



Authorization Guide

/ ForgeRock Access Management 6.5

Latest update: 6.5.5

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

Copyright © 2011-2022 ForgeRock AS.

Abstract

Guide to working with authorization. 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, NONINFRINGEMENT, OR THE ABSENCE OF LATENT OR OTHER DEFECTS, ACCURACY, OR THE PRESENCE OF ABSENCE OF ERRORS, WHETHER OR NOT DISCOVERABLE. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO SUCH EXCLUSION MAY NOT APPLY TO YOU.

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

DejaVu Fonts

Bitstream Vera Fonts Copyright

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

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

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

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

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

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

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

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

Arev Fonts Copyright

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

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

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

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

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

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

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

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

FontAwesome Copyright

Copyright (c) 2017 by Dave Gandy, <https://fontawesome.com/>.

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 Authorization	1
Resource Types, Policy Sets, and Policies	1
Policy Decisions	3
Authorization Examples	6
Reaching Policy Decisions	7
2. Implementing Authorization	8
Implementing Authorization Using the AM Console	8
Implementing Authorization Using the REST API	31
Implementing OAuth 2.0 Policies	93
3. Implementing Transactional Authorization	98
Introducing Transactional Authorization	98
Using Transactional Authorization	103
4. Customizing Authorization	116
Customizing Policy Evaluation With a Plug-In	116
Scripting a Policy Condition	122
5. Reference	132
Global Service Properties	132
Authorization API Functionality	139
A. About the REST API	144
Introducing REST	144
About ForgeRock Common REST	144
Cross-Site Request Forgery (CSRF) Protection	162
REST API Versioning	163
Specifying Realms in REST API Calls	166
Authentication and Logout using REST	167
Using the Session Token After Authentication	186
Server Information	187
Token Encoding	188
Logging	188
Reference	190
B. About Scripting	193
The Scripting Environment	193
Global Scripting API Functionality	198
Managing Scripts	200
Scripting	213
C. Getting Support	217
Glossary	218

Preface

This guide covers concepts, implementation procedures, and customization techniques for working with the authorization features of ForgeRock Access Management.

This guide is written for anyone using AM to manage authorization.

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 Authorization

This chapter provides an overview of authorization.

AM provides *access management*, which consists of:

- Authentication: determining who is trying to access a resource
- Authorization: determining whether to grant or deny access to the resource

Whether access is granted depends on what the policies about access are, who is trying to gain access, and perhaps some other conditions, such as whether the access itself needs to happen over a secure channel or what time of day it is.

Resource Types, Policy Sets, and Policies

Define authorization *policies* to allow AM to determine whether to grant a subject access to a resource.

A policy defines the following:

resources

The resource definitions constrain which resources, such as web pages or access to the boarding area, the policy applies to.

actions

The actions are verbs that describe what the policy allows users to do to the resources, such as read a web page, submit a web form, or access the boarding area.

subject conditions

The subject conditions constrain whom the policy applies to. Examples are: all authenticated users, only administrators, or only passengers with valid tickets for planes leaving soon.

Note that a subject condition that applies to all authenticated users includes the anonymous user.

environment conditions

The environment conditions set the circumstances under which the policy applies, such as only during work hours, only when accessing from a specific IP address, or only when the flight is scheduled to leave within the next four hours.

response attributes

The response attributes define information that AM attaches to a response following a policy decision, such as a name, email address, or frequent flyer status.

To help with the creation of policies, AM uses *resource types* and *policy sets*.

Resource types

Resource types define a template for the resources that policies apply to, and the actions that could be performed on those resources.

AM includes two resource types by default: **URL** and **OAuth2 Scope**.

URL resource type

The **URL** resource type acts as a template for protecting web pages or applications. It contains resource patterns, such as `*://*:*/**?`, which can be made more specific when used in the policy. The resource supports the following actions:

GET
POST
PUT
HEAD
PATCH
DELETE
OPTIONS

For example, an application for Example.com's HR service might contain resource types that constrain all policies to apply to URL resource types under `http://example.com/hr*` and `http://example.com/hr*?`, and only the HTTP **GET** and **POST** actions.

AM also includes a resource type to protect REST endpoints, with patterns including `https://*:*/**?` and the CRUDPAQ actions:

CREATE
READ
UPDATE
DELETE
PATCH
ACTION
QUERY

OAuth2 Scope resource type

The **OAuth2 Scope** resource type acts as a template for granting or denying OAuth 2.0 scopes. It contains a string-based scope pattern, `*`, and two URL-based scope patterns, such as `*://*:*/**?` and `*://*:*/**?`. The resource supports the **GRANT** action, which can be allowed or denied.

Policy Sets

Policy Sets are associated with a set of resource types, and contain one or more policies based upon the template it provides.

AM includes the following default policy sets:

- The Default Policy Set, `iPlanetAMWebAgentService` is the policy set configured by default for web and Java agents. You can create new policy sets for agents and configure them in the agent profile.
- The Default OAuth2 Scopes Policy Set, `oauth2Scopes`, is the policy set configured for the OAuth 2.0 service on the realm.

Important

The OAuth 2.0 service cannot be configured to use a different policy set. Configure all policies required for your OAuth 2.0 service in the Default OAuth2 Scopes Policy Set.

For more information on viewing, creating, and editing policies and resource types, see "Configuring Resource Types, Policy Sets, and Policies".

Policy Decisions

AM relies on policies to reach authorization decisions, such as whether to grant or to deny access to a resource or grant or deny OAuth 2.0 scopes.

Protecting Resources Through Policy Decisions

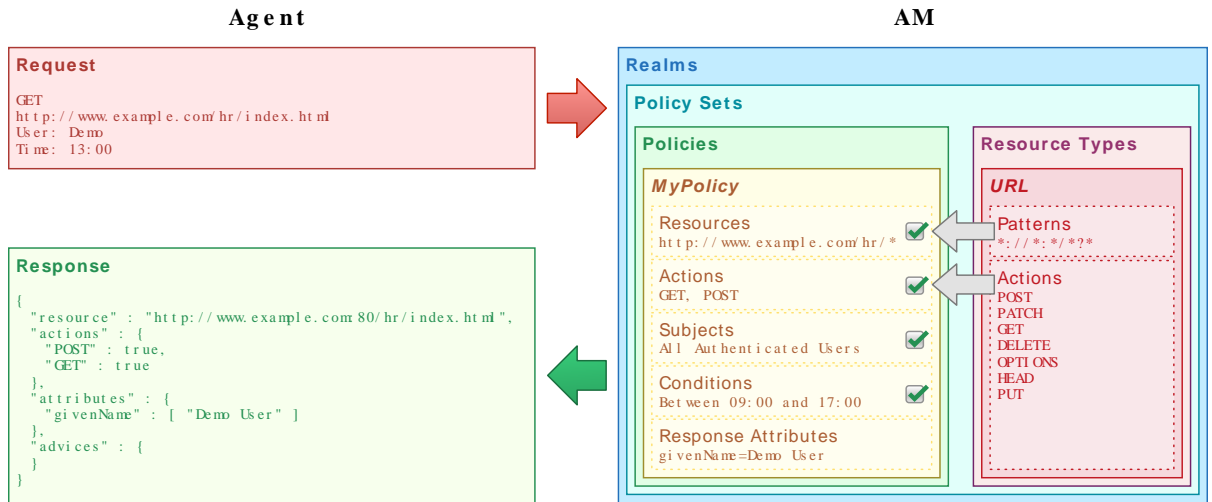
When you configure policy sets to protect resources, AM acts as the *policy decision point* (PDP), whereas AM web and Java agents act as *policy enforcement points* (PEP). In other words, an agent or other PEP takes responsibility only for enforcing a policy decision rendered by AM. When you configured applications and their policies in AM, you used AM as a *policy administration point* (PAP).

Concretely speaking, when a PEP requests a policy decision from AM it specifies the target resource(s), the policy set (default: `iPlanetAMWebAgentService`), and information about the subject and the environment. AM as the PDP retrieves policies within the specified policy set that apply to the target resource(s). AM then evaluates those policies to make a decision based on the conditions matching those of the subject and environment. When multiple policies apply for a particular resource, the default logic for combining decisions is that the first evaluation resulting in a decision to deny access takes precedence over all other evaluations. AM only allows access if all applicable policies evaluate to a decision to allow access.

AM communicates the policy decision to the PEP. The concrete decision, applying policy for a subject under the specified conditions, is called an *entitlement*.

The entitlement indicates the resource(s) it applies to, the actions permitted and denied for each resource, and optionally response attributes and *advice*.

Protecting Pages or Applications



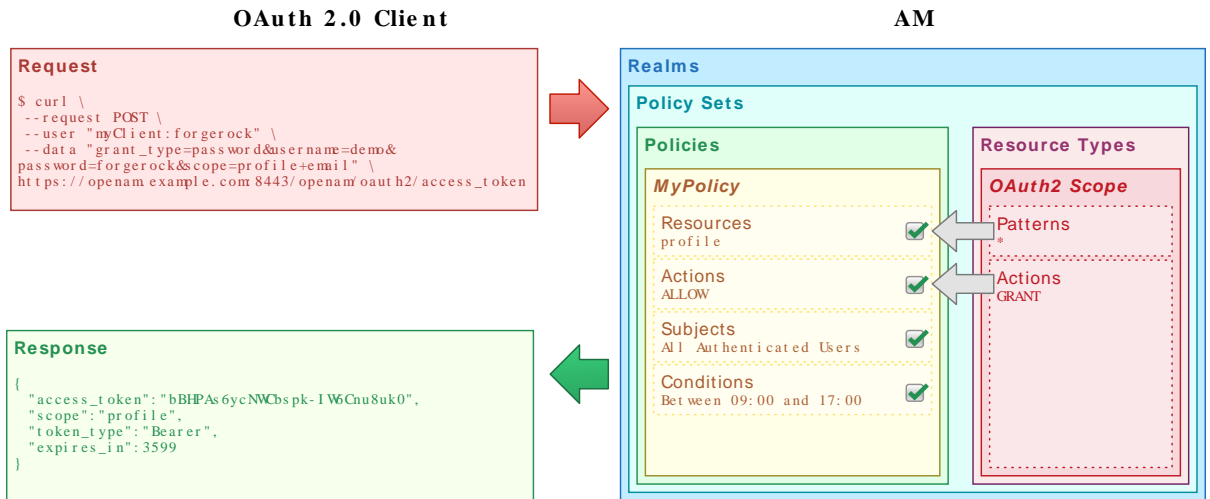
When AM denies a request due to a failed condition, AM can send advice to the PEP, and the PEP can then take remedial action. For instance, suppose a user comes to a web site having authenticated with an email address and password, which is configured as authentication level 0. Had the user authenticated using a one-time password, the user would have had authentication level 1 in their session. Yet, because they have authentication level 0, they currently cannot access the desired page, as the policy governing access requires authentication level 1. AM sends advice, prompting the PEP to have the user reauthenticate using a one-time password module, gaining authentication level 1, and thus having AM grant access to the protected page.

Granting or Denying OAuth 2.0 Scopes Through Policy Decisions

You can configure AM to grant scopes statically or dynamically:

- **Statically (Default).** You configure several OAuth 2.0 clients with different subsets of scopes and resource owners are redirected to a specific client depending on the scopes required. As long as the resource owner can authenticate and the client can deliver the same or a subset of the requested scopes, AM issues the token with the scopes requested. Therefore, two different users requesting scopes A and B to the same client will always receive scopes A and B.
- **Dynamically.** You configure an OAuth 2.0 client with a comprehensive list of scopes and resource owners authenticate against it. When AM receives a request for scopes, AM's Authorization Service grants or denies access scopes dynamically by evaluating authorization policies at runtime. Therefore, two different users requesting scopes A and B to the same client can receive different scopes based on policy conditions.

Granting or Denying OAuth2 2.0 Scopes Dynamically



When issuing access tokens, AM deduces consent status based on policy result: an **allow** policy result means consent is granted, while a **deny** result means it is denied.

If AM cannot deduce consent using a policy, for example, because none is defined, it is down to the resource owner to decide whether to grant consent. Note that this is only possible in flows where the resource owner interacts with the consent screen. If the resource owner cannot interact with the consent screen, for example, during the ROPC Grant flow, AM denies the scope.

Just like when granting scopes statically, AM only evaluates default scopes configured in the OAuth 2.0 client profile when no scope is requested. AM follows the same rules to deduce consent for both default and requested scopes.

When issuing refresh tokens, AM issues any scope that was previously consented to either by policy or by the resource owner on the consent screen, unless it is explicitly denied by a policy.

To understand which flows are interactive and which ones are not, see the examples in *"Implementing OAuth 2.0 Grant Flows"* in the *OAuth 2.0 Guide* and *"Implementing OpenID Connect Grant Flows"* in the *OpenID Connect 1.0 Guide*.

For examples of how to grant or deny scopes through policy decisions, see *"Implementing OAuth 2.0 Policies"*.

Authorization Examples

Protecting Resources

Consider the case where AM protects a user profile web page. An AM web agent installed in the web server intercepts client requests to enforce policy. The policy says that only authenticated users can access the page to view and to update their profiles.

When a user browses to the profile page, the AM agent intercepts the request. The web agent notices that the request is to access a protected resource, but the request is coming from a user who has not yet logged in and consequently has no authorization to visit the page. The web agent therefore redirects the user's browser to AM to authenticate.

AM receives the redirected user, serving a login page that collects the user's email and password. With the email and password credentials, AM authenticates the user, and creates a session for the user. AM then redirects the user to the web agent, which gets the policy decision from AM for the page to access, and grants access to the page.

While the user has a valid session with AM, the user can go away to another page in the browser, come back to the profile page, and gain access without having to enter their email and password again.

