



Development Guide

/ ForgeRock Access Management 6.5

Latest update: 6.5.4

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

Copyright © 2011-2020 ForgeRock AS.

Abstract

Guide to developing REST clients and scripts. ForgeRock® Access Management provides authentication, authorization, entitlement and federation software.



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

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

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

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

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

DejaVu Fonts

Bitstream Vera Fonts Copyright

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

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

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

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

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

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

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

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

Arev Fonts Copyright

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

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

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

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

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

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

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

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

FontAwesome Copyright

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

This Font Software is licensed under the SIL Open Font License, Version 1.1. See <https://opensource.org/licenses/OFL-1.1>.

Table of Contents

Preface	iv
1. Introducing AM APIs	1
1.1. IPv4 and IPv6	1
2. Developing with the REST API	2
2.1. Introducing REST	2
2.2. Introducing the API Explorer	2
2.3. About ForgeRock Common REST	6
2.4. REST API Versioning	24
2.5. Cross-Site Request Forgery (CSRF) Protection	28
2.6. Specifying Realms in REST API Calls	28
2.7. Authentication and Logout using REST	29
2.8. Using the Session Token After Authentication	48
2.9. Server Information	49
2.10. Token Encoding	50
2.11. Logging	50
2.12. REST Goto URL Validation	52
2.13. Reference	53
3. Developing with Scripts	56
3.1. The Scripting Environment	56
3.2. Managing Scripts	60
3.3. Global Scripting API Functionality	73
3.4. Authentication API Functionality	75
3.5. Scripted Decision Node API Functionality	78
3.6. Authorization API Functionality	84
3.7. OpenID Connect 1.0 Claims API Functionality	88
4. Reference	91
4.1. Scripting	91
A. Getting Support	95
Glossary	96

Preface

This guide provides an introduction to the ForgeRock Access Management REST API, and to developing scripts for client-side and server-side authentication, policy conditions, and handling OpenID Connect claims.

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

- Custom OAuth 2.0 scopes plugins define how AM, 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 ["Customizing OAuth 2.0 Scope Handling"](#) in the *OAuth 2.0 Guide*.

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

For more information, see ["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 ["Creating Post-Authentication Plugins for Chains"](#) 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 ["Customizing Policy Evaluation With a Plug-In"](#) in the *Authorization Guide*.

- Identity repository plugins let AM employ a new or custom identity store.

For more information, see ["Customizing Identity Data Storage"](#) in the *Setup and Maintenance Guide*.

About ForgeRock Identity Platform™ Software

ForgeRock Identity Platform™ serves as the basis for our simple and comprehensive Identity and Access Management solution. We help our customers deepen their relationships with their customers, and improve the productivity and connectivity of their employees and partners. For more information about ForgeRock and about the platform, see <https://www.forgerock.com>.

Chapter 1

Introducing AM APIs

Although web and Java 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 APIs.

AM provides the following application programming interfaces:

- **REST APIs**

AM REST APIs 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.

For more information, see "*Developing with the REST API*".

- **Java APIs**

AM Java APIs let your Java and Java applications call on AM for authentication and authorization.

For more information, see the *ForgeRock Access Management Java API Specification*.

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 administration console. It displays the REST API endpoints that allow client applications to access AM's services.

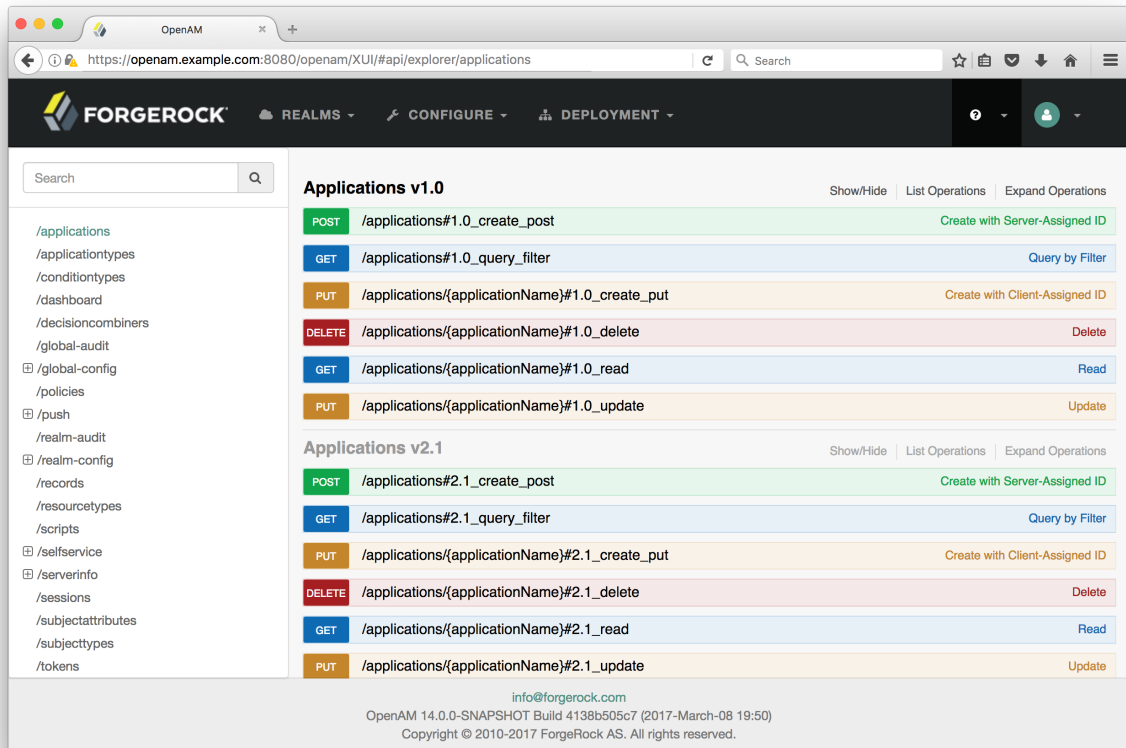
Tip

The API Explorer is enabled by default. To disable it in production environments, navigate to [Configure > Global Services > REST APIs](#), and select Disabled in the API Descriptors drop-down list.

The key features of the API Explorer are the following:

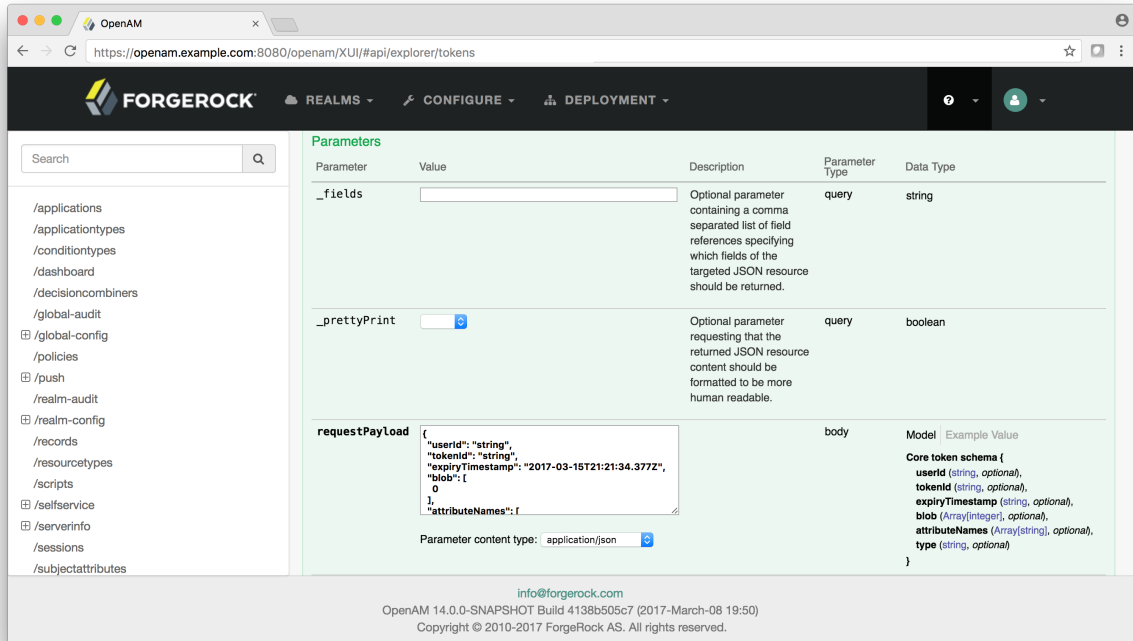
- **API Versioning.** The API Explorer displays the different API versions available depending on your deployment.

API Explorer



- 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:

API Explorer Request Payload



The screenshot shows the OpenAM API Explorer interface. The left sidebar contains a search bar and a list of API endpoints. The main area displays the details for the `/api/explorer/tokens` endpoint. It includes a table of parameters and a section for the request payload.

