



Developer's Guide

/ OpenAM 13.5

Latest update: 13.5.2

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

ForgeRock AS
201 Mission St, Suite 2900
San Francisco, CA 94105, USA
+1 415-599-1100 (US)
www.forgerock.com

Copyright © 2011-2018 ForgeRock AS.

Abstract

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



This work is licensed under the Creative Commons Attribution-NonCommercial-NoDerivs 3.0 Unported License.

To view a copy of this license, visit <https://creativecommons.org/licenses/by-nc-nd/3.0/> or send a letter to Creative Commons, 444 Castro Street, Suite 900, Mountain View, California, 94041, USA.

ForgeRock® and ForgeRock Identity Platform™ are trademarks of ForgeRock Inc. or its subsidiaries in the U.S. and in other countries. Trademarks are the property of their respective owners.

UNLESS OTHERWISE MUTUALLY AGREED BY THE PARTIES IN WRITING, LICENSOR OFFERS THE WORK AS-IS AND MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND CONCERNING THE WORK, EXPRESS, IMPLIED, STATUTORY OR OTHERWISE, INCLUDING, WITHOUT LIMITATION, WARRANTIES OF TITLE, MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NONINFRINGEMENT, OR THE ABSENCE OF LATENT OR OTHER DEFECTS, ACCURACY, OR THE PRESENCE OF ABSENCE OF ERRORS, WHETHER OR NOT DISCOVERABLE. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO SUCH EXCLUSION MAY NOT APPLY TO YOU.

EXCEPT TO THE EXTENT REQUIRED BY APPLICABLE LAW, IN NO EVENT WILL LICENSOR BE LIABLE TO YOU ON ANY LEGAL THEORY FOR ANY SPECIAL, INCIDENTAL, CONSEQUENTIAL, PUNITIVE OR EXEMPLARY DAMAGES ARISING OUT OF THIS LICENSE OR THE USE OF THE WORK, EVEN IF LICENSOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

DejaVu Fonts

Bitstream Vera Fonts Copyright

Copyright (c) 2003 by Bitstream, Inc. All Rights Reserved. Bitstream Vera is a trademark of Bitstream, Inc.

Permission is hereby granted, free of charge, to any person obtaining a copy of the fonts accompanying this license ("Fonts") and associated documentation files (the "Font Software"), to reproduce and distribute the Font Software, including without limitation the rights to use, copy, merge, publish, distribute, and/or sell copies of the Font Software, and to permit persons to whom the Font Software is furnished to do so, subject to the following conditions:

The above copyright and trademark notices and this permission notice shall be included in all copies of one or more of the Font Software typefaces.

The Font Software may be modified, altered, or added to, and in particular the designs of glyphs or characters in the Fonts may be modified and additional glyphs or characters may be added to the Fonts, only if the fonts are renamed to names not containing either the words "Bitstream" or the word "Vera".

This License becomes null and void to the extent applicable to Fonts or Font Software that has been modified and is distributed under the "Bitstream Vera" names.

The Font Software may be sold as part of a larger software package but no copy of one or more of the Font Software typefaces may be sold by itself.

THE FONT SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OF COPYRIGHT, PATENT, TRADEMARK, OR OTHER RIGHT. IN NO EVENT SHALL BITSTREAM OR THE GNOME FOUNDATION BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, INCLUDING ANY GENERAL, SPECIAL, INDIRECT, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF THE USE OR INABILITY TO USE THE FONT SOFTWARE OR FROM OTHER DEALINGS IN THE FONT SOFTWARE.

Except as contained in this notice, the names of Gnome, the Gnome Foundation, and Bitstream Inc., shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Font Software without prior written authorization from the Gnome Foundation or Bitstream Inc., respectively. For further information, contact: fonts at gnome dot org.

Arev Fonts Copyright

Copyright (c) 2006 by Tavmjong Bah. All Rights Reserved.

Permission is hereby granted, free of charge, to any person obtaining a copy of the fonts accompanying this license ("Fonts") and associated documentation files (the "Font Software"), to reproduce and distribute the modifications to the Bitstream Vera Font Software, including without limitation the rights to use, copy, merge, publish, distribute, and/or sell copies of the Font Software, and to permit persons to whom the Font Software is furnished to do so, subject to the following conditions:

The above copyright and trademark notices and this permission notice shall be included in all copies of one or more of the Font Software typefaces.

The Font Software may be modified, altered, or added to, and in particular the designs of glyphs or characters in the Fonts may be modified and additional glyphs or characters may be added to the Fonts, only if the fonts are renamed to names not containing either the words "Tavmjong Bah" or the word "Arev".

This License becomes null and void to the extent applicable to Fonts or Font Software that has been modified and is distributed under the "Tavmjong Bah Arev" names.

The Font Software may be sold as part of a larger software package but no copy of one or more of the Font Software typefaces may be sold by itself.

THE FONT SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OF COPYRIGHT, PATENT, TRADEMARK, OR OTHER RIGHT. IN NO EVENT SHALL TAVMJONG BAH BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, INCLUDING ANY GENERAL, SPECIAL, INDIRECT, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF THE USE OR INABILITY TO USE THE FONT SOFTWARE OR FROM OTHER DEALINGS IN THE FONT SOFTWARE.

Except as contained in this notice, the name of Tavmjong Bah shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Font Software without prior written authorization from Tavmjong Bah. For further information, contact: tavmjong @ free . fr.

FontAwesome Copyright

Copyright (c) 2017 by Dave Gandy, <http://fontawesome.io>.

This Font Software is licensed under the SIL Open Font License, Version 1.1. This license is available with a FAQ at: <http://scripts.sil.org/OFL>.

Table of Contents

Preface	v
1. Who Should Use This Guide	v
2. Formatting Conventions	v
3. Accessing Documentation Online	vi
4. Using the ForgeRock.org Site	vi
5. Getting Support and Contacting ForgeRock	vi
1. OpenAM APIs and Protocols	1
1.1. OpenAM APIs	2
1.2. OpenAM SPIs	2
1.3. OpenAM, IPv4, and IPv6	3
2. Developing Client Applications	4
2.1. Using the REST API	4
2.2. Using the OpenAM Java SDK	174
2.3. Using the OpenAM C SDK	197
3. Customizing OpenAM	200
3.1. Customizing Profile Attributes	200
3.2. Customizing OAuth 2.0 Scope Handling	203
3.3. Creating a Custom Authentication Module	208
3.4. Customizing Session Quota Exhaustion Actions	222
3.5. Customizing Policy Evaluation	226
3.6. Customizing Identity Data Storage	234
4. Extending OpenAM	241
4.1. Creating a Post Authentication Plugin	241
4.2. Extending UMA Workflow with Extension Points	245
5. Scripting OpenAM	249
5.1. The Scripting Environment	249
5.2. The Scripting API	253
5.3. Using the Default Scripts	263
6. Building SAML v2.0 Service Providers With Fedlets	279
6.1. Using Fedlets in Java Web Applications	279
6.2. Configuring Java Fedlets By Hand	300
7. Working With the Security Token Service	324
7.1. Publishing STS Instances	324
7.2. Consuming STS Instances	329
7.3. Querying, Validating, and Canceling Tokens	336
7.4. Extending STS to Support Custom Token Types	339
8. Using Secure Attribute Exchange	344
8.1. Installing the Samples	345
8.2. Preparing to Secure SAE Communications	345
8.3. Securing the Identity Provider Side	346
8.4. Securing the Service Provider Side	347
8.5. Trying It Out	348
A. Deprecated REST APIs	349
A.1. Deprecated Session Information APIs	349

A.2. Deprecated Self-Service APIs	349
Index	357

Preface

This guide demonstrates how to handle sessions to permit single sign-on and single logout 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](https://www.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.

5. Getting Support and Contacting ForgeRock

ForgeRock provides support services, professional services, training through ForgeRock University, and partner services to assist you in setting up and maintaining your deployments. For a general overview of these services, see <https://www.forgerock.com>.

ForgeRock has staff members around the globe who support our international customers and partners. For details, visit <https://www.forgerock.com>, or send an email to ForgeRock at info@forgerock.com.

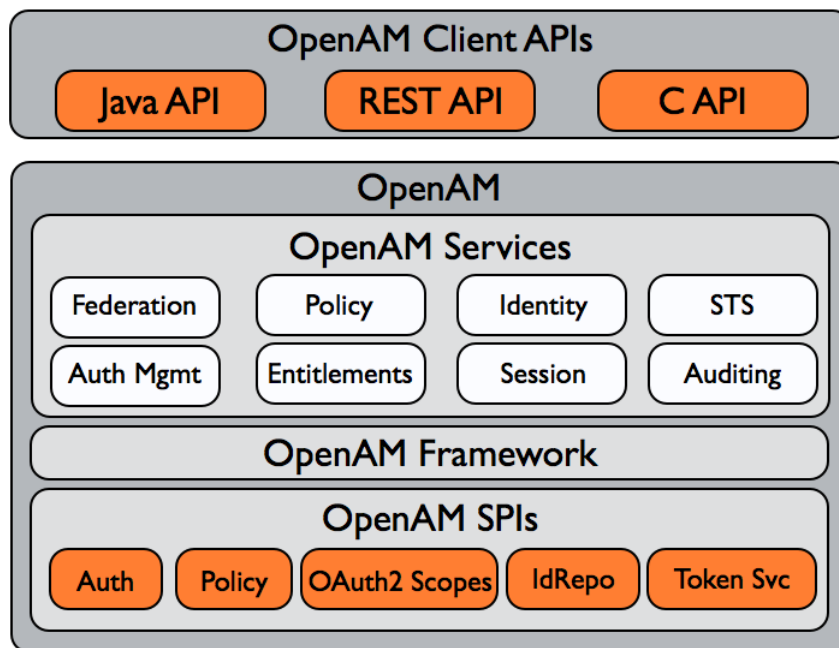
Chapter 1

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 an 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, when playing the role of authorization server, handles scopes, including which token information to return for 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 logout 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 chapter covers client interaction with OpenAM using OpenAM APIs over supported protocols.

2.1. Using the REST API

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

2.1.1. 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 applications. The designs of both HTTP 1.1 and 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, OpenID Connect 1.0, and User-Managed Access (UMA) 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, OpenID Connect 1.0 and UMA 1.0 APIs follow their respective standards.) APIs covered in [Deprecated REST APIs](#) predate CREST, do not use the CREST verbs, and are deprecated in OpenAM 13.5.2-4.

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 POST (or PUT).

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 the API's capabilities 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

OpenAM offers RESTful APIs for access and identity management as follows:

- Authentication and Logout
- Token Validation and Session Information
- Logging
- REST Goto URL Validation
- REST Status Codes
- RESTful Authorization and Policy Management Services
- OAuth 2.0
- OpenID Connect 1.0
- User-Managed Access (UMA)
- Registering Users
- Retrieving Forgotten Usernames
- Replacing Forgotten Passwords
- Displaying Dashboard Applications
- Resetting Device Profiles
- Identity Management
- Realm Management
- Script Management
- Security Token Service
- Troubleshooting Information Recording

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

2.1.2. 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.

2.1.2.1. Supported REST API Versions

The REST API version numbers supported in OpenAM 13.5.2-4 are as follows:

Supported protocol versions

The *protocol* versions supported in OpenAM 13.5.2-4 are:

1.0

Supported resource versions

The *resource* versions supported in OpenAM 13.5.2-4 are shown in the following table.

Table 2.1. Supported resource Versions

Base	End Point	Supported Versions
/json	/authenticate	1.1, 2.0
	/users	1.1, 1.2, 2.0, 2.1, 3.0
	/groups	1.1, 2.0, 2.1, 3.0
	/agents	1.1, 2.0, 2.1, 3.0
	/realms	1.0
	/dashboard	1.0
	/sessions	1.1
	/serverinfo/*	1.1
	/users/{user}/devices/trusted	1.0

Base	End Point	Supported Versions
	/users/{user}/uma/policies	1.0
	/applications	1.0, 2.0
	/resourcetypes	1.0
	/policies	1.0, 2.0
	/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, Chapter 4, "*Changes and Deprecated Functionality*" in the *Release Notes* describes the differences between API versions.

2.1.2.2. 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 "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 Chapter 20, "*Configuring REST APIs*" in the *Administration Guide*.

2.1.2.3. 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 Chapter 20, "*Configuring REST APIs*" in the *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 "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...TU30Q*",
  "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 Chapter 20, "Configuring REST APIs" in the *Administration Guide*.

2.1.3. 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 section. The following is an example token that has not been encoded:

```
AQIC5wM2LY4SfczntBbXvEA0uECbqMY3J4NW3byH6xwgkGE=@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:

```
AQIC5wM2LY4SfczntBbXvEA0uECbqMY3J4NW3byH6xwgkGE.*AAJTSQACMDE.*
```

2.1.4. Specifying Realms in REST API Calls

This section describes how to work with realms when making REST API calls to OpenAM.

Realms can be specified in three ways when making a REST API call to OpenAM:

DNS Alias

When making a REST API call, the DNS alias of a realm can be specified in the subdomain and domain name components of the REST endpoint.

To list all users in the top-level realm use the DNS alias of the OpenAM instance, for example the REST endpoint would be:

```
https://openam.example.com:8443/openam/json/users?_queryId=*
```

To list all users in a realm with DNS alias `suppliers.example.com` the REST endpoint would be:

```
https://suppliers.example.com:8443/openam/json/users?_queryId=*
```


Path

When making a REST API call, the realm, or realm alias, can be specified in the path component of the REST endpoint.

To authenticate a user in the top-level realm the REST endpoint would be:

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

To authenticate a user in a realm named `customers` the REST endpoint would be:

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

Subrealms are supported and should be separated with a forward slash (/).

For example, to authenticate to a subrealm named `europa` of a realm named `partners`, the REST endpoint would be:

```
https://openam.example.com:8443/openam/json/partners/europa/authenticate
```

Query Parameter

When making a REST API call the realm, or realm alias, can be specified as the value of a query parameter named `realm`.

To list the groups in the top-level realm the REST endpoint would be:

```
https://openam.example.com:8443/openam/json/groups?_queryId=*
```

To list the groups in a realm named `partners` the REST endpoint would be:

```
https://openam.example.com:8443/openam/json/groups?realm=/partners&_queryId=*
```

Important

When working with a named subrealm of the top-level realm a forward slash preceding the realm name is required. You should not use a forward slash when using a realm alias.

Subrealms are supported and should be separated with a forward slash (/).

To authenticate a user in a subrealm named `europa` of a realm named `partners` the REST endpoint would be:

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

If more than one of the above methods is used to specify realms in a REST endpoint, OpenAM applies the following rules to determine the realm to use.

1. If realms are specified using both the DNS alias and path methods, they are concatenated together.

For example, the following REST endpoint returns users in a subrealm named `europa` of a realm with DNS alias `suppliers`.

```
https://suppliers.example.com:8443/openam/json/europa/users?_queryId=*
```

2. If realms are specified using the `realm` query parameter, they override anything specified in either the DNS alias or path method.

For example, the following REST endpoint returns users in a subrealm of the `customers` realm, named `asia`.

```
https://suppliers.example.com:8443/openam/json/europe/users?realm=/customers/asia&_queryId=*
```

2.1.5. Authentication and Logout

You can use REST-like APIs under `/json/authenticate` and `/json/sessions` for authentication and for logout.

The `/json/authenticate` endpoint does not support the CRUDPAQ verbs and therefore does not technically satisfy REST architectural requirements. The term *REST-like* describes this endpoint better than *REST*.

The simplest user name/password authentication returns a `tokenId` that applications can present as a cookie value for other operations that require authentication. The type of `tokenId` returned varies depending on whether stateless sessions are enabled in the realm to which the user authenticates:

- If stateless sessions are not enabled, the `tokenId` is an OpenAM SSO token.
- If stateless sessions are enabled, the `tokenId` is an OpenAM SSO token that includes an encoded OpenAM session.

Developers should be aware that the size of the `tokenId` for stateless sessions—2000 bytes or greater—is considerably longer than for stateful sessions—approximately 100 bytes. For more information about stateful and stateless session tokens, see Section 9.2, "Session Cookies" in the *Administration Guide*.

When authenticating with a user name and password, 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 "Content-Type: application/json" \
  --header "X-OpenAM-Username: demo" \
  --header "X-OpenAM-Password: changeit" \
  --data "{}" \
  https://openam.example.com:8443/openam/json/authenticate
{ "tokenId": "AQIC5w...NTcy*", "successUrl": "/openam/console" }
```

To use UTF-8 user names and passwords in calls to the `/json/authenticate` endpoint, base64-encode the string, and then wrap the string as described in RFC 2047:

```
encoded-word = "=?" charset "?" encoding "?" encoded-text "=?"
```

For example, to authenticate using a UTF-8 username, such as `dēmø`, perform the following steps:

1. Encode the string in base64 format: `yZfDq8mxw7g=`.
2. Wrap the base64-encoded string as per RFC 2047: `=?UTF-8?B?yZfDq8mxw7g=?=`.
3. Use the result in the `X-OpenAM-Username` header passed to the authentication endpoint as follows:

```
$ curl \
--request POST \
\
--header "Content-Type: application/json" \
\
--header "X-OpenAM-Username: =?UTF-8?B?yZfDq8mxw7g=?=" \
\
--header "X-OpenAM-Password: changeit" \
\
--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 \
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: " } ] }, { "type": "PasswordCallback", "output": [ { "name": "prompt", "value": " Password: " } ] },
{ "type": "NameCallback", "input": [ { "name": "IDToken1", "value": "demo" } ] } ] }' \
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: " } ] }, { "type": "PasswordCallback", "output": [ { "name": "prompt", "value": " Password: " } ] },
{ "type": "NameCallback", "input": [ { "name": "IDToken1", "value": "demo" } ] } ] }' \
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 Section 2.4, "Configuring Core Authentication Attributes" 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 "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 Section 2.10, "Authentication Levels and 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..*" \
  "https://openam.example.com:8443/openam/json/sessions/?_action=logout"
{"result":"Successfully logged out"}
```

2.1.5.1. Load Balancer and Proxy Layer Requirements

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 Section 4.4, "Handling HTTP Request Headers" in the *Installation Guide*.

2.1.5.2. Windows Desktop SSO Requirements

When authenticating with Windows Desktop SSO, add an `Authorization` header containing the string `Basic`, followed by a base64-encoded string of the username, a colon character, and the password. In the following example, the credentials `demo:changeit` are base64-encoded into the string `ZGVtbzpjajGFuZ2VpdA==`:

```
$ curl \
--request POST \
--header "Content-Type: application/json" \
--header "X-OpenAM-Username: demo" \
--header "X-OpenAM-Password: changeit" \
--header "Authorization: Basic ZGVtbzpjajGFuZ2VpdA==" \
--data "{}" \
https://openam.example.com:8443/openam/json/authenticate

{ "tokenId": "AQIC5w...NTcy*", "successUrl": "/openam/console" }
```

2.1.6. Using the Session Token After Authentication

The following is a common scenario when accessing OpenAM by using REST API calls:

- First, call the `/json/authenticate` endpoint to log a user in to OpenAM. This REST API call returns a `tokenId` value, which is used in subsequent REST API calls to identify the user:

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

{ "tokenId": "AQIC5w...NTcy*", "successUrl": "/openam/console" }
```

The returned `tokenId` is known as a session token (also referred to as an SSO token). REST API calls made after successful authentication to OpenAM must present the session token in the HTTP header as proof of authentication.

- Next, call one or more additional REST APIs on behalf of the logged-in user. Each REST API call passes the user's `tokenId` back to OpenAM in the HTTP header as proof of previous authentication.

The following is a *partial* example of a **curl** command that inserts the token ID returned from a prior successful OpenAM authentication attempt into the HTTP header:

```
$ curl \
--request POST
\
--header "Content-Type: application/json"
\
--header "iPlanetDirectoryPro: AQC5w...NTcy*"
\
--data '{
  ...
}
```

Observe that the session token is inserted into a header field named `iPlanetDirectoryPro`. This header field name must correspond to the name of the OpenAM session cookie—by default, `iPlanetDirectoryPro`. You can find the cookie name in the OpenAM console by navigating to Deployment > Servers > *Server Name* > Security > Cookie, in the Cookie Name field of the OpenAM console.

Once a user has authenticated, it is *not* necessary to insert login credentials in the HTTP header in subsequent REST API calls. Note the absence of `X-OpenAM-Username` and `X-OpenAM-Password` headers in the preceding example.

Users are required to have appropriate privileges in order to access OpenAM functionality using the REST API. For example, users who lack administrative privileges cannot create OpenAM realms. For more information on the OpenAM privilege model, see Section 4.1, "Managing Realms" in the *Administration Guide*.

- Finally, call the REST API to log the user out of OpenAM as described in Section 2.1.5, "Authentication and Logout". As with other REST API calls made after a user has authenticated, the REST API call to log out of OpenAM requires the user's `tokenID` in the HTTP header.

2.1.7. Filtering, Sorting, and Paging Results

Some OpenAM endpoints support additional query string parameters when querying the REST APIs to manipulate the returned data.

The query string parameters for manipulating returned results are:

`_queryFilter`

The `_queryFilter` parameter can take `true` to return every result, `false` to return no results, 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.

Note

Supported fields and operator codes vary depending on the endpoint.

The operators codes are as follows:

- **co**: contains
- **eq**: equals
- **ge**: greater than or equal to
- **gt**: greater than
- **le**: less than or equal to
- **lt**: less than
- **pr**: *field* exists, *field* is present

Note

Do not set a *value* when using this operator.

- **sw**: starts with

Filters can be composed of multiple expressions by a using boolean operator **AND**, **OR**, or **!** (NOT) and by using parentheses, (*expression*) to group expressions.

Regular expressions are implemented for some operators, so you can create a filter that includes or excludes certain records.

You must URL-encode the *filter* expression in `_queryFilter=filter`.

The following example returns resource types with a *name* that contains **Service** and also has a *pattern* that starts with **http**:

```
$ curl \
--header "iPlanetDirectoryPro: AQIC5..." \
--get \
--data-urlencode \
'_queryFilter=name co "Service" and patterns sw "http" ' \
https://openam.example.com:8443/openam/json/resourcetypes
```

_fields

You can use `_fields=field-name[,field-name...]` to limit the fields returned in the output.

The following example returns the *name* and *creationDate* of all policies in the top level realm:

```
$ curl \
--header "iPlanetDirectoryPro: AQIC5..."
\
--get
\
--data-urlencode '_queryFilter=true'
\
--data-urlencode '_fields=name,creationDate' \
https://openam.example.com:8443/openam/json/policies
```

_prettyPrint

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

_pageSize

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

_pagedResultsOffset

You can use `_pagedResultsOffset=integer` to return results starting at a specified result when using paged results.

_sortKeys

You can use `_sortKeys=[+-]field-name[,field-name...]` to sort the results returned, where *field-name* represents a field in the returned JSON. Optionally use the `+` prefix to sort in ascending order (the default), or `-` to sort in descending order.

The following example returns all applications in the top level realm, sorted in descending *creationDate* order:

```
$ curl \
--header "iPlanetDirectoryPro: AQIC5..."
\
--get
\
--data-urlencode '_queryFilter=true'
\
--data-urlencode '_sortKeys=-creationDate' \
https://openam.example.com:8443/openam/json/applications
```

2.1.8. Server Information

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

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

```

    ".example.com"
  ],
  "protectedUserAttributes": [],
  "cookieName": "iPlanetDirectoryPro",
  "secureCookie": false,
  "forgotPassword": "false",
  "forgotUsername": "false",
  "kbaEnabled": "false",
  "selfRegistration": "false",
  "lang": "en-US",
  "successfulUserRegistrationDestination": "default",
  "socialImplementations": [
    {
      "iconPath": "XUI/images/logos/facebook.png",
      "authnChain": "FacebookSocialAuthenticationService",
      "displayName": "Facebook",
      "valid": true
    }
  ],
  "referralsEnabled": "false",
  "zeroPageLogin": {
    "enabled": false,
    "referrerWhitelist": [
      ""
    ],
    "allowedWithoutReferer": true
  },
  "realm": "/",
  "xuiUserSessionValidationEnabled": true,
  "FQDN": "openam.example.com"
}

```

2.1.9. Token Validation and Session Information

OpenAM provides REST APIs under `/json/sessions` for validating SSO tokens and getting information about active sessions.

2.1.9.1. Validating Sessions

To check over REST whether a session token is valid, perform an HTTP POST to the `/json/sessions/` endpoint using the `getSessionInfo` action. The endpoint validates the session token provided in the `iPlanetDirectoryPro` header by default. To validate a different session token, include it as the value of the `tokenId` query parameter.

If the session token is not valid, a `"valid": false` JSON message is returned, as shown below:

```

$ curl \
--request POST \
\
--header "iPlanetDirectoryPro: AQIC4Dm...NTcy*" \
http://openam.example.com:8080/openam/json/sessions/?_action=getSessionInfo&tokenId=64d.5e5510n.1d
{
  "valid": false
}

```

2.1.9.2. Obtaining Information About Sessions

To obtain information about a session, perform an HTTP POST to the `/json/sessions/` endpoint, using the `getSessionInfo` action. The endpoint will return information about the session token provided in the `iPlanetDirectoryPro` header by default. To get information about a different session token, include it as the value of the `tokenId` query parameter.

For example, the following shows an administrative user passing their session token in the `iPlanetDirectoryPro` header, and the session token of the `demo` user as the `tokenId` query parameter:

```
$ curl \
--request POST \
--header "iPlanetDirectoryPro: AQIC4Dm...NTcy*" \
http://openam.example.com:8080/openam/json/sessions/?_action=getSessionInfo&tokenId=AQIC5...QAA*
{
  "username": "demo",
  "universalId": "id=demo,ou=user,dc=openam,dc=forgerock,dc=org",
  "realm": "/",
  "sessionHandle": "shandle:AQIC5wM2LY4SfcwbAHB6MVwCq-0Yvy9j0vjlbjLrT-797oE
.*AAJTSQACMDEAALNLABQtMzQ20TawMTU3MTg5MTUzNDUzOAACUzEAAA.*",
  "latestAccessTime": "2017-01-16T13:37:44Z",
  "maxIdleExpirationTime": "2017-01-16T14:07:44Z",
  "maxSessionExpirationTime": "2017-01-16T15:34:41Z"
}
```

2.1.9.3. Refreshing Stateful Sessions

To reset the idle time of a stateful session using REST, perform an HTTP POST to the `/json/sessions/` endpoint, using the `refresh` action. The endpoint will refresh the session token provided in the `iPlanetDirectoryPro` header by default. To refresh a different session token, include it as the value of the `tokenId` query parameter.

The following example shows an administrative user passing their session token in the `iPlanetDirectoryPro` header, and the session token of the `demo` user as the `tokenId` query parameter:

```
$ curl \
--request POST \
--header "iPlanetDirectoryPro: AQIC5w...NTcy*" \
http://openam.example.com:8080/openam/json/sessions/?_action=refresh&tokenId=BXCCq...NX*1*
{
  "uid": "demo",
  "realm": "/",
  "idletime": 4,
  "maxidletime": 30,
  "maxsessiontime": 120,
  "maxtime": 7195
}
```

On success, OpenAM resets the idle time for the stateful session, and returns timeout details of the session.

Resetting a stateful session's idle time triggers a write operation to the Core Token Service token store. Therefore, to avoid the overhead of write operations to the token store, be careful to use the `refresh` action only if you want to reset a stateful session's idle time.

Because OpenAM does not monitor idle time for stateless sessions, do not use the `tokenId` of a stateless session when refreshing a session's idle time.

2.1.9.4. Invalidating Sessions

To invalidate a session, perform an HTTP POST to the `/json/sessions/` endpoint using the `logout` action. The endpoint will invalidate the session token provided in the `iPlanetDirectoryPro` header by default. To refresh a different session token, include it as the value of the `tokenId` query parameter.

For example, the following shows an administrative user passing their session token in the `iPlanetDirectoryPro` header, and the session token of the `demo` user as the `tokenId` query parameter:

```
$ curl \
--request POST \
--header "iPlanetDirectoryPro: AQIC5w...NTcy*" \
http://openam.example.com:8080/openam/json/sessions/?_action=logout&tokenId=BXCCq...NX*1*
{"result": "Successfully logged out"}
```

On success, OpenAM invalidates the session and returns a success message.

If the token is not valid and cannot be invalidated an error message is returned, as follows:

```
{"result": "Token has expired"}
```

2.1.9.5. Getting and Setting Session Properties

OpenAM lets you read and update properties on users' sessions using REST API calls.

Before you can perform operations on session properties using the REST API, you must first define the properties you want to set in the Session Property Whitelist Service configuration. For information on whitelisting session properties, see Section 1.4.21, "Session Property Whitelist" in the *Reference*.

You can use REST API calls for the following purposes:

- To retrieve the names of the properties that you can read or update. This is the same set of properties configured in the Session Property Whitelist Service.
- To read property values.
- To update property values.

Session state affects the ability to set and delete properties as follows:

- You can set and delete properties on a stateful session at any time during the session's lifetime.
- You can only set and update properties on a stateless session during the authentication process, before the user receives the session token from OpenAM. For example, you could set or delete properties on a stateless session from within a post-authentication plugin.

Differentiate the user who performs the operation on session properties from the session affected by the operation as follows:

- Specify the session token of the user performing the operation on session properties in the `iPlanetDirectoryPro` header.
- Specify the session token of the user whose session is to be read or modified as the `tokenId` parameter to the REST API call.
- Omit the `tokenId` parameter from the REST API call if the session of the user performing the operation is the session that you want to read or modify.

The following examples assume that you configured a property named `LoginLocation` in the Session Property Whitelist Service configuration.

To retrieve the names of the properties you can get or set, and their values, perform an HTTP POST to the resource URL, `/json/sessions/`, using the `getSessionProperties` action as shown in the following example:

```
$ curl \
--request POST
\
--header "iPlanetDirectoryPro: AQIC5w...NTcy*" \
http://openam.example.com:8080/openam/json/sessions/?_action=getSessionProperties&tokenId=BXCCq...NX*1*
{
  "LoginLocation": ""
}
```

To set the value of a session property, perform an HTTP POST to the resource URL, `/json/sessions/`, using the `updateSessionProperties` action. If no `tokenId` parameter is present in the REST API call, the session affected by the operation is the session specified in the `iPlanetDirectoryPro` header, as follows:

```
$ curl \
--request POST
\
--header "Content-Type: application/json"
\
--header "iPlanetDirectoryPro: BXCCq...NX*1*"
\
--data '{"LoginLocation":"40.748440, -73.984559"}' \
http://openam.example.com:8080/openam/json/sessions/?_action=updateSessionProperties
{
  "LoginLocation": "40.748440, -73.984559"
}
```

You can set multiple properties in a single REST API call by specifying a set of fields and their values in the JSON data. For example:

```
--data '{"property1":"value1", "property2":"value2"}'
```

To set the value of a session property on another user's session, specify the session token of the user performing the `updateSessionProperties` action in the `iPlanetDirectoryPro`, and specify the session token to be modified as the value of the `tokenId` parameter:

```
$ curl \
--request POST
\
--header "Content-Type: application/json"
\
--header "iplanetDirectoryPro: AQIC5w...NTcy*"
\
--data '{"LoginLocation":"40.748440, -73.984559"}' \
http://openam.example.com:8080/openam/json/sessions/?_action=updateSessionProperties&tokenId=BXCCq...NX*1*
{
  "LoginLocation": "40.748440, -73.984559"
}
```

If the user attempting to modify the session does not have sufficient access privileges, the preceding examples result in a 403 Forbidden error.

You cannot set properties internal to OpenAM sessions. If you try to modify an internal property in a REST API call, a 403 Forbidden error is returned. For example:

```
$ curl \
--request POST
\
--header "Content-Type: application/json"
\
--header "iplanetDirectoryPro: AQIC5w...NTcy*"
\
--data '{"AuthLevel":"5"}' \
http://openam.example.com:8080/openam/json/sessions/?_action=updateSessionProperties&tokenId=BXCCq...NX*1*
{
  "code": 403,
  "reason": "Forbidden",
  "message": "Forbidden"
}
```

2.1.10. REST Goto URL Validation

You can set valid goto URLs using the OpenAM console by following the instructions in Section 2.13, "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/"}
```

2.1.11. Logging

OpenAM 13.5.2-4 supports two Audit Logging Services: a new common REST-based Audit Logging Service, and the legacy Logging Service, which is based on a Java SDK and is available in OpenAM versions prior to OpenAM 13. The legacy Logging Service is deprecated in OpenAM 13.5.2-4.

Both audit facilities log OpenAM REST API calls.

2.1.11.1. Common Audit Logging of REST API Calls

OpenAM logs information about all REST API calls to the `access` topic. For more information about OpenAM audit topics, see Section 6.2, "Audit Log Topics" in the *Administration Guide*.

Locate specific REST endpoints in the `http.path` log file property.

2.1.11.2. Legacy Logging of REST API Calls

OpenAM logs information about REST API calls 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 contains 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.

2.1.12. 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 "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."
}
```

2.1.13. RESTful Authorization and Policy Management Services

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

2.1.13.1. About the REST Policy Endpoints

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

- Under `/json{/realm}/resourcetypes`, you find a JSON-based API for managing resource types.
- Under `/json{/realm}/applications` and `/json/applicationtypes` you find JSON-based APIs for administering policy sets and reading application types.
- Under `/json{/realm}/policies`, you find a JSON-based API for policy management and evaluation.
- 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.

Before making a REST API call to request a policy decision or manage a policy component, make sure that you have:

- Authenticated successfully to OpenAM as a user with sufficient privileges to make the REST API call
- Obtained the session token returned after successful authentication

When making the REST API call, pass the session token in the HTTP header. For more information about the OpenAM session token and its use in REST API calls, see Section 2.1.6, "Using the Session Token After Authentication".

2.1.13.2. 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 Section 2.1.13.2.1, "Requesting Policy Decisions For Specific Resources".

To request decisions for a resource and all resources beneath it, see Section 2.1.13.2.3, "Requesting Policy Decisions For a Tree of Resources".

2.1.13.2.1. 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 policy set, `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 policy set, and defaults to `iPlanetAMWebAgentService` if not specified.

For more on policy sets, see Section 2.1.13.5, "Managing Policy Sets".

"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 Section 2.1.5, "Authentication and 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 "Content-Type: application/json" \
--header "iPlanetDirectoryPro: AQIC5..." \
--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 Section 2.1.13.2.2, "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.

2.1.13.2.2. 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 Section 2.1.13.6, "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 stateful or stateless session has been active longer than allowed by the condition. The advice will also show if 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

2.1.13.2.3. 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 policy set, `iPlanetAMWebAgentService`, you can request decisions for resource URLs.

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

"application"

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

For more on policy sets, see Section 2.1.13.5, "Managing Policy Sets".

"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, Section 2.1.5, "Authentication and 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 "Content-Type: application/json" \
--header "iPlanetDirectoryPro: AQIC5...NDU1*" \
--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.

2.1.13.3. Managing Resource Types

This section describes the process of using the OpenAM REST API for managing resource types, which define a template for the resources that policies apply to, and the actions associated with those resources.

For information on creating resource types by using the OpenAM console, see Section 3.1.1, "OpenAM Resource Types, Policy Sets, and Policies" in the *Administration Guide*.

OpenAM provides the `resourcetypes` REST endpoint for the following:

- Section 2.1.13.3.1, "Querying Resource Types"
- Section 2.1.13.3.2, "Reading a Specific Resource Type"
- Section 2.1.13.3.3, "Creating a Resource Type"
- Section 2.1.13.3.4, "Updating a Resource Type"
- Section 2.1.13.3.5, "Deleting a Specific Resource Type"

Resource types are realm specific, hence the URI for the resource types API can contain a realm component, such as `/json{/realm}/resourcetypes`. If the realm is not specified in the URI, the top level realm is used.

Resource types are represented in JSON and take the following form. Resource types are built from standard JSON objects and values (strings, numbers, objects, sets, arrays, `true`, `false`, and `null`). Each resource type has a unique, system-generated UUID, which must be used when modifying existing resource types. Renaming a resource type will not affect the UUID.

```
{
  "uuid": "12345a67-8f0b-123c-45de-6fab78cd01e2",
  "name": "URL",
  "description": "The built-in URL Resource Type available to OpenAM Policies.",
  "patterns": [
    "*/**/*/*?*",
    "*/**/*/*"
  ],
  "actions": {
    "POST": true,
    "PATCH": true,
    "GET": true,
    "DELETE": true,
    "OPTIONS": true,
    "HEAD": true,
    "PUT": true
  },
  "createdBy": "id=dsameuser,ou=user,dc=openam,dc=forgerock,dc=org",
  "creationDate": 1422892465848,
  "lastModifiedBy": "id=dsameuser,ou=user,dc=openam,dc=forgerock,dc=org",
  "lastModifiedDate": 1422892465848
}
```

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

"uuid"

String matching the unique identifier OpenAM generated for the resource type when created.

"name"

The name provided for the resource type.

"description"

An optional text string to help identify the resource type.

"patterns"

An array of resource patterns specifying individual URLs or resource names to protect.

For more information on patterns in resource types and policies, see Section 3.3.4, "Specifying Resource Patterns with Wildcards" in the *Administration Guide*

"actions"

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

"createdBy"

A string containing the universal identifier DN of the subject that created the resource type.

"creationDate"

An integer containing the creation date and time, in ISO 8601 format.

"LastModifiedBy"

A string containing the universal identifier DN of the subject that most recently updated the resource type.

If the resource type has not been modified since it was created, this will be the same value as `createdBy`.

"LastModifiedDate"

An string containing the last modified date and time, in ISO 8601 format.

If the resource type has not been modified since it was created, this will be the same value as `creationDate`.

2.1.13.3.1. Querying Resource Types

To list all the resource types in a realm, perform an HTTP GET to the `/json{/realm}/resourcetypes` endpoint, with a `_queryFilter` parameter set to `true`.

Note

If the realm is not specified in the URL, OpenAM returns resource types in the top level realm.

The `iPlanetDirectoryPro` header is required and should contain the SSO token of an administrative user, such as `amAdmin`, who has access to perform the operation.

```
$ curl \
--header "iPlanetDirectoryPro: AQIC5..." \
https://openam.example.com:8443/openam/json/myrealm/resourcetypes?_queryFilter=true
{
  "result": [
    {
      "uuid": "12345a67-8f0b-123c-45de-6fab78cd01e3",
      "name": "LIGHTS",
      "description": "",
      "patterns": [
        "light://*/*"
      ],
      "actions": {
        "switch_off": true,
        "switch_on": true
      },
      "createdBy": "id=amadmin,ou=user,dc=openam,dc=forgerock,dc=org",
      "creationDate": 1431013059131,
      "lastModifiedBy": "id=amadmin,ou=user,dc=openam,dc=forgerock,dc=org",
      "lastModifiedDate": 1431013069803
    }
  ],
  "resultCount": 1,
  "pagedResultsCookie": null,
  "remainingPagedResults": 0
}
```

Additional query strings can be specified to alter the returned results. For more information, see Section 2.1.7, "Filtering, Sorting, and Paging Results".

Table 2.2. Supported `_queryFilter` Fields and Operators

Field	Supported Operators
<code>uuid</code>	Equals (<code>eq</code>), Contains (<code>co</code>), Starts with (<code>sw</code>)
<code>name</code>	Equals (<code>eq</code>), Contains (<code>co</code>), Starts with (<code>sw</code>)
<code>description</code>	Equals (<code>eq</code>), Contains (<code>co</code>), Starts with (<code>sw</code>)
<code>patterns</code>	Equals (<code>eq</code>), Contains (<code>co</code>), Starts with (<code>sw</code>)
<code>actions</code>	Equals (<code>eq</code>), Contains (<code>co</code>), Starts with (<code>sw</code>)

2.1.13.3.2. Reading a Specific Resource Type

To read an individual resource types in a realm, perform an HTTP GET to the `/json{/realm}/resourcetypes` endpoint, and specify the UUID in the URL.

Note

If the realm is not specified in the URL, OpenAM uses the top level realm.

The `iPlanetDirectoryPro` header is required and should contain the SSO token of an administrative user, such as `amAdmin`, who has access to perform the operation.

```
$ curl \
--header "iPlanetDirectoryPro: AQIC5..." \
https://openam.example.com:8443/openam/json/myrealm/resourcetypes/12345a67-8f0b-123c-45de-6fab78cd01e3
{
  "uuid": "12345a67-8f0b-123c-45de-6fab78cd01e3",
  "name": "LIGHTS",
  "description": "",
  "patterns": [
    "light://*/*"
  ],
  "actions": {
    "switch_off": true,
    "switch_on": true
  },
  "createdBy": "id=amadmin,ou=user,dc=openam,dc=forgerock,dc=org",
  "creationDate": 1431013059131,
  "lastModifiedBy": "id=amadmin,ou=user,dc=openam,dc=forgerock,dc=org",
  "lastModifiedDate": 1431013069803
}
```

2.1.13.3.3. Creating a Resource Type

To create a resource type in a realm, perform an HTTP POST to the `/json{/realm}/resourcetypes` endpoint, with an `_action` parameter set to `create`. Include a JSON representation of the resource type in the POST data.

Note

If the realm is not specified in the URL, OpenAM creates the resource type in the top level realm.

The `iPlanetDirectoryPro` header is required and should contain the SSO token of an administrative user, such as `amAdmin`, who has access to perform the operation.

Do not use special characters within resource type, policy, or policy set names (for example, "my+resource+type") when using the console or REST endpoints. Using the special characters listed below causes OpenAM to return a 400 Bad Request error. The special characters are: double quotes ("), plus sign (+), comma (,), less than (<), equals (=), greater than (>), backslash (\), forward slash (/), semicolon (;), and null (`\u0000`).

```
$ curl \
--request POST
\
--header "Content-Type: application/json"
\
--header "iPlanetDirectoryPro: AQIC5..."
\
--data '{
  "name": "My Resource Type",
  "actions": {
    "LEFT": true,
    "RIGHT": true,
    "UP": true,
    "DOWN": true
  },
  "patterns": [
    "http://device/location/*"
  ]
}' \
https://openam.example.com:8443/openam/json/myrealm/resourcetypes/?_action=create
{
  "uuid": "12345a67-8f0b-123c-45de-6fab78cd01e4",
  "name": "My Resource Type",
  "description": null,
  "patterns": [
    "http://device/location/*"
  ],
  "actions": {
    "RIGHT": true,
    "DOWN": true,
    "UP": true,
    "LEFT": true
  },
  "createdBy": "id=amadmin,ou=user,dc=openam,dc=forgerock,dc=org",
  "creationDate": 1431099940616,
```

```

    "lastModifiedBy": "id=amadmin,ou=user,dc=openam,dc=forgerock,dc=org",
    "lastModifiedDate": 1431099940616
  }

```

2.1.13.3.4. Updating a Resource Type

To update an individual resource type in a realm, perform an HTTP PUT to the `/json{/realm}/resourcetypes` endpoint, and specify the UUID in both the URL and the PUT body. Include a JSON representation of the updated resource type in the PUT data, alongside the UUID.

Note

If the realm is not specified in the URL, OpenAM uses the top level realm.

The `iPlanetDirectoryPro` header is required and should contain the SSO token of an administrative user, such as `amAdmin`, who has access to perform the operation.

Do not use special characters within resource type, policy, or policy set names (for example, "my+resource+type") when using the console or REST endpoints. Using the special characters listed below causes OpenAM to return a 400 Bad Request error. The special characters are: double quotes ("), plus sign (+), comma (,), less than (<), equals (=), greater than (>), backslash (\), forward slash (/), semicolon (;), and null (`\u0000`).