Notice how AM and the web agent handle the access in the example. The web site developer can offer a profile page, but the web site developer never has to manage login, or handle who can access a page. As AM administrator, you can change authentication and authorization independently of updates to the web site. You might need to agree with web site developers on how AM identifies users so web developers can identify users by their own names when they log in. By using AM and web or Java agents for authentication and authorization, your organization no longer needs to update web applications when you want to add external access to your Intranet for roaming users, open some of your sites to partners, only let managers access certain pages of your HR web site, or allow users already logged in to their desktops to visit protected sites without having to type their credentials again.

Granting or Denying OAuth 2.0 Scopes Dynamically

Consider the case of a company deployment that supports custom OAuth 2.0 clients and internal applications. The use of the internal application is bound by the terms and conditions specified by the company; therefore, the user does not need to consent to provide with his user profile information (for example, the `profile` scope).

To provide the internal application with the user profile automatically, the administrator creates a policy that grants the `profile` scope to all requests made by authenticated users using a particular OAuth 2.0 client.

Reaching Policy Decisions

AM has to match policies to resources to take policy decisions. For a policy to match, the resource has to match one of the resource patterns defined in the policy. The user making the request has to match a subject. Furthermore, at least one condition for each condition type has to be satisfied.

If more than one policy matches, AM has to reconcile differences. When multiple policies match, the order in which AM uses them to make a policy decision is not deterministic. However, a deny decision overrides an allow decision, and so by default once AM reaches a deny decision it stops checking further policies. If you want AM to continue checking despite the deny, navigate to [Configure > Global Services](#), select [Policy Configuration](#), and then enable [Continue Evaluation on Deny Decision](#).

Chapter 2

Implementing Authorization

This chapter covers how to implement authorization using the AM console and the REST API.

Implementing Authorization Using the AM Console

This section covers the following topics:

- "Configuring Resource Types, Policy Sets, and Policies"
- "Importing and Exporting Policies"
- "Delegating Policy Management"

Configuring Resource Types, Policy Sets, and Policies

You can configure resource types, policy sets, and policies by using the AM console. You can also manage them using the REST API and the **ssoadm** command. For more information, see "Implementing Authorization Using the REST API" and `ssoadm(1)` in the *Reference*.

Configuring Resource Types

This section describes the process of using the AM console for creating resource types, which define a template for the resources that policies apply to, and the actions that could be performed on those resources.

To Configure a Resource Type by Using the AM Console

1. In the AM console, select **Realms > Realm Name > Authorization > Resource Types**.
 - a. To create a new resource type, select **New Resource Type**.
 - b. To modify an existing resource type, select the resource type name.
 - c. To delete an existing resource type, in the row containing the resource type select the **Delete** button.

You can only delete resource types that are not being used by policy sets or policies. Trying to delete a resource type that is in use returns an HTTP 409 Conflict status code.

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

2. Provide a name for the resource type, and optionally a description.

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

3. To define resource patterns that policies using this resource type can expand upon, follow the steps below:
 - a. In the Add a new pattern box, enter a pattern with optional wildcards that the policies will use as a template.

For information on specifying patterns for matching resources, see "Specifying Resource Patterns with Wildcards".

- b. Select the Add Pattern button to confirm the pattern.

Tip

To remove a pattern, select the Delete icon.

4. To define the actions that policies using this resource type can allow or deny, follow the steps below:
 - a. In the Add a new action box, enter an action related to the types of resources being described, and then select Add Action.
 - b. Select either allow or deny as the default state for the action.

To remove an action, select the Delete icon.

5. Continue adding the patterns and actions that your resource type requires.

Configuring Resource Types in the AM Console

New Resource Type Help

DETAILS

Name

SPECIFY PATTERNS

Define resource patterns that policies using this resource type can expand upon.

PATTERNS

light://*	✕
-----------	---

SPECIFY ACTIONS

Define the actions that policies using this resource type can allow or deny.

ACTION	DEFAULT STATE	
switch_on	<input checked="" type="radio"/> Allow <input type="radio"/> Deny	✕

6. Select Create Resource Type to save a new resource type or Save Changes to save modifications to an existing resource type.

Configuring Policy Sets

This section describes how to use the AM console to create policy sets, which are used as templates for policies protecting Web sites, Web applications, or other resources.

When creating policy sets, consider the following points:

- To make use of a policy set configured with policies containing the **URL** resource type or any other custom resource type, you must configure a web or Java agent to use the policy set for policy decisions. For details see "To Specify the Realm and Application for Policy Decisions" in the *Setup and Maintenance Guide*.
- AM only honors **OAuth2 Scope** resource type policies configured in the Default OAuth2 Scopes Policy Set. Configure all policies required for your OAuth 2.0 service in the Default OAuth2 Scopes Policy Set.

To Configure a Policy Set Using the AM Console

1. In the AM console, navigate to Realms > *Realm Name* > Authorization > Policy Sets.
 - a. To create a new policy set, select New Policy Set.
 - b. To modify an existing policy set, select it from the table.
2. Enter an ID for the policy set. This is a required parameter.

Once a policy set is created, you cannot change its **ID**.

3. Enter a name for the policy set. The name is optional and is for display purposes only.

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

4. From the Resource Types drop-down list, select one or more resource types that policies in this policy set will use.

To remove a resource type from the policy set, select the label, and then press **Delete** or **Backspace**.

Important

Configure the **OAuth2 Scope** resource type only in the Default OAuth2 Scopes Policy Set. Any policy configured for the **OAuth2 Scope** resource type outside the default policy set will not be evaluated.

5. Select Create to save the new policy set or Save Changes to save modifications to an existing policy set.

Configuring Policies

This section describes the process of using the AM console to configure policies, which are used to protect a web site, web application, or other resource.

To Configure a Policy Using the AM Console

1. In the AM console, select Realms > *Realm Name* > Authorization > Policy Sets, and then select the name of the policy set in which to configure a policy:
2. To create a new policy, select Add a Policy.
3. In the Name field, enter a descriptive name for the policy.

Note

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

4. To define resources that the policy applies to, follow the steps below:
 - a. Select a resource type from the Resource Type drop-down list. The set of resource patterns within the selected resource type will populate the Resources drop-down list. For information on configuring resource types, see "Configuring Resource Types".
 - b. Select a resource pattern from the Resources drop-down list.
 - c. (Optional) Replace the asterisks with values to define the resources that the policy applies to. For information on specifying patterns for matching resources, see "Specifying Resource Patterns with Wildcards".

Editing Resource Patterns

New Policy ? Help

Name

Resource Type

Select the type of resource for which this policy will manage access.

Resources

::*

*

://

hr.example.com

:

*

/

sales

The **OAuth2 Scope** resource type has the same resource patterns as the **URL** resource type, and also the ***** pattern. Use the resource patterns that are most relevant for the scopes in your environment.

Editing OAuth2 Scope Resource Type Resource Patterns

Important

Before testing your OAuth 2.0 policies, ensure your OAuth 2.0 service is configured to interact with AM's Authorization Service. Perform the following steps:

1. Navigate to Realms > Realm Name > Services > OAuth2 Provider
2. Ensure that Use Policy Engine for Scope decisions is enabled.

For more information about testing OAuth 2.0 policies, see "Implementing OAuth 2.0 Policies".

- d. Select Add to save the resource.

The AM console displays a page for your new policy. The Tab pages let you modify the policy's properties.

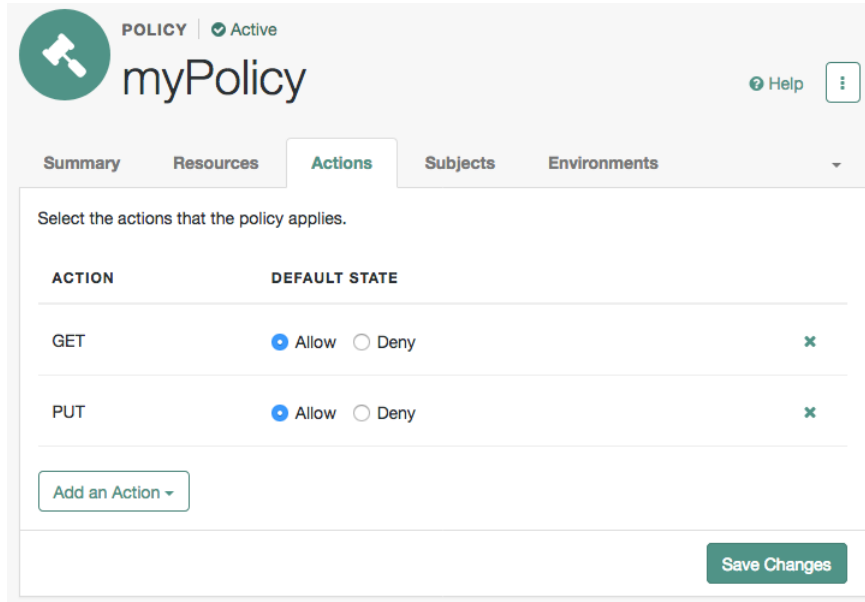
Tip

To remove a resource, select Delete.

5. Repeat these steps to add all the resources to which your policy applies, and then select Create.
6. To configure the policy's actions, select the Actions tab and perform the following:

- a. Select an action that the policy applies to by selecting them from the Add an Action drop-down list.
- b. Select whether to allow or deny the action on the resources specified earlier.

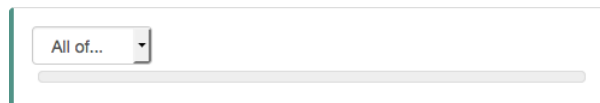
Allowing or Denying the Action for the Resource



- c. Repeat these steps to add all the appropriate actions, and then select Save Changes.
7. Define conditions in the AM console by combining logical operators with blocks of configured parameters to create a rule set that the policy uses to filter requests for resources. Use drag and drop to nest logical operators at multiple levels to create complex rule sets.

Valid drop-points in which to drop a block are displayed with a grey horizontal bar.

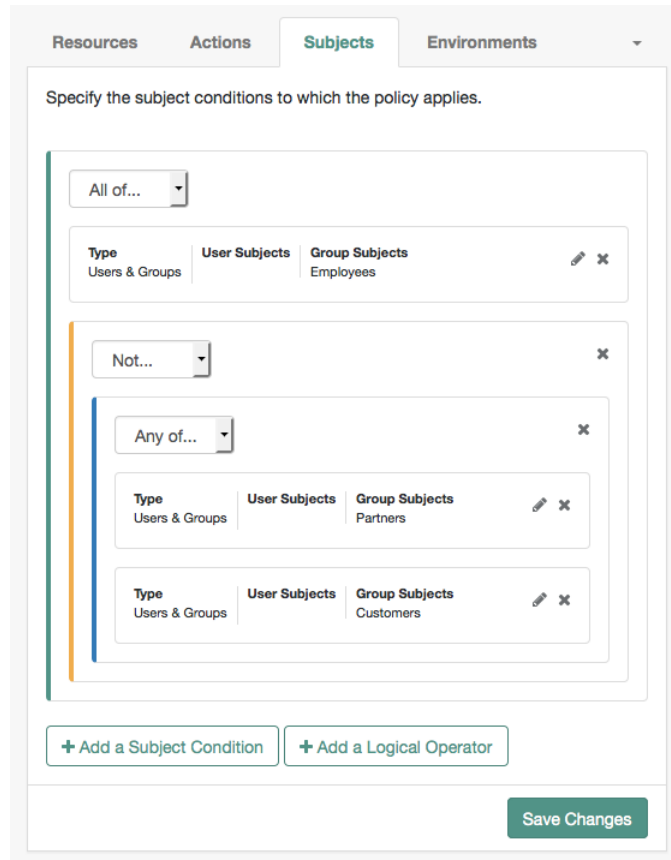
Valid Drop Point



- a. To define the subjects that the policy applies to, complete the following steps on the Subjects tab:

- i. Select Add a Subject Condition, choose the type from the drop-down menu, specify any required subject values, select the checkmark to the right when done, and then drag the block into a valid drop point in the rule set above.

Nesting subject conditions



The available subject condition types are:

Authenticated Users

Any user that has successfully authenticated with AM.

Users & Groups

A user or group as defined in the realm containing the policy. To manage the identities and groups in a realm, navigate to Realms > *Realm Name* > Identities.

Select one or more users or groups from the Identities or Groups tabs, which display the identities and groups available within the realm.

To remove an entry, select the value, and then press **Delete** (Windows/GNU/Linux) or **Backspace** (Mac OS X).

OpenID Connect/Jwt Claim

Validate a claim within a JSON Web Token (JWT).

Type the name of the claim to validate in the Claim Name field, for example `sub`, and the required value in the Claim Value field, and then select the checkmark.

Repeat the step to enter additional claims.

The claim(s) will be part of the JWT payload together with the JWT header and signature. The JWT is sent in the authorization header of the bearer token.

This condition type only supports string equality comparisons, and is case-sensitive.

Never Match

Never match any subject. Has the effect of disabling the policy, as it will never match a subject.

If you do not set a subject condition, "Never Match" is the default. In other words, you must set a subject condition for the policy to apply.

To match regardless of the subject, configure a subject condition that is "Never Match" inside a logical **Not** block.

- ii. To add a logical operator, select the Add a Logical Operator button, choose between **ALL Of**, **Not**, and **Any Of** from the drop-down list, and then drag the block into a valid drop point in the rule set above.
 - iii. Continue combining logical operators and subject conditions. To edit an item, select the Edit button. To remove an item, select the Delete button. When complete, select Save Changes.
- b. To configure environment conditions in the policy, complete the following steps on the Environments tab:
- i. To add an environment condition, select the Environment Condition button, choose the type from the drop-down list, specify any required parameters, and then drag the block into a drop-point in a logical block above.

