as `amadmin` to the OpenAM server console where you set up the hosted identity provider (IDP).

2. As the sample includes a `branch` attribute not found in user profiles by default, under Access Control > *Realm Name* > Authentication > All Core Settings..., set User Profile to Ignored, and then Save your work.

3. Under Federation > Entity Providers, click the name for the Hosted IDP in order to access the IDP configuration.

   - Under Assertion Processing > Attribute Mapper, add both `mail=mail` and `branch=branch` to the attribute map, and then Save your work.

   - Under Advanced > SAE Configuration, make sure the IDP URL reflects an endpoint on the IDP such as `http://idp.example.com:8080/openam/idpsaehandler/metaAlias/idp`, and then Save your work.

   - Also under Advanced > SAE Configuration > Application Security Configuration, add the URL value for the kind of encryption you are using, and then Save your work.

     When using the defaults, the value is something like `url=http://idp.example.com:8080/samples/saml2/sae/saeIDPApp.jsp|type=symmetric|secret=`*`encoded-secret`*, where the OpenAM SDK sample web application is deployed on the IDP side with context root `/samples` and the *encoded-secret* is something like `AQICEcFhDWmb6sVmMuCJuVh43306HVacDte9`.

If you use a different mechanism to secure the communications between the SAE application and the provider, read the online help in the console to see how to construct your URL value.

4. Under Federation > Entity Providers, click the name for the Remote SP in order to access the SP configuration on the IDP side.

   - Under Assertion Processing > Attribute Mapper, add both `mail=mail` and `branch=branch` to the attribute map, and then Save your work.
   - Under Advanced > SAE Configuration, make sure the SP URL reflects an endpoint on the SP such as `http://sp.example.com:8080/openam/spsaehandler/metaAlias/sp`, and then Save your work.
   - Also under Advanced > SAE Configuration, add the URL to the sample SAE application as the SP Logout URL such as `http://sp.example.com:8080/samples/saml2/sae/saeSPApp.jsp`, and then Save your work.

15.4. Securing the Service Provider Side

This configuration uses the default sample setting of symmetric encryption, with a shared secret of `secret12`.

Login as `amadmin` to the OpenAM server console where you set up the hosted service provider (SP).

1. As the sample includes a `branch` attribute not found in user profiles by default, under Access Control > *Realm Name* > Authentication > All Core Settings..., set User Profile to Ignored, and then Save your work.

2. Under Federation > Entity Providers, click the name for the Hosted SP in order to access the SP configuration.

   - Under Assertion Processing > Attribute Mapper, add both `mail=mail` and `branch=branch` to the attribute map, and then Save your work.
   - Also under Assertion Processing > Attribute Mapper > Auto Federation, select Enabled, set the Attribute to `mail`, and then Save your work.
   - Under Advanced > SAE Configuration, make sure the SP URL reflects an endpoint on the SP such as `http://sp.example.com:8080/openam/spsaehandler/metaAlias/sp`, and then Save your work.
   - Furthermore, under Advanced > SAE Configuration, add the URL to the sample SAE application as the SP Logout URL such as `http://sp.example.com:8080/samples/saml2/sae/saeSPApp.jsp`, and then Save your work.
   - Also under Advanced > SAE Configuration > Application Security Configuration, add the URL value for the kind of encryption you are using, and then Save your work.

When using the defaults, the value is something like `url=http://sp.example.com:8080/samples/saml2/sae/saeSPApp.jsp|type=symmetric|secret=encoded-secret`, where the OpenAM SDK sample web application is deployed on the IDP side with context root `/samples` and the *encoded-secret* is something like `AQICkX24RbZboAVgr2FG1kWoqRv1zM2a6KEH`.

If you use a different mechanism to secure the communications between the SAE application and the provider, read the online help in the console to see how to construct your URL value.

15.5. Trying It Out

After completing the setup described above, navigate to the IDP side SAE application, for example at `http://idp.example.com:8080/samples/saml2/sae/saeIDPApp.jsp`.

Make sure you set at least the "SP App URL" and "SAE URL on IDP end" to fit your configuration. For example if you used the settings above then use the following values.

SP App URL

`http://sp.example.com:8080/samples/saml2/sae/saeSPApp.jsp`

SAE URL on IDP end

`http://idp.example.com:8080/openam/idpsaehandler/metaAlias/idp`

Check the settings, and then click Generate URL to open the Secure Attributes Exchange IDP APP SAMPLE page.

Click the `ssourl` link in the page to start the exchange.

The resulting web page shows the attributes exchanged, including the mail and branch values used. The text of that page is something like the following.

```
SAE SP APP SAMPLE

Secure Attrs :
mail            testuser@foo.com
sun.idpentityid http://idp.example.com:8080/openam
sun.spentityid  http://sp.example.com:8080/openam
branch          mainbranch
sun.authlevel   0
```

Chapter 16

Using the OpenAM C API

This chapter introduces OpenAM C SDK. OpenAM C SDK is available for selected platforms on the OpenAM nightly builds page. Contact info@forgerock.com if you need OpenAM C SDK support.

To prepare to install OpenAM C SDK, first download the version for your platform and unpack the archive as in the following example.

```
$ mkdir -p /path/to/openam-client
$ cd /path/to/openam-client
$ unzip ~/Downloads/common_3_0_Linux_64bit.zip
```

All C SDK deliveries are .zip files, and the filenames are self-explanatory. The `SunOS` in some of the .zip files refer to the Solaris OS.

- `common_3_0_Linux.zip`
- `common_3_0_Linux_64bit.zip`
- `common_3_0_windows.zip`
- `common_3_0_windows_64bit.zip`
- `common_3_0_SunOS_x86.zip`
- `common_3_0_SunOS_64bit.zip`
- `common_3_0_SunOS_sparc.zip`
- `common_3_0_SunOS_sparc_64bit.zip`

Once unpacked, you have several directories that include the SDK, and also sample client applications.

`bin/`

The **crypt_util** or **cryptit.exe** command for encrypting passwords

`config/`

Configuration data for the SDK

`include/`

Header files for the SDK

`lib/`

SDK and other required libraries

`samples/`

Sample code

To Build OpenAM C SDK Samples

1. Review the `samples/README.TXT` file to complete any specific instructions required for your platform. The two commands shown here confirm that the specified system is a 64-bit Linux OS. Make sure it matches the C SDK package that you have downloaded.

```
$ uname -s
Linux
$ uname -m
x86_64
```

2. Set up `OpenSSOAgentBootstrap.properties` and `OpenSSOAgentConfiguration.properties` as appropriate for your environment.

   Base your work on the template files in the `config/` directory. You can find the Password Encryption Key in the OpenAM console under Configuration > Servers and Sites > *Server Name* > Security.

3. Try one of the samples you built to test your build.

```
$ LD_LIBRARY_PATH=../lib \
 ./am_auth_test \
 -f ../config/OpenSSOAgentBootstrap.properties \
 -u demo \
 -p changeit \
 -o /
   Login  1 Succeeded!
      SSOToken = AQIC5wM2LY4SfcxZfk4EzC9Y46P9cXG9ogwf2ixnYOeZ0K0.*AAJTSQACMDE.*
      Organization = /
      Module Instance Name [0] = SAE
      Module Instance Name [1] = LDAP
      Module Instance Name [2] = WSSAuthModule
      Module Instance Name [3] = Federation
      Module Instance Name [4] = HOTP
      Module Instance Name [5] = DataStore
   Logout 1 Succeeded!
```

Chapter 17

Extending OpenAM

OpenAM services solve a wide range of access and federation management problems out of the box. Yet, OpenAM also exposes APIs and SPIs that enable you extend OpenAM services when built-in functionality does not fit your deployment.

This part of the guide covers OpenAM mechanisms for plugging in additional functionality not available out of the box.

Chapter 18

Customizing Profile Attributes

You can extend user profiles by adding custom attributes. This chapter demonstrates how to add a custom attribute to a user profile when storing user profiles in the embedded LDAP directory.

Adding a custom attribute involves both updating the `iPlanetAMUserService`, and also updating the identity repository schema to hold the new attribute. Furthermore, to allow users to update the attribute in their own profiles, you must also update the OpenAM policy configuration stored in the configuration directory.

This chapter includes the following procedures.

- "To Update the AMUser Service For the New Attribute"
- "To Update the Identity Repository For the New Attribute"
- "To Allow Users To Update the New Attribute"

To Update the AMUser Service For the New Attribute

Follow the steps below to create a custom attribute in OpenAM.

1. Create a backup copy of the configuration file for the `iPlanetAmUserService`.

   ```
   $ cp ~/openam/config/xml/amUser.xml ~/openam/config/xml/amUser.xml.orig
   ```

2. Edit the file to add your attribute as one of the list of `<User>` attributes.

   ```
   <AttributeSchema name="customAttribute"
       type="single"
       syntax="string"
       any="display"
       i18nKey="Custom Attribute">
   </AttributeSchema>
   ```

   Here, the name refers to the attribute type name used in LDAP. The `i18nKey` holds either the reference, or in this case the content, of the text that appears in the user interface.

3. Delete `iPlanetAMUserService`, and then create it from your updated configuration file.

```
$ cd /path/to/tools/openam/bin/
$ ssoadm \
 delete-svc \
 --adminid amadmin \
 --password-file /tmp/pwd.txt \
 --servicename iPlanetAMUserService

Service was deleted.
$ ssoadm \
 create-svc \
 --adminid amadmin \
 --password-file /tmp/pwd.txt \
 --xmlfile $HOME/openam/config/xml/amUser.xml

Service was added.
```

To Update the Identity Repository For the New Attribute

Follow the steps below to update the identity repository LDAP schema for the custom attribute, and then update OpenAM to use the custom attribute and object class.

If you are adding an existing attribute that is already allowed on user profile entries, you can skip this procedure.

Tip

If you are using OpenDJ as the identity repository, you can update the schema through OpenDJ Control Panel > Schema > Manage Schema, as described in the OpenDJ documentation.

1. Prepare the attribute type object class definitions in LDIF format.

```
$ cat custom-attr.ldif
dn: cn=schema
changetype: modify
add: attributeTypes
attributeTypes: ( temp-custom-attr-oid NAME 'customAttribute' EQUALITY case
 IgnoreMatch ORDERING caseIgnoreOrderingMatch SUBSTR caseIgnoreSubstrings
 Match SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 USAGE
 userApplications )
-
add: objectClasses
objectClasses: ( temp-custom-oc-oid NAME 'customObjectclass' SUP top AUXILIARY
  MAY customAttribute )
```

2. Add the schema definitions to the directory.

```
$ /path/to/opendj/bin/ldapmodify \
 --port 1389 \
 --hostname openam.example.com \
 --bindDN "cn=Directory Manager" \
 --bindPassword password \
 --filename custom-attr.ldif
Processing MODIFY request for cn=schema
MODIFY operation successful for DN cn=schema
```

3. In OpenAM console, browse to Access Control > *Realm Name* > Data Stores > *Data Store Name*.
4. Add the object class, here `customObjectclass`, to the LDAP User Object Class list.
5. Add the attribute type, here `customAttribute`, to the LDAP User Attributes list.
6. Save your work.

To Allow Users To Update the New Attribute

Follow these steps to make the new attribute editable by users. The steps imply use of the embedded configuration directory. If you use a different directory server to store the configuration, then adapt them for your tools.