```

$ curl \
--header "iPlanetDirectoryPro: AQIC5..." \
--request PUT \
--header "Content-Type: application/json" \
--data '{
  "uuid": "12345a67-8f0b-123c-45de-6fab78cd01e4",
  "name": "My Updated Resource Type",
  "actions": {
    "LEFT": false,
    "RIGHT": false,
    "UP": false,
    "DOWN": false
  },
  "patterns": [
    "http://device/location/*"
  ]
}' \
https://openam.example.com:8443/openam/json/myrealm/resourcetypes/12345a67-8f0b-123c-45de-6fab78cd01e4
{
  "uuid": "12345a67-8f0b-123c-45de-6fab78cd01e4",
  "name": "My Updated Resource Type",
  "description": null,
  "patterns": [
    "http://device/location/*"
  ],
  "actions": {
    "RIGHT": false,
    "DOWN": false,
    "UP": false,

```

```
    "LEFT": false
  },
  "createdBy": "id=amadmin,ou=user,dc=openam,dc=forgerock,dc=org",
  "creationDate": 1431099940616,
  "lastModifiedBy": "id=amadmin,ou=user,dc=openam,dc=forgerock,dc=org",
  "lastModifiedDate": 1431101016427
}
```

2.1.13.3.5. Deleting a Specific Resource Type

To delete an individual resource types in a realm, perform an HTTP DELETE to the `/json{/realm}/resourcetypes` endpoint, and specify the UUID in the URL.

Note

If the realm is not specified in the URL, OpenAM uses the top level realm.

The `iPlanetDirectoryPro` header is required and should contain the SSO token of an administrative user, such as `amAdmin`, who has access to perform the operation.

```
$ curl \
--request DELETE \
--header "iPlanetDirectoryPro: AQIC5..." \
https://openam.example.com:8443/openam/json/myrealm/resourcetypes/12345a67-8f0b-123c-45de-6fab78cd01e4
{}
```

You can only delete resource types that are not being used by a policy set or policy. Trying to delete a resource type that is in use will return an HTTP 409 Conflict status code, with a message such as:

```
{
  "code": 409,
  "reason": "Conflict",
  "message": "Unable to remove resource type 12345a67-8f0b-123c-45de-6fab78cd01e4 because it is
             referenced in the policy model."
}
```

Remove the resource type from any associated policy sets or policies to be able to delete it.

2.1.13.4. Managing Application Types

Application types act as templates for policy sets, and define how to compare resources and index policies. OpenAM provides a default application type that represents web resources called `iPlanetAMWebAgentService`. OpenAM policy agents use a default policy set that is based on this type, which is also called `iPlanetAMWebAgentService`.

OpenAM provides the `applicationtypes` REST endpoint for the following:

- Section 2.1.13.4.1, "Querying Application Types"

- Section 2.1.13.4.2, "Reading a Specific Application Type"

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": "iPlanetAMWebAgentService",
  "actions": {
    "POST": true,
    "PATCH": true,
    "GET": true,
    "DELETE": true,
    "OPTIONS": true,
    "PUT": true,
    "HEAD": 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"
}
```

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

"name"

The name provided for the application type.

"actions"

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

"resourceComparator"

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

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 type implementation, such as `"com.sun.identity.entitlement.Application"`.

2.1.13.4.1. Querying Application Types

To list all application types, perform an HTTP GET to the `/json/applicationtypes` endpoint, with a `_queryFilter` parameter set to `true`.

The `iPlanetDirectoryPro` header is required and should contain the SSO token of an administrative user, such as `amAdmin`, who has access to perform the operation.

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

Additional query strings can be specified to alter the returned results. For more information, see Section 2.1.7, "Filtering, Sorting, and Paging Results".

2.1.13.4.2. Reading a Specific Application Type

To read an individual application type, perform an HTTP GET to the `/json/applicationtypes` endpoint, and specify the application type name in the URL.

The `iPlanetDirectoryPro` header is required and should contain the SSO token of an administrative user, such as `amAdmin`, who has access to perform the operation.

```
$ curl \
--header "iPlanetDirectoryPro: AQIC5..." \
https://openam.example.com:8443/openam/json/applicationtypes/iPlanetAMWebAgentService
{
  "name": "iPlanetAMWebAgentService",
  "actions": {
    "POST": true,
    "PATCH": true,
    "GET": true,
    "DELETE": true,
    "OPTIONS": true,
    "PUT": true,
    "HEAD": 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"
}
```

2.1.13.5. Managing Policy Sets

This section describes the process of using the OpenAM REST API for managing policy sets.

Policy set definitions set constraints for defining *policies*. The default built-in policy set is called `iPlanetAMWebAgentService`, which OpenAM policy agents use to allow policy management through the console.

For information on creating policy sets by using the OpenAM console, see Section 3.1.1, "OpenAM Resource Types, Policy Sets, and Policies" in the *Administration Guide*.

OpenAM provides the `applications` REST endpoint for the following:

- Section 2.1.13.5.1, "Querying Policy Sets"
- Section 2.1.13.5.2, "Reading a Specific Policy Set"
- Section 2.1.13.5.3, "Creating Policy Sets"
- Section 2.1.13.5.4, "Updating Policy Sets"
- Section 2.1.13.5.5, "Deleting Policy Sets"

Policy sets are realm specific, hence the URI for the policy set API can contain a realm component, such as `/json/{realm}/applications`. If the realm is not specified in the URI, the top level realm is used.

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

```
{
  "creationDate": 1431351677264,
  "lastModifiedDate": 1431351677264,
  "conditions": [
```

```

        "AuthenticateToService",
        "Script",
        "AuthScheme",
        "IPv6",
        "SimpleTime",
        "OAuth2Scope",
        "IPv4",
        "AuthenticateToRealm",
        "OR",
        "AMIdentityMembership",
        "LDAPFilter",
        "AuthLevel",
        "SessionProperty",
        "LEAuthLevel",
        "Session",
        "NOT",
        "AND",
        "ResourceEnvIP"
    ],
    "applicationType": "iPlanetAMWebAgentService",
    "subjects": [
        "JwtClaim",
        "AuthenticatedUsers",
        "Identity",
        "NOT",
        "AND",
        "NONE",
        "OR"
    ],
    "entitlementCombiner": "DenyOverride",
    "saveIndex": null,
    "searchIndex": null,
    "resourceComparator": null,
    "resourceTypeUuids": [
        "12345a67-8f0b-123c-45de-6fab78cd01e4"
    ],
    "attributeNames": [ ],
    "editable": true,
    "createdBy": "id=dsameuser,ou=user,dc=openam,dc=forgerock,dc=org",
    "lastModifiedBy": "id=dsameuser,ou=user,dc=openam,dc=forgerock,dc=org",
    "description": "The built-in Application used by OpenAM Policy Agents.",
    "realm": "/",
    "name": "iPlanetAMWebAgentService"
}

```

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

"conditions"

Condition types allowed in the context of this policy set.

For information on condition types, see Section 2.1.13.6, "Managing Policies" and Section 2.1.13.8, "Managing Environment Condition Types".

"applicationType"

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

"subjects"

Subject types allowed in the context of this policy set.

For information on subject types, see Section 2.1.13.6, "Managing Policies" and Section 2.1.13.9, "Managing Subject Condition Types".

"entitlementCombiner"

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

For more on decision combiners, see Section 2.1.13.11, "Managing 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.

"resourceComparator"

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

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"
```

"resourceTypeUuids"

A list of the UUIDs of the resource types associated with the policy set.

"attributeNames"

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

"description"

String describing the policy set.

"realm"

Name of the realm where this policy set is defined. You must specify the realm in the policy set JSON even though it can be derived from the URL that is used when creating the policy set.

"name"

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

"createdBy"

A string containing the universal identifier DN of the subject that created the policy set.

"creationDate"

An integer containing the creation date and time, in number of seconds since the Unix epoch (1970-01-01T00:00:00Z).

"lastModifiedBy"

A string containing the universal identifier DN of the subject that most recently updated the policy set.

If the policy set has not been modified since it was created, this will be the same value as `createdBy`.

"lastModifiedDate"

An integer containing the last modified date and time, in number of seconds since the Unix epoch (1970-01-01T00:00:00Z).

If the policy set has not been modified since it was created, this will be the same value as `creationDate`.

2.1.13.5.1. Querying Policy Sets

To list all the policy sets in a realm, perform an HTTP GET to the `/json{/realm}/applications` endpoint, with a `_queryFilter` parameter set to `true`.

Note

If the realm is not specified in the URL, OpenAM returns policy sets in the top level realm.

The `iPlanetDirectoryPro` header is required and should contain the SSO token of an administrative user, such as `amAdmin`, who has access to perform the operation.

```
$ curl \
--header "iPlanetDirectoryPro: AQIC5..." \
https://openam.example.com:8443/openam/json/applications?_queryFilter=true
{
  "result": [
    {
      "_id": "iPlanetAMWebAgentService",
      "name": "iPlanetAMWebAgentService",
      "displayName": "Default Policy Set",
      "subjects": [
        "NOT",
        "OR",
        "JwtClaim",
        "AuthenticatedUsers",
        "AND",
        "Identity",
```

```

"NONE"
],
"saveIndex": null,
"searchIndex": null,
"entitlementCombiner": "DenyOverride",
"resourceComparator": null,
"attributeNames": [
],
"lastModifiedBy": "id=dsameuser,ou=user,dc=openam,dc=forgerock,dc=org",
"editable": true,
"resourceTypeUuids": [
    "76656a38-5f8e-401b-83aa-4ccb74ce88d2"
],
"creationDate": 1480651214923,
"lastModifiedDate": 1480651214923,
"createdBy": "id=dsameuser,ou=user,dc=openam,dc=forgerock,dc=org",
"description": "The built-in Application used by OpenAM Policy Agents.",
"applicationType": "iPlanetAMWebAgentService",
"conditions": [
    "LEAuthLevel",
    "Script",
    "AuthenticateToService",
    "SimpleTime",
    "AMIdentityMembership",
    "OR",
    "IPv6",
    "IPv4",
    "SessionProperty",
    "AuthScheme",
    "AuthLevel",
    "NOT",
    "AuthenticateToRealm",
    "AND",
    "ResourceEnvIP",
    "LDAPFilter",
    "OAuth2Scope",
    "Session"
]
},
{
    "_id": "sunAMDelegationService",
    "name": "sunAMDelegationService",
    "displayName": "Delegation Policy Set",
    "subjects": [
        "NOT",
        "OR",
        "AuthenticatedUsers",
        "AND",
        "Identity"
    ],
    "saveIndex": null,
    "searchIndex": null,
    "entitlementCombiner": "DenyOverride",
    "resourceComparator": null,
    "attributeNames": [
    ],
    "lastModifiedBy": "id=dsameuser,ou=user,dc=openam,dc=forgerock,dc=org",
    "editable": true,
    "resourceTypeUuids": [

```

```

    "20a13582-1f32-4f83-905f-f71ff4e2e00d"
  ],
  "creationDate": 1480651214933,
  "lastModifiedDate": 1480651214933,
  "createdBy": "id=dsameuser,ou=user,dc=openam,dc=forgerock,dc=org",
  "description": null,
  "applicationType": "sunAMDelegationService",
  "conditions": [
  ]
}
],
"resultCount": 2,
"pagedResultsCookie": null,
"totalPagedResultsPolicy": "NONE",
"totalPagedResults": -1,
"remainingPagedResults": 0
}

```

Additional query strings can be specified to alter the returned results. For more information, see Section 2.1.7, "Filtering, Sorting, and Paging Results".

Table 2.3. Supported `_queryFilter` Fields and Operators

Field	Supported Operators
<code>name</code>	Equals (<code>eq</code>)
<code>description</code>	Equals (<code>eq</code>)
<code>createdBy</code>	Equals (<code>eq</code>)
<code>creationDate</code>	Equals (<code>eq</code>), Greater than or equal to (<code>ge</code>), Greater than (<code>gt</code>), Less than or equal to (<code>le</code>), Less than (<code>lt</code>) <div style="background-color: #e1f5fe; padding: 10px; margin-top: 10px;"> <p>Note</p> <p>The implementation of <code>eq</code> for this date field does not use regular expression pattern matching.</p> </div>
<code>lastModifiedBy</code>	Equals (<code>eq</code>)
<code>lastModifiedDate</code>	Equals (<code>eq</code>), Greater than or equal to (<code>ge</code>), Greater than (<code>gt</code>), Less than or equal to (<code>le</code>), Less than (<code>lt</code>)

Field	Supported Operators
	<p>Note</p> <p>The implementation of <code>eq</code> for this date field does not use regular expression pattern matching.</p>

2.1.13.5.2. Reading a Specific Policy Set

To read an individual policy set in a realm, perform an HTTP GET to the `/json{/realm}/applications` endpoint, and specify the policy set name in the URL.

Note

If the realm is not specified in the URL, OpenAM uses the top level realm.

The `iPlanetDirectoryPro` header is required and should contain the SSO token of an administrative user, such as `amAdmin`, who has access to perform the operation.

```
$ curl \
--header "iPlanetDirectoryPro: AQIC5..." \
https://openam.example.com:8443/openam/json/applications/mypolicyset
{
  "creationDate": 1431360678810,
  "lastModifiedDate": 1431360678810,
  "conditions": [
    "AuthenticateToService",
    "AuthScheme",
    "IPv6",
    "SimpleTime",
    "OAuth2Scope",
    "IPv4",
    "AuthenticateToRealm",
    "OR",
    "AMIdentityMembership",
    "LDAPFilter",
    "SessionProperty",
    "AuthLevel",
    "LEAuthLevel",
    "Session",
    "NOT",
    "AND",
    "ResourceEnvIP"
  ],
  "applicationType": "iPlanetAMWebAgentService",
  "subjects": [
    "JwtClaim",
    "AuthenticatedUsers",
    "Identity",
    "NOT",
    "AND",
    "OR"
  ]
}
```



```

    ],
    "entitlementCombiner": "DenyOverride",
    "saveIndex": null,
    "searchIndex": null,
    "resourceComparator": "com.sun.identity.entitlement.URLResourceName",
    "resourceTypeUuids": [
      "12345a67-8f0b-123c-45de-6fab78cd01e2"
    ],
    "attributeNames": [ ],
    "editable": true,
    "createdBy": "id=amadmin,ou=user,dc=openam,dc=forgerock,dc=org",
    "lastModifiedBy": "id=amadmin,ou=user,dc=openam,dc=forgerock,dc=org",
    "description": "My example policy set.",
    "realm": "/",
    "name": "mypolicyset"
  }
}

```

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.

2.1.13.5.3. Creating Policy Sets

To create a policy set in a realm, perform an HTTP POST to the `/json{/realm}/applications` endpoint, with an `_action` parameter set to `create`. Include a JSON representation of the policy set in the POST data.

Note

If the realm is not specified in the URL, OpenAM creates the policy set in the top level realm.

The `iPlanetDirectoryPro` header is required and should contain the SSO token of an administrative user, such as `amAdmin`, who has access to perform the operation.

Do not use special characters within resource type, policy, or policy set names (for example, "my+resource+type") when using the console or REST endpoints. Using the special characters listed below causes OpenAM to return a 400 Bad Request error. The special characters are: double quotes ("), plus sign (+), comma (,), less than (<), equals (=), greater than (>), backslash (\), forward slash (/), semicolon (;), and null (`\u0000`).

```

$ curl \
--request POST \
--header "Content-Type: application/json" \
--header "iPlanetDirectoryPro: AQIC5..." \
--data '{
  "name": "mypolicyset",
  "resourceTypeUuids": [
    "12345a67-8f0b-123c-45de-6fab78cd01e2"
  ],
  "realm": "/",
  "conditions": [
    "AND",
    "OR",
    "NOT",

```

```

        "AMIdentityMembership",
        "AuthLevel",
        "AuthScheme",
        "AuthenticateToRealm",
        "AuthenticateToService",
        "IPv4",
        "IPv6",
        "LDAPFilter",
        "LEAuthLevel",
        "OAuth2Scope",
        "ResourceEnvIP",
        "Session",
        "SessionProperty",
        "SimpleTime"
    ],
    "applicationType": "iPlanetAMWebAgentService",
    "description": "My example policy set.",
    "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
{
    "creationDate": 1431360678810,
    "lastModifiedDate": 1431360678810,
    "conditions": [
        "AuthenticateToService",
        "AuthScheme",
        "IPv6",
        "SimpleTime",
        "OAuth2Scope",
        "IPv4",
        "AuthenticateToRealm",
        "OR",
        "AMIdentityMembership",
        "LDAPFilter",
        "SessionProperty",
        "AuthLevel",
        "LEAuthLevel",
        "Session",
        "NOT",
        "AND",
        "ResourceEnvIP"
    ],
    "applicationType": "iPlanetAMWebAgentService",
    "subjects": [
        "JwtClaim",
        "AuthenticatedUsers",
        "Identity",

```

```

    "NOT",
    "AND",
    "OR"
  ],
  "entitlementCombiner": "DenyOverride",
  "saveIndex": null,
  "searchIndex": null,
  "resourceComparator": "com.sun.identity.entitlement.URLResourceName",
  "resourceTypeUuids": [
    "12345a67-8f0b-123c-45de-6fab78cd01e2"
  ],
  "attributeNames": [ ],
  "editable": true,
  "createdBy": "id=amadmin,ou=user,dc=openam,dc=forgerock,dc=org",
  "lastModifiedBy": "id=amadmin,ou=user,dc=openam,dc=forgerock,dc=org",
  "description": "My example policy set.",
  "realm": "/",
  "name": "mypolicyset"
}

```

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.

2.1.13.5.4. Updating Policy Sets

To update an individual policy set in a realm, perform an HTTP PUT to the `/json{/realm}/applications` endpoint, and specify the policy set name in the URL. Include a JSON representation of the updated policy set in the PUT data.

Note

If the realm is not specified in the URL, OpenAM uses the top level realm.

The `iPlanetDirectoryPro` header is required and should contain the SSO token of an administrative user, such as `amAdmin`, who has access to perform the operation.

Do not use special characters within resource type, policy, or policy set names (for example, "my+resource+type") when using the console or REST endpoints. Using the special characters listed below causes OpenAM to return a 400 Bad Request error. The special characters are: double quotes ("), plus sign (+), comma (,), less than (<), equals (=), greater than (>), backslash (\), forward slash (/), semicolon (;), and null (`\u0000`).

```

$ curl \
--request PUT \
--header "iPlanetDirectoryPro: AQIC5..." \
--header "Content-Type: application/json" \
--data '{
  "name": "myupdatedpolicyset",
  "description": "My updated policy set - new name and fewer allowable conditions/subjects.",
  "conditions": [
    "NOT",
    "SimpleTime"
  ]
}'

```

```

    ],
    "subjects": [
      "AND",
      "OR",
      "NOT",
      "AuthenticatedUsers",
      "Identity"
    ],
    "applicationType": "iPlanetAMWebAgentService",
    "entitlementCombiner": "DenyOverride",
    "resourceTypeUuids": [
      "76656a38-5f8e-401b-83aa-4ccb74ce88d2"
    ]
  }' \
https://openam.example.com:8443/openam/json/applications/mypolicyset
{
  "creationDate": 1431362370739,
  "lastModifiedDate": 1431362390817,
  "conditions": [
    "NOT",
    "SimpleTime"
  ],
  "resourceComparator": "com.sun.identity.entitlement.URLResourceName",
  "resourceTypeUuids": [
    "76656a38-5f8e-401b-83aa-4ccb74ce88d2"
  ],
  "createdBy": "id=amadmin,ou=user,dc=openam,dc=forgerock,dc=org",
  "lastModifiedBy": "id=amadmin,ou=user,dc=openam,dc=forgerock,dc=org",
  "applicationType": "iPlanetAMWebAgentService",
  "subjects": [
    "AuthenticatedUsers",
    "Identity",
    "NOT",
    "AND",
    "OR"
  ],
  "entitlementCombiner": "DenyOverride",
  "saveIndex": null,
  "searchIndex": null,
  "attributeNames": [ ],
  "editable": true,
  "description": "My updated policy set - new name and fewer allowable conditions/subjects.",
  "realm": "/",
  "name": "myupdatedpolicyset"
}

```

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.

2.1.13.5.5. Deleting Policy Sets

To delete an individual policy set in a realm, perform an HTTP DELETE to the `/json{/realm}/applications` endpoint, and specify the policy set name in the URL.

Note

If the realm is not specified in the URL, OpenAM uses the top level realm.

The `iPlanetDirectoryPro` header is required and should contain the SSO token of an administrative user, such as `amAdmin`, who has access to perform the operation.

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

2.1.13.6. Managing Policies

This section describes the process of using the OpenAM REST API for managing policies.

For information on creating policies by using the OpenAM console, see Section 3.1.1, "OpenAM Resource Types, Policy Sets, and Policies" in the *Administration Guide*.

OpenAM provides the `policies` REST endpoint for the following:

- Section 2.1.13.6.1, "Querying Policies"
- Section 2.1.13.6.2, "Reading a Specific Policy"
- Section 2.1.13.6.3, "Creating Policies"
- Section 2.1.13.6.4, "Updating Policies"
- Section 2.1.13.6.5, "Deleting Policies"
- Section 2.1.13.6.6, "Copying and Moving Policies"

Policies are realm specific, hence the URI for the policies API can contain a realm component, such as `/json/{realm}/policies`. If the realm is not specified in the URI, the top level realm is used.

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": "mypolicy",
  "active": true,
  "description": "My Policy.",
  "applicationName": "iPlanetAMWebAgentService",
  "actionValues": {
    "POST": true,
    "GET": true
  },
}
```

```
"resources": [
  "http://www.example.com:80/*",
  "http://www.example.com:80/*?*\"",
],
"subject": {
  "type": "AuthenticatedUsers"
},
"condition": {
  "type": "SimpleTime",
  "startTime": "09:00",
  "endTime": "17:00",
  "startDay": "mon",
  "endDay": "fri",
  "enforcementTimeZone": "GMT"
},
"resourceTypeUuid": "76656a38-5f8e-401b-83aa-4ccb74ce88d2",
"resourceAttributes": [
  {
    "type": "User",
    "propertyName": "givenName",
    "propertyValues": [ ]
  }
],
"lastModifiedBy": "id=amadmin,ou=user,dc=openam,dc=forgerock,dc=org",
"lastModifiedDate": "2015-05-11T17:39:09.393Z",
"createdBy": "id=amadmin,ou=user,dc=openam,dc=forgerock,dc=org",
"creationDate": "2015-05-11T17:37:24.556Z"
}
```

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. Must conform to the pattern templates provided by the associated `resource type`.

"applicationName"

String containing the policy set name, such as `iPlanetAMWebAgentService`, or `mypolicyset`.

"actionValues"

Set of string action names, each set to a boolean indicating whether the action is allowed. Chosen from the available actions provided by the associated resource type.

Tip

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"
}
```

Warning

The **AuthenticatedUsers** subject condition does not take into account the realm to which a user authenticated. Any user that has authenticated successfully to any realm passes this subject condition.

To test whether a user has authenticated successfully to a specific realm, also add the **AuthenticateToRealm** environment condition.

- **"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 Section 2.1.13.9, "Managing 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 policy set name, and to set a timeout for 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 stateful or stateless session has been active, and to terminate the session if deemed too old, such that the user must authenticate again. Note that OpenAM terminates stateless sessions only if session blacklisting is in effect. For more information about session blacklisting, see Section 9.4, "Session Termination" in the *Administration Guide*.

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

- **"SessionProperty"** to specify attributes set in the user's stateful or stateless 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 Section 2.1.13.8, "Managing Environment Condition Types".

"resourceTypeUuid"

The UUIDs of the resource type associated with the policy.

"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.

"createdBy"

A string containing the universal identifier DN of the subject that created the policy.

"creationDate"

An integer containing the creation date and time, in number of seconds since the Unix epoch (1970-01-01T00:00:00Z).

"lastModifiedBy"

A string containing the universal identifier DN of the subject that most recently updated the policy.

If the policy has not been modified since it was created, this will be the same value as `createdBy`.

"LastModifiedDate"

An integer containing the last modified date and time, in number of seconds since the Unix epoch (1970-01-01T00:00:00Z).

If the policy has not been modified since it was created, this will be the same value as `creationDate`.

2.1.13.6.1. Querying Policies

Use REST calls to list all the policies in a realm, or to find policies that explicitly apply to a given user or group, by using the procedures below:

- Procedure 2.1, "To List All Policies in a Realm"
- Procedure 2.2, "To Query Policies in a Realm by User or Group"

Procedure 2.1. To List All Policies in a Realm

- To list all the policies in a realm, perform an HTTP GET to the `/json{/realm}/policies` endpoint, with an `_queryFilter` parameter set to `true`.

Note

If the realm is not specified in the URL, OpenAM returns policies in the top level realm.

The `iPlanetDirectoryPro` header is required and should contain the SSO token of an administrative user, such as `amAdmin`, who has access to perform the operation.

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

{
  "result": [
    {
      "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=demo,ou=People,dc=example,dc=com"
        ]
      }
    }
  ]
}
```

```

    ],
    "resourceTypeUuid": "12345a67-8f0b-123c-45de-6fab78cd01e4",
    "lastModifiedBy": "id=amadmin,ou=user,dc=openam,dc=forgerock,dc=org",
    "lastModifiedDate": "2015-05-11T14:48:08.711Z",
    "createdBy": "id=amadmin,ou=user,dc=openam,dc=forgerock,dc=org",
    "creationDate": "2015-05-11T14:48:08.711Z"
  }
],
"resultCount": 1,
"pagedResultsCookie": null,
"remainingPagedResults": 0
}

```

Additional query strings can be specified to alter the returned results. For more information, see Section 2.1.7, "Filtering, Sorting, and Paging Results".

Table 2.4. Supported `_queryFilter` Fields and Operators

Field	Supported Operators
<code>name</code>	Equals (<code>eq</code>)
<code>description</code>	Equals (<code>eq</code>)
<code>applicationName</code>	Equals (<code>eq</code>)
<code>createdBy</code>	Equals (<code>eq</code>)
<code>creationDate</code>	Equals (<code>eq</code>), Greater than or equal to (<code>ge</code>), Greater than (<code>gt</code>), Less than or equal to (<code>le</code>), Less than (<code>lt</code>)
	<p>Note</p> <p>The implementation of <code>eq</code> for this date field does not use regular expression pattern matching.</p>
<code>lastModifiedBy</code>	Equals (<code>eq</code>)
<code>lastModifiedDate</code>	Equals (<code>eq</code>), Greater than or equal to (<code>ge</code>), Greater than (<code>gt</code>), Less than or equal to (<code>le</code>), Less than (<code>lt</code>)

Field	Supported Operators
	<p>Note</p> <p>The implementation of <code>eq</code> for this date field does not use regular expression pattern matching.</p>

Procedure 2.2. To Query Policies in a Realm by User or Group

You can query policies that explicitly reference a given subject by providing the universal ID (UID) of either a user or group. OpenAM returns any policies that explicitly apply to the user or group as part of a subject condition.

Tip

You can obtain the universal ID for a user or group by using REST. See Section 2.1.16.1.2, "Reading Identities".

The following caveats apply to querying policies by user or group:

- Group membership is not considered. For example, querying policies for a specific user will not return policies that only use groups in their subject conditions, even if the user is a member of any of those groups.
- Wildcards are not supported, only exact matches.
- Only policies with a subject condition type of `Identity` are queried—environment conditions are not queried. The `Identity` subject condition type is labelled as *Users & Groups* in the policy editor in the OpenAM console.
- Policies with subject conditions that only contain the user or group in a logical *NOT* operator are not returned.

To query policies by user or group:

- Perform an HTTP GET to the `/json{/realm}/policies` endpoint, with an `_queryId` parameter set to `queryByIdentityUid`, and a `uid` parameter containing the universal ID of the user or group:

```
$ curl \
  --get \
  --header "iPlanetDirectoryPro: AQIC5w..." \
  --data "_queryId=queryByIdentityUid" \
  --data "uid=id=demo,ou=user,o=myrealm,ou=services,dc=openam,dc=forgerock,dc=org" \
  https://openam.example.com:8443/openam/json/myrealm/policies
{
  "result" : [ {
    "name" : "mySubRealmPolicy",
    "active" : true,
```

```

"description" : "",
"resources" : [ "*/:**/*/*?*", "*/:**/*/*" ],
"applicationName" : "iPlanetAMWebAgentService",
"actionValues" : {
  "POST" : true,
  "PATCH" : true,
  "GET" : true,
  "DELETE" : true,
  "OPTIONS" : true,
  "PUT" : true,
  "HEAD" : true
},
"subject" : {
  "type" : "Identity",
  "subjectValues" :
  [
    "id=demo,ou=user,o=myrealm,ou=services,dc=openam,dc=forgerock,dc=org"
  ]
},
"resourceTypeUuid" : "76656a38-5f8e-401b-83aa-4ccb74ce88d2",
"lastModifiedBy" : "id=amAdmin,ou=user,dc=openam,dc=forgerock,dc=org",
"lastModifiedDate" : "2016-05-05T08:45:35.716Z",
"createdBy" : "id=amadmin,ou=user,dc=openam,dc=forgerock,dc=org",
"creationDate" : "2016-05-03T13:45:38.137Z"
} ],
"resultCount" : 1,
"pagedResultsCookie" : null,
"totalPagedResultsPolicy" : "NONE",
"totalPagedResults" : -1,
"remainingPagedResults" : 0
}

```

Note

If the realm is not specified in the URL, OpenAM searches the top level realm.

The `iPlanetDirectoryPro` header is required and should contain the SSO token of an administrative user, such as `amAdmin`, who has access to perform the operation.

2.1.13.6.2. Reading a Specific Policy

To read an individual policy in a realm, perform an HTTP GET to the `/json{/realm}/policies` endpoint, and specify the policy name in the URL.

Note

If the realm is not specified in the URL, OpenAM uses the top level realm.

The `iPlanetDirectoryPro` header is required and should contain the SSO token of an administrative user, such as `amAdmin`, who has access to perform the operation.

```
$ curl \
```



```
--header "iPlanetDirectoryPro: AQIC5..." \
https://openam.example.com:8443/openam/json/policies/example
{
  "result": [
    {
      "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=demo,ou=People,dc=example,dc=com"
        ]
      },
      "resourceTypeUuid": "12345a67-8f0b-123c-45de-6fab78cd01e4",
      "lastModifiedBy": "id=amadmin,ou=user,dc=openam,dc=forgerock,dc=org",
      "lastModifiedDate": "2015-05-11T14:48:08.711Z",
      "createdBy": "id=amadmin,ou=user,dc=openam,dc=forgerock,dc=org",
      "creationDate": "2015-05-11T14:48:08.711Z"
    }
  ],
  "resultCount": 1,
  "pagedResultsCookie": null,
  "remainingPagedResults": 0
}
```

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.

2.1.13.6.3. Creating Policies

To create a policy in a realm, perform an HTTP POST to the `/json{/realm}/policies` endpoint, with an `_action` parameter set to `create`. Include a JSON representation of the policy in the POST data.

Note

If the realm is not specified in the URL, OpenAM uses the top level realm.

The `iPlanetDirectoryPro` header is required and should contain the SSO token of an administrative user, such as `amAdmin`, who has access to perform the operation.

Do not use special characters within resource type, policy, or policy set names (for example, "my+resource+type") when using the console or REST endpoints. Using the special characters listed below causes OpenAM to return a 400 Bad Request error. The special characters are: double quotes

("), plus sign (+), comma (,), less than (<), equals (=), greater than (>), backslash (\), forward slash (/), semicolon (;), and null (\u0000).

```
$ curl \
--request POST \
--header "Content-Type: application/json" \
--header "iPlanetDirectoryPro: AQIC5..." \
--data '{
  "name": "mypolicy",
  "active": true,
  "description": "My Policy.",
  "applicationName": "iPlanetAMWebAgentService",
  "actionValues": {
    "POST": false,
    "GET": true
  },
  "resources": [
    "http://www.example.com:80/*",
    "http://www.example.com:80/*?*\"
  ],
  "subject": {
    "type": "Identity",
    "subjectValues": [
      "uid=demo,ou=People,dc=example,dc=com"
    ]
  },
  "resourceTypeUuid": "12345a67-8f0b-123c-45de-6fab78cd01e4"
}' \
https://openam.example.com:8443/openam/json/policies?_action=create
{
  "name": "mypolicy",
  "active": true,
  "description": "My Policy.",
  "applicationName": "iPlanetAMWebAgentService",
  "actionValues": {
    "POST": false,
    "GET": true
  },
  "resources": [
    "http://www.example.com:80/*",
    "http://www.example.com:80/*?*\"
  ],
  "subject": {
    "type": "Identity",
    "subjectValues": [
      "uid=demo,ou=People,dc=example,dc=com"
    ]
  },
  "resourceTypeUuid": "12345a67-8f0b-123c-45de-6fab78cd01e4",
  "lastModifiedBy": "id=amadmin,ou=user,dc=openam,dc=forgerock,dc=org",
  "lastModifiedDate": "2015-05-11T14:48:08.711Z",
  "createdBy": "id=amadmin,ou=user,dc=openam,dc=forgerock,dc=org",
  "creationDate": "2015-05-11T14:48:08.711Z"
}
```

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.

2.1.13.6.4. Updating Policies

To update an individual policy in a realm, perform an HTTP PUT to the `/json{/realm}/policies` endpoint, and specify the policy name in the URL. Include a JSON representation of the updated policy in the PUT data.

Note

If the realm is not specified in the URL, OpenAM uses the top level realm.

The `iPlanetDirectoryPro` header is required and should contain the SSO token of an administrative user, such as `amAdmin`, who has access to perform the operation.

Do not use special characters within resource type, policy, or policy set names (for example, "my+resource+type") when using the console or REST endpoints. Using the special characters listed below causes OpenAM to return a 400 Bad Request error. The special characters are: double quotes ("), plus sign (+), comma (,), less than (<), equals (=), greater than (>), backslash (\), forward slash (/), semicolon (;), and null (\u0000).

```
$ curl \
--request PUT \
--header "iPlanetDirectoryPro: AQIC5w..." \
--header "Content-Type: application/json" \
--data '{
  "name": "myupdatedpolicy",
  "active": true,
  "description": "My Updated 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",
      "uid=bjenson,ou=People,dc=example,dc=com"
    ]
  },
  "resourceTypeUuid": "12345a67-8f0b-123c-45de-6fab78cd01e4"
}' \
https://openam.example.com:8443/openam/json/policies/mypolicy
{
  "name": "myupdatedpolicy",
  "active": true,
  "description": "My Updated Policy.",
  "applicationName": "iPlanetAMWebAgentService",
  "actionValues": {
    "POST": true,
    "GET": true
  },
}
```

```

"resources": [
  "http://www.example.com:80/*",
  "http://www.example.com:80/*?*?"
],
"subject": {
  "type": "Identity",
  "subjectValues": [
    "uid=bjenson,ou=People,dc=example,dc=com",
    "uid=scarter,ou=People,dc=example,dc=com"
  ]
},
"resourceTypeUuid": "12345a67-8f0b-123c-45de-6fab78cd01e4",
"lastModifiedBy": "id=amadmin,ou=user,dc=openam,dc=forgerock,dc=org",
"lastModifiedDate": "2015-05-11T17:26:59.116Z",
"createdBy": "id=amadmin,ou=user,dc=openam,dc=forgerock,dc=org",
"creationDate": "2015-05-11T17:25:18.632Z"
}
    
```

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.

2.1.13.6.5. Deleting Policies

To delete an individual policy in a realm, perform an HTTP DELETE to the `/json{/realm}/policies` endpoint, and specify the policy name in the URL.

Note

If the realm is not specified in the URL, OpenAM uses the top level realm.

The `iPlanetDirectoryPro` header is required and should contain the SSO token of an administrative user, such as `amAdmin`, who has access to perform the operation.

```

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

2.1.13.6.6. Copying and Moving Policies

You can copy or move an individual policy by performing an HTTP POST to the `/json{/realm}/policies/policyName` endpoint as follows:

- Specify the `_action=copy` or `_action=move` URL parameter.
- Specify the realm in which the input policy resides in the URL. If the realm is not specified in the URL, OpenAM copies or moves a policy from the top level realm.
- Specify the policy to be copied or moved in the URL.

- Specify the SSO token of an administrative user who has access to perform the operation in the `iPlanetDirectoryPro` header.

Specify JSON input data as follows:

Table 2.5. JSON Input Data for Copying or Moving Individual Policies

Object	Property	Description
<code>to</code>	<code>name</code>	The name of the output policy. Required unless you are copying or moving a policy to a different realm and you want the output policy to have the same name as the input policy.
<code>to</code>	<code>application</code>	The policy set in which to place the output policy. Required when copying or moving a policy to a different policy set.
<code>to</code>	<code>realm</code>	The realm in which to place the output policy. If not specified, OpenAM copies or moves the policy within the realm identified in the URL. Required when copying or moving a policy to a different realm.
<code>to</code>	<code>resourceType</code>	The UUID of the output policy's resource type. Required when copying or moving a policy to a different realm.

The follow example copies the policy `myPolicy` to `myNewPolicy`. The output policy is placed in the `myRealm` realm, in the same policy set as the input policy:

```
$ curl \
--request POST \
--header "Content-Type: application/json" \
--header "iPlanetDirectoryPro: AQIC5w..." \
--data '{
  "to": {
    "name": "myNewPolicy"
  }
}' \
https://openam.example.com:8443/openam/json/myRealm/policies/myPolicy?_action=copy
{
  "name": "myNewPolicy",
  "active": true,
  "description": "",
  "applicationName": "iPlanetAMWebAgentService",
  "actionValues": {},
  "resources": [ "*"/*:*/*/*" ],
  "subject": { "type": "NONE" },
  "resourceTypeUuid": "d98e59c9-766a-4934-b5de-8a28a9edc158",
  "lastModifiedBy": "id=amadmin,ou=user,dc=example,dc=com",
  "lastModifiedDate": "2015-12-19T15:22:44.861Z",
  "createdBy": "id=amadmin,ou=user,dc=example,dc=com",
  "creationDate": "2015-12-19T15:22:44.861Z"
}
```

The following example moves a policy named `myPolicy` in the `myRealm` realm to `myMovedPolicy` in the `myOtherRealm` realm. The output policy is placed in the `iPlanetAMWebAgentService` policy set, which is the policy set in which the input policy is located.

The realm `myOtherRealm` must be configured as follows for the example to run successfully:

- It must have a resource type that has the same resources as the resource type configured for the `myPolicy` policy.
- It must have a policy set named `iPlanetAMWebAgentService`.

```
$ curl \
--request POST \
--header "Content-Type: application/json" \
--header "iPlanetDirectoryPro: AQIC5w..." \
--data '{
  "to": {
    "name": "myMovedPolicy",
    "realm": "/myOtherRealm",
    "resourceType": "616b3d02-7a8d-4422-b6a7-174f62afd065"
  }
}' \
https://openam.example.com:8443/openam/json/myRealm/policies/myPolicy?_action=move
{
  "name": "myMovedPolicy",
  "active": true,
  "description": "",
  "actionValues": {},
  "applicationName": "iPlanetAMWebAgentService",
  "resources": [ "*"/*:*/*/*" ],
  "subject": { "type": "NONE" },
  "resourceTypeUuid": "616b3d02-7a8d-4422-b6a7-174f62afd065",
  "lastModifiedBy": "id=amadmin,ou=user,dc=example,dc=com",
  "lastModifiedDate": "2015-12-21T19:32:59.502Z",
  "createdBy": "id=amadmin,ou=user,dc=example,dc=com",
  "creationDate": "2015-12-21T19:32:59.502Z"
}
```

You can also copy and move multiple policies—all the policies in a policy set—in a single operation by performing an HTTP POST to the `/json{/realm}/policies` endpoint as follows:

- Specify the `_action=copy` or `_action=move` URL parameter.
- Specify the realm in which the input policies reside as part of the URL. If no realm is specified in the URL, OpenAM copies or moves policies within the top level realm.
- Specify the SSO token of an administrative user who has access to perform the operation in the `iPlanetDirectoryPro` header.

Specify JSON input data as follows:

Table 2.6. JSON Input Data for Copying or Moving Multiple Policies

Object	Property	Description
<code>from</code>	<code>application</code>	The policy set in which the input policies are located. Required.
<code>to</code>	<code>application</code>	The policy set in which to store output policies. Required when copying or moving policies to a different policy set.

Object	Property	Description
to	realm	The realm in which to store output policies. Required when copying or moving policies to a different realm.
to	namePostfix	A value appended to output policy names in order to prevent name clashes. Required.
resourceTypeMapping	Varies; see Description	One or more resource types mappings, where the left side of the mapping specifies the UUID of a resource type used by the input policies and the right side of the mapping specifies the UUID of a resource type used by the output policies. The two resource types should have the same resource patterns. Required when copying or moving policies to a different realm.

The following example copies all the policies in the `iPlanetAMWebAgentService` policy set in the `myRealm` realm to the `iPlanetAMWebAgentService` policy set in the `myOtherRealm` realm, appending the string `-copy` to the output policy names.

The realm `myOtherRealm` must be configured as follows for the example to run successfully:

- It must have a resource type that maps to the `ccb50c1a-206d-4946-9106-4164e8f2b35b` resource type. The two resource types should have the same resource patterns.
- It must have a policy set named `iPlanetAMWebAgentService`.

The JSON output shows that a single policy is copied. The policy `myNewPolicy` is copied to realm `myOtherRealm`. The copied policy receives the name `myOtherRealm-copy`:

```
$ curl \
--request POST \
--header "Content-Type: application/json" \
--header "iPlanetDirectoryPro: AQC5w..." \
--data '{
  "from": {
    "application": "iPlanetAMWebAgentService"
  },
  "to": {
    "realm": "/myOtherRealm",
    "namePostfix": "-copy"
  },
  "resourceTypeMapping": {
    "ccb50c1a-206d-4946-9106-4164e8f2b35b": "616b3d02-7a8d-4422-b6a7-174f62afd065"
  }
}' \
https://openam.example.com:8443/openam/json/myRealm/policies?_action=copy
```



```
{
  "name": "myNewPolicy-copy",
  "active": true,
  "description": "",
  "actionValues": {},
  "applicationName": "iPlanetAMWebAgentService",
  "resources": [{"*://*:*/"}, {"subject": {"type": "NONE"}},
  "resourceTypeUuid": "616b3d02-7a8d-4422-b6a7-174f62afd065",
  "lastModifiedBy": "id=amadmin,ou=user,dc=example,dc=com",
  "lastModifiedDate": "2015-12-21T20:01:42.410Z",
  "createdBy": "id=amadmin,ou=user,dc=example,dc=com",
  "creationDate": "2015-12-21T20:01:42.410Z"
}
```

2.1.13.7. Importing and Exporting XACML 3.0

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 functionality is identical to that of the `ssoadm create-xacml` and `ssoadm list-xacml` commands. For more information, see Section 3.4, "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:

Table 2.7. 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
xacml rule target	entitlement excluded resource names
Rule Deny Overrides	Rule Combining Algorithm ID
Any of: <ul style="list-style-type: none"> • Entitlement Subject • Resource Names • Policy Set Names • Action Values 	Target

OpenAM Policy	XACML Policy
Any of: <ul style="list-style-type: none"> • Policy Set 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.

2.1.13.7.1. 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:

Table 2.8. Policy Set Mappings

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

Table 2.9. XACML Export Search Filter Format

Element	Description
Attribute Name	The name of the attribute to be searched for. The only permissible values are: <code>application</code> (keyword for policy set), <code>createdby</code> , <code>lastmodifiedby</code> , <code>creationdate</code> , <code>lastmodifieddate</code> , <code>name</code> , <code>description</code> .
Operator	The type of comparison operation to perform. <ul style="list-style-type: none"> • = Equals (text) • < Less Than or Equal To (numerical) • > Greater Than or Equal To (numerical)
Attribute Value	The matching value. Asterisk wildcards are supported.

Procedure 2.3. 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..." \
  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?*?">
    ...
```

Procedure 2.4. 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..." \
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..." \
http://openam.example.com:8080/openam/xacml/policies?filter=createdby=amadmin&
filter=creationdate=135563832
```

2.1.13.7.2. 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

Procedure 2.5. 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 "Content-Type: application/xml" \
--header "iPlanetDirectoryPro: AQIC5..." \
--data @xacml-policy.xml \
http://openam.example.com:8080/openam/xacml/policies?dryrun=true
[
{
  "status": "A",
  "name": "aNewPolicy"
},
{
  "status": "U",
  "name": "anExistingPolicy"
},
{
  "status": "U",
  "name": "anotherExistingPolicy"
}
]
```

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

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

2.1.13.8. Managing Environment Condition Types

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

OpenAM provides the `conditiontypes` REST endpoint for the following:

- Section 2.1.13.8.1, "Querying Environment Condition Types"
- Section 2.1.13.8.2, "Reading a Specific Environment Condition Type"

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`.