Parameter	Value	Description	Parameter Type	Data Type
<code>_fields</code>	<input type="text"/>	Optional parameter containing a comma separated list of field references specifying which fields of the targeted JSON resource should be returned.	query	string
<code>_prettyPrint</code>	<input type="checkbox"/>	Optional parameter requesting that the returned JSON resource content should be formatted to be more human readable.	query	boolean

requestPayload

```
{
  "userid": "string",
  "tokenId": "string",
  "expiryTimestamp": "2017-03-15T21:21:34.377Z",
  "blob": {
    0
  },
  "attributeNames": f
}
```

Parameter content type: `application/json`

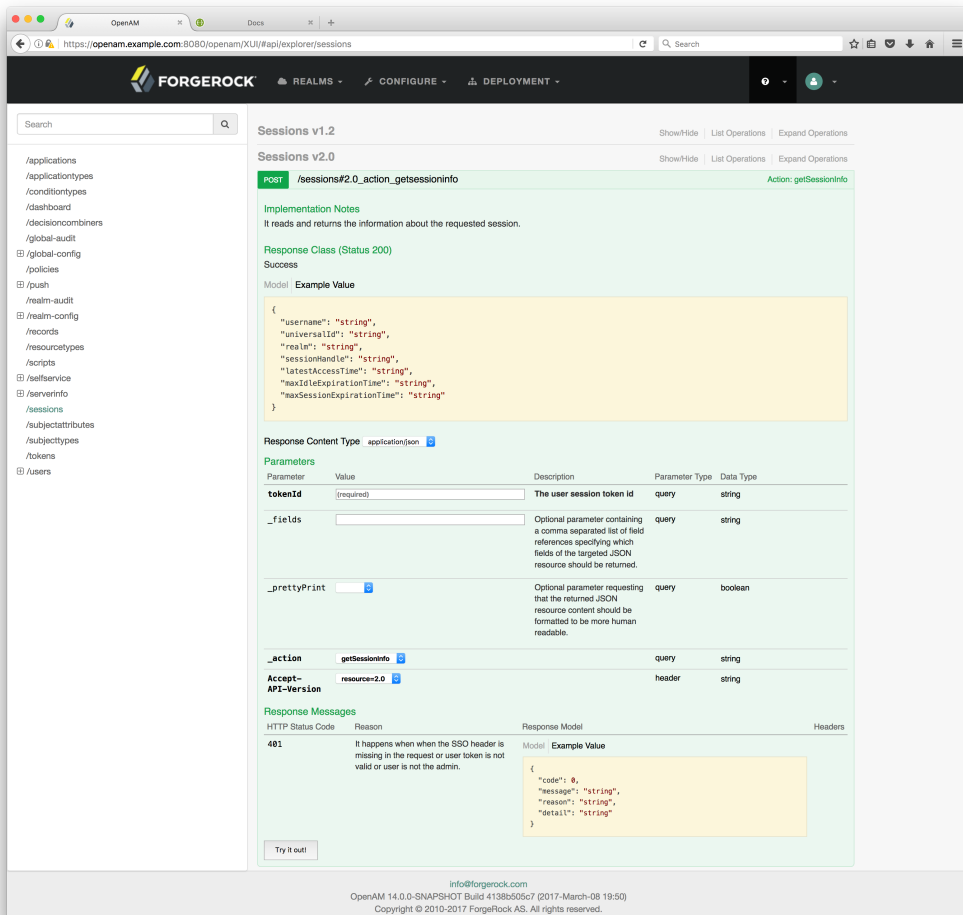
Model Example Value

```
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.

API Explorer Detailed Information



To Access the API Explorer

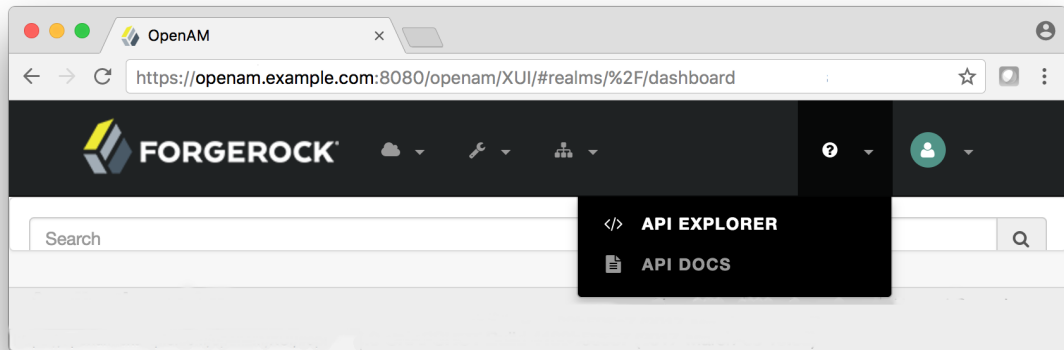
1. Log into the AM console as an administrator.
2. You can access the API Explorer in one of two ways:

Point your browser to the following URL:

```
https://openam.example.com:8443/openam/XUI/#api/explorer/applications
```

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

API Explorer



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.

Note

This section describes the full Common REST framework. Some platform component products do not implement all Common REST behaviors exactly as described in this section. For details, refer to the product-specific examples and reference information in other sections of this documentation set.

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 "Create".

Read

Retrieve a single resource.

This verb maps to HTTP GET.

For details, see "Read".

Update

Replace an existing resource.

This verb maps to HTTP PUT.

For details, see "Update".

Delete

Remove an existing resource.

This verb maps to HTTP DELETE.

For details, see "Delete".

Patch

Modify part of an existing resource.

This verb maps to HTTP PATCH.

For details, see "Patch".

Action

Perform a predefined action.

This verb maps to HTTP POST.

For details, see "Action".

Query

Search a collection of resources.

This verb maps to HTTP GET.

For details, see "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 "To Publish OpenAPI Documentation" instead.

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. (Optional) 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"`.

If the `field` is left blank, the server returns all default values.

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"`.

If the `field` is left blank, the server returns all default values.

`_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"`.

If the `field` is left blank, the server returns all default values.

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"`.

If the `field` is left blank, the server returns all default values.

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 "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 "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 "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 `URLConnection.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"`.

If the `field` is left blank, the server returns all default values.

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 "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"`.

If the `field` is left blank, the server returns all default values.

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.
- The value it points to is not `null`.

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.

Because ascending order is the default, including the `+` character in the query is unnecessary. If you do include the `+`, it must be URL-encoded as `%2B`, for example:

```
http://localhost:8080/api/users?_prettyPrint=true&_queryFilter=true&_sortKeys=%2Bname/givenName
```

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"`.

If the `field` is left blank, the server returns all default values.

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.

To ensure your clients are always compatible with a newer version of AM, you should always include resource versions in your REST calls.

2.4.1. Supported REST API Versions

For information about the supported protocol and resource versions available in AM, see the [API Explorer](#) available in the AM console.

The *AM Release Notes* section, "*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. The following example requests *resource* version 2.0 and *protocol* version 1.0:

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

- "Configure Versioning Behavior by using the AM Console"
- "Configure Versioning Behavior by Using the ssoadm Command"

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 "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.

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. (Optional) Optionally, enable **Warning Header** to include warning messages in the headers of responses to requests.
5. Save your work.

Configure Versioning Behavior by Using the ssoadm Command

- 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 "Content-Type: application/json" \
--header "X-OpenAM-Username: demo" \
--header "X-OpenAM-Password: changeit" \
--header "Accept-API-Version: resource=2.0, protocol=1.0" \
'https://openam.example.com:8443/openam/json/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...TU30Q*",
  "successUrl": "/openam/console"
}
```

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

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

2.5. Cross-Site Request Forgery (CSRF) Protection

AM includes a global filter to harden AM's protection against CSRF attacks. The filter applies to all REST endpoints under `json/` and requires that all requests other than GET, HEAD, or OPTIONS include, at least, one of the following headers:

- `X-Requested-With`

This header is often sent by Javascript frameworks, and the XUI already sends it on all requests.

- `Accept-API-Version`

This header specifies which version of the REST API to use. Use this header in your requests to ensure future changes to the API do not affect your clients.

For more information about API versioning, see "REST API Versioning".

Failure to include at least one of the headers would cause the REST call to fail with a `403 Forbidden` error, even if the SSO token is valid.

To disable the filter, navigate to Configure > Global Services > REST APIs > and turn off Enable CSRF Protection.

The `json/` endpoint is not vulnerable to CSRF attacks when the filter is disabled, since it requires the `"Content-Type: application/json"` header, which currently triggers the same protection in browsers. This may change in the future, so it is recommended to enable the CSRF filter.

2.6. 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=*
```

2.7. Authentication and Logout using REST

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*.

After a successful authentication, AM returns a `tokenId` object that applications can present as a cookie value for other operations that require authentication. This object is a `session` in the *Authentication and Single Sign-On Guide* token—a representation of the exchange of information and credentials between AM and the user or identity.

The type of `tokenId` returned varies depending on where AM stores the sessions for the realm to which the user authenticates:

- If CTS-based sessions are enabled, the `tokenId` object is a reference to the session state stored in the CTS token store.
- If client-based sessions are enabled, the `tokenId` object is the session state for that particular user or identity.

Developers should be aware that the size of the `tokenId` for client-based sessions—2000 bytes or greater—is considerably longer than for CTS-based sessions—approximately 100 bytes. For more

information about session tokens, see "Session Cookies" in the *Authentication and Single Sign-On Guide*.

2.7.1. Authenticating to AM using REST

To log in to AM using REST, make an HTTP POST request to the `/json/authenticate` 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`.

AM uses the default authentication service configured for the realm. You can override the default by specifying authentication services and other options in the REST request.

AM provides both simple authentication methods, such as providing user name and password, and complex authentication journeys that may involve a tree with inner tree evaluation and/or multi-factor authentication.

For authentication journeys where providing a user name and password is enough, you can log in to AM using a `curl` command similar to the following:

```
$ curl \
--request POST \
--header "Content-Type: application/json" \
--header "X-OpenAM-Username: demo" \
--header "X-OpenAM-Password: changeit" \
--header "Accept-API-Version: resource=2.0, protocol=1.0" \
'https://openam.example.com:8443/openam/json/realms/root/authenticate'
{
  "tokenId": "AQIC5w...NTcy*",
  "successUrl": "/openam/console",
  "realm": "/"
}
```

The user name and password are sent in headers. This zero page login mechanism works only for name/password authentication.

Note that the POST body is empty; otherwise, AM interprets the body as a continuation of an existing authentication attempt, one that uses a supported `callback` mechanism. AM implements callback mechanisms to support complex authentication journeys, such as those where the user needs to be redirected to a third party or interact with a device as part of multi-factor authentication.

When a client makes a call to the `/json/authenticate` endpoint appending a valid SSO token, AM returns the `tokenId` field **empty** when `HttpOnly` cookies are enabled. For example:

```
{
  "tokenId": "",
  "successUrl": "/openam/console",
  "realm": "/"
}
```

Tip

About Success and Failure URLs

On authentication success, AM returns an SSO token and a success URL. By default, users are redirected to `/openam/console`.

No failure URL is configured by default. When configured, on authentication failure, AM returns HTTP status code 401 Unauthorized and the failure URL:

```
{
  "code": 401,
  "reason": "Unauthorized",
  "message": "Login failure",
  "failureUrl": "http://www.example.com/401.html"
}
```

For more information about configuring successful or failed authentication, see "Configuring Success and Failure Redirection URLs" in the *Authentication and Single Sign-On Guide*.

Using UTF-8 User Names

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

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

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

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