Note

Script is the only environmental condition available for OAuth 2.0 policies.

The available environment condition types are:

Active Session Time

Make the policy test how long the user's session has been active, as specified in Max Session Time. To terminate the session if it has been active for longer than the specified time, set Terminate Sessions to **True**. The user will need to reauthenticate.

Authentication by Module Chain

Make the policy test the service that was used to authenticate the user.

Authentication by Module Instance

Make the policy test the authentication module used to authenticate, specified in Authentication Scheme. Specify a timeout for application authentication in Application Idle Timeout Scheme and the name of the application in Application Name.

Authentication Level (greater than or equal to)

Make the policy test the minimum acceptable authentication level specified in Authentication Level.

Authentication Level (less than or equal to)

Make the policy test the maximum acceptable authentication level specified in Authentication Level.

Authentication to a Realm

Make the policy test the realm to which the user authenticated.

A session can only belong to one realm and session upgrade between realms is not allowed.

Current Session Properties

Make the policy test property values set in the user's session.

Set Ignore Value Case to **True** to make the test case-insensitive.

Specify one or more pairs of session properties and values using the format `property:value`. For example, specify `clientType:genericHTML` to test whether the value of the `clientType` property is equal to `genericHTML`.

Identity Membership

Make the policy apply if the UUID of the invocator is a member of at least one of the AMIdentity objects specified in AM Identity Name.

Often used to filter requests on the identity of a Web Service Client (WSC).

Note

Java Agents and Web Agents do not support the Identity Membership environment condition. Instead, use the equivalent Users & Groups subject condition.

IPv4 Address/DNS Name

Make the policy test the IP version 4 address that the request originated from.

The IP address is taken from the `requestIp` value of policy decision requests. If this is not provided, the IP address stored in the SSO token is used instead.

Specify a range of addresses to test against by entering four sets of up to three digits, separated by full stops (.) in both Start IP and End IP.

If only one of these values is provided, it is used as a single IP address to match.

Optionally, specify a DNS name in DNS Name to filter requests to that domain.

IPv6 Address/DNS Name

Make the policy test the IP version 6 address that the request originated from.

The IP address is taken from the `requestIp` value of policy decision requests. If this is not provided, the IP address stored in the SSO token is used instead.

Specify a range of addresses to test against by entering eight sets of four hexadecimal characters, separated by a colon (:) in both Start IP and End IP.

If only one of these values is provided, it is used as a single IP address to match.

Optionally, specify a DNS name in DNS Name to filter requests to those coming from the specified domain.

Use an asterisk (*) in the DNS name to match multiple subdomains. For example `*.example.com` applies to requests coming from `www.example.com`, `secure.example.com`, or any other subdomain of `example.com`.

LDAP Filter Condition

Make the policy test whether the user's entry can be found using the LDAP search filter you specify in the directory configured for the policy service, which by default is the identity repository. Navigate to Configure > Global Services, and then select Policy Configuration to see the global LDAP configuration.

Alternatively, to configure these settings for a realm, navigate to Realms > *Realm Name* > Services, and then select Policy Configuration.

OAuth2 Scope

Make the policy test whether an authorization request includes all of the specified OAuth 2.0 scopes.

Scope names must follow OAuth 2.0 scope syntax described in RFC 6749, *Access Token Scope*. As described in that section, separate multiple scope strings with spaces, such as `openid profile`.

The scope strings match regardless of order in which they occur, so `openid profile` is equivalent to `profile openid`.

The condition is also met when additional scope strings are provided beyond those required to match the specified list. For example, if the condition specifies `openid profile`, then `openid profile email` also matches.

Resource/Environment/IP Address

Make the policy apply to a complex condition such as whether the user is making a request from the localhost and has also authenticated with the LDAP authentication module.

Entries must take the form of an `IF...ELSE` statement. The `IF` statement can specify either `IP` to match the user's IP address, or `dnsName` to match their DNS name.

If the `IF` statement is true, the `THEN` statement must also be true for the condition to be fulfilled. If not, relevant advice is returned in the policy evaluation request.

The available parameters for the `THEN` statement are as follows:

`module`

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

`service`

The service that was used to authenticate the user.

authlevel

The minimum required authentication level.

role

The role of the authenticated user.

user

The name of the authenticated user.

redirectURL

The URL the user was redirected from.

realm

The realm that was used to authenticate the user.

The IP address can be IPv4, IPv6, or a hybrid of the two.

Example: `IF IP=[127.0.0.1] THEN role=admin.`

Script

Make the policy depend on the outcome of a JavaScript or Groovy script executed at the time of the policy evaluation.

For information on scripting policy conditions, see "Scripting a Policy Condition".

`Script` is the only environmental condition available for OAuth 2.0 policies. Use scripts to capture the `ClientId` environmental attribute.

Time (day, date, time, and timezone)

Make the policy test when the policy is evaluated.

The values for day, date and time must be set in pairs that comprise a start and an end.

Create conditions that apply between a start and end date and time.

The screenshot shows a configuration form for a 'Time' condition. The 'Type' dropdown is set to 'Time (day, date, time, and timezone)'. Below this, there are seven input fields arranged in two rows:

- Start Time: 09:00
- End Time: 17:30
- Start Day: Monday
- End Day: Friday
- Start Date: 2015:12:16
- End Date: YYYY:MM:DD
- Time Zone: GMT

Transaction

Make the policy depend on the successful completion of a transaction performed by the user.

Configure a transaction with an authentication strategy that asks the user to reauthenticate before being allowed access to the resource.

Transactions support the following authentication strategies:

- **Authenticate to Chain:** Specify the name of an authentication chain the user must successfully complete to access the protected resource.
- **Authenticate to Realm:** Specify the full path of a realm in which the user must successfully authenticate to access the protected resource.

For example, `/sales/internal`.

- **Authenticate to Tree:** Specify the name of an authentication tree the user must successfully traverse to access the protected resource.
- **Authenticate to Module:** Specify the name of an authentication module the user must successfully authenticate against to access the protected resource.
- **Auth Level:** Specify the minimum authentication level the user must achieve to access the protected resource.

Note

If you specify a minimum, you must ensure there are methods available to users to reach that level. If none are found, the policy will return a `400 Bad request` error when attempting to complete the transaction.

For more information on transactional authorization, see "*Implementing Transactional Authorization*".

- ii. To add a logical operator, select the Logical button, choose between **All Of**, **Not**, and **Any Of** from the drop-down list, and then drag the block into a valid drop point in the rule set above.
 - iii. Continue combining logical operators and environment conditions, and when finished, select Save Changes.
8. (Optional) You can add response attributes, retrieved from the user entry in the identity repository, into the headers of the request at policy decision time (not available for the **OAuth2 Scope** resource type). The web or Java agent for the protected resources/applications or the protected resources/applications themselves retrieve the policy response attributes to customize

or personalize the application. Policy response attributes come in two formats: subject attributes and static attributes.

To configure response attributes in the policy, complete the following steps on the Response attributes tab:

- a. To add subject attributes, select them from the Subject attributes drop-down list

To remove an entry, select the value, and then press **Delete** (Windows/GNU/Linux) or **Backspace** (Mac OS X)

- b. To add a static attribute, specify the key-value pair for each static attribute. Enter the Property Name and its corresponding Property Value in the fields, and then select the Add (+) icon.

Note

To edit an entry, select the Edit icon in the row containing the attribute, or select the row itself. To remove an entry, select the Delete icon in the row containing the attribute.

- c. Continue adding subject and static attributes, and when finished, select Save Changes.

Specifying Resource Patterns with Wildcards

Resource patterns can specify an individual URL or resource name to protect. Alternatively, a resource pattern can match URLs or resource names by using wildcards.

- The wildcards you can use are `*` and `-*`.

These wildcards can be used throughout resource patterns to match URLs or resource names. For a resource pattern used to match URLs, wildcards can be employed to match the scheme, host, port, path, and query string of a resource.

- When used within the path segment of a resource, the wildcard `*` matches multiple path segments.

For example, `http://www.example.com/*` matches `http://www.example.com/`, `http://www.example.com/index.html`, and also `http://www.example.com/company/images/logo.png`.

- When used within the path segment of a resource, the wildcard `-*` will only match a single path segment.

For example, `http://www.example.com/-*` matches `http://www.example.com/index.html` but does not match `http://www.example.com/company/resource.html` or `http://www.example.com/company/images/logo.png`.

- Wildcards do not match `?`. You must explicitly add patterns to match URLs with query strings.

- When matching URLs sent from a web or Java agent, an asterisk (*) used at the end of a pattern after a ? character matches one or more characters, not zero or more characters.

For example, `http://www.example.com/*?*` matches `http://www.example.com/users?_action=create`, but not `http://www.example.com/users?`.

To match everything under `http://www.example.com/` specify three patterns, one for `http://www.example.com/*`, one for `http://www.example.com/*?`, and one for `http://www.example.com/*?*`.

- When matching resources by using the `policies?_action=evaluate` REST endpoint, an asterisk (*) used at the end of a pattern after a ? character matches zero or more characters.

For example, `http://www.example.com/*?*` matches `http://www.example.com/users?_action=create`, as well as `http://www.example.com/users?`.

To match everything under `http://www.example.com/` specify two patterns, one for `http://www.example.com/*`, one for `http://www.example.com/*?*`.

- When defining patterns to match URLs with query strings, AM sorts the query string field-value pairs alphabetically by field name when normalizing URLs before checking whether a policy matches. Therefore the query string `?subject=SPBnfm+t5PlP+ISyQhVlpLE22A8=&action=get` is equivalent to the query string `?action=get&subject=SPBnfm+t5PlP+ISyQhVlpLE22A8=`.

- Duplicate slashes (/) are not considered part of the resource name to match. A trailing slash is considered by AM as part of the resource name.

For example, `http://www.example.com//path/`, and `http://www.example.com/path//` are treated in the same way.

`http://www.example.com/path`, and `http://www.example.com/path/` are considered two distinct resources.

- Wildcards can be used to match protocols, host names, and port numbers.

For example, `*://*:*/*` matches `http://www.example.com:80/index.html`, `https://www.example.com:443/index.html`, and `http://www.example.net:8080/index.html`.

When a port number is not explicitly specified, then the default port number is implied. Therefore `http://www.example.com/*` is the same as `http://www.example.com:80/*`, and `https://www.example.com/*` is the same as `https://www.example.com:443/*`.

- Wildcards cannot be escaped.
- Do not mix `*` and `-*` in the same pattern.
- To match a resource that uses non-ASCII characters, percent-encode the resource when creating the rule.

For example, to match resources under an Internationalized Resource Identifier (IRI) such as `http://www.example.com/forstå`, specify the following percent-encoded pattern:

```
http://www.example.com:80/forst%C3%A5/*
```

- By default, comparisons are not case sensitive. The delimiter, wildcards and case sensitivity are configurable. To see examples of other configurations, in the AM console, navigate to Configure > Global Services, select Policy Configuration, and scroll to Resource Comparator.

Importing and Exporting Policies

You can import and export policies to and from files.

You can use these files to backup policies, transfer policies between AM instances, or store policy configuration in a version control system such as Git or Subversion.

AM supports exporting policies in JSON and *eXtensible Access Control Markup Language (XACML) Version 3.0* format. The features supported by each format are summarized in the table below:

Comparison of Policy Import/Export Formats

Feature	Supported?	
	JSON	XACML
Can be imported/exported from within the AM console?	No	Yes
Can be imported/exported on the command line, using the ssoadm command?	Yes	Yes
Exports policies?	Yes	Yes
Exports policy sets?	Yes	Partial ^a
Exports resource types?	Yes	Partial ^a
Creates an exact copy of the original policy sets, resource types, and policies upon import?	Yes	Partial ^b

^aOnly the details of policy sets and resource types that are actually used within a policy is exported to the XACML format. The full definition is not exported.

^bPolicy sets and resource types will be generated from the details in the XML, but may not match the definitions of the originals, for example the names are auto-generated.

Note

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

You can import and export policies by using the policy editor in the AM console, using the REST API, or with the **ssoadm** command.

- "To Export Policies in XACML Format (AM Console)"
- "To Import Policies in XACML Format (AM Console)"
- "To Export Policies in JSON Format (Command Line)"

- "To Import Policies in JSON Format (Command Line)"
- "To Export Policies in XACML Format (Command Line)"
- "To Import Policies in XACML Format (Command Line)"

For information on importing and exporting policies in XACML format by using the REST API, see "Importing and Exporting XACML 3.0".

To Export Policies in XACML Format (AM Console)

- In the AM console, select Realms > *Realm Name* > Authorization > Policy Sets, and then select Export Policy Sets.

All policy sets, and the policies within will be exported in XACML format.

To Import Policies in XACML Format (AM Console)

1. In the AM console, select Realms > *Realm Name* > Authorization > Policy Sets, and then select Import Policy Sets.
2. Browse to the XACML format file, select it, and then select Open.

Any policy sets, and the policies within will be imported from the selected XACML format file.

Note

Policy sets and resource types will be generated from the details in the XACML format file, but may not match the definitions of the originals, for example the names are auto-generated.

To Export Policies in JSON Format (Command Line)

- Use the **ssoadm policy-export** command:

```
$ ssoadm \  
  policy-export \  
  --realm "/" \  
  --servername "https://openam.example.com:8443/openam" \  
  --jsonfile "myPolicies.json" \  
  --adminid amadmin \  
  --password-file /tmp/pwd.txt  
{  
  "RESOURCE_TYPE" : 1,  
  "POLICY" : 1,  
  "APPLICATION" : 1  
}
```

If exporting from a subrealm, include the top level realm ("/") in the `--realm` value. For example `--realm "/myRealm"`.