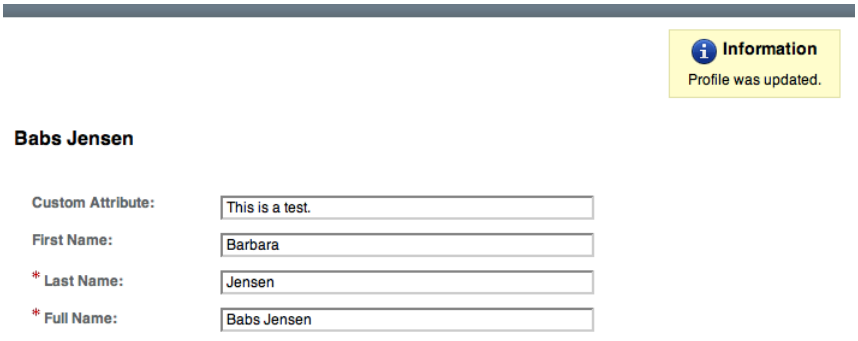
1. Login to the control panel for the embedded configuration directory.

   ```
   $ ./openam/opends/bin/control-panel &
   ```

   Connect using bind DN `cn=Directory Manager` and the the password for `amadmin`.
2. Select Manage Entries to open the LDAP browser.
3. Search with `LDAP Filter:` set to `ou=SelfWriteAttributes`, and then expand the tree views to see the two entries found.
4. In the entry under `iPlanetAMPolicyService`, edit the `sunKeyValue` attribute to add your custom attribute to the list of self-writable attributes, as in `<Value>customAttribute</Value>`.
5. In the entry under `sunEntitlementIndexes`, edit the `sunKeyValue` attribute to add your custom attribute to the list of self-writable attributes, as in replacing the last `\n` in the list with `,\n \"customAttribute \"\n`.
6. Restart OpenAM or the web container where it runs. The following example applies to Tomcat.

   ```
   $ /path/to/tomcat/bin/shutdown.sh
   $ /path/to/tomcat/bin/startup.sh
   ```

7. Login to OpenAM console as a user to check that a user can save a value for your new, custom attribute.



Chapter 19

Customizing OAuth 2.0 Scope Handling

RFC 6749, *The OAuth 2.0 Authorization Framework*, describes access token scopes as a set of case-sensitive strings defined by the authorization server. Clients can request scopes, and resource owners can authorize them.

The default scopes implementation in OpenAM treats scopes as profile attributes for the resource owner. When a resource server or other entity uses the access token to get token information from OpenAM, OpenAM populates the scopes with profile attribute values. For example, if one of the scopes is `mail`, OpenAM sets `mail` to the resource owner's email address in the token information returned.

You can change this behavior by writing your own scope validator plugin. This chapter shows how to write a custom OAuth 2.0 scope validator plugin for use in an OAuth 2.0 provider (authorization server) configuration.

19.1. Designing an OAuth 2.0 Scope Validator Plugin

A scope validator plugin implements the `org.forgerock.oauth2.core.ScopeValidator` interface. As described in the API specification, the `ScopeValidator` interface has several methods that your plugin overrides.

The following example plugin sets whether `read` and `write` permissions were granted.

```
package org.forgerock.openam.examples;

import org.forgerock.oauth2.core.AccessToken;
import org.forgerock.oauth2.core.ClientRegistration;
import org.forgerock.oauth2.core.OAuth2Request;
import org.forgerock.oauth2.core.ScopeValidator;
import org.forgerock.oauth2.core.Token;
import org.forgerock.oauth2.core.exceptions.InvalidClientException;
import org.forgerock.oauth2.core.exceptions.ServerException;
import org.forgerock.oauth2.core.exceptions.UnauthorizedClientException;

import java.util.HashMap;
import java.util.HashSet;
import java.util.Map;
import java.util.Set;

/**
 * Custom scope validators implement the
 * {@link org.forgerock.oauth2.core.ScopeValidator} interface.
```

```
 *
 * <p>
 * This example sets read and write permissions according to the scopes set.
 * </p>
 *
 * <ul>
 *
 * <li>
 * The {@code validateAuthorizationScope} method
 * adds default scopes, or any allowed scopes provided.
 * </li>
 *
 * <li>
 * The {@code validateAccessTokenScope} method
 * adds default scopes, or any allowed scopes provided.
 * </li>
 *
 * <li>
 * The {@code validateRefreshTokenScope} method
 * adds the scopes from the access token,
 * or any requested scopes provided that are also in the access token scopes.
 * </li>
 *
 * <li>
 * The {@code getUserInfo} method
 * populates scope values and sets the resource owner ID to return.
 * </li>
 *
 * <li>
 * The {@code evaluateScope} method
 * populates scope values to return.
 * </li>
 *
 * <li>
 * The {@code additionalDataToReturnFromAuthorizeEndpoint} method
 * returns no additional data (an empty Map).
 * </li>
 *
 * <li>
 * The {@code additionalDataToReturnFromTokenEndpoint} method
 * adds no additional data.
 * </li>
 *
 * </ul>
 */
public class CustomScopeValidator implements ScopeValidator {
    @Override
    public Set<String> validateAuthorizationScope(
            ClientRegistration clientRegistration,
            Set<String> scope) {
        if (scope == null || scope.isEmpty()) {
            return clientRegistration.getDefaultScopes();
        }

        Set<String> scopes = new HashSet<String>(
                clientRegistration.getAllowedScopes());
        scopes.retainAll(scope);
        return scopes;
    }
```

```
    @Override
    public Set<String> validateAccessTokenScope(
            ClientRegistration clientRegistration,
            Set<String> scope,
            OAuth2Request request) {
        if (scope == null || scope.isEmpty()) {
            return clientRegistration.getDefaultScopes();
        }

        Set<String> scopes = new HashSet<String>(
                clientRegistration.getAllowedScopes());
        scopes.retainAll(scope);
        return scopes;
    }

    @Override
    public Set<String> validateRefreshTokenScope(
            ClientRegistration clientRegistration,
            Set<String> requestedScope,
            Set<String> tokenScope,
            OAuth2Request request) {
        if (requestedScope == null || requestedScope.isEmpty()) {
            return tokenScope;
        }

        Set<String> scopes = new HashSet<String>(tokenScope);
        scopes.retainAll(requestedScope);
        return scopes;
    }

    /**
     * Set read and write permissions according to scope.
     *
     * @param token The access token presented for validation.
     * @return The map of read and write permissions,
     *         with permissions set to {@code true} or {@code false},
     *         as appropriate.
     */
    private Map<String,Object> mapScopes(AccessToken token) {
        Set<String> scopes = token.getScope();
        Map<String, Object> map = new HashMap<String, Object>();
        final String[] permissions = {"read", "write"};

        for (String scope : permissions) {
            if (scopes.contains(scope)) {
                map.put(scope, true);
            } else {
                map.put(scope, false);
            }
        }
        return map;
    }

    @Override
    public Map<String, Object> getUserInfo(
            AccessToken token,
            OAuth2Request request)
            throws UnauthorizedClientException {
```

```
        Map<String, Object> response = mapScopes(token);
        response.put("sub", token.getResourceOwnerId());
        return response;
    }

    @Override
    public Map<String, Object> evaluateScope(AccessToken token) {
        return mapScopes(token);
    }

    @Override
    public Map<String, String> additionalDataToReturnFromAuthorizeEndpoint(
            Map<String, Token> tokens,
            OAuth2Request request) {
        return new HashMap<String, String>(); // No special handling
    }

    @Override
    public void additionalDataToReturnFromTokenEndpoint(
            AccessToken token,
            OAuth2Request request)
            throws ServerException, InvalidClientException {
        // No special handling
    }
}
```

19.2. Building the OAuth 2.0 Scope Validator Sample Plugin

The sample scope validator plugin source is available online. Get a local clone so that you can try the sample on your system. In the sources you find the following files.

`pom.xml`

Apache Maven project file for the module

This file specifies how to build the sample scope validator plugin, and also specifies its dependencies on OpenAM components.

`src/main/java/org/forgerock/openam/examples/CustomScopeValidator.java`

Core class for the sample OAuth 2.0 scope validator plugin

See "Designing an OAuth 2.0 Scope Validator Plugin" for a listing.

Build the module using Apache Maven.

```
$ cd /path/to/openam-scope-sample
$ mvn install
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building openam-scope-sample 1.0.0-SNAPSHOT
[INFO] ------------------------------------------------------------------------

...

[INFO]
[INFO] --- maven-jar-plugin:2.3.2:jar (default-jar) @ openam-scope-sample ---
[INFO] Building jar: .../target/openam-scope-sample-1.0.0-SNAPSHOT.jar

...

[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 1.827s
[INFO] Finished at: Tue Jun 03 10:40:31 CEST 2014
[INFO] Final Memory: 27M/357M
[INFO] ------------------------------------------------------------------------
```

After you successfully build the module, you find the .jar in the `target/` directory of the project.

19.3. Configuring OpenAM to Use the Plugin

After building your plugin .jar file, copy the .jar file under `WEB-INF/lib/` where you deployed OpenAM.

Restart OpenAM or the container in which it runs.

In OpenAM console, you can either configure a specific OAuth 2.0 provider to use your plugin, or configure your plugin as the default for new OAuth 2.0 providers. In either case, you need the class name of your plugin. The class name for the sample plugin is `org.forgerock.openam.examples.CustomScopeValidator`.

- To configure a specific OAuth 2.0 provider to use your plugin, add the class name of your scopes plugin under Access Control > *Realm Name* > Services > *OAuth2 Provider Name* > Scope Implementation Class.
- To configure your plugin as the default for new OAuth 2.0 providers, add the class name of your scopes plugin under Configuration > Global > OAuth2 Provider > Scope Implementation Class.

19.4. Trying the Sample Plugin

In order to try the sample plugin, make sure you have configured an OAuth 2.0 provider to use the sample plugin. Also, set up an OAuth 2.0 client of the provider that takes scopes `read` and `write`.

Next try the provider as shown in the following example.

```
$ curl \
 --request POST \
 --data "grant_type=client_credentials&username=demo&password=changeit\
&client_id=myClientID&client_secret=password&scope=read" \
 https://openam.example.com:8443/openam/oauth2/access_token

{
    "scope": "read",
    "expires_in": 59,
    "token_type": "Bearer",
    "access_token": "c8860442-daba-4af0-a1d9-b607c03e5a0b"
}

$ curl https://openam.example.com:8443/openam/oauth2/tokeninfo
\
?access_token=0d492486-11a7-4175-b116-2fc1cbff6d78
{
    "scope": [
        "read"
    ],
    "grant_type": "client_credentials",
    "realm": "/",
    "write": false,
    "read": true,
    "token_type": "Bearer",
    "expires_in": 24,
    "access_token": "c8860442-daba-4af0-a1d9-b607c03e5a0b"
}
```

As seen in this example, the requested scope `read` is authorized, but the `write` scope has not been authorized.

Chapter 20

Scripted Authentication

This chapter demonstrates how to develop scripts for a scripted authentication module.

A scripted authentication module allows you to develop a custom authentication module by adding Groovy or JavaScript to the module configuration. Scripted authentication modules are an alternative to developing custom authentication modules in Java as described in *Customizing Authentication Modules*.

20.1. About Authentication Module Scripts

This section explains how client-side and server-side scripts cooperate to perform authentication.

A scripted authentication module runs scripts to authenticate a user. The configuration for the module can hold two scripts, one to include in the web page run on the client user-agent, another to run in OpenAM on the server side.

The client-side script is intended to retrieve data from the user-agent. This must be in a language the the user-agent, such as JavaScript, even if the server-side script is written in Groovy.

The client-side script data is returned to OpenAM by self-submitting form. This makes the client-side data available to the server-side script. The client-side script does this by adding data to a String object, `clientScriptOutputData`.

The server-side script is intended to handle authentication.

The server-side script runs after the client-side script has completed. The server-side script has access both to information added by the client-side script to `clientScriptOutputData`, and also to several objects from OpenAM described in "Authentication Script API", allowing it to make an authentication decision. The server-side script thus must at minimum set the authentication state to success or failure.

20.2. Authentication Script API

Client-side scripts have access only to the user-agent API. That API of course depends on the user agent.

Server-side scripts have access to objects from OpenAM for the following uses.