```
$ curl \
--request POST \
--header "Content-Type: application/json" \
--header "X-OpenAM-Username: =?UTF-8?B?yZfDq8mxw7g=?=" \
--header "X-OpenAM-Password: changeit" \
--header "Accept-API-Version: resource=2.0, protocol=1.0" \
'https://openam.example.com:8443/openam/json/realms/root/authenticate'
{
  "tokenId": "AQIC5w...NTcy*",
  "successUrl": "/openam/console",
  "realm": "/"
}
```

2.7.1.1. Authenticating to Specific Authentication Services

You can provide AM with additional information about how you are authenticating. For example, you can specify the authentication tree you want to use, or request from AM a list of the authentication services that would satisfy a particular authentication condition.

The following example shows how to specify the `ldapService` chain by using the `authIndexType` and `authIndexValue` query string parameters:

```
$ 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?
authIndexType=service&authIndexValue=ldapService'
```

You can exchange the `ldapService` chain with any other chain or tree.

For more information about using the `authIndexType` parameter to authenticate to specific services, see "Authenticate Endpoint Parameters".

2.7.1.2. Authenticate Endpoint Parameters

To authenticate to AM using REST, make an HTTP POST request to the `/json/authenticate` 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`.

The following list describes the `/json/authenticate` endpoint supported parameters:

`authIndexType`

Specifies the type of authentication the user will perform. Always use in conjunction with the `authIndexValue` parameter to provide additional information about the way the user is authenticating.

If not specified, AM authenticates the user against the default authentication service configured for the realm.

The `authIndexType` parameter supports the following types:

- `composite_advice`

Specifies that the value of the `authIndexValue` parameter is a URL-encoded composite advice string.

Use `composite_advice` when you want to give AM hints of which authentication services to use when logging in a user. For example, use an authentication service that provides an authentication level of 10 or higher:

```
$ curl -get \
--request POST \
--header "Content-Type: application/json" \
--header 'Accept-API-Version: resource=2.0, protocol=1.0' \
--data-urlencode 'authIndexType=composite_advice' \
--data-urlencode 'authIndexValue=<AdVICES>
  <AttributeValuePair>
    <Attribute name="AuthLevelConditionAdvice"/>
    <Value>10</Value>
  </AttributeValuePair>
</AdVICES>' \
'https://openam.example.com:8443/openam/json/realms/root/authenticate'
```

Note that the previous `curl` command URL-encodes the XML values, and the `-G` parameter appends them as query string parameters to the URL.

Possible options for advices are:

- **TransactionConditionAdvice**. Requires the unique ID of a transaction token. For example:

```
<Advices>
  <AttributeValuePair>
    <Attribute name="TransactionConditionAdvice"/>
    <Value>9dae2c80-fe7a-4a36-b57b-4fb1271b0687</Value>
  </AttributeValuePair>
</Advices>
```

For more information, see *"Implementing Transactional Authorization"* in the *Authorization Guide*.

- **AuthenticateToServiceConditionAdvice**. Requires the name of an authentication chain or tree. For example:

```
<Advices>
  <AttributeValuePair>
    <Attribute name="AuthenticateToServiceConditionAdvice"/>
    <Value>myExampleTree</Value>
  </AttributeValuePair>
</Advices>
```

- **AuthSchemeConditionAdvice**. Requires the name of an authentication module. For example:

```
<Advices>
  <AttributeValuePair>
    <Attribute name="AuthSchemeConditionAdvice"/>
    <Value>DataStoreModule</Value>
  </AttributeValuePair>
</Advices>
```

- **AuthenticateToRealmConditionAdvice**. Requires the name of a realm. For example:

```
<Advices>
  <AttributeValuePair>
    <Attribute name="AuthenticateToRealmConditionAdvice"/>
    <Value>myRealm</Value>
  </AttributeValuePair>
</Advices>
```

- **AuthLevelConditionAdvice**. Requires an authentication level. For example:

```
<Advices>
  <AttributeValuePair>
    <Attribute name="AuthLevelConditionAdvice"/>
    <Value>10</Value>
  </AttributeValuePair>
</Advices>
```

- **AuthenticateToTreeConditionAdvice**. Requires the name of an authentication tree. For example:

```
<Advices>
  <AttributeValuePair>
    <Attribute name="AuthenticateToTreeConditionAdvice"/>
    <Value>PersistentCookieTree</Value>
  </AttributeValuePair>
</Advices>
```

You can specify multiple advice conditions and combine them. For example:

```
<Advices>
  <AttributeValuePair>
    <Attribute name="AuthenticateToServiceConditionAdvice"/>
    <Value>ldapService</Value>
  </AttributeValuePair>
  <AttributeValuePair>
    <Attribute name="AuthenticateToServiceConditionAdvice"/>
    <Value>Example</Value>
  </AttributeValuePair>
  <AttributeValuePair>
    <Attribute name="AuthLevelConditionAdvice"/>
    <Value>10</Value>
  </AttributeValuePair>
</Advices>
```

- **level**

Specifies that the value of the **authIndexValue** parameter is the minimum authentication level an authentication service must satisfy to log in the user.

For example, to log into AM using an authentication service that provides a minimum authentication level of 10, you could use the following:

```
$ curl \
--request POST \
--header 'Accept-API-Version: resource=2.0, protocol=1.0' \
'https://openam.example.com:8443/openam/json/realms/root/authenticate?
authIndexType=level&authIndexValue=10'
```

- **module**

Specifies that the value of the **authIndexValue** parameter is the name of the authentication module AM must use to log in the user.

For example, to log into AM using the built-in **DataStore** authentication module, you could use the following:

```
$ curl \
--request POST \
--header 'Accept-API-Version: resource=2.0, protocol=1.0' \
'https://openam.example.com:8443/openam/json/realms/root/authenticate?
authIndexType=module&authIndexValue=DataStore'
```

- **resource**

Specifies that the value of the `authIndexValue` parameter is a URL protected by an AM policy.

For example, to log into AM using a policy matching the `http://www.example.com` resource, you could use the following:

```
$ curl \
--request POST \
--header 'Accept-API-Version: resource=2.0, protocol=1.0' \
'https://openam.example.com:8443/openam/json/realms/root/authenticate?
authIndexType=resource&authIndexValue=http%3A%2F%2Fwww.example.com'
```

Note that the resource must be URL-encoded. Authentication will fail if no policy matches the resource.

- service

Specifies that the value of the `authIndexValue` parameter is the name of an authentication tree or authentication chain AM must use to log in the user.

For example, to log in to AM using the built-in `ldapService` authentication chain, you could use the following:

```
$ curl \
--request POST \
--header 'Accept-API-Version: resource=2.0, protocol=1.0' \
'https://openam.example.com:8443/openam/json/realms/root/authenticate?
authIndexType=service&authIndexValue=ldapService'
```

- user

Specifies that the value of the `authIndexValue` parameter is a valid user ID. AM will then authenticate the user against the chain configured in the User Authentication Configuration field of that user's profile.

For example, for the user `demo` to log into AM using the chain specified in their user profile, you could use the following:

```
$ curl \
--request POST \
--header 'Accept-API-Version: resource=2.0, protocol=1.0' \
'https://openam.example.com:8443/openam/json/realms/root/authenticate?
authIndexType=user&authIndexValue=demo'
```

Authentication will fail if the User Authentication Configuration field is empty for the user.

If several authentication services that satisfy the authentication requirements are available, AM presents them as a choice callback to the user. Return the required `callbacks` to AM to authenticate.

Required: No.

authIndexValue

Specifies the value of the `authIndexType` parameter.

Required: Yes, when using the `authIndexType` parameter.

noSession

When set to `true`, specifies that AM should not return a session when authenticating a user. For example:

```
$ curl \
--request POST \
--header "Content-Type: application/json" \
--header "Accept-API-Version: resource=2.0, protocol=1.0" \
--header "X-OpenAM-Username: demo" \
--header "X-OpenAM-Password: changeit" \
'https://openam.example.com:8443/openam/json/realms/root/authenticate?noSession=true'
{
  "message": "Authentication Successful",
  "successUrl": "/openam/console",
  "realm": "/"
}
```

Required: No.

2.7.1.3. Returning Callback Information to AM

The `/json/authenticate` endpoint supports callback mechanisms to perform complex authentication journeys. Whenever AM needs to return or request information, it will return a JSON object with the authentication step, the authentication identifier, and the related callbacks.

The following types of callbacks are available:

- *Read-only callbacks.* AM uses read-only callbacks to provide information to the user, such as text messages or the amount of time that the user needs to wait before continuing their authentication journey.
- *Interactive callbacks.* AM uses interactive callbacks ask the user for information. For example, to request their user name and password, or to request that the user chooses between different options.
- *Backchannel callbacks.* AM uses backchannel callbacks when it needs to access additional information from the user's request. For example, when it requires a particular header or a certificate.

Read-only and interactive callbacks have an array of `output` elements suitable for displaying to the end user. The JSON returned in interactive callbacks also contains an array of `input` elements, which must be completed and returned to AM. For example:

```
"output": [
  {
    "name": "prompt",
    "value": " User Name: "
  }
],
"input": [
  {
    "name": "IDToken1",
    "value": ""
  }
]
]
```

The value of some interactive callbacks can be returned as headers, such as the `X-OpenAM-Username` and `X-OpenAM-Password` headers, but most of them must be returned in JSON as a response to the request.

Depending on how complex the authentication journey is, AM may return several callbacks sequentially. Each must be completed and returned to AM until authentication is successful.

The following example shows a request for authentication, and AM's response of the `NameCallback` and `PasswordCallback` callbacks:

```
$ curl \
--request POST \
--header "Content-Type: application/json" \
--header "Accept-API-Version: resource=2.0, protocol=1.0" \
'https://openam.example.com:8443/openam/json/realms/root/authenticate'
```

```
{
  "authId": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJvdGsiOiJ...\"", ❶
  "template": "", ❷
  "stage": "DataStore1", ❸
  "callbacks": [
    {
      "type": "NameCallback", ❹
      "output": [ ❺
        {
          "name": "prompt",
          "value": " User Name: "
        }
      ],
      "input": [ ❻
        {
          "name": "IDToken1",
          "value": ""
        }
      ]
    },
    {
      "type": "PasswordCallback", ❹
      "output": [ ❺
        {
          "name": "prompt",
          "value": " Password: "
        }
      ]
    }
  ]
}
```

