Development Guide

ForgeRock Access Management 5.1

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

Abstract

Guide to developing client applications and service providers. ForgeRock® Access Management provides authentication, authorization, entitlement and federation software.

Table of Contents

Preface

This guide provides an introduction to three ForgeRock Access Management APIs: the REST API, the Java SDK, and the C SDK.

This guide is an introduction for developers who adapt client applications to use Access Management.

For more specific examples of the customizations you can write, see the list below:

- Custom OAuth 2.0 scopes plugins define how Access Management, when playing the role of authorization server, handles scopes, including which token information to return for scopes set when authorization was granted.

  For more information, see Section 4.1, "Customizing OAuth 2.0 Scope Handling" in the *OAuth 2.0 Guide*.

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

  For more information, see Section 10.1, "Creating a Custom Authentication Module" in the *Authentication and Single Sign-On Guide*.

- 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.

  For more information, see Section 10.3, "Creating a Post Authentication Plugin" in the *Authentication and Single Sign-On Guide*.

- 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.

  For more information, see Section 3.1, "Customizing Policy Evaluation With a Plug-In" in the *Authorization Guide*.

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

  For more information, see Section 3.3.2, "Customizing Identity Data Storage" in the *Setup and Maintenance Guide*.

About ForgeRock Identity Platform™ Software

ForgeRock Identity Platform™ is the only offering for access management, identity management, user-managed access, directory services, and an identity gateway, designed and built as a single, unified platform.

The platform includes the following components that extend what is available in open source projects to provide fully featured, enterprise-ready software:

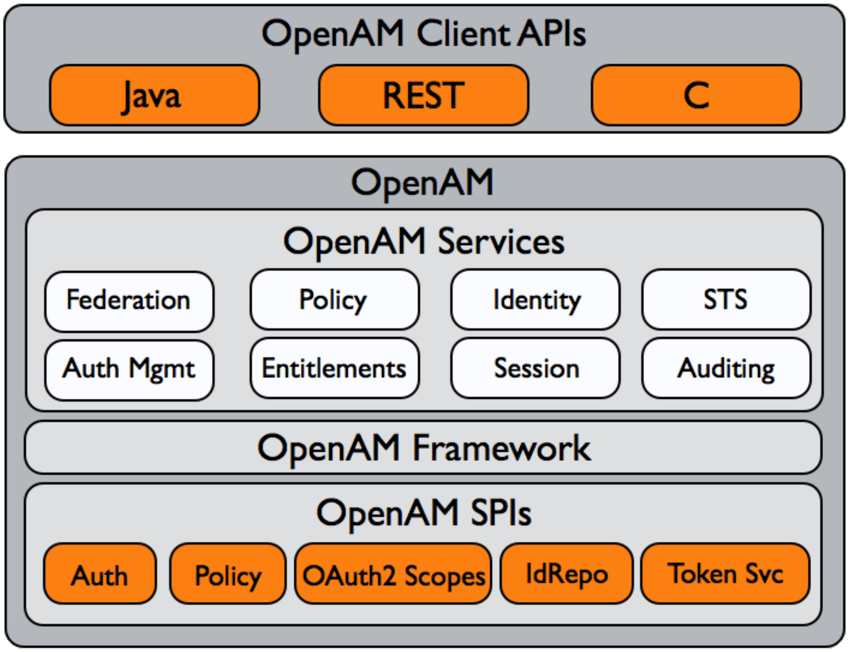
- ForgeRock Access Management (AM)
- ForgeRock Identity Management (IDM)
- ForgeRock Directory Services (DS)
- ForgeRock Identity Gateway (IG)

Chapter 1
Introducing APIs and Protocols

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

AM 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 AM capabilities to meet even those deployment requirements not yet covered in AM.

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



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

- AM 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.

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

  Detailed reference information is provided in the *ForgeRock Access Management Java SDK API Specification*.

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

1.1. IPv4 and IPv6

AM 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 with the REST API

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

2.1. Introducing REST

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).

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

Interface Stability: Evolving

Most native AM REST APIs use the CREST verbs. (In contrast, OAuth 2.0, OpenID Connect 1.0 and UMA APIs follow their respective standards.)

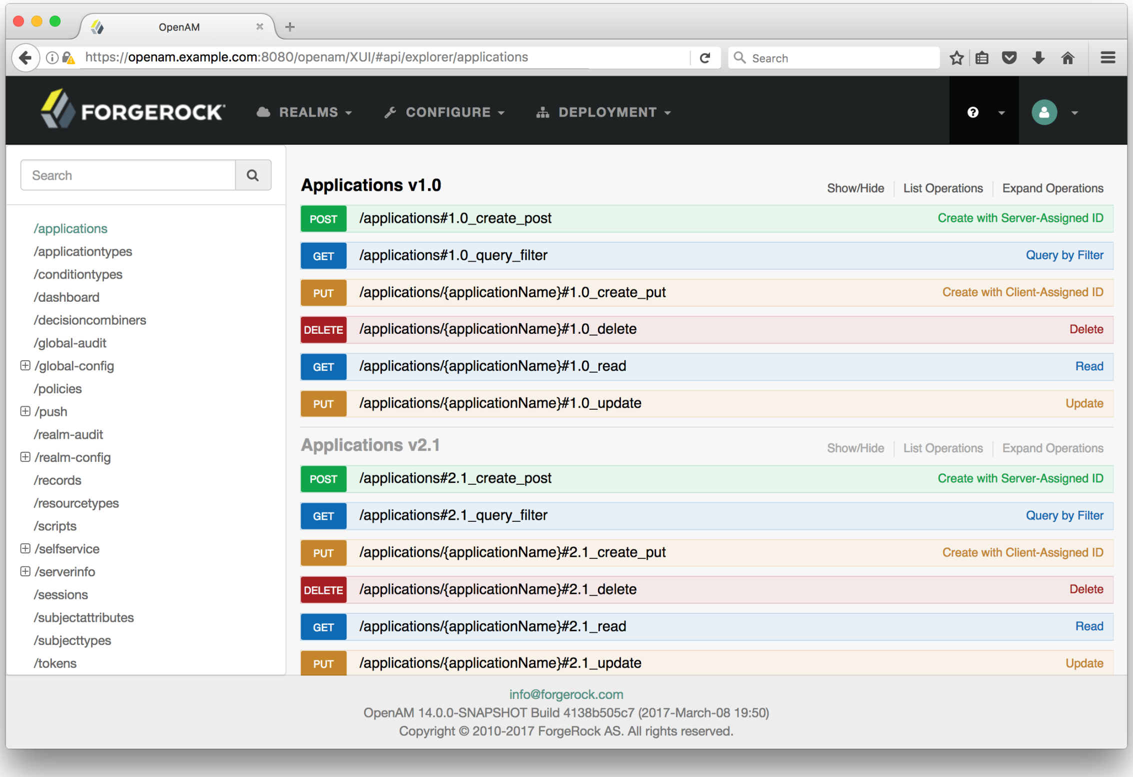
2.2. Introducing the API Explorer

AM provides an online AM REST API reference that can be accessed through the AM console. The API provides useful reference information for developers to create client applications to access AM's services.

The API Explorer displays the REST API endpoints that allow client applications to access the AM's services. The key features of the API Explorer are the following:

- **API Versioning**. The API Explorer displays the different API versions available depending on your deployment.

Figure 2.1. API Explorer



/applications
/applicationtypes
/conditiontypes
/dashboard
/decisioncombiners
/global-audit
⊞ /global-config
/policies
⊞ /push
/realm-audit
⊞ /realm-config
/records
/resourcetypes
/scripts
⊞ /selfservice
⊞ /serverinfo
/sessions
/subjectattributes
/subjecttypes
/tokens

Applications v1.0

POST	/applications#1.0_create_post	Create with Server-Assigned ID
GET	/applications#1.0_query_filter	Query by Filter
PUT	/applications/{applicationName}#1.0_create_put	Create with Client-Assigned ID
DELETE	/applications/{applicationName}#1.0_delete	Delete
GET	/applications/{applicationName}#1.0_read	Read
PUT	/applications/{applicationName}#1.0_update	Update

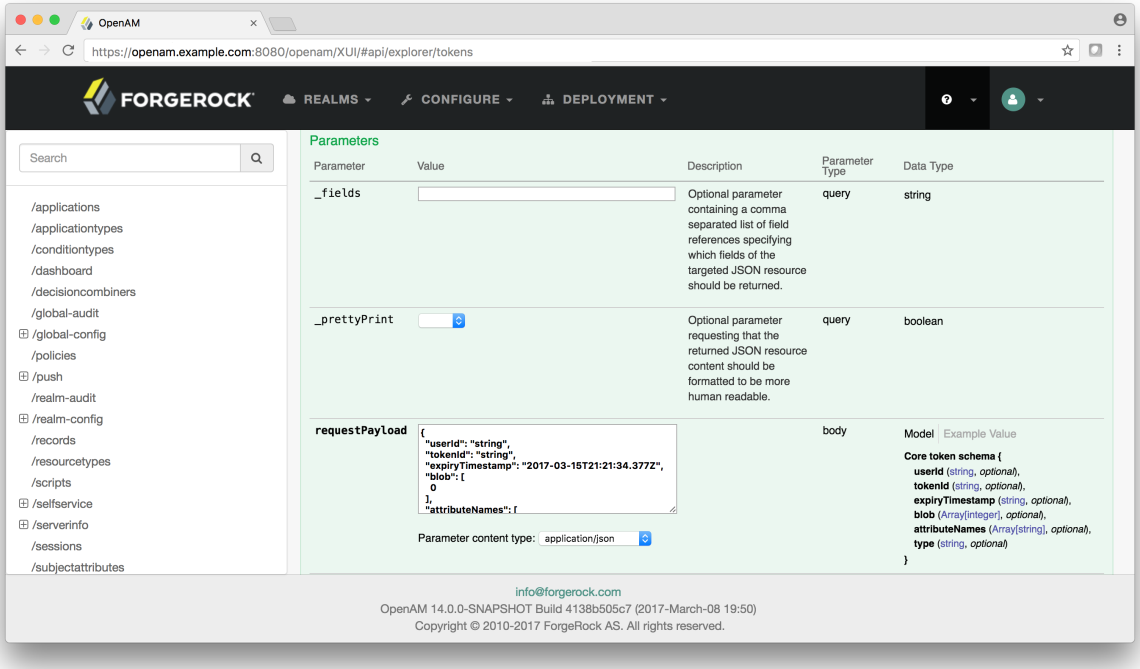
Applications v2.1

POST	/applications#2.1_create_post	Create with Server-Assigned ID
GET	/applications#2.1_query_filter	Query by Filter
PUT	/applications/{applicationName}#2.1_create_put	Create with Client-Assigned ID
DELETE	/applications/{applicationName}#2.1_delete	Delete
GET	/applications/{applicationName}#2.1_read	Read
PUT	/applications/{applicationName}#2.1_update	Update

info@forgerock.com
OpenAM 14.0.0-SNAPSHOT Build 4138b505c7 (2017-March-08 19:50)
Copyright © 2010-2017 ForgeRock AS. All rights reserved.

- **Detailed Information**. The API Explorer provides an Expand Operations button for each available CRUDPAQ method. When Expand Operations is pressed, you can view implementation notes, successful response class, headers, parameters, and response messages with examples. For example, the `requestPayload` field can be populated with an example value. Also, if you select `Model`, you can view the schema for each parameter, as seen below:

Figure 2.2. API Explorer Request Payload



Parameters

Parameter	Value	Description	Parameter Type	Data Type
`_fields`		Optional parameter containing a comma separated list of field references specifying which fields of the targeted JSON resource should be returned.	query	string
`_prettyPrint`		Optional parameter requesting that the returned JSON resource content should be formatted to be more human readable.	query	boolean
`requestPayload`	see below		body	Model / Example Value

```
{
 "userId": "string",
 "tokenId": "string",
 "expiryTimestamp": "2017-03-15T21:21:34.377Z",
 "blob": [
  0
 ],
 "attributeNames": [
```

Parameter content type: application/json

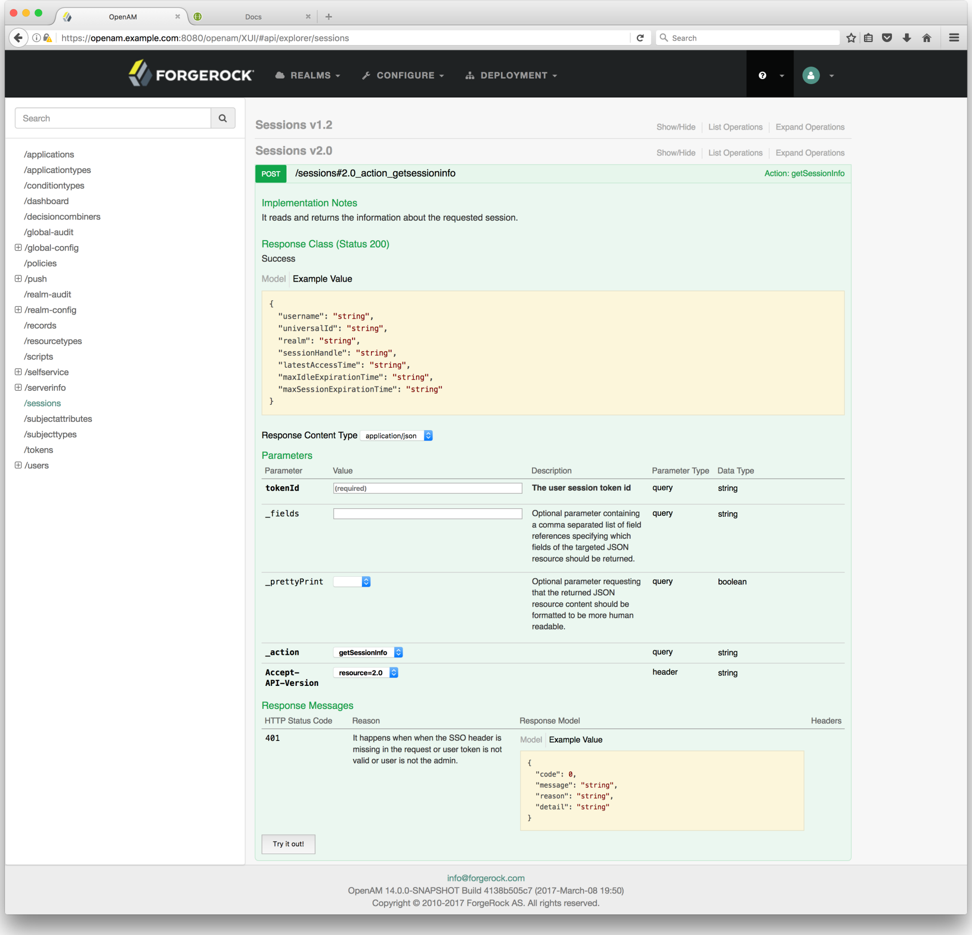
Model | Example Value

Core token schema {
userId (string, *optional*),
tokenId (string, *optional*),
expiryTimestamp (string, *optional*),
blob (Array[integer], *optional*),
attributeNames (Array[string], *optional*),
type (string, *optional*)
}

info@forgerock.com
OpenAM 14.0.0-SNAPSHOT Build 4138b505c7 (2017-March-08 19:50)
Copyright © 2010-2017 ForgeRock AS. All rights reserved.