- "Accessing Authentication State"

- "Access Client-Side Script Output Data"
- "Accessing HTTP Services"
- "Accessing Profile Data"
- "Logging"
- "Accessing Request Data"

20.2.1. Accessing Authentication State

OpenAM passes `authState` and `sharedState` to server-side scripts in order for the scripts to access authentication state.

Server-side scripts can access the current authentication state through the `authState` object.

The `authState` value is `SUCCESS` if the authentication is currently successful, or `FAILED` if authentication has failed. Server-side scripts must set a value for `authState` before completing.

If an earlier authentication module in the authentication chain has set the login name of the user, server-side scripts can access the login name through `username`. This is shared by Anonymous, Certificate, Data Store, Federation, HTTP Basic, JDBC, LDAP, Membership, RADIUS, SecurID, Windows Desktop SSO, & Windows NT authentication modules.

20.2.2. Access Client-Side Script Output Data

Client-side scripts add data they gather into a String object named `clientScriptOutputData`. Client-side scripts then cause the user-agent automatically to return the data to OpenAM by HTTP POST of a self-submitting form.

20.2.3. Accessing HTTP Services

OpenAM passes an HTTP client object, `httpClient`, to server-side scripts. Server-side scripts can call HTTP services with the `httpClient.get` and `httpClient.post` methods. The methods return an `HttpClientResponse`.

HTTP Client Methods

Method	Parameters	Return Type	Description
`get`	*`uri`* (type: `String`), *`requestData`* (type: `Map`)	`HttpClientResponse`	Perform an HTTP GET on the specified URI with the specified request data, and return the response retrieved.
`post`	*`uri`* (type: `String`), *`body`* (type: `String`), *`requestData`* (type: `Map`)	`HttpClientResponse`	Perform an HTTP POST to the specified URI with the specified body and request data, and return the response retrieved.

The `requestData` object is a map where the keys are `cookies` and `headers`. OpenAM ignores other keys.

The `cookies` value, specifying the cookie headers in the request, is a list of Maps where the keys are `domain`, `field`, and `value`.

The `headers` value, specifying the headers in the request, is a list of Maps where the keys are `field`, and `value`.

An example `requestData` JavaScript object using GET would be as follows:

```
var response = httpClient.get("http://example.com:8080/openam/json/users" + username,
 {
   cookies:[
    {
       "domain": ".example.com",
       "field": "iPlanetDirectoryPro,
       "value": "E8cDkvlad83kdKDodkIEIxqJ938DKSDLIE31djkiefjekda.*DLEDLKJKD09dkleuhDK3kEJKj"
    }
   ],
   headers:[
    {
       "field": "Content-type",
       "value": "application/json"
    }
   ]
 });
```

An example `requestData` JavaScript object using POST would be as follows:

```
var response = httpClient.post("http://example.com:8080/openam/json/authenticate","{
  "authId": "eyAiYWxnIjogIkhTMjU2IiwgInR5cCI6ICJqd3QiIH0.eyAib3RrIjogIm03ODVzN2x
   sbnR1bjZvbGZ1MHZhOGVtYTQxIiwgInNlc3Npb25JZCI6ICJBUUlDNXdNMkxZNFNmY3lEeDY3QnB
   PdzJtRU9rUzNpLWhfNDdRWlMwNHBEN1ppdy4qQUFKVFNRQUNNREVBQWxOTEFCUXROak15TURjNU1
   UZzROVFUwTXpnNE5qRTNNQS4uKiIsICJyZWFsbSI6ICJkYz1vcGVuYW0sZGM9Zm9yZ2Vyb2NrLGR
   jPW9yZyIgfQ.VDRqaekQuXBm2lNI29hfwVADLxjepezuO0241VNDsIM",
  "template": "",
  "stage": "DataStore1",
  "callbacks": [
    {
      "type": "NameCallback",
      "output": [
        {
          "name": "prompt",
          "value": "User Name:"
        }
       ],
      "input": [
        {
          "name": "IDToken1",
          "value": "demo"
        }
       ]
    },
    {
      "type": "PasswordCallback",
```

```
      "output": [
        {
          "name": "prompt",
          "value": "Password:"
        }
       ],
      "input": [
        {
          "name": "IDToken2",
          "value": "changeit"
        }
      ]
    }
  ]
}",
{
  cookies:[
  ],
  headers:[
   {
     "field": "Content-Type",
     "value": "application/json"
   }
  ]
});
```

Note

To get the form data, you can access the `sharedState` object to get the data that previous modules in the chain have obtained. For example, if you have a DataStore module in your chain, you can get the username and password from the `sharedState` object in the script.

HTTP client requests are synchronous, blocking until they return. You can, however, set a global timeout for server-side scripts. For details, see the *Administration Guide* section, *Hints For Scripted Authentication Modules* in the *Administration Guide*.

Server-side scripts can access response data by using the methods listed in the table below.

HTTP Client Response Methods

Method	Parameters	Return Type	Description
`getCookies`	`void`	`Map<String, String>`	Get the cookies for the returned response, if any exist.
`getEntity`	`void`	`String`	Get the entity of the returned response.
`getHeaders`	`void`	`Map<String, String>`	Get the headers for the returned response, if any exist.
`getReasonPhrase`	`void`	`String`	Get the reason phrase of the returned response.
`getStatusCode`	`void`	`Integer`	Get the status code of the returned response.

Method	Parameters	Return Type	Description
`hasCookies`	`void`	`boolean`	Indicates whether the returned response had any cookies.
`hasHeaders`	`void`	`boolean`	Indicates whether the returned response had any headers.

20.2.4. Accessing Profile Data

Server-side scripts can access profile data through the methods of the `idRepository` object.

Profile Data Methods

Method	Parameters	Return Type	Description
`getAttribute`	*`userName`* (type: String), *`attributeName`* (type: String)	Set	Return the values of the named attribute for the named user.
`setAttribute`	*`userName`* (type: String), *`attributeName`* (type: String), *`attributeValue`* (type: String)	`void`	Set the named attribute to the specified value for the named user, and persist the result in the user's profile.
`addAttribute`	*`userName`* (type: String), *`attributeName`* (type: String), *`attributeValue`* (type: String)	`void`	Adds an attribute value to the list of attribute values associated with the attribute name for a particular user.

20.2.5. Logging

Server-side scripts can log messages to OpenAM debug logs by using the methods of the `logger` object.

By default, OpenAM does not log debug messages from scripts. You can configure OpenAM to log such messages by setting the debug log level for the `amScript` service. For details, see the *Administration Guide* section, *Debug Logging by Service* in the *Administration Guide*.

The following table lists the `logger` methods.

Logger Methods

Method	Parameters	Return Type	Description
`error`	`String` *`message`*	`void`	Logs *message* to OpenAM debug logs if ERROR level logging is enabled.
`errorEnabled`	`void`	`boolean`	`true` when ERROR level debug messages are enabled.
`message`	`String` *`message`*	`void`	Logs *message* to OpenAM debug logs if MESSAGE level logging is enabled.
`messageEnabled`	`void`	`boolean`	`true` when MESSAGE level debug messages are enabled.
`warning`	`String` *`message`*	`void`	Logs *message* to OpenAM debug logs if WARNING level logging is enabled.
`warningEnabled`	`void`	`boolean`	`true` when WARNING level debug messages are enabled.

20.2.6. Accessing Request Data

Server-side scripts can get access to the login request by using the methods of the `requestData` object.

The following table lists the methods of the `requestData` object. Note that this object differs from the client-side `requestData` object (see "HTTP Client Methods") and contains information about the original authentication request made by the user.

Request Data Methods

Method	Parameters	Return Type	Description
`getHeader`	*`name`* (type: String)	`String`	Return the String value of the named request header, or `null` if parameter is not set.
`getHeaders`	*`name`* (type: String)	`String[]`	Return the array of String values of the named request header, or `null` if parameter is not set.
`getParameter`	*`name`* (type: String)	`String`	Return the String value of the named request parameter, or `null` if parameter is not set.
`getParameters`	*`name`* (type: String)	`String[]`	Return the array of String values of the named request parameter, or `null` if parameter is not set.

20.3. Example JavaScript Authentication Module

This section demonstrates an authentication module with server-side JavaScript.

Before you start, make sure OpenAM is installed and configured, and create a new Scripted Module authentication module configuration in the realm / (Top Level Realm), called `JavaScript`.

To create the module configuration, browse in OpenAM Console to Access Control > / (Top Level Realm) > Authentication > Module Instances, and then click New. The name of your new module should be `JavaScript`, and the type should be `Scripted Module`.

Next, browse to the new authentication module configuration in OpenAM console under Module Instances > JavaScript.

You notice that an example server-side script already exists. Replace that script with the following program listing.

```
var START_TIME = 9;  // 9am
var END_TIME   = 17; // 5pm

logger.message("Starting server-side JavaScript");
logger.message("User: " + username);

var now = new Date();
logger.message("Current time: " + now.getHours());

if (now.getHours() < START_TIME || now.getHours() > END_TIME) {
    logger.error("Login forbidden outside work hours!");
    authState = FAILED;
} else {
    logger.message("Authentication allowed!");
    authState = SUCCESS;
}
```

This server-side script simply allows users to authenticate successfully if they login between 9 AM and 5 PM (according to the server time zone). It demonstrates logging messages to OpenAM debug logs.

If you are trying this script outside "work hours", then edit the `START_TIME` or `END_TIME`.

In addition, debug logging for server-side scripts is not enabled by default. Configure message-level debug logging for scripted authentication modules by setting the debug log level for the `amScript` service. For details, see the *Administration Guide* section, *Debug Logging by Service* in the *Administration Guide*.

At this point, the authentication module is ready for use. You can either add an example Groovy scripted authentication module as well as described in "Example Groovy Authentication Module", or use the current module alone as described in "Trying It Out".

20.4. Example Groovy Authentication Module

This section demonstrates an authentication module with a server-side Groovy script.

Create a new Scripted Module authentication module for a server-side Groovy script as described for the JavaScript module in "Example JavaScript Authentication Module". The name of your new module should be `Groovy`, and the type should be `Scripted Module`.

Next, browse to the new authentication module configuration in OpenAM console under Module Instances > Groovy.

Set the Server-Side Script Language to Groovy.

Also notice that an example server-side script already exists. Replace that script with the following program listing.

```
START_TIME = 9   // 9am
END_TIME   = 17  // 5pm

logger.message("Starting server-side Groovy script")
logger.message("User: " + username)

now = new Date()
logger.message("Current time: " + now.getHours())

if (now.getHours() < START_TIME || now.getHours() > END_TIME) {
    logger.error("Login forbidden outside work hours!")
    authState = FAILED
} else {
    logger.message("Authentication allowed!")
    authState = SUCCESS
}
```

This server-side script simply allows users to authenticate successfully if they login between 9 AM and 5 PM (according to the server time zone). It demonstrates logging messages to OpenAM debug logs.

If you are trying this script outside "work hours", then edit the `START_TIME` or `END_TIME`.

In addition, debug logging for server-side scripts is not enabled by default. If you have not already done so, configure message-level debug logging for scripted authentication modules by setting the debug log level for the `amScript` service. For details, see the *Administration Guide* section, *Debug Logging by Service* in the *Administration Guide*.

At this point, the authentication module is ready for use. You can use the module as described in "Trying It Out".

20.5. Trying It Out

You can put the example scripted authentication modules into an authentication chain, and then try them out.

First create a new authentication chain. In OpenAM console under Access Control > / (Top Level Realm) > Authentication > Authentication Chaining click New, and then name your new chain `scripted`.

After creating the chain, click Authentication Chaining > scripted to edit the configuration for the chain.

Add a DataStore module as REQUIRED to the chain first. The DataStore module checks the user credentials, whereas the scripted authentication modules do not check credentials, but instead only check that the authentication request is processed during working hours. Without the DataStore module, therefore, the `username` in the scripted authentication modules could not be determined.