```

    ],
    "input": [ ❹
      {
        "name": "IDToken2",
        "value": ""
      }
    ]
  }
}
}

```

Key:

- ❶ The JWT that uniquely identifies the authentication context to AM.
- ❷ A template to customize the look of the authentication module, if exists. For more information, see [How do I customize the Login page?](#) in the *ForgeRock Knowledge Base*.
- ❸ The authentication module stage where the authentication journey is at the moment.
- ❹ The type of callback. It must be one the "Supported Callbacks".
- ❺ The information AM offers about this callback. Usually, this information would be displayed to the user in the UI.
- ❻ The information AM is requesting. The user must fill the `"value": ""` object with the required information.

To respond to a callback, send back the whole JSON object with the missing values filled. The following example shows how to respond to the `NameCallback` and `PasswordCallback` callbacks, with the `demo` and `changeit` values filled:

```

$ curl \
--request POST \
--header "Content-Type: application/json" \
--header "Accept-API-Version: resource=2.0, protocol=1.0" \
--data '{
  "authId": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJvdiGsiOiJ...",
  "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",
    "realm": "/"
}

```

On complex authentication journeys, AM may send several callbacks sequentially. Each must be completed and returned to AM until authentication is successful.

2.7.1.3.1. Supported Callbacks

The following types of callbacks are available:

- Interactive Callbacks
- Read-only Callbacks
- Backchannel Callbacks

Interactive Callbacks

AM returns the following callbacks to request information from the user:

ChoiceCallback

Used to display a list of choices and retrieve the selected choice. To indicate that the user selected the first choice, return a value of **0** to AM. For the second choice, return a value of **1**, and so forth. For example:

```
"callbacks":[
  {
    "type":"ChoiceCallback",
    "output":[
      {
        "name":"prompt",
        "value":"Choose one"
      },
      {
        "name":"choices",
        "value":[
          "Choice A",
          "Choice B",
          "Choice C"
        ]
      },
      {
        "name":"defaultChoice",
        "value":2
      }
    ],
    "input":[
      {
        "name":"IDToken1",
        "value":0
      }
    ]
  }
]
```

Class to import: `javax.security.auth.callback.ChoiceCallback`

ConfirmationCallback

Used to ask for a boolean-style confirmation, such as yes/no or true/false, and retrieve the response. Also can present a "Cancel" option. To indicate that the user selected the first choice, return a value of `0` to AM. For the second choice, return a value of `1`, and so forth. For example:

```
"callbacks":[
  {
    "type":"ConfirmationCallback",
    "output":[
      {
        "name":"prompt",
        "value":""
      },
      {
        "name":"messageType",
        "value":0
      },
      {
        "name":"options",
        "value":[
          "Submit",
          "Start Over",
          "Cancel"
        ]
      }
    ]
  }
]
```

```

    },
    {
      "name": "optionType",
      "value": -1
    },
    {
      "name": "defaultOption",
      "value": 1
    }
  ],
  "input": [
    {
      "name": "IDToken2",
      "value": 0
    }
  ]
}
]

```

Class to import: `javax.security.auth.callback.ConfirmationCallback`

NameCallback

Used to retrieve a data string which can be entered by the user. Usually used for collecting user names. For example:

```

"callbacks": [
  {
    "type": "NameCallback",
    "output": [
      {
        "name": "prompt",
        "value": "User Name"
      }
    ],
    "input": [
      {
        "name": "IDToken1",
        "value": ""
      }
    ]
  }
]

```

Class to import: `javax.security.auth.callback.NameCallback`

PasswordCallback

Used to retrieve a password value. For example:

```
"callbacks":[
  {
    "type":"PasswordCallback",
    "output":[
      {
        "name":"prompt",
        "value":"Password"
      }
    ],
    "input":[
      {
        "name":"IDToken1",
        "value":""
      }
    ]
  }
]
```

Class to import: `javax.security.auth.callback.PasswordCallback`

TextInputCallback

Used to retrieve text input from the end user. For example

```
"callbacks":[
  {
    "type":"TextInputCallback",
    "output":[
      {
        "name":"prompt",
        "value":"User Name"
      }
    ],
    "input":[
      {
        "name":"IDToken1",
        "value":""
      }
    ]
  }
]
```

Class to import: `javax.security.auth.callback.TextInputCallback`

Read-only Callbacks

HiddenValueCallback

Used to return form values that are not visually rendered to the end user. For example:

```
"callbacks":[
  {
    "type":"HiddenValueCallback",
    "output":[
      {
        "name":"value",
        "value":"6186c911-b3be-4dbc-8192-bdf251392072"
      },
      {
        "name":"id",
        "value":"jwt"
      }
    ],
    "input":[
      {
        "name":"IDToken1",
        "value":"jwt"
      }
    ]
  }
]
```

Class to import: `com.sun.identity.authentication.callbacks.HiddenValueCallback`

MetadataCallback

Used to inject key-value meta data into the authentication process. For example:

```
"callbacks":[
  {
    "type":"MetadataCallback",
    "output":[
      {
        "name":"data",
        "value":{"
          "myParameter": "MyValue"
        }
      }
    ]
  }
]
```

Class to import: `com.sun.identity.authentication.spi.MetadataCallback`

PollingWaitCallback

Tells the user the amount of time to wait before responding to the callback.


```
"callbacks":[
  {
    "type":"PollingWaitCallback",
    "output":[
      {
        "name":"waitTime",
        "value":"8000"
      },
      {
        "name":"message",
        "value":"Waiting for response..."
      }
    ]
  }
]
```

Class to import: `org.forgerock.openam.authentication.callbacks.PollingWaitCallback`

RedirectCallback

Used to redirect the user's browser or user-agent.

```
"callbacks":[
  {
    "type":"RedirectCallback",
    "output":[
      {
        "name":"redirectUrl",
        "value":"https://accounts.google.com/o/oauth2/v2/auth?nonce..."
      },
      {
        "name":"redirectMethod",
        "value":"GET"
      },
      {
        "name":"trackingCookie",
        "value":true
      }
    ]
  }
]
```

Class to import: `com.sun.identity.authentication.spi.RedirectCallback`

TextOutputCallback

Used to display a message to the end user.

```
"callbacks":[
  {
    "type":"TextOutputCallback",
    "output":[
      {
        "name":"message",
        "value":"Default message"
      },
      {
        "name":"messageType",
        "value":""
      }
    ]
  }
]
```

Class to import: `javax.security.auth.callback.TextOutputCallback`

Backchannel Callbacks

AM uses backchannel callbacks when it needs to recover additional information from the user's request. For example, when it requires a particular header or a certificate.

HttpCallback

Used to access user credentials sent in the Authorization header. For example:

```
Authorization: Basic bXlDbGllbnQ6Zm9yZ2Vyb2Nr
```

Class to import: `com.sun.identity.authentication.spi.HttpCallback`

LanguageCallback

Used to retrieve the locale for localizing text presented to the end user. The locale is sent in the request as a header.

Class to import: `javax.security.auth.callback.LanguageCallback`

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.

Class to import: `com.sun.identity.authentication.callbacks.ScriptTextOutputCallback`

X509CertificateCallback

Used to retrieve the content of an x.509 certificate, for example, from a header.

Class to import: `com.sun.identity.authentication.spi.X509CertificateCallback`

2.7.2. Logging out of AM Using REST

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..*" \
--header "Accept-API-Version: resource=3.1, protocol=1.0" \
https://openam.example.com:8443/openam/json/realms/root/sessions/?_action=logout
{
  "result": "Successfully logged out"
}
```

2.7.3. Invalidating All Sessions for a Given User

To log out all sessions for a given user, first obtain a list of session handles of their active sessions, by performing an HTTP GET to the `/json/sessions/` endpoint, using the SSO token of an administrative user, such as `amAdmin` as the value of the `iPlanetDirectoryPro` header. You must also specify a `queryFilter` parameter.

The `queryFilter` parameter requires the name of the user, and the realm to search. For example, to obtain a list of session handles for a user named `demo` in the top-level realm, the query filter value would be:

```
username eq "demo" and realm eq "/"
```

Note

The query filter value must be URL encoded when sent over HTTP.

For more information on query filter parameters, see "Query" in the *Authentication and Single Sign-On Guide*.

In the following example, there are two active sessions:

```
$ curl \
--request GET \
--header "Content-Type: application/json" \
--header "Cache-Control: no-cache" \
--header "iPlanetDirectoryPro: AQICS...NzEz*" \
--header "Accept-API-Version: resource=3.1, protocol=1.0" \
https://openam.example.com:8443/openam/json/realms/root/sessions?_queryFilter=username%20eq%20demo%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:SJ80.*AA....JT.*",
    }
  ]
}
```

```
{
  "latestAccessTime": "2018-10-23T09:37:54.387Z",
  "maxIdleExpirationTime": "2018-10-23T10:07:54Z",
  "maxSessionExpirationTime": "2018-10-23T11:37:54Z"
},
{
  "username": "demo",
  "universalId": "id=demo,ou=user,dc=openam,dc=forgerock,dc=org",
  "realm": "/",
  "sessionHandle": "shandle:H4CV.*DV...FM.*",
  "latestAccessTime": "2018-10-23T09:37:43.780Z",
  "maxIdleExpirationTime": "2018-10-23T10:07:43Z",
  "maxSessionExpirationTime": "2018-10-23T11:37:43Z"
}
],
"resultCount": 2,
"pagedResultsCookie": null,
"totalPagedResultsPolicy": "NONE",
"totalPagedResults": -1,
"remainingPagedResults": -1
}
```

To log out all sessions for the specific user, perform an HTTP POST to the `/json/sessions/` endpoint, using the SSO token of an administrative user, such as `amAdmin` as the value of the `iPlanetDirectoryPro` header. You must also specify the `logoutByHandle` action, and include an array of the session handles to invalidate in the POST body, in a property named `sessionHandles`, as shown below:

```
$ curl \
--request POST \
--header "Content-Type: application/json" \
--header "Cache-Control: no-cache" \
--header "iPlanetDirectoryPro: AQIC5w...NTcy*" \
--header "Accept-API-Version: resource=3.1, protocol=1.0" \
--data '{
  "sessionHandles": [
    "shandle:SJ80.*AA...JT.*",
    "shandle:H4CV.*DV...FM.*"
  ]
}' \
https://openam.example.com:8443/openam/json/realms/root/sessions/?_action=logoutByHandle
{
  "result": {
    "shandle:SJ80.*AA...JT.*": true,
    "shandle:H4CV.*DV...FM.*": true
  }
}
```

2.7.4. 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 "Handling HTTP Request Headers" in the *Installation Guide*.