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

```
{
  "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.

```
{
  "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.

```
{
  "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": [
    {
      ...
    },
    {
      ...
    },
    ...
  ]
}
```

2.1.13.8.1. Querying Environment Condition Types

To list all environment condition types, perform an HTTP GET to the `/json/conditiontypes` endpoint, with a `_queryFilter` parameter set to `true`.

The `iPlanetDirectoryPro` header is required and should contain the SSO token of an administrative user, such as `amAdmin`, who has access to perform the operation.

```
$ curl \
--header "iPlanetDirectoryPro: AQIC5..." \
https://openam.example.com:8443/openam/json/conditiontypes?_queryFilter=true
{
  "result" : [
    {
      "title": "IPv4",
      "logical": false,
      "config": {
        "type": "object",
        "properties": {
          "startIp": {
            "type": "string"
          },
          "endIp": {
            "type": "string"
          },
          "dnsName": {
            "type": "array",
            "items": {
              "type": "string"
            }
          }
        }
      }
    },
    {
      "title": "NOT",
      "logical": true,
      "config": {
        "type": "object",
        "properties": {
          "condition": {
            "type": "object",
            "properties": { }
          }
        }
      }
    },
    {...},
    {...},
    {...}
  ],
  "resultCount" : 18,
  "pagedResultsCookie" : null,
  "remainingPagedResults" : 0
}
```


Additional query strings can be specified to alter the returned results. For more information, see Section 2.1.7, "Filtering, Sorting, and Paging Results".

2.1.13.8.2. Reading a Specific Environment Condition Type

To read an individual environment condition type, perform an HTTP GET to the `/json/conditiontypes` endpoint, and specify the environment condition type name in the URL.

The `iPlanetDirectoryPro` header is required and should contain the SSO token of an administrative user, such as `amAdmin`, who has access to perform the operation.

```
$ 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"
        }
      }
    }
  }
}
```

2.1.13.9. Managing Subject Condition Types

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

OpenAM provides the `subjecttypes` REST endpoint for the following:

- Section 2.1.13.9.1, "Querying Subject Condition Types"
- Section 2.1.13.9.2, "Reading a Specific Subject Condition Type"

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/subjecttypes`.

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

```
{
  "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.

```
{
  "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": [
    {...},
    {...},
    {...},
    {...}
  ]
}
```

2.1.13.9.1. Querying Subject Condition Types

To list all environment condition types, perform an HTTP GET to the `/json/subjecttypes` endpoint, with a `_queryFilter` parameter set to `true`.

The `iPlanetDirectoryPro` header is required and should contain the SSO token of an administrative user, such as `amAdmin`, who has access to perform the operation.

```
$ curl \
  --header "iPlanetDirectoryPro: AQIC5..." \
  https://openam.example.com:8443/openam/json/subjecttypes?_queryFilter=true
{
  "result" : [
    {
      "title": "JwtClaim",
      "logical": false,
      "config": {
        "type": "object",
        "properties": {
          "claimName": {
            "type": "string"
          },
          "claimValue": {
            "type": "string"
          }
        }
      }
    },
    {
      "title": "NOT",
      "logical": true,
      "config": {
        "type": "object",
        "properties": {
          "subject": {
            "type": "object",
            "properties": { }
          }
        }
      }
    },
    {...},
    {...},
    {...}
  ],
}
```

```
"resultCount" : 5,  
"pagedResultsCookie" : null,  
"remainingPagedResults" : 0  
}
```

Additional query strings can be specified to alter the returned results. For more information, see Section 2.1.1.7, "Filtering, Sorting, and Paging Results".

2.1.13.9.2. Reading a Specific Subject Condition Type

To read an individual subject condition type, perform an HTTP GET to the `/json/subjecttypes` endpoint, and specify the subject condition type name in the URL.

The `iPlanetDirectoryPro` header is required and should contain the SSO token of an administrative user, such as `amAdmin`, who has access to perform the operation.

```
$ 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"  
        }  
      }  
    }  
  }  
}
```

2.1.13.10. Managing 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.

OpenAM provides the `subjectattributes` REST endpoint for the following:

- Section 2.1.13.10.1, "Querying Subject Attributes"

Subject attributes derive from the list of LDAP user attributes configured for the Identity data store. For more information, see Section 4.3, "Configuring Data Stores" in the *Administration Guide*.

2.1.13.10.1. Querying Subject Attributes

To list all subject attributes, perform an HTTP GET to the `/json/subjectattributes` endpoint, with a `_queryFilter` parameter set to `true`.

The `iPlanetDirectoryPro` header is required and should contain the SSO token of an administrative user, such as `amAdmin`, who has access to perform the operation.

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

{
  "result" : [
    "sunIdentityServerPPInformalName",
    "sunIdentityServerPPFacadeGreetSound",
    "uid",
    "manager",
    "sunIdentityServerPPCommonNameMN",
    "sunIdentityServerPPLegalIdentityGender",
    "preferredLocale",
    "...",
    "...",
    "...",
    "...",
  ],
  "resultCount": 87,
  "pagedResultsCookie": null,
  "remainingPagedResults": 0
}
```

Note that no pagination cookie is set and the subject attribute names are all returned as part of the "result" array.

2.1.13.11. Managing Decision Combiners

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

OpenAM provides the `decisioncombiners` REST endpoint for the following:

- Section 2.1.13.11.1, "Querying Decision Combiners"
- Section 2.1.13.11.2, "Reading a Specific Decision Combiner"

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`.

2.1.13.11.1. Querying Decision Combiners

To list all decision combiners, perform an HTTP GET to the `/json/decisioncombiners` endpoint, with a `_queryFilter` parameter set to `true`.

The `iPlanetDirectoryPro` header is required and should contain the SSO token of an administrative user, such as `amAdmin`, who has access to perform the operation.

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

Additional query strings can be specified to alter the returned results. For more information, see Section 2.1.7, "Filtering, Sorting, and Paging Results".

2.1.13.11.2. Reading a Specific Decision Combiner

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

To read an individual decision combiner, perform an HTTP GET to the `/json/decisioncombiners` endpoint, and specify the decision combiner name in the URL.

The `iPlanetDirectoryPro` header is required and should contain the SSO token of an administrative user, such as `amAdmin`, who has access to perform the operation.

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

2.1.14. RESTful OAuth 2.0, OpenID Connect 1.0 and UMA 1.0 Services

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

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

2.1.14.1. OAuth 2.0

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 additional `tokeninfo` and `introspect` endpoints useful to resource servers and clients.

- An endpoint for reading OAuth 2.0 resource sets. This is specific to OpenAM.
- 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 Section 2.1.5, "Authentication and Logout", whereby OpenAM returns a `token.id` value. Clients pass the `token.id` value in a header named after the authentication cookie, by default `iplanetDirectoryPro`.

2.1.14.1.1. OAuth 2.0 Client and Resource Server Endpoints

OpenAM exposes REST endpoints for making calls to OpenAM acting as an authorization server, as described in Chapter 13, "*Managing OAuth 2.0 Authorization*" in the *Administration Guide*.

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, and an introspection endpoint to retrieve metadata about a token, such as approved scopes and the context in which the token was issued. 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 you created after OAuth 2.0 authorization server configuration, which grants all authenticated users access.

The following is an example URL for obtaining consent:

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

After logging in, the URL above presents the OAuth 2.0 consent screen, similar to the following:

Figure 2.1. OAuth 2.0 Consent Screen

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 Section 1.4.11, "OAuth2 Provider" 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.

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 Section 2.8, "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*.
- `code_challenge`. Use this parameter when *Proof Key for Code Exchange* (PKCE) support is enabled in the OAuth2 Provider service. To configure it, navigate to *Realms > Realm Name > Services > OAuth2 Provider > Advanced* and enable the *Code Verifier Parameter Required*

property. For more information about the PKCE support, see Proof Key for Code Exchange by OAuth Public Clients - *RFC 7636*.

`/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

The `/oauth2/access_token` endpoint can take an additional parameter, `auth_chain=authentication-chain`, which allows client to specify the authentication chain to use for Password Grant Type.

The following example shows how a client can specify the authentication chain, `myAuthChain`:

```
$ curl \
--request POST
\
--user "myClientID:password"
\
--data "grant_type=password&username=amadmin&password=cangetinam&scope=profile&auth_chain=myAuthChain"
\
https://openam.example.com:8443/openam/oauth2/access_token
```

The `/oauth2/access_token` endpoint can take additional parameters. In particular, 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/access_token`.

`/oauth2/device`

Device flow endpoint as defined by the Internet-Draft OAuth 2.0 Device Flow, used by a client device to obtain a device code or an access token.

Example: <https://openam.example.com:8443/openam/oauth2/device/code>

For more information, see Section 2.1.14.1.2, "Using Endpoints for OAuth 2.0 Device Flow".

`/oauth2/token/revoke`

When a user logs out of an application, the application revokes any OAuth 2.0 tokens (access and refresh tokens) that are associated with the user. The client can also revoke a token without the need of an `SSOToken` by sending a request to the `/oauth2/token/revoke` endpoint as follows:

```
$ curl \
--request POST
\
--data "token=d06ab31e-9cdb-403e-855f-bd77652add84"
\
--data "client_id=MyClientID"
\
--data "client_secret=password" \
https://openam.example.com:8443/openam/oauth2/token/revoke
```

If you are revoking an access token, then that token will be revoked. If you are revoking a refresh token, then both the refresh token and any other associated access tokens will also be revoked. *Associated access tokens* means that any other tokens that have come out of the same authorization grant will also be revoked. For cases where a client has multiple access tokens for a single user that were obtained via different authorization grants, then the client will have to make multiple calls to the `/oauth2/token/revoke` endpoint to invalidate each token.

`/oauth2/tokeninfo`

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

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.

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 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",
  "grant_type": "password",
  "scope": [
    "mail",
    "cn"
  ],
  "cn": "demo",
  "realm": "/",
}
```

```
"cnf": {
  "jwk": {
    "alg": "RS512",
    "e": "AQAB",
    "n": "k7qLlj...G2oucQ",
    "kty": "RSA",
    "use": "sig",
    "kid": "myJWK"
  }
}
"token_type": "Bearer",
"expires_in": 577,
"client_id": "MyClientID",
"access_token": "f9063e26-3a29-41ec-86de-1d0d68aa85e9"
}
```

Note

Running a GET method to the `/oauth2/tokeninfo` endpoint as shown in the previous example writes the token ID to the access log. To not expose the token ID in the logs, send the OAuth 2.0 access token as part of the authorization bearer header:

```
$ curl \
--request GET \
--header "Authorization Bearer aec6b050-b0a4-4ece-a86f-bd131decbb9c" \
"https://openam.example.com:8443/openam/oauth2/tokeninfo"
```

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 Section 13.1.3, "Using Your Own Client and Resource Server" 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`.

`/oauth2/introspect`

Endpoint defined in `draft-ietf-oauth-introspection-04`, used to retrieve metadata about a token, such as approved scopes and the context in which the token was issued.

Given an access token, a client can perform an HTTP POST on `/oauth2/introspect?token=access_token` to retrieve a JSON object indicating the following:

`active`

Is the token active.

`scope`

A space-separated list of the scopes associated with the token.

client_id

Client identifier of the client that requested the token.

user_id

The user who authorized the token.

token_type

The type of token.

exp

When the token expires, in seconds since January 1 1970 UTC.

sub

Subject of the token.

iss

Issuer of the token.

The `/oauth2/introspect` endpoint requires authentication, and supports basic authorization (a base64-encoded string of `client_id:client_secret`), `client_id` and `client_secret` passed as header values, or a JWT bearer token.

The following example demonstrates the `/oauth2/introspect` endpoint with basic authorization:

```
$ curl \
--request POST \
--header "Authorization: Basic ZGVtbzpjajGFuZ2VpdA==" \
https://openam.example.com:8443/openam/oauth2/introspect \
?token=f9063e26-3a29-41ec-86de-1d0d68aa85e9
{
  "active": true,
  "scope": "mail cn",
  "client_id": "my0Auth2Client",
  "user_id": "demo",
  "token_type": "Bearer",
  "exp": 1419356238,
  "sub": "https://resources.example.com/",
  "iss": "https://openam.example.com/"
}
```

Note

Running a POST method to the `/oauth2/introspect` endpoint as shown in the previous example writes the token ID to the access log. To hide the token ID in the logs, send the OAuth 2.0 access token as part of the POST body:

```
$ curl \
--request POST \
--header "Authorization Basic ZGVtbzpjagFuZ2VpdA==" \
--data "token=f9063e26-3a29-41ec-86de-1d0d68aa85e9" \
"https://openam.example.com:8443/openam/oauth2/introspect"
```

2.1.14.1.2. Using Endpoints for OAuth 2.0 Device Flow

If a client device has a limited user interface, it can obtain an OAuth 2.0 device code and ask a user to authorize the client on a more full-featured user agent, such as an Internet browser.

OpenAM provides the `/oauth2/device/code`, `/oauth2/device/user`, and `/oauth2/access_token` endpoints to support the OAuth 2.0 Device Flow.

The following procedures show how to use the OAuth 2.0 device flow endpoints:

- Procedure 2.6, "To Request a User Code in the OAuth 2.0 Device Flow".
- Procedure 2.7, "To Grant Consent in the OAuth 2.0 Device Flow".
- Procedure 2.8, "To Poll for Authorization in the OAuth 2.0 Device Flow".

Note

In the examples `nonce` and `state` OAuth 2.0 parameters are omitted, but should be used in production.

Procedure 2.6. To Request a User Code in the OAuth 2.0 Device Flow

Devices can display a user code and instructions to a user, which can be used on a separate client to provide consent, allowing the device to access resources.

As user codes may be displayed on lower resolution devices, the list of possible characters used has been optimized to reduce ambiguity. User codes consist of a random selection of eight of the following characters:

```
234567ABCDEFGHJKLMNPQRSTUVWXYZabcdefghijklmnopqrstvwxyz
```

To request a user code in the OAuth 2.0 device flow:

1. Ensure that an OAuth 2.0/OpenID Connect client Agent profile is configured in OpenAM, as described in Section 5.8, "Configuring OAuth 2.0 and OpenID Connect 1.0 Clients" in the *Administration Guide*.
2. Create a POST request to the `/oauth2/device/code` endpoint to acquire a device code. The following URL parameters are required:

`response_type`

Specifies the response type required by the request. Must be set to `token`.

scope

Specifies the list of scopes requested by the client, separated by URL-encoded space characters.

client_id

Specifies the name of the client agent profile in OpenAM.

```
$ curl \
--data response_type=token \
--data scope=phone%20email%20profile%20address \
--data client_id=myDeviceAgentProfile \
http://openam.example.com:8080/openam/oauth2/device/code
{
  "interval": 5,
  "device_code": "7a95a0a4-6f13-42e3-ac3e-d3d159c94c55",
  "verification_url": "http://openam.example.com:8080/openam/oauth2/device/user",
  "user_code": "VAL12e0v",
  "expires_in": 300
}
```

On success, OpenAM returns a verification URL, and a user code to enter at that URL. OpenAM also returns an interval, in seconds, that the client device must wait for in between requests for an access token.

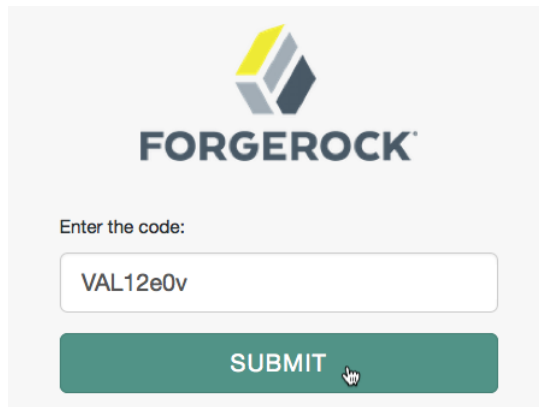
3. The client device should now provide instructions to the user to enter the user code and grant access to the OAuth 2.0 device. The client may choose an appropriate method to convey the instructions, for example text instructions on screen, or a QR code. See Procedure 2.7, "To Grant Consent in the OAuth 2.0 Device Flow".
4. The client device should also begin polling the authorization server for the access token, once consent has been given. See Procedure 2.8, "To Poll for Authorization in the OAuth 2.0 Device Flow".

Procedure 2.7. To Grant Consent in the OAuth 2.0 Device Flow

OAuth 2.0 device flow requires that the user grants consent to allow the client device to access resources.

- You can grant consent in the OAuth 2.0 device flow using the OpenAM user interface, or by making calls to OpenAM endpoints.
 - To use the OpenAM user interface, the user should visit the verification URL in a web browser and enter the user code:

Figure 2.2. OAuth 2.0 User Code



FORGEROCK

Enter the code:

SUBMIT

The user can then authorize the device flow client by allowing the requested scopes:

Figure 2.3. OAuth 2.0 Consent Page

FORGEROCK

DEVICE FLOW CLIENT
An example OAuth Device Flow Client.

This application is requesting the following private information:

Your postal address ^

Address: 60 Queen Square, Bristol, BS1 4JZ

Your email address v

Your profile information ^

First name: Demo
Last name: User
Full name: Demo User

You are signed in as: Demo User

Save Consent

Deny Allow

- To use endpoint calls, create a POST request to the `/oauth2/device/user` endpoint. The following URL parameter is required:

user_code

The user code as provided by the `/oauth2/device/code` endpoint.

The form data should be in `x-www-form-urlencoded` format, and contain the following fields:

user_code

The user code as provided by the `/oauth2/device/code` endpoint.

scope

Specifies the list of scopes consented to by the user, separated by URL-encoded space characters.

client_id

Specifies the name of the client agent profile in OpenAM.

response_type

Must be `token`.

decision

To allow client access, specify `allow`. Any other value will deny consent.

csrf

Duplicates the contents of the `iPlanetDirectoryPro` cookie, which contains the SSO token of the user granting access.

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

The `iPlanetDirectoryPro` cookie is required and should contain the SSO token of the user granting access to the client.

```
$ curl \
-X POST \
--header "Cookie: iPlanetDirectoryPro=AQIC5..." \
--header "Content-Type: application/x-www-form-urlencoded" \
--data scope=phone%20email%20profile%20address \
--data user_code=VAL12e0v \
--data response_type=token \
--data client_id=myDeviceAgentProfile \
--data decision=allow \
--data csrf=AQIC5... \
http://openam.example.com:8080/openam/oauth2/device/user?user_code=VAL12e0v
```

OpenAM returns HTML containing a JavaScript fragment named `pageData`, with details of the result.

Successfully allowing or denying access returns:

```
pageData = {
  locale: "en-us",
  baseUrl : "http://openam.example.com:8080/openam/XUI",
  realm : "//XUI",
  done: true
};
```

If the supplied user code has already been used, or is incorrect, the following is returned:

```
pageData = {
  locale: "*",
  errorCode: "not_found",
  realm: "/",
  baseUrl: "http://openam.example.com:8080/openam/XUI"
};
```

If the user gives consent, OpenAM adds the OAuth 2.0 client to the user's profile page in the *Authorized Apps* section. For more information, see Section 13.4.3, "User Consent Management" in the *Administration Guide*.

Important

As per Section 4.1.1 of the OAuth 2.0 authorization framework, it is required that the authorization server legitimately obtains an authorization decision from the resource owner.

Any client using the endpoints to register consent is responsible for ensuring this requirement, OpenAM cannot assert that consent was given in these cases.

Procedure 2.8. To Poll for Authorization in the OAuth 2.0 Device Flow

- On the client device, create a POST request to poll the `/oauth2/access_token` endpoint to request an access token. Include the client ID, client secret, and the device code as query parameters in the request. You must also specify a `grant_type` of `http://oauth.net/grant_type/device/1.0`.

The client device must wait for the number of seconds previously provided as the value of `interval` between polling OpenAM for an access token.

```
$ curl \
--data client_id=myDeviceAgentProfile \
--data client_secret=password \
--data grant_type=http://oauth.net/grant_type/device/1.0 \
--data code=7a95a0a4-6f13-42e3-ac3e-d3d159c94c55 \
http://openam.example.com:8080/openam/oauth2/access_token
{
  "scope": "phone email address profile",
  "code": "20c1fc0c-3153-4a11-8d1f-d815c1a522b5"
}
```

If the user has authorized the client device, an HTTP 200 status code is returned, with an access token that can be used to request resources.

```
{
  "expires_in": 3599,
  "token_type": "Bearer",
  "access_token": "c1e9c8a4-6a6c-45b2-919c-335f2cec5a40"
}
```

If the user has not yet authorized the client device, an HTTP 403 status code is returned, with the following error message:

```
{
  "error": "authorization_pending",
  "error_description": "The user has not yet completed authorization"
}
```

If the client device is polling faster than the specified interval, an HTTP 400 status code is returned, with the following error message:

```
{
  "error": "slow_down",
  "error_description": "The polling interval has not elapsed since the last request"
}
```

2.1.14.1.3. OAuth 2.0 Resource Set Endpoint

OpenAM provides a read-only REST endpoint for viewing a resource set registered to a particular user. The endpoint is `/json{/realm}/users/user/oauth2/resourcesets/resource_set_ID`, so for example <https://openam.example.com:8443/openam/json/myrealm/users/demo/oauth2/resourcesets/43225628-4c5b-4206-b7cc-5164da81decd0>.

To read a resource set, either the resource set owner or an administrator such as `amadmin` must have logged in to OpenAM (the authorization server) and have been issued an SSO token.

Procedure 2.9. To Read an OAuth 2.0 Resource Set

- Create a GET request to the `resourcesets` endpoint, including the SSO token in a header based on the configured session cookie name (for example: `iPlanetDirectoryPro`), and with the resource set ID in the URL.

The following example uses an SSO token acquired by the `amadmin` user to view a resource set, and related policy, belonging to the `demo` user in the top level realm:

```
$ curl \
--header "iPlanetDirectoryPro: AQIC5wM2LY4Sfcxs...EwNDU2NjE0*" \
https://openam.example.com:8443/openam/json/users/demo
\
/oauth2/resourcesets/43225628-4c5b-4206-b7cc-5164da81decd0
{
  "scopes": [
    "http://photoz.example.com/dev/scopes/view",
    "http://photoz.example.com/dev/scopes/comment"
  ],
  "_id": "43225628-4c5b-4206-b7cc-5164da81decd0",
  "resourceServer": "UMA-Resource-Server",
  "name": "My Videos",
  "icon_uri": "http://www.example.com/icons/cinema.png",
  "policy": {
    "permissions": [
      {
        "subject": "user.1",
        "scopes": [
          "http://photoz.example.com/dev/scopes/view"
        ]
      }
    ]
  }
}
```

```
    ],
    {
      "subject": "user.2",
      "scopes": [
        "http://photoz.example.com/dev/scopes/comment",
        "http://photoz.example.com/dev/scopes/view"
      ]
    }
  ],
  "type": "http://www.example.com/rsets/videos"
}
```

Tip

You can specify the fields that are returned with the `_fields` query string filter. For example `?_fields=scopes, resourceServer, name`

On success, an HTTP 200 OK status code is returned, with a JSON body representing the resource set. If a policy relating to the resource set exists, a representation of the policy is also returned in the JSON.

If the specified resource set does not exist, an HTTP 404 Not Found status code is returned, as follows:

```
{
  "code": 404,
  "reason": "Not Found",
  "message": "No resource set with id, bad-id-3e28-4c19-8a2b-36fc24c899df0, found."
}
```

If the SSO token used is not that of the resource set owner or an administrator, an HTTP 403 Forbidden status code is returned, as follows:

```
{
  "code": 403,
  "reason": "Forbidden",
  "message": "User, user.1, not authorized."
}
```

2.1.14.1.4. 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 \
  --request GET \
  --header "iplanetDirectoryPro: AQIC5wM2LY4Sfcxs...EwNDU2NjE0*" \
  https://openam.example.com:8443/openam/frrest/oauth2/token/f5fb4833-ba3d-41c8-bba4-833b49c3fe2c
{
  "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=access_token` to request the list of access tokens for the current user.

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

```
$ curl \
--request GET \
--header "iplanetDirectoryPro: AQIC5wM2LY4Sfcw..." \
https://openam.example.com:8443/openam/frrest/oauth2/token/?_queryId=access_token
{
  "_rev": "1753454107",
  "tokenName": [
    "access_token"
  ],
  "expireTime": "Indefinitely",
  "scope": [
    "openid"
  ],
  "grant_type": [
    "password"
  ],
  "clientID": [
    "myOAuth2Client"
  ],
  "tokenType": [
    "Bearer"
  ],
  "redirectURI": [],
  "nonce": [],
  "realm": [
    "/test"
  ],
  "userName": [
    "user.4"
  ],
  "display_name": "",
  "scopes": "openid"
},
{
  "_rev": "1753454107",
  "tokenName": [
    "access_token"
  ],
  "expireTime": "Indefinitely",
  "scope": [
    "openid"
  ],
  "grant_type": [
    "password"
  ],
  "clientID": [
    "myOAuth2Client"
  ],
  "tokenType": [
    "Bearer"
  ],
  "redirectURI": [],
  "nonce": [],
```

```

"realm": [
  "/test"
],
"userName": [
  "user.4"
],
"display_name": "",
"scopes": "openid"
}
},
"resultCount": 2,
"pagedResultsCookie": null,
"totalPagedResultsPolicy": "NONE",
"totalPagedResults": -1,
"remainingPagedResults": -1
}
    
```

To list a specific user's tokens, perform an HTTP GET on `/ffrest/oauth2/token/?_queryId=userName=string`, where *string* is the user, such as `user.4`. You must use an `amadmin` token with this GET method. Delegated admins are not supported here.

```

$ curl \
--request GET \
--header "iplanetDirectoryPro: AQIC5wM2LY4Sfcxs...EwNDU2NjE0*" \
https://openam.example.com:8443/openam/frrest/oauth2/token/?_queryId=userName=user.4
{
  "result": [
    {
      "_id": "2aaddde8-586b-4cb7-b431-eb86af57aabc",
      "_rev": "-549186065",
      "tokenName": [
        "access_token"
      ],
      "expireTime": "Indefinitely",
      "scope": [
        "openid"
      ],
      "grant_type": [
        "password"
      ],
      "authGrantId": [
        "50e9f80b-d193-4aeb-93e9-e383ea2cabd3"
      ],
      "clientID": [
        "myOAuth2Client"
      ],
      "parent": [],
      "refreshToken": [
        "5e1423a2-d2cd-40d5-8f54-5b695836cd44"
      ],
      "id": [
        "2aaddde8-586b-4cb7-b431-eb86af57aabc"
      ],
      "tokenType": [
        "Bearer"
      ]
    },
  ],
}
    
```

```

"auditTrackingId": [
  "6ac90d13-9cac-444b-bfbc-c7aca16713de-777"
],
"redirectURI": [],
"nonce": [],
"realm": [
  "/test"
],
"user_name": [
  "user.4"
],
"display_name": "",
"scopes": "openid"
},
{
  "_id": "5e1423a2-d2cd-40d5-8f54-5b695836cd44",
  "_rev": "1171292923",
  "tokenName": [
    "refresh_token"
  ],
  "expireTime": "Oct 18, 2016 10:51 AM",
  "scope": [
    "openid"
  ],
  "grant_type": [
    "password"
  ],
  "authGrantId": [
    "50e9f80b-d193-4aeb-93e9-e383ea2cabd3"
  ],
  "clientId": [
    "myOAuth2Client"
  ],
  "authModules": [],
  "id": [
    "5e1423a2-d2cd-40d5-8f54-5b695836cd44"
  ],
  "tokenType": [
    "Bearer"
  ],
  "auditTrackingId": [
    "6ac90d13-9cac-444b-bfbc-c7aca16713de-776"
  ],
  "redirectURI": [],
  "realm": [
    "/test"
  ],
  "user_name": [
    "user.4"
  ],
  "acr": [],
  "display_name": "",
  "scopes": "openid"
},
],
"resultCount": 2,
"pagedResultsCookie": null,
"totalPagedResultsPolicy": "NONE",
"totalPagedResults": -1,

```



```
"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"
}
```

2.1.14.1.5. 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 "Content-Type: application/json" \
--header "iplanetDirectoryPro: AQIC5wM...3MTYxOA..*" \
--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 *locale* 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"}
```

2.1.14.2. 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 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.
- Endpoint for validating OpenID Connect 1.0 ID Tokens

2.1.14.2.1. Endpoints for 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:

- `/oauth2/.well-known/openid-configuration` allows clients to retrieve OpenID Provider configuration by HTTP GET as specified by OpenID Connect Discovery 1.0.
- `/oauth2/.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*.

Note

OpenAM supports a provider service that allows the realm to have a configured option for obtaining the base URL (including protocol) for components that need to return a URL to the client. This service is used to provide the URL base that is used in the `.well-known` endpoints used in OpenID Connect 1.0 and UMA.

For more information, see Section 14.4, "Configuring the Base URL Source Service" in the *Administration Guide*.

2.1.14.2.2. Endpoints for 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 Procedure 14.4, "To Register a Relying Party Dynamically" in the *Administration Guide* for details.

2.1.14.2.3. Endpoints for 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 Section 2.1.14.1.1, "OAuth 2.0 Client and 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 Section 14.7, "Relying Party Examples" in the *Administration Guide*.

2.1.14.2.4. Endpoints for 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 Section 14.6, "Managing OpenID Connect User Sessions" in the *Administration Guide*.

2.1.14.2.5. Endpoint for Validating OpenID Connect 1.0 ID Tokens

Clients can use an OpenID Connect 1.0 endpoint on OpenAM to quickly validate a *stateless* OIDC ID token and optionally retrieve any claims within the token. The endpoint is used globally and not within a realm. For information on configuring stateless OIDC tokens, see Section 2.1.14.2.6, "Configuring Stateless OpenID Connect 1.0 Tokens".

- `/openam/oauth2/idthokeninfo`

Note

The endpoint does not support the validation of encrypted OIDC ID tokens.

The endpoint validates an OIDC ID token based on rules 1-10 in section 3.1.3.7 of the OpenID Connect Core and runs the following steps:

1. Extracts the first `aud` (audience) claim from the ID token. The `client_id`, which is passed in as authentication of the request, will be used as the client and validated against the `aud` claim.
2. Extracts the `realm` claim, if present, default to the root realm if the token was not issued by OpenAM.
3. Looks up the client in the given realm, producing an error if it does not exist.
4. Verifies the signature of the ID token, according to the settings for the client (ID token signed response algorithm, public key selector).
5. Verifies the `issuer`, `audience`, `expiry`, `not-before`, and `issued-at` claims as per the specification.

To invoke the endpoint, the client sends an HTTP POST request to `/openam/oauth2/idthokeninfo` using the following parameters in the POST body in application/x-www-form-urlencoded format or as query parameters:

- `id_token` - OIDC ID token to validate (required)
- `claims` - optional comma-separated list of claims to return from the ID token

For example, you can run the following command:

```
curl -X POST -d "id_token=$IDTOKEN" http://openam.example.com:8080/openam/oauth2/idthokeninfo
```

where \$IDTOKEN is an OIDC ID token

If the ID token validates successfully, the endpoint unpacks the claims from the ID token and returns them as JSON. You can also use an optional `claims` parameter in the request to return those specific claims. If a claim is requested that does not exist, no error occurs; it will simply not be present in the response.

For example, you can run the following command to retrieve the claims in an OIDC ID token:

```
curl -i -X POST -d "id_token=$IDTOKEN" \
'http://openam.example.com:8080/openam/oauth2/idthokeninfo?claims=sub,exp,realm'
HTTP/1.1 200 OK
Date: Wed, 01 Jun 2016 07:31:39 GMT
Accept-Ranges: bytes
Server: Restlet-Framework/2.3.4
Vary: Accept-Charset, Accept-Encoding, Accept-Language, Accept
Content-Type: application/json;charset=UTF-8
Content-Length: 50
{
  "sub": "demo",
  "exp": 1461065147,
  "realm": "/"
}
```

For invalid requests, the endpoint returns a 400 HTTP code with a JSON error response:

```
curl -i -X POST 'http://openam.example.com:8080/openam/oauth2/idthokeninfo?claims=sub,exp,realm'
HTTP/1.1 400 Bad Request
Date: Wed, 01 Jun 2016 08:32:43 GMT
Accept-Ranges: bytes
Server: Restlet-Framework/2.3.4
Vary: Accept-Charset, Accept-Encoding, Accept-Language, Accept
Content-Type: application/json
Transfer-Encoding: chunked
Connection: close
{
  "error": "bad_request",
  "error_description": "no id_token in request"
}
```

2.1.14.2.6. Configuring Stateless OpenID Connect 1.0 Tokens

OpenAM supports *stateless* access, refresh, and ID tokens for OpenID Connect 1.0 (OIDC). Stateless tokens allow clients to directly validate the tokens by storing session information within the token itself and bypassing storage in an external CTS data store. This feature also allows any OpenAM instance in the issuing cluster to validate an OIDC tokens without cross-talk.

For configuration procedures, see Section 14.9, "Stateless OpenID Connect 1.0 Access and Refresh Tokens" in the *Administration Guide*.

2.1.14.3. User-Managed Access (UMA)

User-Managed Access (UMA) is a profile of OAuth 2.0 that lets resource owners control access to protected resources on any number of resource servers from arbitrary requesting parties.

OpenAM acts as the centralized authorization server and governs access using policies created by resource owners.

OpenAM exposes the following REST endpoints for User-Managed Access purposes:

- An endpoint for automatic configuration and registration of the Authorization Server.
- An endpoint for registering sets of resources.
- An endpoint for managing UMA policies.
- Endpoints for requesting and granting authorization for access to resources.

2.1.14.3.1. Discovering UMA Configuration

OpenAM exposes an endpoint for discovering information about the UMA provider configuration.

A resource server or client can perform an HTTP GET on `/uma{/realm}/.well-known/uma-configuration` to retrieve a JSON object indicating the UMA configuration.

For an example, see [Configuring OpenAM For UMA Discovery](#) in the *Administration Guide*.

2.1.14.3.2. Managing UMA Resource Sets

UMA uses the *OAuth 2.0 Resource Set Registration standard* for registration and management of resources. The endpoint is `/oauth2/resource_set/`. For details, see Section 15.4, "Managing UMA Resource Sets" in the *Administration Guide*.

OpenAM also provides a read-only endpoint for viewing a user's resource sets, and if available policy definitions. For more information, see Section 2.1.14.1.3, "OAuth 2.0 Resource Set Endpoint".

2.1.14.3.3. Managing UMA Policies

OpenAM exposes the following endpoint for managing UMA policies:

`/json/users/username/uma/policies`

For managing UMA policies. For details, see Section 15.6, "Managing UMA Policies" in the *Administration Guide*.

2.1.14.3.4. Accessing UMA Protected Resources

OpenAM exposes the following endpoints for managing UMA workflow and accessing protected resources:

`/uma/permission_request`

For registering permission requests. For more information, see Procedure 15.22, "To Register an UMA Permission Request" in the *Administration Guide*.

`/uma/authz_request`

For acquiring requesting party tokens. For more information, see Procedure 15.24, "To Acquire a Requesting Party Token" in the *Administration Guide*.

2.1.15. RESTful User Self-Service

This section shows how to use the OpenAM RESTful interfaces for user self-service functionality: user self-registration, forgotten password reset, forgotten username retrieval, dashboard configuration, and device profile reset.

The steps to perform user self-service via the REST APIs varies depending on the configured user self-service process flow. For more information, see Section 8.2, "User Self-Service Process Flows" in the *Administration Guide*.

When performing user self-service functions, you can enable one or more security methods, such as email validation, Google reCAPTCHA, knowledge-based authentication, or custom plugins. Each configured security method requires requests to be sent from OpenAM to the client, and completed responses returned to OpenAM to verify.

Important

At least one security method should be enabled for each self-service feature.

A unique token is provided in the second request to the client that must be used in any subsequent responses, so that OpenAM can maintain the state of the self-service process.

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

2.1.15.1. Registering Users

This section explains how to use the REST APIs for registering a user in OpenAM.

From OpenAM 13.5.1, the user self-registration flow validates the email address after the user has provided their details. This section provides procedures for OpenAM 13.5 and OpenAM 13.5.1 and later:

- For information about how to register a user with the REST APIs in OpenAM 13.5.1 or later, see Procedure 2.10, "To Register a User with the REST APIs (13.5.1 or later)".
- For information about how to register a user with the REST APIs in OpenAM 13.5, see Procedure 2.11, "To Register a User with the REST APIs (13.5)".

Procedure 2.10. To Register a User with the REST APIs (13.5.1 or later)

1. Create a GET request to the `/selfservice/userRegistration` endpoint. Notice that the request does not require any form of authentication.

```
$ curl \
https://openam.example.com:8443/openam/json/selfservice/userRegistration
{
  "requirements": {
    "$schema": "http://json-schema.org/draft-04/schema#",
    "description": "New user details",
    "properties": {
      "user": {
        "description": "User details",
        "type": "object"
      }
    },
    "required": [
      "user"
    ],
    "type": "object"
  },
  "tag": "initial",
  "type": "userDetails"
}
```

OpenAM sends a request to complete the user details. The `required` array defines the data that must be returned to OpenAM to progress past this step of the registration. In the example, the required type is a `user` object that contains the user details.

2. Create a POST response back to the `/selfservice/userRegistration` endpoint with a query string containing `_action=submitRequirements`. In the POST data, include an `input` element in the JSON structure, which should contain values for each element in the `required` array of the request.

In this example, OpenAM requests an object named `user`. This object should contain values for the `username`, `givenName`, `sn`, `mail`, `userPassword`, and `inetUserStatus` properties.

```
$ curl \
--request POST
\
--header "Content-Type: application/json"
\
--data \
'{
  "input": {
    "user": {
      "username": "DEMO",
      "givenName": "Demo User",
      "sn": "User",
      "mail": "demo@example.com",
      "userPassword": "forgerock",
      "inetUserStatus": "Active"
    }
  }
}
```

```
}' \ https://openam.example.com:8443/openam/json/selfservice/userRegistration?
action=submitRequirements
{
  "requirements": {
    "$schema": "http://json-schema.org/draft-04/schema#",
    "description": "Verify emailed code",
    "properties": {
      "code": {
        "description": "Enter code emailed",
        "type": "string"
      }
    },
    "required": [
      "code"
    ],
    "type": "object"
  },
  "tag": "validateCode",
  "token": "eyJ0eXAiOiJKV...QiLCJmqrUfQ",
  "type": "emailValidation"
}
```

If the response is accepted, OpenAM continues with the registration process and sends the next request for information.

The value of the `token` element should be included in this and any subsequent responses to OpenAM for this registration; OpenAM uses this information to track which stage of the registration process is being completed.

Note that the request for information is of the type `emailValidation`. Other possible types include:

- `captcha`, if the Google reCAPTCHA plugin is enabled
- `kbaSecurityAnswerDefinitionStage`, if knowledge-based security questions are required

For an example of Google reCAPTCHA validation, see Section 2.1.15.2, "Retrieving Forgotten Usernames".

3. Return the information required by the next step of the registration, along with the `token` element.

In this example, the user information was accepted and a code was emailed to the email address. OpenAM requires this code in the response in an element named `code` before continuing:

```
$ curl \
--request POST \
--header "Content-Type: application/json" \
--data \
'{"input": {
  "code": "cf53fcb6-3bf2-44eb-a437-885296899699"
},
"token": "eyJ0eXAiOiJKV1QiLCJmqrLqUfQ"}' https://openam.example.com:8443/openam/json/selfservice/userRegistration
?_action=submitRequirements
{
  "type": "selfRegistration",
  "tag": "end",
  "status": {
    "success": true
  },
  "additions": {
  }
}
```

When the process is complete, the response from OpenAM has a `tag` property with value of `end`. If the `success` property in the `status` object has a value of `true`, then self-registration is complete and the user account was created.

In the example, OpenAM only required email verification to register a new user. In flows containing Google reCAPTCHA validation or knowledge-based security questions, you would continue returning POST data to OpenAM containing the requested information until the process is complete.

Note

The User Self-Service feature provides options to set the user's destination after a successful self-registration. These options include redirecting the user to a 'successful registration' page, to the login page, or automatically logging the user into the system. Use the `Destination After Successful Self-Registration` property to set the option (on the console: *Realm Name* > Services > User Self-Service > User Registration). When you select `User sent to 'successful registration' page` or `User sent to login page`, the JSON response after a successful registration is as follows:

```
{
  "type": "selfRegistration",
  "tag": "end",
  "status": {
    "success": true
  },
  "additions": {}
}
```

If you select `User is automatically logged in`, the JSON response is:

```
{
  "type": "autoLoginStage",
  "tag": "end",
  "status": {
    "success": true
  },
  "additions": {
    "tokenId": "AQIC5...MQAA*",
    "successUrl": "/openam/console"
  }
}
```

Procedure 2.11. To Register a User with the REST APIs (13.5)

1. Create a GET request to the `/selfservice/userRegistration` endpoint. Notice that the request does not require any form of authentication.

```
$ curl \
https://openam.example.com:8443/openam/json/selfservice/userRegistration
{
  "type": "emailValidation",
  "tag": "initial",
  "requirements": {
    "$schema": "http://json-schema.org/draft-04/schema#",
    "description": "Verify your email address",
    "type": "object",
    "required": [
      "mail"
    ],
    "properties": {
      "mail": {
        "description": "Email address",
        "type": "string"
      }
    }
  }
}
```

OpenAM sends the first request for security information. In this example, the first request is of type `emailValidation`, but other types include `captcha` if the Google reCAPTCHA plugin is enabled, and `kbaSecurityAnswerDefinitionStage` if knowledge-based authentication is required.

The `required` array defines the data that must be returned to OpenAM to progress past this step of the registration.

The `properties` element contains additional information about the required response, such as a description of the required field, or the site key required to generate a reCAPTCHA challenge.

2. Create a POST response back to the `/selfservice/userRegistration` endpoint with a query string containing `_action=submitRequirements`. In the POST data, include an `input` element in the JSON structure, which should contain values for each element in the `required` array of the request.

In this example, a `mail` value was requested.

```
$ curl \
--request POST \
--header "Content-Type: application/json" \
--data \
'{
  "input": {
    "mail": "demo.user@example.com"
  }
}' \
https://openam.example.com:8443/openam/json/selfservice/userRegistration\
?_action=submitRequirements
{
  "type": "emailValidation",
  "tag": "validateCode",
  "requirements": {
    "$schema": "http://json-schema.org/draft-04/schema#",
    "description": "Verify emailed code",
    "type": "object",
    "required": [
      "code"
    ],
    "properties": {
      "code": {
        "description": "Enter code emailed",
        "type": "string"
      }
    }
  },
  "token": "eyJhcHis...PIF-lN4s"
}
```

If the response was accepted, OpenAM continues with the registration process and sends the next request for information. In this example, the email address was accepted and a code was emailed to the address, which OpenAM requires in the response in an element named `code` before continuing.

The value of the `token` element should be included in this and any subsequent responses to OpenAM for this registration.