For more information on the syntax of this command, see "ssoadm policy-export" in the *Reference*.

To Import Policies in JSON Format (Command Line)

- Use the **ssoadm policy-import** command:

```
$ ssoadm \  
policy-import \  
--realm "/myRealm" \  
--servername "https://openam.example.com:8443/openam" \  
--jsonfile "myPolicies.json" \  
--adminid amadmin \  
--password-file /tmp/pwd.txt  
{  
  "POLICY" : {  
    "CREATE_SUCCESS" : {  
      "count" : 1  
    }  
  },  
  "RESOURCE_TYPE" : {  
    "CREATE_SUCCESS" : {  
      "count" : 1  
    }  
  },  
  "APPLICATION" : {  
    "CREATE_SUCCESS" : {  
      "count" : 1  
    }  
  }  
}
```

If importing to a subrealm, include the top level realm ("/") in the `--realm` value. For example `--realm "/myRealm"`.

For more information on the syntax of this command, see "ssoadm policy-import" in the *Reference*.

To Export Policies in XACML Format (Command Line)

- Use the **ssoadm list-xacml** command:

```
$ ssoadm \  
list-xacml \  
--realm "/" \  
--adminid amadmin \  
--password-file /tmp/pwd.txt  
  
<?xml version="1.0" encoding="UTF-8"?>  
<PolicySet  
  xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"  
  PolicyCombiningAlgId="urn...rule-combining-algorithm:deny-overrides"  
  Version="2014.11.25.17.41.15.597"
```

```

PolicySetId="/:2014.11.25.17.41.15.597">
<Target />
<Policy
  RuleCombiningAlgId="urn...rule-combining-algorithm:deny-overrides"
  Version="2014.11.25.17.40.08.067"
  PolicyId="myPolicy">
<Description />
<Target>
  <AnyOf>
    <AllOf>
      <Match
        MatchId="urn...entitlement:json-subject-match">
        <AttributeValue
          DataType="urn...entitlement.conditions.subject.AuthenticatedUsers">
          {}
        </AttributeValue>
        <AttributeDesignator
          MustBePresent="true"
          DataType="urn...entitlement.conditions.subject.AuthenticatedUsers"
          AttributeId="urn...entitlement:json-subject"
          Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject" />
        </Match>
      </AllOf>
    </AnyOf>
    <AnyOf>
      <AllOf>
        <Match
          MatchId="urn...entitlement:resource-match:application:iPlanetAMWebAgentService">
          <AttributeValue
            DataType="http://www.w3.org/2001/XMLSchema#string">
            http://www.example.com:8000/*?*
          </AttributeValue>
          <AttributeDesignator
            MustBePresent="true"
            DataType="http://www.w3.org/2001/XMLSchema#string"
            AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
            Category="urn...attribute-category:resource" />
          </Match>
        </AllOf>
      </AnyOf>
    <AnyOf>
      <AllOf>
        <Match
          MatchId="urn...application-match">
          <AttributeValue
            DataType="http://www.w3.org/2001/XMLSchema#string">
            iPlanetAMWebAgentService
          </AttributeValue>
          <AttributeDesignator
            MustBePresent="false"
            DataType="http://www.w3.org/2001/XMLSchema#string"
            AttributeId="urn...application-id"
            Category="urn...application-category" />
          </Match>
        </AllOf>
      </AnyOf>
    <AnyOf>
      <AllOf>
        <Match

```



```

MatchId="urn...entitlement:action-match:application:iPlanetAMWebAgentService">
  <AttributeValue
    DataType="http://www.w3.org/2001/XMLSchema#string">
    POST
  </AttributeValue>
  <AttributeDesignator
    MustBePresent="true"
    DataType="http://www.w3.org/2001/XMLSchema#string"
    AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
    Category="urn...attribute-category:action" />
  </Match>
</AllOf>
<AllOf>
  <Match
    MatchId="urn...entitlement:action-match:application:iPlanetAMWebAgentService">
    <AttributeValue
      DataType="http://www.w3.org/2001/XMLSchema#string">
      GET
    </AttributeValue>
    <AttributeDesignator
      MustBePresent="true"
      DataType="http://www.w3.org/2001/XMLSchema#string"
      AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
      Category="urn...attribute-category:action" />
    </Match>
  </AllOf>
</AnyOf>
</Target>
<VariableDefinition
  VariableId="...entitlement.applicationName">
  <AttributeValue
    DataType="http://www.w3.org/2001/XMLSchema#string">
    iPlanetAMWebAgentService
  </AttributeValue>
</VariableDefinition>
<VariableDefinition
  VariableId="...privilege.createdBy">
  <AttributeValue
    DataType="http://www.w3.org/2001/XMLSchema#string">
    id=amadmin,ou=user,dc=openam,dc=forgerock,dc=org
  </AttributeValue>
</VariableDefinition>
<VariableDefinition
  VariableId="...privilege.lastModifiedBy">
  <AttributeValue
    DataType="http://www.w3.org/2001/XMLSchema#string">
    id=amadmin,ou=user,dc=openam,dc=forgerock,dc=org
  </AttributeValue>
</VariableDefinition>
<VariableDefinition
  VariableId="...privilege.creationDate">
  <AttributeValue
    DataType="http://www.w3.org/2001/XMLSchema#dateTime">
    2014-11-25T17:40:08.067
  </AttributeValue>
</VariableDefinition>
<VariableDefinition
  VariableId="...privilege.lastModifiedDate">
  <AttributeValue

```

```

    DataType="http://www.w3.org/2001/XMLSchema#dateTime">
    2014-11-25T17:40:08.067
  </AttributeValue>
</VariableDefinition>
<Rule
  Effect="Permit"
  RuleId="null:permit-rule">
  <Description>Permit Rule</Description>
  <Target>
  <AnyOf>
  <AllOf>
  <Match
    MatchId="urn...entitlement:action-match:application:iPlanetAMWebAgentService">
  <AttributeValue
    DataType="http://www.w3.org/2001/XMLSchema#string">
    POST
  </AttributeValue>
  <AttributeDesignator
    MustBePresent="true"
    DataType="http://www.w3.org/2001/XMLSchema#string"
    AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
    Category="urn...attribute-category:action" />
  </Match>
  </AllOf>
  <AllOf>
  <Match
    MatchId="urn...entitlement:action-match:application:iPlanetAMWebAgentService">
  <AttributeValue
    DataType="http://www.w3.org/2001/XMLSchema#string">
    GET
  </AttributeValue>
  <AttributeDesignator
    MustBePresent="true"
    DataType="http://www.w3.org/2001/XMLSchema#string"
    AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
    Category="urn...attribute-category:action" />
  </Match>
  </AllOf>
  </AnyOf>
  </Target>
  <Condition>
  <Apply
    FunctionId="urn...entitlement:json-subject-and-condition-satisfied">
  <AttributeValue
    DataType="urn...entitlement.conditions.subject.AuthenticatedUsers"
    privilegeComponent="entitlementSubject">
    {}
  </AttributeValue>
  </Apply>
  </Condition>
  </Rule>
</Policy>
</PolicySet>

```

Policy definitions were returned under realm, /.

For more information on the syntax of this command, see "ssoadm list-xacml" in the *Reference*.

To Import Policies in XACML Format (Command Line)

- Use the **ssoadm create-xacml** command:

```
$ ssoadm \  
  create-xacml \  
  --realm "/" \  
  --adminid amadmin \  
  --password-file /tmp/pwd.txt \  
  --xmlfile policy.xml  
Policies were created under realm, /.
```

For more information on the syntax of this command, see "ssoadm create-xacml" in the *Reference*.

Delegating Policy Management

To delegate policy management and other administrative tasks, use privileges. You set privileges in the AM console under Realms > *Realm Name* > Identities > Groups.

For more information, see "Delegating Realm Administration Privileges" in the *Setup and Maintenance Guide*.

Implementing Authorization Using the REST API

This section describes how to manage and evaluate policies using AM's REST API.

For general information about the REST API, see "*About the REST API*".

About the REST Policy Endpoints

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

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`.

- Under `/json/realms/root/resourcetypes`, you find a JSON-based API for managing resource types.
- Under `/json/realms/root/applications` and `/json/applicationtypes` you find JSON-based APIs for administering policy sets and reading application types.
- Under `/json/realms/root/policies`, you find a JSON-based API for policy management and evaluation.
- Under `/json/conditiontypes` you find a JSON-based API for viewing what types of conditions you can use when defining policies.

- Under [/json/subjecttypes](#) you find a JSON-based API for viewing what types of subjects you can use when defining policies.
- Under [/json/subjectattributes](#) you find a JSON-based API for viewing subjects' attributes you can use when defining response attributes in policies.
- Under [/json/decisioncombiners](#) you find a JSON-based API for viewing implementations you can use when defining policies to specify how to combine results when multiple policies apply.

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

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

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

Requesting Policy Decisions

You can request policy decisions from AM by using the REST APIs described in this section. AM evaluates requests based on the context and the policies configured, and returns decisions that indicate what actions are allowed or denied, as well as any attributes or advice for the resources specified.

Note

This section does not apply to OAuth 2.0 policies.

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

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

Requesting Policy Decisions For Specific Resources

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

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 request policy decisions for specific resources, perform an HTTP POST using the evaluation action to the appropriate path under the URI where AM is deployed, `/json{/realms/root}/policies?_action=evaluate`. The payload for the HTTP POST is a JSON object that specifies at least the resources, and takes the following form.

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

The values for the fields shown above are explained below:

"resources"

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

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

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

"application"

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

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

"subject"

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

"ssoToken"

The value is the SSO token ID string for the subject, returned for example on successful authentication as described in "Authentication and Logout using REST".

You can use an OpenID Connect ID token if the client that the token has been issued for is authorized to use ID tokens as session tokens. For more information, see "Using ID Tokens as Session Tokens" in the *OpenID Connect 1.0 Guide*.

"jwt"

The value is a JWT string.

Note

If you pass the subject details as a JWT, AM does not attempt to validate the JWT signature or the claims in the JWT. It is assumed that you have already validated the JWT before calling the authorization endpoint. For AM-issued ID Tokens, you can, instead, pass the ID Token as the value of the `ssoToken` field (after adding your client to the Authorized SSO Clients list). In this case, AM will validate the token. For more information, see "Using ID Tokens as Subjects in Policy Decision" in the *OpenID Connect 1.0 Guide*.

"claims"

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

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

"environment"

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

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

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

```
$ curl \
--request POST \
--header "Content-Type: application/json" \
--header "Accept-API-Version: resource=2.1" \
--header "iPlanetDirectoryPro: AQIC5..." \
--data '{
  "resources": [
    "http://www.example.com/index.html",
    "http://www.example.com/do?action=run"
  ],
  "application": "iPlanetAMWebAgentService"
}
```

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

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

"resource"

A resource specified in the request.

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

"actions"

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

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

"attributes"

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

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

"advices"

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

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

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

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

See "Policy Decision Advice" for details.

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

Policy Decision Advice

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

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

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

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

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

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

Leads to this advice:


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

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

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

Leads to this advice:

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

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

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

Leads to this advice:

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

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

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

Leads to this advice:

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

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

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

Leads to this advice:

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

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

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

Leads to this advice:

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

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

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

Leads to this advice:

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

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

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

Requesting Policy Decisions For a Tree of Resources

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

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 request policy decisions for a tree of resources, perform an HTTP POST using the evaluation action to the appropriate path under the URI where AM is deployed, for example `/json/realms/root/realms/myRealm/policies?_action=evaluateTree`. The payload for the HTTP POST is a JSON object that specifies at least the root resource, and takes the following form.

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

The values for the fields shown above are explained below:

"resource"

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

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

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

"application"

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

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

"subject"

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

"ssoToken"

The value is the SSO token ID string for the subject, returned for example on successful authentication as described in, "Authentication and Logout using REST".

"jwt"

The value is a JWT string.

Note

If you pass the subject details as a JWT, AM does not attempt to validate the JWT signature or the claims in the JWT. It is assumed that you have already validated the JWT before calling the authorization endpoint. For AM-issued ID Tokens, you can, instead, pass the ID Token as the value of the `ssoToken` field (after adding your client to the Authorized SSO Clients list). In this case, AM will validate the token. For more information, see "Using ID Tokens as Subjects in Policy Decision" in the *OpenID Connect 1.0 Guide*.

"claims"

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

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

"environment"

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

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

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

```
$ curl \
--request POST \
--header "Content-Type: application/json" \
--header "iPlanetDirectoryPro: AQIC5...NDU1*" \
--header "Accept-API-Version: resource=1.0" \
--data '{
  "resource": "http://www.example.com/",
  "subject": { "ssoToken": "AQIC5...zE4*" }
}' \
"https://openam.example.com:8443/openam/json/realms/root/policies?_action=evaluateTree"
[
  {
    "resource": "http://www.example.com/",
    "actions": {
      "GET": true,
      "OPTIONS": true,
      "HEAD": true
    },
    "attributes": {
    },
    "advices": {
    }
  },
  {
    "resource": "http://www.example.com/*",
    "actions": {
      "POST": false,
      "PATCH": false,
      "GET": true,
      "DELETE": true,
      "OPTIONS": true,
      "HEAD": true,
      "PUT": true
    },
    "attributes": {
      "myStaticAttr": [
        "myStaticValue"
      ]
    },
    "advices": {
    }
  },
  {
    "resource": "http://www.example.com/*?*\"",
    "actions": {
      "POST": false,
      "PATCH": false,
      "GET": false,
      "DELETE": false,
      "OPTIONS": true,
      "HEAD": false,
      "PUT": false
    }
  }
]
```

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

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

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

"resource"

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

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

"actions"

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

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

"attributes"

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

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

"advices"

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

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

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

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

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

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

Managing Resource Types

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

For information on creating resource types by using the AM console, see "Configuring Resource Types".