After the DataStore module, add as REQUIRED either or both of the example scripted authentication modules you configured to the chain. For example, if you add both the JavaScript and Groovy modules, then your `scripted` chain would have three REQUIRED modules, first the DataStore module, and then JavaScript and Groovy modules. Make sure you save the chain configuration.

To try the new authentication chain, first log out of OpenAM. Next, browse to the login URI for the chain `/XUI/#login/&service=scripted`, where the full URL depends on your deployment, and is something like `http://openam.example.com:8080/openam/XUI/#login/&service=scripted`.

Login as the sample user having username `demo` and password `changeit`.

When you complete the login process successfully, you should see the sample user profile page. You should also see messages such as the following in the `debug/Authentication` log file.

```
amScript:07/08/2014 11:31:21:835 AM CEST: Thread[pool-19-thread-5,5,main]
Starting server-side JavaScript
amScript:07/08/2014 11:31:21:837 AM CEST: Thread[pool-19-thread-5,5,main]
User: demo
amScript:07/08/2014 11:31:21:837 AM CEST: Thread[pool-19-thread-5,5,main]
Current time: 11
amScript:07/08/2014 11:31:21:837 AM CEST: Thread[pool-19-thread-5,5,main]
Authentication allowed!
amScript:07/08/2014 11:31:22:567 AM CEST: Thread[pool-19-thread-6,5,main]
Starting server-side Groovy script
amScript:07/08/2014 11:31:22:567 AM CEST: Thread[pool-19-thread-6,5,main]
User: demo
amScript:07/08/2014 11:31:22:568 AM CEST: Thread[pool-19-thread-6,5,main]
Current time: 11
amScript:07/08/2014 11:31:22:568 AM CEST: Thread[pool-19-thread-6,5,main]
Authentication allowed!
```

If the login process fails due to errors in your scripts, then see the same log file for information about the errors.

20.6. Scripted Authentication Sandbox

The OpenAM console provides a sandboxing feature to test your Javascript or Groovy scripted authentication modules, ensuring that malicious Java classes are not directly called. The sandbox validates the script by checking that all directly-called Java classes match those in a whitelist of classes that are allowed to be invoked.

You can configure the sandbox in the console by navigating to Configuration > Authentication > Scripted Module.

The sandbox supports the following features:

- **Java Class Whitelist**. Specifies the list of class-name patterns allowed to be invoked by the script. Every class accessed by the script must match at least one of these patterns. You can specify the classes as is or use "*" wildcards. The default whitelist values are stored in the `amAuthScripted.xml` file.
- **Java Class Blacklist**. Specifies the list of class-name patterns that are NOT allowed to be invoked by the script. The blacklist is applied AFTER the whitelist to exclude those classes. You can specify the classes to exclude as is or use "*" wildcards. The default blacklist values are stored in the `amAuthScripted.xml` file.
- **Using System SecurityManager**. If enabled, the sandbox will make an additional call to `System.getSecurityManager().checkPackageAccess(...)` for each class that is accessed. The method throws a SecurityException if the calling thread is not allowed to access the package. This feature only takes effect if the security manager is enabled for the JVM.

Points about the Scripted Authentication Sandbox

Sandbox only validates directly accessible classes.

The sandbox only applies to classes that the script *directly* accesses. If the script calls `Foo.a()` and then that method calls Bar.b(), then the sandbox will allow it. You must consider the whole chain of accessible classes from each white-listed class.

> **Note**
>
> "Access" means importing/loading a class, accessing any instance of that class (for example, passed as a parameter to the script), calling a static method on that class, calling a method on an instance of that class, accessing a method or field that returns an instance of that class, etc.

All Java reflection classes are blacklisted by default.

All Java reflection classes (`java.lang.Class, java.lang.reflect.*`) are blacklisted by default to avoid bypassing the sandbox. Do not enable these reflection classes.

`java.security.AccessController` is blacklisted by default.

The `java.security.AccessController` is blacklisted by default to prevent access to the `doPrivileged()` methods.

No knowledge about inheritance.

The whitelist patterns apply only to the exact class or package names involved. The sandbox does not know anything about inheritance, so it is best to white list known, specific classes.

Chapter 21

Creating a Custom Authentication Module

This chapter shows how to customize authentication with a sample custom authentication module. For deployments with particular requirements not met by existing OpenAM authentication modules, determine whether you can adapt one of the built-in or extension modules for your needs. If not, build the functionality into a custom authentication module.

21.1. About the Sample Authentication Module

The sample authentication module prompts for a user name and password to authenticate the user, and handles error conditions. The sample shows how you integrate an authentication module into OpenAM such that you can configure the module through OpenAM console, and also localize the user interface.

For information on downloading and building OpenAM sample source code, see How do I access and build the sample code provided for OpenAM 12.x, 13.x and AM (All versions)? in the *Knowledge Base*.

Get a local clone so that you can try the sample on your system. In the sources, you find the following files under the `/path/to/openam-source/openam-samples/custom-authentication-module` directory:

`pom.xml`

Apache Maven project file for the module

This file specifies how to build the sample authentication module, and also specifies its dependencies on OpenAM components and on the Java Servlet API.

`src/main/java/org/forgerock/openam/examples/SampleAuth.java`

Core class for the sample authentication module

This class is called by OpenAM to initialize the module and to process authentication. See "The Sample Authentication Logic" for details.

`src/main/java/org/forgerock/openam/examples/SampleAuthPrincipal.java`

Class implementing `java.security.Principal` interface that defines how to map credentials to identities

This class is used to process authentication. See "The Sample Auth Principal" for details.

`src/main/resources/amAuthSampleAuth.properties`

Properties file mapping UI strings to property values

This file makes it easier to localize the UI. See "Sample Auth Properties" for details.

`src/main/resources/amAuthSampleAuth.xml`

Configuration file for the sample authentication service

This file is used when registering the authentication module with OpenAM. See "The Sample Auth Service Configuration" for details.

`src/main/resources/config/auth/default/SampleAuth.xml`

Callback file for OpenAM classic UI authentication pages

The sample authentication module does not include localized versions of this file. See "Sample Auth Callbacks" for details.

21.2. Sample Auth Properties

OpenAM uses a Java properties file per locale to retrieve the appropriate, localized strings for the authentication module.

The following is the Sample Authentication Module properties file, `amAuthSampleAuth.properties`.

```
sampleauth-service-description=Sample Authentication Module
a500=Authentication Level
a501=Service Specific Attribute

sampleauth-ui-login-header=Login
sampleauth-ui-username-prompt=User Name:
sampleauth-ui-password-prompt=Password:

sampleauth-error-1=Error 1 occurred during the authentication
sampleauth-error-2=Error 2 occurred during the authentication
```

21.3. Sample Auth Callbacks

OpenAM callbacks XML files are used to build the classic UI to prompt the user for identity information needed to process the authentication. The document type for a callback XML file is described in `WEB-INF/Auth_Module_Properties.dtd` where OpenAM is deployed.

Sample Auth Callbacks File

The following is the `SampleAuth.xml` callbacks file.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ModuleProperties PUBLIC
 "=//iPlanet//Authentication Module Properties XML Interface 1.0 DTD//EN"
        "jar://com/sun/identity/authentication/Auth_Module_Properties.dtd">

<ModuleProperties moduleName="SampleAuth" version="1.0" >
    <Callbacks length="0" order="1" timeout="600" header="#NOT SHOWN#" />
    <Callbacks length="2" order="2" timeout="600" header="#TO BE SUBSTITUTED#">
        <NameCallback isRequired="true">
            <Prompt>#USERNAME#</Prompt>
        </NameCallback>
        <PasswordCallback echoPassword="false" >
            <Prompt>#PASSWORD#</Prompt>
        </PasswordCallback>
    </Callbacks>
    <Callbacks length="1" order="3" timeout="600" header="#TO BE SUBSTITUTED#"
        error="true" >
        <NameCallback>
            <Prompt>#THE DUMMY WILL NEVER BE SHOWN#</Prompt>
        </NameCallback>
    </Callbacks>
</ModuleProperties>
```

This file specifies three states.

1. The initial state (order="1") is used dynamically to replace the dummy strings shown between hashes (for example, `#USERNAME#`) by the `substituteUIStrings()` method in `SampleAuth.java`.

2. The next state (order="2") handles prompting the user for authentication information.

3. The last state (order="3") has the attribute `error="true"`. If the authentication module state machine reaches this order then the authentication has failed. The `NameCallback` is not used and not displayed to user. OpenAM requires that the callbacks array have at least one element. Otherwise OpenAM does not permit header substitution.

21.4. The Sample Authentication Logic

An OpenAM authentication module must extend the `com.sun.identity.authentication.spi.AMLoginModule` abstract class, and must implement the methods shown below.

See the *OpenAM Java SDK API Specification* for reference.

```
// OpenAM calls the init() method once when the module is created.
public void init(Subject subject, Map sharedState, Map options)

// OpenAM calls the process() method when the user submits authentication
// information. The process() method determines what happens next:
// success, failure, or the next state specified by the order
// attribute in the callbacks XML file.
public int process(Callback[] callbacks, int state) throws LoginException

// OpenAM expects the getPrincipal() method to return an implementation of
// the java.security.Principal interface.
public Principal getPrincipal()
```

OpenAM does not reuse authentication module instances. This means that you can store information specific to the authentication process in the instance.

The implementation, `SampleAuth.java`, is shown below.

```
/**
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS HEADER.
 *
 * Copyright (c) 2011-2013 ForgeRock AS. All Rights Reserved
 *
 * The contents of this file are subject to the terms
 * of the Common Development and Distribution License
 * (the License). You may not use this file except in
 * compliance with the License.
 *
 * You can obtain a copy of the License at
 * http://forgerock.org/license/CDDLv1.0.html
 * See the License for the specific language governing
 * permission and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL
 * Header Notice in each file and include the License file
 * at http://forgerock.org/license/CDDLv1.0.html
 * If applicable, add the following below the CDDL Header,
 * with the fields enclosed by brackets [] replaced by
 * your own identifying information:
 * "Portions Copyrighted [year] [name of copyright owner]"
 *
 */

package org.forgerock.openam.examples;

import java.security.Principal;
import java.util.Map;
import java.util.ResourceBundle;

import javax.security.auth.Subject;
import javax.security.auth.callback.Callback;
import javax.security.auth.callback.NameCallback;
import javax.security.auth.callback.PasswordCallback;
import javax.security.auth.login.LoginException;

import com.sun.identity.authentication.spi.AMLoginModule;
import com.sun.identity.authentication.spi.AuthLoginException;
```

```
import com.sun.identity.authentication.spi.InvalidPasswordException;
import com.sun.identity.authentication.util.ISAuthConstants;
import com.sun.identity.shared.datastruct.CollectionHelper;
import com.sun.identity.shared.debug.Debug;



public class SampleAuth extends AMLoginModule
{

    // Name for the debug-log
    private final static String DEBUG_NAME = "SampleAuth";

    // Name of the resource bundle
    private final static String amAuthSampleAuth = "amAuthSampleAuth";

    // User names for authentication logic
    private final static String USERNAME = "demo";
    private final static String ERROR_1_NAME = "test1";
    private final static String ERROR_2_NAME = "test2";

    // Orders defined in the callbacks file
    private final static int STATE_BEGIN = 1;
    private final static int STATE_AUTH = 2;
    private final static int STATE_ERROR = 3;

    private final static Debug debug = Debug.getInstance(DEBUG_NAME);

    private Map options;
    private ResourceBundle bundle;



    public SampleAuth()
    {
        super();
    }



    @Override
    // This method stores service attributes and localized properties
    // for later use.
    public void init(Subject subject, Map sharedState, Map options)
    {
        if (debug.messageEnabled())
        {
            debug.message("SampleAuth::init");
        }
        this.options = options;
        bundle = amCache.getResBundle(amAuthSampleAuth, getLoginLocale());
    }