- **Try It Out**. The API Explorer also provides a Try It Out feature, which allows you to send a sample request to the endpoint and view the possible responses.

Figure 2.3. API Explorer Detailed Information



Sessions v1.2

Sessions v2.0

POST /sessions#2.0_action_getsessioninfo Action: getSessionInfo

Implementation Notes

It reads and returns the information about the requested session.

Response Class (Status 200)

Success

```
{
  "username": "string",
  "universalId": "string",
  "realm": "string",
  "sessionHandle": "string",
  "latestAccessTime": "string",
  "maxIdleExpirationTime": "string",
  "maxSessionExpirationTime": "string"
}
```

Response Content Type application/json

Parameters

Parameter	Value	Description	Parameter Type	Data Type
tokenId	(required)	The user session token id	query	string
_fields		Optional parameter containing a comma separated list of field references specifying which fields of the targeted JSON resource should be returned.	query	string
_prettyPrint		Optional parameter requesting that the returned JSON resource content should be formatted to be more human readable.	query	boolean
_action	getSessionInfo		query	string
Accept-API-Version	resource=2.0		header	string

Response Messages

HTTP Status Code	Reason	Response Model	Headers
401	It happens when when the SSO header is missing in the request or user token is not valid or user is not the admin.	Model Example Value	

```
{
  "code": 0,
  "message": "string",
  "reason": "string",
  "detail": "string"
}
```

info@forgerock.com
OpenAM 14.0.0-SNAPSHOT Build 4138b505c7 (2017-March-08 19:50)
Copyright © 2010-2017 ForgeRock AS. All rights reserved.

Procedure 2.1. To Access the API Explorer

1. Log into the AM console as an administrator.

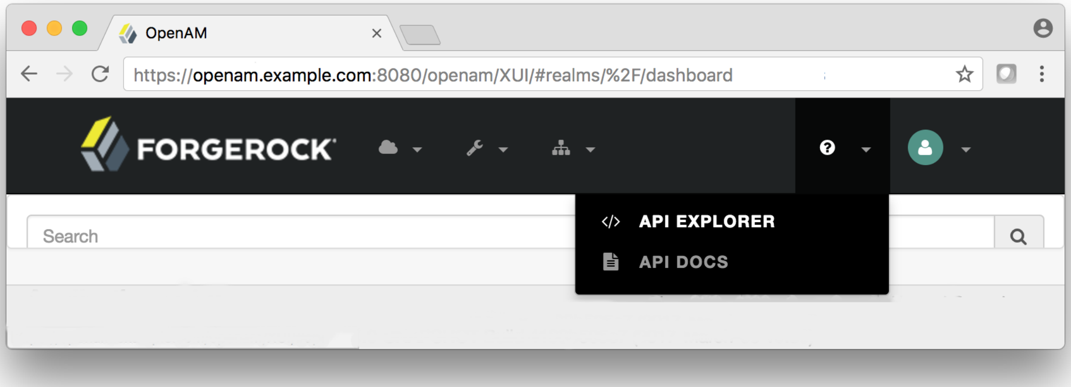
2. You can access the API Explorer in one of two ways:

   Point your browser to the following URL:

   ```
   https://openam.example.com:8080/openam/XUI/#api/explorer/applications
   ```

You can also click the help icon in the top-right corner, and then click API Explorer.

Figure 2.4. API Explorer


OpenAM
https://openam.example.com:8080/openam/XUI/#realms/%2F/dashboard
FORGEROCK
Search
API EXPLORER
API DOCS

2.3. About ForgeRock Common REST

ForgeRock® Common REST is a common REST API framework. It works across the ForgeRock platform to provide common ways to access web resources and collections of resources. Adapt the examples in this section to your resources and deployment.

2.3.1. Common REST Resources

Servers generally return JSON-format resources, though resource formats can depend on the implementation.

Resources in collections can be found by their unique identifiers (IDs). IDs are exposed in the resource URIs. For example, if a server has a user collection under `/users`, then you can access a user at `/users/user-id`. The ID is also the value of the `_id` field of the resource.

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.

2.3.2. Common REST Verbs

The Common REST APIs use the following verbs, sometimes referred to collectively as CRUDPAQ. For details and HTTP-based examples of each, follow the links to the sections for each verb.

Create

Add a new resource.

This verb maps to HTTP PUT or HTTP POST.

For details, see Section 2.3.6, "Create".

Read

Retrieve a single resource.

This verb maps to HTTP GET.

For details, see Section 2.3.7, "Read".

Update

Replace an existing resource.

This verb maps to HTTP PUT.

For details, see Section 2.3.8, "Update".

Delete

Remove an existing resource.

This verb maps to HTTP DELETE.

For details, see Section 2.3.9, "Delete".

Patch

Modify part of an existing resource.

This verb maps to HTTP PATCH.

For details, see Section 2.3.10, "Patch".

Action

Perform a predefined action.

This verb maps to HTTP POST.

For details, see Section 2.3.11, "Action".

Query

Search a collection of resources.

This verb maps to HTTP GET.

For details, see Section 2.3.12, "Query".

2.3.3. Common REST Parameters

Common REST reserved query string parameter names start with an underscore, `_`.

Reserved query string parameters include, but are not limited to, the following names:

`_action`
`_api`
`_crestapi`
`_fields`
`_mimeType`
`_pageSize`
`_pagedResultsCookie`
`_pagedResultsOffset`
`_prettyPrint`
`_queryExpression`
`_queryFilter`
`_queryId`
`_sortKeys`
`_totalPagedResultsPolicy`

> **Note**
>
> Some parameter values are not safe for URLs, so URL-encode parameter values as necessary.

Continue reading for details about how to use each parameter.

2.3.4. Common REST Extension Points

The *action* verb is the main vehicle for extensions. For example, to create a new user with HTTP POST rather than HTTP PUT, you might use `/users?_action=create`. A server can define additional actions. For example, `/tasks/1?_action=cancel`.

A server can define *stored queries* to call by ID. For example, `/groups?_queryId=hasDeletedMembers`. Stored queries can call for additional parameters. The parameters are also passed in the query string. Which parameters are valid depends on the stored query.

2.3.5. Common REST API Documentation

Common REST APIs often depend at least in part on runtime configuration. Many Common REST endpoints therefore serve *API descriptors* at runtime. An API descriptor documents the actual API as it is configured.

Use the following query string parameters to retrieve API descriptors:

`_api`

Serves an API descriptor that complies with the OpenAPI specification.

This API descriptor represents the API accessible over HTTP. It is suitable for use with popular tools such as Swagger UI.

`_crestapi`

Serves a native Common REST API descriptor.

This API descriptor provides a compact representation that is not dependent on the transport protocol. It requires a client that understands Common REST, as it omits many Common REST defaults.

> **Note**
>
> Consider limiting access to API descriptors in production environments in order to avoid unnecessary traffic.
>
> To provide documentation in production environments, see Procedure 2.2, "To Publish OpenAPI Documentation" instead.

Procedure 2.2. To Publish OpenAPI Documentation

In production systems, developers expect stable, well-documented APIs. Rather than retrieving API descriptors at runtime through Common REST, prepare final versions, and publish them alongside the software in production.

Use the OpenAPI-compliant descriptors to provide API reference documentation for your developers as described in the following steps:

1. Configure the software to produce production-ready APIs.

   In other words, the software should be configured as in production so that the APIs are identical to what developers see in production.

2. Retrieve the OpenAPI-compliant descriptor.

   The following command saves the descriptor to a file, `myapi.json`:

   ```
   $ curl -o myapi.json endpoint?_api
   ```

3. If necessary, edit the descriptor.

   For example, you might want to add security definitions to describe how the API is protected.

   If you make any changes, then also consider using a source control system to manage your versions of the API descriptor.

4. Publish the descriptor using a tool such as Swagger UI.

   You can customize Swagger UI for your organization as described in the documentation for the tool.

2.3.6. Create

There are two ways to create a resource, either with an HTTP POST or with an HTTP PUT.

To create a resource using POST, perform an HTTP POST with the query string parameter `_action=create` and the JSON resource as a payload. Accept a JSON response. The server creates the identifier if not specified:

```
POST /users?_action=create HTTP/1.1
Host: example.com
Accept: application/json
Content-Length: ...
Content-Type: application/json
{ JSON resource }
```

To create a resource using PUT, perform an HTTP PUT including the case-sensitive identifier for the resource in the URL path, and the JSON resource as a payload. Use the `If-None-Match: *` header. Accept a JSON response:

```
PUT /users/some-id HTTP/1.1
Host: example.com
Accept: application/json
Content-Length: ...
Content-Type: application/json
If-None-Match: *
{ JSON resource }
```

The `_id` and content of the resource depend on the server implementation. The server is not required to use the `_id` that the client provides. The server response to the create request indicates the resource location as the value of the `Location` header.

If you include the `If-None-Match` header, its value must be `*`. In this case, the request creates the object if it does not exist, and fails if the object does exist. If you include the `If-None-Match` header with any value other than `*`, the server returns an HTTP 400 Bad Request error. For example, creating an object with `If-None-Match: revision` returns a bad request error. If you do not include `If-None-Match: *`, the request creates the object if it does not exist, and *updates* the object if it does exist.

Parameters

You can use the following parameters:

`_prettyPrint=true`

   Format the body of the response.

_fields=*field*[,*field*...]

Return only the specified fields in the body of the response.

The `field` values are JSON pointers. For example if the resource is `{"parent":{"child":"value"}}`, `parent/child` refers to the `"child":"value"`.

2.3.7. Read

To retrieve a single resource, perform an HTTP GET on the resource by its case-sensitive identifier (`_id`) and accept a JSON response:

```
GET /users/some-id HTTP/1.1
Host: example.com
Accept: application/json
```

Parameters

You can use the following parameters:

_prettyPrint=true

Format the body of the response.

_fields=*field*[,*field*...]

Return only the specified fields in the body of the response.

The `field` values are JSON pointers. For example if the resource is `{"parent":{"child":"value"}}`, `parent/child` refers to the `"child":"value"`.

_mimeType=*mime-type*

Some resources have fields whose values are multi-media resources such as a profile photo for example.

By specifying both a single *`field`* and also the *`mime-type`* for the response content, you can read a single field value that is a multi-media resource.

In this case, the content type of the field value returned matches the *`mime-type`* that you specify, and the body of the response is the multi-media resource.

The `Accept` header is not used in this case. For example, `Accept: image/png` does not work. Use the `_mimeType` query string parameter instead.

2.3.8. Update

To update a resource, perform an HTTP PUT including the case-sensitive identifier (`_id`) as the final element of the path to the resource, and the JSON resource as the payload. Use the `If-Match: _rev` header to check that you are actually updating the version you modified. Use `If-Match: *` if the version does not matter. Accept a JSON response:

```
PUT /users/some-id HTTP/1.1
Host: example.com
Accept: application/json
Content-Length: ...
Content-Type: application/json
If-Match: _rev
{ JSON resource }
```

When updating a resource, include all the attributes to be retained. Omitting an attribute in the resource amounts to deleting the attribute unless it is not under the control of your application. Attributes not under the control of your application include private and read-only attributes. In addition, virtual attributes and relationship references might not be under the control of your application.

Parameters

You can use the following parameters:

`_prettyPrint=true`

Format the body of the response.

`_fields=field[,field...]`

Return only the specified fields in the body of the response.

The `field` values are JSON pointers. For example if the resource is `{"parent":{"child":"value"}}`, `parent/child` refers to the `"child":"value"`.

2.3.9. Delete

To delete a single resource, perform an HTTP DELETE by its case-sensitive identifier (`_id`) and accept a JSON response:

```
DELETE /users/some-id HTTP/1.1
Host: example.com
Accept: application/json
```

Parameters

You can use the following parameters:

`_prettyPrint=true`

Format the body of the response.

`_fields=field[,field...]`

Return only the specified fields in the body of the response.

The `field` values are JSON pointers. For example if the resource is `{"parent":{"child":"value"}}`, `parent/child` refers to the `"child":"value"`.

2.3.10. Patch

To patch a resource, send an HTTP PATCH request with the following parameters:

- `operation`
- `field`
- `value`
- `from` (optional with copy and move operations)

You can include these parameters in the payload for a PATCH request, or in a JSON PATCH file. If successful, you'll see a JSON response similar to:

```
PATCH /users/some-id HTTP/1.1
Host: example.com
Accept: application/json
Content-Length: ...
Content-Type: application/json
If-Match: _rev
{ JSON array of patch operations }
```

PATCH operations apply to three types of targets:

- **single-valued**, such as an object, string, boolean, or number.
- **list semantics array**, where the elements are ordered, and duplicates are allowed.
- **set semantics array**, where the elements are not ordered, and duplicates are not allowed.

ForgeRock PATCH supports several different `operations`. The following sections show each of these operations, along with options for the `field` and `value`:

2.3.10.1. Patch Operation: Add

The `add` operation ensures that the target field contains the value provided, creating parent fields as necessary.

If the target field is single-valued, then the value you include in the PATCH replaces the value of the target. Examples of a single-valued field include: object, string, boolean, or number.