3. Continue returning POST data to OpenAM containing the requested information, in the format specified in the request. Also return the `token` value in the POST data, so that OpenAM can track which stage of the registration process is being completed.

```
$ curl \
--request POST \
--header "Content-Type: application/json" \
--data \
'{
  "input": {
    "code": "cf53fcb6-3bf2-44eb-a437-885296899699"
  },
  "token": "eyJhcHis...PIF-lN4s"
}
```

```

}' \
https://openam.example.com:8443/openam/json/selfservice/userRegistration\
?_action=submitRequirements
{
  "type": "userDetails",
  "tag": "initial",
  "requirements": {
    "$schema": "http://json-schema.org/draft-04/schema#",
    "description": "New user details",
    "type": "object",
    "required": [
      "user"
    ],
    "properties": {
      "user": {
        "description": "User details",
        "type": "object"
      }
    }
  },
  "token": "eyJhcHis...PIF-lN4s"
}
    
```

4. When requested—when the `type` value in the request is `userDetails`—supply the details of the new user as an object in the POST data.

```

$ curl \
--request POST \
--header "Content-Type: application/json" \
--data \
'{
  "input": {
    "user": {
      "username": "demo",
      "givenName": "Demo User",
      "sn": "User",
      "userPassword": "d3m0",
      "inetUserStatus": "Active"
    }
  },
  "token": "eyJhcHis...PIF-lN4s"
}' \
https://openam.example.com:8443/openam/json/selfservice/userRegistration\
?_action=submitRequirements
{
  "type": "selfRegistration",
  "tag": "end",
  "status": {
    "success": true
  },
  "additions": {}
}
    
```

When the process is complete, the `tag` element has a value of `end`. If the `success` element in the `status` element has a value of `true`, then self-registration is complete and the user account was created.

2.1.15.2. Retrieving Forgotten Usernames

This section explains how to use the REST APIs to retrieve a forgotten username.

Procedure 2.12. To Retrieve a Forgotten Username with the REST APIs

1. Create a GET request to the `/selfservice/forgottenUsername` endpoint. Notice that the request does not require any form of authentication.

```
$ curl \
https://openam.example.com:8443/openam/json/selfservice/forgottenUsername
{
  "type": "captcha",
  "tag": "initial",
  "requirements": {
    "$schema": "http://json-schema.org/draft-04/schema#",
    "description": "Captcha stage",
    "type": "object",
    "required": [
      "response"
    ],
    "properties": {
      "response": {
        "recaptchaSiteKey": "6Lfr1...cIqbd",
        "description": "Captcha response",
        "type": "string"
      }
    }
  }
}
```

In this example, the Google reCAPTCHA plugin is enabled, so the first request is of the `captcha` type.

2. Create a POST response back to the `/selfservice/forgottenUsername` endpoint with a query string containing `_action=submitRequirements`. In the POST data, include an `input` element in the JSON structure, which should contain values for each element in the `required` array of the request.

In this example, a `response` value was requested, which should be the user input as provided after completing the Google reCAPTCHA challenge.

```
$ curl \
--request POST \
--header "Content-Type: application/json" \
--data \
'{
  "input": {
    "response": "03AHJ...qiE1x4"
  }
}' \
https://openam.example.com:8443/openam/json/selfservice/forgottenUsername\
?_action=submitRequirements
{
  "type": "userQuery",
  "tag": "initial",
  "requirements": {
    "$schema": "http://json-schema.org/draft-04/schema#",
    "description": "Find your account",
    "type": "object",
    "required": [
      "queryFilter"
    ],
    "properties": {
      "queryFilter": {
        "description": "filter string to find account",
        "type": "string"
      }
    }
  },
  "token": "eyJhcHis...PIF-lN4s"
}
```

If the response was accepted, OpenAM continues with the username retrieval process and sends the next request for information. In this example, the Google reCAPTCHA was verified and OpenAM is requesting details about the account name to retrieve, which must be provided in a `queryFilter` element.

The value of the `token` element should be included in this and all subsequent responses to OpenAM for this retrieval process.

3. Create a POST response to OpenAM with a `queryFilter` value in the POST data containing the user's email address associated with their account.

For more information on query filters, see Section 2.1.7, "Filtering, Sorting, and Paging Results".

```
$ curl \
--request POST \
--header "Content-Type: application/json" \
--data \
'{
  "input": {
    "queryFilter": "mail eq \"demo.user@example.com\""
  },
  "token": "eyJhcHis...PIF-lN4s"
}' \
```



```
https://openam.example.com:8443/openam/json/selfservice/forgottenUsername\
?_action=submitRequirements
{
  "type": "kbaSecurityAnswerVerificationStage",
  "tag": "initial",
  "requirements": {
    "$schema": "http://json-schema.org/draft-04/schema#",
    "description": "Answer security questions",
    "type": "object",
    "required": [
      "answer1"
    ],
    "properties": {
      "answer1": {
        "systemQuestion": {
          "en": "What was the model of your first car?"
        },
        "type": "string"
      }
    }
  },
  "token": "eyJhcHis...PIF-lN4s"
}
```

If a single subject is located that matches the provided query filter, the retrieval process continues.

If KBA is enabled, OpenAM requests answers to the configured number of KBA questions, as in this example.

If a subject is not found, an HTTP 400 Bad Request status is returned, and an error message in the JSON data:

```
{
  "code": 400,
  "reason": "Bad Request",
  "message": "Unable to find account"
}
```

4. Return a POST response with the answers as values of the elements specified in the `required` array, in this example `answer1`. Ensure the same `token` value is sent with each response.

```
$ curl \
--request POST \
--header "Content-Type: application/json" \
--data \
'{"input": {
  "answer1": "Mustang"
},
"token": "eyJhc2kiOiJPIF-lN4s"}' \
https://openam.example.com:8443/openam/json/selfservice/forgottenUsername\
?_action=submitRequirements
{"type": "retrieveUsername",
"tag": "end",
"status": {
  "success": true
},
"additions": {
  "userName": "demo"
}
}
```

When the process is complete, the `tag` element has a value of `end`. If the `success` element in the `status` element has a value of `true`, then username retrieval is complete and the username is emailed to the registered address.

If the Show Username option is enabled for username retrieval, the username retrieved is also returned in the JSON response as the value of the `userName` element, as in the example above.

2.1.15.3. Replacing Forgotten Passwords

This section explains how to use the REST APIs to replace a forgotten password.

Procedure 2.13. To Replace a Forgotten Password with the REST APIs

1. Create a GET request to the `/selfservice/forgottenPassword` endpoint. Notice that the request does not require any form of authentication.

```
$ curl \
https://openam.example.com:8443/openam/json/selfservice/forgottenPassword
{
  "type": "captcha",
  "tag": "initial",
  "requirements": {
    "$schema": "http://json-schema.org/draft-04/schema#",
    "description": "Captcha stage",
    "type": "object",
    "required": [
      "response"
    ],
    "properties": {
      "response": {
        "recaptchaSiteKey": "6Lfr1...cIqbd",
        "description": "Captcha response",
        "type": "string"
      }
    }
  }
}
```

In this example the Google reCAPTCHA plugin is enabled, so the first request is of the `captcha` type.

2. Create a POST response back to the `/selfservice/forgottenPassword` endpoint with a query string containing `_action=submitRequirements`. In the POST data, include an `input` element in the JSON structure, which should contain values for each element in the `required` array of the request.

In this example, a `response` value was requested, which should be the user input as provided after completing the Google reCAPTCHA challenge.

```
$ curl \
--request POST
\
--header "Content-Type: application/json"
\
--data \
'{
  "input": {
    "response": "03AHJ...qiE1x4"
  }
}' \
https://openam.example.com:8443/openam/json/selfservice/forgottenPassword\
?_action=submitRequirements
{
  "type": "userQuery",
  "tag": "initial",
  "requirements": {
    "$schema": "http://json-schema.org/draft-04/schema#",
    "description": "Find your account",
    "type": "object",
    "required": [
      "queryFilter"
    ],
    "properties": {
      "queryFilter": {
        "description": "filter string to find account",
        "type": "string"
      }
    }
  },
  "token": "eyJhcHis...PIF-lN4s"
}
```

If the response was accepted, OpenAM continues with the password reset process and sends the next request for information. In this example, the Google reCAPTCHA was verified and OpenAM is requesting details about the account with the password to replace, which must be provided in a `queryFilter` element.

The value of the `token` element should be included in this and all subsequent responses to OpenAM for this reset process.

3. Create a POST response to OpenAM with a `queryFilter` value in the POST data containing the username of the subject with the password to replace.

For more information on query filters, see Section 2.1.7, "Filtering, Sorting, and Paging Results".

```
$ curl \
--request POST
\
--header "Content-Type: application/json"
\
--data \
'{
  "input": {
    "queryFilter": "uid eq \"demo\""
  },
}
```

```

        "token": "eyJhcHis...PIF-lN4s"
    }' \
https://openam.example.com:8443/openam/json/selfservice/forgottenPassword\
?_action=submitRequirements
{
  "type": "kbaSecurityAnswerVerificationStage",
  "tag": "initial",
  "requirements": {
    "$schema": "http://json-schema.org/draft-04/schema#",
    "description": "Answer security questions",
    "type": "object",
    "required": [
      "answer1"
    ],
    "properties": {
      "answer1": {
        "systemQuestion": {
          "en": "What was the model of your first car?"
        },
        "type": "string"
      }
    }
  }
},
"token": "eyJhcHis...PIF-lN4s"
}
    
```

If a single subject is located that matches the provided query filter, the password reset process continues.

If a subject is not found, an HTTP 400 Bad Request status is returned, and an error message in the JSON data:

```

{
  "code": 400,
  "reason": "Bad Request",
  "message": "Unable to find account"
}
    
```

4. Continue returning POST data to OpenAM containing the requested information, in the format specified in the request. Also return the `token` value in the POST data, so that OpenAM can track which stage of the password reset process is being completed.

```

$ curl \
--request POST \
--header "Content-Type: application/json" \
--data \
'{"input": {
  "answer1": "Mustang"
},
"token": "eyJhcHis...PIF-lN4s"}' \
https://openam.example.com:8443/openam/json/selfservice/forgottenPassword\
?_action=submitRequirements
{
  "type": "resetStage",
  "tag": "initial",
  "requirements": {
    "$schema": "http://json-schema.org/draft-04/schema#",
    "description": "Reset password",
    "type": "object",
    "required": [
      "password"
    ],
    "properties": {
      "password": {
        "description": "Password",
        "type": "string"
      }
    }
  },
  "token": "eyJhcHis...PIF-lN4s"
}
    
```

- When requested—when the `type` value in the request is `resetStage`—supply the new password in the POST data.

```
$ curl \
--request POST \
--header "Content-Type: application/json" \
--data \
'{"input": {
  "password": "User1234"
},
"token": "eyJhcHis...PIF-lN4s"}' \
https://openam.example.com:8443/openam/json/selfservice/forgottenPassword\
?_action=submitRequirements
{
  "type": "resetStage",
  "tag": "end",
  "status": {
    "success": true
  },
  "additions": {}
}
```

When the process is complete, the `tag` element has a value of `end`. If the `success` element in the `status` element has a value of `true`, then password reset is complete and the new password is now active.

If the password is not accepted, an HTTP 400 Bad Request status is returned, and an error message in the JSON data:

```
{
  "code": 400,
  "reason": "Bad Request",
  "message": "Minimum password length is 8."
}
```

2.1.15.4. Displaying Dashboard Applications

OpenAM lets administrators configure online applications to display applications on user Dashboards. You can use the 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.

2.1.15.5. Resetting Device Profiles

The OpenAM REST API provides an action that lets a user reset their own profile for a registered device running an authenticator app used for two-step verification. Administrators can also use this REST API to reset a user's registered device profile.

Resetting a device profile deletes information about a user's registered device from OpenAM.

Resetting a device profile is useful when:

- A user loses a registered device—for example, a mobile phone—but had not saved the device recovery codes available in the OpenAM dashboard.
- A user loses a registered device and has no device recovery codes remaining.

An administrator or a user can perform an HTTP POST on `/json/subrealm/users/user/devices/2fa/oath?_action=reset` to reset the user's device profile:

```
$ curl \
  --request POST \
  --header "Content-Type: application/json" \
  --header "iplanetDirectoryPro: AQIC5w...2NzEz*" \
  --data '{}' \
  https://openam.example.com:8443/openam/json/mySubrealm/users/myUser/devices/2fa/oath?_action=reset
{"result":true}
```

For more information about device registration, see Section 2.9.3, "Managing Devices for Multi-Factor Authentication" in the *Administration Guide*.

2.1.16. RESTful Identity and Realm Management Services

This section shows how to use the OpenAM RESTful interfaces for identity and realm management.

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

Before making a REST API call to manage a realm or an identity, make sure that you have:

- Authenticated successfully to OpenAM as a user with sufficient privileges to make the REST API call
- Obtained the session token returned after successful authentication

When making the REST API call, pass the session token in the HTTP header. For more information about the OpenAM session token and its use in REST API calls, see Section 2.1.6, "Using the Session Token After Authentication".

2.1.16.1. 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 and 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 [Section 2.1.16.2, "Realm Management"](#) for examples related to realms.

The following sections cover this JSON-based API:

- [Section 2.1.16.1.1, "Creating Identities"](#)
- [Section 2.1.16.1.2, "Reading Identities"](#)
- [Section 2.1.16.1.3, "Updating Identities"](#)
- [Section 2.1.16.1.4, "Deleting Identities"](#)
- [Section 2.1.16.1.5, "Listing Identities"](#)
- [Section 2.1.16.1.7, "Changing Passwords"](#)

2.1.16.1.1. 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 "Content-Type: application/json" \
--header "iplanetDirectoryPro: AQIC5w...2NzEz*" \
--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"
    ],
    "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"
],
"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 "Content-Type: application/json" \
--header "iplanetDirectoryPro: AQIC5w...2NzEz*" \
--data \
'{
  "username": "myAgent",
  "com.sun.identity.agents.config.fqdn.default":
    ["www.example.com"],
  "com.sun.identity.agents.config.repository.location":
    ["centralized"],
  "agenttype": ["WebAgent"],
  "serverurl": ["https://openam.example.com:8443/openam/"],
  "agenturl": ["http://www.example.com:80/"],

```

```

"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
{
  "username": "myAgent",
  "realm": "/",
  "com.sun.identity.agents.config.fqdn.default": [
    "www.example.com"
  ],
  "com.sun.identity.agents.config.repository.location": [
    "centralized"
  ],
  "AgentType": [
    "WebAgent"
  ],
  "com.sun.identity.agents.config.login.url": [
    "[0]=https://openam.example.com:8443/openam/UI/Login"
  ],
  "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"
  ]
}

```

Note

The command output above has been truncated to be more readable. When you create a policy agent profile, OpenAM returns the full profile in JSON format.

```

$ curl \
  --request POST \
  --header "Content-Type: application/json" \
  --header "iplanetDirectoryPro: AQIC5w...2NzEz*" \
  --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"
  ]
}

```

2.1.16.1.2. 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 [Section 2.1.16.1.3, "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 own accounts, and administrators can also read other user's profiles.

Note

If an administrator user is reading their own profile, an additional `roles` element, with a value of `ui-admin` is returned in the JSON response. The XUI verifies this element to grant or deny access to the OpenAM Console.

The following example shows an administrator accessing user data belonging to `demo`:

```
$ 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"
  ],
  "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/myAgent
{
  "username": "myAgent",
  "realm": "/",
  "com.sun.identity.agents.config.fqdn.default": [
    "www.example.com"
  ],
  "com.sun.identity.agents.config.repository.location": [
    "centralized"
  ],
  "AgentType": [
    "WebAgent"
  ],
  "com.sun.identity.agents.config.login.url": [
    "[0]=https://openam.example.com:8443/openam/UI/Login"
  ],
  "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"
  ]
}
```

Note

The command output above has been truncated to be more readable. When you read a policy agent profile, OpenAM returns the full profile in JSON format.

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/newGroup?_prettyPrint=true
{
  "username": "newGroup",
  "realm": "dc=openam,dc=forgerock,dc=org",
  "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"
  ]
}
```

2.1.16.1.3. 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"
  ],
  "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 updates a web policy agent profile:

```

$ curl \
  --request PUT \
  --header "iPlanetDirectoryPro: AQIC5...Y3MTAx*" \
  --header "Content-Type: application/json" \
  --data '{
    "sunIdentityServerDeviceStatus" : [ "Inactive" ]
  }' \
  https://openam.example.com:8443/openam/json/agents/myAgent?_prettyPrint=true
{
  "username": "myAgent",
  "realm": "/",
  "com.sun.identity.agents.config.fqdn.default": [
    "www.example.com"
  ],
  "com.sun.identity.agents.config.repository.location": [
    "centralized"
  ],
  "AgentType": [
    "WebAgent"
  ],
  "com.sun.identity.agents.config.login.url": [
    "[0]=https://openam.example.com:8443/openam/UI/Login"
  ],
  "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": [
        "Inactive"
    ]
}

```

Note

The command output above has been truncated to be more readable. When you update a policy agent profile, OpenAM returns the full profile in JSON format.

Notice in the following example that updates `newGroup` 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 '{
    "uniqueMember": [
      "uid=newUser,ou=people,dc=openam,dc=forgerock,dc=org",
      "uid=demo,ou=people,dc=openam,dc=forgerock,dc=org"
    ]
  }' \
  https://openam.example.com:8443/openam/json/groups/newGroup
{
  "name": "newGroup",
  "realm": "/",
  "uniqueMember": [
    "uid=newUser,ou=people,dc=openam,dc=forgerock,dc=org",
    "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"
  ]
}

```

2.1.16.1.4. 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/my0Auth2ClientAgent
{"success": "true"}
```

```
$ curl \
  --request DELETE \
  --header "iPlanetDirectoryPro: AqIC5w...2NzEz*" \
  https://openam.example.com:8443/openam/json/groups/newGroup
{"success": "true"}
```

Note

Deleting a user does not automatically remove any of the user's sessions. If you are using stateful sessions, you can remove a user's sessions by checking for any sessions for the user and then removing them using the console's Sessions tab. If you are using stateless sessions, you cannot remove users' sessions; you must wait for the sessions to expire.

2.1.16.1.5. 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": [
    {
      "username": "amAdmin",
```

```

"realm": "dc=openam,dc=forgerock,dc=org",
"sunIdentityMSISDNNumber": [],
"mail": [],
"sn": [
  "amAdmin"
],
"givenName": [
  "amAdmin"
],
"universalid": [
  "id=amAdmin,ou=user,dc=openam,dc=forgerock,dc=org"
],
"cn": [
  "amAdmin"
],
"iplanet-am-user-success-url": [],
"telephoneNumber": [],
"roles": [
  "ui-global-admin",
  "ui-realm-admin"
],
"iplanet-am-user-failure-url": [],
"inetuserstatus": [
  "Active"
],
"postalAddress": [],
"dn": [
  "uid=amAdmin,ou=people,dc=openam,dc=forgerock,dc=org"
],
"employeeNumber": [],
"iplanet-am-user-alias-list": []
},
{
  "username": "demo",
  "realm": "dc=openam,dc=forgerock,dc=org",
  "uid": [
    "demo"
  ],
  "createTimestamp": [
    "20160108155628Z"
  ],
  "inetUserStatus": [
    "Active"
  ],
  "mail": [
    "demo.user@example.com"
  ],
  "sn": [
    "demo"
  ],
  "cn": [
    "demo"
  ],
  "objectClass": [
    "devicePrintProfilesContainer",
    "person",
    "sunIdentityServerLibertyPPService",
    "sunFederationManagerDataStore",
    "inetorgperson",
  ]
}

```

```

"oathDeviceProfilesContainer",
"iPlanetPreferences",
"iplanet-am-auth-configuration-service",
"sunFMSAML2NameIdentifier",
"organizationalperson",
"inetuser",
"kbaInfoContainer",
"forgerock-am-dashboard-service",
"iplanet-am-managed-person",
"iplanet-am-user-service",
"sunAMAuthAccountLockout",
"top"
],
"kbaInfo": [
  {
    "questionId": "2",
    "answer": {
      "$crypto": {
        "value": {
          "algorithm": "SHA-256",
          "data": "VXGtsnjJMC...MQJ/goU5hkfF"
        },
        "type": "salted-hash"
      }
    }
  },
  {
    "questionId": "1",
    "answer": {
      "$crypto": {
        "value": {
          "algorithm": "SHA-256",
          "data": "cfYYzi9U...rVfFl0Tdw0iX"
        },
        "type": "salted-hash"
      }
    }
  }
],
"dn": [
  "uid=demo,ou=people,dc=openam,dc=forgerock,dc=org"
],
"universalid": [
  "id=demo,ou=user,dc=openam,dc=forgerock,dc=org"
],
"modifyTimestamp": [
  "20160113010610Z"
]
}
],
"resultCount": 2,
"pagedResultsCookie": null,
"totalPagedResultsPolicy": "NONE",
"totalPagedResults": -1,
"remainingPagedResults": -1
}

```


The `users` endpoint also supports the `_queryFilter` parameter to alter the returned results. For more information, see Section 2.1.7, "Filtering, Sorting, and Paging Results".

The `_queryId=*` parameter 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" : [ "newGroup", "anotherGroup" ],
  "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.

2.1.16.1.6. 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 "Content-Type: application/json" \
--header "iPlanetDirectoryPro: AqIC5wM2LY4SfczUFNs-TJwFrCVAKgR0NuLIayNaIkQmjis.*AAJTSQACMDEA
A1NLABQtNTQ3NDE2Njc50Dk4MjYzMzA2MQ..*" \
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
}
```

2.1.16.1.7. Changing Passwords

Users other than the top-level administrator can change their own passwords with an HTTP POST to `/json/subrealm/users/username?_action=changePassword` including the new password as the value of `userpassword` in the request data.

Note

Changing the top-level administrator's password requires a more complex procedure. See Section 27.5, "Administering the amadmin Account" in the *Administration Guide* for more information.

Users must provide the current password, which is set in the request as the value of the `currentpassword`.

For cases where users have forgotten their password, see Section 2.1.15.2, "Retrieving Forgotten Usernames" instead.

The following example shows a successful request to change the `demo` user's password to `password`:

```
$ curl \
--request POST \
--header "Content-Type: application/json" \
--header "iPlanetDirectoryPro: AqIC5w...NTcy*" \
--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 Section 2.1.12, "REST Status Codes" for more information.

Administrators can change non-administrative users' passwords with an HTTP PUT to `/json/subrealm/users/username` including the new password as the value of `userpassword` in the request data.

Unlike users, administrators do not provide users' current passwords when changing passwords.

The following example shows a successful request by an administrator to change the `demo` user's password to `cangetin`:

```
$ curl \
--request PUT \
--header "iPlanetDirectoryPro: AqIC5w...NTcy*" \
--header "Content-Type: application/json" \
--data '{
  "userpassword":"cangetin"
}' \
https://openam.example.com:8443/openam/json/users/demo
{}
```

```
{
  "username": "demo",
  "realm": "/",
  "uid": [
    "demo"
  ],
  "universalid": [
    "id=demo,ou=user,dc=example,dc=com"
  ],
  "objectClass": [
    "iplanet-am-managed-person",
    "inetuser", "sunFederationManagerDataStore",
    "sunFMSAML2NameIdentifier",
    "devicePrintProfilesContainer",
    "inetorgperson",
    "sunIdentityServerLibertyPPService",
    "iplanetPreferences",
    "iplanet-am-user-service",
    "forgerock-am-dashboard-service",
    "organizationalperson",
    "top",
    "sunAMAuthAccountLockout",
    "person",
    "oathDeviceProfilesContainer",
    "iplanet-am-auth-configuration-service"
  ],
  "inetUserStatus": [
    "Active"
  ],
  "dn": [
    "uid=demo,ou=people,dc=example,dc=com"
  ],
  "sn": [
    "demo"
  ],
  "cn": [
    "demo"
  ],
  "modifyTimestamp": [
    "20151006213634Z"
  ],
  "createTimestamp": [
    "20151005134244Z"
  ]
}
```

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. See also Section 2.1.12, "REST Status Codes" for more information.

2.1.16.2. 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.

- Section 2.1.16.2.1, "Default Parameters for Realms"
- Section 2.1.16.2.2, "Creating Realms"
- Section 2.1.16.2.3, "Reading Realms"
- Section 2.1.16.2.4, "Listing Realms"
- Section 2.1.16.2.5, "Updating Realms"
- Section 2.1.16.2.6, "Deleting Realms"

2.1.16.2.1. 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 whether they can be updated, added, or deleted when updating a realm.

Table 2.10. 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: <code>myRealm</code>
sunOrganizationStatus	Active	The status of the realm <code>Active</code> or <code>Inactive</code>
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 will remove the alias from that realm and you will lose configuration. Example: <code>opensso.example.com</code>
serviceNames	<code>sunAMAuthHOTPSERVICE</code> <code>iPlanetAMAuthConfiguration</code> <code>sunAMAuthFederationService</code> <code>sunIdentityRepositoryService</code> <code>iPlanetAMPolicyConfigService</code> <code>iPlanetAMAuthService</code> <code>iPlanetAMAuthLDAPService</code> <code>sunAMAuthDataStoreService</code> <code>sunAMAuthSAESERVICE</code> <code>sunAMDelegationService</code> <code>sunAMAuthWSSAuthModuleService</code> <code>iPlanetAMAuthOATHService</code>	Services needed for the realm, including authentication modules

2.1.16.2.2. 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 data field is `realm`:

```
$ curl \
--request POST \
--header "Content-Type: application/json" \
--header "iplanetDirectoryPro: AQIC5w...2NzEz*" \
--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 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 "If-None-Match: *" \
--header "iplanetDirectoryPro: AQIC5w...2NzEz*" \
--header "Content-Type: application/json" \
--data '{ }' \
https://openam.example.com:8443/openam/json/realms/myRealm

{
  "realmCreated": "/myRealm"
}
```

OpenAM returns an HTTP 201 Created status code, and the JSON representation of the realm on success. On failure, OpenAM returns a JSON representation of the error including the HTTP status code. For example, if the `If-None-Match` header with a value of `*` is absent, an HTTP 404 Not Found status code is returned.

2.1.16.2.3. 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 Section 2.1.16.2.5, "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"
  ]
}
```

2.1.16.2.4. Listing Realms

To list a realm and its subrealms, perform an HTTP GET on the endpoint, set 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
}
```

2.1.16.2.5. 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.

2.1.16.2.6. 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" }
```

2.1.17. RESTful Script Management

This section shows you how to manage scripts used for client-side and server-side scripted authentication, custom policy conditions, and handling OpenID Connect claims by using the REST API.

For information on managing scripts by using the OpenAM console, see Chapter 22, "*Managing Scripts*" in the *Administration Guide*. For information on configuring script settings, see Section 1.4.19, "Scripting" in the *Reference*.

OpenAM provides the `scripts` REST endpoint for the following:

- Section 2.1.17.1, "Querying Scripts"
- Section 2.1.17.2, "Reading a Script"
- Section 2.1.17.3, "Validating a Script"
- Section 2.1.17.4, "Creating a Script"
- Section 2.1.17.5, "Updating a Script"
- Section 2.1.17.6, "Deleting a Script"

User-created scripts are realm-specific, hence the URI for the scripts' API can contain a realm component, such as `/json{/realm}/scripts`. If the realm is not specified in the URI, the top level realm is used.

Tip

OpenAM includes some global example scripts that can be used in any realm.

Scripts are represented in JSON and take the following form. Scripts are built from standard JSON objects and values (strings, numbers, objects, sets, arrays, `true`, `false`, and `null`). Each script has a system-generated *universally unique identifier* (UUID), which must be used when modifying existing scripts. Renaming a script will not affect the UUID:

```
{
  "_id": "7e3d7067-d50f-4674-8c76-a3e13a810c33",
  "name": "Scripted Module - Server Side",
  "description": "Default global script for server side Scripted Authentication Module",
  "script": "dmFyIFNUNUQVJUX1RJ...",
  "language": "JAVASCRIPT",
  "context": "AUTHENTICATION_SERVER_SIDE",
  "createdBy": "id=dsameuser,ou=user,dc=openam,dc=forgerock,dc=org",
  "creationDate": 1433147666269,
  "lastModifiedBy": "id=dsameuser,ou=user,dc=openam,dc=forgerock,dc=org",
  "lastModifiedDate": 1433147666269
}
```

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

`_id`

The UUID that OpenAM generates for the script.

name

The name provided for the script.

description

An optional text string to help identify the script.

script

The source code of the script. The source code is in UTF-8 format and encoded into Base64.

For example, a script such as the following:

```
var a = 123;
var b = 456;
```

When encoded into Base64 becomes:

```
dmFyIGEGPSAxMjM7IA0KdmFyIGIyPSA0NTY7
```

Language

The language the script is written in - **JAVASCRIPT** or **GROOVY**.

Table 2.11. Language Support per Context

Script Context	Supported Languages
POLICY_CONDITION	JAVASCRIPT, GROOVY
AUTHENTICATION_SERVER_SIDE	JAVASCRIPT, GROOVY
AUTHENTICATION_CLIENT_SIDE	JAVASCRIPT
OIDC_CLAIMS	JAVASCRIPT, GROOVY

context

The context type of the script.

Supported values are:

POLICY_CONDITION

Policy Condition

AUTHENTICATION_SERVER_SIDE

Server-side Authentication

AUTHENTICATION_CLIENT_SIDE

Client-side Authentication

Note

Client-side scripts must be written in JavaScript.

OIDC_CLAIMS

OIDC Claims

createdBy

A string containing the universal identifier DN of the subject that created the script.

creationDate

An integer containing the creation date and time, in ISO 8601 format.

lastModifiedBy

A string containing the universal identifier DN of the subject that most recently updated the resource type.

If the script has not been modified since it was created, this property will have the same value as **createdBy**.

lastModifiedDate

A string containing the last modified date and time, in ISO 8601 format.

If the script has not been modified since it was created, this property will have the same value as **creationDate**.

2.1.17.1. Querying Scripts

To list all the scripts in a realm, as well as any global scripts, perform an HTTP GET to the `/json{/realm}/scripts` endpoint with a `_queryFilter` parameter set to `true`.

Note

If the realm is not specified in the URL, OpenAM returns scripts in the top level realm, as well as any global scripts.

The `iPlanetDirectoryPro` header is required and should contain the SSO token of an administrative user, such as `amAdmin`, who has access to perform the operation.

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

```

{
  "result": [
    {
      "_id": "9de3eb62-f131-4fac-a294-7bd170fd4acb",
      "name": "Scripted Policy Condition",
      "description": "Default global script for Scripted Policy Conditions",
      "script": "LyoqCiAqIFRoaxMg...",
      "language": "JAVASCRIPT",
      "context": "POLICY_CONDITION",
      "createdBy": "id=dsameuser,ou=user,dc=openam,dc=forgerock,dc=org",
      "creationDate": 1433147666269,
      "lastModifiedBy": "id=dsameuser,ou=user,dc=openam,dc=forgerock,dc=org",
      "lastModifiedDate": 1433147666269
    },
    {
      "_id": "7e3d7067-d50f-4674-8c76-a3e13a810c33",
      "name": "Scripted Module - Server Side",
      "description": "Default global script for server side Scripted Authentication Module",
      "script": "dmFyIFNUQVJUX1RJ...",
      "language": "JAVASCRIPT",
      "context": "AUTHENTICATION_SERVER_SIDE",
      "createdBy": "id=dsameuser,ou=user,dc=openam,dc=forgerock,dc=org",
      "creationDate": 1433147666269,
      "lastModifiedBy": "id=dsameuser,ou=user,dc=openam,dc=forgerock,dc=org",
      "lastModifiedDate": 1433147666269
    }
  ],
  "resultCount": 2,
  "pagedResultsCookie": null,
  "remainingPagedResults": -1
}
    
```

Additional query strings can be specified to alter the returned results. For more information, see Section 2.1.7, "Filtering, Sorting, and Paging Results".

Table 2.12. Supported `_queryFilter` Fields and Operators

Field	Supported Operators
<code>_id</code>	Equals (<code>eq</code>), Contains (<code>co</code>), Starts with (<code>sw</code>)
<code>name</code>	Equals (<code>eq</code>), Contains (<code>co</code>), Starts with (<code>sw</code>)
<code>description</code>	Equals (<code>eq</code>), Contains (<code>co</code>), Starts with (<code>sw</code>)
<code>script</code>	Equals (<code>eq</code>), Contains (<code>co</code>), Starts with (<code>sw</code>)
<code>language</code>	Equals (<code>eq</code>), Contains (<code>co</code>), Starts with (<code>sw</code>)
<code>context</code>	Equals (<code>eq</code>), Contains (<code>co</code>), Starts with (<code>sw</code>)

2.1.17.2. Reading a Script

To read an individual script in a realm, perform an HTTP GET using the `/json{/realm}/scripts` endpoint, specifying the UUID in the URL.

Tip

To read a script in the top-level realm, or to read a built-in global script, do not specify a realm in the URL.

The `iPlanetDirectoryPro` header is required and should contain the SSO token of an administrative user, such as `amAdmin`, who has access to perform the operation.

```
$ curl \
--header "iPlanetDirectoryPro: AQIC5..." \
https://openam.example.com:8443/openam/json/myrealm/scripts/9de3eb62-f131-4fac-a294-7bd170fd4acb
{
  "_id": "9de3eb62-f131-4fac-a294-7bd170fd4acb",
  "name": "Scripted Policy Condition",
  "description": "Default global script for Scripted Policy Conditions",
  "script": "LyoqCiAqIFRoaxMg...",
  "language": "JAVASCRIPT",
  "context": "POLICY_CONDITION",
  "createdBy": "id=dsameuser,ou=user,dc=openam,dc=forgerock,dc=org",
  "creationDate": 1433147666269,
  "lastModifiedBy": "id=dsameuser,ou=user,dc=openam,dc=forgerock,dc=org",
  "lastModifiedDate": 1433147666269
}
```

2.1.17.3. Validating a Script

To validate a script, perform an HTTP POST using the `/json{/realm}/scripts` endpoint, with an `_action` parameter set to `validate`. Include a JSON representation of the script and the script language, `JAVASCRIPT` or `GROOVY`, in the POST data.

The value for `script` must be in UTF-8 format and then encoded into Base64.

The `iPlanetDirectoryPro` header is required and should contain the SSO token of an administrative user, such as `amAdmin`, who has access to perform the operation.

```
$ curl \
--request POST
\
--header "Content-Type: application/json"
\
--header "iPlanetDirectoryPro: AQIC5..."
\
--data '{
  "script": "dmFyIGEGPSAxMjM7dmFyIGIGPSA0NTY7Cg==",
  "language": "JAVASCRIPT"
}' \
https://openam.example.com:8443/openam/json/myrealm/scripts/?_action=validate
{
  "success": true
}
```

If the script is valid the JSON response contains a `success` key with a value of `true`.

If the script is invalid the JSON response contains a `success` key with a value of `false`, and an indication of the problem and where it occurs, as shown below:

```
$ curl \
--request POST \
--header "Content-Type: application/json" \
--header "iPlanetDirectoryPro: AQIC5..." \
--data '{
  "script": "dmFyIGEGPSAxMjM7dmFyIGIgPSA0NTY7ID1WQUxJREFUSU90IFNIT1VMRCBGQU1MPQo=",
  "language": "JAVASCRIPT"
}' \
https://openam.example.com:8443/openam/json/myrealm/scripts/?_action=validate
{
  "success": false,
  "errors": [
    {
      "line": 1,
      "column": 27,
      "message": "syntax error"
    }
  ]
}
```

2.1.17.4. Creating a Script

To create a script in a realm, perform an HTTP POST using the `/json{/realm}/scripts` endpoint, with an `_action` parameter set to `create`. Include a JSON representation of the script in the POST data.

The value for `script` must be in UTF-8 format and then encoded into Base64.

Note

If the realm is not specified in the URL, OpenAM creates the script in the top level realm.

The `iPlanetDirectoryPro` header is required and should contain the SSO token of an administrative user, such as `amAdmin`, who has access to perform the operation.

```
$ curl \
--request POST \
--header "Content-Type: application/json" \
--header "iPlanetDirectoryPro: AQIC5..." \
--data '{
  "name": "MyJavaScript",
  "script": "dmFyIGEGPSAxMjM7CnZhciBiID0gNDU2Ow==",
  "language": "JAVASCRIPT",
  "context": "POLICY_CONDITION",
  "description": "An example script"
}' \
https://openam.example.com:8443/openam/json/myrealm/scripts/?_action=create
{
  "_id": "0168d494-015a-420f-ae5a-6a2a5c1126af",
  "name": "MyJavaScript",
  "description": "An example script",
  "script": "dmFyIGEGPSAxMjM7CnZhciBiID0gNDU2Ow==",
  "language": "JAVASCRIPT",
  "context": "POLICY_CONDITION",
  "createdBy": "id=amadmin,ou=user,dc=openam,dc=forgerock,dc=org",
  "creationDate": 1436807766258,
  "lastModifiedBy": "id=amadmin,ou=user,dc=openam,dc=forgerock,dc=org",
  "lastModifiedDate": 1436807766258
}
```

2.1.17.5. Updating a Script

To update an individual script in a realm, perform an HTTP PUT using the `/json{/realm}/scripts` endpoint, specifying the UUID in both the URL and the PUT body. Include a JSON representation of the updated script in the PUT data, alongside the UUID.

Note

If the realm is not specified in the URL, OpenAM uses the top level realm.

The `iPlanetDirectoryPro` header is required and should contain the SSO token of an administrative user, such as `amAdmin`, who has access to perform the operation.

```
$ curl \
--header "iPlanetDirectoryPro: AQIC5..." \
--header "Content-Type: application/json" \
--request PUT \
--data '{
  "name": "MyUpdatedJavaScript",
  "script": "dmFyIGEgPSAxMjM7CnZhciBiID0gNDU2Ow==",
  "language": "JAVASCRIPT",
  "context": "POLICY_CONDITION",
  "description": "An updated example script configuration"
}' \
https://openam.example.com:8443/openam/json/myrealm/scripts/0168d494-015a-420f-ae5a-6a2a5c1126af
{
  "_id": "0168d494-015a-420f-ae5a-6a2a5c1126af",
  "name": "MyUpdatedJavaScript",
  "description": "An updated example script configuration",
  "script": "dmFyIGEgPSAxMjM7CnZhciBiID0gNDU2Ow==",
  "language": "JAVASCRIPT",
  "context": "POLICY_CONDITION",
  "createdBy": "id=amadmin,ou=user,dc=openam,dc=forgerock,dc=org",
  "creationDate": 1436807766258,
  "lastModifiedBy": "id=amadmin,ou=user,dc=openam,dc=forgerock,dc=org",
  "lastModifiedDate": 1436808364681
}
```

2.1.17.6. Deleting a Script

To delete an individual script in a realm, perform an HTTP DELETE using the `/json{/realm}/scripts` endpoint, specifying the UUID in the URL.

Note

If the realm is not specified in the URL, OpenAM uses the top level realm.

The `iPlanetDirectoryPro` header is required and should contain the SSO token of an administrative user, such as `amAdmin`, who has access to perform the operation.

```
$ curl \
--request DELETE \
--header "iPlanetDirectoryPro: AQIC5..." \
https://openam.example.com:8443/openam/json/myrealm/scripts/0168d494-015a-420f-ae5a-6a2a5c1126af
{}
```

2.1.18. RESTful Troubleshooting Information Recording

This section shows you how to start, stop, and get the status of a troubleshooting recording event using the REST API.

OpenAM provides the `/json/records` REST endpoint for the following:

- **Starting a recording event.** See Section 2.1.18.1, "Starting a Recording Event".
- **Getting the status of a recording event.** See Section 2.1.18.2, "Getting the Status of a Recording Event".
- **Stopping a recording event.** See Section 2.1.18.3, "Stopping a Recording Event".

You must authenticate to OpenAM as an administrative user to obtain an SSO token prior to calling the `/json/records` REST endpoint. You then pass the SSO token in the `iPlanetDirectoryPro` header as proof of authentication.

You can also record troubleshooting information by using the `ssoadm` command. For more information, see Section 24.5, "Recording Troubleshooting Information" in the *Administration Guide*.

Note

The `curl` command output in the examples in this section is indented for ease of reading. The actual output is *not* indented, and the actions available from the `/json/records` endpoint do not support the `_prettyPrint` parameter.

2.1.18.1. Starting a Recording Event

To start a recording event, perform an HTTP POST using the `/json/records` endpoint, specifying the `_action=start` parameter in the URL. Specify a JSON payload identical in format to the input file for the `ssoadm start-recording` command, as described in Section 24.5.2, "The Recording Control File" in the *Administration Guide*:

```
$ curl \
--request POST \
--header "Content-Type: application/json" \
--header "iPlanetDirectoryPro: AQIC5..." \
--data '{
  "issueID": 103572,
  "referenceID": "policyEvalFails",
  "description": "Troubleshooting artifacts in support of case 103572",
  "zipEnable": true,
  "configExport": {
    "enable": true,
    "password": "5x2RR70",
    "sharePassword": false
  },
  "debugLogs": {
    "debugLevel": "MESSAGE",
    "autoStop": {
      "time": {
        "timeUnit": "SECONDS",
        "value": 15
      },
    },
    "fileSize": {
      "sizeUnit": "GB",
      "value": 1
    }
  }
}
```

```

    },
    "threadDump" : {
      "enable": true,
      "delay" : {
        "timeUnit": "SECONDS",
        "value": 5
      }
    }
  }, \
https://openam.example.com:8443/openam/json/records?_action=start
{
  "recording":true,
  "record":{
    "issueID":103572,
    "referenceID":"policyEvalFails",
    "description":"Troubleshooting artifacts in support of case 103572",
    "zipEnable":true,
    "threadDump":{
      "enable":true,
      "delay":{
        "timeUnit":"SECONDS",
        "value":5
      }
    }
  },
  "configExport":{
    "enable":true,
    "password":"xxxxxx",
    "sharePassword":false
  },
  "debugLogs":{
    "debugLevel":"message",
    "autoStop":{
      "time":{
        "timeUnit":"MILLISECONDS",
        "value":15000
      },
      "fileSize":{
        "sizeUnit":"KB",
        "value":1048576
      }
    }
  },
  "status":"RUNNING",
  "folder":"/opt/demo/openam/config/openam/debug/record/103572/policyEvalFails/"
}
}

```

2.1.18.2. Getting the Status of a Recording Event

To get the status of a recording event, perform an HTTP POST using the `/json/records` endpoint, specifying the `_action=status` parameter in the URL:

```

$ curl \
--request POST \
--header "iPlanetDirectoryPro: AQIC5..." \
https://openam.example.com:8443/openam/json/records?_action=status

```

If there is no active recording event, the following output appears:

```
{
  "recording":false
}
```

If there is an active recording event, output similar to the following appears:

```
{
  "recording":true,
  "record":{
    "issueID":103572,
    "referenceID":"policyEvalFails",
    "description":"Troubleshooting artifacts in support of case 103572",
    "zipEnable":true,
    "threadDump":{
      "enable":true,
      "delay":{
        "timeUnit":"SECONDS",
        "value":5
      }
    },
    "configExport":{
      "enable":true,
      "password":"xxxxxx",
      "sharePassword":false
    },
    "debugLogs":{
      "debugLevel":"message",
      "autoStop":{
        "time":{
          "timeUnit":"MILLISECONDS",
          "value":15000
        },
        "fileSize":{
          "sizeUnit":"KB",
          "value":1048576
        }
      }
    },
    "status":"RUNNING",
    "folder":"/opt/demo/openam/config/openam/debug/record/103572/policyEvalFails/"
  }
}
```

2.1.18.3. Stopping a Recording Event

To stop a recording event, perform an HTTP POST using the `/json/records` endpoint, specifying the `_action=stop` parameter in the URL:

```
$ curl \
--request POST \
--header "iPlanetDirectoryPro: AQIC5..." \
https://openam.example.com:8443/openam/json/records?_action=stop
```

If there is no active recording event, OpenAM returns a 400 error code.