    @Override
    public int process(Callback[] callbacks, int state) throws LoginException
    {
```

```
        if (debug.messageEnabled())
        {
            debug.message("SampleAuth::process state: " + state);
        }

        switch (state)
        {

            case STATE_BEGIN:
                // No time wasted here - simply modify the UI and
                // proceed to next state
                substituteUIStrings();
                return STATE_AUTH;

            case STATE_AUTH:
                // Get data from callbacks. Refer to callbacks XML file.
                NameCallback nc = (NameCallback) callbacks[0];
                PasswordCallback pc = (PasswordCallback) callbacks[1];
                String username = nc.getName();
                String password = new String(pc.getPassword());

                // First errorstring is stored in "sampleauth-error-1" property.
                if (username.equals(ERROR_1_NAME))
                {
                    setErrorText("sampleauth-error-1");
                    return STATE_ERROR;
                }

                // Second errorstring is stored in "sampleauth-error-2" property.
                if (username.equals(ERROR_2_NAME))
                {
                    setErrorText("sampleauth-error-2");
                    return STATE_ERROR;
                }

                if (username.equals(USERNAME) && password.equals("changeit"))
                {
                    return ISAuthConstants.LOGIN_SUCCEED;
                }

                throw new InvalidPasswordException("password is wrong", USERNAME);

            case STATE_ERROR:
                return STATE_ERROR;
            default:
                throw new AuthLoginException("invalid state");

        }
    }



    @Override
    public Principal getPrincipal()
    {
        return new SampleAuthPrincipal(USERNAME);
    }
```

```
    private void setErrorText(String err) throws AuthLoginException
    {
        // Receive correct string from properties and substitute the
        // header in callbacks order 3.
        substituteHeader(STATE_ERROR, bundle.getString(err));
    }



    private void substituteUIStrings() throws AuthLoginException
    {
        // Get service specific attribute configured in OpenAM
        String ssa = CollectionHelper.getMapAttr(options,
                "sampleauth-service-specific-attribute");

        // Get property from bundle
        String new_hdr = ssa + " "
                + bundle.getString("sampleauth-ui-login-header");
        substituteHeader(STATE_AUTH, new_hdr);

        Callback[] cbs_phone = getCallback(STATE_AUTH);

        replaceCallback(STATE_AUTH, 0, new NameCallback(bundle
                .getString("sampleauth-ui-username-prompt")));

        replaceCallback(STATE_AUTH, 1, new PasswordCallback(bundle
                .getString("sampleauth-ui-password-prompt"), false));
    }

}
```

21.5. The Sample Auth Principal

The implementation, `SampleAuthPrincipal.java`, is shown below.

```
/**
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS HEADER.
 *
 * Copyright (c) 2011-2013 ForgeRock AS. All Rights Reserved
 *
 * The contents of this file are subject to the terms
 * of the Common Development and Distribution License
 * (the License). You may not use this file except in
 * compliance with the License.
 *
 * You can obtain a copy of the License at
 * http://forgerock.org/license/CDDLv1.0.html
 * See the License for the specific language governing
 * permission and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL
 * Header Notice in each file and include the License file
 * at http://forgerock.org/license/CDDLv1.0.html
 * If applicable, add the following below the CDDL Header,
 * with the fields enclosed by brackets [] replaced by
```

```
 * your own identifying information:
 * "Portions Copyrighted [year] [name of copyright owner]"
 *
 */

package org.forgerock.openam.examples;



import java.io.Serializable;
import java.security.Principal;



public class SampleAuthPrincipal implements Principal, Serializable
{
    private final String name;
    private final static String CLASSNAME = "SampleAuthPrincipal";
    private final static String COLON = " : ";



    public SampleAuthPrincipal(String name)
    {
        if (name == null)
        {
            throw new NullPointerException("illegal null input");
        }

        this.name = name;
    }



    /**
     * Return the LDAP username for this <code> SampleAuthPrincipal </code>.
     *
     * @return the LDAP username for this <code> SampleAuthPrincipal </code>
     */
    @Override
    public String getName()
    {
        return name;
    }



    /**
     * Return a string representation of this <code> SampleAuthPrincipal </code>.
     *
     * @return a string representation of this
     *         <code>TestAuthModulePrincipal</code>.
     */
    @Override
    public String toString()
    {
        return new StringBuilder().append(CLASSNAME).append(COLON)
                .append(name).toString();
    }
```

```
    /**
     * Compares the specified Object with this <code> SampleAuthPrincipal </code>
     * for equality. Returns true if the given object is also a
     * <code> SampleAuthPrincipal </code> and the two SampleAuthPrincipal have
     * the same username.
     *
     * @param o Object to be compared for equality with this
     *          <code> SampleAuthPrincipal </code>.
     * @return true if the specified Object is equal equal to this
     *         <code> SampleAuthPrincipal </code>.
     */
    @Override
    public boolean equals(Object o)
    {
        if (o == null)
        {
            return false;
        }

        if (this == o)
        {
            return true;
        }

        if (!(o instanceof SampleAuthPrincipal))
        {
            return false;
        }
        SampleAuthPrincipal that = (SampleAuthPrincipal) o;

        if (this.getName().equals(that.getName()))
        {
            return true;
        }
        return false;
    }



    /**
     * Return a hash code for this <code> SampleAuthPrincipal </code>.
     *
     * @return a hash code for this <code> SampleAuthPrincipal </code>.
     */
    @Override
    public int hashCode()
    {
        return name.hashCode();
    }
}
```

21.6. The Sample Auth Service Configuration

OpenAM requires that all authentication modules be configured by means of an OpenAM service. At minimum, the service must include an authentication level attribute. Your module can access these configuration attributes in the `options` parameter passed to the `init()` method.

Some observations about the service configuration file follow in the list below.

- The document type for a service configuration file is described in `WEB-INF/sms.dtd` where OpenAM is deployed.
- The service name is taken from the module name: `iPlanetAMAuth`*`module-name`*`Service`. In this case, the service name is `iPlanetAMAuthSampleAuthService`.
- The service must have a localized description, retrieved from a properties file.
- The `i18nFileName` attribute in the service configuration holds the default (non-localized) base name of the Java properties file. The `i18nKey` attributes indicate properties keys to string values in the Java properties file.
- The authentication level attribute name is taken from the module name: `iplanet-am-auth-`*`module-name`*`-auth-level`, where the *module-name* is all lower case. Here, the authentication level attribute is named `iplanet-am-auth-sampleauth-auth-level`.
- The Sample Auth service configuration includes an example `sampleauth-service-specific-attribute`, which can be configured through OpenAM console.

The service configuration file, `amAuthSampleAuth.xml`, is shown below. Save a local copy of this file, which you use when registering the module.

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
   DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS HEADER.

   Copyright (c) 2011 ForgeRock AS. All Rights Reserved

   The contents of this file are subject to the terms
   of the Common Development and Distribution License
   (the License). You may not use this file except in
   compliance with the License.

   You can obtain a copy of the License at
   http://forgerock.org/license/CDDLv1.0.html
   See the License for the specific language governing
   permission and limitations under the License.

   When distributing Covered Code, include this CDDL
   Header Notice in each file and include the License file
   at http://forgerock.org/license/CDDLv1.0.html
   If applicable, add the following below the CDDL Header,
   with the fields enclosed by brackets [] replaced by
   your own identifying information:
   "Portions Copyrighted [year] [name of copyright owner]"
-->
<!DOCTYPE ServicesConfiguration
```

```
    PUBLIC "=//iPlanet//Service Management Services (SMS) 1.0 DTD//EN"
    "jar://com/sun/identity/sm/sms.dtd">

<ServicesConfiguration>
 <Service name="iPlanetAMAuthSampleAuthService" version="1.0">
  <Schema
   serviceHierarchy="/DSAMEConfig/authentication/iPlanetAMAuthSampleAuthService"
   i18nFileName="amAuthSampleAuth" revisionNumber="10"
   i18nKey="sampleauth-service-description">
   <Organization>
    <AttributeSchema name="iplanet-am-auth-sampleauth-auth-level"
     type="single" syntax="number_range" rangeStart="0" rangeEnd="2147483647"
     i18nKey="a500">
     <DefaultValues>
      <Value>1</Value>
     </DefaultValues>
    </AttributeSchema>

    <AttributeSchema name="sampleauth-service-specific-attribute"
     type="single" syntax="string" validator="no" i18nKey="a501">
     <DefaultValues>
      <Value></Value>
     </DefaultValues>
    </AttributeSchema>

    <SubSchema name="serverconfig" inheritance="multiple">
     <AttributeSchema name="iplanet-am-auth-sampleauth-auth-level"
      type="single" syntax="number_range" rangeStart="0" rangeEnd="2147483647"
      i18nKey="a500">
      <DefaultValues>
       <Value>1</Value>
      </DefaultValues>
     </AttributeSchema>

     <AttributeSchema name="sampleauth-service-specific-attribute"
      type="single" syntax="string" validator="no" i18nKey="a501">
      <DefaultValues>
       <Value></Value>
      </DefaultValues>
     </AttributeSchema>

    </SubSchema>
   </Organization>
  </Schema>
 </Service>
</ServicesConfiguration>
```

21.7. Building & Installing the Sample Auth Module

Build the module with Apache Maven, and install the module in OpenAM.

21.7.1. Building the Module

Build the module with Apache Maven, and install the module in OpenAM.

After you successfully build the module, you find the `.jar` file in the `target/` directory of the project.

For information on downloading and building OpenAM sample source code, see How do I access and build the sample code provided for OpenAM 12.x, 13.x and AM (All versions)? in the *Knowledge Base*.

21.7.2. Installing the Module

Installing the sample authentication module consists of copying the .jar to OpenAM's `WEB-INF/lib/` directory, registering the module with OpenAM, and then restarting OpenAM or the web application container where it runs.

1. Copy the sample authentication module .jar file to `WEB-INF/lib/` where OpenAM is deployed.

```
$ cp target/openam-auth-sample*.jar /path/to/tomcat/webapps/openam/WEB-INF/lib/
```

2. Register the module with OpenAM using the **ssoadm** command.

```
$ ssoadm \
 create-svc \
 --adminid amadmin \
 --password-file /tmp/pwd.txt \
 --xmlfile /path/to/amAuthSampleAuth.xml

Service was added.
$ ssoadm \
 register-auth-module \
 --adminid amadmin \
 --password-file /tmp/pwd.txt \
 --authmodule org.forgerock.openam.examples.SampleAuth

Authentication module was registered.
```

   See the **ssoadm** reference in the *Reference* a full list of Authentication Service Management subcommands.

3. Restart OpenAM or the container in which it runs.

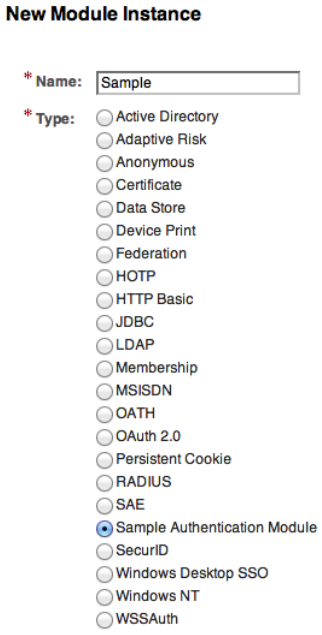
   For example if you deployed OpenAM in Apache Tomcat, then you shut down Tomcat and start it again.

```
$ /path/to/tomcat/bin/shutdown.sh
$ /path/to/tomcat/bin/startup.sh
$ tail -1 /path/to/tomcat/logs/catalina.out
INFO: Server startup in 14736 ms
```

21.8. Configuring & Testing the Sample Auth Module

Authentication modules are registered as services with OpenAM globally, and then set up for use in a particular realm. In this example, you set up the sample authentication module for use in the realm / (Top Level Realm).