An `add` operation has different results on two standard types of arrays:

- **List semantic arrays**: you can run any of these `add` operations on that type of array:
  - If you `add` an array of values, the PATCH operation appends it to the existing list of values.
  - If you `add` a single value, specify an ordinal element in the target array, or use the `{-}` special index to add that value to the end of the list.
- **Set semantic arrays**: The list of values included in a patch are merged with the existing set of values. Any duplicates within the array are removed.

As an example, start with the following list semantic array resource:

```
{
    "fruits" : [ "orange", "apple" ]
}
```

The following add operation includes the pineapple to the end of the list of fruits, as indicated by the `-` at the end of the `fruits` array.

```
{
    "operation" : "add",
    "field" : "/fruits/-",
    "value" : "pineapple"
}
```

The following is the resulting resource:

```
{
    "fruits" : [ "orange", "apple", "pineapple" ]
}
```

2.3.10.2. Patch Operation: Copy

The copy operation takes one or more existing values from the source field. It then adds those same values on the target field. Once the values are known, it is equivalent to performing an `add` operation on the target field.

The following `copy` operation takes the value from a field named `mail`, and then runs a `replace` operation on the target field, `another_mail`.

```
[
  {
     "operation":"copy",
     "from":"mail",
     "field":"another_mail"
  }
]
```

If the source field value and the target field value are configured as arrays, the result depends on whether the array has list semantics or set semantics, as described in Section 2.3.10.1, "Patch Operation: Add".

2.3.10.3. Patch Operation: Increment

The `increment` operation changes the value or values of the target field by the amount you specify. The value that you include must be one number, and may be positive or negative. The value of the target field must accept numbers. The following `increment` operation adds `1000` to the target value of `/user/payment`.

```
[
  {
    "operation" : "increment",
    "field" : "/user/payment",
    "value" : "1000"
  }
]
```

Since the `value` of the `increment` is a single number, arrays do not apply.

2.3.10.4. Patch Operation: Move

The move operation removes existing values on the source field. It then adds those same values on the target field. It is equivalent to performing a `remove` operation on the source, followed by an `add` operation with the same values, on the target.

The following `move` operation is equivalent to a `remove` operation on the source field, `surname`, followed by a `replace` operation on the target field value, `lastName`. If the target field does not exist, it is created.

```
[
  {
     "operation":"move",
     "from":"surname",
     "field":"lastName"
  }
]
```

To apply a `move` operation on an array, you need a compatible single-value, list semantic array, or set semantic array on both the source and the target. For details, see the criteria described in Section 2.3.10.1, "Patch Operation: Add".

2.3.10.5. Patch Operation: Remove

The `remove` operation ensures that the target field no longer contains the value provided. If the remove operation does not include a value, the operation removes the field. The following `remove` deletes the value of the `phoneNumber`, along with the field.

```
[
  {
    "operation" : "remove",
    "field" : "phoneNumber"
  }
]
```

If the object has more than one `phoneNumber`, those values are stored as an array.

A `remove` operation has different results on two standard types of arrays:

- **List semantic arrays**: A `remove` operation deletes the specified element in the array. For example, the following operation removes the first phone number, based on its array index (zero-based):

```
[
   {
      "operation" : "remove",
      "field" : "/phoneNumber/0"
   }
]
```

- **Set semantic arrays**: The list of values included in a patch are removed from the existing array.

2.3.10.6. Patch Operation: Replace

The `replace` operation removes any existing value(s) of the targeted field, and replaces them with the provided value(s). It is essentially equivalent to a `remove` followed by a `add` operation. If the arrays are used, the criteria is based on Section 2.3.10.1, "Patch Operation: Add". However, indexed updates are not allowed, even when the target is an array.

The following `replace` operation removes the existing `telephoneNumber` value for the user, and then adds the new value of `+1 408 555 9999`.

```
[
  {
    "operation" : "replace",
    "field" : "/telephoneNumber",
    "value" : "+1 408 555 9999"
  }
]
```

A PATCH replace operation on a list semantic array works in the same fashion as a PATCH remove operation. The following example demonstrates how the effect of both operations. Start with the following resource:

```
{
    "fruits" : [ "apple", "orange", "kiwi", "lime" ],
}
```

Apply the following operations on that resource:

```
[
  {
    "operation" : "remove",
    "field" : "/fruits/0",
    "value" : ""
  },
  {
    "operation" : "replace",
    "field" : "/fruits/1",
    "value" : "pineapple"
  }
]
```

The PATCH operations are applied sequentially. The `remove` operation removes the first member of that resource, based on its array index, (`fruits/0`), with the following result:

```
[
  {
    "fruits" : [ "orange", "kiwi", "lime" ],
  }
]
```

The second PATCH operation, a `replace`, is applied on the second member (`fruits/1`) of the intermediate resource, with the following result:

```
[
  {
    "fruits" : [ "orange", "pineapple", "lime" ],
  }
]
```

2.3.10.7. Patch Operation: Transform

The `transform` operation changes the value of a field based on a script or some other data transformation command. The following `transform` operation takes the value from the field named `/objects`, and applies the `something.js` script as shown:

```
[
  {
    "operation" : "transform",
    "field" : "/objects",
    "value" : {
      "script" : {
        "type" : "text/javascript",
        "file" : "something.js"
      }
    }
  }
]
```

2.3.10.8. Patch Operation Limitations

Some HTTP client libraries do not support the HTTP PATCH operation. Make sure that the library you use supports HTTP PATCH before using this REST operation.

For example, the Java Development Kit HTTP client does not support PATCH as a valid HTTP method. Instead, the method `HttpURLConnection.setRequestMethod("PATCH")` throws `ProtocolException`.

Parameters

You can use the following parameters. Other parameters might depend on the specific action implementation:

`_prettyPrint=true`

Format the body of the response.

`_fields=field[,field...]`

Return only the specified fields in the body of the response.

The `field` values are JSON pointers. For example if the resource is `{"parent":{"child":"value"}}`, `parent/child` refers to the `"child":"value"`.

2.3.11. Action

Actions are a means of extending Common REST APIs and are defined by the resource provider, so the actions you can use depend on the implementation.

The standard action indicated by `_action=create` is described in Section 2.3.6, "Create".

Parameters

You can use the following parameters. Other parameters might depend on the specific action implementation:

`_prettyPrint=true`

Format the body of the response.

`_fields=field[,field...]`

Return only the specified fields in the body of the response.

The `field` values are JSON pointers. For example if the resource is `{"parent":{"child":"value"}}`, `parent/child` refers to the `"child":"value"`.

2.3.12. Query

To query a resource collection (or resource container if you prefer to think of it that way), perform an HTTP GET and accept a JSON response, including at least a `_queryExpression`, `_queryFilter`, or `_queryId` parameter. These parameters cannot be used together:

```
GET /users?_queryFilter=true HTTP/1.1
Host: example.com
Accept: application/json
```

The server returns the result as a JSON object including a "results" array and other fields related to the query string parameters that you specify.

Parameters

You can use the following parameters:

`_queryFilter=`*`filter-expression`*

Query filters request that the server return entries that match the filter expression. You must URL-escape the filter expression.

The string representation is summarized as follows. Continue reading for additional explanation:

```
Expr           = OrExpr
OrExpr         = AndExpr ( 'or' AndExpr ) *
AndExpr        = NotExpr ( 'and' NotExpr ) *
NotExpr        = '!' PrimaryExpr | PrimaryExpr
PrimaryExpr    = '(' Expr ')' | ComparisonExpr | PresenceExpr | LiteralExpr
ComparisonExpr = Pointer OpName JsonValue
PresenceExpr   = Pointer 'pr'
LiteralExpr    = 'true' | 'false'
Pointer        = JSON pointer
OpName         = 'eq' |  # equal to
                 'co' |  # contains
                 'sw' |  # starts with
                 'lt' |  # less than
                 'le' |  # less than or equal to
                 'gt' |  # greater than
                 'ge' |  # greater than or equal to
                 STRING  # extended operator
JsonValue      = NUMBER | BOOLEAN | '"' UTF8STRING '"'
STRING         = ASCII string not containing white-space
UTF8STRING     = UTF-8 string possibly containing white-space
```

`JsonValue` components of filter expressions follow RFC 7159: *The JavaScript Object Notation (JSON) Data Interchange Format*. In particular, as described in section 7 of the RFC, the escape character in strings is the backslash character. For example, to match the identifier `test\`, use `_id eq 'test\\'`. In the JSON resource, the `\` is escaped the same way: `"_id":"test\\"`.

When using a query filter in a URL, be aware that the filter expression is part of a query string parameter. A query string parameter must be URL encoded as described in RFC 3986: *Uniform Resource Identifier (URI): Generic Syntax* For example, white space, double quotes (`"`), parentheses, and exclamation characters need URL encoding in HTTP query strings. The following rules apply to URL query components:

```
query       = *( pchar / "/" / "?" )
pchar       = unreserved / pct-encoded / sub-delims / ":" / "@"
unreserved  = ALPHA / DIGIT / "-" / "." / "_" / "~"
pct-encoded = "%" HEXDIG HEXDIG
sub-delims  = "!" / "$" / "&" / "'" / "(" / ")"
                  / "*" / "+" / "," / ";" / "="
```

`ALPHA`, `DIGIT`, and `HEXDIG` are core rules of RFC 5234: *Augmented BNF for Syntax Specifications*:

```
ALPHA       =  %x41-5A / %x61-7A   ; A-Z / a-z
DIGIT       =  %x30-39             ; 0-9
HEXDIG      =  DIGIT / "A" / "B" / "C" / "D" / "E" / "F"
```

As a result, a backslash escape character in a *JsonValue* component is percent-encoded in the URL query string parameter as `%5C`. To encode the query filter expression `_id eq 'test\\'`, use `_id+eq+'test%5C%5C'`, for example.

A simple filter expression can represent a comparison, presence, or a literal value.

For comparison expressions use *json-pointer comparator json-value*, where the *comparator* is one of the following:

`eq` (equals)
`co` (contains)
`sw` (starts with)
`lt` (less than)
`le` (less than or equal to)
`gt` (greater than)
`ge` (greater than or equal to)

For presence, use *json-pointer pr* to match resources where the JSON pointer is present.

Literal values include true (match anything) and false (match nothing).

Complex expressions employ `and`, `or`, and `!` (not), with parentheses, `(expression)`, to group expressions.

`_queryId=identifier`

Specify a query by its identifier.

Specific queries can take their own query string parameter arguments, which depend on the implementation.

`_pagedResultsCookie=string`

The string is an opaque cookie used by the server to keep track of the position in the search results. The server returns the cookie in the JSON response as the value of `pagedResultsCookie`.

In the request `_pageSize` must also be set and non-zero. You receive the cookie value from the provider on the first request, and then supply the cookie value in subsequent requests until the server returns a `null` cookie, meaning that the final page of results has been returned.

The `_pagedResultsCookie` parameter is supported when used with the `_queryFilter` parameter. The `_pagedResultsCookie` parameter is not guaranteed to work when used with the `_queryExpression` and `_queryId` parameters.

The `_pagedResultsCookie` and `_pagedResultsOffset` parameters are mutually exclusive, and not to be used together.

`_pagedResultsOffset=integer`

When `_pageSize` is non-zero, use this as an index in the result set indicating the first page to return.

The `_pagedResultsCookie` and `_pagedResultsOffset` parameters are mutually exclusive, and not to be used together.

`_pageSize=`*`integer`*

Return query results in pages of this size. After the initial request, use `_pagedResultsCookie` or `_pageResultsOffset` to page through the results.

`_totalPagedResultsPolicy=`*`string`*

When a `_pageSize` is specified, and non-zero, the server calculates the "totalPagedResults", in accordance with the `totalPagedResultsPolicy`, and provides the value as part of the response. The "totalPagedResults" is either an estimate of the total number of paged results (`_totalPagedResultsPolicy=ESTIMATE`), or the exact total result count (`_totalPagedResultsPolicy=EXACT`). If no count policy is specified in the query, or if `_totalPagedResultsPolicy=NONE`, result counting is disabled, and the server returns value of -1 for "totalPagedResults".

`_sortKeys=[+-]`*`field`*`[,[+-]`*`field`*`...]`

Sort the resources returned based on the specified field(s), either in `+` (ascending, default) order, or in `-` (descending) order.

The `_sortKeys` parameter is not supported for predefined queries (`_queryId`).

`_prettyPrint=true`

Format the body of the response.

`_fields=`*`field`*`[,`*`field`*`...]`

Return only the specified fields in each element of the "results" array in the response.

The `field` values are JSON pointers. For example if the resource is `{"parent":{"child":"value"}}`, `parent/child` refers to the `"child":"value"`.

2.3.13. HTTP Status Codes

When working with a Common REST API over HTTP, client applications should expect at least the following HTTP status codes. Not all servers necessarily return all status codes identified here:

200 OK

The request was successful and a resource returned, depending on the request.

201 Created

The request succeeded and the resource was created.

204 No Content

The action request succeeded, and there was no content to return.

304 Not Modified

The read request included an `If-None-Match` header, and the value of the header matched the revision value of the resource.

400 Bad Request

The request was malformed.

401 Unauthorized

The request requires user authentication.

403 Forbidden

Access was forbidden during an operation on a resource.

404 Not Found

The specified resource could not be found, perhaps because it does not exist.

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, such as a resource or protocol version that is not available.

409 Conflict

The request would have resulted in a conflict with the current state of the resource.

410 Gone

The requested resource is no longer available, and will not become available again. This can happen when resources expire for example.

412 Precondition Failed

The resource's current version does not match the version provided.

415 Unsupported Media Type

The request is in a format not supported by the requested resource for the requested method.

428 Precondition Required

The resource requires a version, but no version was supplied in the request.

500 Internal Server Error

The server encountered an unexpected condition that prevented it from fulfilling the request.

501 Not Implemented

The resource does not support the functionality required to fulfill the request.

503 Service Unavailable

The requested resource was temporarily unavailable. The service may have been disabled, for example.

2.4. 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 releases. The version number of a feature increases when AM introduces a non-backwards-compatible change that affects clients making use of the feature.

AM 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.