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

- "Querying Resource Types"
- "Reading a Specific Resource Type"
- "Creating a Resource Type"
- "Updating a Resource Type"
- "Deleting a Specific Resource Type"

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

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

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

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

"uuid"

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

"name"

The name provided for the resource type.

"description"

An optional text string to help identify the resource type.

"patterns"

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

For more information on patterns in resource types and policies, see "Specifying Resource Patterns with Wildcards"

"actions"

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

"createdBy"

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

"creationDate"

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

"LastModifiedBy"

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

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

"LastModifiedDate"

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

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

Querying Resource Types

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

Note

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

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

```
$ curl \
--header "iPlanetDirectoryPro: AQIC5..." \
--header "Accept-API-Version: resource=1.0" \
https://openam.example.com:8443/openam/json/realm/root/realm/myrealm/resourcetypes?_queryFilter=true
{
  "result": [
    {
      "uuid": "12345a67-8f0b-123c-45de-6fab78cd01e3",
      "name": "LIGHTS",
      "description": "",
      "patterns": [
        "light:/**/*"
      ],
      "actions": {
        "switch_off": true,
        "switch_on": true
      },
      "createdBy": "id=amadmin,ou=user,dc=openam,dc=forgerock,dc=org",
      "creationDate": 1431013059131,
      "lastModifiedBy": "id=amadmin,ou=user,dc=openam,dc=forgerock,dc=org",
      "lastModifiedDate": 1431013069803
    }
  ],
  "resultCount": 1,
  "pagedResultsCookie": null,
  "remainingPagedResults": 0
}
```

Additional query strings can be specified to alter the returned results. For more information, see "Query".

Supported `_queryFilter` Fields and Operators

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

Reading a Specific Resource Type

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

Note

If the realm is not specified in the URL, 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 "Accept-API-Version: resource=1.0" \
https://openam.example.com:8443/openam/json/realms/root/realms/myrealm/
resourcetypes/12345a67-8f0b-123c-45de-6fab78cd01e3
{
  "uuid": "12345a67-8f0b-123c-45de-6fab78cd01e3",
  "name": "LIGHTS",
  "description": "",
  "patterns": [
    "light://*/*"
  ],
  "actions": {
    "switch_off": true,
    "switch_on": true
  },
  "createdBy": "id=amadmin,ou=user,dc=openam,dc=forgerock,dc=org",
  "creationDate": 1431013059131,
  "lastModifiedBy": "id=amadmin,ou=user,dc=openam,dc=forgerock,dc=org",
  "lastModifiedDate": 1431013069803
}
```

Creating a Resource Type

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

Note

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

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

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

```
$ curl \
--request POST \
--header "Content-Type: application/json" \
--header "iPlanetDirectoryPro: AQIC5..." \
--header "Accept-API-Version: resource=1.0" \
--data '{
  "name": "My Resource Type",
  "actions": {
    "LEFT": true,
    "RIGHT": true,
    "UP": true,
    "DOWN": true
  },
  "patterns": [
    "http://device/location/*"
  ]
}' \
https://openam.example.com:8443/openam/json/realm/root/realm/myrealm/resourcetypes/?_action=create
{
  "uuid": "12345a67-8f0b-123c-45de-6fab78cd01e4",
  "name": "My Resource Type",
  "description": null,
  "patterns": [
    "http://device/location*"
  ],
  "actions": {
    "RIGHT": true,
    "DOWN": true,
    "UP": true,
    "LEFT": true
  },
  "createdBy": "id=amadmin,ou=user,dc=openam,dc=forgerock,dc=org",
  "creationDate": 1431099940616,
  "lastModifiedBy": "id=amadmin,ou=user,dc=openam,dc=forgerock,dc=org",
  "lastModifiedDate": 1431099940616
}
```

Updating a Resource Type

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

Note

If the realm is not specified in the URL, 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.

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

```
$ curl \
--request PUT \
--header "Content-Type: application/json" \
--header "iPlanetDirectoryPro: AQIC5..." \
--header "Accept-API-Version: resource=1.0" \
--data '{
  "name": "My Resource Type",
  "uuid": "12345a67-8f0b-123c-45de-6fab78cd01e4"
  "actions": {
    "LEFT": true,
    "RIGHT": true,
    "UP": false,
    "DOWN": false
  },
  "patterns": [
    "http://device/location/*"
  ]
}' \
https://openam.example.com:8443/openam/json/realm/root/realm/myrealm/
resourcetypes/12345a67-8f0b-123c-45de-6fab78cd01e4
{
  "uuid": "12345a67-8f0b-123c-45de-6fab78cd01e4",
  "name": "My Resource Type",
  "description": null,
  "patterns": [
    "http://device/location/*"
  ],
  "actions": {
    "RIGHT": true,
    "DOWN": true,
    "UP": false,
    "LEFT": false
  },
  "createdBy": "id=amadmin,ou=user,dc=openam,dc=forgerock,dc=org",
  "creationDate": 1431099940616,
  "lastModifiedBy": "id=amadmin,ou=user,dc=openam,dc=forgerock,dc=org",
```

```
"lastModifiedDate":1637667798885
}
```

Deleting a Specific Resource Type

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

Note

If the realm is not specified in the URL, 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=1.0" \
https://openam.example.com:8443/openam/json/realms/root/realms/myrealm/
resourcetypes/12345a67-8f0b-123c-45de-6fab78cd01e4
{}
```

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

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

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

Managing Application Types

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

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

- "Querying Application Types"
- "Reading a Specific Application Type"

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

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

```
{
  "name": "iPlanetAMWebAgentService",
  "actions": {
    "POST": true,
    "PATCH": true,
    "GET": true,
    "DELETE": true,
    "OPTIONS": true,
    "PUT": true,
    "HEAD": true
  },
  "resourceComparator": "com.sun.identity.entitlement.URLResourceName",
  "saveIndex": "org.forgerock.openam.entitlement.indextree.TreeSaveIndex",
  "searchIndex": "org.forgerock.openam.entitlement.indextree.TreeSearchIndex",
  "applicationClassName": "com.sun.identity.entitlement.Application"
}
```

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

"name"

The name provided for the application type.

"actions"

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

"resourceComparator"

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

The following implementations are available:

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

"saveIndex"

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

"searchIndex"

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

"applicationClassName"

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

Querying Application Types

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

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

```
$ curl \
--header "iPlanetDirectoryPro: AQIC5..." \
--header "Accept-API-Version: resource=1.0" \
https://openam.example.com:8443/openam/json/realms/root/applicationtypes?_queryFilter=true
{
  "result" : [ ... application types ... ],
  "resultCount" : 8,
  "pagedResultsCookie" : null,
  "remainingPagedResults" : -1
}
```

Additional query strings can be specified to alter the returned results. For more information, see "Query".

Reading a Specific Application Type

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

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

```
$ curl \
--header "iPlanetDirectoryPro: AQIC5..." \
--header "Accept-API-Version: resource=1.0" \
https://openam.example.com:8443/openam/json/realms/root/applicationtypes/iPlanetAMWebAgentService
{
  "name": "iPlanetAMWebAgentService",
  "actions": {
    "POST": true,
    "PATCH": true,
    "GET": true,
    "DELETE": true,
    "OPTIONS": true,
    "PUT": true,
    "HEAD": true
  },
  "resourceComparator": "com.sun.identity.entitlement.URLResourceName",
  "saveIndex": "org.forgerock.openam.entitlement.indextree.TreeSaveIndex",
  "searchIndex": "org.forgerock.openam.entitlement.indextree.TreeSearchIndex",
  "applicationClassName": "com.sun.identity.entitlement.Application"
}
```

Managing Policy Sets

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

Policy set definitions set constraints for defining policies. AM includes the following default policy sets:

- The Default Policy Set, `iPlanetAMWebAgentService` is the policy set configured by default for web and Java agents. You can create new policy sets for agents and configure them in the agent profile.
- The Default OAuth2 Scopes Policy Set, `oauth2Scopes`, is the policy set configured for the OAuth 2.0 service on the realm.

Important

The OAuth 2.0 service cannot be configured to use a different policy set. Configure all policies required for your OAuth 2.0 service in the Default OAuth2 Scopes Policy Set.

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

- "Querying Policy Sets"
- "Reading a Specific Policy Set"
- "Creating Policy Sets"
- "Updating Policy Sets"
- "Deleting Policy Sets"

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

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

```
{
  "creationDate": 1431351677264,
  "lastModifiedDate": 1431351677264,
  "conditions": [
    "AuthenticateToService",
    "Script",
    "AuthScheme",
    "IPv6",
    "SimpleTime",
    "OAuth2Scope",
    "IPv4",
    "AuthenticateToRealm",
    "OR",
    "AMIdentityMembership",
    "LDAPFilter",
    "AuthLevel",
    "SessionProperty",
    "LEAuthLevel",
    "Session",
    "NOT",
    "AND",
    "ResourceEnvIP"
  ],
  "applicationType": "iPlanetAMWebAgentService",
  "subjects": [
    "JwtClaim",
    "AuthenticatedUsers",
    "Identity",
    "NOT",
    "AND",
    "NONE",
    "OR"
  ],
  "entitlementCombiner": "DenyOverride",
  "saveIndex": null,
  "searchIndex": null,
  "resourceComparator": null,
  "resourceTypeUuids": [
    "12345a67-8f0b-123c-45de-6fab78cd01e4"
  ],
  "attributeNames": [ ],
  "editable": true,
  "createdBy": "id=dsameuser,ou=user,dc=openam,dc=forgerock,dc=org",
  "lastModifiedBy": "id=dsameuser,ou=user,dc=openam,dc=forgerock,dc=org",
  "description": "The built-in Application used by AM Policy Agents.",
  "realm": "/",
  "name": "iPlanetAMWebAgentService"
}
```

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

"conditions"

Condition types allowed in the context of this policy set.

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

"applicationType"

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

"subjects"

Subject types allowed in the context of this policy set.

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

"entitlementCombiner"

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

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

"saveIndex"

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

"searchIndex"

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

"resourceComparator"

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

The following implementations are available:

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

"resourceTypeUuids"

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

"attributeNames"

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

"description"

String describing the policy set.

"realm"

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

"name"

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

"createdBy"

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

"creationDate"

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

"lastModifiedBy"

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

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

"lastModifiedDate"

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

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

Querying Policy Sets

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

Note

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

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

```
$ [${resources.dir}/endpoints/applications.bash:#applications-query-filter]
[${resources.dir}/endpoints/applications.bash:#applications-query-filter-expected]
```

Additional query strings can be specified to alter the returned results. For more information, see "Query".

Supported `_queryFilter` Fields and Operators

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

Reading a Specific Policy Set

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

Note

If the realm is not specified in the URL, 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 "Accept-API-Version: resource=1.0" \
```

```
https://openam.example.com:8443/openam/json/realms/root/applications/mypolicyset
{
  "creationDate":1431360678810,
  "lastModifiedDate":1431360678810,
  "conditions":[
    "AuthenticateToService",
    "AuthScheme",
    "IPv6",
    "SimpleTime",
    "OAuth2Scope",
    "IPv4",
    "AuthenticateToRealm",
    "OR",
    "AMIdentityMembership",
    "LDAPFilter",
    "SessionProperty",
    "AuthLevel",
    "LEAuthLevel",
    "Session",
    "NOT",
    "AND",
    "ResourceEnvIP"
  ],
  "applicationType":"iPlanetAMWebAgentService",
  "subjects":[
    "JwtClaim",
    "AuthenticatedUsers",
    "Identity",
    "NOT",
    "AND",
    "OR"
  ],
  "entitlementCombiner":"DenyOverride",
  "saveIndex":null,
  "searchIndex":null,
  "resourceComparator":"com.sun.identity.entitlement.URLResourceName",
  "resourceTypeUuids":[
    "12345a67-8f0b-123c-45de-6fab78cd01e2"
  ],
  "attributeNames":[]
},
  "editable":true,
  "createdBy":"id=amadmin,ou=user,dc=openam,dc=forgerock,dc=org",
  "lastModifiedBy":"id=amadmin,ou=user,dc=openam,dc=forgerock,dc=org",
  "description":"My example policy set.",
  "realm":"/",
  "name":"mypolicyset"
}
```

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

Creating Policy Sets

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

Note

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

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

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

```
$ curl \
--request POST \
--header "Content-Type: application/json" \
--header "iPlanetDirectoryPro: AQIC5..." \
--header "Accept-API-Version: resource=2.1" \
--data '{
  "name": "mypolicysset",
  "resourceTypeUuids": [
    "12345a67-8f0b-123c-45de-6fab78cd01e2"
  ],
  "realm": "/",
  "conditions": [
    "AND",
    "OR",
    "NOT",
    "AMIdentityMembership",
    "AuthLevel",
    "AuthScheme",
    "AuthenticateToRealm",
    "AuthenticateToService",
    "IPv4",
    "IPv6",
    "LDAPFilter",
    "LEAuthLevel",
    "OAuth2Scope",
    "ResourceEnvIP",
    "Session",
    "SessionProperty",
    "SimpleTime"
  ],
  "applicationType": "iPlanetAMWebAgentService",
  "description": "My example policy set.",
  "resourceComparator": "com.sun.identity.entitlement.URLResourceName",
  "subjects": [
    "AND",
    "OR",
```