2.7.5. 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 "Accept-API-Version: resource=2.0, protocol=1.0" \
--header "X-OpenAM-Username: demo" \
--header "X-OpenAM-Password: changeit" \
--header "Authorization: Basic ZGVtbzpjagFuZ2VpdA==" \
'https://openam.example.com:8443/openam/json/realms/root/authenticate'
{
  "tokenId": "AQIC5w...NTcy*",
  "successUrl": "/openam/console",
  "realm": "/"
}
```

2.8. 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" \
--header "Accept-API-Version: resource=2.0, protocol=1.0" \
'https://openam.example.com:8443/openam/json/realms/root/authenticate'
{
  "tokenId": "AQIC5wM...TU30Q*",
  "successUrl": "/openam/console",
  "realm": "/"
}
```

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*" \
--header "Accept-API-Version: resource=2.0, protocol=1.0" \
--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 ["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 ["Authentication and Logout using REST"](#). 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.

2.9. Server Information

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

```
$ curl \
--request GET \
--header "Content-Type: application/json" \
--header "Accept-API-Version: resource=1.1, protocol=1.0" \
https://openam.example.com:8443/openam/json/serverinfo/*
{
  "domains": [
    ".example.com"
  ],
  "protectedUserAttributes": [],
  "cookieName": "iPlanetDirectoryPro",
  "secureCookie": false,
  "forgotPassword": "false",
  "forgotUsername": "false",
  "kbaEnabled": "false",
  "selfRegistration": "false",
  "lang": "en-US",
  "successfulUserRegistrationDestination": "default",
  "socialImplementations": [
    {
      "iconPath": "XUI/images/logos/facebook.png",
      "authnChain": "FacebookSocialAuthenticationService",
    }
  ]
}
```

```

        "displayName": "Facebook",
        "valid": true
    }
},
"referralsEnabled": "false",
"zeroPageLogin": {
    "enabled": false,
    "referrerWhitelist": [
        ""
    ],
    "allowedWithoutReferer": true
},
"realm": "/",
"xuiUserSessionValidationEnabled": true,
"FQDN": "openam.example.com"
}

```

2.10. Token Encoding

Valid tokens in AM require 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:

```
AQIC5wM2LY4SfczntBbXvEA0uECbqMY3J4NW3byH6xwgkGE=@AAJTSQACMDE=#
```

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

- `+` is replaced with `-`
- `/` is replaced with `_`
- `=` is replaced with `.`
- `@` is replaced with `*`
- `#` is replaced with `*`
- `*` (first instance) is replaced with `@`
- `*` (subsequent instances) is replaced with `#`

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

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

2.11. Logging

AM 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.

2.11.1. 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 "Audit Log Topics" in the *Setup and Maintenance Guide*.

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

2.11.2. 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.

2.12. REST Goto URL Validation

You can set valid goto URLs using the AM console by following the instructions in ["Configuring Success and Failure Redirection URLs"](#) 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 "Accept-API-Version: protocol=2.1,resource=3.0" \
--header "Content-Type: application/json" \
--header "iPlanetDirectoryPro: AQIC5...ACMDE.*" \
--data '{"goto":"http://www.example.com/"}' \
https://openam.example.com:8443/openam/json/users?action=validateGoto
{
  "successURL":"http://www.example.com/"
}
```

2.13. Reference

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

2.13.1. REST APIs

amster service name: `RestApis`

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**

amster 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`

amster attribute: `warningHeader`

API Descriptions

Whether API Explorer and API Docs are enabled in AM 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 AM was built. Note that dynamic documentation generation may not work in some application containers.

The possible values for this property are:

- `DYNAMIC`. Enabled with Dynamic Documentation
- `STATIC`. Enabled with Static Documentation
- `DISABLED`

Default value: `STATIC`

amster 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`

amster attribute: `defaultProtocolVersion`

Enable CSRF Protection

If enabled, all non-read/query requests will require the X-Requested-With header to be present.

Requiring a non-standard header ensures requests can only be made via methods (XHR) that have stricter same-origin policy protections in Web browsers, preventing Cross-Site Request Forgery (CSRF) attacks. Without this filter, cross-origin requests are prevented by the use of the application/json Content-Type header, which is less robust.

Default value: `true`

amster attribute: `csrfFilterEnabled`

Chapter 3

Developing with Scripts

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

3.1. The Scripting Environment

AM supports scripts written in either JavaScript, or Groovy ¹, and the same variables and bindings are delivered to scripts of either language.

You can use a script to check the version of the JavaScript engine AM is using. You could temporarily add the following script to a Scripted Decision node, for example, to output the engine version to the debug log:

```
var rhino = JavaImporter(
    org.mozilla.javascript.Context
)

var currentContext = rhino.Context.getCurrentContext()
var rhinoVersion = currentContext.getImplementationVersion()

logger.error("JS Script Engine: " + rhinoVersion)

outcome = "true"
```

Note

Ensure the following are listed in the Java class whitelist property of the scripting engine.

- `org.mozilla.javascript.Context`
- `org.forgerock.openam.scripting.timeouts.*`

To view the Java class whitelist, go to [Configure > Global Services > Scripting > Secondary Configurations](#). Select the script type, and on the Secondary Configurations tab, click `engineConfiguration`.

For information on the capabilities of the JavaScript engine AM uses, see [Mozilla Rhino](#).

You can use a script to check the version of the Groovy scripting engine AM is using. You could temporarily add the following script to a Scripted Decision node, for example, to output the engine version to the debug log:

¹Scripts used for client-side authentication must be in written in JavaScript.

```
logger.error("Groovy Script Engine: " + GroovySystem.version)

outcome = "true"
```

Note

Ensure the following are listed in the Java class whitelist property of the scripting engine.

- `groovy.lang.GroovySystem`

To view the Java class whitelist, go to [Configure > Global Services > Scripting > Secondary Configurations](#). Select the script type, and on the Secondary Configurations tab, click `engineConfiguration`.

For information on the capabilities of the Groovy engine AM uses, see [Apache Groovy](#).

To access the functionality AM provides, import the required Java class or package, as follows:

JavaScript

```
var fr = JavaImporter(
    org.forgerock.openam.auth.node.api,
    javax.security.auth.callback.NameCallback
);
with (fr) {
    ...
}
```

Groovy

```
import org.forgerock.openam.auth.node.api.*;
import javax.security.auth.callback.NameCallback;
```

You may need to whitelist the classes you use in scripts. See ["Security"](#).

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 within a chain and are executed on the server during authentication.

Authentication Trees

Scripts are included in an authentication node within a tree and are executed on the server during authentication.

Policy Condition

Scripts used as conditions within policies.

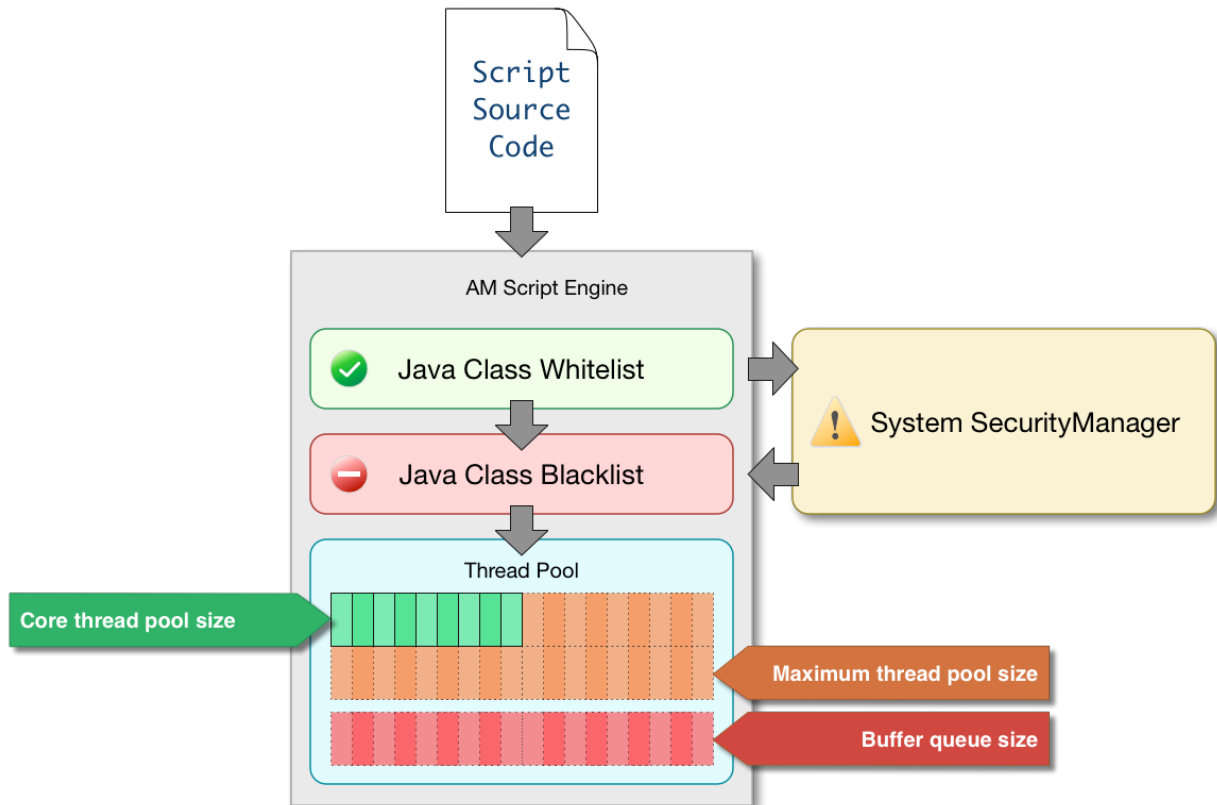
OIDC Claims

Scripts that gather and populate the claims in a request when issuing an ID token or making a request to the `userinfo` endpoint.

For information on the global API, available to all script types, see "Global Scripting API Functionality".

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.



3.1.1. 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 "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.

3.1.2. 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 "Scripting".

3.2. 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.

3.2.1. Managing Scripts With the AM Console

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

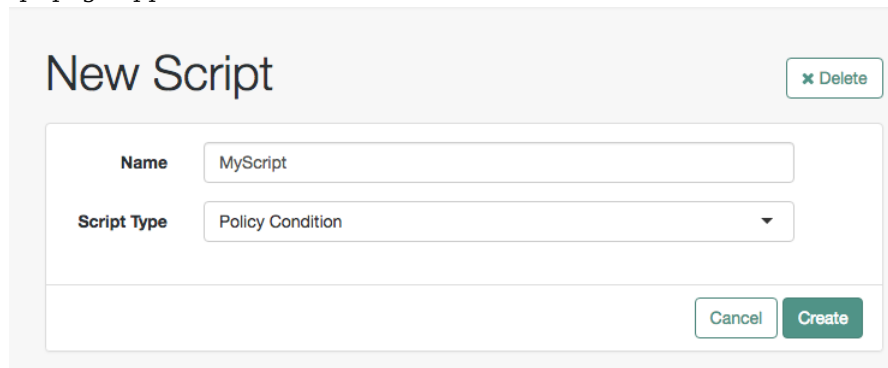
- "To Create Scripts by Using the AM Console"
- "To Modify Scripts by Using the AM Console"
- "To Delete Scripts by Using the AM Console"

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:



The screenshot shows a web interface for creating a new script. The main heading is "New Script". In the top right corner, there is a button labeled "Delete" with a small 'x' icon. Below the heading is a form containing two input fields. The first field is labeled "Name" and contains the text "MyScript". The second field is labeled "Script Type" and is a dropdown menu currently showing "Policy Condition". At the bottom right of the form, there are two buttons: "Cancel" and "Create".

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

Delete

Name

MyScript

Description

Script Type

Policy Condition

Change

Language

☒ JavaScript
 ☐ Groovy

Script