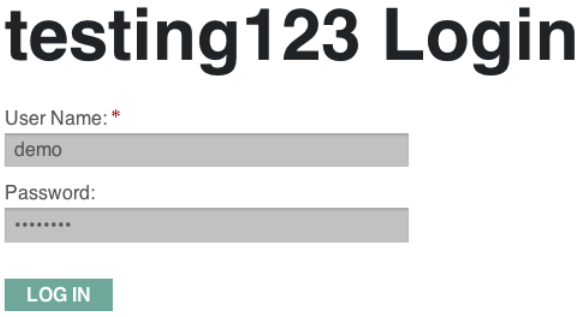
Login to OpenAM Console as OpenAM administrator, `amadmin`, and browse to Access Control > / (Top Level Realm) > Authentication > Module Instances. Then click New, and create an instance of the Sample Authentication Module. Name the module `Sample`.



After creating the module, click the name in the Module Instances list, and configure as appropriate.

Now that the module is configured, logout of OpenAM console.

Finally, try the module by specifying the `Sample` module using a query string parameter. Browse to the login URL such as `http://openam.example.com:8080/openam/UI/Login?module=Sample`, and then authenticate with user name `demo` and password `changeit`.



After authentication you are redirected to the end user page for the demo user. You can logout of OpenAM console, and then try the authentication as users `test1` or `test2` to see what the error handling looks like to the user.

Chapter 22

Customizing Session Quota Exhaustion Actions

This chapter demonstrates a custom session quota exhaustion action plugin. OpenAM calls a session quota exhaustion action plugin when a user tries to open more sessions than their quota allows.

You only need a custom session quota exhaustion action plugin if the built-in actions, described in the *Administration Guide* procedure, *To Configure Session Quotas & Exhaustion Actions* in the *Administration Guide*, are not flexible enough for your deployment.

22.1. Creating & Installing a Custom Session Quota Exhaustion Action

You build custom session quota exhaustion actions into a .jar that you then plug in to OpenAM. You must also add your new action to the Session service configuration, and restart OpenAM in order to be able to configure it for your use.

Your custom session quota exhaustion action implements the `com.iplanet.dpro.session.service.QuotaExhaustionAction` interface, overriding the `action` method. The `action` method performs the action when the session quota is met, and returns `true` only if the request for a new session should *not* be granted.

The example in this chapter simply removes the first session it finds as the session quota exhaustion action.

```
package org.forgerock.openam.examples.quotaexhaustionaction;

import com.iplanet.dpro.session.Session;
import com.iplanet.dpro.session.SessionException;
import com.iplanet.dpro.session.SessionID;
import com.iplanet.dpro.session.service.InternalSession;
import com.iplanet.dpro.session.service.QuotaExhaustionAction;
import com.iplanet.dpro.session.service.SessionService;
import com.sun.identity.shared.debug.Debug;
import java.util.Map;

/**
 * This is a sample {@link QuotaExhaustionAction} implementation,
 * which randomly kills the first session it finds.
 */
public class SampleQuotaExhaustionAction implements QuotaExhaustionAction {
```

```
    private static Debug debug = SessionService.sessionDebug;

    /**
     * Check if the session quota for a given user has been exhausted and
     * if so perform the necessary actions. This implementation randomly
     * destroys the first session it finds.
     *
     * @param is               The InternalSession to be activated.
     * @param existingSessions All existing sessions that belong to the same
     *                         uuid (Map:sid->expiration_time).
     * @return true If the session activation request should be rejected,
     *              otherwise false.
     */
    @Override
    public boolean action(
            InternalSession is,
            Map<String, Long> existingSessions) {
        for (Map.Entry<String, Long> entry : existingSessions.entrySet()) {
            try {
                // Get an actual Session instance based on the session ID.
                Session session =
                        Session.getSession(new SessionID(entry.getKey()));
                // Use the session to destroy itself.
                session.destroySession(session);
                // Only destroy the first session.
                break;
            } catch (SessionException se) {
                if (debug.messageEnabled()) {
                    debug.message("Failed to destroy existing session.", se);
                }
                // In this case, deny the session activation request.
                return true;
            }
        }
        return false;
    }
}
```

The sample plugin source is available online. Get a local clone so that you can try the sample on your system. In the sources you find the following files.

pom.xml

Apache Maven project file for the module

This file specifies how to build the sample plugin, and also specifies its dependencies on OpenAM components and on the Servlet API.

src/main/java/org/forgerock/openam/examples/quotaexhaustionaction/SampleQuotaExhaustionAction.java

Core class for the sample quota exhaustion action plugin

Build the module using Apache Maven.

```
$ cd /path/to/openam-examples-quotaexhaustionaction
$ mvn install
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building OpenAM Example Quota Exhaustion Action 1.0.0-SNAPSHOT
[INFO] ------------------------------------------------------------------------

...

[INFO]
[INFO] --- maven-jar-plugin:2.3.1:jar (default-jar) @ quotaexhaustionaction ---
[INFO] Building jar: .../target/quotaexhaustionaction-1.0.0-SNAPSHOT.jar

...

[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 10.138s
[INFO] Finished at: Mon Nov 25 15:59:10 CET 2013
[INFO] Final Memory: 18M/129M
[INFO] ------------------------------------------------------------------------
```

Copy the .jar to `WEB-INF/lib/` where OpenAM is deployed.

```
$ cp target/*.jar /path/to/tomcat/webapps/openam/WEB-INF/lib/
```

Using the **ssoadm** command or the `ssoadm.jsp` page in OpenAM Console, update the Session service configuration.

```
$ ssoadm \
 set-attr-choicevals \
 --adminid amadmin \
 --password-file /tmp/pwd.txt \
 --servicename iPlanetAMSessionService \
 --schematype Global \
 --attributename iplanet-am-session-constraint-handler \
 --add \
 --choicevalues myKey=\
org.forgerock.openam.examples.quotaexhaustionaction.SampleQuotaExhaustionAction

Choice Values were set.
```

Extract `amSession.properties` and if necessary the localized versions of this file from `openam-core-12.0.0.jar` to `WEB-INF/classes/` where OpenAM is deployed. For example, if OpenAM is deployed under `/path/to/tomcat/webapps/openam`, then you could run the following commands.

```
$ cd /path/to/tomcat/webapps/openam/WEB-INF/classes/
$ jar -xvf ../lib/openam-core-12.0.0.jar amSession.properties
 inflated: amSession.properties
```

Add the following line to `amSession.properties`.

```
myKey=Randomly Destroy Session
```

Restart OpenAM or the container in which it runs.

You can now use the new session quota exhaustion action in OpenAM Console under Configuration > Global > Session > Resulting behavior if session quota exhausted.

Before moving to your test and production environments, be sure to add your .jar and updates to `amSession.properties` into a custom .war that you can then deploy. You must still update the Session service configuration in order to use your custom session quota exhaustion action.

22.2. Listing Session Quota Exhaustion Actions

List session quota exhaustion actions by using the **ssoadm** command or by using the `ssoadm.jsp` page.

```
$ ssoadm \
 get-attr-choicevals \
 --adminid amadmin \
 --password-file /tmp/pwd.txt \
 --servicename iPlanetAMSessionService \
 --schematype Global \
 --attributename iplanet-am-session-constraint-handler

I18n Key                  Choice Value
------------------------- ---...----------------------------------------
choiceDestroyOldSession   org...session.service.DestroyOldestAction
choiceDenyAccess          org...session.service.DenyAccessAction
choiceDestroyNextExpiring org...session.service.DestroyNextExpiringAction
choiceDestroyAll          org...session.service.DestroyAllAction
myKey                     org...examples...SampleQuotaExhaustionAction
```

22.3. Removing a Session Quota Exhaustion Action

Remove a session quota exhaustion action by using the **ssoadm** command or by using the `ssoadm.jsp` page.

```
$ ssoadm \
 remove-attr-choicevals \
 --adminid amadmin \
 --password-file /tmp/pwd.txt \
 --servicename iPlanetAMSessionService \
 --schematype Global \
 --attributename iplanet-am-session-constraint-handler \
 --choicevalues \
 org.forgerock.openam.examples.quotaexhaustionaction.SampleQuotaExhaustionAction

Choice Values were removed.
```

Chapter 23

Creating a Post Authentication Plugin

Post authentication plugins (PAP) let you include custom processing at the end of the authentication process, immediately before the subject is authenticated. Common uses of post authentication plugins include setting cookies and session variables. Post authentication plugins are often used in conjunction with policy agents. The post authentication plugin sets custom session properties, and then the policy agent injects the custom properties into the request header to the protected application.

This chapter explains how to create a post authentication plugin.

23.1. Designing Your Post Authentication Plugin

Your post authentication plugin class implements the `AMPostAuthProcessInterface` interface, and in particular the following three methods.

```
public void onLoginSuccess(
  Map requestParamsMap,
  HttpServletRequest request,
  HttpServletResponse response,
  SSOToken token
) throws AuthenticationException

public void onLoginFailure(
  Map requestParamsMap,
  HttpServletRequest request,
  HttpServletResponse response
) throws AuthenticationException

public void onLogout(
  HttpServletRequest request,
  HttpServletResponse response,
  SSOToken token
) throws AuthenticationException
```

OpenAM calls the `onLoginSuccess()` and `onLoginFailure()` methods immediately before informing the user of login success or failure, respectively. OpenAM calls the `onLogout()` method only when the user actively logs out, not when a user's session times out.

See the *OpenAM Java SDK API Specification* for reference.

These methods can perform whatever processing you require. Yet, know that OpenAM calls your methods synchronously as part of the authentication process. Therefore, if your methods take a

long time to complete, you will keep users waiting. Minimize the processing done in your post authentication methods.

23.2. Building Your Sample Post Authentication Plugin

The following example post authentication plugin sets a session property during successful login, writing to its debug log if the operation fails.

```
package com.forgerock.openam.examples;

import java.util.Map;

import com.iplanet.sso.SSOException;
import com.iplanet.sso.SSOToken;

import com.sun.identity.authentication.spi.AMPostAuthProcessInterface;
import com.sun.identity.authentication.spi.AuthenticationException;
import com.sun.identity.shared.debug.Debug;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class SamplePAP implements AMPostAuthProcessInterface {
    private final static String PROP_NAME = "MyProperty";
    private final static String PROP_VALUE = "MyValue";
    private final static String DEBUG_FILE = "SamplePAP";

    protected Debug debug = Debug.getInstance(DEBUG_FILE);

    public void onLoginSuccess(
            Map requestParamsMap,
            HttpServletRequest request,
            HttpServletResponse response,
            SSOToken token
    ) throws AuthenticationException {
        try {
            token.setProperty(PROP_NAME, PROP_VALUE);
        } catch (SSOException e) {
            debug.error("Unable to set property");
        }
    }

    public void onLoginFailure(
            Map requestParamsMap,
            HttpServletRequest request,
            HttpServletResponse response
    ) throws AuthenticationException {
        // Not used
    }

    public void onLogout(
            HttpServletRequest request,
            HttpServletResponse response,
            SSOToken token
    ) throws AuthenticationException {
        // Not used
```

```
    }
}
```

The sample post authentication plugin source is available online. Get a local clone so that you can try the sample on your system. In the sources you find the following files.

`pom.xml`

Apache Maven project file for the module

This file specifies how to build the sample post authentication plugin, and also specifies its dependencies on OpenAM components and on the Servlet API.