```

    "NOT",
    "AuthenticatedUsers",
    "Identity",
    "JwtClaim"
  ],
  "entitlementCombiner": "DenyOverride",
  "saveIndex": null,
  "searchIndex": null,
  "attributeNames": [
  ]
} \
https://openam.example.com:8443/openam/json/realms/root/applications/?_action=create

```

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

Updating Policy Sets

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

Note

If the realm is not specified in the URL, 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.

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

```

$ curl \
--request PUT \
--header "iPlanetDirectoryPro: AQIC5..." \
--header "Content-Type: application/json" \
--header "Accept-API-Version: resource=2.1" \
--data '{
  "name": "myupdatedpolicyset",
  "description": "My updated policy set - new name and fewer allowable conditions/subjects.",
  "conditions": [
    "NOT",
    "SimpleTime"
  ],
  "subjects": [
    "AND",
    "OR",
    "NOT",
    "AuthenticatedUsers",
    "Identity"
  ]
}'

```

```

    ],
    "applicationType": "iPlanetAMWebAgentService",
    "entitlementCombiner": "DenyOverride",
    "resourceTypeUuids": [
        "76656a38-5f8e-401b-83aa-4ccb74ce88d2"
    ]
} \
https://openam.example.com:8443/openam/json/realms/root/applications/mypolicyset
{
    "creationDate": 1431362370739,
    "lastModifiedDate": 1431362390817,
    "conditions": [
        "NOT",
        "SimpleTime"
    ],
    "resourceComparator": "com.sun.identity.entitlement.URLResourceName",
    "resourceTypeUuids": [
        "76656a38-5f8e-401b-83aa-4ccb74ce88d2"
    ],
    "createdBy": "id=amadmin,ou=user,dc=openam,dc=forgerock,dc=org",
    "lastModifiedBy": "id=amadmin,ou=user,dc=openam,dc=forgerock,dc=org",
    "applicationType": "iPlanetAMWebAgentService",
    "subjects": [
        "AuthenticatedUsers",
        "Identity",
        "NOT",
        "AND",
        "OR"
    ],
    "entitlementCombiner": "DenyOverride",
    "saveIndex": null,
    "searchIndex": null,
    "attributeNames": [

    ],
    "editable": true,
    "description": "My updated policy set - new name and fewer allowable conditions/subjects.",
    "realm": "/",
    "name": "myupdatedpolicyset"
}

```

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

Deleting Policy Sets

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

Note

If the realm is not specified in the URL, 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=2.1" \
https://openam.example.com:8443/openam/json/realms/root/applications/myupdatedpolicysset
```

Managing Policies

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

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

- "Querying Policies"
- "Reading a Specific Policy"
- "Creating Policies"
- "Updating Policies"
- "Deleting Policies"
- "Copying and Moving Policies"

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

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

```
{
  "name": "mypolicy",
  "active": true,
  "description": "My Policy.",
  "applicationName": "iPlanetAMWebAgentService",
  "actionValues": {
    "POST": true,
    "GET": true
  },
  "resources": [
    "http://www.example.com:80/*",
    "http://www.example.com:80/*?*"
  ],
  "subject": {
    "type": "AuthenticatedUsers"
  },
  "condition": {
    "type": "SimpleTime",
    "startTime": "09:00",
    "endTime": "17:00",
    "startDay": "mon",
    "endDay": "fri",
    "enforcementTimeZone": "GMT"
  },
}
```

```
"resourceTypeUuid": "76656a38-5f8e-401b-83aa-4ccb74ce88d2",
"resourceAttributes": [
  {
    "type": "User",
    "propertyName": "givenName",
    "propertyValues": [ ]
  }
],
"lastModifiedBy": "id=amadmin,ou=user,dc=openam,dc=forgerock,dc=org",
"lastModifiedDate": "2015-05-11T17:39:09.393Z",
"createdBy": "id=amadmin,ou=user,dc=openam,dc=forgerock,dc=org",
"creationDate": "2015-05-11T17:37:24.556Z"
}
```

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

"name"

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

"active"

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

"description"

String describing the policy.

"resources"

List of the resource name pattern strings to which the policy applies. Must conform to the pattern templates provided by the associated resource type.

"applicationName"

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

"actionValues"

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

Tip

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

"subject"

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

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

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

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

Warning

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

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

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

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

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

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

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

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

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

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

```

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

```

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

"condition"

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

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

```

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

```

Note

Java Agents and Web Agents do not support the `AMIdentityMembership` environment condition. Instead, use the equivalent `Identity` subject condition.

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

```

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

```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

The available parameters for the THEN statement are as follows:

module

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

service

The authentication chain that was used to authenticate the user.

authLevel

The minimum required authentication level.

role

The role of the authenticated user.

user

The name of the authenticated user.

redirectURL

The URL from which the user was redirected.

realm

The realm to which the user authenticated.

- **"Session"** to specify how long the user's session has been active, and to terminate the session if deemed too old, such that the user must authenticate again. Note that AM terminates client-based sessions only if session blacklisting is in effect. For more information about session blacklisting, see "Session Termination" in the *Authentication and Single Sign-On Guide*.

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

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

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

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

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

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

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

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

"resourceTypeUuid"

The UUIDs of the resource type associated with the policy.

"resourceAttributes"

List of attributes to return with decisions. These attributes are known as *response attributes*, and do not apply to `OAuth2 Scope` resource types.

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

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

"createdBy"

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

"creationDate"

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

"lastModifiedBy"

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

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

"lastModifiedDate"

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

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

Querying Policies

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

- "To List All Policies in a Realm"
- "To Query Policies in a Realm by User or Group"

To List All Policies in a Realm

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

Note

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

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

```
$ curl \
--header "iPlanetDirectoryPro: AQIC5w..." \
--header "Accept-API-Version: resource=1.0, protocol=2.1" \
"https://openam.example.com:8443/openam/json/realm/root/realm/myrealm/policies?_queryFilter=true"
{
  "result": [
    {
      "name": "example",
      "active": true,
      "description": "Example Policy",
      "applicationName": "iPlanetAMWebAgentService",
      "actionValues": {
        "POST": false,
        "GET": true
      },
      "resources": [
        "http://www.example.com:80/*",
        "http://www.example.com:80/*?*?"
      ],
      "subject": {
        "type": "Identity",
        "subjectValues": [
```

```

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

```

Additional query strings can be specified to alter the returned results. For more information, see "Query".

Supported `_queryFilter` Fields and Operators

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

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

To Query Policies in a Realm by User or Group

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

Tip

You can obtain the universal ID for a user or group by using REST. See "Reading Identities using the REST API" in the *Setup and Maintenance Guide*.

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

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

To query policies by user or group:

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

```
$ curl \
--get \
--header "iPlanetDirectoryPro: AQIC5w..." \
--header "Accept-API-Version: resource=1.0" \
--data "_queryId=queryByIdentityUid" \
--data "uid=id=demo,ou=user,o=myrealm,ou=services,dc=openam,dc=forgerock,dc=org" \
"https://openam.example.com:8443/openam/json/realm/root/realm/myrealm/policies"
{
  "result": [
    {
      "name": "mySubRealmPolicy",
      "active": true,
    }
  ]
}
```

```

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

```

Note

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

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

Reading a Specific Policy

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

Note

If the realm is not specified in the URL, 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..." \
"https://openam.example.com:8443/openam/json/realms/root/policies/example"
{
  "result": [
    {
      "name": "example",
      "active": true,
      "description": "Example Policy",
      "applicationName": "iPlanetAMWebAgentService",
      "actionValues": {
        "POST": false,
        "GET": true
      },
      "resources": [
        "http://www.example.com:80/*",
        "http://www.example.com:80/*?"
      ],
      "subject": {
        "type": "Identity",
        "subjectValues": [
          "uid=demo,ou=People,dc=example,dc=com"
        ]
      },
      "resourceTypeUuid": "12345a67-8f0b-123c-45de-6fab78cd01e4",
      "lastModifiedBy": "id=amadmin,ou=user,dc=openam,dc=forgerock,dc=org",
      "lastModifiedDate": "2015-05-11T14:48:08.711Z",
      "createdBy": "id=amadmin,ou=user,dc=openam,dc=forgerock,dc=org",
      "creationDate": "2015-05-11T14:48:08.711Z"
    }
  ],
  "resultCount": 1,
  "pagedResultsCookie": null,
  "remainingPagedResults": 0
}
    
```

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

Creating Policies

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

Note

If the realm is not specified in the URL, 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.

Important

Configure the **OAuth2 Scope** resource type only in the Default OAuth2 Scopes Policy Set. Any policy configured for the **OAuth2 Scope** resource type outside the default policy set will not be evaluated.

Before testing your OAuth 2.0 policies, ensure your OAuth 2.0 service is configured to interact with AM's Authorization Service. Perform the following steps:

- Navigate to Realms > Realm Name > Services > OAuth2 Provider
- Ensure that Use Policy Engine for Scope decisions is enabled.

For more information about testing OAuth 2.0 policies, see "Implementing OAuth 2.0 Policies".

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

```
$ curl \
--request POST \
--header "Content-Type: application/json" \
--header "iPlanetDirectoryPro: AQIC5..." \
--header "Accept-API-Version: resource=1.0" \
--data '{
  "name": "mypolicy",
  "active": true,
  "description": "My Policy.",
  "applicationName": "iPlanetAMWebAgentService",
  "actionValues": {
    "POST": false,
    "GET": true
  },
  "resources": [
    "http://www.example.com:80/*",
    "http://www.example.com:80/*?*?"
  ],
  "subject": {
    "type": "Identity",
    "subjectValues": [
      "uid=demo,ou=People,dc=example,dc=com"
    ]
  },
  "resourceTypeUuid": "12345a67-8f0b-123c-45de-6fab78cd01e4"
}' \
"https://openam.example.com:8443/openam/json/realms/root/policies?_action=create"
{
  "name": "mypolicy",
  "active": true,
  "description": "My Policy.",
  "applicationName": "iPlanetAMWebAgentService",
  "actionValues": {
    "POST": false,
    "GET": true
  },
}
```

```
"resources": [
  "http://www.example.com:80/*",
  "http://www.example.com:80/*?*\"",
],
"subject": {
  "type": "Identity",
  "subjectValues": [
    "uid=demo,ou=People,dc=example,dc=com"
  ]
},
"resourceTypeUuid": "12345a67-8f0b-123c-45de-6fab78cd01e4",
"lastModifiedBy": "id=amadmin,ou=user,dc=openam,dc=forgerock,dc=org",
"lastModifiedDate": "2015-05-11T14:48:08.711Z",
"createdBy": "id=amadmin,ou=user,dc=openam,dc=forgerock,dc=org",
"creationDate": "2015-05-11T14:48:08.711Z"
}
```

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

Updating Policies

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

Note

If the realm is not specified in the URL, 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.

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

```
$ curl \
--request PUT \
--header "iPlanetDirectoryPro: AQIC5w..." \
--header "Content-Type: application/json" \
--header "Accept-API-Version: resource=1.0" \
--data '{
  "name": "myupdatedpolicy",
  "active": true,
  "description": "My Updated Policy.",
  "resources": [
    "http://www.example.com:80/*",
    "http://www.example.com:80/*?"
  ],
  "actionValues": {
    "POST": true,
    "GET": true
  },
  "subject": {
    "type": "Identity",
    "subjectValues": [
      "uid=scarter,ou=People,dc=example,dc=com",
      "uid=bjenson,ou=People,dc=example,dc=com"
    ]
  },
  "resourceTypeUuid": "12345a67-8f0b-123c-45de-6fab78cd01e4"
}' \
"https://openam.example.com:8443/openam/json/realms/root/policies/mypolicy"
```

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

Deleting Policies

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

Note

If the realm is not specified in the URL, 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: AQIC5w..." \
--header "Accept-API-Version: resource=2.1" \
--request DELETE \
"https://openam.example.com:8443/openam/json/realms/root/policies/myupdatedpolicy"
```

Copying and Moving Policies

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

- Specify the `_action=copy` or `_action=move` URL parameter.
- Specify the realm in which the input policy resides in the URL. If the realm is not specified in the URL, AM copies or moves a policy from the top level realm.
- Specify the policy to be copied or moved in the URL.
- Specify the SSO token of an administrative user who has access to perform the operation in the `iPlanetDirectoryPro` header.

Specify JSON input data as follows:

JSON Input Data for Copying or Moving Individual Policies

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

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

```
$ curl \
--header "iPlanetDirectoryPro: AQIC5w..." \
--header "Accept-API-Version: resource=2.1" \
--request DELETE \
"https://openam.example.com:8443/openam/json/realms/root/policies/myupdatedpolicy"
{}
```

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

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

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

```
$ curl \
--request POST \
--header "Content-Type: application/json" \
--header "iPlanetDirectoryPro: AQIC5w..." \
--header "Accept-API-Version: resource=2.1" \
--data '{
  "to": {
    "name": "myMovedPolicy",
    "realm": "/myOtherRealm",
    "resourceType": "616b3d02-7a8d-4422-b6a7-174f62afd065"
  }
}' \
"https://openam.example.com:8443/openam/json/realms/root/realms/myRealm/policies/myPolicy?_action=move"
{
  "name": "myMovedPolicy",
  "active": true,
  "description": "",
  "actionValues": {},
  "applicationName": "iPlanetAMWebAgentService",
  "resources": ["*://*/**"],
  "subject": {"type": "NONE"},
  "resourceTypeUuid": "616b3d02-7a8d-4422-b6a7-174f62afd065",
  "lastModifiedBy": "id=amadmin,ou=user,dc=example,dc=com",
  "lastModifiedDate": "2015-12-21T19:32:59.502Z",
  "createdBy": "id=amadmin,ou=user,dc=example,dc=com",
  "creationDate": "2015-12-21T19:32:59.502Z"
}
```

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

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

Specify JSON input data as follows:

JSON Input Data for Copying or Moving Multiple Policies

Object	Property	Description
<code>from</code>	<code>application</code>	The policy set in which the input policies are located.