```

1  /**
2   * This is a Policy Condition example script. It demon
3   * use that information in external HTTP calls and mak
4   */
5
6   var userAddress, userIP, resourceHost;
7
8   if (validateAndInitializeParameters()) {
9
10      var countryFromUserAddress = getCountryFromUserAdd
11      logger.message("Country retrieved from user's addr
12      var countryFromUserIP = getCountryFromUserIP();
13      logger.message("Country retrieved from user's IP:
14      var countryFromResourceURI = getCountryFromResourc
15      logger.message("Country retrieved from resource UR
16
17      if (countryFromUserAddress === countryFromUserIP &
18          logger.message("Authorization Succeeded");
19          responseAttributes.put("countryOfOrigin", {cou
20          authorized = true;
21      } else {

```

Upload

Validate

Edit Fullscreen

Save Changes

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

- Enter a description of the script.
- Choose the script language, either JavaScript or Groovy. Note that not every script type supports both languages.
- 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.

- 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.

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.

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.

3.2.2. 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:

- "Querying Scripts"
- "Reading a Script"

- "Validating a Script"
- "Creating a Script"
- "Updating a Script"
- "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:

```
dmFyIGVgPSAxMjM7IA0KdmFyIGIgPSA0NTY7
```

Language

The language the script is written in - [JAVASCRIPT](#) or [GROOVY](#).

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
AUTHENTICATION_TREE_DECISION_NODE	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

AUTHENTICATION_TREE_DECISION_NODE

Authentication scripts used by Scripted Tree Decision authentication nodes.

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`.

3.2.2.1. Querying Scripts

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

Note

If the realm is not specified in the URL, 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..." \
--header "Accept-API-Version: resource=1.1" \
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": "LyqCiAqIFRoXMg...",
    "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
}

```

Supported `_queryFilter` Fields and Operators

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

3.2.2.2. Reading a Script

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

Tip

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

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


```
$ curl \
--header "iPlanetDirectoryPro: AQIC5..." \
--header "Accept-API-Version: resource=1.1" \
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": "LyqCiAqIFRoaxMg...",
  "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
}
```

3.2.2.3. Validating a Script

To validate a script, perform an HTTP POST using the `/json{/realm}/scripts` endpoint, with an `_action` parameter set to `validate`. Include a JSON representation of the script and the script language, `JAVASCRIPT` or `GR00VY`, 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..." \
--header "Accept-API-Version: resource=1.1" \
--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..." \
--header "Accept-API-Version: resource=1.1" \
--data '{
  "script": "dmFyIGEGPSAxMjM7dmFyIGIgPSA0NTY7ID1WQUxJREFUSU90IFNIT1VMRCBGQUIMPQo=",
  "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"
    }
  ]
}
```

3.2.2.4. Creating a Script

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

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

Note

If the realm is not specified in the URL, 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..." \
--header "Accept-API-Version: resource=1.1" \
--data '{
  "name": "MyJavaScript",
  "script": "dmFyIGVPSAxBmM7CnZhciBiID0gNDU2Ow==",
  "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": "dmFyIGVPSAxBmM7CnZhciBiID0gNDU2Ow==",
  "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
}
```

3.2.2.5. Updating a Script

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

Note

If the realm is not specified in the URL, 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" \
--header "Accept-API-Version: resource=1.1" \
--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
}
```

3.2.2.6. Deleting a Script

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

Note

If the realm is not specified in the URL, 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..." \
--header "Accept-API-Version: resource=1.1" \
https://openam.example.com:8443/openam/json/realms/root/realms/myrealm/scripts/0168d494-015a-420f-ae5a-6a2a5c1126af
{}
```

3.2.3. 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, for example `/path/to/myScriptConfigurationFile.txt`, containing the following:

```
script-file=/path/to/myScriptFile.js
language=JAVASCRIPT ❶
name=My New Script
context=AUTHENTICATION_SERVER_SIDE ❷
```

- ❶ Possible values for the `language` property are:

- JAVASCRIPT
- GROOVY

- ❷ Possible values for the `context` property are:

- POLICY_CONDITION
- AUTHENTICATION_SERVER_SIDE
- AUTHENTICATION_CLIENT_SIDE
- OIDC_CLAIMS
- AUTHENTICATION_TREE_DECISION_NODE

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 myScriptID \
  --datafile /path/to/myScriptConfigurationFile.txt
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/myScriptID  
createdBy=  
lastModifiedDate=  
lastModifiedBy=  
name=My New Script  
context=AUTHENTICATION_SERVER_SIDE  
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/myScriptID  
Sub Configuration scriptConfigurations/myScriptID was deleted from realm /myRealm
```

3.3. 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

3.3.1. 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 ["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.

HTTP Client Response Methods

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

Method	Parameters	Return Type	Description
<code>HttpClientResponse.hasHeaders</code>	<code>Void</code>	<code>Boolean</code>	Indicate whether the returned response had any headers.

3.3.2. 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 "Debug Logging By Service" in the *Setup and Maintenance Guide*.

The following table lists the `logger` methods.

Logger Methods

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

3.4. Authentication API Functionality

This section covers the available functionality when Scripting authentication modules use client-side and server-side authentication script types.

Tip

When developing server-side scripts, it can be useful to increase the debug level of the `org.apache.http.wire` and `org.apache.http.headers` appenders to `Message`.

By default, these appenders are always set to the **Warning** level unless logging is disabled. For more information, see the [org.forgerock.allow.http.client.debug](#) advanced server property.

Authentication API functionality includes:

- "Accessing Authentication State"
- "Accessing Profile Data"
- "Accessing Client-Side Script Output Data"
- "Accessing Request Data"
- "Redirecting the User After Authentication Failure"

3.4.1. Accessing Authentication State

AM passes **authState** and **sharedState** objects to server-side scripts in order for the scripts to access authentication state.

Server-side scripts can access the current authentication state through the **authState** object.

The **authState** value is **SUCCESS** if the authentication is currently successful, or **FAILED** if authentication has failed. Server-side scripts must set a value for **authState** before completing.

If an earlier authentication module in the authentication chain has set the login name of the user, server-side scripts can access the login name through **username**.

The following authentication modules set the login name of the user:

- Anonymous
- Certificate
- Data Store
- Federation
- HTTP Basic
- JDBC
- LDAP
- Membership
- RADIUS
- SecurID
- Windows Desktop SSO

3.4.2. Accessing Profile Data

Server-side authentication scripts can access profile data through the methods of the `idRepository` object.

Profile Data Methods

Method	Parameters	Return Type	Description
<code>idRepository.getAttribute</code>	<code>User Name</code> (type: <code>String</code>) <code>Attribute Name</code> (type: <code>String</code>)	<code>Set</code>	Return the values of the named attribute for the named user.
<code>idRepository.setAttribute</code>	<code>User Name</code> (type: <code>String</code>) <code>Attribute Name</code> (type: <code>String</code>) <code>Attribute Values</code> (type: <code>Array</code>)	<code>Void</code>	Set the named attribute as specified by the attribute value for the named user, and persist the result in the user's profile.
<code>idRepository.addAttribute</code>	<code>User Name</code> (type: <code>String</code>) <code>Attribute Name</code> (type: <code>String</code>) <code>Attribute Value</code> (type: <code>String</code>)	<code>Void</code>	Add an attribute value to the list of attribute values associated with the attribute name for a particular user.

3.4.3. Accessing Client-Side Script Output Data

Client-side scripts add data they gather into a string object named `clientScriptOutputData`. Client-side scripts then cause the user-agent automatically to return the data to AM by HTTP POST of a self-submitting form.

3.4.4. Accessing Request Data

Server-side scripts can get access to the login request by using the methods of the `requestData` object.

The following table lists the methods of the `requestData` object. Note that this object differs from the client-side `requestData` object and contains information about the original authentication request made by the user.

Request Data Methods

Method	Parameters	Return Type	Description
<code>requestData.getHeader</code>	<code>Header Name</code> (type: <code>String</code>)	<code>String</code>	Return the <code>String</code> value of the named request header, or <code>null</code> if parameter is not set.
<code>requestData.getHeaders</code>	<code>Header Name</code> (type: <code>String</code>)	<code>String[]</code>	Return the array of <code>String</code> values of the named request header, or <code>null</code> if parameter is not set.

Method	Parameters	Return Type	Description
<code>requestData.getParameter</code>	<i>Parameter Name</i> (type: <code>String</code>)	<code>String</code>	Return the <code>String</code> value of the named request parameter, or <code>null</code> if parameter is not set.
<code>requestData.getParameters</code>	<i>Parameter Name</i> (type: <code>String</code>)	<code>String[]</code>	Return the array of <code>String</code> values of the named request parameter, or <code>null</code> if parameter is not set.

3.4.5. Redirecting the User After Authentication Failure

Server-side scripts can redirect the user to a specific URL in case of authentication failure by adding a `gotoOnFailureUrl` property to the chain's shared state.

When the script reaches a `FAILED` authentication state (defined by the `authState` variable), it checks if the `gotoOnFailureUrl` property is stored in the shared state. If so, the script redirects the user to the specified URL.

You can redirect the user to a page relative to AM's URL, or to an absolute URL:

Relative URL