If there is an active recording event, output similar to the following appears:

```
{
  "recording":false,
  "record":{
    "issueID":103572,
    "referenceID":"policyEvalFails",
    "description":"Troubleshooting artifacts in support of case 103572",
    "zipEnable":true,
    "threadDump":{
      "enable":true,
      "delay":{
        "timeUnit":"SECONDS",
        "value":5
      }
    },
    "configExport":{
      "enable":true,
      "password":"xxxxxx",
      "sharePassword":false
    },
    "debugLogs":{
      "debugLevel":"message",
      "autoStop":{
        "time":{
          "timeUnit":"MILLISECONDS",
          "value":15000
        },
        "fileSize":{
          "sizeUnit":"KB",
          "value":1048576
        }
      }
    },
    "status":"STOPPED",
    "folder":"/opt/demo/openam/config/openam/debug/record/103572/policyEvalFails/"
  }
}
```

2.2. Using the OpenAM Java SDK

This section introduces OpenAM Java SDK. OpenAM Java SDK is delivered with the full version of OpenAM, [OpenAM-13.5.2.zip](#).

2.2.1. Installing OpenAM Client SDK Samples

The full OpenAM download, [OpenAM-13.5.2.zip](#), contains the Java Client SDK library, [ClientSDK-13.5.2.jar](#), as well as samples for use on the command line in [ExampleClientSDK-CLI-13.5.2.zip](#), and samples in a web application, [ExampleClientSDK-WAR-13.5.2.war](#). The *OpenAM Java SDK API Specification* provides a reference to the public APIs.

Procedure 2.14. 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-13.5.2.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.

Server Protocol:	<input type="text" value="http"/>
Server Host:	<input type="text" value="openam.example.com"/>
Server Port:	<input type="text" value="8080"/>
Server Deployment URI:	<input type="text" value="/openam"/>
Debug directory	<input type="text" value="/tmp/client-debug"/>
Application user name	<input type="text" value="UrlAccessAgent"/>
Application user password	<input type="password" value="*****"/>
<input type="button" value="Configure"/> <input type="button" value="Reset"/>	

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...CMDEAA\NLABQt0DY0Mjc5MDUwNDQz0TA2MzYxNg..*
...
User Attributes: {... givenName=[amAdmin], ...roles=[Top-level Admin Role], ...}
```

Procedure 2.15. 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-13.5.2.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!!
```

2.2.2. 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**, keystore 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
```

2.2.3. Authenticating Using OpenAM Java SDK

This section 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 section is `com.sun.identity.samples.authentication.Login`.

Before you continue, make sure that the packages described in Section 2.2.1, "Installing OpenAM Client SDK Samples" 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 SSO token 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 UnsupportedOperationException {
    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 UnsupportedOperationException(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);
}
}
```

2.2.3.1. Encoding Passwords and Password Reset Questions and Answers

OpenAM uses symmetric encryption algorithms to encrypt and decrypt stored passwords, so that they can be retrieved or modified at later date if necessary. The OpenAM Java SDK provides the capability to encode passwords using the `EncodeAction` class in standalone applications. For example, you can encrypt and decrypt a password as follows:

```
String plainText = "helloworld";
String encrypted = AccessController.doPrivileged(new EncodeAction(plainText));
String decrypted = AccessController.doPrivileged(new DecodeAction(encrypted));
Assert plainText.equals(decrypted);
```

To use this class, you must ensure that the symmetric encryption key has the same value as configured in the server instances. You can run `ssoadm` to retrieve the password encryption key as follows:

```
ssoadm am.encryption.pwd
```

Next, in your application's `AMConfig.properties` file, replace the `@ENCRYPTION_KEY@` with the value of the password encryption key. The property ensures that OpenAM can decrypt the password.

```
am.encryption.pwd=@ENCRYPTION_KEY@
```

OpenAM's password reset question and answer also uses symmetric key encryption in its configuration. You can use the `encodeAction` class to encrypt a password reset question and answer:

```
String encrypted = AccessController.doPrivileged(new EncodeAction(question + "\t" + \
    answer "+" "1"));
```

The last number in the previous example indicates whether the question/answer is enabled or disabled:

- 0 = default question/answer that is disabled
- 1 = default question/answer that is enabled
- 2 = personal question/answer that is disabled
- 3 = personal question/answer that is enabled

To encrypt or decrypt the password reset question and answer, you must retrieve the password encryption key using `ssoadm am.encryption.key`, and then set the `am.encryption.key` property with the value of the password encryption key in the `AMConfig.properties` file.

For additional information, see *EncodeAction*.

2.2.4. Handling Single Sign-On Using OpenAM Java SDK

This section looks at handling session tokens with the OpenAM Java SDK. The class shown in this section 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 Section 2.2.1, "Installing OpenAM Client SDK Samples" 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. Applications can receive notifications about changes to *stateful sessions only*. Adding an `SSOTokenListener` for a stateless session token does *not* generate notifications.

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.ipланet.sso.SSOException;
import com.ipланet.sso.SSOToken;
import com.ipланet.sso.SSOTokenID;
import com.ipланet.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...A1NLABQtNjI40TkyNTUxNTc4MDQ3NzEzOQ..*
$ sh scripts/SSOTokenSample.sh
Enter SSOToken ID: AQIC5wM2LY4Sfcyy10grl...A1NLABQtNjI40TkyNTUxNTc4MDQ3NzEzOQ..*
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...A1NLABQtNjI40TkyNTUxNTc4MDQ3NzEzOQ..*
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.

2.2.4.1. Receiving Notifications

If your application implements a listener for change notification, such as a `SessionListener` to handle notification when a stateful 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/notificationsservice`.

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/notificationsservice`.

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.

2.2.5. Requesting Policy Decisions Using OpenAM Java SDK

This section 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 Section 2.2.1, "Installing OpenAM Client SDK Samples" 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 SSO token, 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 SSO token 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 Realms > *Realm Name* > Authorization. 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...50DM4NDY0MzE00DYz0DQ1*
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>
```

2.2.6. Requesting a XACML Policy Decision Using OpenAM Java SDK

This section 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 Section 2.2.1, "Installing OpenAM Client SDK Samples" 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;
```

¹The `PolicyDecision` element is defined in `openam/WEB-INF/remoteInterface.dtd` where `openam` is the location where the OpenAM web application is deployed.

```

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 Configure > Global Services, click SAMLv2 SOAP Binding, and then 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 Section 1.3, "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 Chapter 3, "Defining Authorization Policies" in the *Administration Guide* 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>

```

2.3. Using the OpenAM C SDK

This section introduces OpenAM C SDK, which is available for selected platforms. 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

Procedure 2.16. 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 Deployment > Servers > *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 = AQIC5wM2LY4SfcxZfk4EzC9Y46P9cXG9ogwf2ixnY0eZ0K0.*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 3

Customizing OpenAM

Service provider interfaces (SPI's) provide a framework to customize all OpenAM service modules such as adding custom authentication modules, federation plugins, and policy conditions.

This part of the guide covers customizing OpenAM functionality.

3.1. Customizing Profile Attributes

You can extend user profiles by adding custom attributes. This section 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.

Important

In OpenAM 13.5.2-4, the ability to edit custom profile attributes is limited to the classic UI. Custom profile attributes do not appear in the user profile when users log in to OpenAM using the XUI.

This section includes the following procedures.

- Procedure 3.1, "To Update the AMUser Service For the New Attribute"
- Procedure 3.2, "To Update the Identity Repository For the New Attribute"
- Procedure 3.3, "To Allow Users To Update the New Attribute"

Procedure 3.1. 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.
```

Procedure 3.2. 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/opensj/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 Realms > *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.

Procedure 3.3. 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/opensj/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 by adding your custom attribute to the `attributes` JSON array.
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.

Babs Jensen

Custom Attribute:

First Name:

* Last Name:

* Full Name:

3.2. 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 section shows how to write a custom OAuth 2.0 scope validator plugin for use in an OAuth 2.0 provider (authorization server) configuration.

3.2.1. 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.UserInfoClaims;
import org.forgerock.oauth2.core.exceptions.InvalidClientException;
import org.forgerock.oauth2.core.exceptions.NotFoundException;
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,
        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> 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;
    }

    @Override
    public UserInfoClaims getUserInfo(
        AccessToken token,
        OAuth2Request request)

```

```

        throws UnauthorizedClientException, NotFoundException {
    Map<String, Object> response = mapScopes(token);
    response.put("sub", token.getResourceOwnerId());
    UserInfoClaims claims = new UserInfoClaims(response, null);
    return claims;
}

/**
 * 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> 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
}
}

```

3.2.2. 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 Section 3.2.1, "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.

3.2.3. 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, navigate to Realms > *Realm Name* > Services, click OAuth2 Provider, and enter the class name of your scopes plugin to the Scope Implementation Class field.
- To configure your plugin as the default for new OAuth 2.0 providers, add the class name of your scopes plugin. Navigate to Configure > Global Services, click OAuth2 Provider, and set Scope Implementation Class.

3.2.4. 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 \
  &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.

3.3. Creating a Custom Authentication Module

This section 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.

3.3.1. 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 Section 3.3.4, "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 Section 3.3.5, "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 Section 3.3.2, "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 Section 3.3.6, "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 Section 3.3.3, "Sample Auth Callbacks" for details.

3.3.2. 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
```

3.3.3. 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.

The value of the `moduleName` property in the callbacks file must match your custom authentication module's class name. Observe that the module name in Example 3.1, "Sample Auth Callbacks File", `SampleAuth`, matches the class name in Example 3.2, "Sample Authentication Module Code".

Example 3.1. Sample Auth Callbacks File

The following is the `SampleAuth.xml` callbacks file.


```

<!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.

3.3.4. 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.

Example 3.2. Sample Authentication Module Code

The implementation, `SampleAuth.java`, is shown below.

```
/**
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS HEADER.
 *
 * Copyright (c) 2011-2018 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 legal/CDDLv1.0.txt.
 * 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 legal/CDDLv1.0.txt.
 * 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;

/**
 * SampleAuth authentication module example.
 *
 * If you create your own module based on this example, you must modify all
 * occurrences of "SampleAuth" in addition to changing the name of the class.
 *
 * Please refer to OpenAM documentation for further information.
 *
 * Feel free to look at the code for authentication modules delivered with
```

```

* OpenAM, as they implement this same API.
*/
public class SampleAuth extends AMLoginModule {

    // Name for the debug-log
    private final static String DEBUG_NAME = "SampleAuth";
    private final static Debug debug = Debug.getInstance(DEBUG_NAME);

    // 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 PASSWORD = "changeit";

    private final static String ERROR_1_USERNAME = "test1";
    private final static String ERROR_2_USERNAME = "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;

    // Errors properties
    private final static String SAMPLE_AUTH_ERROR_1 = "sampleauth-error-1";
    private final static String SAMPLE_AUTH_ERROR_2 = "sampleauth-error-2";

    private Map<String, String> options;
    private ResourceBundle bundle;
    private Map<String, String> sharedState;

    public SampleAuth() {
        super();
    }

    /**
     * This method stores service attributes and localized properties for later
     * use.
     * @param subject
     * @param sharedState
     * @param options
     */
    @Override
    public void init(Subject subject, Map sharedState, Map options) {

        debug.message("SampleAuth::init");

        this.options = options;
        this.sharedState = sharedState;
        this.bundle = amCache.getResBundle(amAuthSampleAuth, getLoginLocale());
    }

    @Override
    public int process(Callback[] callbacks, int state) throws LoginException {

        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 = String.valueOf(pc.getPassword());

        //First errorstring is stored in "sampleauth-error-1" property.
        if (ERROR_1_USERNAME.equals(username)) {
            setErrorText(SAMPLE_AUTH_ERROR_1);
            return STATE_ERROR;
        }

        //Second errorstring is stored in "sampleauth-error-2" property.
        if (ERROR_2_USERNAME.equals(username)) {
            setErrorText(SAMPLE_AUTH_ERROR_2);
            return STATE_ERROR;
        }

        if (USERNAME.equals(username) && PASSWORD.equals(password)) {
            debug.message("SampleAuth::process User '{}' " +
                "authenticated with success.", username);
            return ISAuthConstants.LOGIN_SUCCEEDED;
        }

        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, "specificAttribute");

    // Get property from bundle
    String new_hdr = ssa + " " +

```

```

        bundle.getString("sampleauth-ui-login-header");
        substituteHeader(STATE_AUTH, new_hdr);

        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));
    }
}

```

3.3.5. The Sample Auth Principal

The implementation, `SampleAuthPrincipal.java`, is shown below.

```

/**
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS HEADER.
 *
 * Copyright (c) 2011-2018 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 legal/CDDLv1.0.txt.
 * 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 legal/CDDLv1.0.txt.
 * 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;

/**
 * SampleAuthPrincipal represents the user entity.
 */
public class SampleAuthPrincipal implements Principal, Serializable {
    private final static String COLON = " : ";

    private final String name;

    public SampleAuthPrincipal(String name) {

        if (name == null) {
            throw new NullPointerException("illegal null input");
        }

        this.name = name;

```

```
}

/**
 * Return the LDAP username for this SampleAuthPrincipal.
 *
 * @return the LDAP username for this SampleAuthPrincipal
 */
@Override
public String getName() {
    return name;
}

/**
 * Return a string representation of this SampleAuthPrincipal.
 *
 * @return a string representation of this
 *         TestAuthModulePrincipal.
 */
@Override
public String toString() {
    return new StringBuilder().append(this.getClass().getName())
        .append(COLON).append(name).toString();
}

/**
 * Compares the specified Object with this SampleAuthPrincipal
 * for equality. Returns true if the given object is also a
 * SampleAuthPrincipal and the two SampleAuthPrincipal have
 * the same username.
 *
 * @param o Object to be compared for equality with this
 *         SampleAuthPrincipal.
 * @return true if the specified Object is equal equal to this
 *         SampleAuthPrincipal.
 */
@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 SampleAuthPrincipal.
 *
 */
```

```
    * @return a hash code for this SampleAuthPrincipal.  
    */  
    @Override  
    public int hashCode() {  
        return name.hashCode();  
    }  
}
```

3.3.6. 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 derived from the module name. The service name must have the following format:
 - It must start with either `iPlanetAMAuth` or `sunAMAuth`.
 - The module name must follow. The case of the module name must match the case of the class that implements the custom authentication module.
 - It must end with `Service`.

In the Sample Auth service configuration, the module name is `SampleAuth` and 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 must have the following format:
 - It must start with `iplanet-am-auth-`, `sun-am-auth-`, or `forgerock-am-auth-`.
 - The module name must follow, and must appear in lower case if the attribute name starts with `iplanet-am-auth-` or `forgerock-am-auth-`. If the attribute name starts with `sun-am-auth-`, it must exactly match the case of the module name as it appears in the service name.
 - It must end with `-auth-level`.

In the Sample Auth service configuration, the authentication level attribute name is `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) 2011-2018 ForgeRock AS.

  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 legal/CDDLv1.0.txt.
  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 legal/CDDLv1.0.txt.
  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" resourceName="sample">
      <Organization>
        <!-- Specify resourceName for a JSON-friendly property in the REST SMS -->
        <AttributeSchema name="iplanet-am-auth-sampleauth-auth-level" resourceName="authLevel"
          type="single" syntax="number_range" rangeStart="0" rangeEnd="2147483647"
          i18nKey="a500">
          <DefaultValues>
            <Value>1</Value>
          </DefaultValues>
        </AttributeSchema>

        <!-- No need for resourceName when the name is JSON-compatible -->
        <AttributeSchema name="specificAttribute"
          type="single" syntax="string" validator="no" i18nKey="a501" />
      </Organization>
    </Schema>
  </Service>
  <!--
    For Auth Modules, the parent Schema element specifies the REST SMS resourceName,
    and the nested SubSchema must have resourceName="USE-PARENT"
  -->
  <SubSchema name="serverconfig" inheritance="multiple" resourceName="USE-PARENT">
    <AttributeSchema name="iplanet-am-auth-sampleauth-auth-level" resourceName="authLevel"
```



```
type="single" syntax="number_range" rangeStart="0" rangeEnd="2147483647"
i18nKey="a500">
<DefaultValues>
  <Value>1</Value>
</DefaultValues>
</AttributeSchema>

<!-- No need for a DefaultValues element when the default is blank -->
<AttributeSchema name="specificAttribute"
  type="single" syntax="string" validator="no" i18nKey="a501" />

</SubSchema>
</Organization>
</Schema>
</Service>
</ServicesConfiguration>
```

3.3.7. Building and Installing the Sample Auth Module

Build the module with Apache Maven, and install the module in OpenAM.

3.3.7.1. 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*.

3.3.7.2. Installing the Module

Installing the sample authentication module consists of copying the `.jar` file 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/custom*.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 src/main/resources/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(1)` 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
```

3.3.8. 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).

Log in to the OpenAM console as an administrator, such as `amadmin`, and browse to Realms > Top Level Realm > Authentication > Modules. Click Add Module to create an instance of the Sample Authentication Module. Name the module `Sample`.


New Module

Name

Type

- RADIUS
- SAE
- SAML2
- Sample Authentication Module
- Scripted Module
- SecurID
- Windows Desktop SSO
- Windows NT

Click Create, and then configure the authentication module as appropriate.

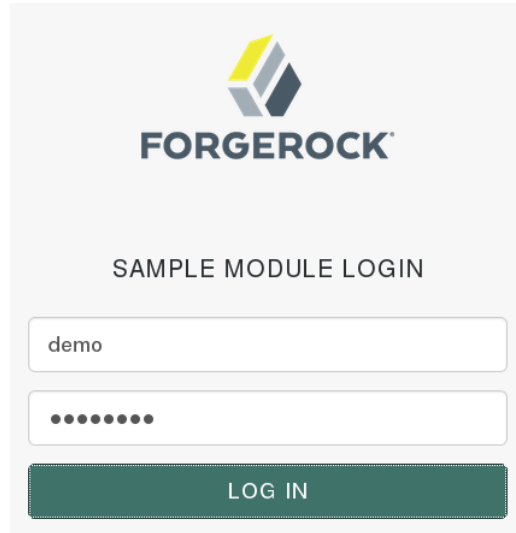
 SAMPLE
Sample

Authentication Level

Service Specific Attribute

Now that the module is configured, log out of the 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/XUI/#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 log out of OpenAM console, and then try to authenticate as the non-existent user `test123` to see what the error handling looks like to the user.

3.4. Customizing Session Quota Exhaustion Actions

This section demonstrates a custom session quota exhaustion action plugin. OpenAM calls a session quota exhaustion action plugin when a user tries to open more stateful sessions than their quota allows. Note that session quotas are not available for stateless sessions.

You only need a custom session quota exhaustion action plugin if the built-in actions are not flexible enough for your deployment. See Procedure 2.18, "To Configure Session Quotas and Exhaustion Actions" in the *Administration Guide*.

3.4.1. 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.ipplanet.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 section simply removes the first session it finds as the session quota exhaustion action.

```
package org.forgerock.openam.examples.quotaexhaustionaction;

import com.ipianet.dpro.session.Session;
import com.ipianet.dpro.session.SessionException;
import com.ipianet.dpro.session.SessionID;
import com.ipianet.dpro.session.service.InternalSession;
import com.ipianet.dpro.session.service.QuotaExhaustionAction;
import com.ipianet.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] --- 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-13.5.2.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-13.5.2.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 the OpenAM Console, navigate to Configure > Global Services, click Session, scroll to Resulting behavior if session quota exhausted, and then choose an option.

Before moving to your test and production environments, be sure to add your `.jar` file and updates to `amSession.properties` into a custom `.war` file that you can then deploy. You must still update the Session service configuration in order to use your custom session quota exhaustion action.

3.4.2. 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

3.4.3. 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.

3.5. 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 section explains how to customize policy evaluation for deployments with particular requirements not met by built-in OpenAM functionality.

This section shows how to build and use a custom policy plugin that implements a custom subject condition, a custom environment condition, and a custom resource attribute.

3.5.1. 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`.

3.5.2. Building the Sample Plugin

Follow the steps in this procedure to build the sample plugin:

Procedure 3.4. 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. Check out the `master` branch of the OpenAM source.
3. Build the module using Apache Maven:

```
$ cd /path/to/openam-source/openam-samples
$ cd policy-evaluation-plugin
$ mvn install
[INFO] Scanning for projects...
[INFO]
[INFO] -----
[INFO] Building policy-evaluation-plugin 13.5.2-4
[INFO] -----
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @
[INFO] policy-evaluation-plugin ---
...
[INFO] Building jar: ../target/policy-evaluation-plugin-13.5.2-4.jar
[INFO]
...
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 5.264 s
[INFO] Finished at: 2016-05-11T19:39:23+02:00
[INFO] Final Memory: 32M/85M
[INFO] -----
```

4. Copy the `.jar` to the `WEB-INF/lib` directory where you deployed OpenAM:

```
$ cp target/*.jar /path/to/tomcat/webapps/openam/WEB-INF/lib/
```

5. Edit the `/path/to/tomcat/webapps/openam/XUI/locales/en/translation.json` file to update the user interface to include the custom subject and environment conditions:
 - a. Locate the line that contains the following text:

```
"subjectTypes": {
```

- b. Insert the following text after the line you located in the previous step:

```
"SampleSubject": {  
  "title": "Sample Subject",  
  "props": {  
    "name": "Name"  
  }  
},
```

- c. Locate the line that contains the following text:

```
"conditionTypes": {
```

- d. Insert the following text after the line you located in the previous step:

```
"SampleCondition": {  
  "title": "Sample Condition",  
  "props": {  
    "nameLength": "Minimum username length"  
  }  
},
```

6. If you require additional translations under `/path/to/tomcat/webapps/openam/XUI/locales`, modify other `translation.json` files as needed.
7. Clear your browser's cache and restart your browser.

Clearing the cache and refreshing the browser is required when you modify the `translation.json` file.

8. Restart OpenAM or the container in which it runs.

3.5.3. Adding Custom Policy Implementations to Existing Policy Sets

In order to use your custom policy in existing policy sets, you must update the policy sets. Note that you cannot update a policy set that already has policies configured. When there are already policies configured for a policy set, you must instead first delete the policies, and then update the policy set.

Update the `iPlanetAMWebAgentService` policy set in the top level realm of a fresh installation. First, authenticate to OpenAM as the `amadmin` user:

```
$ curl \
--request POST \
--header "Content-Type: application/json" \
--header "X-OpenAM-Username: amadmin" \
--header "X-OpenAM-Password: password" \
--data "{}" \
https://openam.example.com:8443/openam/json/authenticate
{"tokenId":"AQIC5wM2...", "successUrl":"/openam/console"}
```

Then update the `iPlanetAMWebAgentService` policy set by adding the `SampleSubject` subject condition and the `SampleCondition` environment condition:

```
$ curl \
--request PUT \
--header "iPlanetDirectoryPro: AQIC5wM2..." \
--header "Content-Type: application/json" \
--data '{
  "name": "iPlanetAMWebAgentService",
  "conditions": [
    "LEAuthLevel",
    "Script",
    "AuthenticateToService",
    "SimpleTime",
    "AMIdentityMembership",
    "OR",
    "IPv6",
    "IPv4",
    "SessionProperty",
    "AuthScheme",
    "AuthLevel",
    "NOT",
    "AuthenticateToRealm",
    "AND",
    "ResourceEnvIP",
    "LDAPFilter",
    "OAuth2Scope",
    "Session",
    "SampleCondition"
  ],
  "subjects": [
    "NOT",
    "OR",
    "JwtClaim",
    "AuthenticatedUsers",
    "AND",
    "Identity",
    "NONE",
    "SampleSubject"
  ],
  "applicationType": "iPlanetAMWebAgentService",
  "entitlementCombiner": "DenyOverride"
}' https://openam.example.com:8443/openam/json/applications/iPlanetAMWebAgentService
```

3.5.4. Trying the Sample Subject and Environment Conditions

Using the OpenAM console, add a policy to the `iPlanetAMWebAgentService` policy set in the top level realm that allows HTTP GET access for URLs based on the template `http://www.example.com:80/*` and uses the custom subject and environment conditions.

Create the policy with the following properties:

Table 3.1. Sample Policy Properties

Property	Value
Name	Sample Policy
Resource Type	URL
Resources	Use the <code>*://*:*/*</code> resource template to specify the resource <code>http://www.example.com:80/*</code> .
Actions	Allow GET
Subject Conditions	Add a subject condition of type <code>Sample Subject</code> and a name of <code>demo</code> so that the <code>demo</code> user is the only user who can access the resource.
Environment Conditions	Add an environment condition of type <code>Sample Condition</code> and a minimum username length of 4 so that only users with a username length of 4 characters or greater can access the resource.

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, such as `demo` in the example above. Both calls return `tokenId` values for use in the policy decision request.

```
$ curl \
  --request POST \
  --header "Content-Type: application/json" \
  --header "X-OpenAM-Username: amadmin" \
  --header "X-OpenAM-Password: password" \
  --data "{}" \
  https://openam.example.com:8443/openam/json/authenticate

{"tokenId": "AQIC5wM2LY4Sfcw...", "successUrl": "/openam/console"}

$ curl \
  --request POST \
  --header "Content-Type: application/json" \
  --header "X-OpenAM-Username: demo" \
  --header "X-OpenAM-Password: changeit" \
  --data "{}" \
  https://openam.example.com:8443/openam/json/authenticate

{"tokenId": "AQIC5wM2LY4Sfcy...", "successUrl": "/openam/console"}
```

Use the administrator `tokenId` as the header of the policy decision request, and the user `tokenId` as the subject `ssoToken` value.

```
$ curl \
--request POST \
--header "Content-Type: application/json" \
--header "iPlanetDirectoryPro: AQIC5wM2LY4Sfcw..." \
--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 returned from the policy evaluation call are set in accordance with the policy.

3.5.5. 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, list the resource type for the **URL** resource type to obtain its UUID, and then update your policy to return a **test** attribute:

```
$ curl \
--request GET \
--header "iPlanetDirectoryPro: AQIC5wM2..." \
https://openam.example.com:8443/openam/json/resourcetypes?_queryFilter=name%20eq%20%22URL%22
{
  "result":[
    {
      "uuid":"URL-resource-type-UUID",
      "name":"URL",
      "description":"The built-in URL Resource Type available to OpenAM Policies.",
      "patterns":["*://*:*/*", "*://*:*/*?*" ],
      ...
    }
  ],
  "resultCount":1,
  "pagedResultsCookie":null,
  "totalPagedResultsPolicy":"NONE",
  "totalPagedResults":-1,
  "remainingPagedResults":0
}
```

```
$ curl \
--request PUT \
--header "iPlanetDirectoryPro: AQIC5wM2LY4Sfcw..." \
--header "Content-Type: application/json" \
--data '{
  "name": "Sample Policy",
  "active": true,
  "description": "Try sample policy plugin",
  "resourceTypeUuid": "URL-resource-type-UUID",
  "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 "Content-Type: application/json" \
--header "iPlanetDirectoryPro: AQIC5wM2LY4Sfcw..." \
--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": {}
  }
]
```

3.5.6. 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 ...
```

3.6. 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.

3.6.1. About the Identity Repository Plugin

This section describes how to create a custom identity repository plugin. OpenAM includes built-in support for LDAP 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`.

3.6.1.1. 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.

3.6.1.2. 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.

3.6.1.3. 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.

3.6.2. 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, SS0Exception

public Map getAttributes(
    SS0Token token,
    IdType type,
    String name,
    Set attrNames
) throws IdRepoException, SS0Exception

public Map getBinaryAttributes(
    SS0Token token,
    IdType type,
    String name,
    Set attrNames
) throws IdRepoException, SS0Exception

public RepoSearchResults search(
    SS0Token token,
    IdType type,
    String pattern,
    Map avPairs,
    boolean recursive,
    int maxResults,
    int maxTime,
    Set returnAttrs
) throws IdRepoException, SS0Exception

public RepoSearchResults search(
    SS0Token token,
    IdType type,
    String pattern,
    int maxTime,
    int maxResults,
    Set returnAttrs,
    boolean returnAllAttrs,
    int filterOp,
    Map avPairs,
    boolean recursive
) throws IdRepoException, SS0Exception
    
```

Edit

OpenAM calls the following methods to update a subject in the identity repository.

```

public void setAttributes(
    SS0Token token,
    IdType type,
    String name,
    Map attributes,
    boolean isAdd
) throws IdRepoException, SS0Exception

public void setBinaryAttributes(
    SS0Token token,
    IdType type,
    String name,
    Map attributes,
    boolean isAdd
) throws IdRepoException, SS0Exception
    
```

```

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.

3.6.3. Identity Repository Plugin Deployment

When you build your IdRepo plugin, include `openam-core-13.5.2.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-13.5.2.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-13.5.2.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
```

Log in to the OpenAM console as administrator, then browse to `Realms > 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 Realms > *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
```

Chapter 4

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.

4.1. 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.

Two issues should be considered when writing a post authentication plugin for an OpenAM deployment that uses stateless sessions:

Cookie size

You can set an unlimited number of session properties in a post authentication plugin. When OpenAM creates a stateless session, it writes the session properties into the session cookie, increasing the size of the cookie. Very large session cookies can exceed browser limitations. Therefore, when implementing a post authentication plugin in a deployment with stateless sessions, be sure to monitor the session cookie size and verify that you have not exceeded browser cookie size limits.

For more information about stateless session cookies, see Section 9.2, "Session Cookies" in the *Administration Guide*.

Cookie security

The OpenAM administrator secures custom session properties residing on the OpenAM server for stateful sessions by using firewalls and other typical security techniques.

However, when using stateless sessions, custom session properties are written in cookies and reside on end users' systems. Cookies can be long-lasting and might represent a security issue if

any session properties are of a sensitive nature. When developing a post authentication plugin for a deployment that uses stateless sessions, be sure that you are aware of the measures securing the session contained within the cookie.

For more information about stateless session cookie security, see Section 9.2.1, "Stateless Session Cookie Security" in the *Administration Guide*.

This section explains how to create a post authentication plugin.

4.1.1. 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.

4.1.2. 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.ipplanet.sso.SSOException;
import com.ipplanet.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.

4.1.3. Configuring Your Post Authentication Plugin

You can associate post authentication plugins with realms or services (authentication chains). Where you configure the plugin depends on the scope to which the plugin should apply:

- Plugins configured at the realm level are executed when authenticating to any authentication chain in the realm, provided the authentication chain does not have an associated plugin.
- Plugins configured at the service level are executed if that authentication chain is used for authentication. Any plugins configured at the realm level will not execute.

In OpenAM Console, navigate to `Realms > Realm Name > Authentication > Settings > Post Authentication Processing`. 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.
```

4.1.4. 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.

4.2. Extending UMA Workflow with Extension Points

OpenAM provides a number of extension points for extending the UMA workflow. These extension points are provided as filters and are dynamically loaded by using the Java `ServiceLoader` framework during the UMA workflow.

The extension points available are described in the sections below:

- Section 4.2.1, "Resource Set Registration Extension Point"
- Section 4.2.2, "Permission Request Extension Point"
- Section 4.2.3, "Authorization Request Extension Point"
- Section 4.2.4, "Resource Sharing Extension Point"

4.2.1. Resource Set Registration Extension Point

OpenAM provides the `ResourceRegistrationFilter` extension point, which can be used to extend UMA resource set registration functionality.

Table 4.1. Resource Set Registration Extension Methods

Method	Parameters	Description
<code>beforeResourceRegistration</code>	<code>resourceSet</code> (type: <code>ResourceSetDescription</code>)	Invoked before a resource set is registered in the backend. Changes made to the <code>resourceSet</code> object at this stage <i>will</i> be persisted.
<code>afterResourceRegistration</code>	<code>resourceSet</code> (type: <code>ResourceSetDescription</code>)	Invoked after a resource set is registered in the backend. Changes made to the <code>resourceSet</code> object at this stage <i>will not</i> be persisted.

4.2.2. Permission Request Extension Point

OpenAM provides the `PermissionRequestFilter` extension point, which can be used to extend UMA permission request functionality.

Table 4.2. Permission Request Extension Methods

Method	Parameters	Description
<code>onPermissionRequest</code>	<code>resourceSet</code> (type: <code>ResourceSetDescription</code>) <code>requestedScopes</code> (type: <code>Set<String></code>) <code>requestingClientId</code> (type: <code>String</code>)	Invoked before a permission request is created.

4.2.3. Authorization Request Extension Point

OpenAM provides the `RequestAuthorizationFilter` extension point, which can be used to extend UMA authorization functionality.

Table 4.3. Authorization Request Extension Methods

Method	Parameters	Description
<code>beforeAuthorization</code>	<code>permissionTicket</code> (type: <code>PermissionTicket</code>) <code>requestingParty</code> (type: <code>Subject</code>) <code>resourceOwner</code> (type: <code>Subject</code>)	Invoked before authorization of a request is attempted.

Method	Parameters	Description
		Throws <code>UmaException</code> if authorization of the request should not be attempted.
<code>afterAuthorization</code>	<p><code>isAuthorized</code> (type: <code>boolean</code>)</p> <p><code>permissionTicket</code> (type: <code>PermissionTicket</code>)</p> <p><code>requestingParty</code> (type: <code>Subject</code>)</p> <p><code>resourceOwner</code> (type: <code>Subject</code>)</p>	<p>Invoked before authorization of a request is attempted.</p> <p>If the authorization request was successful, <code>isAuthorized</code> will be <code>true</code>.</p>

4.2.4. Resource Sharing Extension Point

OpenAM provides the `ResourceDelegationFilter` extension point, which can be used to extend UMA resource sharing functionality.

Table 4.4. Resource Sharing Extension Methods

Method	Parameters	Description
<code>beforeResourceShared</code>	<code>umaPolicy</code> (type: <code>UmaPolicy</code>)	<p>Invoked before creating a sharing policy for a resource.</p> <p>Changes to the <code>umaPolicy</code> object at this stage <i>will</i> be persisted.</p> <p>Throws <code>ResourceException</code> if a sharing policy for the resource should not be created.</p>
<code>afterResourceShared</code>	<code>umaPolicy</code> (type: <code>UmaPolicy</code>)	<p>Invoked after creating a sharing policy for a resource.</p> <p>Changes to the <code>umaPolicy</code> object at this stage <i>will not</i> be persisted.</p>
<code>beforeResourceSharedModification</code>	<p><code>currentUmaPolicy</code> (type: <code>UmaPolicy</code>)</p> <p><code>updatedUmaPolicy</code> (type: <code>UmaPolicy</code>)</p>	<p>Invoked before altering the sharing policy of a resource.</p> <p>Changes to the <code>updatedUmaPolicy</code> object at this stage <i>will</i> be persisted.</p> <p>Throws <code>ResourceException</code> if the sharing policy of the</p>

Method	Parameters	Description
		resource should not be modified.
<code>onResourceSharedDeletion</code>	<code>umaPolicy</code> (type: <code>UmaPolicy</code>)	<p>Invoked before deleting the sharing policy of a resource.</p> <p>Throws <code>ResourceException</code> if the sharing policy of the resource should not be deleted.</p>
<code>beforeQueryResourceSets</code>	<code>userId</code> (type: <code>String</code>) <code>queryFilter</code> (type: <code>QueryFilter<JsonPointer></code>)	<p>Invoked before querying the resource sets owned or shared with a user.</p> <p>The <code>userId</code> parameter provides the ID of the user making the query request.</p> <p>The <code>queryFilter</code> parameter provides the incoming request query filter.</p> <p>Returns a <code>QueryFilter</code> that can be used to return the user's resource sets.</p>

Chapter 5

Scripting OpenAM

This chapter explains how to use scripting to exact fine control over various aspects of OpenAM.

You can use scripts for client-side and server-side authentication, policy conditions, and handling OpenID Connect claims.

This chapter covers the following topics:

- The Scripting Environment
- The Scripting API
- Using the Default Scripts

For information on managing scripts, see [Chapter 22, "Managing Scripts"](#) in the *Administration Guide* and [Section 2.1.17, "RESTful Script Management"](#).

5.1. The Scripting Environment

This section introduces how OpenAM executes scripts, and covers thread pools and security configuration.

You can use scripts to modify default OpenAM behavior in the following situations, also known as *contexts*:

Client-side Authentication

Scripts that are executed on the client during authentication.

Note

Client-side scripts must be in JavaScript.

Server-side Authentication

Scripts are included in an authentication module and are executed on the server during authentication.

Scripted authentication modules are an alternative to developing custom authentication modules by using Java as described in Section 3.3, "Creating a Custom Authentication Module". A scripted authentication module allows you to customize default authentication behavior by adding Groovy or JavaScript code to the module configuration.

To see an example server-side authentication script, see Section 5.3.1, "Default Server-side Authentication Script".

Policy Condition

Scripts used as conditions within policies.

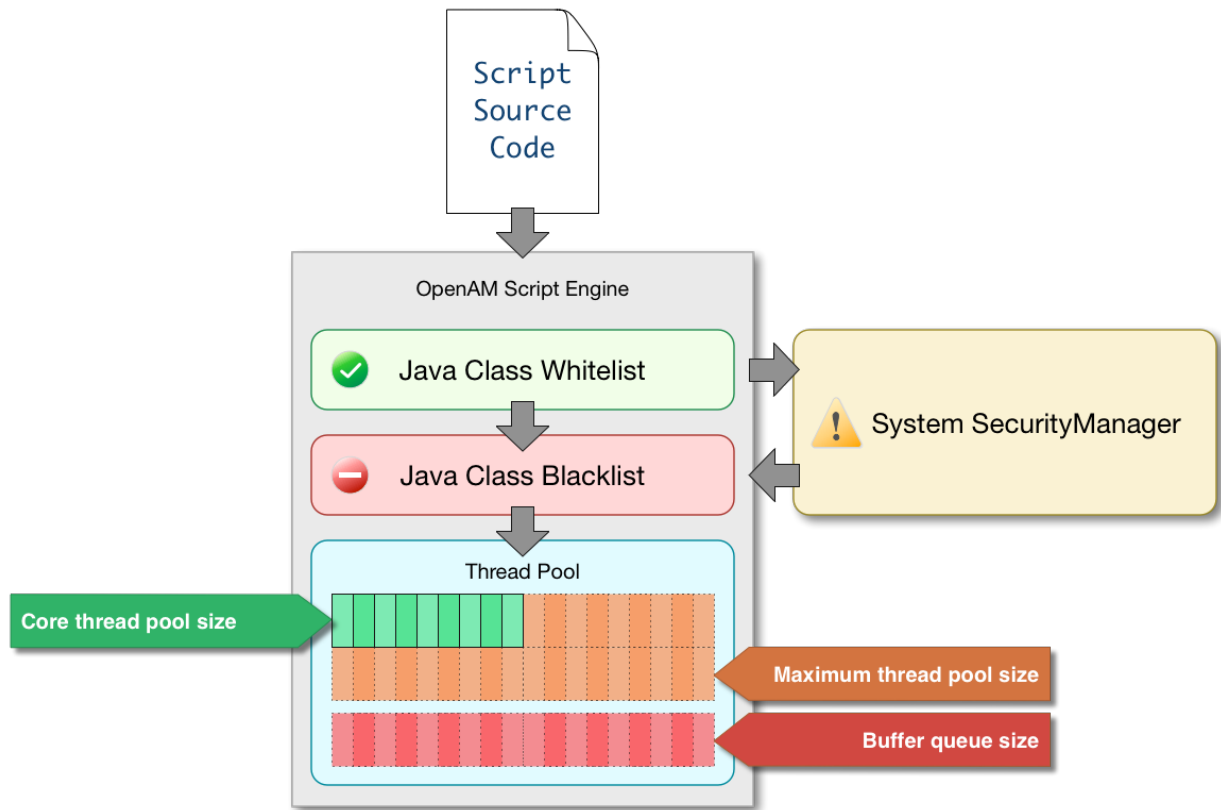
To see an example policy condition script, see Section 5.3.2, "Default Policy Condition Script".

OIDC Claims

Scripts that gather and populate the claims in a request when issuing an ID token or making a request to the `userinfo` endpoint.

OpenAM implements a configurable scripting engine for each of the context types that are executed on the server.

The scripting engines in OpenAM have two main components: security settings, and the thread pool.



5.1.1. Security

OpenAM scripting engines provide security features for ensuring that malicious Java classes are not directly called. The engines validate scripts by checking all directly-called Java classes against a configurable blacklist and whitelist, and, optionally, against the JVM SecurityManager, if it is configured.

Whitelists and blacklists contain class names that are allowed or denied execution respectively. Specify classes in whitelists and blacklists by name or by using regular expressions.

Classes called by the script are checked against the whitelist first, and must match at least one pattern in the list. The blacklist is applied after the whitelist, and classes matching any pattern are disallowed.

You can also configure the scripting engine to make an additional call to the JVM security manager for each class that is accessed. The security manager throws an exception if a class being called is not allowed to execute.

For more information on configuring script engine security, see Section 1.4.19, "Scripting" in the *Reference*.

Important Points About Script Engine Security

The following points should be considered when configuring the security settings within each script engine:

The scripting engine only validates directly accessible classes.

The security settings only apply to classes that the script *directly* accesses. If the script calls `Foo.a()` and then that method calls `Bar.b()`, the scripting engine will be unable to prevent it. You must consider the whole chain of accessible classes.

Note

Access includes actions such as:

- Importing or 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.

Potentially dangerous Java classes are blacklisted by default.

All Java reflection classes (`java.lang.Class`, `java.lang.reflect.*`) are blacklisted by default to avoid bypassing the security settings.

The `java.security.AccessController` class is also blacklisted by default to prevent access to the `doPrivileged()` methods.

Caution

You should not remove potentially dangerous Java classes from the blacklist.

The whitelists and blacklists match class or package names only.

The whitelist and blacklist patterns apply only to the exact class or package names involved. The script engine does not know anything about inheritance, so it is best to whitelist known, specific classes.

5.1.2. Thread Pools

Each script is executed in an individual thread. Each scripting engine starts with an initial number of threads available for executing scripts. If no threads are available for execution, OpenAM creates a new thread to execute the script, until the configured maximum number of threads is reached.

If the maximum number of threads is reached, pending script executions are queued in a number of buffer threads, until a thread becomes available for execution. If a created thread has completed script execution and has remained idle for a configured amount of time, OpenAM terminates the thread, shrinking the pool.

For more information on configuring script engine thread pools, see Section 1.4.19, "Scripting" in the *Reference*.

5.2. The Scripting API

Client-side scripts have access only to the user agent API. The functionality provided by each user agent is different, refer to the API provided by your user agent for more information.

5.2.1. Global API Functionality

This section covers functionality available to each of the server-side script types.

Global API functionality includes:

- Accessing HTTP Services
- Debug Logging

5.2.1.1. 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` object.

Table 5.1. HTTP Client Methods

Method	Parameters	Return Type	Description
<code>httpClient.get</code>	<code>URI</code> (type: <code>String</code>) <code>Request Data</code> (type: <code>Map</code>)	<code>HttpClientResponse</code>	Perform an HTTP GET on the specified URI with the specified request data and return the response retrieved.
<code>httpClient.post</code>	<code>URI</code> (type: <code>String</code>) <code>Body</code> (type: <code>String</code>) <code>Request Data</code> (type: <code>Map</code>)	<code>HttpClientResponse</code>	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 in which 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": "E8cDkvlad83kd...KDodkIEIx*DLEDLK...JKD09d"
    }
  ],
  headers:[
    {
      "field": "Content-type",
      "value": "application/json"
    }
  ]
});
```

An example `requestData` JavaScript object using POST follows:

```
var response = httpClient.post("http://example.com:8080/openam/json/authenticate", "{
  "authId": "eyJiYWxnIjogIkhTMjU2IiwgInR5cCI6ICJqd3Q0iIH0.eyJhb3RrIjogIm03ODVzN2xsbmR1bjZvbGZ1MHZhOGVtYTQxIiwgInNlc3Npb25JZCI6ICJBUU'LDNXdNMkxZNFNmY3'LEeDY3QnBpdzJtRU9rUzNpLWhfNDdRWlMwNHBEN1ppdy4qQUFKVFNRQU'NNREVBQWx0TEFCUXR0ak15TURjNU1UZzROVFUwTXpnNE5qRTNNQS4uKiIsICJyZWZsbSI6ICJkYz1vcG'VuYW0sZGM9Zm9yZ2Vyb2NrlGRjPW9yZyIgZGF0. VDRqaekQuXBm2'LNi29hfwVADLxjepezuo0241VNDsIM",
  "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 Data Store 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 Section 2.5.24, "Hints for the Scripted Authentication Module" in the *Administration Guide*.