`src/main/java/com/forgerock/openam/examples/SamplePAP.java`

Core class for the sample post authentication plugin

Build the module using Apache Maven.

```
$ cd /path/to/openam-post-auth-sample
$ mvn install
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building openam-post-auth-sample 1.0.0-SNAPSHOT
[INFO] ------------------------------------------------------------------------

...

[INFO]
[INFO] --- maven-jar-plugin:2.3.1:jar (default-jar) @ openam-post-auth-sample --
[INFO] Building jar: .../target/openam-post-auth-sample-1.0.0-SNAPSHOT.jar

...

[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 6.727s
[INFO] Finished at: Mon Nov 25 17:07:23 CET 2013
[INFO] Final Memory: 20M/227M
[INFO] ------------------------------------------------------------------------
```

Copy the .jar to the `WEB-INF/lib` directory where you deployed OpenAM.

```
$ cp target/*.jar /path/to/tomcat/webapps/openam/WEB-INF/lib/
```

Restart OpenAM or the container in which it runs.

23.3. Configuring Your Post Authentication Plugin

You can configure the post authentication plugin for a realm, for a service (authentication chain), or for a role. Where you configure the plugin depends on the scope to which the plugin should apply. Configuring the plugin at the realm level as shown here, for example, ensures that OpenAM calls your plugin for all authentications to the realm.

In OpenAM Console, browse to Access Control > *Realm Name* > Authentication > All Core Settings. In the Authentication Post Processing Classes list, add the sample plugin class, `com.forgerock.openam.examples.SamplePAP`, and then click Save.

Alternatively, you can configure sample plugin for the realm by using the **ssoadm** command.

```
$ ssoadm
 set-svc-attrs
 --adminid amadmin
 --password-file /tmp/pwd.txt
 --servicename iPlanetAMAuthService
 --realm /myRealm
 --attributevalues iplanet-am-auth-post-login-process-class=
 com.forgerock.openam.examples.SamplePAP

iPlanetAMAuthService under /myRealm was
 modified.
```

23.4. Testing Your Post Authentication Plugin

To test the sample post authentication plugin, login successfully to OpenAM in the scope where the plugin is configured. For example, if you configured your plugin for the realm, `/myRealm`, specify the realm in the login URL.

```
http://openam.example.com:8080/openam/UI/Login?realm=myRealm
```

Although as a user you do not notice anywhere in the user interface that OpenAM calls your plugin, a policy agent or custom client code could retrieve the session property that your plugin added to the user session.

Chapter 24

Customizing Policy Evaluation

OpenAM policies let you restrict access to resources based both on identity and group membership, and also on a range of conditions including session age, authentication chain or module used, authentication level, realm, session properties, IP address and DNS name, user profile content, resource environment, date, day, time of day, and time zone. Yet, some deployments require further distinctions for policy evaluation. This chapter explains how to customize policy evaluation for deployments with particular requirements not met by built-in OpenAM functionality.

This chapter shows how to build and use a custom policy plugin that implements a subject condition, an environment condition, and resource attributes.

24.1. About the Sample Plugin

The OpenAM policy framework lets you build plugins that extend subject conditions, environment conditions, and resource attributes.

For information on downloading and building OpenAM sample source code, see How do I access and build the sample code provided for OpenAM 12.x, 13.x and AM (All versions)? in the *Knowledge Base*.

Get a local clone so that you can try the sample on your system. In the sources, you find the following files under the `/path/to/openam-source/openam-samples/policy-evaluation-plugin` directory:

`pom.xml`

Apache Maven project file for the module

This file specifies how to build the sample policy evaluation plugin, and also specifies its dependencies on OpenAM components.

`src/main/java/org/forgerock/openam/examples/SampleAttributeType.java`

Extends the `com.sun.identity.entitlement.ResourceAttribute` interface, and shows an implementation of a resource attribute provider to send an attribute with the response.

`src/main/java/org/forgerock/openam/examples/SampleConditionType.java`

Extends the `com.sun.identity.entitlement.EntitlementCondition` interface, and shows an implementation of a condition that is the length of the user name.

A condition influences whether the policy applies for a given access request. If the condition is fulfilled, then OpenAM includes the policy in the set of policies to evaluate in order to respond to a policy decision request.

`src/main/java/org/forgerock/openam/examples/SampleSubjectType.java`

Extends the `com.sun.identity.entitlement.EntitlementSubject` interface, and shows an implementation that defines a user to whom the policy applies.

A subject, like a condition, influences whether the policy applies. If the subject matches in the context of a given access request, then the policy applies.

`src/main/java/org/forgerock/openam/examples/SampleEntitlementModule.java`
`src/main/resources/META-INF/services/org.forgerock.openam.entitlement.EntitlementModule`

These files serve to register the plugin with OpenAM.

The Java class, `SampleEntitlementModule`, implements the `org.forgerock.openam.entitlement.EntitlementModule` interface. In the sample, this class registers "SampleAttribute", "SampleCondition", and "SampleSubject".

The services file, `org.forgerock.openam.entitlement.EntitlementModule`, holds the fully qualified class name of the `EntitlementModule` that registers the custom implementations. In this case, `org.forgerock.openam.entitlement.EntitlementModule`.

To Build the Sample Plugin

1. If you have not already done so, download and build the samples.

   For information on downloading and building OpenAM sample source code, see How do I access and build the sample code provided for OpenAM 12.x, 13.x and AM (All versions)? in the *Knowledge Base*.

2. Build the module using Apache Maven.

```
$ cd /path/to/openam-policy-eval-sample
$ mvn install
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building openam-policy-eval-sample 1.0.0-SNAPSHOT
[INFO] ------------------------------------------------------------------------

...

[INFO] --- maven-jar-plugin:2.3.2:jar (default-jar) @ openam-policy-eval-sample
[INFO] Building jar: .../target/openam-policy-eval-sample-1.0.0-SNAPSHOT.jar
[INFO]

...

[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 1.480s
[INFO] Finished at: Tue Dec 09 11:45:54 CET 2014
[INFO] Final Memory: 16M/257M
[INFO] ------------------------------------------------------------------------
```

3. Copy the .jar to the `WEB-INF/lib` directory where you deployed OpenAM.

```
$ cp target/*.jar /path/to/tomcat/webapps/openam/WEB-INF/lib/
```

4. Restart OpenAM or the container in which it runs.

24.2. Configuring a Sample Policy Plugin

This section shows how to edit OpenAM policy editor settings to allow policy administrators to configure policies that use the custom subject condition and custom environment condition.

To Configure the Sample Policy Plugin

OpenAM policy editor retrieves resources from the JSON file `policyEditor/locales/locale/translation.json`, where the default *locale* is `en`. The file is located under the directory where the OpenAM war file is unpacked for deployment, such as `/path/to/tomcat/webapps/openam`.

1. Open the `translation.json` file for editing.

   If your editor does not perform JSON syntax checking, then use a service such as http://jsonlint.com to make sure your edits result in valid JSON.

2. In the `policy.subjectTypes` object, add an object defining the strings for the policy editor to use when displaying the custom subject condition.

```
"subjectTypes": {
  "SampleSubject": {
    "title": "Sample Subject",
    "props": {
      "name": "Name"
    }
  },
```

3. In the `policy.conditionTypes` object, add an object defining the strings for the policy editor to use when displaying the custom environment condition.

```
"conditionTypes": {
  "SampleCondition": {
    "title": "Sample Condition",
    "props": {
      "nameLength": "Min. username length"
    }
  },
```

4. When satisfied the JSON is correct, save the file.

   Your custom policy plugin can now be used for new policy applications.

In order to use your custom policy plugin with an existing policy application, see "Adding Custom Policy Implementations to Existing Policy Applications".

24.3. Adding Custom Policy Implementations to Existing Policy Applications

In order to use your custom policy in existing applications, you must update the applications. Note that you cannot update an application that already has policies configured. When there are already policies configured for an application, you must instead first delete the policies, and then update the application.

The following example updates the `iPlanetAMWebAgentService` application in the top level realm of a fresh installation.

```
$ curl \
 --request POST \
 --header "X-OpenAM-Username: amadmin" \
 --header "X-OpenAM-Password: password" \
 --header "Content-Type: application/json" \
 --data "{}" \
 https://openam.example.com:8443/openam/json/authenticate
{"tokenId":"AQIC5wM2...","successUrl":"/openam/console"}

$ curl \
```

```
 --request PUT \
 --header "iPlanetDirectoryPro: AQIC5wM2..." \
 --header "Content-Type: application/json" \
 --data '{
    "name": "iPlanetAMWebAgentService",
    "resources": [
        "*://*:*/*?*",
        "*://*:*/*"
    ],
    "actions": {
        "POST": true,
        "PATCH": true,
        "GET": true,
        "DELETE": true,
        "OPTIONS": true,
        "HEAD": true,
        "PUT": true
    },
    "description": "The built-in Application used by OpenAM Policy Agents.",
    "realm": "/",
    "conditions": [
        "AuthenticateToService",
        "AuthLevelLE",
        "AuthScheme",
        "IPv6",
        "SimpleTime",
        "OAuth2Scope",
        "IPv4",
        "AuthenticateToRealm",
        "OR",
        "AMIdentityMembership",
        "LDAPFilter",
        "AuthLevel",
        "SessionProperty",
        "Session",
        "NOT",
        "AND",
        "ResourceEnvIP",
        "SampleCondition"
    ],
    "resourceComparator": null,
    "applicationType": "iPlanetAMWebAgentService",
    "subjects": [
        "JwtClaim",
        "AuthenticatedUsers",
        "Identity",
        "NOT",
        "AND",
        "NONE",
        "OR",
        "SampleSubject"
    ],
    "attributeNames": [],
    "saveIndex": null,
    "searchIndex": null,
    "entitlementCombiner": "DenyOverride"
}' https://openam.example.com:8443/openam/json/applications/iPlanetAMWebAgentService
```

Notice that the command adds "SampleCondition" to "conditions", and "SampleSubject" to "subjects".

24.4. Trying the Sample Subject and Environment Conditions

Using OpenAM policy editor, create a policy in the "iPlanetAMWebAgentService" of the top level realm that allows HTTP GET access to "http://www.example.com:80/*" and that makes use of the custom subject and environment conditions.

The following listing shows a JSON representation of the policy.

```
{
    "name": "Sample Policy",
    "active": true,
    "description": "Try sample policy plugin",
    "resources": [
        "http://www.example.com:80/*"
    ],
    "applicationName": "iPlanetAMWebAgentService",
    "actionValues": {
        "GET": true
    },
    "subject": {
        "type": "SampleSubject",
        "name": "demo"
    },
    "condition": {
        "type": "SampleCondition",
        "nameLength": 4
    }
}
```

With the policy in place, authenticate both as a user who can request policy decisions and also as a user trying to access a resource. Both of these calls return "tokenId" values for use in the policy decision request.

```
$ curl \
 --request POST \
 --header "X-OpenAM-Username: amadmin" \
 --header "X-OpenAM-Password: password" \
 --header "Content-Type: application/json" \
 --data "{}" \
 https://openam.example.com:8443/openam/json/authenticate

{"tokenId":"AQIC5wM2LY4Sfcw...","successUrl":"/openam/console"}

$ curl \
 --request POST \
 --header "X-OpenAM-Username: demo" \
 --header "X-OpenAM-Password: changeit" \
 --header "Content-Type: application/json" \
 --data "{}" \
 https://openam.example.com:8443/openam/json/authenticate

{"tokenId":"AQIC5wM2LY4Sfcy...","successUrl":"/openam/console"}
```

Use the administrator token ID as the header of the policy decision request, and the user token Id as the subject "ssoToken" value.

```
$ curl \
 --request POST \
 --header "iPlanetDirectoryPro: AQIC5wM2LY4Sfcw..." \
 --header "Content-Type: application/json" \
 --data '{
    "subject": {
      "ssoToken": "AQIC5wM2LY4Sfcy..."},
    "resources": [
        "http://www.example.com:80/index.html"
    ],
    "application": "iPlanetAMWebAgentService"
 }' \
 https://openam.example.com:8443/openam/json/policies?_action=evaluate

[
   {
       "resource": "http://www.example.com:80/index.html",
       "actions": {
           "GET": true
       },
       "attributes": {},
       "advices": {}
   }
]
```

Notice that the "actions" are set in accordance with the policy.