```
...
sharedState.put("gotoOnFailureUrl", "/openam/XUI/?service=testChain#failedLogin");
authState = FAILED;
...
```

Absolute URL

```
...
sharedState.put("gotoOnFailureUrl", "http://www.example.com");
authState = FAILED;
...
```

Note that the failure URL relative to AM's domain includes the authentication service; this is so that when the user clicks on the link to log in again, AM constructs the login page with the appropriate service instead of with the default one for the realm.

When redirecting the user to an absolute URL different from AM's scheme, FQDN, and port, you must configure the URL in the Validation Service of the realm. Otherwise, AM will ignore the redirection. For more information, see ["To Configure the Validation Service"](#) in the *Authentication and Single Sign-On Guide*.

3.5. Scripted Decision Node API Functionality

This section covers the functionality provided by the Decision node script for authentication trees script type.

Scripted Decision Node API functionality includes:

- Accessing Request Header Data
- Accessing Shared State Data
- Encrypting and Decrypting Shared State Data
- Accessing Existing Session Data
- Using Callbacks

The node also has access to the functionality provided in the [global scripting API](#), for example:

- Accessing HTTP Services
- Debug Logging

3.5.1. Accessing Request Header Data

Scripted Decision Node scripts can access the headers provided by the login request by using the methods of the `requestHeaders` object.

Note that the script has access to a copy of the headers. Changing their values does not affect the request itself.

The following table lists the methods of the `requestHeaders` object:

Request Headers Methods

Method	Parameters	Return Type	Description
<code>requestHeaders.getHeaderName</code>	<code>Header Name</code> (type: <code>String</code>)	<code>String[]</code>	<p>Return the array of string values of the named request header, or <code>null</code> if the property is not set. Note that header names are case-sensitive.</p> <p>For example:</p> <pre>var headerName = "user-agent"; if (requestHeaders.getHeaderName(headerName).get(0).indexOf("Chrome") != -1) { outcome = "true"; } else { outcome = "false"; }</pre>

3.5.2. Accessing Shared State Data

Scripted Decision Node scripts can get access to the shared state within the tree by using the `sharedState` and `transientState` objects.

The following table lists the available methods:

Shared State Methods

Method	Parameters	Return Type	Description
<code>sharedState.get</code>	<i>Property Name</i> (type: <code>String</code>)	<code>String</code>	<p>Return the string value of the named shared state property, or <code>null</code> if the property is not set. Note that property names are case-sensitive.</p> <p>For example, use the following code to get the current authentication level:</p> <pre>var currentAuthLevel = sharedState.get("authLevel");</pre>
<code>transientState.get</code>	<i>Property Name</i> (type: <code>String</code>)	<code>String</code>	<p>Return the string value of the named transient state property, or <code>null</code> if the property is not set. Note that property names are case-sensitive.</p> <p>For example, use the following code to get the value of a previously supplied password:</p> <pre>var givenPassword = transientState.get("password");</pre>

3.5.3. Encrypting and Decrypting Shared State Data

(Added in AM 6.5.4) Scripted Decision Node scripts can encrypt and decrypt data from the shared state using the `sharedStateCrypto` object.

Important

Use of the `sharedStateCrypto` object is intended to encrypt and decrypt one-time passwords along with their timestamps in the *Authentication and Single Sign-On Guide* only.

The `org.forgerock.openam.auth.nodes.crypto.NodeSharedStateCrypto` Java class included in AM 6.5.4 or later and its `sharedStateCrypto` scripting binding do not exist in AM 7 or later because `org.forgerock.openam.auth.nodes.crypto.NodeSharedStateCrypto` no longer exists.

Any scripts or authentication nodes using this class and/or binding will need to be updated accordingly when upgrading to AM 7 or later because `org.forgerock.openam.auth.nodes.crypto.NodeSharedStateCrypto` no longer exists.

The following table lists the available methods:

Encrypting and Decrypting Methods

Method	Parameters	Return Type	Description
<code>sharedStateCrypto</code>	<i>Property Name</i> (type: <code>String</code>)	<code>JSON</code>	Decrypts a string value and returns a JSON, or <code>null</code> if the string payload is not set. If something is missing or is misconfigured in terms of the secret, a secret-

Method	Parameters	Return Type	Description
			<p>related exception is thrown. Note that property names are case-sensitive.</p> <p>For example, use the following code to decrypt the string containing the one-time password, and to assign the decrypted password's to a variable:</p> <pre>var otpEncrypted = sharedState.get("oneTimePasswordEncrypted"); var otpJsonValue = sharedStateCrypto.decrypt(otpEncrypted); var otp = otpJsonValue.get("oneTimePassword").asString();</pre>
<code>sharedStateCrypto</code>	<code>Property Name</code> (type: <code>JSON</code>)	<code>String</code>	<p>Encrypts a JSON object.</p> <p>For example, use the following code to encrypt the JSON containing the one-time password:</p> <pre>var otpJson = org.forgerock.json.JsonValue.json(org.forgerock.json.JsonVal org.forgerock.json.JsonValue.field("oneTimePassword", "19219283"), org.forgerock.json.JsonValue.field("oneTimePasswordTimestamp", 1623328298))); var otpEncrypted = sharedStateCrypto.encrypt(otpJson); sharedState.put("oneTimePasswordEncrypted", otpEncrypted);</pre>

3.5.4. Accessing Existing Session Data

Scripted Decision Node scripts can access any existing session information provided during a session upgrade request by using the `existingSession` object.

The following table lists the methods of the `existingSession` object:

Existing Session Methods

Method	Parameters	Return Type	Description
<code>existingSession.get</code>	<code>Property Name</code> (type: <code>String</code>)	<code>String</code>	Return the string value of the named existing session property, or <code>null</code> if the property is not set. Note that property names are case-sensitive.

Method	Parameters	Return Type	Description
			<p>Warning</p> <p>If the current request is not a session upgrade and does not provide an existing session, the <code>existingSession</code> variable is not declared. Check for a declaration before attempting to access the variable.</p> <p>For example, use the following code to get the authentication level of the existing session:</p> <pre>if (typeof existingSession !== 'undefined') { existingAuthLevel = existingSession.get("AuthLevel"); } else { logger.error("Variable existingSession not declared - not a session upgrade."); }</pre>

3.5.5. Using Callbacks

The scripted decision node can use callbacks to provide or request additional information during the authentication process.

For example, the following scripts use the `NameCallback` callback to request a "Nickname" value from the user, and adds the returned value to the `sharedState` map for use elsewhere in the authentication tree:

Groovy

```
import org.forgerock.openam.auth.node.api.*;
import javax.security.auth.callback.NameCallback;

if (callbacks.isEmpty()) {
    action = Action.send(new NameCallback("Enter Your Nickname")).build();
} else {
    sharedState = sharedState.put("Nickname", callbacks.get(0).getName());
    action = Action.goTo("true").build();
}
```

JavaScript

```
var fr = JavaImporter(
    org.forgerock.openam.auth.node.api,
    javax.security.auth.callback.NameCallback
);
with (fr) {
    if (callbacks.isEmpty()) {
        action = Action.send(new NameCallback("Enter Your Nickname")).build();
    } else {
        sharedState.put("Nickname", callbacks.get(0).getName());
        action = Action.goTo("true").build();
    }
}
```

For a list of supported callbacks, see ["Supported Callbacks"](#).

3.5.6. OAuth 2.0 Access Token Modification Scripting API

The following properties are available to scripts:

clientProperties

A map of properties configured in the relevant client profile. Only present if the client was correctly identified.

The keys in the map are as follows:

clientId

The URI of the client.

allowedGrantTypes

The list of the allowed grant types (`org.forgerock.oauth2.core.GrantType`) for the client.

allowedResponseTypes

The list of the allowed response types for the client.

allowedScopes

The list of the allowed scopes for the client.

customProperties

A map of any custom properties added to the client.

Lists or maps are included as sub-maps. For example, a custom property of `customMap[Key1]=Value1` is returned as `customMap > Key1 > Value1`.

To add custom properties to a client, go to OAuth 2.0 > Clients > *Client ID* > Advanced, and then update the Custom Properties field.

requestProperties

A map of the properties present in the request. Always present.

The keys in the map are as follows:

requestUri

The URI of the request.

realm

The realm to which the request was made.

requestParams

The request parameters, and/or posted data. Each value in this map is a list of one, or more, properties.

Important

To mitigate the risk of reflection type attacks, use OWASP best practices when handling these properties. For example, see [Unsafe use of Reflection](#).

scopes

Contains a set of the requested scopes. For example:

```
[  
  "profile",  
  "friends"  
]
```

scriptName

The display name of the script. Always present.

3.6. Authorization API Functionality

This section covers functionality available when scripting authorization using the policy condition script context type.

3.6.1. Accessing Authorization State

Server-side scripts can access the current authorization state through the following objects:

Authorization State Objects

Object	Type	Description
<code>authorized</code>	<code>Boolean</code>	Return <code>true</code> if the authorization is currently successful, or <code>false</code> if authorization has failed. Server-side scripts must set a value for <code>authorized</code> before completing.
<code>environment</code>	<code>Map<String, Set<String>></code>	<p>Describe the environment passed from the client making the authorization request.</p> <p>For example, the following shows a simple <code>environment</code> map with a single entry:</p> <pre>"environment": { "IP": ["127.0.0.1"] }</pre>
<code>resourceURI</code>	<code>String</code>	Specify the URI of the resource to which authorization is being requested.
<code>username</code>	<code>String</code>	Specify the user ID of the subject that is requesting authorization.

3.6.2. Accessing Profile Data

Server-side authorization scripts can access profile data of the subject of the authorization request through the methods of the `identity` object.

Note

To access the profile data of the subject, they must be logged in and their SSO token must be available.

Authorization Script Profile Data Methods

Method	Parameters	Return Type	Description
<code>identity.getAttribute</code>	<i>Attribute Name</i> (type: <code>String</code>)	<code>Set</code>	Return the values of the named attribute for the subject of the authorization request.
<code>identity.setAttribute</code>	<i>Attribute Name</i> (type: <code>String</code>) <i>Attribute Values</i> (type: <code>Array</code>)	<code>Void</code>	Set the named attribute to the values specified by the attribute value for the subject of the authorization request.
<code>identity.addAttribute</code>	<i>Attribute Name</i> (type: <code>String</code>) <i>Attribute Value</i> (type: <code>String</code>)	<code>Void</code>	Add an attribute value to the list of attribute values associated with the attribute name for the subject of the authorization request.

Method	Parameters	Return Type	Description
<code>identity.store</code>	None	<code>Void</code>	Commit any changes to the identity repository. Caution You must call <code>store()</code> otherwise changes will be lost when the script completes.

3.6.3. Accessing Session Data

Server-side authorization scripts can access session data of the subject of the authorization request through the methods of the `session` object.

Note

To access the session data of the subject, they must be logged in and their SSO token must be available.

Authorization Script Session Methods

Method	Parameters	Return Type	Description
<code>session.getProperty</code>	<i>Property Name</i> (type: String)	<code>String</code>	Retrieve properties from the session associated with the subject of the authorization request. See the table below for example properties and their values.

The following table demonstrates some of the session properties available to the `session.getProperty()` method, and example values:

Get Session Data Example Keys and Values

Key	Sample value
<code>AMCtxId</code>	<code>e370cca2-02d6-41f9-a244-2b107206bd2a-122934</code>
<code>amlbcookie</code>	<code>01</code>
<code>authInstant</code>	<code>2018-04-04T09:19:05Z</code>
<code>AuthLevel</code>	<code>0</code>
<code>CharSet</code>	<code>UTF-8</code>
<code>clientType</code>	<code>genericHTML</code>

Key	Sample value
FullLoginURL	/openam/UI/Login?realm=%2F
Host	198.51.100.1
HostName	openam.example.com
Locale	en_US
Organization	dc=openam,dc=forgerock,dc=org
Principal	id=amadmin,ou=user,dc=openam,dc=forgerock,dc=org
Principals	amadmin
Service	ldapService
successURL	/openam/console
sun.am.UniversalIdentifier	id=amadmin,ou=user,dc=openam,dc=forgerock,dc=org
UserId	amadmin
UserProfile	Required
UserToken	amadmin
webhooks	myWebHook

3.6.4. Setting Authorization Responses

Server-side authorization scripts can return information in the response to an authorization request.

Authorization Script Response Methods

Method	Parameters	Return Type	Description
<code>responseAttributes.put</code>	<i>Attribute Name</i> (type: String) <i>Attribute Values</i> (type: Array)	Void	Add an attribute to the response to the authorization request.
<code>advice.put</code>	<i>Advice Key</i> (type: String) <i>Advice Values</i> (type: Array)	Void	Add advice key-value pairs to the response to a failing authorization request.
<code>tll</code>	<i>TTL Value</i> (type: Integer)	Void	Add a time-to-live value, which is a timestamp in milliseconds to the response to a successful authorization. After the time-to-live value

Method	Parameters	Return Type	Description
			<p>the decision is no longer valid.</p> <p>If no value is set, TTL Value defaults to <code>Long.MAX_VALUE</code> (9223372036854775807), which means the decision has no timeout, and can live for as long as the calling client holds on to it. In the case of policy enforcement points, they hold onto the decision for their configured cache timeout.</p>

3.7. OpenID Connect 1.0 Claims API Functionality

This section covers functionality available when scripting OIDC claim handling using the OIDC claims script context type.

Server-side scripts can access the OpenID Connect request through the following objects:

`claims`

Contains a map of the claims the server provides by default. For example:

```
{
  "sub": "248289761001",
  "updated_at": "1450368765"
}
```

`clientProperties`

A map of properties configured in the relevant client profile. Only present if the client was correctly identified.

The keys in the map are as follows:

`clientId`

The URI of the client.

`allowedGrantTypes`

The list of the allowed grant types (`org.forgerock.oauth2.core.GrantType`) for the client.

`allowedResponseTypes`

The list of the allowed response types for the client.

`allowedScopes`

The list of the allowed scopes for the client.

`customProperties`

A map of any custom properties added to the client.

Lists or maps are included as sub-maps. For example, a custom property of `customMap[Key1]=Value1` is returned as `customMap > Key1 > Value1`.

To add custom properties to a client, go to OAuth 2.0 > Clients > *Client ID* > Advanced, and then update the Custom Properties field.

`identity`

Contains a representation of the identity of the resource owner.

For more details, see the `com.sun.identity.idm.AMIdentity` class in the ForgeRock Access Management Javadoc.

`requestedClaims`

Contains requested claims if the `claims` query parameter is used in the request, and Enable "claims_parameter_supported" is checked in the OAuth 2.0 provider service configuration; otherwise, this property is empty.

For more information see "Requesting Claims using the "claims" Request Parameter" in the *OpenID Connect Core 1.0* specification.

Example:

```
{
  "given_name": {
    "essential": true,
    "values": [
      "Demo User",
      "D User"
    ]
  },
  "nickname": null,
  "email": {
    "essential": true
  }
}
```

`requestProperties`

A map of the properties present in the request. Always present.

The keys in the map are as follows:

`requestUri`

The URI of the request.

`realm`

The realm to which the request was made.

`requestParams`

The request parameters, and/or posted data. Each value in this map is a list of one, or more, properties.

Important

To mitigate the risk of reflection type attacks, use OWASP best practices when handling these properties. For example, see [Unsafe use of Reflection](#).

`scopes`

Contains a set of the requested scopes. For example:

```
[  
  "profile",  
  "openid"  
]
```

`scriptName`

The display name of the script. Always present.

`session`

Contains a representation of the user's session object if the request contained a session cookie.

For more details, see the `com.iplanet.sso.SSOToken` class in the ForgeRock Access Management Javadoc.

Chapter 4

Reference

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

4.1. Scripting

amster service name: `Scripting`

4.1.1. 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`. Policy Condition
- `AUTHENTICATION_SERVER_SIDE`. Server-side Authentication
- `AUTHENTICATION_CLIENT_SIDE`. Client-side Authentication
- `OIDC_CLAIMS`. OIDC Claims
- `AUTHENTICATION_TREE_DECISION_NODE`. Decision node script for authentication trees
- `OAuth2_ACCESS_TOKEN_MODIFICATION`. OAuth2 Access Token Modification

Default value: `POLICY_CONDITION`

amster attribute: `defaultContext`

4.1.2. Secondary Configurations

This service has the following Secondary Configurations.

4.1.2.1. 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 AM.

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. AM 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). AM 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. AM 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, AM creates new threads in the pool for script execution up to the configured maximum. It is recommended to set the maximum number of threads to 300.

ssoadm attribute: `maxThreads`

Thread pool queue size

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

For short, CPU-bound scripts, consider a small pool size and larger queue length. For I/O-bound scripts, for example, REST calls, consider a larger maximum pool size and a smaller queue.

Not hot-swappable: restart server for changes to take effect.

ssoadm attribute: `queueSize`

Thread idle timeout (seconds)

Length of time (in seconds) for a thread to be idle before AM 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, AM 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

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

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

ForgeRock publishes comprehensive documentation online:

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

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

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

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 identities 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 agent profile configuration information, typically created to delegate agent profile creation to the user installing a web or Java 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	<p>In general terms, a service exposing protected resources.</p> <p>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.</p>
Application type	<p>Application types act as templates for creating policy applications.</p> <p>Application types define a preset list of actions and functional logic, such as policy lookup and resource comparator logic.</p>

	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.
Client-based OAuth 2.0 tokens	After a successful OAuth 2.0 grant flow, AM returns a token to the client. This differs from CTS-based OAuth 2.0 tokens, where AM returns a <i>reference</i> to token to the client.
Client-based sessions	AM sessions for which AM returns session state to the client after each request, and require it to be passed in with the subsequent

request. For browser-based clients, AM sets a cookie in the browser that contains the session information.

For browser-based clients, AM sets a cookie in the browser that contains the session state. When the browser transmits the cookie back to AM, AM decodes the session state from the cookie.

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.

CTS-based OAuth 2.0 tokens

After a successful OAuth 2.0 grant flow, AM returns a *reference* to the token to the client, rather than the token itself. This differs from [client-based OAuth 2.0 tokens](#), where AM returns the entire token to the client.

CTS-based sessions

AM [sessions](#) that reside in the Core Token Service's token store. CTS-based sessions might also be cached in memory on one or more AM servers. AM tracks these 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.

Delegation

Granting users administrative privileges with AM.

Entitlement

Decision that defines which resource names can and cannot be accessed for a given identity 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 agent	Java web application installed in a web container that acts as a policy enforcement point, 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	Java, web, or custom 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 identities 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 identity 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).
Authentication Session	The interval while the user or entity is authenticating to AM.
Session	The interval that starts after the user has authenticated 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 CTS-based sessions and Client-based sessions.

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 CTS-based sessions , 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	<p>Group of AM servers configured the same way, accessed through a load balancer layer. The load balancer handles failover to provide service-level availability.</p> <p>The load balancer can also be used to protect AM services.</p>
Standard metadata	Standard federation configuration information that you can share with other access management software.
Stateless Service	<p>Stateless services do not store any data locally to the service. When the service requires data to perform any action, it requests it from a data store. For example, a stateless authentication service stores session state for logged-in users in a database. This way, any server in the deployment can recover the session from the database and service requests for any user.</p> <p>All AM services are stateless unless otherwise specified. See also Client-based sessions and CTS-based sessions.</p>
Subject	<p>Entity that requests access to a resource</p> <p>When an identity successfully authenticates, AM associates the identity with the Principal that distinguishes it from other identities. An identity can be associated with multiple principals.</p>
Identity store	Data storage service holding principals' profiles; underlying storage can be an LDAP directory service or a custom IdRepo implementation.
Web Agent	Native library installed in a web server that acts as a policy enforcement point with policies based on web page URLs.