Server-side scripts can access response data by using the methods listed in the table below.

Table 5.2. HTTP Client Response Methods

Method	Parameters	Return Type	Description
<code>HttpClientResponse.getCookies</code>	Void	<code>Map<String, String></code>	Get the cookies for the returned response, if any exist.
<code>HttpClientResponse.getEntity</code>	Void	<code>String</code>	Get the entity of the returned response.
<code>HttpClientResponse.getHeaders</code>	Void	<code>Map<String, String></code>	Get the headers for the returned response, if any exist.
<code>HttpClientResponse.getReasonPhrase</code>	Void	<code>String</code>	Get the reason phrase of the returned response.
<code>HttpClientResponse.getStatusCode</code>	Void	<code>Integer</code>	Get the status code of the returned response.

Method	Parameters	Return Type	Description
<code>HttpClientResponse.hasCookies</code>	Void	Boolean	Indicate whether the returned response had any cookies.
<code>HttpClientResponse.hasHeaders</code>	Void	Boolean	Indicate whether the returned response had any headers.

5.2.1.2. Debug Logging

Server-side scripts can write messages to OpenAM debug logs by using the `logger` object.

OpenAM does not log debug messages from scripts by default. You can configure OpenAM to log such messages by setting the debug log level for the `amScript` service. For details, see Section 24.4.3, "Debug Logging By Service" in the *Administration Guide*.

The following table lists the `logger` methods.

Table 5.3. Logger Methods

Method	Parameters	Return Type	Description
<code>logger.error</code>	<i>Error Message</i> (type: String)	Void	Write <i>Error Message</i> to OpenAM debug logs if ERROR level logging is enabled.
<code>logger.errorEnabled</code>	Void	Boolean	Return <code>true</code> when ERROR level debug messages are enabled.
<code>logger.message</code>	<i>Message</i> (type: String)	Void	Write <i>Message</i> to OpenAM debug logs if MESSAGE level logging is enabled.
<code>logger.messageEnabled</code>	Void	Boolean	Return <code>true</code> when MESSAGE level debug messages are enabled.
<code>logger.warning</code>	<i>Warning Message</i> (type: String)	Void	Write <i>Warning Message</i> to OpenAM debug logs if WARNING level logging is enabled.
<code>logger.warningEnabled</code>	Void	Boolean	Return <code>true</code> when WARNING level debug messages are enabled.

5.2.2. Authentication API Functionality

This section covers the available functionality when Scripting authentication modules use client-side and server-side authentication script types.

Authentication API functionality includes:

- Accessing Authentication State

- Accessing Profile Data
- Accessing Client-Side Script Output Data
- Accessing Request Data

5.2.2.1. Accessing Authentication State

OpenAM passes `authState` and `sharedState` objects 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`.

The following authentication modules set the login name of the user:

- Anonymous
- Certificate
- Data Store
- Federation
- HTTP Basic
- JDBC
- LDAP
- Membership
- RADIUS
- SecurID,
- Windows Desktop SSO
- Windows NT

5.2.2.2. Accessing Profile Data

Server-side authentication scripts can access profile data through the methods of the `idRepository` object.

Table 5.4. Profile Data Methods

Method	Parameters	Return Type	Description
<code>idRepository.getAttribute</code>	<i>User Name</i> (type: String) <i>Attribute Name</i> (type: String)	Set	Return the values of the named attribute for the named user.
<code>idRepository.setAttribute</code>	<i>User Name</i> (type: String) <i>Attribute Name</i> (type: String) <i>Attribute Values</i> (type: Array)	Void	Set the named attribute as specified by the attribute value for the named user, and persist the result in the user's profile.
<code>idRepository.addAttribute</code>	<i>User Name</i> (type: String) <i>Attribute Name</i> (type: String) <i>Attribute Value</i> (type: String)	Void	Add an attribute value to the list of attribute values associated with the attribute name for a particular user.

5.2.2.3. Accessing 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.

5.2.2.4. 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 Table 5.1, "HTTP Client Methods") and contains information about the original authentication request made by the user.

Table 5.5. Request Data Methods

Method	Parameters	Return Type	Description
<code>requestData.getHeader</code>	<i>Header Name</i> (type: String)	String	Return the String value of the named request header, or <code>null</code> if parameter is not set.
<code>requestData.getHeaders</code>	<i>Header Name</i> (type: String)	String[]	Return the array of String values of the named request header, or <code>null</code> if parameter is not set.
<code>requestData.getParameter</code>	<i>Parameter Name</i> (type: String)	String	Return the String value of the named request parameter, or <code>null</code> if parameter is not set.

Method	Parameters	Return Type	Description
<code>requestData.getParameters</code>	<i>Parameter Name</i> (type: <code>String</code>)	<code>String[]</code>	Return the array of <code>String</code> values of the named request parameter, or <code>null</code> if parameter is not set.

5.2.3. Authorization API Functionality

This section covers functionality available when scripting authorization using the policy condition script context type.

5.2.3.1. Accessing Authorization State

Server-side scripts can access the current authorization state through the following objects:

Table 5.6. Authorization State Objects

Object	Type	Description
<code>authorized</code>	<code>Boolean</code>	Return <code>true</code> if the authorization is currently successful, or <code>false</code> if authorization has failed. Server-side scripts must set a value for <code>authorized</code> before completing.
<code>environment</code>	<code>Map<String, Set<String>></code>	Describe the environment passed from the client making the authorization request. For example, the following shows a simple <code>environment</code> map with a single entry: <pre> "environment": { "IP": ["127.0.0.1"] } </pre>
<code>resourceURI</code>	<code>String</code>	Specify the URI of the resource to which authorization is being requested.
<code>username</code>	<code>String</code>	Specify the user ID of the subject that is requesting authorization.

5.2.3.2. Accessing Profile Data

Server-side authorization scripts can access profile data of the subject of the authorization request through the methods of the `identity` object.

Note

To access the profile data of the subject, they must be logged in and their SSO token must be available.

Table 5.7. Authorization Script Profile Data Methods

Method	Parameters	Return Type	Description
<code>identity.getAttribute</code>	<i>Attribute Name</i> (type: String)	Set	Return the values of the named attribute for the subject of the authorization request.
<code>identity.setAttribute</code>	<i>Attribute Name</i> (type: String) <i>Attribute Values</i> (type: Array)	Void	Set the named attribute to the values specified by the attribute value for the subject of the authorization request.
<code>identity.addAttribute</code>	<i>Attribute Name</i> (type: String) <i>Attribute Value</i> (type: String)	Void	Add an attribute value to the list of attribute values associated with the attribute name for the subject of the authorization request.
<code>identity.store</code>	None	Void	Commit any changes to the identity repository. Caution You must call <code>store()</code> otherwise changes will be lost when the script completes.

5.2.3.3. Accessing Session Data

Server-side authorization scripts can access session data of the subject of the authorization request through the methods of the `session` object.

Note

To access the session data of the subject, they must be logged in and their SSO token must be available.

Table 5.8. Authorization Script Session Methods

Method	Parameters	Return Type	Description
<code>session.getProperty</code>	<i>Property Name</i> (type: String)	String	Retrieve properties from the session associated with the subject

Method	Parameters	Return Type	Description
			of the authorization request. For example, <code>AuthLevel</code> .

5.2.3.4. Setting Authorization Responses

Server-side authorization scripts can return information in the response to an authorization request.

Table 5.9. Authorization Script Response Methods

Method	Parameters	Return Type	Description
<code>responseAttributes.put</code>	<code>Attribute Name</code> (type: <code>String</code>) <code>Attribute Values</code> (type: <code>Array</code>)	<code>Void</code>	Add an attribute to the response to the authorization request.
<code>advice.put</code>	<code>Advice Key</code> (type: <code>String</code>) <code>Advice Values</code> (type: <code>Array</code>)	<code>Void</code>	Add advice key-value pairs to the response to a failing authorization request.
<code>ttl</code>	<code>TTL Value</code> (type: <code>Integer</code>)	<code>Void</code>	Add a time-to-live value to the response to a successful authorization, after which the decision is no longer valid.

5.2.4. OIDC Claims API Functionality

This section covers functionality available when scripting OIDC claim handling using the OIDC claims script context type.

5.2.4.1. Accessing OpenID Connect Requests

Server-side scripts can access the OpenID Connect request through the following objects:

Table 5.10. OIDC Request Objects

Object	Type	Description
<code>scopes</code>	<code>Set<String></code>	Contains a set of the requested scopes. For example:

Object	Type	Description
		<pre>["profile", "openid"]</pre>
<code>identity</code>	<code>Class</code>	<p>Contains a representation of the identity of the resource owner.</p> <p>For more details, see the <code>com.sun.identity.idm.AMIdentity</code> class in the OpenAM Javadoc.</p>
<code>session</code>	<code>Class</code>	<p>Contains a representation of the user's session object if the request contained a session cookie.</p> <p>For more details, see the <code>com.ipplanet.sso.SSOToken</code> class in the OpenAM Javadoc.</p>
<code>claims</code>	<code>Map<String, Object></code>	<p>Contains a map of the claims the server provides by default. For example:</p> <pre>{ "sub": "248289761001", "updated_at": "1450368765" }</pre>
<code>requestedClaims</code>	<code>Map<String, Set<String>></code>	<p>Contains requested claims if the <code>claims</code> query parameter is used in the request and <code>Enable "claims parameter supported"</code> is checked in the OAuth2 provider service configuration, otherwise is empty.</p> <p>For more information see "Requesting Claims using the "claims" Request Parameter" in the <i>OpenID Connect Core 1.0</i> specification.</p> <p>Example:</p> <pre>{ "given_name": { "essential": true, "values": ["Demo User", "D User"] }, "nickname": null, "email": { "essential": true } }</pre>

5.3. Using the Default Scripts

This section covers the default scripts provided in OpenAM. These scripts act as templates for creating your own scripts. They are global and can be used in any realm, and cannot be deleted.

Warning

Editing a default script will affect every authentication module, policy condition, or OIDC claim configuration that uses the script.

5.3.1. Default Server-side Authentication Script

This section demonstrates how to use the default server-side authentication script in a Scripted Authentication module.

The default server-side authentication script only authenticates a subject when the current time on the OpenAM server is between 09:00 and 17:00. The script also uses the `logger` and `httpClient` functionality provided in the scripting API.

To examine the contents of the default server-side authentication script in the OpenAM console browse to Realms > Top Level Realm > Scripts, and then click Scripted Module - Server Side.

For more information on the functions available for use in server-side authentication scripts, see Section 5.2, "The Scripting API".

5.3.1.1. Preparing OpenAM

OpenAM requires a small amount of configuration before trying the example server-side authentication script. You must first create a Scripted authentication module, and then include it in an authentication chain, which can then be used when logging in to OpenAM.

The procedures in this section are:

- Procedure 5.1, "To Create a Scripted Authentication Module that Uses the Default Server-side Authentication Script"
- Procedure 5.2, "To Create an Authentication Chain that Uses a Scripted Authentication Module"

Procedure 5.1. To Create a Scripted Authentication Module that Uses the Default Server-side Authentication Script

In this procedure create a Scripted Authentication module, and link it to the default server-side authentication script.

1. Log in as an OpenAM administrator, for example `amadmin`.
2. Click Realms > Top Level Realm > Authentication > Modules.

3. On the Authentication Modules page, click Add Module.
4. On the New Module page, enter a module name, such as `myScriptedAuthModule`, in the Type drop-down menu, select `Scripted Module`, and then click Create.
5. On the module configuration page:
 - a. Uncheck the Client-side Script Enabled checkbox.
 - b. In the Server-side Script drop-down menu, select `Scripted Module - Server Side`.
 - c. Click Save Changes.

Procedure 5.2. To Create an Authentication Chain that Uses a Scripted Authentication Module

In this procedure create an authentication chain that uses a Data Store authentication module and the Scripted authentication module created in the previous procedure.

1. Log in as an OpenAM administrator, for example `amadmin`.
2. Click Realms > Top Level Realm > Authentication > Chains.
3. On the Authentication Chains page, click Add Chain.
4. On the Add Chain page, enter a name, such as `myScriptedChain`, and then click Create.
5. On the Edit Chain tab, click Add a Module.
6. In the New Module dialog box:
 - a. In the Select Module drop-down menu, select `DataStore`.
 - b. In the Select Criteria drop-down menu, select `Required`.
 - c. Click OK.

Note

The Data Store authentication module checks the user credentials, whereas the Scripted authentication module does not check credentials, but instead only checks that the authentication request is processed during working hours. Without the Data Store module, the username in the Scripted authentication module cannot be determined. Therefore, do not configure the Scripted authentication module (server-side script) as the *first* module in an authentication chain, because it needs a username.

7. On the Edit Chain tab, click Add Module.
8. In the New Module dialog box:
 - a. In the Select Module drop-down menu, select the Scripted Module from the previous procedure, for example `myScriptedAuthModule`.

- b. In the Select Criteria drop-down menu, select **Required**.
- c. Click OK.

The resulting chain resembles the following:

The screenshot shows the 'CHAINS' configuration page for 'myScriptedChain'. The interface includes a 'Delete' button, an 'Edit Chain' tab, and a 'Settings' tab. A '+ Add Module' button is visible. The chain configuration is shown as a vertical sequence of two modules, numbered 1 and 2. Module 1 is 'DataStore' (Data Store) and Module 2 is 'myScriptedAuthModule' (Scripted Module). Both modules have a 'Required' dropdown menu and an 'Options 0' indicator. Below each module, there are 'FAIL' and 'PASS' status indicators with corresponding icons. At the bottom of the chain configuration, there is a summary: 'Successful authentication requires: At least one PASS flag No FAIL flags'. A 'Save Changes' button is located at the bottom right of the configuration area.

9. On the Edit Chain tab, click Save Changes.

5.3.1.2. Trying the Default Server-side Authentication Script

This section shows how to log in using an authentication chain that contains a Scripted authentication module, which in turn uses the default server-side authentication script.

Procedure 5.3. To Login to OpenAM Using a Chain Containing a Scripted Authentication Module

1. Log out of OpenAM.
2. In a browser, navigate to the OpenAM login URL, and specify the authentication chain created in the previous procedure as the value of the `service` query parameter.

For example:

```
https://openam.example.com:8443/openam/XUI/#login/&service=myScriptedChain
```

3. Log in as user `demo` with password `changeit`.

If login is successful, the user profile page appears. The script will also output messages, such as the following in the `debug/Authentication` log file:

```
amScript:05/08/2015 11:31:21:835 AM CEST: Thread[pool-19-thread-5,5,main]
Starting server-side JavaScript
amScript:05/08/2015 11:31:21:837 AM CEST: Thread[pool-19-thread-5,5,main]
User: demo
amScript:05/08/2015 11:31:21:837 AM CEST: Thread[pool-19-thread-5,5,main]
Current time: 11
amScript:05/08/2015 11:31:21:837 AM CEST: Thread[pool-19-thread-5,5,main]
Authentication allowed!
```

Tip

The default server-side authentication script outputs log messages at the `message` and `error` level.

OpenAM does not log debug messages from scripts by default. You can configure OpenAM to log such messages by setting the debug log level for the `amScript` service. For details, see Section 24.4.3, "Debug Logging By Service" in the *Administration Guide*.

4. (Optional) To test that the script is being used as part of the login process, edit the script to alter the times when authentication is allowed:
 - a. Log out the `demo` user.
 - b. Log in as an OpenAM administrator, for example `amadmin`.
 - c. Click Realms > Top Level Realm > Scripts > Scripted Module - Server Side.
 - d. In the script, swap the values for `START_TIME` and `END_TIME`, for example:

```
var START_TIME = 17;
var END_TIME   = 9; //
```

- e. Click Save.

- f. Repeat steps 1, 2, and 3 above, logging into the module as the `demo` user as before. The authentication result will be the opposite of the previous result, as the allowed times have inverted.

5.3.2. Default Policy Condition Script

This section demonstrates how to use the sample policy condition script as part of an authorization policy. To examine the contents of the sample policy condition script in the OpenAM console browse to Realms > Top Level Realm > Scripts, and then click *Scripted Policy Condition*.

The default policy condition script demonstrates how to access a user's profile information, use that information in HTTP calls, and make a policy decision based on the outcome.

For more information on the functions available for use in policy condition scripts, see Section 5.2, "The Scripting API".

5.3.2.1. Preparing OpenAM

OpenAM requires a small amount of configuration before trying the default policy condition script. The default policy condition script requires that the subject of the policy has an address in their profile. The script compares the address to the country in the resource URL and to the country from which the request originated, as determined by an external GeoIP web service. The `demo` user also requires access to evaluate policies.

The procedures in this section are:

- Procedure 5.4, "To Add an Address to the Demo User"
- Procedure 5.5, "To Allow the Demo User to Evaluate a Policy"
- Procedure 5.6, "To Create a Policy that Uses the Default Policy Condition Script"
- Procedure 5.7, "To Enable Message-level Logging for Policy Evaluation"

Procedure 5.4. To Add an Address to the Demo User

In this procedure, add an address value to the `demo` user's profile. The default policy condition script uses the address when performing policy evaluation.

1. Log in as an OpenAM administrator, for example `amadmin`.
2. Click Realms > Top Level Realm > Subjects.
3. On the User tab, click the `demo` user.
4. In Home Address, enter a valid address. For example:

```
201 Mission St, Suite 2900, San Francisco, CA 94105
```

5. Click Save.

Procedure 5.5. To Allow the Demo User to Evaluate a Policy

In this procedure, add the `demo` user to a group and assign the privilege required to perform policy evaluations.

1. Log in as an OpenAM administrator, for example `amadmin`.
2. Click Realms > Top Level Realm > Subjects.
3. On the Group tab, click New, enter an ID for the group, such as `policyEval`, and then click OK.
4. On the User tab:
 - a. Click the `demo` user.
 - b. Click the Group tab.
 - c. In the Available box, select the group created in step 3, and then click Add.
 - d. Click Save.
5. Click Realms > Top Level Realm > Privileges.
6. Click the group created in step 3, for example `policyEval`.
7. On the Privileges page, select `Read and write access to all realm and policy properties`.
8. Click Save.

Procedure 5.6. To Create a Policy that Uses the Default Policy Condition Script

In this procedure, create a policy that uses the default policy condition script. Policy evaluations can then be performed to test the script functionality.

1. Log in as an OpenAM administrator, for example `amadmin`.
2. Click Realms > Top Level Realm > Authorization > Policy Sets.
3. On the Policy Sets page, select `Default Policy Set`.
4. On the Default Policy Set page, click Add a Policy.
5. Define the policy as follows:
 - a. Enter a name for the policy.
 - b. Define resources to which the policy applies:
 - i. Select `URL` from the Resource Type drop down list.

- ii. Select the resource pattern `*://*:*/*` from the Resources drop down list.
- iii. Click Add.

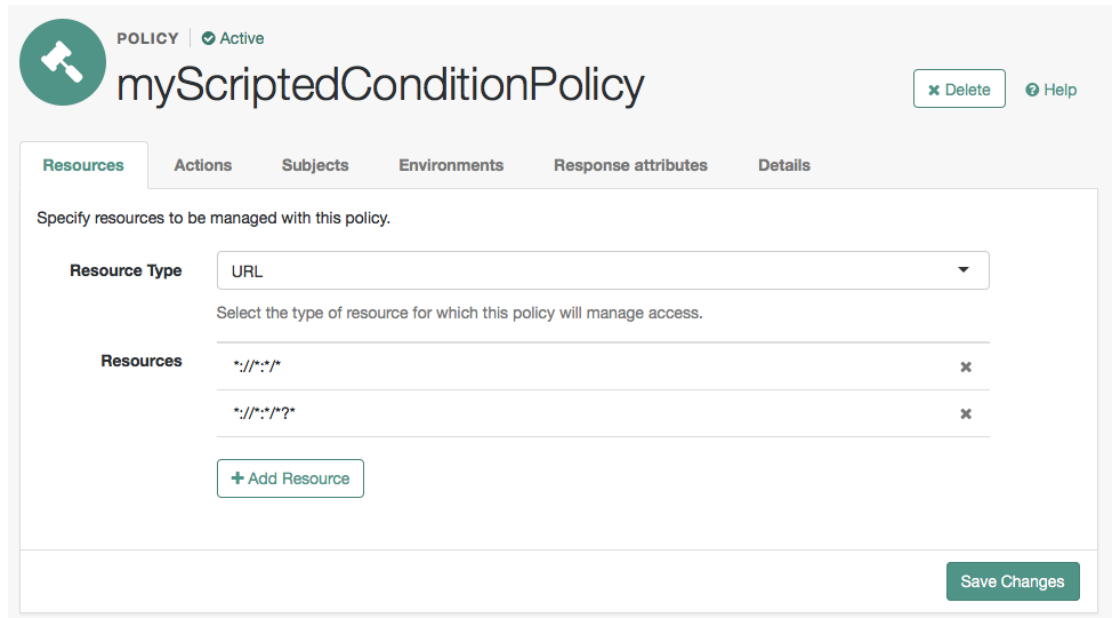
The `*://*:*/*` resource appears in the Resources field.

- iv. Click Add Resource to add a second resource to the policy.
- v. Select the resource pattern `*://*:*/?*` from the Resources drop down list.
- vi. Click Add.

The `*://*:*/?*` resource appears along with the `*://*:*/*` resource in the Resources field.

- vii. Click Create to create the policy.

The Resources tab appears as follows:



- c. Specify actions to which the policy applies:
 - i. Select the Actions tab.
 - ii. Select GET from the Add an Action drop down list.
 - iii. The GET action appears in the list of actions. The default state for the GET action is Allow.

The Actions tab appears as follows:

The screenshot shows the OpenAM console interface for a policy named "myScriptedConditionPolicy". The policy is in an "Active" state. The "Actions" tab is selected, showing a table with columns for "ACTION" and "DEFAULT STATE". A single action "GET" is listed with "Allow" selected as the default state. There is an "Add an Action" button and a "Save Changes" button.

POLICY | Active

myScriptedConditionPolicy

Resources **Actions** Subjects Environments Response attributes Details

Select the actions that the policy applies.

ACTION	DEFAULT STATE	
GET	<input checked="" type="radio"/> Allow <input type="radio"/> Deny	<input type="button" value="x"/>

- iv. Click Save Changes.
- d. Configure subjects to which the policy applies:
 - i. Select the Subjects tab.
 - ii. Click the edit icon—the pencil.
 - iii. Select Authenticated Users from the Type drop down list.
 - iv. Click the OK icon—the check mark.

The Subjects tab appears as follows:

The screenshot shows the configuration page for a policy named 'myScriptedConditionPolicy'. The page is titled 'POLICY | Active' and includes a 'Delete' button and a 'Help' icon. Below the title are tabs for 'Resources', 'Actions', 'Subjects' (which is selected), 'Environments', 'Response attributes', and 'Details'. The main content area is titled 'Specify the subject conditions to which the policy applies.' It features a dropdown menu set to 'All of...', a list of subject conditions with one entry: 'Type: Authenticated Users', and two buttons: '+ Add a Subject Condition' and '+ Add a Logical Operator'. A 'Save Changes' button is located at the bottom right of the configuration area.

- v. Click Save Changes.
- e. Configure environments in which the policy applies:
 - i. Select the Environments tab.
 - ii. Click Add an Environment Condition.
 - iii. Select Script from the Type drop down list.
 - iv. Select Scripted Policy Condition from the Script Name drop down list.
 - v. Click the OK icon—the check mark.

The Environments tab appears as follows:

vi. Click Save Changes.

f. No additional configuration is required in the Response Attributes or Details tabs.

Procedure 5.7. To Enable Message-level Logging for Policy Evaluation

The default policy condition script writes to the debug logs at the `message` level. Message-level debug logging is not enabled for policy evaluation by default.

This section shows how to enable message-level debug logging for policy evaluation, so that logger output from the default policy condition script can be viewed in the `Entitlement` debug log.

1. Log in as an OpenAM administrator, for example `amadmin`.
2. Visit the `Debug.jsp` page, for example: <https://openam.example.com:8443/openam/Debug.jsp>.
3. In the Debug instances drop-down, select `Entitlement`.
4. In the Level drop-down, choose the debug level required. In this example, select `Message`.

5. Click Submit, and on the summary page that appears, click Confirm.

Message-level debug logging is now enabled for policy evaluation.

5.3.2.2. Trying the Default Policy Condition Script

This section demonstrates using a policy that contains the default policy condition script.

To evaluate against a policy, you must first obtain an SSO token for the subject performing the evaluation, in this case the `demo` user. You can then make a call to the `policies?action=evaluate` endpoint, including some environment information, which the policy uses to make an authorization decision.

Procedure 5.8. To Evaluate a Policy

1. Obtain an SSO Token for the `demo` user:

```
curl \
--request POST
\
--header "X-OpenAM-Username: demo"
\
--header "X-OpenAM-Password: changeit" \
https://openam.example.com:8443/openam/json/authenticate
{
  "tokenId": "AQIC5wM2...",
  "successUrl": "/openam/console"
}
```

2. Send an evaluation request to the `policies` endpoint, using the SSO token of the `demo` user in the `iPlanetDirectoryPro` header.

In the JSON data, set the `subject` property to also be the SSO token of the `demo` user. In the `resources` property, include a URL that resides on a server in the same country as the address set for the `demo` user. In the `environment` property, include an IP address that is also based in the same country as the user and the resource. The example below uses the ForgeRock Community web site URL and an IP address from a ForgeRock office, both located in the United States:

```
curl \
--request POST
\
--header "Content-Type: application/json"
\
--header "iPlanetDirectoryPro: AQIC5wM2..."
\
--data '{
  "resources": [
    "http://www.forgerock.org:80/index.html"
  ],
  "application": "iPlanetAMWebAgentService",
  "subject": { "ssoToken": "AQIC5wM2..." },
}
```

```
"environment": {
  "IP": [
    "38.99.39.210"
  ]
}
}' \
https://openam.example.com:8443/openam/json/policies?_action=evaluate
[
  {
    "advices": {},
    "ttl": 9223372036854775807,
    "resource": "http://www.forgerock.org:80/index.html",
    "actions": {
      "POST": true,
      "GET": true
    },
    "attributes": {
      "countryOfOrigin": [
        "United States"
      ]
    }
  }
]
```

If the country in the subject's profile matches the country determined from the source IP in the environment and the country determined from the resource URL, then OpenAM returns a list of actions available. The script will also add an attribute to the response called `countryOfOrigin` with the country as the value.

If the countries do not match, no actions are returned. In the following example, the resource URL is based in France, while the IP and user's address in the profile are based in the United States:


```
curl -X POST
\
-H "Content-Type: application/json"
\
-H "iPlanetDirectoryPro: AQIC5wM2..."
\
-d '{
  "resources": [
    "http://www.forgerock.fr:80/index.html"
  ],
  "application": "iPlanetAMWebAgentService",
  "subject": { "ssoToken": "AQIC5wM2..."},
  "environment": {
    "IP": [
      "38.99.39.210"
    ]
  }
}' \
'https://openam.example.com:8443/openam/json/policies?_action=evaluate'
[
  {
    "advices": {},
    "ttl": 9223372036854775807,
    "resource": "http://www.forgerock.fr:80/index.html",
    "actions": {},
    "attributes": {}
  }
]
```

5.3.3. Default OIDC Claims Script

This section demonstrates how to use the default OIDC claims script to return the profile attributes of a user in response to an OpenID Connect request for the `profile` scope.

The default OIDC claims script maps the following claims to the `profile` scope:

- zoneinfo
- family_name
- locale
- name

To examine the contents of the default OIDC claims script in the OpenAM console browse to Realms > Top Level Realm > Scripts, and then click OIDC Claims Script.

For more information on the functions available for use in OIDC claim scripts, see Section 5.2, "The Scripting API".

5.3.3.1. Preparing OpenAM

OpenAM requires a small amount of configuration before trying the example OIDC claims script. You must first create an OAuth2 provider with OpenID Connect settings, and register an OpenID Connect client, before you can authenticate to the client using a web browser.

The procedures in this section are:

- Procedure 5.9, "To Create an OpenID Connect Provider Service"
- Procedure 5.10, "To Register an OpenID Connect Client"

Procedure 5.9. To Create an OpenID Connect Provider Service

Follow the steps in this procedure to create an OpenID Connect provider service by using the wizard.

1. Log in to OpenAM as an administrator, for example `amadmin`.
2. Click Realms > Top Level Realm > Configure OAuth Provider > Configure OpenID Connect.
3. On the Configure OpenID Connect page, accept the default values and then click Create.
4. Navigate to Realms > Top Level Realm > Services, click OAuth2 Provider, and verify that the value for OIDC Claims Script is the default script, `OIDC Claims Script`.

For a more detailed explanation and example of creating an OpenID Connect provider service, see Section 14.2, "Configuring OpenAM As OpenID Connect Provider" in the *Administration Guide*.

Procedure 5.10. To Register an OpenID Connect Client

Follow the steps in this procedure to create an OpenID Connect client agent profile.

1. Log in to OpenAM as an administrator, for example `amadmin`.
2. Click Realms > Top Level Realm > Agents > OAuth 2.0/OpenID Connect Client.
3. In the Agent table, click New.
4. Enter a name for the client, such as `oidcTest`, provide a password, and then click Create.
5. On the OAuth 2.0/OpenID Connect Client page, click the agent name to configure the agent.
6. On the edit client page:
 - a. In Redirection URIs, enter an example URI such as `http://www.example.com`.
 - b. In Scope(s), enter both `profile` and `openid`.

The `profile` scope will return details about the subject such as given name and timezone. The `openid` scope indicates this is an OpenID Connect client.

- c. In Display name, enter the name of the client as it will be displayed on the consent page, for example `OIDC Claims Script Client`.

7. Save your work.

For a more detailed explanation and examples of registering an OpenID Connect client, see Section 14.5, "Registering OpenID Connect Relying Parties" and Section 5.8, "Configuring OAuth 2.0 and OpenID Connect 1.0 Clients" in the *Administration Guide*.

5.3.3.2. Trying the Default OIDC Claims Script

This section shows how to authenticate to a registered OpenID Connect client and request scopes from OpenAM, which in turn uses the default OIDC Claims script to populate the scope with claims and profile values.

Procedure 5.11. To Authenticate to an OIDC Client and use the Default OIDC Claims Script

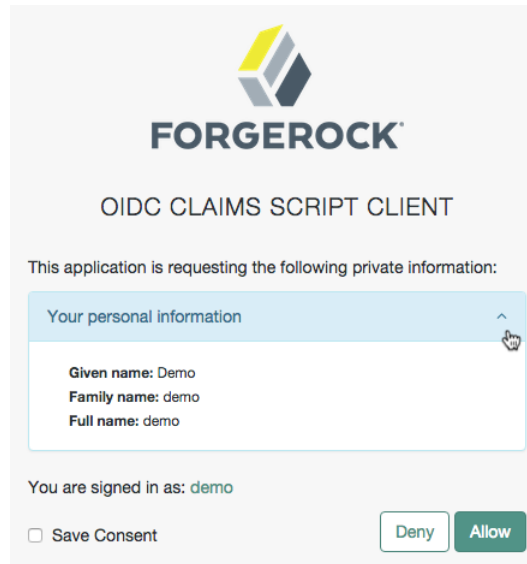
1. Log out of OpenAM.
2. In an Internet browser, navigate to the OpenAM OAuth 2.0 authorization endpoint, `/oauth2/authorize`, and specify the following query parameters, with the values you configured in the agent profile:

Table 5.11. Query parameters for OpenID Connect Authorization to an Agent Profile

Query Parameter	Agent Profile Property Value
<code>client_id</code>	Name of the agent, for example <code>oidcTest</code> .
<code>redirect_uri</code>	Redirection URIs, for example <code>http://www.example.com</code> .
<code>response_type</code>	Response Types, for example <code>code</code> .
<code>scope</code>	Scope(s), for example <code>openid profile</code> .

For example: `http://openam.example.com:8080/openam/oauth2/authorize?client_id=oidcTest&redirect_uri=http://www.example.com&response_type=code&scope=openid profile`

3. Log in to OpenAM as `demo`, with password `changeit`.
4. On the consent page, expand the panel labelled Your personal information to see the claim values the default OIDC script has populated into the requested `profile` scope.



The screenshot shows a consent screen for the 'OIDC CLAIMS SCRIPT CLIENT'. At the top is the ForgeRock logo. Below it, the text reads 'OIDC CLAIMS SCRIPT CLIENT'. A message states: 'This application is requesting the following private information:'. A light blue box titled 'Your personal information' contains the following details: 'Given name: Demo', 'Family name: demo', and 'Full name: demo'. Below this box, it says 'You are signed in as: demo'. At the bottom, there is a checkbox for 'Save Consent' and two buttons: 'Deny' and 'Allow'.

5. Click Allow to be redirected to the Redirection URI specified in the agent profile. The authorization code is appended to the redirection URI as the value of the `code` query parameter.

Chapter 6

Building SAML v2.0 Service Providers With Fedlets

This chapter introduces OpenAM Fedlets, and shows how to use the Fedlet as part of your Java web application.

An OpenAM *Fedlet* is a small web application that acts as a lightweight SAML v2.0 service provider.

Fedlets are easy to integrate into Java web applications. 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.

6.1. Using Fedlets in Java Web Applications

This section introduces OpenAM Fedlets and shows how to use the Fedlet as part of your Java web application.

6.1.1. Creating and Installing a Java Fedlet

The following sections provide procedures for creating, installing, configuring, and testing a Java Fedlet to perform single sign-on and single logout with an identity provider:

- Section 6.1.1.1, "Generating the Fedlet Configuration on the Identity Provider"
- Section 6.1.1.2, "Installing and Configuring the Fedlet on the Service Provider"
- Section 6.1.1.3, "Testing Fedlet Single Sign-on and Single Logout"

You can also use the Fedlet to query attributes of users on identity providers configured with the Attribute Authority (**AttrAuth**) and the XACML Policy Decision Point (**XACML PDP**) types. See the following sections for additional configuration requirements:

- Section 6.1.1.4, "Querying an Attribute Authority"
- Section 6.1.1.5, "Querying an XACML Policy Decision Point"

6.1.1.1. Generating the Fedlet Configuration on the Identity Provider

Perform the following steps on the identity provider to create a `Fedlet.zip` file containing the configuration files for the Java fedlet:

Procedure 6.1. To Create Configuration Files for a Fedlet

1. If you have not already done so, create a hosted identity provider, using the test certificate for the signing key.

For information about how to create a hosted identity provider, see Section 12.4.1, "Creating a Hosted Identity Provider" in the *Administration Guide*.

2. Under Realms > *Realm Name* > Dashboard, click Create Fedlet Configuration, and then click Create Fedlet Configuration a second time.

Note that the circle of trust includes your hosted identity provider.

3. Name the Fedlet, and 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.

4. Click Create to generate the `Fedlet.zip` file.

A message appears indicating Fedlet creation was successful. Note the location of the generated output file in the message. For example, `/path/to/openam/config/myfedlets/httpopenamexamplecom8080fedlet/Fedlet.zip`.

5. Click OK to close the message informing you that the Fedlet was created.
6. Transfer the `Fedlet.zip` file to the service provider administrator.

6.1.1.2. Installing and Configuring the Fedlet on the Service Provider

Having obtained the `Fedlet.zip` file from the identity provider administrator, the service provider administrator unzips and installs the file, and then installs the `fedlet.war` file from the OpenAM distribution:

Procedure 6.2. To Install and Configure the Fedlet as a Demo Application

1. Create the `fedlet` directory in the home directory of the user that runs the OpenAM web container:

```
$ cd $HOME
$ mkdir fedlet
```

2. Copy the `Fedlet.zip` file obtained from the identity provider administrator to the `$HOME/fedlet` directory.

- Unzip the `Fedlet.zip` file.
- Move all the files under `$HOME/fedlet/conf` to `$HOME/fedlet`.
- Obtain the `Fedlet-13.5.2.zip` file from the full OpenAM distribution file, `OpenAM-13.5.2.zip`.
- Unzip the `Fedlet-13.5.2.zip` file:

```
$ cd /path/to/openam-distribution/openam
$ unzip Fedlet-13.5.2.zip
```

- Deploy the Fedlet in your web container:

```
$ cp /path/to/openam-distribution/openam/fedlet.war /path/to/tomcat/webapps
```

6.1.1.3. Testing Fedlet Single Sign-on and Single Logout

To test single sign-on and single logout from the Fedlet, browse to the Fedlet URL. For example, <http://openam.example.com:8080/fedlet>.

Try one or more examples from the Fedlet home page:

Validate Fedlet Setup

Click following links to start Fedlet(SP) and/or IDP initiated Single Sign-On. Upon successful completion, you will be presented with a page to display the Single Sign-On Response, Assertion and AttributeStatement received from IDP side.

Fedlet (SP) Configuration Directory:	/home/mark/fedlet
Fedlet (SP) Entity ID:	http://www.example.com:8080/fedlet
IDP Entity ID:	http://openam.example.com:8080/openam

[Run Fedlet \(SP\) initiated Single Sign-On using HTTP POST binding](#)
[Run Fedlet \(SP\) initiated Single Sign-On using HTTP Artifact binding](#)
[Run Identity Provider initiated Single Sign-On using HTTP POST binding](#)
[Run Identity Provider initiated Single Sign-On using HTTP Artifact binding](#)

You can log in to the identity provider with username `demo` and password `changeit`.

6.1.1.4. Querying an Attribute Authority

You can use the Fedlet to query attributes on an identity provider. 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.

Perform the following steps:

Procedure 6.3. To Use the Fedlet to Query an Attribute Authority

- Add the Attribute Authority type to the hosted identity provider.

- a. On OpenAM where the identity provider is hosted, log in to the OpenAM console as an administrator, such 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. Generate the Fedlet configuration files as described in Procedure 6.1, "To Create Configuration Files for a Fedlet", making sure you map the attributes.

- `cn=cn`
- `sn=sn`
- `uid=uid`

This step creates a Fedlet configuration 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 as described in Procedure 6.2, "To Install and Configure 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 keystore 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 keystore and private key passwords are both the same.

Use the encoded value to create the password files as in the following example.

```
$ echo AQIC5BHNSjLwT303GqndmHbyYvzP9Tz70AnK > ~/fedlet/.storepass
$ echo AQIC5BHNSjLwT303GqndmHbyYvzP9Tz70AnK > ~/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/xmlsig#">
    <ds:X509Data>
    <ds:X509Certificate>
    MIICQDCCAakCBEeNB0swDQYJKoZIhvcNAQEEBQAwZzELMAkGA1UEBhMCVVMxEzARBgNVBAgTCKNh
    bGmb3JuaWExFDASBgNVBAcTC1NhbhRrIENsYXJhMQwwCgYDVQQKEwNTdW4xEDA0BgNVBAsTB09w
    ZW5TU08xDALBgNVBAMTBHRlc3QwHhcNMDgwMTE1MTkxOTM5WhcNMTgwMTEyMTkxOTM5WjBnMQsw
    CQYDVQQGEwJVUzETMBEGA1UECBMKQ2FsaWZvcml5pYTEUMBIGA1UEBxMLU2FudGEGQ2xhcmExDDAK
    BgNVBAoTA1N1b1JlEQMA4GA1UECXMHT3BlbnNTTzENMAAsGA1UEAxMEdGVzdDcBnzANBkgqhkiG9w0B
    AQEFAA0BjQAwgYkCgYEArsQC/U75GB2AtKhbGS5piiLkmJzqEsp64rDxbMJ+xDrye0EN/q1U50f+
    RkDsaN/igkAvV1cuXEGTL6RLafFPcUX7QxDhZBhsYF9pbwtMzi4A4su9hnxIhURRebGEmxKW9qJNY
    Js0Vo5+IgjxuEwnjnnVgHTs1+mq5QYTA7E6ZyL8CAwEAATANBgkqhkiG9w0BAQQFAA0BgQB3Pw/U
    QzPKTPTYi9upbFXlrAKMwtFf20W4yvGWVlCwcNSZJmTJ8ARvVY0MEVnbsT40Fcfu2/PeYoAdiDA
    cGy/F2Zuj8XJJpuQRSE6PtQqBuDEHj m0QJ0rV/r8m01ZCtHRhpZ5zYRj hRC9eCbJx9vRFax0JDC
    /FfwWigmrW0Y0Q==
    </ds:X509Certificate>
    </ds:X509Data>
    </ds:KeyInfo>
    </KeyDescriptor>
    <KeyDescriptor use="encryption">
    <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmlsig#">
    <ds:X509Data>
    <ds:X509Certificate>
    MIICQDCCAakCBEeNB0swDQYJKoZIhvcNAQEEBQAwZzELMAkGA1UEBhMCVVMxEzARBgNVBAgTCKNh
    bGmb3JuaWExFDASBgNVBAcTC1NhbhRrIENsYXJhMQwwCgYDVQQKEwNTdW4xEDA0BgNVBAsTB09w
    ZW5TU08xDALBgNVBAMTBHRlc3QwHhcNMDgwMTE1MTkxOTM5WhcNMTgwMTEyMTkxOTM5WjBnMQsw
    CQYDVQQGEwJVUzETMBEGA1UECBMKQ2FsaWZvcml5pYTEUMBIGA1UEBxMLU2FudGEGQ2xhcmExDDAK
    BgNVBAoTA1N1b1JlEQMA4GA1UECXMHT3BlbnNTTzENMAAsGA1UEAxMEdGVzdDcBnzANBkgqhkiG9w0B
    AQEFAA0BjQAwgYkCgYEArsQC/U75GB2AtKhbGS5piiLkmJzqEsp64rDxbMJ+xDrye0EN/q1U50f+
    RkDsaN/igkAvV1cuXEGTL6RLafFPcUX7QxDhZBhsYF9pbwtMzi4A4su9hnxIhURRebGEmxKW9qJNY
    Js0Vo5+IgjxuEwnjnnVgHTs1+mq5QYTA7E6ZyL8CAwEAATANBgkqhkiG9w0BAQQFAA0BgQB3Pw/U
    QzPKTPTYi9upbFXlrAKMwtFf20W4yvGWVlCwcNSZJmTJ8ARvVY0MEVnbsT40Fcfu2/PeYoAdiDA
    cGy/F2Zuj8XJJpuQRSE6PtQqBuDEHj m0QJ0rV/r8m01ZCtHRhpZ5zYRj hRC9eCbJx9vRFax0JDC
    /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 the 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.

Validate Fedlet Setup

Click following links to start Fedlet(SP) and/or IDP initiated Single Sign-On. Upon successful completion, you will be presented with a page to display the Single Sign-On Response, Assertion and AttributeStatement received from IDP side.

Fedlet (SP) Configuration Directory:	/Users/mark/fedlet
Fedlet (SP) Entity ID:	http://openam.example.com:8080/fedlet
IDP Entity ID:	http://openam.example.com:8080/openam

[Run Fedlet \(SP\) initiated Single Sign-On using HTTP POST binding](#)
[Run Fedlet \(SP\) initiated Single Sign-On using HTTP Artifact binding](#)
[Run Identity Provider initiated Single Sign-On using HTTP POST binding](#)
[Run Identity Provider initiated Single Sign-On using HTTP Artifact binding](#)

- b. Try SSO with username **demo**, password **changeit**.

Single Sign-On successful with IDP <http://openam.example.com:8080/openam>.

Name ID format: urn:oasis:names:tc:SAML:2.0:nameid-format:transient
Name ID value: 4RN2HcPc/bLjHcH+GQIF23d9l8qI
Sessionindex: s24385c71963b22e2bdd4855cd6189b4beceb77201
Attributes: uid=demo
 sn=demo
 cn=demo

[Click to view SAML2 Response XML](#)

[Click to view Assertion XML](#)

[Click to view Subject XML](#)

Test Attribute Query:

[Fedlet Attribute Query](#)

Test XACML Policy Decision Query:

[Fedlet XACML Query](#)

Test Single Logout:

[Run Identity Provider initiated Single Logout using SOAP binding](#)
[Run Identity Provider initiated Single Logout using HTTP Redirect binding](#)
[Run Identity Provider initiated Single Logout using HTTP POST binding](#)
[Run Fedlet initiated Single Logout using SOAP binding](#)
[Run Fedlet initiated Single Logout using HTTP Redirect binding](#)
[Run Fedlet initiated Single Logout using HTTP POST binding](#)

- c. Click Fedlet Attribute Query, set the attributes in the Attribute Query page to match the mapped attributes, and then click Submit.

Attribute Query

Subject
SAML2 Token (Transient)

Attribute 1

Attribute 2

Attribute 3

Profile Name

Default

X.509

X.509 Subject DN

- 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

6.1.1.5. Querying an XACML Policy Decision Point

You can use the Fedlet to query an XACML policy decision point on an identity provider. The identity provider must have a policy configured, must be configured with the Policy Decision Point (XACML PDP) type, and must have a SAML v2.0 SOAP Binding PDP handler configured.

Perform the following steps:

Procedure 6.4. To Use the Fedlet to Query an XACML Policy Decision Point

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 an administrator, such as `amadmin`.
- b. Access the policy editor under Realms > *Realm Name* > Authorization.
- 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 SAML v2.0 SOAP Binding.
 - a. Navigate to Configure > Global Services, click SAMLv2 SOAP Binding, and then 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's configuration files as described in Procedure 6.1, "To Create Configuration Files for a Fedlet".

This step creates Fedlet configuration files 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 as described in Procedure 6.2, "To Install and Configure the Fedlet as a Demo Application".
6. Try the XACML Query test.
 - a. Access the Fedlet.

Validate Fedlet Setup

Click following links to start Fedlet(SP) and/or IDP initiated Single Sign-On. Upon successful completion, you will be presented with a page to display the Single Sign-On Response, Assertion and AttributeStatement received from IDP side.

Fedlet (SP) Configuration Directory:	/Users/mark/fedlet
Fedlet (SP) Entity ID:	http://openam.example.com:8080/fedlet
IDP Entity ID:	http://openam.example.com:8080/openam

[Run Fedlet \(SP\) initiated Single Sign-On using HTTP POST binding](#)
[Run Fedlet \(SP\) initiated Single Sign-On using HTTP Artifact binding](#)
[Run Identity Provider initiated Single Sign-On using HTTP POST binding](#)
[Run Identity Provider initiated Single Sign-On using HTTP Artifact binding](#)

- b. Try SSO with username **demo**, password **changeit**.

Single Sign-On successful with IDP <http://openam.example.com:8080/openam>.

Name ID format: urn:oasis:names:tc:SAML:2.0:nameid-format:transient
Name ID value: 4RN2HcPc/bLjHcH+GQIF23d9l8qI
SessionIndex: s24385c71963b22e2bdd4855cd6189b4beceb77201
Attributes: uid=demo
 sn=demo
 cn=demo

[Click to view SAML2 Response XML](#)

[Click to view Assertion XML](#)

[Click to view Subject XML](#)

Test Attribute Query:

[Fedlet Attribute Query](#)

Test XACML Policy Decision Query:

[Fedlet XACML Query](#)

Test Single Logout:

[Run Identity Provider initiated Single Logout using SOAP binding](#)
[Run Identity Provider initiated Single Logout using HTTP Redirect binding](#)
[Run Identity Provider initiated Single Logout using HTTP POST binding](#)
[Run Fedlet initiated Single Logout using SOAP binding](#)
[Run Fedlet initiated Single Logout using HTTP Redirect binding](#)
[Run Fedlet initiated Single Logout using HTTP POST binding](#)

- c. Click XACML Attribute Query, set the Resource URL in the XACML Query page to <http://www.example.com/>, and then click Submit.

XACML Query

Resource URL

Action

 GET
 POST

- d. Check that you see the permit decision in the response.

Fedlet XACML Query Response

Resource	Policy Decision
http://www.example.com/	Permit

6.1.2. Enabling Signing and Encryption in 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.

Enabling 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 keystore and private key are both `changeit`.

- Configure the Fedlet to perform signing and encryption by ensuring the Fedlet has access to the keystore, 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.