2.4.1. Supported REST API Versions

The REST API version numbers supported in AM 5.1 are as follows:

Supported protocol versions

The *protocol* versions supported in AM 5.1 are:

`1.0`

Supported resource versions

The *resource* versions supported in AM 5.1 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

Base	End Point	Supported Versions
	/realms	1.0
	/dashboard	1.0
	/sessions	1.1
	/serverinfo/*	1.1
	/users/*{user}*/devices/trusted	1.0
	/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 *AM Release Notes* section, Chapter 4, "*Changes and Deprecated Functionality*" in the *Release Notes* describes the differences between API versions.

2.4.2. 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/realms/root/authenticate
```

You can configure the default behavior AM will take when a REST call does not specify explicit version information. For more information, see Section 2.4.3, "Configuring the Default REST API Version for a Deployment".

2.4.3. Configuring the Default REST API Version for a Deployment

You can configure the default behavior AM will take when a REST call does not specify explicit version information using either of the following procedures:

- Procedure 2.3, "Configure Versioning Behavior by using the AM Console"
- Procedure 2.4, "Configure Versioning Behavior by using the ssoadm"

The available options for default behavior are as follows:

Latest

The latest available supported version of the API is used.

This is the preset default for new installations of AM.

Oldest

The oldest available supported version of the API is used.

This is the preset default for upgraded AM instances.

> **Note**
>
> The oldest supported version may not be the first that was released, as APIs versions become deprecated or unsupported. See Section 4.2, "Deprecated Functionality" in the *Release Notes*.

None

No version will be used. When a REST client application calls a REST API without specifying the version, AM returns an error and the request fails.

Procedure 2.3. Configure Versioning Behavior by using the AM Console

1. Log in as AM administrator, `amadmin`.
2. Click Configure > Global Services, and then click REST APIs.
3. In Default Version, select the required response to a REST API request that does not specify an explicit version: `Latest`, `Oldest`, or `None`.
4. Optionally, enable `Warning Header` to include warning messages in the headers of responses to requests.
5. Save your work.

Procedure 2.4. Configure Versioning Behavior by using the ssoadm

- Use the **ssoadm set-attr-defs** command with the `openam-rest-apis-default-version` attribute set to either `LATEST`, `OLDEST` or `NONE`, as in the following example:

```
$ ssh openam.example.com
$ cd /path/to/openam-tools/admin/openam/bin
$ ./ssoadm \
 set-attr-defs \
 --adminid amadmin \
 --password-file /tmp/pwd.txt \
 --servicename RestApisService \
 --schematype Global \
 --attributevalues openam-rest-apis-default-version=NONE

Schema attribute defaults were set.
```

2.4.4. REST API Versioning Messages

AM provides REST API version messages in the JSON response to a REST API call. You can also configure AM to return version messages in the response headers.

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/realms/root/authenticate

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

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

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

```
$ curl \
 --header "Content-Type: application/json" \
 --header "Accept-API-Version: protocol=1.0" \
 https://openam.example.com:8443/openam/json/realms/root/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 AM, 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/realms/root/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 Section 2.4.2, "Specifying an Explicit REST API Version".

2.5. Specifying Realms in REST API Calls

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

Realms can be specified in the following ways when making a REST API call to AM:

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 AM 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, specify the realm in the path component of the endpoint. You must specify the entire hierarchy of the realm, starting at the top-level realm. Prefix each realm in the hierarchy with the `realms/` keyword. For example `/realms/root/realms/customers/realms/europe`.

To authenticate a user in the top-level realm, use the `root` keyword. For example:

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

To authenticate a user in a subrealm named `customers` within the top-level realm, the REST endpoint would be:

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

If realms are specified using both the DNS alias and path methods, the path is used to determine the realm.

For example, the following REST endpoint returns users in a subrealm of the top-level realm named `europe`, not the realm with DNS alias `suppliers.example.com`:

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

2.6. 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 AM SSO token.
- If stateless sessions are enabled, the `tokenId` is an AM SSO token that includes an encoded AM 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 1.8.1.6, "Session Cookies" in the *Authentication and Single Sign-On 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/realms/root/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 `dëmø`, 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/realms/root/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, AM 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 AM. The default is still user name/password authentication:

```
$ curl \
 --request POST \
```

```
 https://openam.example.com:8443/openam/json/realms/root/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 AM, and so must also be sent back with the requests.

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

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

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

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

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

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

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

AM 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 set per realm; see Section 11.1.7, "Post Authentication Processing" in the *Authentication and Single Sign-On Guide* for details. Alternatively, failure URLs can be configured per authentication chain, which your client can specify using the `service` parameter described below. On failure AM 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/realms/root/authenticate
{
  "code":401,
  "reason":"Unauthorized",
  "message":"Invalid Password!!",
  "failureUrl": "http://www.example.com/401.html"
}
```

When making a REST API call, specify the realm in the path component of the endpoint. You must specify the entire hierarchy of the realm, starting at the top-level realm. Prefix each realm in the hierarchy with the `realms/` keyword. For example `/realms/root/realms/customers/realms/europe`.

For example, to authenticate to a subrealm `customers` within the top-level realm, then the authentication endpoint URL is as follows: `https://openam.example.com:8443/openam/json/realms/root/realms/customers/authenticate`

The following additional parameters are supported:

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 AM policy.

role

Set the value to an AM role.

service

Set the value to the name of an authentication chain.

user

Set the value to an AM user ID.

You can use the query string parameter, `sessionUpgradeSSOTokenId=tokenId`, to request session upgrade. Before the *`tokenId`* is searched for in the query string for session upgrade, the token is grabbed from the cookie. For an explanation of session upgrade, see Section 1.8.2, "Session Upgrade" in the *Authentication and Single Sign-On Guide*.

AM 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.

2.6.1. Logout

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

```
$ curl \
 --request POST \
 --header "Content-Type: application/json" \
 --header "Cache-Control: no-cache" \
 --header "iplanetDirectoryPro: AQIC5wM2...U3MTE4NA..*" \
https://openam.example.com:8443/openam/json/realms/root/sessions/?_action=logout

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

2.6.2. logoutByHandle

To log out a session using a session handle, first perform an HTTP GET to the resource URL, `/json/sessions/`, using the `queryFilter` action to get the session handle:

```
$ curl \
 --request GET \
 --header "Content-Type: application/json" \
 --header "Cache-Control: no-cache" \
 --header "iPlanetDirectoryPro: AQICS...NzEz*" \
http://openam.example.com:8080/openam/json/realms/root/sessions?_queryFilter=username%20eq%20%22demo
%22%20and%20realm%20eq%20%22%2F%22
{
  "result": [
   {
    "username": "demo",
    "universalId": "id=demo,ou=user,dc=openam,dc=forgerock,dc=org",
    "realm": "\/",
    "sessionHandle": "shandle:AQIC5w...MTY3*",
    "latestAccessTime": "2016-11-09T14:14:11Z",
    "maxIdleExpirationTime": "2016-11-09T14:44:11Z",
    "maxSessionExpirationTime": "2016-11-09T16:14:11Z"
   }
  ],
   "resultCount": 1,
   "pagedResultsCookie": null,
   "totalPagedResultsPolicy": "NONE",
   "totalPagedResults": -1,
   "remainingPagedResults": -1
}
```

To log out a session using a session handle, perform an HTTP POST to the resource URL, `/json/sessions/`, using the `logoutByHandle` action.

```
$ curl \
 --request POST \
 --header "Content-Type: application/json" \
 --header "Cache-Control: no-cache" \
 --header "iplanetDirectoryPro: AQIC5w...NTcy*" \
 --data '{"sessionHandles": ["shandle:AQIC5w...MTY3*","shandle:AQIC5w...NDcx*"]}' \
http://openam.example.com:8080/openam/json/realms/root/sessions/?_action=logoutByHandle
{
  "result": {
    "shandle:AQIC5w...NDcx*": true,
    "shandle:AQIC5w...MTY3*": true
  }
}
```

2.6.3. Load Balancer and Proxy Layer Requirements

When authentication depends on the client IP address and AM 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 AM to consume and forward the header as necessary. For details, see Section 2.2.4, "Handling HTTP Request Headers" in the *Installation Guide*.

2.6.4. 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 `ZGVtbzpjaGFuZ2VpdA==`:

```
$ curl \
--request POST
 \
--header "Content-Type: application/json"
 \
--header "X-OpenAM-Username: demo"
 \
--header "X-OpenAM-Password: changeit"
 \
--header "Authorization: Basic ZGVtbzpjaGFuZ2VpdA=="
 \
--data "{}" \
https://openam.example.com:8443/openam/json/realms/root/authenticate

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

2.7. Using the Session Token After Authentication

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

- First, call the `/json/authenticate` endpoint to log a user in to AM. 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/realms/root/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 AM 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 AM 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 AM authentication attempt into the HTTP header:

  ```
  $ curl \
  --request POST
   \
  --header "Content-Type: application/json"
   \
  --header "iPlanetDirectoryPro: AQIC5w...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 AM session cookie—by default, `iPlanetDirectoryPro`. You can find the cookie name in the AM console by navigating to Deployment > Servers > *Server Name* > Security > Cookie, in the Cookie Name field of the AM 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 AM functionality using the REST API. For example, users who lack administrative privileges cannot create AM realms. For more information on the AM privilege model, see Section 2.4.1, "Delegating Realm Administration Privileges" in the *Setup and Maintenance Guide*.

- Finally, call the REST API to log the user out of AM as described in Section 2.6, "Authentication and Logout". As with other REST API calls made after a user has authenticated, the REST API call to log out of AM requires the user's `tokenID` in the HTTP header.

2.8. Server Information

You can retrieve AM 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,
        "refererWhitelist": [
            ""
        ],
        "allowedWithoutReferer": true
    },
    "realm": "/",
    "xuiUserSessionValidationEnabled": true,
    "FQDN": "openam.example.com"
}
```

2.9. Token Encoding

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

```
AQIC5wM2LY4SfczntBbXvEAOuECbqMY3J4NW3byH6xwgkGE=@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:

```
AQIC5wM2LY4SfcztntBbXvEAOuECbqMY3J4NW3byH6xwgkGE.*AAJTSQACMDE.*
```

2.10. Logging

AM 5.1 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 AM versions prior to OpenAM 13. The legacy Logging Service is deprecated.

Both audit facilities log AM REST API calls.

2.10.1. Common Audit Logging of REST API Calls

AM logs information about all REST API calls to the `access` topic. For more information about AM audit topics, see Section 6.1.2, "Audit Log Topics" in the *Setup and Maintenance Guide*.

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

2.10.2. Legacy Logging of REST API Calls

AM 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
```

AM 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.

2.11. REST Goto URL Validation

You can set valid goto URLs using the AM console by following the instructions in Section 8.1.3, "Constraining Post-Login Redirects" in the *Authentication and Single Sign-On 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/"}
```

2.12. Reference

This reference section covers return codes and system settings relating to REST API support in AM.

2.12.1. REST APIs

ssoadm service name: `rest`

The following settings are available in this service:

Default Resource Version

The API resource version to use when the REST request does not specify an explicit version. Choose from:

- `Latest`. If an explicit version is not specified, the latest resource version of an API is used.
- `Oldest`. If an explicit version is not specified, the oldest supported resource version of an API is used. Note that since APIs may be deprecated and fall out of support, the oldest *supported* version may not be the first version.
- `None`. If an explicit version is not specified, the request will not be handled and an error status is returned.

The possible values for this property are:

```
Latest
Oldest
None
```

Default value: `Latest`

ssoadm attribute: `defaultVersion`

Warning Header

Whether to include a warning header in the response to a request which fails to include the `Accept-API-Version` header.

Default value: `false`

ssoadm attribute: `warningHeader`

API Descriptions

Whether API Explorer and API Docs are enabled in OpenAM and how the documentation for them is generated. Dynamic generation includes descriptions from any custom services and authentication modules you may have added. Static generation only includes services and authentication modules that were present when OpenAM was built. Note that dynamic documentation generation may not work in some application containers.

The possible values for this property are:

```
DYNAMIC
STATIC
DISABLED
```

Default value: `STATIC`

ssoadm attribute: `descriptionsState`

Default Protocol Version

The API protocol version to use when a REST request does not specify an explicit version. Choose from:

- `Oldest`. If an explicit version is not specified, the oldest protocol version is used.
- `Latest`. If an explicit version is not specified, the latest protocol version is used.
- `None`. If an explicit version is not specified, the request will not be handled and an error status is returned.

The possible values for this property are:

```
Oldest
Latest
None
```

Default value: `Latest`

ssoadm attribute: `defaultProtocolVersion`

Chapter 3

Developing with the Java SDK

This chapter introduces the Java SDK. the Java SDK is delivered with the full version of AM, AM-5.1.0.zip.

3.1. Installing Client SDK Samples

The full AM download, AM-5.1.0.zip, contains the Java Client SDK library, ClientSDK-13.5.0-1.jar, as well as samples for use on the command line in ExampleClientSDK-CLI-13.5.0-1.zip, and samples in a web application, ExampleClientSDK-WAR-13.5.0-1.war. The *ForgeRock Access Management Java SDK API Specification* provides a reference to the public APIs.

Procedure 3.1. To Deploy the Sample Web Application

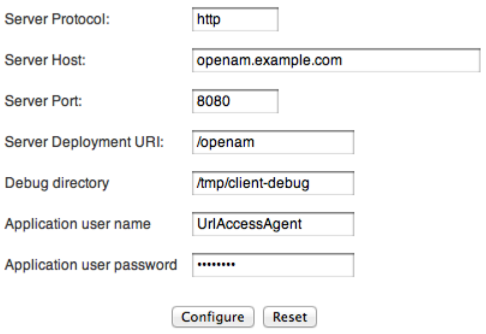
The sample web application deploys in your container to show you the client SDK samples in action.

1. Deploy the .war in your Java web application container such as Apache Tomcat or JBoss.

```
$ cp ExampleClientSDK-WAR-13.5.0-1.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 AM using the application user name, UrlAccessAgent, and password configured when you set up AM.



Server Protocol:	http
Server Host:	openam.example.com
Server Port:	8080
Server Deployment URI:	/openam
Debug directory	/tmp/client-debug
Application user name	UrlAccessAgent
Application user password	••••••••

Use the following hints to complete the configuration.