24.5. Trying the Sample Resource Attributes

The sample custom policy plugin can have OpenAM return an attribute with the policy decision. In order to make this work, update your policy to return a "test" attribute.

```
$ curl \
 --request PUT \
 --header "iPlanetDirectoryPro: AQIC5wM2LY4Sfcw..." \
 --header "Content-Type: application/json" \
 --data '{
    "name": "Sample Policy",
    "active": true,
    "description": "Try sample policy plugin",
    "resources": [
        "http://www.example.com:80/*"
    ],
    "applicationName": "iPlanetAMWebAgentService",
    "actionValues": {
        "GET": true
    },
    "subject": {
        "type": "SampleSubject",
        "name": "demo"
    },
    "condition": {
        "type": "SampleCondition",
        "nameLength": 4
    },
    "resourceAttributes": [
        {
            "type": "SampleAttribute",
            "propertyName": "test"
        }
    ]
}' http://openam.example.com:8088/openam/json/policies/Sample%20Policy
```

When you now request the same policy decision as before, OpenAM returns the "test" attribute that you configured in the policy.

```
$ curl \
 --request POST \
 --header "iPlanetDirectoryPro: AQIC5wM2LY4Sfcw..." \
 --header "Content-Type: application/json" \
 --data '{
    "subject": {
      "ssoToken": "AQIC5wM2LY4Sfcy..."},
    "resources": [
        "http://www.example.com:80/index.html"
    ],
    "application": "iPlanetAMWebAgentService"
 }' \
 http://openam.example.com:8080/openam/json/policies?_action=evaluate

[
    {
        "resource": "http://www.example.com/profile",
        "actions": {
            "GET": true
        },
        "attributes": {
            "test": [
                "sample"
            ]
        },
        "advices": {}
    }
]
```

24.6. Extending the ssoadm Classpath

After customizing your OpenAM deployment to use policy evaluation plugins, inform **ssoadm** users to add the jar file containing the plugins to the classpath before running policy management subcommands.

To add a jar file to the **ssoadm** classpath, set the `CLASSPATH` environment variable before running the **ssoadm** command.

```
$ export CLASSPATH=/path/to/jarfile:$CLASSPATH
$ ssoadm ...
```

Chapter 25

Customizing Identity Data Storage

OpenAM maps user and group identities into a realm using data stores. An OpenAM data store relies on a Java identity repository (IdRepo) plugin to implement interaction with the identity repository where the users and groups are stored.

25.1. About the Identity Repository Plugin

This chapter describes how to create a custom identity repository plugin. OpenAM includes built-in support for LDAP and JDBC identity repositories. For most deployments, you therefore do not need to create your own custom identity repository plugin. Only create custom identity repository plugins for deployments with particular requirements not met by built-in OpenAM functionality.

Tip

Before creating your own identity repository plugin, start by reading the OpenAM source code for the `FilesRepo` or `DatabaseRepo` plugins under `com.sun.identity.idm.plugins`.

25.1.1. IdRepo Inheritance

Your identity repository plugin class must extend the `com.sun.identity.idm.IdRepo` abstract class, and must include a constructor method that takes no arguments.

25.1.2. IdRepo Lifecycle

When OpenAM instantiates your IdRepo plugin, it calls the `initialize()` method.

```
public void initialize(Map configParams)
```

The `configParams` are service configuration parameters for the realm where the IdRepo plugin is configured. The `configParams` normally serve to set up communication with the underlying identity data store. OpenAM calls the `initialize()` method once, and considers the identity repository ready for use.

If you encounter errors or exceptions during initialization, catch and store them in your plugin for use later when OpenAM calls other plugin methods.

After initialization, OpenAM calls the `addListener()` and `removeListener()` methods to register listeners that inform OpenAM client code of changes to identities managed by your IdRepo.

```
public int addListener(SSOToken token, IdRepoListener listener)
public void removeListener()
```

You must handle listener registration in your IdRepo plugin, and also return events to OpenAM through the `IdRepoListener`.

When stopping, OpenAM calls your IdRepo plugin `shutdown()` method.

```
public void shutdown()
```

You are not required to implement `shutdown()` unless your IdRepo plugin has shut down work of its own to do, such as close connections to the underlying identity data store.

25.1.3. IdRepo Plugin Capabilities

Your IdRepo plugin provides OpenAM with a generic means to manage subjects—including users and groups but also special types such as roles, realms, and agents— and to create, read, update, delete, and search subjects. In order for OpenAM to determine your plugin's capabilities, it calls the methods described in this section.

```
public Set getSupportedTypes()
```

The `getSupportedTypes()` method returns a set of `IdType` objects, such as `IdType.USER` and `IdType.GROUP`. You can either hard-code the supported types into your plugin, or make them configurable through the IdRepo service.

```
public Set getSupportedOperations(IdType type)
```

The `getSupportedOperations()` method returns a set of `IdOperation` objects, such as `IdOperation.CREATE` and `IdOperation.EDIT`. You can also either hard-code these, or make them configurable.

```
public boolean supportsAuthentication()
```

The `supportsAuthentication()` method returns true if your plugin supports the `authenticate()` method.

25.2. Identity Repository Plugin Implementation

Your IdRepo plugin implements operational methods depending on what you support. These methods perform the operations in your data store.

Create

OpenAM calls `create()` to provision a new identity in the repository, where `name` is the new identity's name, and `attrMap` holds the attributes names and values.

```
public String create(SSOToken token, IdType type, String name, Map attrMap)
  throws IdRepoException, SSOException
```

Read

OpenAM calls the following methods to retrieve subjects in the identity repository, and to check account activity. If your data store does not support binary attributes, return an empty `Map` for `getBinaryAttributes()`.

```
public boolean isExists(
  SSOToken token,
  IdType type,
  String name
) throws IdRepoException, SSOException

public boolean isActive(
  SSOToken token,
  IdType type,
  String name
) throws IdRepoException, SSOException

public Map getAttributes(
  SSOToken token,
  IdType type,
  String name
) throws IdRepoException, SSOException

public Map getAttributes(
  SSOToken token,
  IdType type,
  String name,
  Set attrNames
) throws IdRepoException, SSOException

public Map getBinaryAttributes(
  SSOToken token,
  IdType type,
  String name,
  Set attrNames
) throws IdRepoException, SSOException

public RepoSearchResults search(
  SSOToken token,
  IdType type,
  String pattern,
  Map avPairs,
  boolean recursive,
  int maxResults,
  int maxTime,
  Set returnAttrs
) throws IdRepoException, SSOException

public RepoSearchResults search(
  SSOToken token,
  IdType type,
  String pattern,
  int maxTime,
  int maxResults,
  Set returnAttrs,
  boolean returnAllAttrs,
  int filterOp,
```

```
  Map avPairs,
  boolean recursive
) throws IdRepoException, SSOException
```

Edit

OpenAM calls the following methods to update a subject in the identity repository.

```
public void setAttributes(
  SSOToken token,
  IdType type,
  String name,
  Map attributes,
  boolean isAdd
) throws IdRepoException, SSOException

public void setBinaryAttributes(
  SSOToken token,
  IdType type,
  String name,
  Map attributes,
  boolean isAdd
) throws IdRepoException, SSOException

public void removeAttributes(
  SSOToken token,
  IdType type,
  String name,
  Set attrNames
) throws IdRepoException, SSOException

public void modifyMemberShip(
  SSOToken token,
  IdType type,
  String name,
  Set members,
  IdType membersType,
  int operation
) throws IdRepoException, SSOException

public void setActiveStatus(
  SSOToken token,
  IdType type,
  String name,
  boolean active
)
```

Authenticate

OpenAM calls `authenticate()` with the credentials from the `DataStore` authentication module.

```
public boolean authenticate(Callback[] credentials)
  throws IdRepoException, AuthLoginException
```

Delete

The `delete()` method removes the subject from the identity repository. The `name` specifies the subject.

```
public void delete(SSOToken token, IdType type, String name)
  throws IdRepoException, SSOException
```

Service

The `IdOperation.SERVICE` operation is rarely used in recent OpenAM deployments.

25.3. Identity Repository Plugin Deployment

When you build your IdRepo plugin, include `openam-core-12.0.0.jar` in the classpath. This file is found under `WEB-INF/lib/` where OpenAM is deployed.

You can either package your plugin as a .jar, and then add it to `WEB-INF/lib/`, or add the classes under `WEB-INF/classes/`.

To register your plugin with OpenAM, you add a `SubSchema` to the `sunIdentityRepositoryService` using the **ssoadm** command. First, you create the `SubSchema` document having the following structure.

```
<SubSchema i18nKey="x4000" inheritance="multiple" maintainPriority="no"
           name="CustomRepo" supportsApplicableOrganization="no" validate="yes">
 <AttributeSchema cosQualifier="default" isSearchable="no"
                  name="RequiredValueValidator" syntax="string"
                  type="validator" >
  <DefaultValues>
   <Value>com.sun.identity.sm.RequiredValueValidator</Value>
  </DefaultValues>
 </AttributeSchema>
 <AttributeSchema any="required" cosQualifier="default"
                  i18nKey="x4001" isSearchable="no"
                  name="sunIdRepoClass" syntax="string"
                  type="single" validator="RequiredValueValidator" >
  <DefaultValues>
   <Value>org.test.CustomRepo</Value>
  </DefaultValues>
 </AttributeSchema>
 <AttributeSchema cosQualifier="default" i18nKey="x4002" isSearchable="no"
                  name="sunIdRepoAttributeMapping" syntax="string" type="list">
  <DefaultValues>
    <Value></Value>
  </DefaultValues>
 </AttributeSchema>
</SubSchema>
```

Also include the `AttributeSchema` required to configure your IdRepo plugin.

Notice the `i18nKey` attributes on `SubSchema` elements. The `i18nKey` attribute values correspond to properties in the `amIdRepoService.properties` file under `WEB-INF/classes/` where OpenAM is deployed. OpenAM console displays the label for the configuration user interface that it retrieves from the value of the `i18nKey` property in the `amIdRepoService.properties` file.

To make changes to the properties, first extract `amIdRepoService.properties` and if necessary the localized versions of this file from `openam-core-12.0.0.jar` to `WEB-INF/classes/` where OpenAM is deployed.

For example, if OpenAM is deployed under `/path/to/tomcat/webapps/openam`, then you could run the following commands.

```
$ cd /path/to/tomcat/webapps/openam/WEB-INF/classes/
$ jar -xvf ../lib/openam-core-12.0.0.jar amIdRepoService.properties
 inflated: amIdRepoService.properties
```

Register your plugin using the **ssoadm** command after copy the files into place.

```
$ ssoadm \
 add-sub-schema \
 --adminid amadmin \
 --password-file /tmp/pwd.txt \
 --servicename sunIdentityRepositoryService \
 --schematype Organization \
 --filename customIdRepo.xml
```

Login to OpenAM console as administrator, then then Browse to Access Control > *Realm Name* > Data Stores. In the Data Stores table, click New... to create a Data Store corresponding to your custom IdRepo plugin. In the first screen of the wizard, name the Data Store and select the type corresponding to your plugin. In the second screen of the wizard, add the configuration for your plugin.

After creating the Data Store, create a new subject in the realm to check that your plugin works as expected. You can do this under Access Control > *Realm Name* > Subjects.

If your plugin supports authentication, then users should now be able to authenticate using the `DataStore` module for the realm.

```
http://openam.example.com:8080/openam/UI/Login?realm=test&module=DataStore
```

Index