```

</KeyDescriptor>
<KeyDescriptor use="encryption">
  <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
    <ds:X509Data>
      <ds:X509Certificate>
MIICQDCCAakCBEeNB0swDQYJKoZIhvcNAQEEBQAwwZELMAkGA1UEBhMCVVMxEzARBgNVBAgTCkNh
bGlb3JuaWExFDASBgNVBAcTC1NhbnRlIENsYXJhMQwwCgYDVQQKEWNTdW4xEDA0BgNVBAsTB09w
ZW5TU08xDALBgNVBAMTBHRlc3QwHhcNMDgwMTE1MTkxOTM5WWhcNMTgMTkxOTM5WjBnMQsw
CQYDVQQGEwJVUzETMBEGA1UECBMKQ2FsaWZvcml5PTEUMBIGA1UEBxMLU2FudGEgQ2xhcmExDDAK
BgNVBAoTA1N1bWVjEQA4GA1UECXMHT3BlbnNTTzENMAAsGA1UEAxMEdGVzZdDCBnzANBgkqhkiG9w0B
AQEFAA0BjQAwgYkCgYEArsQc/U75GB2AtKhbGS5piiLkmJzqEsp64rDxbMJ+xDrye0EN/q1U50f+
RkDsaN/igkAvV1cuXegTL6RlafFPcUX7QxDhZBhsYF9pbwtMzi4A4su9hnXihURRebGEmxKw9qJNY
Js0Vo5+IgjxuEwnjnnVgHTs1+mQ5QYTA7E6ZyL8CAwEAATANBgkqhkiG9w0BAQQFAA0BgQB3Pw/U
QzPKTPTYi9upbFxlAKMwtFf20W4yvGWwvlcwcNSZJmTJ8ARvVYOMEVNBst40FcFu2/PeYoAdiDA
cGy/F2Zuj8XJJpuQRSE6PtQqBuDEHjjm0QJ0rV/r8m01ZCtHRhpZ5zYRjhRC9eCbjx9VrFax0JDC
/FfwWigmrW0Y0Q==
      </ds:X509Certificate>
    </ds:X509Data>
    </ds:KeyInfo>
    <EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmenc#aes128-cbc">
      <xenc:KeySize xmlns:xenc="http://www.w3.org/2001/04/xmenc#">
        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>
<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.

6.1.3. Customizing a Java Fedlet

You can customize the Java Fedlet to perform many of the SAML v2.0 service provider operations. The Java Fedlet has the SAML v2.0 capabilities identified in Table 12.2, "Fedlet Support for SAML v2.0 Features" in the *Administration Guide*.

Procedure 6.6. 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 SAML v2.0 specifications,
    // such as XML signature validation, Audience and Recipient
    // validation etc.
    map = SPACSUtills.processResponseForFedlet(request, response,
      new PrintWriter(out, true));
    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`.

6.1.3.1. 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 `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 `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`

6.1.3.2. 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 `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 `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.

6.1.3.3. Performing Attribute Queries

As seen in Procedure 6.3, "To Use the Fedlet to Query an Attribute Authority", 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.

6.1.3.4. Performing XACML Queries

As seen in Procedure 6.4, "To Use the Fedlet to Query an XACML Policy Decision Point", 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.

6.2. Configuring Java Fedlets By Hand

An OpenAM Fedlet is a small web application that makes it easy to add SAML v2.0 Service Provider (SP) capabilities to your Java web application. The OpenAM console offers a wizard for configuring a

Java Fedlet as a SAML v2.0 Service Provider with OpenAM as the Identity Provider (IDP). If that fits your purposes, then read the chapter Section 6.1, "Using Fedlets in Java Web Applications" instead.

The full distribution file, [OpenAM-13.5.2.zip](#), also includes a Java Fedlet, [Fedlet-13.5.2.zip](#), that you can configure by hand. This chapter covers how to configure a Java Fedlet using that distribution, by manually editing the Circle of Trust, Java properties, and IDP and SP XML configuration templates.

Seen from a high level, what you must do is this:

- Determine the roles that the IDP(s) and Fedlet play in SAML v2.0 Circles of Trust.
- Unpack the unconfigured Fedlet from the full OpenAM distribution to access the Fedlet war and template configuration files.
- Begin preparing the Fedlet configuration, including setting up a configuration directory and keystore if needed.
- Obtain SAML v2.0 metadata configuration files from the IDP(s), and add them to the Fedlet configuration.

The IDP must provide at least the standard SAML v2.0 metadata.

- Finish preparing the Fedlet configuration by editing the remaining Fedlet template configuration files.
- Share the Fedlet SAML v2.0 configuration files at least for the standard SAML v2.0 metadata with the IDP(s).

An IDP relies on the standard SAML v2.0 metadata to communicate with the Fedlet.

- Deploy and test the Fedlet with each IDP.

6.2.1. Java Fedlet Layout

Unpack the Java Fedlet distribution into a working directory.

```
$ mkdir fedlet && cd fedlet
$ unzip ../Fedlet-13.5.2.zip
```

When you unpack the [Fedlet-13.5.2.zip](#) file, you find the following files.

[Fedlet-13.5.2.war](#)

This file contains a Java Fedlet web application that serves as an example, and that you can embed in your applications.

[README](#)

This file succinctly describes how to configure some Fedlet features.

conf/

This folder contains the Fedlet configuration templates that you edit as appropriate for your deployment.

When editing the templates, place copies of the files in the Fedlet home directory on the system where you deploy the Fedlet. By default the Fedlet home directory is `user.home/uri`, where `user.home` is the value of the Java system property `user.home` for the user running the web container where you deploy the Fedlet, and `uri` is the path of the URI where you deploy the Fedlet, such as `/fedlet`.

For example, if `user.home` is `/home/user`, that user could have a `/home/user/fedlet` folder for Fedlet configuration files.

```
$ mkdir ~/fedlet
```

To change the location, set the system property `com.sun.identity.fedlet.home` when starting the container where the Fedlet runs.

```
$ java -Dcom.sun.identity.fedlet.home=/path/to/fedlet/conf ...
```

conf/FederationConfig.properties

This file defines settings for the Fedlet as a web application. It does not address the SAML v2.0 configuration.

For more about this file, see Section 6.2.2, "Configuring Java Fedlet Properties".

conf/fedlet.cot-template

This template defines settings for a SAML v2.0 Circle of Trust to which the Fedlet belongs.

For more about this file, see Section 6.2.3, "Configuring Circles of Trust".

conf/idp.xml (not provided)

The `idp.xml` file is standard SAML v2.0 metadata that describes the IDP configuration.

Templates for other SAML v2.0 configuration files are provided, but no `idp.xml` template file is provided.

Instead you must obtain the SAML v2.0 metadata from the IDP, and add it as `idp.xml` here, alongside the other SAML v2.0 configuration files. How you obtain this file from the IDP depends on the IDP implementation.

conf/idp-extended.xml-template

This template holds extended SAML v2.0 IDP settings that OpenAM uses.

For more about this file, see Section 6.2.4, "Configuring the Identity Providers".

`conf/sp.xml-template`

This template describes standard SAML v2.0 SP settings.

For more about this file, see Section 6.2.5, "Configuring the Service Providers".

`conf/sp-extended.xml-template`

This template describes extended SAML v2.0 SP settings that the Fedlet uses.

For more about this file, see Section 6.2.5, "Configuring the Service Providers".

6.2.2. Configuring Java Fedlet Properties

The Java Fedlet to configure by hand includes a `FederationConfig.properties` file that defines settings for the Fedlet as a web application. The configuration for a single Java Fedlet includes only one `FederationConfig.properties` file, regardless of how many IDP and SP configurations are involved. This file does not address the SAML v2.0 configuration.

When configured this file contains sensitive properties such as the value of `am.encrypted.pwd`. Make sure it is readable only by the user running the Fedlet application.

This section categorizes the settings as follows:

- Deployment URL Settings
- Log and Statistics Settings
- Public and Private Key Settings
- Alternative Implementation Settings

Deployment URL Settings

The following settings define the Fedlet deployment URL.

`com.ipplanet.am.server.protocol`

Set this to the protocol portion of the URL, such as HTTP or HTTPS.

`com.ipplanet.am.server.host`

Set this to the host portion of the URL, such as `sp.example.com`.

`com.ipplanet.am.server.port`

Set this to the port portion of the URL, such as 80, 443, 8080, or 8443.

`com.ipplanet.am.services.deploymentDescriptor`

Set this to path portion of the URL, starting with a /, such as `/fedlet`.

Log and Statistics Settings

The following settings define the Fedlet configuration for logging and monitoring statistics.

`com.ipplanet.am.logstatus`

This sets whether the Fedlet actively writes debug log files.

Default: `ACTIVE`

`com.ipplanet.services.debug.level`

This sets the debug log level.

The following settings are available, in order of increasing verbosity:

`off`
`error`
`warning`
`message`

Default: `message`

`com.ipplanet.services.debug.directory`

This sets the location of the debug log folder.

Trailing spaces in the file names are significant. Even on Windows systems, use slashes to separate directories.

Examples: `/home/user/fedlet/debug`, `C:/fedlet/debug`

`com.ipplanet.am.stats.interval`

This sets the interval at which statistics are written, in seconds.

The shortest interval supported is 5 seconds. Settings less than 5 (seconds) are taken as 5 seconds.

Default: 60

`com.ipplanet.services.stats.state`

This sets how the Fedlet writes monitoring statistics.

The following settings are available:

`off`
`console` (write to the container logs)
`file` (write to Fedlet stats logs)

Default: `file`

`com.iplanet.services.stats.directory`

This sets the location of the stats file folder.

Trailing spaces in the file names are significant. Even on Windows systems, use slashes to separate directories.

Examples: `/home/user/fedlet/stats`, `C:/fedlet/stats`

Public and Private Key Settings

The following settings define settings for access to certificates and private keys used in signing and encryption.

Other sections in this guide explain how to configure a Fedlet for signing and encryption including how to work with the keystores that these settings reference, and how to specify public key certificates in standard SAML v2.0 metadata. When working with a Java Fedlet, see the section on Section 6.1.2, "Enabling Signing and Encryption in a Fedlet".

Tip

Although this section focuses on Java Fedlets, if you are working with a .NET Fedlet see [How do I use Fedlets in .NET applications in OpenAM \(All versions\)?](#) in the *ForgeRock Knowledge Base*.

`com.sun.identity.saml.xmlsig.keystore`

This sets the path to the keystore file that holds public key certificates of IDPs and key pairs for the Fedlet.

For hints on generating a keystore file with a key pair, see Procedure 23.5, "To Change OpenAM Default test Signing Key " in the *Administration Guide*.

Example: `@FEDLET_HOME@/keystore.jks`

`com.sun.identity.saml.xmlsig.storepass`

This sets the path to the file that contains the keystore password encoded by using the symmetric key set as the value of `am.encrypted.pwd`.

When creating the file, encode the clear text password by using your own test copy (not a production version) of OpenAM.

- Log in to the OpenAM Console as administrator `amadmin`.

- Under Deployment > Servers > *Server Name* > Security > Encryption, set the Password Encryption Key to your symmetric key, and save your work.

Do not do this in a production system where the existing symmetric key is already in use!

- Switch to the `encode.jsp` page, such as `http://openam.example.com:8080/openam/encode.jsp`, enter the clear text password to encode with your symmetric key, and click Encode.
- Copy the encoded password to your file.

Example: `@FEDLET_HOME@/.storepass`

`com.sun.identity.saml.xmlsig.keypass`

This sets the path to the file that contains the private key password encoded by using the symmetric key set as the value of `am.encryption.pwd`.

To encode the clear text password, follow the same steps for the password used when setting `com.sun.identity.saml.xmlsig.storepass`.

Example: `@FEDLET_HOME@/.keypass`

`com.sun.identity.saml.xmlsig.certalias`

This sets the alias of the Fedlet's public key certificate.

Example: `fedlet-cert`

`com.sun.identity.saml.xmlsig.storetype`

The sets the type of keystore.

Default: `JKS`

`am.encryption.pwd`

This sets the symmetric key that used to encrypt and decrypt passwords.

Example: `uu4dHvBkJJpIjPQWM74pxH3brZJ5gJje`

Alternative Implementation Settings

The Java Fedlet properties file includes settings that let you plug in alternative implementations of Fedlet capabilities. You can safely use the default settings, as specified in the following list. The list uses the same order for the keys you find in the file.

`com.sun.identity.plugin.configuration.class`

Default: `com.sun.identity.plugin.configuration.impl.FedletConfigurationImpl`

com.sun.identity.plugin.datastore.class.defaultDefault: `com.sun.identity.plugin.datastore.impl.FedletDataStoreProvider`**com.sun.identity.plugin.log.class**Default: `com.sun.identity.plugin.log.impl.FedletLogger`**com.sun.identity.plugin.session.class**Default: `com.sun.identity.plugin.session.impl.FedletSessionProvider`**com.sun.identity.plugin.monitoring.agent.class**Default: `com.sun.identity.plugin.monitoring.impl.FedletAgentProvider`**com.sun.identity.plugin.monitoring.saml1.class**Default: `com.sun.identity.plugin.monitoring.impl.FedletMonSAML1SvcProvider`**com.sun.identity.plugin.monitoring.saml2.class**Default: `com.sun.identity.plugin.monitoring.impl.FedletMonSAML2SvcProvider`**com.sun.identity.plugin.monitoring.idff.class**Default: `com.sun.identity.plugin.monitoring.impl.FedletMonIDFFSvcProvider`**com.sun.identity.saml.xmlsig.keyprovider.class**Default: `com.sun.identity.saml.xmlsig.JKSKeyProvider`**com.sun.identity.saml.xmlsig.signatureprovider.class**Default: `com.sun.identity.saml.xmlsig.AMSignatureProvider`**com.sun.identity.common.serverMode**Default: `false`**com.sun.identity.webcontainer**Default: `WEB_CONTAINER`**com.sun.identity.saml.xmlsig.passwordDecoder**Default: `com.sun.identity.fedlet.FedletEncodeDecode`**com.iplanet.services.comm.server.pllrequest.maxContentLength**Default: `16384`

com.iplanet.security.SecureRandomFactoryImplDefault: `com.iplanet.am.util.SecureRandomFactoryImpl`**com.iplanet.security.SSLSocketFactoryImpl**Default: `com.sun.identity.shared.ldap.factory.JSSESocketFactory`**com.iplanet.security.encryptor**Default: `com.iplanet.services.util.JCEEncryption`**com.sun.identity.jss.donotInstallAtHighestPriority**Default: `true`**com.iplanet.services.configpath**Default: `@BASE_DIR@`

6.2.3. Configuring Circles of Trust

As described in Section 6.2.1, "Java Fedlet Layout", this template defines settings for a SAML v2.0 Circle of Trust. The Fedlet belongs to at least one Circle of Trust.

This section includes the following procedures:

- Procedure 6.7, "To Configure a Circle of Trust With a Single IDP"
- Procedure 6.8, "To Configure Multiple Circles of Trust"
- Procedure 6.9, "To Configure a Circle of Trust With Multiple IDPs"

Procedure 6.7. To Configure a Circle of Trust With a Single IDP

When the Fedlet is involved in only a single Circle of Trust with one IDP and the Fedlet as an SP, the only settings to change are `cot-name` and `sun-fm-trusted-providers`.

1. Save a copy of the template as `fedlet.cot` in the configuration folder, as in the following example.

```
$ cp ~/Downloads/fedlet/conf/fedlet.cot-template ~/fedlet/fedlet.cot
```

2. Set `cot-name` to the name of the Circle of Trust.
3. Set `sun-fm-trusted-providers` to a comma-separated list of the entity names for the IDP and SP.

For example, if the IDP is OpenAM with entity ID `https://openam.example.com:8443/openam` and the SP is the Fedlet with entity ID `https://sp.example.net:8443/fedlet`, then set the property as follows.

```
sun-fm-trusted-providers=https://openam.example.com:8443/openam,\  
https://sp.example.net:8443/fedlet
```

Procedure 6.8. To Configure Multiple Circles of Trust

This procedure concerns deployments where the Fedlet participates as SP in multiple Circles of Trust, each involving their own IDP.

1. For each Circle of Trust, save a copy of the template in the configuration folder.

The following example involves two Circles of Trust.

```
$ cp ~/Downloads/fedlet/conf/fedlet.cot-template ~/fedlet/fedlet.cot  
$ cp ~/Downloads/fedlet/conf/fedlet.cot-template ~/fedlet/fedlet2.cot
```

2. Set up IDP XML files for each IDP as described in Section 6.2.4, "Configuring the Identity Providers".
3. For each Circle of Trust, set up the cot file as described in Procedure 6.7, "To Configure a Circle of Trust With a Single IDP".
4. In the extended SP XML file described in Section 6.2.4, "Configuring the Identity Providers", set the Attribute element with name `cotlist` to include values for all Circles of Trust. The values are taken from the `cot-name` settings in the cot files.

The following example works with two Circles of Trust, `cot` and `cot2`.

```
<Attribute name="cotlist">  
  <Value>cot</Value>  
  <Value>cot2</Value>  
</Attribute>
```

The same Attribute element is also available in extended IDP XML files for cases where an IDP belongs to multiple Circles of Trust.

Procedure 6.9. To Configure a Circle of Trust With Multiple IDPs

When the Circle of Trust involves multiple IDPs, use the Fedlet in combination with the OpenAM IDP Discovery service.

Note

For this to work, the IDPs must be configured to use IDP discovery, and users must have preferred IDPs.

1. Set up the OpenAM IDP Discovery service.

For details see Section 12.4.6, "Deploying the Identity Provider Discovery Service" in the *Administration Guide*.

2. Configure the Circle of Trust as described in Procedure 6.7, "To Configure a Circle of Trust With a Single IDP", but specifying multiple IDPs, including the IDP that provides the IDP Discovery service.
3. Set the `sun-fm-saml2-readerservice-url` and the `sun-fm-saml2-writerservice-url` properties as defined for the IDP Discovery service.

6.2.4. Configuring the Identity Providers

As described in Section 6.2.1, "Java Fedlet Layout", the IDP provides its standard SAML v2.0 metadata as XML, which you save in the configuration folder as `idp.xml`. If the IDP uses OpenAM, the IDP can also provide extended SAML v2.0 metadata as XML, which you save in the configuration folder as `idp-extended.xml`, rather than using the template for extended information.

If you have multiple identity providers, then number the configuration files, as in `idp.xml`, `idp2.xml`, `idp3.xml`, and also `idp-extended.xml`, `idp2-extended.xml`, `idp3-extended.xml` and so on.

6.2.4.1. Identity Provider Standard XML

This section covers the configuration in `idp.xml`. The `idp.xml` file contains standard SAML v2.0 metadata for an IDP in a Circle of Trust that includes the Fedlet as SP. The IDP provides you the content of this file.

If the IDP uses OpenAM then the administrator can export the metadata by using either the **soadm create-metadata-templ** command or the `/saml2/jsp/exportmetadata.jsp` endpoint under the OpenAM deployment URL.

If the IDP uses an implementation different from OpenAM, see the documentation for details on obtaining the standard metadata. The standard, product-independent metadata are covered in *Metadata for the OASIS Security Assertion Markup Language (SAML) V2.0*. The standard XML namespace describing the XML document has identifier `urn:oasis:names:tc:SAML:2.0:metadata`. An XML schema description for this namespace is found online at <http://docs.oasis-open.org/security/saml/v2.0/saml-schema-metadata-2.0.xsd>.

6.2.4.2. Identity Provider Extended XML

This section covers the configuration in `idp-extended.xml`. Most extended metadata are specific to the OpenAM implementation of SAML v2.0. If the IDP runs OpenAM, have the IDP provide the extended metadata exported by using the **soadm create-metadata-templ** command. This section covers only the basic settings relative to all IDPs.

The extended metadata file describes an EntityConfig element, defined by the namespace with the identifier `urn:sun:fm:SAML:2.0:entityconfig`. The XML schema definition is described in `entity-config-`

`schema.xsd`, available online as part of the OpenAM source code, though not included in the OpenAM war file.

The unconfigured Fedlet includes a template file, `conf/idp-extended.xml-template`. This extended metadata template for the IDP requires that you edit at least the `IDP_ENTITY_ID` and `fedletcot` values to reflect the IDP entity ID used in the standard metadata and the Circle of Trust name defined in `fedlet.cot`, respectively. The `hosted` attribute on the `EntityConfig` element must remain set to `hosted="0"`, meaning that the IDP is remote. The IDP is likely to play at least the role of SSO Identity Provider, though the namespace defines elements for the Attribute Authority and Policy Decision Point roles shown in the template, as well as the others defined in the standard governing SAML v2.0 metadata.

The extended metadata file is essentially a series of XML maps of key-value pairs specifying IDP configuration for each role. All role-level elements can take a `metaAlias` attribute that the Fedlet uses when communicating with the IDP. Each child element of a role element defines an `Attribute` whose `name` is the key. Each `Attribute` element can contain multiple `Value` elements. The `Value` elements' contents comprise the values for the key. All values are strings, sometimes with a format that is meaningful to OpenAM. The basic example in the IDP template shows the minimal configuration for the SSO IDP role.

In the following example, the `description` is empty and the name of the Circle of Trust is `fedletcot`.

```
<IDPSSOConfig>
  <Attribute name="description">
    <Value/>
  </Attribute>
  <Attribute name="cotlist">
    <Value>fedletcot</Value>
  </Attribute>
</IDPSSOConfig>
<AttributeAuthorityConfig>
  <Attribute name="cotlist">
    <Value>fedletcot</Value>
  </Attribute>
</AttributeAuthorityConfig>
<XACMLPDPConfig>
  <Attribute name="wantXACMLAuthzDecisionQuerySigned">
    <Value></Value>
  </Attribute>
  <Attribute name="cotlist">
    <Value>fedletcot</Value>
  </Attribute>
</XACMLPDPConfig>
```

When functioning as IDP, OpenAM can take many other `Attribute` values. These are implementation dependent. You can obtain the extended metadata from OpenAM either as part of the pre-packaged Java Fedlet that you create by using the OpenAM console wizard as described in Procedure 6.1, "To Create Configuration Files for a Fedlet", or by using the `ssoadm create-metadata-templ` subcommand.

Note

Custom authentication contexts can be loaded and saved when they are loaded via `ssoadm` as part of the hosted IDP/SP extended metadata and the saves are made in the console. Any custom contexts loaded via `ssoadm` are also visible in the console.

For example, you can specify custom entries in the `idpAuthncontextClassrefMapping` element of the extended metadata for a hosted IDP as follows:

```
<Attribute name="idpAuthncontextClassrefMapping">
  <Value>urn:oasis:names:tc:SAML:2.0:ac:classes:PasswordProtectedTransport
    |1||default</Value>
  <Value>http://idmanagement.gov/ns/assurance/loa/4|4||</Value>
  <Value>http://idmanagement.gov/ns/assurance/loa/3|3||</Value>
  <Value>http://idmanagement.gov/ns/assurance/loa/2|2||</Value>
  <Value>http://idmanagement.gov/ns/assurance/loa/1|1||</Value>
</Attribute>
```

6.2.4.3. Identity Provider Extended XML: IDPSSOConfig Settings

This section covers elements for the IDP SSO role, arranged in the order they appear in the template.

description

Description of the file.

cotlist

Specifies the circle of trust(s) to which the provider belongs.

Default: `fedletcot`

6.2.4.4. Identity Provider Extended XML: Attribute Authority Configuration Settings

This section covers the element for the AttributeAuthorityConfig role, arranged in the order they appear in the template.

cotlist

Specifies the circle of trust(s) to which the provider belongs.

Default: `fedletcot`

6.2.4.5. Identity Provider Extended XML: XACML PDP Configuration

This section covers the elements for the XACMLPDPConfig role, arranged in the order they appear in the template.

wantXACMLAuthzDecisionQuerySigned

If the IdP requires signed XACML AuthzDecision queries, then set this attribute to `true`.

cotlist

Specifies the circle of trust(s) to which the provider belongs.

Default: `fedletcot`

6.2.5. Configuring the Service Providers

As mentioned in Section 6.2.1, "Java Fedlet Layout", the Fedlet SAML v2.0 configuration is defined in two XML files, the standard metadata in `sp.xml` and the extended metadata in `sp-extended.xml`.

If the Fedlet has multiple service provider personalities, then number the configuration files, as in `sp.xml`, `sp2.xml`, `sp3.xml`, and also `sp-extended.xml`, `sp2-extended.xml`, `sp3-extended.xml` and so on.

6.2.5.1. Service Provider Standard XML

This section covers the configuration in `sp.xml`. The `sp.xml` file contains standard SAML v2.0 metadata for the Fedlet as SP. If you edit the standard metadata, make sure that you provide the new version to your IDP, as the IDP software relies on the metadata to get the Fedlet's configuration.

The standard metadata are covered in *Metadata for the OASIS Security Assertion Markup Language (SAML) V2.0*. The standard XML namespace describing the XML document has identifier `urn:oasis:names:tc:SAML:2.0:metadata`. An XML schema description for this namespace is found online at <http://docs.oasis-open.org/security/saml/v2.0/saml-schema-metadata-2.0.xsd>.

A standard metadata file describes the SAML v2.0 roles that the Fedlet plays. The default base element of the file is an `EntityDescriptor`, which is a container for role descriptor elements. The `EntityDescriptor` element can therefore contain multiple role descriptor elements. The namespace for the standard metadata document is `urn:oasis:names:tc:SAML:2.0:metadata`. You can get the corresponding XML schema description online at <http://docs.oasis-open.org/security/saml/v2.0/saml-schema-metadata-2.0.xsd>. In general, you can find standard SAML v2.0-related XML schema definitions at <http://docs.oasis-open.org/security/saml/v2.0/>.

Fedlets do not support all arbitrary SP configurations. As lightweight Service Provider components, Fedlets are built to play the SP role in web single sign-on and single logout, to perform attribute queries and XACML policy decision requests, and to work with multiple IDPs including Circles of Trust with an IDP discovery service. For a list of what Fedlets support, see the table Table 12.2, "Fedlet Support for SAML v2.0 Features" in the *Administration Guide*.

When preparing a standard SP metadata file, follow these suggestions.

- Start either with an existing example or with the template, `conf/sp.xml-template`.
- When using the template, replace the following placeholders.

FEDLET_ENTITY_ID

The Fedlet entity ID used when communicating with the IDP.

OpenAM often uses the deployment URL as the entity ID, though that is a convention rather than a requirement.

FEDLET_PROTOCOL

The Fedlet deployment protocol ([http](#), [https](#))

FEDLET_HOST

The Fedlet deployment host name

FEDLET_PORT

The Fedlet deployment port number

FEDLET_DEPLOY_URI

The Fedlet application deployment path

- Add and edit role elements as children depending on the roles the Fedlet plays as described in the following sections.

6.2.5.1.1. Single Sign-On and Logout: SPSSODescriptor Element

Add an SPSSODescriptor element to play the SP role in web single sign-on and logout. An SPSSODescriptor element has attributes specifying whether requests and assertion responses should be digitally signed.

- The **AuthnRequestsSigned** attribute indicates whether the Fedlet signs authentication requests.

If you set the **AuthnRequestsSigned** attribute to true, then you must also configure the SPSSODescriptor element to allow the Fedlet to sign requests. For details see the section on Section 6.1.2, "Enabling Signing and Encryption in a Fedlet".

- The **WantAssertionsSigned** attribute indicates whether the Fedlet requests signed assertion responses from the IDP.

An SPSSODescriptor element's children indicate what name ID formats the Fedlet supports, and where the IDP can call the following services on the Fedlet.

- The AssertionConsumerService elements specify endpoints that support the SAML Authentication Request protocols.

You must specify at least one of these. The template specifies two, with the endpoint supporting the HTTP POST binding as the default.

- The optional SingleLogoutService elements specify endpoints that support the SAML Single Logout protocols.

6.2.5.1.2. Attribute Queries: RoleDescriptor Element

Add a RoleDescriptor element with **type="query:AttributeQueryDescriptorType"** to perform attribute queries.

Attribute queries require the IDP to act as Attribute Authority and call for signing and encryption to be configured for the Fedlet. For details see the example in the procedure Procedure 6.3, "To Use the Fedlet to Query an Attribute Authority". For example, you can set the attribute mapping on the Fedlet by editing the extended metadata attribute `attributeMap` in the `SPSSOConfig` element as described in Section 6.2.5.2.1, "Service Provider Extended XML: SPSSOConfig Settings".

6.2.5.1.3. XACML Requests: XACMLAuthzDecisionQueryDescriptor Element

Add an `XACMLAuthzDecisionQueryDescriptor` element to perform XACML policy decision queries.

Attribute queries require the IDP to act as XACML PDP. For details see the example in the procedure Procedure 6.4, "To Use the Fedlet to Query an XACML Policy Decision Point".

6.2.5.2. Service Provider Extended XML

This section covers the configuration in the `sp-extended.xml` file. The extended metadata are specific to the OpenAM implementation of SAML v2.0.

The extended metadata file describes an `EntityConfig` element, defined by the namespace with the identifier `urn:sun:fm:SAML:2.0:entityconfig`. The XML schema definition is described in `entity-config-schema.xsd`, available online as part of the OpenAM source code, though not included with the unconfigured Fedlet.

The unconfigured Fedlet does include a template file, `conf/sp-extended.xml-template`. This extended metadata template for the IDP requires that you edit at least the `FEDLET_ENTITY_ID` placeholder value, the `appLogoutUrl` attribute value in the `SPSSOConfig` element, and the `fedletcot` values. The `FEDLET_ENTITY_ID` value must reflect the SP entity ID used in the standard metadata. For the single logout profile, the `appLogoutUrl` attribute value must match the Fedlet URL based on the values used in the `FederationConfig.properties` file. The `fedletcot` values must correspond to the Circle of Trust name defined in `fedlet.cot`.

The `hosted` attribute on the `EntityConfig` element must remain set to `hosted="1"`, meaning that the SP is hosted (local to the Fedlet). If you provide a copy of the file to your IDP running OpenAM, however, then set `hosted="0"` for the IDP, as the Fedlet is remote to the IDP.

The extended metadata file is essentially a series of XML maps of key-value pairs specifying IDP configuration for each role. All role-level elements can take a `metaAlias` attribute that the Fedlet uses when communicating with the IDP. Each child element of a role element defines an Attribute whose `name` is the key. Each Attribute element can contain multiple `Value` elements. The `Value` elements' contents comprise the values for the key. All values are strings, sometimes with a format that is meaningful to the Fedlet. The basic example in the SP template shows the configuration options, documented in the following lists.

6.2.5.2.1. Service Provider Extended XML: SPSSOConfig Settings

This section covers elements for the SP SSO role, arranged in the order they appear in the template.

description

Human-readable description of the Fedlet in the SP SSO role

signingCertAlias

Alias of the public key certificate for the key pair used when signing messages to the IDP

The key pair is found in the Fedlet's keystore, and the certificate is included in the standard metadata. See [Public and Private Key Settings](#) for details on how to specify access to the keystore, and Section 6.2.5.1, "Service Provider Standard XML" for details on how to set up standard metadata.

encryptionCertAlias

Alias of the public key certificate for the key pair used when encrypting messages to the IDP

The key pair is found in the Fedlet's keystore, and the certificate is included in the standard metadata. See [Public and Private Key Settings](#) for details on how to specify access to the keystore, and Section 6.2.5.1, "Service Provider Standard XML" for details on how to set up standard metadata.

basicAuthOn

Set this to true to use HTTP Basic authorization with the IDP.

Default: false

basicAuthUser

When using HTTP Basic authorization with the IDP, this value is the user name.

basicAuthPassword

When using HTTP Basic authorization with the IDP, this value is the password.

Encrypt the password using the [encode.jsp](#) page of your test copy of OpenAM that you might also have used to encode keystore passwords as described in [Public and Private Key Settings](#).

autofedEnabled

Set this to true to enable automatic federation with OpenAM based on the value of a profile attribute that is common to user profiles both in OpenAM and in the Fedlet's context.

Default: false

autofedAttribute

When automatic federation is enabled, set this to the name of the user profile attribute used for automatic federation.

transientUser

Use this effective identity for users with transient identifiers.

Default: anonymous

spAdapter

Class name for a plugin service provider adapter

This class must extend `com.sun.identity.saml2.plugins.SAML2ServiceProviderAdapter`.

spAdapterEnv

When using a plugin service provider adapter, this attribute's values optionally take a map of settings *key=value* used to initialize the plugin.

fedletAdapter

Class name for an alternate fedlet adapter. Default is an empty value.

fedletAdapterEnv

When using an alternate fedlet adapter, this attribute's values optionally take a map of settings *key=value* used to initialize the plugin.

spAccountMapper

Class name for an implementation mapping SAML protocol objects to local user profiles

Default: `com.sun.identity.saml2.plugins.DefaultLibrarySPAccountMapper`

spAttributeMapper

Class name for an implementation mapping SAML assertion attributes to local user profile attributes

Default: `com.sun.identity.saml2.plugins.DefaultSPAttributeMapper`

spAuthnContextMapper

Class name for an implementation determining the authentication context to set in an authentication request, and mapping the authentication context to an authentication level

Default: `com.sun.identity.saml2.plugins.DefaultSPAuthnContextMapper`

spAuthnContextClassrefMapping

String defining how the SAML authentication context classes map to authentication levels and indicate the default context class

Format: `authnContextClass|authLevel[|default]`

Default: `urn:oasis:names:tc:SAML:2.0:ac:classes:PasswordProtectedTransport|0|default`

`spAuthnContextComparisonType`

How to evaluate authentication context class identifiers.

`exact`

Assertion context must exactly match a context in the list

`minimum`

Assertion context must be at least as strong as a context in the list

`maximum`

Assertion context must be no stronger than a context in the list

`better`

Assertion context must be stronger than all contexts in the list

Default: `exact`

`attributeMap`

Map of SAML assertion attributes to local user profile attributes

Default: `*=*`

`saml2AuthModuleName`

Name of an alternative SAML v2.0 authentication module

`localAuthURL`

URL to a login page on the Fedlet side

Use this to override the Assertion Consumer Service URL from the standard metadata when consuming assertions.

`intermediateUrl`

URL to an intermediate page returned before the user accesses the final protected resource

`defaultRelayState`

If no RelayState is specified in a SAML request, redirect to this URL after successful single sign-on.

URL-encode the `defaultRelayState` value.

`appLogoutUrl`

One or more Fedlet URLs that initiate single logout

Replace the placeholders in the default with the values for your Fedlet.

Default: `FEDLET_PROTOCOL://FEDLET_HOST:FEDLET_PORT/FEDLET_DEPLOY_URI/logout`

`assertionTimeSkew`

Tolerate clock skew between the Fedlet and the IDP of at most this number of seconds

Default: 300

`wantAttributeEncrypted`

Set to true to request that the IDP encrypt attributes in the response

`wantAssertionEncrypted`

Set to true to request that the IDP encrypt the SAML assertion in the response

`wantNameIDEncrypted`

Set to true to request that the IDP encrypt the name ID in the response

`wantPOSTResponseSigned`

Set to true to request that the IDP sign the response when using HTTP POST

`wantArtifactResponseSigned`

Set to true to request that the IDP sign the response when using HTTP Artifact

`wantLogoutRequestSigned`

Set to true to request that the IDP sign single logout requests

`wantLogoutResponseSigned`

Set to true to request that the IDP sign single logout responses

`wantMNIRRequestSigned`

Set to true to request that the IDP manage name ID requests

`wantMNIRResponseSigned`

Set to true to request that the IDP manage name ID responses

`cotlist`

Set this to the Circle of Trust name used in Section 6.2.3, "Configuring Circles of Trust".

Default: `fedletcot`

`saeAppSecretList`

When using Secure Attribute Exchange with OpenAM this represents the Application Security Configuration settings.

Values take the format `url=FedletURL|type=symmetric|secret=EncodedSharedSecret[|encryptionalgorithm=EncAlg|encryptionkeystrength=EncStrength]` or `url=FedletURL|type=asymmetric|privatekeyalias=FedletSigningCertAlias[|encryptionalgorithm=EncAlg|encryptionkeystrength=EncStrength|pubkeyalias=FedletPublicKeyAlias]`

You can omit the `privatekeyalias` setting if the signing certificate is specified in the standard metadata.

`saeSPUrl`

When using Secure Attribute Exchange (SAE) with OpenAM this is the Fedlet URL that handles SAE requests. If this is omitted, then SAE is not enabled.

`saeSPLogoutUrl`

When using Secure Attribute Exchange with OpenAM this is the Fedlet URL that handles SAE global logout requests.

`ECPRequestIDPListFinderImpl`

When using the Enhanced Client and Proxy profile this is the class name for the implementation that returns a list of preferred IDPs trusted by the ECP.

Default: `com.sun.identity.saml2.plugins.ECPIDPFinder`

`ECPRequestIDPList`

When using the Enhanced Client and Proxy profile this is the list of IDPs for the ECP to contact.

When not specified the list finder implementation is used.

`enableIDPProxy`

Set this to true to enable IDP proxy functionality.

Default: false

`idpProxyList`

A list of preferred IDPs that the Fedlet can proxy to

idpProxyCount

Number of IDP proxies that the Fedlet can have

Default: 0

useIntroductionForIDPProxy

Set this to true to pick a preferred IDP based on a SAML v2.0 introduction cookie.

Default: false

6.2.5.2.2. Service Provider Extended XML: AttributeQueryConfig Settings

This section covers elements for the Attribute Requester role, arranged in the order they appear in the template.

signingCertAlias

Alias of the public key certificate for the key pair used when signing messages to the IDP

The key pair is found in the Fedlet's keystore, and the certificate is included in the standard metadata. See [Public and Private Key Settings](#) for details on how to specify access to the keystore, and [Section 6.2.5.1, "Service Provider Standard XML"](#) for details on how to set up standard metadata.

encryptionCertAlias

Alias of the public key certificate for the key pair used when encrypting messages to the IDP

The key pair is found in the Fedlet's keystore, and the certificate is included in the standard metadata. See [Public and Private Key Settings](#) for details on how to specify access to the keystore, and [Section 6.2.5.1, "Service Provider Standard XML"](#) for details on how to set up standard metadata.

wantNameIDEncrypted

Set to true to request that the IDP encrypt the name ID

cotlist

Set this to the Circle of Trust name used in [Section 6.2.3, "Configuring Circles of Trust"](#).

Default: `fedletcot`

6.2.5.2.3. Service Provider Extended XML: XACMLAuthzDecisionQueryConfig Settings

This section covers elements for the XACML decision requester role, enabling the Fedlet to act as a Policy Enforcement Point, arranged in the order they appear in the template.

signingCertAlias

Alias of the public key certificate for the key pair used when signing messages to the IDP

The key pair is found in the Fedlet's keystore, and the certificate is included in the standard metadata. See [Public and Private Key Settings](#) for details on how to specify access to the keystore, and [Section 6.2.5.1, "Service Provider Standard XML"](#) for details on how to set up standard metadata.

encryptionCertAlias

Alias of the public key certificate for the key pair used when encrypting messages to the IDP

The key pair is found in the Fedlet's keystore, and the certificate is included in the standard metadata. See [Public and Private Key Settings](#) for details on how to specify access to the keystore, and [Section 6.2.5.1, "Service Provider Standard XML"](#) for details on how to set up standard metadata.

basicAuthOn

Set to true to use HTTP Basic authorization when contacting the Policy Decision Provider

Default: false

basicAuthUser

When using Basic authorization to contact the Policy Decision Provider, use this value as the user name

basicAuthPassword

When using Basic authorization to contact the Policy Decision Provider, use this value as the password

Encrypt the password using the [encode.jsp](#) page of your test copy of OpenAM that you might also have used to encode keystore passwords as described in [Public and Private Key Settings](#).

wantXACMLAuthzDecisionResponseSigned

Set this to true to request that the Policy Decision Provider sign the XACML response.

wantAssertionEncrypted

Set this to true to request that the Policy Decision Provider encrypt the SAML assertion response

cotlist

Set this to the Circle of Trust name used in [Section 6.2.3, "Configuring Circles of Trust"](#).