Server Protocol

Protocol to access AM (`http` or `https`)

Server Host

Fully qualified domain name for AM, such as `openam.example.com`

Server Port

AM port number such as 8080 or 8443

Server Deployment URI

URI entry point to AM such as `/openam`

Debug directory

Where to write the debug messages for the client samples

Application user name

An user agent configured to access AM, such as `UrlAccessAgent` set up when AM 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 AM as the AM Administrator, `amadmin`.

      This signs you into AM, storing the cookie in your browser.

   b. On the Samples tab page, click the link under Single Sign On Token Verification Servlet.

      If the sample web application is properly configured, you should see something like the following text in your browser.

```
SSOToken host name: 127.0.0.1
SSOToken Principal name: id=amadmin,ou=user,dc=openam,dc=forgerock,dc=org
Authentication type used: DataStore
IPAddress of the host: 127.0.0.1
SSO Token validation test succeeded
The token id is AQIC5...CMDEAAlNLABQtODY0Mjc5MDUwNDQzOTA2MzYxNg..*
...
User Attributes: {... givenName=[amAdmin], ...roles=[Top-level Admin Role], ...}
```

Procedure 3.2. 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-13.5.0-1.zip
```

2. Configure the samples to access AM.

```
$ 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!!
```

3.2. About the 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`: AM 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 AM 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 AM 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
```

3.3. Authenticating Using Java SDK

This section looks at authentication with the AM Java SDK and at the sample client, `Login.java`, which demonstrates authenticating to AM 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 3.1, "Installing Client SDK Samples" are installed.

With AM, 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 AM's Authentication Service.

4. After handling all authentication callbacks, your application calls the `AuthContext` `getStatus()` method.

   On login success, AM 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 AM 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 *ForgeRock Access Management Public API JavaDoc* for reference.

As the sample client gets the realm (called organization in the sample), locale, and authentication module to set up the authentication context, there is not need for a language callback to get the local afterwards. The `Login.java` example does, however, show simple ways of handling callbacks for the command-line context. The implementation of the sample client follows.

```
package com.sun.identity.samples.authentication;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.IOException;
import javax.security.auth.callback.Callback;
import javax.security.auth.callback.ChoiceCallback;
import javax.security.auth.callback.NameCallback;
import javax.security.auth.callback.PasswordCallback;
import javax.security.auth.callback.TextInputCallback;
import javax.security.auth.callback.TextOutputCallback;
import javax.security.auth.callback.UnsupportedCallbackException;
import com.sun.identity.authentication.AuthContext;
import com.sun.identity.authentication.spi.AuthLoginException;
import com.sun.identity.shared.debug.Debug;

public class Login {
    private String loginIndexName;
    private String orgName;
    private String locale;

    private Login(String loginIndexName, String orgName) {
        this.loginIndexName = loginIndexName;
        this.orgName = orgName;
    }

    private Login(String loginIndexName, String orgName, String locale) {
        this.loginIndexName = loginIndexName;
        this.orgName = orgName;
        this.locale = locale;
    }

    protected AuthContext getAuthContext()
        throws AuthLoginException {
        AuthContext lc = new AuthContext(orgName);
        AuthContext.IndexType indexType = AuthContext.IndexType.MODULE_INSTANCE;
        if (locale == null || locale.length() == 0) {
            lc.login(indexType, loginIndexName);
        } else {
            lc.login(indexType, loginIndexName, locale);
        }
        debugMessage(loginIndexName + ": Obtained login context");
        return lc;
    }

    private void addLoginCallbackMessage(Callback[] callbacks)
    throws UnsupportedCallbackException {
        int i = 0;
        try {
            for (i = 0; i < callbacks.length; i++) {
                if (callbacks[i] instanceof TextOutputCallback) {
                    handleTextOutputCallback((TextOutputCallback)callbacks[i]);
                } else if (callbacks[i] instanceof NameCallback) {
                    handleNameCallback((NameCallback)callbacks[i]);
                } else if (callbacks[i] instanceof PasswordCallback) {
                    handlePasswordCallback((PasswordCallback)callbacks[i]);
                } else if (callbacks[i] instanceof TextInputCallback) {
                    handleTextInputCallback((TextInputCallback)callbacks[i]);
                } else if (callbacks[i] instanceof ChoiceCallback) {
                    handleChoiceCallback((ChoiceCallback)callbacks[i]);
                }
            }
        } catch (IOException e) {
            e.printStackTrace();
            throw new UnsupportedCallbackException(callbacks[i],e.getMessage());
        }
    }

    private void handleTextOutputCallback(TextOutputCallback toc) {
        debugMessage("Got TextOutputCallback");
        // display the message according to the specified type

        switch (toc.getMessageType()) {
            case TextOutputCallback.INFORMATION:
                debugMessage(toc.getMessage());
                break;
            case TextOutputCallback.ERROR:
                debugMessage("ERROR: " + toc.getMessage());
                break;
            case TextOutputCallback.WARNING:
                debugMessage("WARNING: " + toc.getMessage());
                break;
            default:
                debugMessage("Unsupported message type: " +
                    toc.getMessageType());
        }
    }
    private void handleNameCallback(NameCallback nc)
        throws IOException {
        // prompt the user for a username
        System.out.print(nc.getPrompt());
        System.out.flush();
        nc.setName((new BufferedReader
            (new InputStreamReader(System.in))).readLine());
    }

    private void handleTextInputCallback(TextInputCallback tic)
        throws IOException {
        // prompt for text input
        System.out.print(tic.getPrompt());
        System.out.flush();
        tic.setText((new BufferedReader
            (new InputStreamReader(System.in))).readLine());
    }

    private void handlePasswordCallback(PasswordCallback pc)
        throws IOException {
        // prompt the user for sensitive information
        System.out.print(pc.getPrompt());
        System.out.flush();
        String passwd = (new BufferedReader(new InputStreamReader(System.in))).
            readLine();
        pc.setPassword(passwd.toCharArray());
    }

    private void handleChoiceCallback(ChoiceCallback cc)
        throws IOException {
        // ignore the provided defaultValue
        System.out.print(cc.getPrompt());

        String[] strChoices = cc.getChoices();
        for (int j = 0; j < strChoices.length; j++) {
            System.out.print("choice[" + j + "] : " + strChoices[j]);
        }
        System.out.flush();
        cc.setSelectedIndex(Integer.parseInt((new BufferedReader
            (new InputStreamReader(System.in))).readLine()));
    }

    protected boolean login(AuthContext lc)
        throws UnsupportedCallbackException {
        boolean succeed = false;
        Callback[] callbacks = null;

        // get information requested from module
        while (lc.hasMoreRequirements()) {
            callbacks = lc.getRequirements();
            if (callbacks != null) {
                addLoginCallbackMessage(callbacks);
                lc.submitRequirements(callbacks);
            }
        }

        if (lc.getStatus() == AuthContext.Status.SUCCESS) {
            System.out.println("Login succeeded.");
            succeed = true;
        } else if (lc.getStatus() == AuthContext.Status.FAILED) {
            System.out.println("Login failed.");
        } else {
            System.out.println("Unknown status: " + lc.getStatus());
        }

        return succeed;
    }

    protected void logout(AuthContext lc)
        throws AuthLoginException {
        lc.logout();
        System.out.println("Logged Out!!");
    }

    static void debugMessage(String msg) {
        System.out.println(msg);
    }

    public static void main(String[] args) {
        try {
            System.out.print("Realm (e.g. /): ");
            String orgName = (new BufferedReader(
                new InputStreamReader(System.in))).readLine();

            System.out.print("Login module name (e.g. DataStore or LDAP): ");
            String moduleName = (new BufferedReader(
                new InputStreamReader(System.in))).readLine();

            System.out.print("Login locale (e.g. en_US or fr_FR): ");
            String locale = (new BufferedReader(
                new InputStreamReader(System.in))).readLine();

            Login login = new Login(moduleName, orgName, locale);
            AuthContext lc = login.getAuthContext();
            if (login.login(lc)) {
                login.logout(lc);
            }
        } catch (IOException e) {
            e.printStackTrace();
        } catch (AuthLoginException e) {
            e.printStackTrace();
        } catch (UnsupportedCallbackException e) {
            e.printStackTrace();
        }
        System.exit(0);
    }
}
```

3.3.1. Encoding Passwords and Password Reset Questions and Answers

AM uses symmetric encryption algorithms to encrypt and decrypt stored passwords, so that they can be retrieved or modified at later date if necessary. The AM 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 AM can decrypt the password.

```
am.encryption.pwd=@ENCRYPTION_KEY@
```

AM'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*.

3.4. Handling Single Sign-On Using the Java SDK

This section looks at handling session tokens with the AM Java SDK. The class shown in this section is `com.sun.identity.samples.sso.SSOTokenSample`.

When a user authenticates successfully, AM 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. AM 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 3.1, "Installing 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 AM 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. AM 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 AM, and then displays some information from the SSO token. The implementation of the sample client follows.

```
package com.sun.identity.samples.sso;

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.IOException;
import java.net.InetAddress;
import com.iplanet.sso.SSOException;
import com.iplanet.sso.SSOToken;
import com.iplanet.sso.SSOTokenID;
import com.iplanet.sso.SSOTokenManager;

public class SSOTokenSample {
    private SSOTokenManager manager;
    private SSOToken token;

    private SSOTokenSample(String tokenID)
        throws SSOException
    {
        if (validateToken(tokenID)) {
            setGetProperties(token);
        }
    }

    private boolean validateToken(String tokenID)
        throws SSOException
    {
        boolean validated = false;
        manager = SSOTokenManager.getInstance();
        token = manager.createSSOToken(tokenID);

        // isValid method returns true for valid token.
        if (manager.isValidToken(token)) {
                // let us get all the values from the token
            String host = token.getHostName();
            java.security.Principal principal = token.getPrincipal();
            String authType = token.getAuthType();
            int level = token.getAuthLevel();
            InetAddress ipAddress = token.getIPAddress();
            long maxTime = token.getMaxSessionTime();
            long idleTime = token.getIdleTime();
            long maxIdleTime = token.getMaxIdleTime();

            System.out.println("SSOToken host name: " + host);
            System.out.println("SSOToken Principal name: " +
                principal.getName());
            System.out.println("Authentication type used: " + authType);
            System.out.println("IPAddress of the host: " +
                ipAddress.getHostAddress());
            validated = true;
        }

        return validated;
    }

    private void setGetProperties(SSOToken token)
        throws SSOException
    {
        /*
         * Validate the token again, with another method
         * if token is invalid, this method throws an exception
         */
        manager.validateToken(token);
        System.out.println("SSO Token validation test Succeeded.");

        // Get the SSOTokenID associated with the token and print it.
        SSOTokenID id = token.getTokenID();
        String tokenId = id.toString();
        System.out.println("Token ID: " + tokenId);

        // Set and get properties in the token.
        token.setProperty("TimeZone", "PST");
        token.setProperty("County", "SantaClara");
        String tZone = token.getProperty("TimeZone");
        String county = token.getProperty("County");

        System.out.println("Property: TimeZone: " + tZone);
        System.out.println("Property: County: " + county);
    }

    public static void main(String[] args) {
        try {
            System.out.print("Enter SSOToken ID: ");
            String ssoTokenID = (new BufferedReader(
                new InputStreamReader(System.in))).readLine();
            new SSOTokenSample(ssoTokenID.trim());
        } catch (SSOException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
        System.exit(0);
    }

}
```

Before you run the script that calls the sample, authenticate to AM in order to have AM generate the SSO token ID. To see the SSO token ID, use the RESTful `authenticate` command as shown in the following example, or alternatively run the `SSOTokenSampleServlet` web-based sample.

```
$ curl \
 --request POST \
 --data "username=demo&password=changeit" \
 http://openam.example.com:8080/openam/identity/authenticate
token.id=AQIC5wM2LY4Sfcyy10grl...AlNLABQtNjI4OTkyNTUxNTc4MDQ3NzEzOQ..*
$ sh scripts/SSOTokenSample.sh
Enter SSOToken ID: AQIC5wM2LY4Sfcyy10grl...AlNLABQtNjI4OTkyNTUxNTc4MDQ3NzEzOQ..*
SSOToken host name: 172.16.203.239
SSOToken Principal name: id=demo,ou=user,dc=openam,dc=forgerock,dc=org
Authentication type used: DataStore
IPAddress of the host: 172.16.203.239
SSO Token validation test Succeeded.
Token ID: AQIC5wM2LY4Sfcyy10grl...AlNLABQtNjI4OTkyNTUxNTc4MDQ3NzEzOQ..*
Property: TimeZone: PST
Property: County: SantaClara
```

Notice both the properties populated by AM, and also the two properties, `TimeZone` and `County`, that are set by the sample client.

3.4.1. 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/notificationservice`.

com.iplanet.am.sdk.caching.enabled

Set this parameter to `true`.

com.iplanet.am.serverMode

Set this parameter to `false`.

com.sun.identity.client.notification.url

Set this parameter to `http://host:port/context/notificationservice`.

com.sun.identity.idm.cache.enabled

Set this parameter to `true`.

com.sun.identity.idm.remote.notification.enabled

Set this parameter to `true`.

com.sun.identity.sm.cache.enabled

Set this parameter to `true`.

com.sun.identity.sm.enableDataStoreNotification

Set this parameter to `true`.

The above configuration to access the notification service also applies for other types of listeners, such as `ServiceListener`, and `IdEventListener` implementations. See the *ForgeRock Access Management Java SDK API Specification* for details on the available listener interfaces.

3.5. Requesting Policy Decisions Using the Java SDK

This section shows how to request policy decision by using AM Java SDK. The chapter focuses on the sample client, `source/samples/policy/PolicyEvaluationSample.java`, which demonstrates making a request to AM for a policy decision about access to a web resource.

Before you continue, make sure that the packages described in Section 3.1, "Installing Client SDK Samples" are installed.