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

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

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

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

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

```

$ url \
--request POST \
--header "Content-Type: application/json" \
--header "iPlanetDirectoryPro: AQIC5w..." \
--header "Accept-API-Version: resource=2.1" \
--data '{
  "from":{
    "application":"iPlanetAMWebAgentService"
  },
  "to":{
    "realm":"/myOtherRealm",
    "namePostfix":"-copy"
  },
  "resourceTypeMapping":{
    "ccb50c1a-206d-4946-9106-4164e8f2b35b":"616b3d02-7a8d-4422-b6a7-174f62afd065"
  }
}' \
"https://openam.example.com:8443/openam/json/realms/root/realms/myRealm/policies?_action=copy"
{
  "name":"myNewPolicy-copy",
  "active":true,
  "description":"",
  "actionValues":{},
  "applicationName":"iPlanetAMWebAgentService",
  "resources":["*:*/*:*/*:*"], "subject":{"type":"NONE"},
  "resourceTypeUuid":"616b3d02-7a8d-4422-b6a7-174f62afd065",
  "lastModifiedBy":"id=amadmin,ou=user,dc=example,dc=com",
  "lastModifiedDate":"2015-12-21T20:01:42.410Z",
  "createdBy":"id=amadmin,ou=user,dc=example,dc=com",
  "creationDate":"2015-12-21T20:01:42.410Z"
}
    
```

Importing and Exporting XACML 3.0

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

Note

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

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

Policies to XACML Mappings

AM Policy	XACML Policy
Policy Name	Policy ID

AM Policy	XACML Policy
Description	Description
Current Time (yyyy.MM.dd.HH.mm.ss.SSS)	Version
xacml rule target	entitlement excluded resource names
Rule Deny Overrides	Rule Combining Algorithm ID
Any of: <ul style="list-style-type: none"> Entitlement Subject Resource Names Policy Set Names Action Values 	Target
Any of: <ul style="list-style-type: none"> Policy Set Name Entitlement Name Privilege Created By Privilege Modified By Privilege Creation Date Privilege Last Modification Date 	Variable Definitions
Single Level Permit/Deny Actions converted to Policy Rules	Rules

Note

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

Exporting to XACML

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

Policy Set Mappings

AM	XACML
Realm: <timestamp>(yyyy.MM.dd.HH.mm.ss.SSS)	PolicySet ID
Current Time (yyyy.MM.dd.HH.mm.ss.SSS)	Version

AM	XACML
Deny Overrides	Policy Combining Algorithm ID
No targets defined	Target

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

```
https://openam.example.com:8443/openam/xacml/policies
https://openam.example.com:8443/openam/xacml/{realm}/policies

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

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

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

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

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

XACML Export Search Filter Format

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

To Export Policies

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

```
$ curl \
--request GET \
--header "iPlanetDirectoryPro: AQIC5..." \
"https://openam.example.com:8443/openam/xacml/policies"
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<PolicySet xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
  PolicyCombiningAlgId="urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:deny-overrides"
  Version="2014.10.08.21.59.39.231" PolicySetId="/:2014.10.08.21.59.39.231">
  <Target/>
  <Policy RuleCombiningAlgId="urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:deny-overrides"
    Version="2014.10.08.18.01.03.626"
    PolicyId="Rockshop_Checkout_https://forgerock-rockshop.openrock.org:443/wp-login.php*?*">
    ...
```

To Export Policies with Search Filters

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

```
$ curl \
--request GET \
--header "iPlanetDirectoryPro: AQIC5..." \
"https://openam.example.com:8443/openam/xacml/policies?filter=createdby=amadmin"
```

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

```
$ curl \
--request GET \
--header "iPlanetDirectoryPro: AQIC5..." \
"https://openam.example.com:8443/openam/xacml/policies?filter=createdby=amadmin&filter=creationdate=135563832"
```

Importing from XACML

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

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

```
https://openam.example.com:8443/openam/xacml/policies
https://openam.example.com:8443/openam/xacml/{realm}/policies
```

where {realm} is the name of a specific realm

To Import a XACML 3.0 Policy

1. You can do a dry run using the `dryrun=true` query to test the import. The dry-run option outputs in JSON format and displays the status of each import policy, where "U" indicates "Updated"; "A" for

"Added". The dry-run does not actually update to the database. When you are ready for an actual import, you need to re-run the command without the `dryrun=true` query.

```
$ curl \
--request POST \
--header "Content-Type: application/xml" \
--header "iPlanetDirectoryPro: AQIC5..." \
--data @xacml-policy.xml \
"https://openam.example.com:8443/openam/xacml/policies?dryrun=true"
[
  {
    "status": "A",
    "name": "aNewPolicy"
  },
  {
    "status": "U",
    "name": "anExistingPolicy"
  },
  {
    "status": "U",
    "name": "anotherExistingPolicy"
  }
]
```

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

```
$ curl \
--request POST \
--header "Content-Type: application/xml" \
--header "iPlanetDirectoryPro: AQIC5..." \
--data @xacml-policy.xml \
"https://openam.example.com:8443/openam/xacml/policies"
```

Tip

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

```
$ curl \
--request POST \
--header "Content-Type: application/xml" \
--header "iPlanetDirectoryPro: AQIC5..." \
--data @xacml-policy.xml \
"https://openam.example.com:8443/openam/xacml/{realm}/policies"
```

Managing Environment Condition Types

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

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

- "Querying Environment Condition Types"
- "Reading a Specific Environment Condition Type"

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

`Script` is the only environmental condition available for OAuth 2.0 policies. Use scripts to capture the `ClientId` environmental attribute.

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

```
{
  "title": "IPv4",
  "logical": false,
  "config": {
    "type": "object",
    "properties": {
      "startIp": {
        "type": "string"
      },
      "endIp": {
        "type": "string"
      },
      "dnsName": {
        "type": "array",
        "items": {
          "type": "string"
        }
      }
    }
  }
}
```

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

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

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

```

{
  "title" : "NOT",
  "logical" : true,
  "config" : {
    "type" : "object",
    "properties" : {
      "condition" : {
        "type" : "object",
        "properties" : {
        }
      }
    }
  }
}
    
```

The concrete NOT condition therefore takes the following form.

```

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

The OR condition takes an array of conditions.

```

{
  "title" : "OR",
  "logical" : true,
  "config" : {
    "type" : "object",
    "properties" : {
      "conditions" : {
        "type" : "array",
        "items" : {
          "type" : "any"
        }
      }
    }
  }
}
    
```

A corresponding concrete OR condition thus takes the following form.

```

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

Querying Environment Condition Types

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

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

```
$ curl \
--header "iPlanetDirectoryPro: AQIC5..." \
--header "Accept-API-Version: resource=1.0, protocol=2.1" \
https://openam.example.com:8443/openam/json/realms/root/conditiontypes?_queryFilter=true
{
  "result" : [
    {
      "title": "IPv4",
    }
    "logical": false,
    "config": {
      "type": "object",
      "properties": {
        "startIp": {
          "type": "string"
        },
        "endIp": {
          "type": "string"
        },
        "dnsName": {
          "type": "array",
          "items": {
            "type": "string"
          }
        }
      }
    }
  },
  {
    "title": "NOT",
    "logical": true,
    "config": {
      "type": "object",
      "properties": {
        "condition": {
          "type": "object",
          "properties": { }
        }
      }
    }
  },
  {...},
  {...},
  {...}
],
"resultCount" : 18,
"pagedResultsCookie" : null,
"remainingPagedResults" : 0
}
```

Additional query strings can be specified to alter the returned results. For more information, see "Query".

Reading a Specific Environment Condition Type

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

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

```
$ curl \
--header "iPlanetDirectoryPro: AQIC5..." \
--header "Accept-API-Version: resource=1.0" \
https://openam.example.com:8443/openam/json/realms/root/conditiontypes/IPv4
{
  "title": "IPv4",
  "logical": false,
  "config": {
    "type": "object",
    "properties": {
      "startIp": {
        "type": "string"
      },
      "endIp": {
        "type": "string"
      },
      "dnsName": {
        "type": "array",
        "items": {
          "type": "string"
        }
      }
    }
  }
}
```

Managing Subject Condition Types

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

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

- "Querying Subject Condition Types"
- "Reading a Specific Subject Condition Type"

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

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

```
{
  "title" : "Identity",
  "logical" : false,
  "config" : {
    "type" : "object",
    "properties" : {
      "subjectValues" : {
        "type" : "array",
        "items" : {
          "type" : "string"
        }
      }
    }
  }
}
```

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

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

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

```
{
  "title" : "AND",
  "logical" : true,
  "config" : {
    "type" : "object",
    "properties" : {
      "subjects" : {
        "type" : "array",
        "items" : {
          "type" : "any"
        }
      }
    }
  }
}
```

The concrete AND subject condition therefore takes the following form.

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

Querying Subject Condition Types

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

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

```
$ curl \
--header "iPlanetDirectoryPro: AQIC5..." \
--header "Accept-API-Version: resource=1.0" \
https://openam.example.com:8443/openam/json/realms/root/subjecttypes?_queryFilter=true
{
  "result" : [
    {
      "title": "JwtClaim"
    },
    {
      "logical": false,
      "config": {
        "type": "object",
        "properties": {
          "claimName": {
            "type": "string"
          },
          "claimValue": {
            "type": "string"
          }
        }
      }
    },
    {
      "title": "NOT",
      "logical": true,
      "config": {
        "type": "object",
        "properties": {
          "subject": {
            "type": "object",
            "properties": { }
          }
        }
      }
    },
    {...},
    {...},
    {...}
  ]
}
```

```

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

```

Additional query strings can be specified to alter the returned results. For more information, see "Query".

Reading a Specific Subject Condition Type

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

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

```

$ curl \
--header "iPlanetDirectoryPro: AQIC5..." \
--header "Accept-API-Version: resource=1.0" \
https://openam.example.com:8443/openam/json/realms/root/subjecttypes/Identity
{
  "title" : "Identity",
  "logical" : false,
  "config" : {
    "type" : "object",
    "properties" : {
      "subjectValues" : {
        "type" : "array",
        "items" : {
          "type" : "string"
        }
      }
    }
  }
}

```

Managing Subject Attributes

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

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

- "Querying Subject Attributes"

Subject attributes derive from the list of LDAP user attributes configured for the Identity data store. For more information, see "Setting Up Identity and Data Stores" in the *Setup and Maintenance Guide*.

Querying Subject Attributes

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

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

```
$ curl \
--header "iPlanetDirectoryPro: AQIC5..." \
--header "Accept-API-Version: resource=1.0" \
https://openam.example.com:8443/openam/json/realms/root/subjectattributes/?_queryFilter=true
{
  "result" : [
    "sunIdentityServerPPInformalName",
    "sunIdentityServerPPFacadeGreetSound",
    "uid",
    "manager",
    "sunIdentityServerPPCommonNameMN",
    "sunIdentityServerPPLegalIdentityGender",
    "preferredLocale",
    "...",
    "...",
    "...",
    "...",
  ],
  "resultCount": 87,
  "pagedResultsCookie": null,
  "remainingPagedResults": 0
}
```

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

Managing Decision Combiners

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

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

- "Querying Decision Combiners"
- "Reading a Specific Decision Combiner"

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

Querying Decision Combiners

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

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


```
$ curl \
--header "iPlanetDirectoryPro: AQIC5..." \
--header "Accept-API-Version: resource=1.0, protocol=2.1" \
https://openam.example.com:8443/openam/json/realms/root/decisioncombiners?_queryFilter=true
{
  "result":[
    {
      "title":"DenyOverride"
    }
  ],
  "resultCount":1,
  "pagedResultsCookie":null,
  "remainingPagedResults":0
}
```

Additional query strings can be specified to alter the returned results. For more information, see "Query".

Reading a Specific Decision Combiner

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

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

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

```
$ curl \
--header "iPlanetDirectoryPro: AQIC5..." \
--header "Accept-API-Version: resource=1.0" \
https://openam.example.com:8443/openam/json/realms/root/decisioncombiners/DenyOverride
{
  "title" : "DenyOverride"
}
```

Implementing OAuth 2.0 Policies

Writing policies may not be straightforward if your environment requires complex conditions. The easiest way to test if your OAuth 2.0 policies are granting or denying the scopes you expect before setting them in production is to configure AM as an OAuth 2.0 client and authorization provider and request some tokens.

This section provides examples to test an OAuth 2.0 policy with an interactive OpenID Connect flow and a non-interactive OAuth 2.0 flow. Before you start, configure your environment as explained in "To Configure AM for the Examples".

Tip

Read Granting or Denying OAuth 2.0 Scopes Through Policy Decisions before trying the examples in this section.

To Configure AM for the Examples

To configure your environment for the OAuth 2.0/OpenID Connect examples, perform the following steps:

1. Navigate to Realms > *Realm Name* > Dashboard and select Configure OAuth Provider.
2. Select Configure OpenID Connect. The wizard configures an OAuth 2.0 provider with the basic configuration for OpenID Connect.

Note

This step will overwrite any OAuth 2.0 provider you have configured for this realm.

3. Accept the default configuration and select Create.
4. Navigate to Realms > *Realm Name* > Services and select OAuth2 Provider.
5. On the Core tab, enable Use Policy Engine for Scope Decisions and save your changes.
6. Navigate to Realms > *Realm Name* > Applications > OAuth 2.0 and add a client. Configure the client properties as follows:
 - Client ID: `myClient`
 - Client secret: `forgerock`
 - Redirection URIs: `https://www.example.com:443/callback`
 - Scope(s): `profile email openid`
 - Grant Types: `Resource Owner Password Credentials`
7. Navigate to Realms > *Realm Name* > Authorization > Policy Sets and select Default OAuth2 Scopes Policy Set.
8. Create a new policy as follows:
 - Name: `myOAuth2Policy`
 - Resource Type: `OAuth2 Scope`
 - Resources: Select the * resource pattern and replace it with `email`.

Once the policy is created, you will see the Summary tab.

9. On the Actions tab, add the **GRANT** action and set the default state to **Allow**. Save your changes.
10. On the Subjects tab, specify the subject condition **All of... Authenticated Users**. Save your changes.
11. (Optional) If the **demo** user is not present in your environment, create it by performing the following step:
 - Navigate to Realms > *Realm Name* > Identities, and add a new user called **demo** with password **changeit**.

To Test OAuth 2.0 Policies in a Non-Interactive Flow

This procedure shows how to test OAuth 2.0 policies with the ROPC Grant flow. In this flow, the resource owner does not interact with the consent screen and, therefore, cannot grant scopes.

1. Request an access token from AM by specifying:
 - The client name, **myClient**, and its password, **forgerock**
 - The Resource Owner Password Credentials grant type, **password**
 - The user and password to authenticate with, **demo** and **changeit**
 - The requested scopes, **email** and **profile**