Default: `fedletcot`

6.2.6. Embedding the Java Fedlet in a Web Application

The Fedlet war file, `Fedlet-13.5.2.war`, serves both as an example and also to provide the code needed to embed the Fedlet in your web application.

The basic steps for using the Fedlet in your application are as follows.

- Unpack the Fedlet war file to a working directory, remove any files you do not want to keep such as `index.jsp` or `fedletEncode.jsp`, and overlay the Fedlet files with those of your web application.
- To integrate single sign-on into your application, modify the functionality in `fedletSampleApp.jsp` or add it to your application's logic.

If you add it to your application's logic, then you must also edit your application's deployment descriptor file, `web.xml`, to set the assertion consumer URI, which by default is `/fedletapplication` in the basic SP XML for the Fedlet. Add `servlet` and `servlet-mapping` elements as shown in the following example.

```
<servlet>
  <servlet-name>yourapplication</servlet-name>
  <jsp-file>/your-application.jsp</jsp-file>
</servlet>
<servlet-mapping>
  <servlet-name>yourapplication</servlet-name>
  <url-pattern>/fedletapplication</url-pattern>
</servlet-mapping>
```

- Build a war file from your web application with embedded Fedlet files.

This is the version of the application to deploy.

- When you deploy your war file, also provide the Fedlet configuration as described in this section.

Chapter 7

Working With the Security Token Service

This chapter covers tasks developers perform when working with OpenAM's Security Token Service (STS).

The Security Token Service transforms security tokens upon request. For example, you can call the Security Token Service to convert a username and password token to a SAML v2.0 token. For a complete description of the Security Token Service, including operational flow, supported token types, configuration, and deployment, see Chapter 17, "Configuring the Security Token Service" in the *Administration Guide*.

This section covers tasks that developers perform when working with the Security Token Service:

- Section 7.1, "Publishing STS Instances"
- Section 7.2, "Consuming STS Instances"
- Section 7.3, "Querying, Validating, and Canceling Tokens"
- Section 7.4, "Extending STS to Support Custom Token Types"

Several sections in this chapter reference STS code examples. The following procedure describes how to access the examples:

Procedure 7.1. To Access the STS Example Code

1. If you have not already done so, download and build the STS 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. Check out the `master` branch of the OpenAM source.

You can find the STS code examples under `/path/to/openam-samples-external/sts-example-code`.

7.1. Publishing STS Instances

You configure STS *instances* to perform one or more token transformations. Each instance provides configuration details about how SAML v2.0 and/or OpenID Connect output tokens are encrypted

or signed. Deployments that support multiple SAML v2.0 and/or OpenID Connect service providers require multiple STS instances.

Publishing an STS instance means creating an STS instance with a given configuration.

OpenAM supports two types of STS instances: REST STS instances and SOAP STS instances. REST STS instances provide token transformations by letting users call REST API endpoints, while SOAP STS instances provide token transformations by letting users call WS-Trust 1.4-compliant SOAP endpoints.

OpenAM provides two techniques for publishing STS instances:

- Creating and configuring the instance by using the OpenAM console
- Executing code that calls the `sts-publish` REST endpoint

Chapter 17, "*Configuring the Security Token Service*" in the *Administration Guide* describes how to create and configure STS instances by using the OpenAM console. This chapter covers how to publish STS instances programmatically.

When you publish a REST STS instance, OpenAM exposes a REST endpoint for accessing the instance, and the instance is immediately available for use to callers.

For SOAP STS instances, there is an additional deployment step. In order for the SOAP endpoint to be exposed, a SOAP STS deployment must exist for the realm in which the SOAP STS instance was created. A SOAP STS deployment is a running web application separate from OpenAM. For information about creating SOAP STS deployments, see Section 17.7, "Deploying SOAP STS Instances" in the *Administration Guide*.

7.1.1. The Publish Service

To publish an STS instance, perform an HTTP POST on one of the `sts-publish` endpoints:

- `/sts-publish/rest`, for REST STS instances
- `/sts-publish/soap`, for SOAP STS instances

Specify the `_action=create` parameter in the URL.

For example, you could publish a REST STS instance named `username-transformer` in the Top Level Realm as follows:

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

```

"invocation_context": "invocation_context_client_sdk",
"instance_state": {
  "saml2-config": {
    "issuer-name": "saml2-issuer",
    ...
  },
  "deployment-config": {
    "deployment-url-element": "username-transformer",
    "deployment-realm": "/",
    ...
  },
  "persist-issued-tokens-in-cts": "false",
  "supported-token-transforms": [{
    "inputTokenType": "USERNAME",
    "outputTokenType": "OPENIDCONNECT",
    "invalidateInterimOpenAMSession": false
  }],
  "oidc-id-token-config": {
    "oidc-issuer": "test",
    ...
  }
} \
https://openam.example.com:8443/openam/sts-publish/rest?_action=create
{
  "_id": "username-transformer",
  "_rev": "21939129",
  "result": "success",
  "url_element": "username-transformer"
}

```

The `instance_state` object in the JSON payload represents the STS instance's configuration. For a complete example of an `instance_state` object, see the sample code for the `RestSTSInstancePublisher` class in Section 7.1.2, "Publishing REST STS Instances".

Accessing the `sts-publish` endpoint requires administrative privileges. Authenticate as an OpenAM administrative user, such as `amadmin`, before attempting to publish an STS instance.

In addition to publishing instances, the `sts-publish` endpoint can also return the configuration of an STS instance when you perform an HTTP GET on the `sts-publish` endpoint for the instance. The endpoint you access differs for REST and SOAP STS instances:

- For REST STS instances, access `/sts-publish/rest/realm/deployment-URL-element`
- For SOAP STS instances, access `/sts-publish/soap/realm/deployment-URL-element`

In the preceding examples, `deployment-URL-element` is the value of the STS instance's deployment URL element—one of the instance's configuration properties. `realm` is the realm in which a SOAP STS instance has been configured.

For example, you could obtain the configuration of a REST STS instance configured in the Top Level Realm with the deployment URL element `username-transformer` as follows:

```
$ curl \
--request GET \
--header "iPlanetDirectoryPro: AQIC5..." \
https://openam.example.com:8443/openam/sts-publish/rest/username-transformer
{
  "_id": "username-transformer",
  "_rev": "-659999943",
  "username-transformer": {
    "saml2-config": {
      "issuer-name": "saml2-issuer",
      ...
    },
    "deployment-config": {
      "deployment-url-element": "username-t-transformer",
      ...
    },
    "persist-issued-tokens-in-cts": "false",
    "supported-token-transforms": [{
      "inputTokenType": "USERNAME",
      "outputTokenType": "OPENIDCONNECT",
      "invalidateInterimOpenAMSession": false
    }],
    "oidc-id-token-config": {
      "oidc-issuer": "test",
      ...
    }
  }
}
```

You can delete STS instances by performing an HTTP DELETE on the `sts-publish` endpoint. The endpoint you access differs for REST and SOAP STS instances:

- For REST STS instances, perform an HTTP DELETE on `/sts-publish/rest/realm/deployment-URL-element`
- For SOAP STS instances, perform an HTTP DELETE on `/sts-publish/soap/realm/deployment-URL-element`

7.1.2. Publishing REST STS Instances

The sample code referenced in this section provides an example of how to programmatically publish REST STS instance. The code is not intended to be a working example. Rather, it is a starting point—code that you can modify to satisfy your organization's specific requirements. To access the sample code, see Procedure 7.1, "To Access the STS Example Code".

After publishing a REST STS instance programmatically, you can view the instance's configuration in the OpenAM console. The instance is ready for consumption.

Sample code is available for the following classes:

RestSTSInstancePublisher

The `RestSTSInstancePublisher` class exposes an API to publish, delete, and update REST STS instances by calling methods that perform an HTTP POST operation on the `soap-sts/publish` endpoint.

RestSTSTInstanceConfigFactory

The `RestSTSTInstancePublisher` class calls the `RestSTSTInstanceConfigFactory` class to create a `RestSTSTInstanceConfig` instance. `RestSTSTInstanceConfig` objects encapsulate all the configuration information of a REST STS instance, and emit JSON values that you can post to the `sts-publish/rest` endpoint to publish a REST STS instance.

STSPublishContext

The sample `STSPublishContext` class specifies the configuration necessary to publish REST and SOAP STS instances. The class provides a programmatic method for setting configuration properties—the same configuration properties available through the OpenAM console under `Realms > Realm Name > STS`.

CustomTokenOperationContext

The sample `CustomTokenOperationContext` class specifies custom validators, token types, and transformations that a REST STS instance can support.

Important

The sample code referenced in this section is *not* compilable, because it uses classes that are not available publicly. The code provides patterns to developers familiar with the problem domain and is intended only to assist developers who want to programmatically publish REST STS instances.

The sample code imports a number of classes, introducing dependencies. Classes imported from the OpenAM client SDK can remain in your code, but other imported classes must be removed and replaced with code that provides similar functionality in your environment. For example, the `RestSTSTInstanceConfigFactory` class uses a constant named `CommonConstants.DEFAULT_CERT_MODULE_NAME` from the imported `com.forgerock.openam.functionaltest.sts.frmwk.common.CommonConstants` utility class. This utility class is not publicly available. Therefore, you need to replace this constant with another construct.

The critical part of the sample code is the idioms that programmatically set all the state necessary to publish a REST STS instance.

7.1.3. Publishing SOAP STS Instances

The sample code referenced in this section provides an example of how to programmatically publish of a SOAP STS instance. The code is not intended to be a working example. Rather, it is starter code that you can modify to satisfy your organization's specific requirements. To access the sample code, see Procedure 7.1, "To Access the STS Example Code".

After publishing a SOAP STS instance programmatically, you can view the instance's configuration in the OpenAM console. However, the instance is not ready for consumption until after you have created and deployed a SOAP STS `.war` file. For information about how to create and deploy a SOAP STS `.war` file, see Section 17.7, "Deploying SOAP STS Instances" in the *Administration Guide*.

Sample code is available for the following classes:

SoapSTSPublisher

The sample `SoapSTSPublisher` class exposes an API to publish, delete, and update SOAP STS instances by calling methods that perform an HTTP POST operation on the `soap-sts-publish/publish` endpoint.

SoapSTSPublisherConfigFactory

The sample `SoapSTSPublisher` class calls the `SoapSTSPublisherConfigFactory` class to create a `SoapSTSPublisherConfig` instance. `SoapSTSPublisherConfig` objects encapsulate all the configuration information of a SOAP STS instance, and emit JSON values that you can post to the `soap-sts-publish/publish` endpoint to publish a REST STS instance.

SoapSTSPublisherCryptoState

The sample `SoapSTSPublisherCryptoState` class specifies the configuration for the keystore used by a SOAP STS instance. The class provides a programmatic method for setting configuration properties—the same configuration properties available through the OpenAM console under `Realms > Realm Name > STS > Soap Keystore Configuration`.

STSPublisherContext

The sample `STSPublisherContext` class specifies the configuration necessary to publish REST and SOAP STS instances. The class provides a programmatic method for setting configuration properties—the same configuration properties available through the OpenAM console under `Realms > Realm Name > STS`.

Important

The sample code referenced in this section is *not* compilable, because it uses classes that are not available publicly. The code provides patterns to developers familiar with the problem domain and is intended only to assist developers who want to programmatically publish SOAP STS instances.

The sample code imports a number of classes, introducing dependencies. Classes imported from the OpenAM client SDK and the SOAP STS client SDK can remain in your code, but other imported classes must be removed and replaced with code that provides similar functionality in your environment. For example, the `SoapSTSPublisherConfigFactory` class uses a constant named `CommonConstants.DEFAULT_CERT_MODULE_NAME` from the imported `com.forgerock.openam.functionaltest.sts.frmwk.common.CommonConstants` utility class. This utility class is not publicly available. Therefore, you need to replace this constant with another construct.

The critical part of the sample code is the idioms that programmatically set all the state necessary to publish a SOAP STS instance.

7.2. Consuming STS Instances

Once REST and SOAP STS instance endpoints have been exposed, they are available for use to consumers as follows:

- Developers access REST STS instances by making REST API calls that support token transformations.

- Developers access SOAP STS instances by sending SOAP messages or by using the SOAP STS client SDK. OpenAM's SOAP STS is WS-Trust 1.4 compliant.

7.2.1. Consuming REST STS Instances

You consume a REST STS instance by sending REST API calls to the instance's endpoint.

7.2.1.1. REST STS Instance Endpoint

REST STS instances' endpoints comprise the following parts:

- The OpenAM context
- The string `rest-sts`
- The realm in which the REST STS instance is configured
- The deployment URL element, which is one of the configuration properties of an STS instance

For example, a REST STS instance configured in the realm `myRealm` with the deployment URL element `username-transformer` exposes the endpoint `/rest-sts/myRealm/username-transformer`.

7.2.1.2. JSON Representation of Token Transformations

Token transformations are represented in JSON as follows:

```
{
  "input_token_state": {
    "token_type": "INPUT_TOKEN_TYPE"
    ... INPUT_TOKEN_TYPE_PROPERTIES ...
  },
  "output_token_state": {
    "token_type": "OUTPUT_TOKEN_TYPE"
    ... OUTPUT_TOKEN_TYPE_PROPERTIES ...
  }
}
```

REST STS supports the following token types and properties:

- Input token types:
 - `USERNAME`
Requires the `username` and `password` properties.
 - `OPENAM`
Requires the `session_id` property, with an SSO token as its value.
 - `X509`

No properties are required, because input X.509 tokens are presented either in HTTP headers or by using TLS. For more information about X.509 tokens, see the configuration details for the Authentication Target Mappings and Client Certificate Header Key properties in Section 17.6.1, "REST STS Configuration Properties" in the *Administration Guide*.

- **OPENIDCONNECT**

Requires the `oidc_id_token` property, with the OpenID Connect token as its value.

- Output token types:

- **SAML2**

Requires the `subject_confirmation` property, the value of which determines the `<saml:ConfirmationMethod>` element for the generated SAML v2.0 assertion. Valid values are **BEARER**, **SENDER_VOUCHES**, and **HOLDER_OF_KEY**.

When generating an assertion with a holder-of-key subject confirmation method, the `proof_token_state` property is required. The value for this property is an object that contains the `base64EncodedCertificate` property.

- **OPENIDCONNECT**

Requires the `nonce` and `allow_access` properties.

The following are examples of JSON payloads that define REST STS token transformations:

1. Transform a username token to a SAML v2.0 token with the bearer subject confirmation method:

```
{
  "input_token_state": {
    "token_type": "USERNAME",
    "username": "demo",
    "password": "changeit"
  },
  "output_token_state": {
    "token_type": "SAML2",
    "subject_confirmation": "BEARER"
  }
}
```

2. Transform an X.509 token to a SAML v2.0 token with the sender vouches subject confirmation method:

```
{
  "input_token_state": {
    "token_type": "X509"
  },
  "output_token_state": {
    "token_type": "SAML2",
    "subject_confirmation": "SENDER_VOUCHES"
  }
}
```



```

"output_token_state": {
  "token_type": "SAML2",
  "subject_confirmation": "BEARER"
}
}
\
https://openam.example.com:8443/openam/rest-sts/username-transformer?_action=translate
{
  "issued_token":
    "<saml:Assertion
      xmlns:saml=\"urn:oasis:names:tc:SAML:2.0:assertion\"
      Version=\"2.0\"
      ID=\"s2c51ebd0ad10aae44fb76e4b400164497c63b4ce6\"
      IssueInstant=\"2016-03-02T00:14:47Z\">\n
      <saml:Issuer>saml2-issuer</saml:Issuer>
      <saml:Subject>\n
        <saml:NameID
          Format=\"urn:oasis:names:tc:SAML:1.1:nameid-format:emailAddress\">demo
        </saml:NameID>
        <saml:SubjectConfirmation
          Method=\"urn:oasis:names:tc:SAML:2.0:cm:bearer\">\n
          <saml:SubjectConfirmationData
            NotOnOrAfter=\"2016-03-02T00:24:47Z\" >
          </saml:SubjectConfirmationData>
          </saml:SubjectConfirmation>\n
        </saml:Subject>
        <saml:Conditions
          NotBefore=\"2016-03-02T00:14:47Z\"
          NotOnOrAfter=\"2016-03-02T00:24:47Z\">\n
          <saml:AudienceRestriction>\n
            <saml:Audience>saml2-issuer-entity</saml:Audience>\n
          </saml:AudienceRestriction>\n</saml:Conditions>\n
          <saml:AuthnStatement
            AuthnInstant=\"2016-03-02T00:14:47Z\">
            <saml:AuthnContext>
              <saml:AuthnContextClassRef>
                urn:oasis:names:tc:SAML:2.0:ac:classes:PasswordProtectedTransport
              </saml:AuthnContextClassRef>
            </saml:AuthnContext>
          </saml:AuthnStatement>
        </saml:Assertion>\n"
    }
}

```

The `iPlanetDirectoryPro` header is required and should contain the SSO token of an administrative user, such as `amAdmin`, who has access to perform the operation.

7.2.1.4. Java Example

The `RestSTSTokenConsumer.java` sample code provides an example of how to consume a published REST STS instance programmatically. Tailor this example as required to provide programmatic consumption of your own REST STS instances. To access the sample code, see Procedure 7.1, "To Access the STS Example Code".

Important

The sample code referenced in this section is *not* compilable, because it uses classes that are not available publicly. The code provides patterns to developers familiar with the problem domain and is intended only to assist developers who want to programmatically consume REST STS instances.

7.2.2. Consuming SOAP STS Instances

You consume a SOAP STS instance by sending it SOAP messages to the instance's endpoint, or by calling it using the OpenAM SOAP STS client SDK.

7.2.2.1. SOAP STS Instance URL

SOAP STS instances' URLs comprise the following parts:

- The SOAP STS deployment context
- The string `sts`
- The realm in which the REST STS instance is configured
- The deployment URL element, which is one of the configuration properties of an STS instance

The SOAP STS deployment context comprises the base URL of the web container to which the SOAP STS `.war` file is deployed and the deployment web application name. For more information about SOAP STS deployments, see Section 17.7.6, "Deploying the SOAP STS Instance to a Web Container" in the *Administration Guide*.

For example, a SOAP STS instance configured in the realm `myRealm` with the deployment URL element `soap-username-transformer` and the a deployment web application name `openam-soap-sts` would expose a URL similar to `https://soap-sts-host.com:8443/openam-soap-sts/sts/myRealm/soap-username-transformer`.

The WSDL for the service would be available at `https://soap-sts-host.com:8443/openam-soap-sts/sts/myRealm/soap-username-transformer?wsdl`.

7.2.2.2. Consuming SOAP STS Instances Using SOAP Messages

Because an OpenAM SOAP STS instance is a WS-Trust 1.4-compliant security token service, users can consume the instance by sending it standard WS-Trust 1.4 SOAP STS framework messages, such as `RequestSecurityToken` messages, passed as the payload to WSDL ports that are implemented by the security token services.

For more information about WS-Trust 1.4 security token services, see the WS-Trust 1.4 specification.

7.2.2.3. Consuming SOAP STS Instances Using the OpenAM SOAP STS Client SDK

You can consume an OpenAM SOAP STS instance by calling it using the OpenAM SOAP STS client SDK.

7.2.2.3.1. About the SOAP STS Client SDK

The SOAP STS client SDK is based on classes in Apache CXF, an open source service framework. Apache CXF provides the `org.apache.cxf.ws.security.trust.STSClient` class, which encapsulates consumption of a SOAP STS service. However, using this class requires considerable expertise.

The SOAP STS client SDK makes it easier to consume OpenAM SOAP STS instances than using Apache CXF for the following reasons:

- The `org.forgerock.openam.sts.soap.SoapSTSConsumer` class in the OpenAM SOAP STS client SDK wraps the Apache CXF class `org.apache.cxf.ws.security.trust.STSClient`, providing a higher level of abstraction that makes consumption of SOAP STS instances easier to achieve.
- The `SoapSTSConsumer` class' `issueToken`, `validateToken`, and `cancelToken` methods provide the three fundamental operations exposed by SOAP STS instances. Supporting classes facilitate the creation of state necessary to invoke these methods.
- Classes in the SDK provide logic to allow OpenAM session tokens to be presented in order to satisfy the security policy bindings that mandate OpenAM sessions as supporting tokens. The STS client obtains secret password state—keystore entry passwords and aliases, username token credential information, and so forth—from a callback handler. The `SoapSTSConsumerCallbackHandler` class provides the means to create a callback handler initialized with state that will be encountered when consuming SOAP STS instances. The `SoapSTSConsumerCallbackHandler` instance can be passed to an STS client. The `TokenSpecification` class provides a way to create the varying token state necessary to obtain specific tokens and create any necessary supporting state.

You can use the classes in the SOAP STS client SDK as is, or you can tailor them to your needs. For more information about the SOAP STS client SDK classes, see the source code and the Javadoc.

7.2.2.3.2. Building a SOAP STS Client SDK .jar File

The SOAP STS client SDK is not part of the OpenAM client SDK.¹ To use the SOAP STS client SDK, you must compile the source code for the SOAP STS client SDK and create a `.jar` file.

Procedure 7.2. To Build the SOAP STS Client SDK

1. Download the OpenAM source code.
2. Change to the `openam-sts/openam-soap-sts` directory.
3. Run the `mvn install` command.
4. Locate the `openam-soap-sts-client-13.5.2-4.jar` file in the `openam-sts/openam-soap-sts/openam-soap-sts-client/target` directory.

¹ The SOAP STS client SDK has a dependency on Apache CXF classes, which is not present in the OpenAM client SDK. Therefore, the two SDKs are not bundled together.

7.3. Querying, Validating, and Canceling Tokens

Both REST and SOAP STS instances support *token persistence*, which is the ability to store tokens issued for the STS instance in the Core Token Service (CTS). You enable token persistence for both REST and SOAP STS instances' configuration under Realms > *Realm Name* > STS > *STS Instance Name* > General Configuration > Persist Issued Tokens in Core Token Store. Tokens are saved in the CTS for the duration of the token lifetime, which is a configuration property for STS-issued SAML v2.0 and OpenID Connect tokens. Tokens with expired durations are periodically removed from the CTS.

With token persistence enabled for an STS instance, OpenAM provides the ability to query, validate, and cancel tokens issued for the instance:

- *Querying tokens* means listing tokens issued for an STS instance or for a user.
- *Validating a token* means verifying that the token is still present in the CTS.
- *Canceling a token* means removing the token from the CTS.

7.3.1. Invoking the sts-tokengen Endpoint

The `sts-tokengen` endpoint provides administrators with the ability to query and cancel tokens issued for both REST and SOAP STS instances using REST API calls.

When using the `sts-tokengen` endpoint, be sure to provide the token ID for an OpenAM administrator, such as `amadmin`, as the value of a header whose name is the name of the SSO token cookie, by default `iPlanetDirectoryPro`.

7.3.1.1. Querying Tokens

List tokens issued for an STS instance by using the `queryFilter` action in an HTTP GET call to the `sts-tokengen` endpoint with the `/sts-id` argument.

The following example lists all the tokens issued for the `username-transformer` STS instance. The results show that OpenAM has issued two OpenID Connect tokens for the `demo` user for the `username-transformer` STS instance:

```
$ curl \
--request GET \
--header "iPlanetDirectoryPro: AQIC5..." \
https://openam.example.com:8443/openam/sts-tokengen\
?_queryFilter=\ /sts_id+eq+\ 'username-transformer'\
{
  "result": [
    {
      "_id": "B663D248CE4C3B63A7422000B03B8F5E0F8E443B",
      "_rev": "",
      "token_id": "B663D248CE4C3B63A7422000B03B8F5E0F8E443B",
      "sts_id": "username-transformer",
      "principal_name": "demo",
```

```

    "token_type": "OPENIDCONNECT",
    "expiration_time": 1459376096
  },
  {
    "_id": "7CB70009970D1AAFF177AC2A08D58405EDC35DF5",
    "_rev": "",
    "token_id": "7CB70009970D1AAFF177AC2A08D58405EDC35DF5",
    "sts_id": "username-transformer",
    "principal_name": "demo",
    "token_type": "OPENIDCONNECT",
    "expiration_time": 1459376098
  }
],
"resultCount": 2,
"pagedResultsCookie": null,
"totalPagedResultsPolicy": "NONE",
"totalPagedResults": -1,
"remainingPagedResults": -1
}

```

List tokens issued for a particular user with the `queryFilter` action in an HTTP GET call to the `sts-tokengen` endpoint with the `/token-principal` argument.

The following example lists all the tokens issued for the `demo` user. The results show that OpenAM has issued two OpenID Connect tokens:

```

$ curl \
--request GET \
--header "iPlanetDirectoryPro: AQIC5..." \
https://openam.example.com:8443/openam/sts-tokengen\
?_queryFilter=/token_principal+eq+'demo' \
{
  "result": [
    {
      "_id": "B663D248CE4C3B63A7422000B03B8F5E0F8E443B",
      "_rev": "",
      "token_id": "B663D248CE4C3B63A7422000B03B8F5E0F8E443B",
      "sts_id": "username-transformer",
      "principal_name": "demo",
      "token_type": "OPENIDCONNECT",
      "expiration_time": 1459376096
    },
    {
      "_id": "7CB70009970D1AAFF177AC2A08D58405EDC35DF5",
      "_rev": "",
      "token_id": "7CB70009970D1AAFF177AC2A08D58405EDC35DF5",
      "sts_id": "username-transformer",
      "principal_name": "demo",
      "token_type": "OPENIDCONNECT",
      "expiration_time": 1459376098
    }
  ],
  "resultCount": 2,
  "pagedResultsCookie": null,
  "totalPagedResultsPolicy": "NONE",
  "totalPagedResults": -1,
  "remainingPagedResults": -1
}

```

```
}
```

7.3.1.2. Cancelling Tokens

Cancel tokens by making an HTTP DELETE call to the `sts-tokengen/token_id` endpoint:

```
$ curl \
--request DELETE \
--header "iPlanetDirectoryPro: AQIC5..." \
https://openam.example.com:8443/openam/sts-tokengen/B663D248CE4C3B63A7422000B03B8F5E0F8E443B
{
  "id": "B663D248CE4C3B63A7422000B03B8F5E0F8E443B",
  "_rev": "B663D248CE4C3B63A7422000B03B8F5E0F8E443B",
  "result": "token with id B663D248CE4C3B63A7422000B03B8F5E0F8E443B successfully removed."
}
```

7.3.2. Validating and Cancelling Tokens by Invoking a REST STS Instance

REST STS users can validate and cancel tokens by making an HTTP POST call to a REST STS instance's endpoint.

To validate a token, use the `validate` action. The following example validates an OpenID Connect token previously issued by the `username-transformer` REST STS instance:

```
$ curl \
--request POST \
--header "iPlanetDirectoryPro: AQIC5..." \
--header "Content-Type: application/json" \
--data '{
  "validated_token_state": {
    "token_type": "OPENIDCONNECT",
    "oidc_id_token": "eyJhdHlwIjogIkpXVCIsIC..."
  }
}' \
https://openam.example.com:8443/openam/rest-sts/username-transformer?action=validate
{
  "token_valid": true
}
```

To cancel a token, use the `cancel` action. The following example cancels an OpenID Connect token previously issued by the `username-transformer` REST STS instance:


```
$ curl \
--request POST \
--header "iPlanetDirectoryPro: AQIC5..." \
--header "Content-Type: application/json" \
--data '{
  "cancelled_token_state": {
    "token_type": "OPENIDCONNECT",
    "oidc_id_token": "eyJhdHlwIjogIkpzcVciOiIC..."
  }
}' \
https://openam.example.com:8443/openam/rest-sts/username-transformer?_action=cancel
{
  "result": "OPENIDCONNECT token cancelled successfully."
}
```

7.3.3. Validating and Cancelling Tokens by Invoking a SOAP STS Instance

The source code for the `validateToken` and `cancelToken` methods in the `org.forgerock.openam.sts.soap.SoapSTSTokenConsumer` class provides information needed to construct WS-Trust 1.4-compliant calls for validating and cancelling tokens.

Locate the `org.forgerock.openam.sts.soap.SoapSTSTokenConsumer` class under `openam-sts/openam-soap-sts/openam-soap-sts-client` in the OpenAM source code.

7.4. Extending STS to Support Custom Token Types

OpenAM supports token transformations to and from a variety of token types, including username, SAML v2.0, OpenID Connect, and X.509. In addition to these supported token types, REST STS instances can use custom token types as the input or output token, or both, in a token transformation. When you configure a REST STS instance to support a token transformation that takes a custom token type, you can also configure a custom validator and provider class for the custom token type. OpenAM uses custom validator classes to validate custom tokens and custom provider classes to produce custom tokens.

Specify custom token validator and provider classes in the OpenAM console by configuring the Custom Token Validators and Custom Token Providers properties under Realms > *Realm Name* > STS > *REST STS Instance Name*.

A custom validator class can be used in transformations that produce standard STS output tokens, such as SAML v2.0 tokens or OpenID Connect tokens, and in transformations that produce custom output token types.

A custom provider class can be used in token transformations that take standard STS input tokens, such as username tokens or OpenAM SSO tokens, and in transformations that take custom input token types.

Before a REST STS instance can use a custom token type validator or provider class, you must bundle the class into the OpenAM `.war` file and restart OpenAM.

OpenAM invokes a single instance of a validator or provider class to run all concurrently dispatched token transformations that use the custom token type. Because there is only a single instance of the class, you must code custom validator and provider classes to be thread-safe.

7.4.1. Developing Custom Token Type Validator Classes

To create a custom token type validator class, implement the `org.forgerock.openam.sts.rest.token.validator.RestTokenTransformValidator` class.

Custom token type validator classes implement the `validateToken` method. This method takes a `RestTokenValidatorParameters` object as input. Note that the generic type of `RestTokenValidatorParameters` is `org.forgerock.json.fluent.JsonValue`. As a result of using this type, custom validator classes can access the JSON representation of the input token passed to the REST STS instance in the `input_token_state` JSON key.

The `validateToken` method returns an `org.forgerock.openam.sts.rest.token.validator.RestTokenTransformValidatorResult` object. At a minimum, this object contains the OpenAM SSO token corresponding to the validated principal. It can also contain additional information specified as a JSON value, allowing a custom validator to pass extra state to a custom provider in a token transformation.

7.4.2. Developing Custom Token Type Provider Classes

To create a custom token type provider class, implement the `org.forgerock.openam.sts.rest.token.provider.RestTokenProvider` class.

Custom token type provider classes implement the `createToken` method. This method takes an `org.forgerock.openam.sts.rest.token.provider.CustomRestTokenProviderParameters` object as input. This object gives the custom provider access to the following information:

- The principal returned by the `RestTokenTransformValidator`
- The OpenAM SSO token corresponding to the validated principal
- Any additional state returned in the `RestTokenValidatorResult` object
- The type of input token validated by the `RestTokenTransformValidator` in the token transformation
- The `JsonValue` corresponding to this validated token, as specified by the `input_token_state` object in the transformation request
- The `JsonValue` corresponding to the `token_output_state` object specified in the token transformation request (which can provide additional information pertinent to the creation of the output token)

The `createToken` method returns a string representation of the custom token in a format that can be transmitted across HTTP in JSON. It should be base64-encoded if binary.

7.4.3. Using Custom Token Type Validators and Providers

This section provides an example of how to use custom token type validators and providers.

The example assumes that you already configured a token transformation by completing the following tasks:

- Implementing the `RestTokenTransformValidator` interface to create a custom token type validator
- Implementing the `RestTokenProvider` interface to create a custom token type provider
- Bundling the two classes into the OpenAM `.war` file
- Restarting OpenAM
- Publishing a REST STS instance with a custom token type named `CUSTOM`, specifying the custom validator and provider classes in the instance's configuration

To transform a `CUSTOM` token to an OpenID Connect token, you might specify a JSON payload similar to the following:

```
{
  "input_token_state":
  {
    "token_type": "CUSTOM",
    "extra_stuff": "very_useful_state"
  },
  "output_token_state":
  {
    "token_type": "OPENIDCONNECT",
    "nonce": "1234",
    "allow_access": true
  }
}
```

With the preceding JSON payload, OpenAM passes a `JsonValue` instance to the `validateToken` method of the custom token type validator class as follows:

```
{
  "token_type": "CUSTOM",
  "extra_stuff": "very_useful_state"
}
```

To transform a username token to a `CUSTOM` token, you might specify a JSON payload similar to the following:

```
{
  "input_token_state":
    {
      "token_type": "USERNAME",
      "username": "unt_user17458687",
      "password": "password"
    },
  "output_token_state":
    {
      "token_type": "CUSTOM",
      "extra_stuff_for_custom": "some_useful_information"
    }
}
```

With the preceding JSON payload, OpenAM passes the following information to the `createToken` method of the custom token type provider:

- The principal returned by the `USERNAME` token validator: `unt_user17458687`.
- The OpenAM SSO token corresponding to this authenticated principal.
- Additional state returned by the token validator, if any. Because the `USERNAME` token validator does not return any additional state, the additional state for this example would be null.
- The input token type: `CUSTOM`
- A `JsonValue` representation of the following:

```
{
  "token_type": "USERNAME",
  "username": "unt_user17458687",
  "password": "password"
}
```

- A `JsonValue` representation of the following:

```
{
  "token_type": "CUSTOM",
  "extra_stuff_for_custom": "some_useful_information"
}
```

To transform a `CUSTOM` token to a `CUSTOM` token, you might specify a JSON payload similar to the following:

```
{
  "input_token_state":
    {
      "token_type": "CUSTOM",
      "extra_stuff": "very_useful_state"
    },
  "output_token_state":
    {
      "token_type": "CUSTOM",
      "extra_stuff_for_custom": "some_useful_information"
    }
}
```

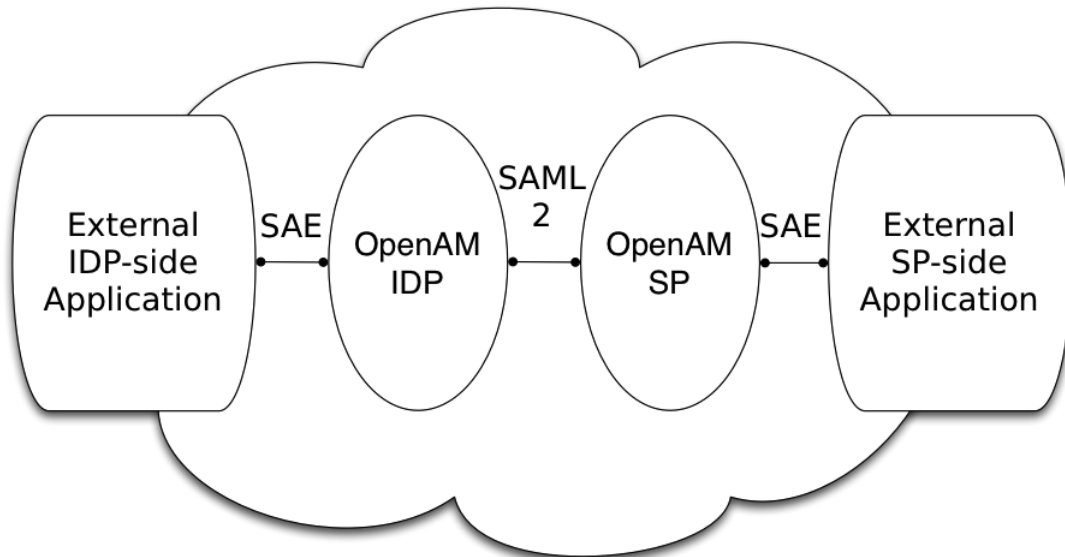
The input to the custom validator and provider would be similar to the preceding examples, with the possible addition of any additional state that the custom validator returned from the `validateToken` method.

Chapter 8

Using Secure Attribute Exchange

Most deployments can rely on OpenAM to handle authentication and provide identity assertions. OpenAM supports 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 v2.0 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 tells OpenAM 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 v2.0 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. 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.

8.1. 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 Section 12.4, "Configuring Identity Providers, Service Providers, and Circles of Trust" 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 Realms > *Realm Name* > Test Federation Connectivity in the OpenAM console to make sure Federation is working before you add secure attribute exchange applications that rely on functioning SAML v2.0 communications between the providers.

Set up the sample web application as described in Section 2.2.1, "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.

8.2. 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 sets 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. For example, you can 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 keystore on the application side. Also store the public key from OpenAM which is acting as the provider in the application's keystore. Make note of the certificate aliases for your application's private key, and for OpenAM's public key. Also note the path to the keystore for your application, the keystore password, and the private key password.

8.3. Securing the Identity Provider Side

This configuration uses the default sample settings with a shared secret of `secret12`, without encryption of the attributes:

1. Log in as `amadmin` to the OpenAM server console where you set up the hosted identity provider (IDP).
2. The sample includes a `branch` attribute not found in user profiles by default. Therefore, under `Realms > Realm Name > Authentication > Settings > User Profile`, 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/sam12/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.

8.4. 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. The sample includes a `branch` attribute not found in user profiles by default. Therefore, under Realms > *Realm Name* > Authentication > Settings > User Profile, 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.

8.5. 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.idpidentityid http://idp.example.com:8080/openam
sun.spidentityid http://sp.example.com:8080/openam
branch        mainbranch
sun.authlevel 0
```

Appendix A. Deprecated REST APIs

This appendix provides information about REST APIs deprecated in OpenAM 13.5.2-4.

A.1. Deprecated Session Information APIs

Interface Stability: Deprecated

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}
```

The `getMaxTime` action has been deprecated in favor of `getTimeLeft`. For more information, see Section 2.1.9.2, "Obtaining Information About Sessions".

A.2. Deprecated Self-Service APIs

Interface Stability: Deprecated

For information about the new self-service APIs, see Section 2.1.15, "RESTful User Self-Service".

A.2.1. Legacy 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.

Procedure A.1. To Set Up Legacy User Self-Registration

1. Configure the Email Service.

You must configure the Email Service to send mail notifications to users who self-register. To configure these globally in OpenAM console, navigate to Configure > Global Services, and then click Email Service.

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

2. Configure Legacy User Self Service.

You must enable self-registration in the User Self Service service. To configure these globally, in OpenAM console navigate to Configure > Global Services, and then click Legacy User Self Service. On the Legacy User Self Service page, click the **Enabled** checkbox next to Legacy Self-Service REST Endpoint, and 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.

Note

In OpenAM 13, the [/users](#) endpoint was updated to version 3.0. Request API resource version 2.0 in the REST API calls to get the behavior provided in previous versions of OpenAM.

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" \
--header "Accept-API-Version: protocol=1.0,resource=2.0" \
--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.

- 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=f4x0Dh6iZCXtX8nhiSb3xahNxr%3D
&email=newuser%40example.com
&tokenId=yA26LZ6SxFEgNuF86%2FSIXfimGlg%3D
&realm=/
```

- 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" \
--header "Accept-API-Version: protocol=1.0,resource=2.0" \
--data \
'{"email": "newuser@example.com",
  "tokenId": "yA26LZ6SxFEgNuF86/SIXfimGlg=",
  "confirmationId": "f4x0Dh6iZCXtX8nhiSb3xahNxr%3D"
}' \
https://openam.example.com:8443/openam/json/users?_action=confirm
{
  "email": "newuser@example.com",
  "tokenId": "yA26LZ6SxFEgNuF86/SIXfimGlg=",
  "confirmationId": "f4x0Dh6iZCXtX8nhiSb3xahNxr%3D"
}
```

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" \
--header "Accept-API-Version: protocol=1.0,resource=2.0" \
--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"
  ],
  "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.

A.2.2. Legacy Forgotten Password Reset

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 Chapter 8, "Configuring User Self-Service Features" in the *Administration Guide*.

Tip

If the current password is known, use the Section 2.1.16.1.7, "Changing Passwords" feature to change a password.

An example follows, showing the steps in more detail.

Procedure A.2. To Set Up Legacy Forgotten Password Reset

1. Configure Legacy User Self Service.

You must enable Forgotten Password Reset in the Legacy User Self Service service. To configure this globally in the OpenAM console, navigate to **Configure > Global Services**, and then click **Legacy User Self Service**. On the **Legacy User Self Service** page, click the **Enabled** checkbox next to **Legacy Self-Service REST Endpoint**, and **Forgot Password for Users**, and then click **Save**.

2. Configure the Email Service.

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

To configure the service globally in the OpenAM Console, navigate to **Configure > Global Services**, and then click **Email Service**.

Alternatively, you can configure it for an individual realm under **Realms > 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.

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

Note

In OpenAM 13, the `/users` endpoint was updated to version 3.0. Request API resource version 2.0 in the REST API calls to get the behavior provided in previous versions of OpenAM.

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"
\
--header "Accept-API-Version: protocol=1.0,resource=2.0"
\
--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"
\
--header "Accept-API-Version: protocol=1.0,resource=2.0"
\
--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.

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+5v58cTs1yQcQl5HCQG0suQk=
&username=demo&realm=/
```

- 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"
\
--header "Accept-API-Version: protocol=1.0,resource=2.0"
\
--data \
' {
  "username": "demo",
  "tokenId": "vkm+5v58cTs1yQcQl5HCQG0suQk=",
  "confirmationId": "sdfsfeM+URcWVQ7vvCDnx4N5Vut7SBIY="
} ' \
https://openam.example.com:8443/openam/json/users?_action=confirm
{
  "username": "demo",
  "tokenId": "vkm+5v58cTs1yQcQl5HCQG0suQk=",
  "confirmationId": "sdfsfeM+URcWVQ7vvCDnx4N5Vut7SBIY="
}
```

The response is a further confirmation that the request is valid, has not expired, and the password can be reset.

- 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"
\
--header "Accept-API-Version: protocol=1.0,resource=2.0"
\
--data ' {
  "username": "demo",
  "userpassword": "password",
  "tokenId": "vkm+5v58cTs1yQcQl5HCQG0suQk=",
  "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.

Index

A

- Authentication
 - Java API, 178
 - Post authentication plugins, 241

D

- Dashboard services, 137

F

- Fedlets
 - Java
 - Manual, 300
 - Wizard, 279
 - SAML v2.0, 279

I

- Installing
 - C SDK, 197
 - Java SDK samples, 174

O

- OAuth 2.0, 203
 - REST API, 96
- OpenID Connect 1.0
 - API, 117

P

- Passwords
 - Change, 156
 - Reset, 155
- Policy
 - Java API, 187
 - REST API, 32

R

- Realm data
 - REST access, 157
- Realms
 - REST, 10
- Recording
 - Using the REST API, 170

- Resource Types, 43
- REST API, 32, 96, 141

S

- Scripts, 249
 - API, 253
 - Managing, 162
 - OIDC Claims, 275
 - Policy Conditions, 267
 - Server-side Authentication, 263
- Secure Attribute Exchange (SAE), 344
- Security Token Service, 324
- Session tokens
 - Java API, 183

T

- Two-step verification
 - resetting device profile, 140

U

- UMA
 - Extending, 245
- User data
 - Custom profile attributes, 200
 - Custom repository, 234
 - REST access, 141
- User self-service
 - REST API, 122
- User-Managed Access (UMA)
 - REST API, 120