AM centralizes policy administration, policy evaluation, and policy decision making so that your applications do not have to do so. In many deployments, AM 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 AM, 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, AM 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 AM that corresponds to the resource in question. You can set up the policy in the AM 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 AM using the script, `scripts/run-policy-evaluation-sample.sh`.

```
$ sh scripts/run-policy-evaluation-sample.sh
Using properties file:policyEvaluationSample
sample properties:
user.password:changeit
service.name:iPlanetAMWebAgentService
user.name:demo
resource.name:http://www.example.com:80/banner.html
action
.name:GET
------------------------------------------------------------------------------:
Entering getSSOToken():userName=demo,password=changeit
TokenID:AQIC5wM2LY4Sfcx3aQGFRKu5-r1a-Vfyjb...5ODM4NDY0MzE0ODYzODQ1*
returning from getSSOToken()
Entering getPolicyDecision():resourceName=http://www.example.com:80/banner.html,
 serviceName=iPlanetAMWebAgentService,actionName=GET
policyDecision:<PolicyDecision>
<ResponseAttributes>
</ResponseAttributes>
<ActionDecision timeToLive="9223372036854775807">
<AttributeValuePair>
<Attribute name="GET"/>
<Value>allow</Value>
</AttributeValuePair>
<Advices>
</Advices>
</ActionDecision>
</PolicyDecision>

returning from getPolicyDecision()
```

As you see, the policy decision response is formatted here as an XML document.[1] Notice here the line showing that AM has allowed access to the resource.

```
<Value>allow</Value>
```

3.6. Requesting a XACML Policy Decision Using the Java SDK

This section shows how to request a XACML policy decision with AM Java SDK, using the sample client, `source/samples/xacml/XACMLClientSample.java`. The sample client relies on an AM server acting as a policy decision point and another AM server acting as a policy enforcement point.

Before you continue, make sure that the packages described in the Section 3.1, "Installing 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 AM. Most of the work in the sample is done setting up the request.

The implementation of the `XACMLClientSample` class follows.

[1] The `PolicyDecision` element is defined in *`openam`*`/WEB-INF/remoteInterface.dtd` where *`openam`* is the location where the AM web application is deployed.

```
package samples.xacml;

import com.sun.identity.saml2.common.SAML2Exception;
import com.sun.identity.xacml.client.XACMLRequestProcessor;
import com.sun.identity.xacml.common.XACMLConstants;
import com.sun.identity.xacml.common.XACMLException;
import com.sun.identity.xacml.context.ContextFactory;
import com.sun.identity.xacml.context.Action;
import com.sun.identity.xacml.context.Attribute;
import com.sun.identity.xacml.context.Environment;
import com.sun.identity.xacml.context.Request;
import com.sun.identity.xacml.context.Resource;
import com.sun.identity.xacml.context.Response;
import com.sun.identity.xacml.context.Subject;
import java.net.URI;
import java.net.URISyntaxException;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.Enumeration;
import java.util.List;
import java.util.MissingResourceException;
import java.util.Properties;
import java.util.ResourceBundle;

public class XACMLClientSample {

    public XACMLClientSample() {
    }

    public static void main(String[] args) throws Exception {
        XACMLClientSample clientSample = new XACMLClientSample();
        clientSample.runSample(args);
        System.exit(0);
    }

    public void runSample(String[] args) throws Exception {
        if (args.length == 0 || args.length > 1) {
            System.out.println("Missing argument:"
                    + "properties file name not specified");
        } else {
            System.out.println("Using properties file:" + args[0]);
            Properties sampleProperties = getProperties(args[0]);
            testProcessRequest(
                (String)sampleProperties.get("pdp.entityId"),
                (String)sampleProperties.get("pep.entityId"),
                (String)sampleProperties.get("subject.id"),
                (String)sampleProperties.get("subject.id.datatype"),
                (String)sampleProperties.get("subject.category"),
                (String)sampleProperties.get("resource.id"),
                (String)sampleProperties.get("resource.id.datatype"),
                (String)sampleProperties.get("resource.servicename"),
                (String)sampleProperties.get("resource.servicename.datatype"),
                (String)sampleProperties.get("action.id"),
                (String)sampleProperties.get("action.id.datatype")
            );
        }
    }
    private void testProcessRequest(
            String pdpEntityId, String pepEntityId,
            String subjectId, String subjectIdType,
            String subjectCategory,
            String resourceId, String resourceIdType,
            String serviceName, String serviceNameType,
            String actionId, String actionIdType)
            throws XACMLException, SAML2Exception,
            URISyntaxException, Exception {

        Request xacmlRequest = createSampleXacmlRequest(
            subjectId, subjectIdType,
            subjectCategory,
            resourceId, resourceIdType,
            serviceName, serviceNameType,
            actionId, actionIdType);

        System.out.println("\ntestProcessRequest():xacmlRequest:\n"
                + xacmlRequest.toXMLString(true, true));

        Response xacmlResponse = XACMLRequestProcessor.getInstance()
                .processRequest(xacmlRequest, pdpEntityId, pepEntityId);

        System.out.println("testProcessRequest():xacmlResponse:\n"
                + xacmlResponse.toXMLString(true, true));
    }

    private Request createSampleXacmlRequest(
            String subjectId, String subjectIdType,
            String subjectCategory,
            String resourceId, String resourceIdType,
            String serviceName, String serviceNameType,
            String actionId, String actionIdType)
            throws XACMLException, URISyntaxException {

        Request request = ContextFactory.getInstance().createRequest();

        //Subject
        Subject subject = ContextFactory.getInstance().createSubject();
        subject.setSubjectCategory(new URI(subjectCategory));

        //set subject id
        Attribute attribute = ContextFactory.getInstance().createAttribute();
        attribute.setAttributeId(new URI(XACMLConstants.SUBJECT_ID));
        attribute.setDataType(new URI(subjectIdType));
        List valueList = new ArrayList();
        valueList.add(subjectId);
        attribute.setAttributeStringValues(valueList);
        List attributeList = new ArrayList();
        attributeList.add(attribute);
        subject.setAttributes(attributeList);

        //set Subject in Request
        List subjectList = new ArrayList();
        subjectList.add(subject);
        request.setSubjects(subjectList);

        //Resource
        Resource resource = ContextFactory.getInstance().createResource();
        //set resource id
        attribute = ContextFactory.getInstance().createAttribute();
        attribute.setAttributeId(new URI(XACMLConstants.RESOURCE_ID));
        attribute.setDataType( new URI(resourceIdType));
        valueList = new ArrayList();
        valueList.add(resourceId);
        attribute.setAttributeStringValues(valueList);
        attributeList = new ArrayList();
        attributeList.add(attribute);

        //set serviceName
        attribute = ContextFactory.getInstance().createAttribute();
        attribute.setAttributeId(new URI(XACMLConstants.TARGET_SERVICE));
        attribute.setDataType(new URI(serviceNameType));
        valueList = new ArrayList();
        valueList.add(serviceName);
        attribute.setAttributeStringValues(valueList);
        attributeList.add(attribute);
        resource.setAttributes(attributeList);

        //set Resource in Request
        List resourceList = new ArrayList();
        resourceList.add(resource);
        request.setResources(resourceList);

        //Action
        Action action = ContextFactory.getInstance().createAction();
        attribute = ContextFactory.getInstance().createAttribute();
        attribute.setAttributeId(new URI(XACMLConstants.ACTION_ID));
        attribute.setDataType(new URI(actionIdType));

        //set actionId
        valueList = new ArrayList();
        valueList.add(actionId);
        attribute.setAttributeStringValues(valueList);
        attributeList = new ArrayList();
        attributeList.add(attribute);
        action.setAttributes(attributeList);

        //set Action in Request
        request.setAction(action);

        //Environment, our PDP does not use environment now
        Environment environment = ContextFactory.getInstance()
            .createEnvironment();
        request.setEnvironment(environment);
        return request;
    }

    private Properties getProperties(String file)
        throws MissingResourceException {
        Properties properties = new Properties();
        ResourceBundle bundle = ResourceBundle.getBundle(file);
        Enumeration e = bundle.getKeys();
        System.out.println("sample properties:");
        while (e.hasMoreElements()) {
            String key = (String) e.nextElement();
            String value = bundle.getString(key);
```

```
                properties.put(key, value);
                System.out.println(key + ":" + value);
            }
            return properties;
        }
}
```

Before running the sample client, you must set up the configuration as described in the comments at the outset of the `scripts/run-xacml-client-sample.sh` script.

- Check `resources/AMConfig.properties` to see which AM 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 AM server acting as the policy enforcement point, configured in a circle of trust with the AM server acting as the policy decision point. When you set up the sample clients, you pointed them to an AM server. For this example, configure that server to function as a policy enforcement point and also as a policy decision point.

1. In the AM 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 AM 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 by using the **ssoadm** command as shown below:

```
$ 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 Section 2.1, "Implementing Authorization Using the AM Console" in the *Authorization Guide* for details.

After you have configured AM and the properties files, run the sample client script, and observe the XACML request and response.

```

$ sh scripts/run-xacml-client-sample.sh

Using properties file:xacmlClientSample
sample properties:
subject.id.datatype:urn:oasis:names:tc:xacml:1.0:data-type:x500Name
pdp.entityId:xacmlPdpEntity
resource.servicename.datatype:http://www.w3.org/2001/XMLSchema#string
resource.id:http://www.example.com:80/banner.html
resource.servicename:iPlanetAMWebAgentService
action.id.datatype:http://www.w3.org/2001/XMLSchema#string
resource.id.datatype:http://www.w3.org/2001/XMLSchema#string
action.id:GET
subject.category:urn:oasis:names:tc:xacml:1.0:subject-category:access-subject
pep.entityId:xacmlPepEntity
subject.id:id=demo,ou=user,dc=openam,dc=forgerock,dc=org

testProcessRequest():xacmlRequest:

<xacml-context:Request
 xmlns:xacml-context="urn:oasis:names:tc:xacml:2.0:context:schema:os"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="urn:oasis:names:tc:xacml:2.0:context:schema:os
  http://docs.oasis-open.org/xacml/access_control-xacml-2.0-context-schema-os.xsd">
<Subject SubjectCategory=
 "urn:oasis:names:tc:xacml:1.0:subject-category:access-subject">
<Attribute
 AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
 DataType="urn:oasis:names:tc:xacml:1.0:data-type:x500Name" >
<AttributeValue
 >id=demo,ou=user,dc=openam,dc=forgerock,dc=org</AttributeValue>
</Attribute>
</Subject>
<xacml-context:Resource>
<Attribute
```

```
 AttributeId="ResourceId"
 DataType="http://www.w3.org/2001/XMLSchema#string" >
<AttributeValue>http://www.example.com:80/banner.html</AttributeValue>
</Attribute>
<Attribute
  AttributeId="urn:sun:names:xacml:2.0:resource:target-service"
  DataType="http://www.w3.org/2001/XMLSchema#string" >
<AttributeValue>iPlanetAMWebAgentService</AttributeValue>
</Attribute>
</xacml-context:Resource>
<xacml-context:Action>
<Attribute
 AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
 DataType="http://www.w3.org/2001/XMLSchema#string" >
<AttributeValue>GET</AttributeValue>
</Attribute>
</xacml-context:Action>
<xacml-context:Environment></xacml-context:Environment>
</xacml-context:Request>

testProcessRequest():xacmlResponse:
<xacml-context:Response
 xmlns:xacml-context="urn:oasis:names:tc:xacml:2.0:context:schema:os" >
<xacml-context:Result ResourceId="http://www.example.com:80/banner.html">
<xacml-context:Decision>Permit</xacml-context:Decision>
<xacml-context:Status>
<xacml-context:StatusCode
 Value="urn:oasis:names:tc:xacml:1.0:status:ok">
</xacml-context:StatusCode>
<xacml-context:StatusMessage>ok</xacml-context:StatusMessage>
<xacml-context:StatusDetail
 xmlns:xacml-context="urn:oasis:names:tc:xacml:2.0:context:schema:cd:04">
<xacml-context:StatusDetail/></xacml-context:StatusDetail>
</xacml-context:Status>
</xacml-context:Result>
</xacml-context:Response>
```

Chapter 4

Developing with the C SDK

This chapter introduces the C SDK. To obtain the C SDK or request support, contact info@forgerock.com.

To prepare to install C SDK, 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 4.1. To Build 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 AM 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 = AQIC5wM2LY4SfcxZfk4EzC9Y46P9cXG9ogwf2ixnYOeZ0K0.*AAJTSQACMDE.*
      Organization = /
      Module Instance Name [0] = SAE
      Module Instance Name [1] = LDAP
      Module Instance Name [2] = WSSAuthModule
      Module Instance Name [3] = Federation
      Module Instance Name [4] = HOTP
      Module Instance Name [5] = DataStore
   Logout 1 Succeeded!
```

Chapter 5
Developing with Scripts

You can use scripts for client-side and server-side authentication, policy conditions, and handling OpenID Connect claims.

5.1. The Scripting Environment

This section introduces how AM executes scripts, and covers thread pools and security configuration.

You can use scripts to modify default AM behavior in the following situations, also known as *contexts*:

Client-side Authentication

Scripts that are executed on the client during authentication. 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.

Policy Condition

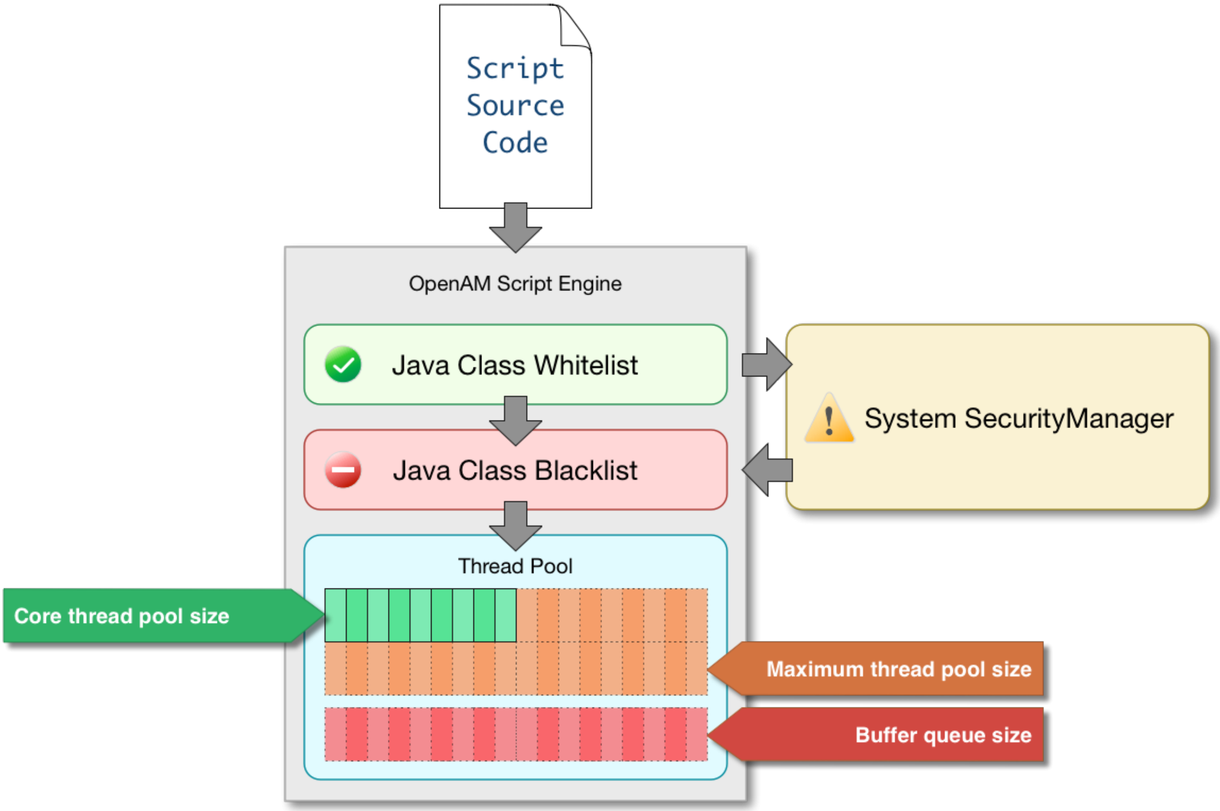
Scripts used as conditions within policies.

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.

AM implements a configurable scripting engine for each of the context types that are executed on the server.

The scripting engines in AM have two main components: security settings, and the thread pool.



5.1.1. Security

AM 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 6.1, "Scripting".

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, AM 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, AM terminates the thread, shrinking the pool.

For more information on configuring script engine thread pools, see Section 6.1, "Scripting".

5.2. Global Scripting 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. Accessing HTTP Services

AM passes an HTTP client object, `httpClient`, to server-side scripts. Server-side scripts can call HTTP services with the `httpClient.send` method. The method returns an `HttpClientResponse` object.

Configure the parameters for the HTTP client object by using the `org.forgerock.http.protocol` package. This package contains the `Request` class, which has methods for setting the URI and type of request.

The following example, taken from the default server-side Scripted authentication module script, uses these methods to call an online API to determine the longitude and latitude of a user based on their postal address:

```
function getLongitudeLatitudeFromUserPostalAddress() {

    var request = new org.forgerock.http.protocol.Request();

    request.setUri("http://maps.googleapis.com/maps/api/geocode/json?address=" +
 encodeURIComponent(userPostalAddress));
    request.setMethod("GET");

    var response = httpClient.send(request).get();
    logResponse(response);

    var geocode = JSON.parse(response.getEntity());
    var i;

    for (i = 0; i < geocode.results.length; i++) {
        var result = geocode.results[i];
        latitude = result.geometry.location.lat;
        longitude = result.geometry.location.lng;

        logger.message("latitude:" + latitude + " longitude:" + longitude);
    }
}
```

HTTP client requests are synchronous and blocking until they return. You can, however, set a global timeout for server-side scripts. For details, see Section 11.2.25, "Scripted Authentication Module Properties" in the *Authentication and Single Sign-On Guide*.

Server-side scripts can access response data by using the methods listed in the table below.

Table 5.1. HTTP Client Response Methods

Method	Parameters	Return Type	Description
`HttpClientResponse.getCookies`	`Void`	`Map<String, String>`	Get the cookies for the returned response, if any exist.
`HttpClientResponse.getEntity`	`Void`	`String`	Get the entity of the returned response.
`HttpClientResponse.getHeaders`	`Void`	`Map<String, String>`	Get the headers for the returned response, if any exist.
`HttpClientResponse.getReasonPhrase`	`Void`	`String`	Get the reason phrase of the returned response.
`HttpClientResponse.getStatusCode`	`Void`	`Integer`	Get the status code of the returned response.
`HttpClientResponse.hasCookies`	`Void`	`Boolean`	Indicate whether the returned response had any cookies.
`HttpClientResponse.hasHeaders`	`Void`	`Boolean`	Indicate whether the returned response had any headers.

5.2.2. Debug Logging

Server-side scripts can write messages to AM debug logs by using the `logger` object.

AM does not log debug messages from scripts by default. You can configure AM to log such messages by setting the debug log level for the `amScript` service. For details, see Section 9.2.3, "Debug Logging By Service" in the *Setup and Maintenance Guide*.

The following table lists the `logger` methods.

Table 5.2. Logger Methods

Method	Parameters	Return Type	Description
`logger.error`	*`Error Message`* (type: `String`)	`Void`	Write *`Error Message`* to AM debug logs if ERROR level logging is enabled.

Method	Parameters	Return Type	Description
`logger.errorEnabled`	`Void`	`Boolean`	Return `true` when ERROR level debug messages are enabled.
`logger.message`	*`Message`* (type: `String`)	`Void`	Write *`Message`* to AM debug logs if MESSAGE level logging is enabled.
`logger.messageEnabled`	`Void`	`Boolean`	Return `true` when MESSAGE level debug messages are enabled.
`logger.warning`	*`Warning Message`* (type: `String`)	`Void`	Write *`Warning Message`* to AM debug logs if WARNING level logging is enabled.
`logger.warningEnabled`	`Void`	`Boolean`	Return `true` when WARNING level debug messages are enabled.

5.3. Managing Scripts

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 using the AM console, the **ssoadm** command, and the REST API.

5.3.1. Managing Scripts With the AM Console

The following procedures describe how to create, modify, and delete scripts using the AM console:

- Procedure 5.1, "To Create Scripts by Using the AM Console"
- Procedure 5.2, "To Modify Scripts by Using the AM Console"
- Procedure 5.3, "To Delete Scripts by Using the AM Console"

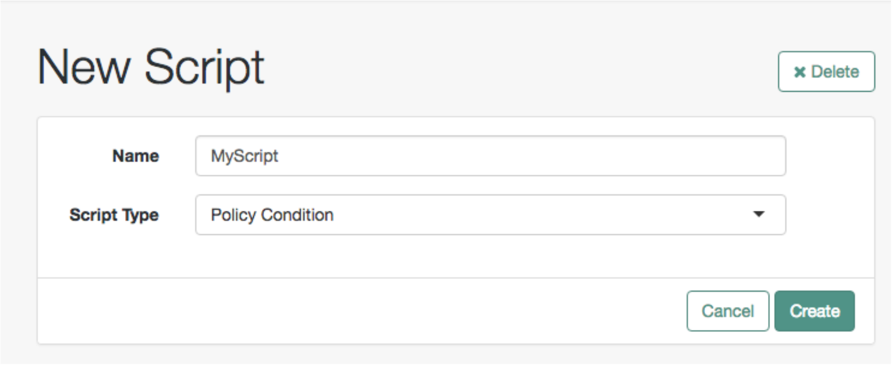
Procedure 5.1. To Create Scripts by Using the AM Console

1. Log in to the AM console as an AM administrator, for example, `amadmin`.

2. Navigate to Realms > *`Realm Name`* > Scripts.

3. Click New Script.

   The New Script page appears:



New Script

Name: MyScript

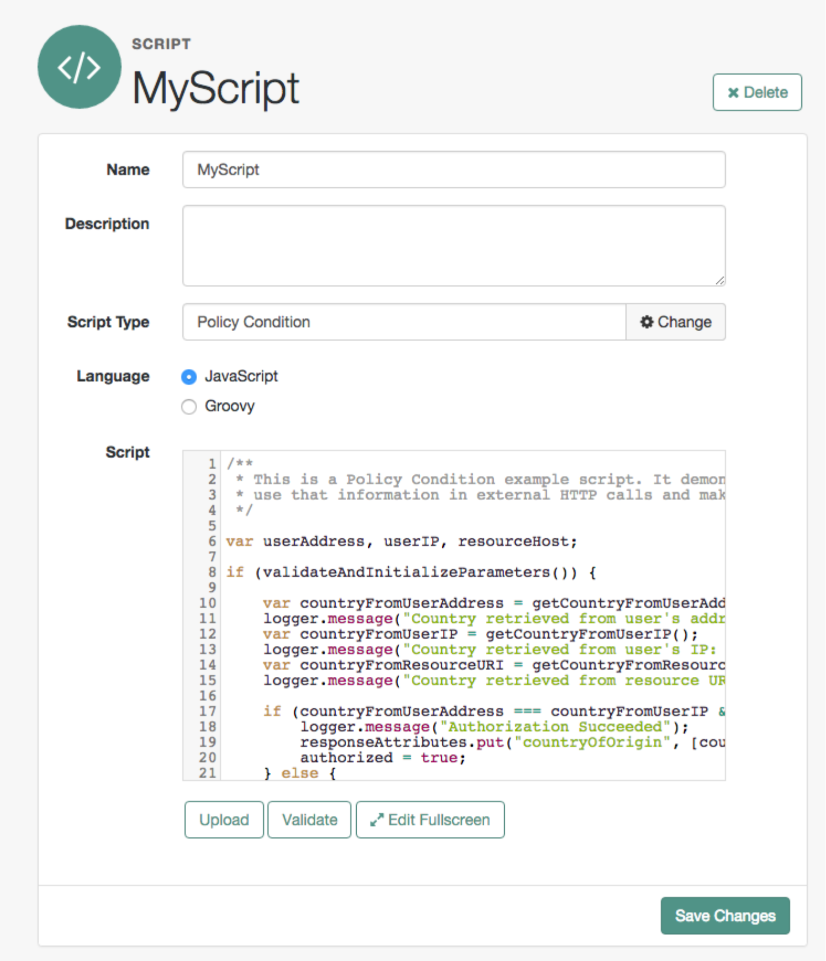
Script Type: Policy Condition

4. Specify a name for the script.

5. Select the type of script from the Script Type drop-down list.

6. Click Create.

   The *Script Name* page appears:



SCRIPT

MyScript

Name: MyScript

Description:

Script Type: Policy Condition

Language: ● JavaScript ○ Groovy

Script

```
/**
 * This is a Policy Condition example script. It demon
 * use that information in external HTTP calls and mak
 */

var userAddress, userIP, resourceHost;

if (validateAndInitializeParameters()) {

    var countryFromUserAddress = getCountryFromUserAdd
    logger.message("Country retrieved from user's addr
    var countryFromUserIP = getCountryFromUserIP();
    logger.message("Country retrieved from user's IP: 
    var countryFromResourceURI = getCountryFromResourc
    logger.message("Country retrieved from resource UR

    if (countryFromUserAddress === countryFromUserIP &
        logger.message("Authorization Succeeded");
        responseAttributes.put("countryOfOrigin", [cou
        authorized = true;
    } else {
```

7. Enter values on the *Script Name* page as follows:

   a. Enter a description of the script.

   b. Choose the script language, either JavaScript or Groovy. Note that not every script type supports both languages.

   c. Enter the source code in the Script field.

      On supported browsers, you can click Upload, navigate to the script file, and then click Open to upload the contents to the Script field.

   d. Click Validate to check for compilation errors in the script.

Correct any compilation errors, and revalidate the script until all errors have been fixed.

e. Save your changes.

Procedure 5.2. To Modify Scripts by Using the AM Console

1. Log in to the AM console as an AM administrator, for example, `amadmin`.
2. Navigate to Realms > *Realm Name* > Scripts.
3. Select the script you want to modify from the list of scripts.

   The *Script Name* page appears.
4. Modify values on the *Script Name* page as needed. Note that if you change the Script Type, existing code in the script is replaced.
5. If you modified the code in the script, click Validate to check for compilation errors.

   Correct any compilation errors, and revalidate the script until all errors have been fixed.
6. Save your changes.

Procedure 5.3. To Delete Scripts by Using the AM Console

1. Log in to the AM console as an AM administrator, for example, `amadmin`.
2. Navigate to Realms > *Realm Name* > Scripts.
3. Choose one or more scripts to delete by activating the checkboxes in the relevant rows. Note that you can only delete user-created scripts—you cannot delete the global sample scripts provided with AM.
4. Click Delete.

5.3.2. Managing Scripts With the ssoadm Command

Use the **ssoadm** command's **create-sub-cfg**, **get-sub-cfg**, and **delete-sub-cfg** subcommands to manage AM scripts.

Create an AM script as follows:

1. Create a script configuration file as follows:

```
script-file=/path/to/script-file
language=JAVASCRIPT|GROOVY
name=myScript
context=AUTHENTICATION_SERVER_SIDE|AUTHENTICATION_CLIENT_SIDE|POLICY_CONDITION|OIDC_CLAIMS
```

2. Run the **ssoadm create-sub-cfg** command. The `--datafile` argument references the script configuration file you created in the previous step:

```
$ ssoadm \
 create-sub-cfg \
 --realm /myRealm \
 --adminid amadmin \
 --password-file /tmp/pwd.txt \
 --servicename ScriptingService \
 --subconfigname scriptConfigurations/scriptConfiguration \
 --subconfigid myScript \
 --datafile /path/to/myScriptConfigurationFile
Sub Configuration scriptConfigurations/scriptConfiguration was added to realm /myRealm
```

To list the properties of a script, run the **ssoadm get-sub-cfg** command:

```
$ ssoadm \
  get-sub-cfg \
  --realm /myRealm \
  --adminid amadmin \
  --password-file /tmp/pwd.txt \
  --servicename ScriptingService \
  --subconfigname scriptConfigurations/myScript
createdBy=
lastModifiedDate=
lastModifiedBy=
name=myScript
context=POLICY_CONDITION
description=
language=JAVASCRIPT
creationDate=
script=...Script output follows...
```

To delete a script, run the **ssoadm delete-sub-cfg** command:

```
$ ssoadm \
  delete-sub-cfg \
  --realm /myRealm \
  --adminid amadmin \
  --password-file /tmp/pwd.txt \
  --servicename ScriptingService \
  --subconfigname scriptConfigurations/myScript
Sub Configuration scriptConfigurations/myScript was deleted from realm /myRealm
```

5.3.3. Managing Scripts With the REST API

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.

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

- Section 5.3.4, "Querying Scripts"
- Section 5.3.5, "Reading a Script"

- Section 5.3.6, "Validating a Script"
- Section 5.3.7, "Creating a Script"
- Section 5.3.8, "Updating a Script"
- Section 5.3.9, "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**
>
> AM 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": "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
}
```

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

`_id`

The UUID that AM 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:

```
dmFyIGEgPSAxMjM7IA0KdmFyIGIgPSA0NTY7
```

`language`

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

Table 5.3. 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`.

5.3.4. 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, AM 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/realms/root/realms/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
}
```

Table 5.4. Supported _queryFilter Fields and Operators

Field	Supported Operators
`_id`	Equals (`eq`), Contains (`co`), Starts with (`sw`)
`name`	Equals (`eq`), Contains (`co`), Starts with (`sw`)
`description`	Equals (`eq`), Contains (`co`), Starts with (`sw`)
`script`	Equals (`eq`), Contains (`co`), Starts with (`sw`)
`language`	Equals (`eq`), Contains (`co`), Starts with (`sw`)
`context`	Equals (`eq`), Contains (`co`), Starts with (`sw`)

5.3.5. 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/realms/root/realms/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
}
```

5.3.6. 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/realms/root/realms/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": "dmFyIGEgPSAxMjM7dmFyIGIgPSA0NTY7ID1WQUxJREFUSU9OIFNIT1VMRCBGQUlMPQo=",
    "language": "JAVASCRIPT"
  }' \
  https://openam.example.com:8443/openam/json/realms/root/realms/myrealm/scripts/?_action=validate
{
  "success": false,
  "errors": [
    {
      "line": 1,
      "column": 27,
      "message": "syntax error"
    }
  ]
}
```

5.3.7. 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, AM 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/realms/root/realms/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
}
```

5.3.8. 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, AM 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/realms/root/realms/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
}
```

5.3.9. 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, AM 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/realms/root/realms/myrealm/scripts/0168d494-015a-420f-ae5a
-6a2a5c1126af
{}
```

Chapter 6
Reference

This reference section covers settings and other information relating to developing with AM.

6.1. Scripting

ssoadm service name: `scripting`

6.1.1. Configuration

The following settings appear on the **Configuration** tab:

Default Script Type

The default script context type when creating a new script.

The possible values for this property are:

```
POLICY_CONDITION
AUTHENTICATION_SERVER_SIDE
AUTHENTICATION_CLIENT_SIDE
OIDC_CLAIMS
```

Default value: `POLICY_CONDITION`

ssoadm attribute: `defaultContext`

6.1.2. Secondary Configurations

This service has the following Secondary Configurations.

6.1.2.1. Engine Configuration

The following properties are available for Scripting Service secondary configuration instances:

Engine Configuration

Configure script engine parameters for running a particular script type in OpenAM.

ssoadm attribute: `engineConfiguration`

To access a secondary configuration instance using the **ssoadm** command, use: `--subconfigname [primary configuration]/[secondary configuration]` For example:

```
$ ssoadm set-sub-cfg \
  --adminid amAdmin \
  --password-file admin_pwd_file \
  --servicename ScriptingService \
  --subconfigname OIDC_CLAIMS/engineConfiguration \
  --operation set \
  --attributevalues maxThreads=300 queueSize=-1
```

> **Note**
>
> Supports server-side scripts only. OpenAM cannot configure engine settings for client-side scripts.

The configurable engine settings are as follows:

Server-side Script Timeout

The maximum execution time any individual script should take on the server (in seconds). OpenAM terminates scripts which take longer to run than this value.

ssoadm attribute: `serverTimeout`

Core thread pool size

The initial number of threads in the thread pool from which scripts operate. OpenAM will ensure the pool contains at least this many threads.

ssoadm attribute: `coreThreads`

Maximum thread pool size

The maximum number of threads in the thread pool from which scripts operate. If no free thread is available in the pool, OpenAM creates new threads in the pool for script execution up to the configured maximum.

ssoadm attribute: `maxThreads`

Thread pool queue size

The number of threads to use for buffering script execution requests when the maximum thread pool size is reached.

ssoadm attribute: `queueSize`

Thread idle timeout (seconds)

Length of time (in seconds) for a thread to be idle before OpenAM terminates created threads. If the current pool size contains the number of threads set in `Core thread pool size` idle threads will not be terminated, to maintain the initial pool size.

ssoadm attribute: `idleTimeout`

Java class whitelist

Specifies the list of class-name patterns allowed to be invoked by the script. Every class accessed by the script must match at least one of these patterns.

You can specify the class name as-is or use a regular expression.

ssoadm attribute: `whiteList`

Java class blacklist

Specifies the list of class-name patterns that are NOT allowed to be invoked by the script. The blacklist is applied AFTER the whitelist to exclude those classes - access to a class specified in both the whitelist and the blacklist will be denied.

You can specify the class name to exclude as-is or use a regular expression.

ssoadm attribute: `blackList`

Use system SecurityManager

If enabled, OpenAM will make a call to `System.getSecurityManager().checkPackageAccess(...)` for each class that is accessed. The method throws `SecurityException` if the calling thread is not allowed to access the package.

Note

This feature only takes effect if the security manager is enabled for the JVM.

ssoadm attribute: `useSecurityManager`

Scripting languages

Select the languages available for scripts on the chosen type. Either `GROOVY` or `JAVASCRIPT`.

ssoadm attribute: `languages`

Default Script

The source code that is presented as the default when creating a new script of this type.

ssoadm attribute: `defaultScript`

Appendix A. Getting Support

For more information or resources about AM and ForgeRock Support, see the following sections:

A.1. 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.
- ForgeRock core 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.

Core documentation therefore follows a three-phase review process designed to eliminate errors:

- Product managers and software architects review project documentation design with respect to the readers' software lifecycle needs.
- Subject matter experts review proposed documentation changes for technical accuracy and completeness with respect to the corresponding software.
- Quality experts validate implemented documentation changes for technical accuracy, completeness in scope, and usability for the readership.

The review process helps to ensure that documentation published for a ForgeRock release is technically accurate and complete.

Fully reviewed, published core documentation is available at http://backstage.forgerock.com/. Use this documentation when working with a ForgeRock Identity Platform release.

A.2. Joining the ForgeRock Community

Visit the Community resource center where you can find information about each project, download trial builds, browse the resource catalog, ask and answer questions on the forums, find community events near you, and find the source code for open source software.

A.3. Getting Support and Contacting ForgeRock

ForgeRock provides support services, professional services, classes 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.

Glossary

Access control

Control to grant or to deny access to a resource.

Account lockout

The act of making an account temporarily or permanently inactive after successive authentication failures.

Actions

Defined as part of policies, these verbs indicate what authorized subjects can do to resources.

Advice

In the context of a policy decision denying access, a hint to the policy enforcement point about remedial action to take that could result in a decision allowing access.

Agent administrator

User having privileges only to read and write policy agent profile configuration information, typically created to delegate policy agent profile creation to the user installing a policy agent.

Agent authenticator

Entity with read-only access to multiple agent profiles defined in the same realm; allows an agent to read web service profiles.

Application

In general terms, a service exposing protected resources.

In the context of AM policies, the application is a template that constrains the policies that govern access to protected resources. An application can have zero or more policies.

Application type

Application types act as templates for creating policy applications.

Application types define a preset list of actions and functional logic, such as policy lookup and resource comparator logic.

Application types also define the internal normalization, indexing logic, and comparator logic for applications.

Attribute-based access control (ABAC)

Access control that is based on attributes of a user, such as how old a user is or whether the user is a paying customer.

Authentication

The act of confirming the identity of a principal.

Authentication chaining

A series of authentication modules configured together which a principal must negotiate as configured in order to authenticate successfully.

Authentication level

Positive integer associated with an authentication module, usually used to require success with more stringent authentication measures when requesting resources requiring special protection.

Authentication module

AM authentication unit that handles one way of obtaining and verifying credentials.

Authorization

The act of determining whether to grant or to deny a principal access to a resource.

Authorization Server

In OAuth 2.0, issues access tokens to the client after authenticating a resource owner and confirming that the owner authorizes the client to access the protected resource. AM can play this role in the OAuth 2.0 authorization framework.

Auto-federation

Arrangement to federate a principal's identity automatically based on a common attribute value shared across the principal's profiles at different providers.

Bulk federation

Batch job permanently federating user profiles between a service provider and an identity provider based on a list of matched user identifiers that exist on both providers.

Circle of trust

Group of providers, including at least one identity provider, who have agreed to trust each other to participate in a SAML v2.0 provider federation.

Client

In OAuth 2.0, requests protected web resources on behalf of the resource owner given the owner's authorization. AM can play this role in the OAuth 2.0 authorization framework.

Conditions

Defined as part of policies, these determine the circumstances under which which a policy applies.

Environmental conditions reflect circumstances like the client IP address, time of day, how the subject authenticated, or the authentication level achieved.

Subject conditions reflect characteristics of the subject like whether the subject authenticated, the identity of the subject, or claims in the subject's JWT.

Configuration datastore

LDAP directory service holding AM configuration data.

Cross-domain single sign-on (CDSSO)

AM capability allowing single sign-on across different DNS domains.

Delegation

Granting users administrative privileges with AM.

Entitlement

Decision that defines which resource names can and cannot be accessed for a given subject in the context of a particular application, which actions are allowed and which are denied, and any related advice and attributes.

Extended metadata

Federation configuration information specific to AM.

Extensible Access Control Markup Language (XACML)

Standard, XML-based access control policy language, including a processing model for making authorization decisions based on policies.

Federation

Standardized means for aggregating identities, sharing authentication and authorization data information between trusted providers, and allowing principals to access services across different providers without authenticating repeatedly.

Fedlet

Service provider application capable of participating in a circle of trust and allowing federation without installing all of AM on the service provider side; AM lets you create Java Fedlets.

Hot swappable

Refers to configuration properties for which changes can take effect without restarting the container where AM runs.

Identity

Set of data that uniquely describes a person or a thing such as a device or an application.

Identity federation

Linking of a principal's identity across multiple providers.

Identity provider (IdP)

Entity that produces assertions about a principal (such as how and when a principal authenticated, or that the principal's profile has a specified attribute value).

Identity repository

Data store holding user profiles and group information; different identity repositories can be defined for different realms.

Java EE policy agent

Java web application installed in a web container that acts as a policy agent, filtering requests to other applications in the container with policies based on application resource URLs.

Metadata

Federation configuration information for a provider.

Policy

Set of rules that define who is granted access to a protected resource when, how, and under what conditions.

Policy Agent

Agent that intercepts requests for resources, directs principals to AM for authentication, and enforces policy decisions from AM.

Policy Administration Point (PAP)

Entity that manages and stores policy definitions.

Policy Decision Point (PDP)

Entity that evaluates access rights and then issues authorization decisions.

Policy Enforcement Point (PEP)

Entity that intercepts a request for a resource and then enforces policy decisions from a PDP.

Policy Information Point (PIP)

Entity that provides extra information, such as user profile attributes that a PDP needs in order to make a decision.

Principal

Represents an entity that has been authenticated (such as a user, a device, or an application), and thus is distinguished from other entities.

When a Subject successfully authenticates, AM associates the Subject with the Principal.

Privilege

In the context of delegated administration, a set of administrative tasks that can be performed by specified subjects in a given realm.

Provider federation

Agreement among providers to participate in a circle of trust.

Realm

AM unit for organizing configuration and identity information.

Realms can be used for example when different parts of an organization have different applications and user data stores, and when different organizations use the same AM deployment.

Administrators can delegate realm administration. The administrator assigns administrative privileges to users, allowing them to perform administrative tasks within the realm.

Resource

Something a user can access over the network such as a web page.

Defined as part of policies, these can include wildcards in order to match multiple actual resources.

Resource owner

In OAuth 2.0, entity who can authorize access to protected web resources, such as an end user.

Resource server

In OAuth 2.0, server hosting protected web resources, capable of handling access tokens to respond to requests for such resources.

Response attributes

Defined as part of policies, these allow AM to return additional information in the form of "attributes" with the response to a policy decision.

Role based access control (RBAC)

Access control that is based on whether a user has been granted a set of permissions (a role).

Security Assertion Markup Language (SAML)

Standard, XML-based language for exchanging authentication and authorization data between identity providers and service providers.

Service provider (SP)

Entity that consumes assertions about a principal (and provides a service that the principal is trying to access).

Session

The interval that starts with the user authenticating through AM and ends when the user logs out, or when their session is terminated. For browser-based clients, AM manages user sessions across one or more applications by setting a session cookie. See also Stateful session and Stateless session.

Session high availability

Capability that lets any AM server in a clustered deployment access shared, persistent information about users' sessions from the CTS token store. The user does not need to log in again unless the entire deployment goes down.

Session token

Unique identifier issued by AM after successful authentication. For a Stateful session, the session token is used to track a principal's session.

Single log out (SLO)

Capability allowing a principal to end a session once, thereby ending her session across multiple applications.

Single sign-on (SSO)

Capability allowing a principal to authenticate once and gain access to multiple applications without authenticating again.

Site

Group of AM servers configured the same way, accessed through a load balancer layer.

The load balancer handles failover to provide service-level availability. Use sticky load balancing based on `amlbcookie` values to improve site performance.

The load balancer can also be used to protect AM services.

Standard metadata

Standard federation configuration information that you can share with other access management software.

Stateful session

An AM session that resides in the Core Token Service's token store. Stateful sessions might also be cached in memory on one or more

AM servers. AM tracks stateful sessions in order to handle events like logout and timeout, to permit session constraints, and to notify applications involved in SSO when a session ends.

Stateless session

An AM session for which state information is encoded in AM and stored on the client. The information from the session is not retained in the CTS token store. For browser-based clients, AM sets a cookie in the browser that contains the session information.

Subject

Entity that requests access to a resource

When a subject successfully authenticates, AM associates the subject with the Principal that distinguishes it from other subjects. A subject can be associated with multiple principals.

User data store

Data storage service holding principals' profiles; underlying storage can be an LDAP directory service, a relational database, or a custom `IdRepo` implementation.

Web policy agent

Native library installed in a web server that acts as a policy agent with policies based on web page URLs.