```
$ curl --request POST \  
--data "grant_type=password" \  
--data "username=demo" \  
--data "password=changeit" \  
--data "scope=profile email" \  
--data "client_id=myClient" \  
--data "client_secret=forgerock" \  
"https://openam.example.com:8443/openam/oauth2/realms/root/access_token"
```

2. Review the JSON response. It should be similar to the following:

```
{  
  "access_token": "B78LPVHaycIr0bh12Qps0n9ynYM",  
  "scope": "email",  
  "token_type": "Bearer",  
  "expires_in": 3599  
}
```

Note how the requested scopes were **email** and **profile**, but the scope granted was **email**, which matches the **GRANT=Allow** action defined in the policy.

To Test OAuth 2.0 Policies in an Interactive OpenID Connect Flow

This procedure shows how to test OAuth 2.0 policies with the OpenID Connect Implicit Grant flow. In this flow, the end user can interact with the consent screen and, therefore, can grant scopes.

1. In a web browser, make a call to the `oauth2/authorize` endpoint with the following parameters:

- `nonce=123`
- `state=456`
- `scope=openid+email+profile`
- `response_type=id_token`
- `client_id=myClient`
- `redirect_uri=https://www.example.com:8443/callback`

For example:

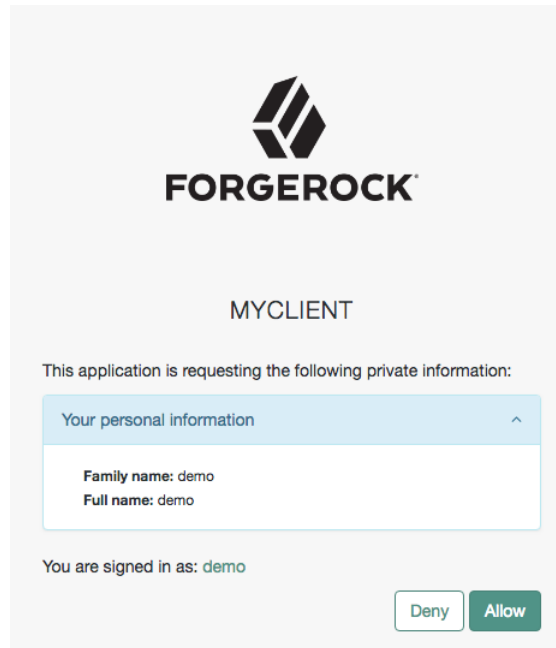
```
https://openam.example.com:8443/openam/oauth2/authorize?nonce=123&state=456&scope=openid+email+profile&response_type=id_token&client_id=myClient&redirect_uri=https://www.example.com:8443/callback
```

You will be prompted to log in to AM.

2. Log in as the `demo` user.

You will be prompted to provide consent for the `profile` scope:

Requesting Consent for the Profile Scope



FORGEROCK

MYCLIENT

This application is requesting the following private information:

Your personal information ^

Family name: demo
Full name: demo

You are signed in as: demo

Deny Allow

Notice that you are not prompted to provide consent for the `email` scope.

3. Allow the flow to continue.
4. Review the URL of the browser. It should show something similar to:

```
http://example.com/#scope=openid%20profile%20email&id_token=eyJ0eXJKV1Q...8WK1eg&state=456
```

Notice that the `email` scope has been granted automatically.

If the authorization policy had been configured as `GRANT=Deny`, you still would have not seen the `email` scope in the consent page, but the scope would not appear in the URL of the browser.

Chapter 3

Implementing Transactional Authorization

This chapter covers transactional authorization. Transactional authorization makes use of the session upgrade functionality, applying it while authorization is in progress. For information on session upgrade, see "About Authentication Levels" in the *Authentication and Single Sign-On Guide*.

Transactional authorization improves security by requiring a user to perform additional actions when trying to access a resource protected by an AM policy. For example, they must reauthenticate to an authentication module or respond to a push notification on their mobile device.

Performing the additional action successfully grants access to the protected resource, but only once. Additional attempts to access the resource will require the user to perform the configured actions again.

Introducing Transactional Authorization

Transactional authorization is implemented as a new environment condition type that can be added to authorization policies. For more information on policies, see "Resource Types, Policy Sets, and Policies".

The transactional authorization environment condition can be combined in policies with the other conditions, for example, only requiring a push notification response when access is attempted to the employees subrealm but outside usual working hours, as shown below:

Combining With Other Environment Conditions

Summary
Resources
Actions
Subjects
Environments

Specify the environment conditions to which the policy applies.

All of...

Type	Authenticate to a Realm	✎ ✕
	Authentication to a Realm /employees	

Type	Authentication Strategy	Strategy Specifier	✎ ✕
Transaction	Authenticate To Chain	pushService	

Not...

Type	Start Time	End Time	Start Day	End Day	Time Zone	✎ ✕
Time (day, date, time, and timezone)	09:00	17:00	mon	fri	GMT	

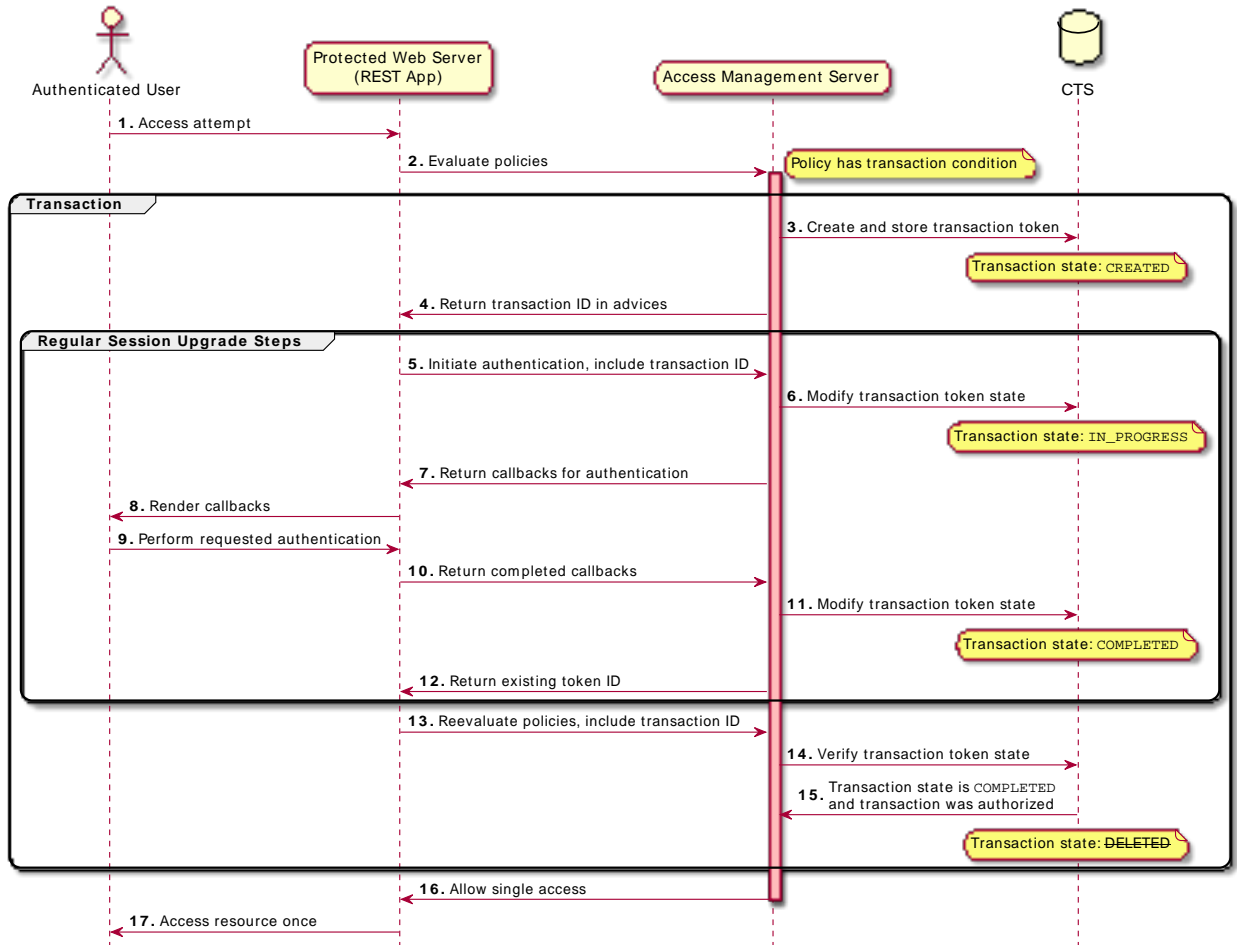
+ Add an Environment Condition

+ Add a Logical Operator

Save Changes

The following diagram describes the sequence of events that occur when accessing a resource that is protected by a REST application, and an AM policy containing a transactional environment condition:

Accessing Resources with Transactional Authorization



The sequence of events for a transaction authorization is as follows:

1. An authenticated user attempts to access a resource that is protected by an AM server.
2. The resource server contacts AM to evaluate the policies that apply.

The resource server can be protected with ForgeRock's Web or Java Agents, which support transactional authorization natively, or a custom application that uses ForgeRock's REST API as per the diagram to manage the transactional authorization.

3. As the policy contains a transaction environment condition, AM creates a transaction token in the Core Token Service (CTS) store. The initial transaction token state is set to **CREATED**.

The transaction token contains information about the policy evaluation, including the:

- Realm
- Resource
- Subject
- Audit tracking ID
- Authentication method

To protect against tampering, AM verifies that these details do not change and match those in the incoming requests for the duration of the transaction.

The transaction token has a time-to-live (default 180 seconds) defined in the Transaction Authentication Service. If the transaction is not completed in this time the token is deleted, and the flow will need to be restarted. Alter the default if the transaction includes authentication actions that take more time to complete, for example, using the HOTP authentication module for one-time password over email.

The time-to-live can be configured globally, or per-realm. See "Transaction Authentication Service".

4. In the JSON response to the policy evaluation request, AM returns the transaction ID—the unique ID of the newly created transaction token—in the **TransactionConditionAdvice** array in the **advices** object:

```
{
  "resource": "http://www.example.com:8000/index.html",
  "actions": {},
  "attributes": {},
  "advices": {
    "TransactionConditionAdvice": [
      "7b8bfd4c-60fe-4271-928d-d09b94496f84"
    ]
  },
  "ttl": 0
}
```

5. As the JSON response to the evaluation does not grant any actions but does contain advices, the REST application on the resource server extracts the transaction ID and returns it to the authentication service to commence the authentication.

The transaction ID is included in the **TransactionConditionAdvice** attribute value pair in the composite advice query parameters sent as part of the request for actions.

ForgeRock web and Java agents manage this interaction natively. For information on using the REST API to handle **advices** elements in policy evaluations, see "Requesting Policy Decisions".

- AM extracts the transaction ID from the composite advice, verifies the corresponding transaction token, and changes the state to `IN_PROGRESS`.

If the transaction ID is not in the expected state or does not exist, a `401 Unauthorized` error is returned. For example:

```
{
  "code": 401,
  "reason": "Unauthorized",
  "message": "Unable to read transaction.",
  "detail": {
    "errorCode": "128"
  }
}
```

- AM responds with the callbacks necessary to satisfy any environment conditions.

Note

The advices returned by transaction environment conditions have the lowest precedence when compared to the other condition advices. end users will have to complete the non-transactional condition advices before they can complete the transactional condition advices.

- The REST application renders the callbacks and presents them to the user.
- The user completes the required actions, for example authenticates to the specified chain, or responds to the push notification on their registered mobile device.

Tip

If the user is unable to complete the actions, AM returns an HTTP 200 message and the user is redirected to the protected resource. Policy evaluation will fail since the transactional authorization process has failed.

To return an HTTP 401 message and redirect the user to the failure URL, configure the `org.forgerock.openam.auth.transactionalauth.returnErrorOnAuthFailure` advanced server property.

- The REST app completes the callbacks and returns the result to AM.
- AM verifies the transaction token, and changes the state to `COMPLETED`.
- With the transaction now complete, AM returns the original token.

Note that the authentication performed as part of an authorization flow does not behave exactly the same as a standard authentication. The differences are:

- The user's original session is not upgraded or altered in any way.
- Failing the authentication during the authorization flow does not increment account lockout counters.

13. The web or Java agent or custom application on the resource server can re-evaluate the policies applying to the protected resources again, but includes the ID of the completed transaction as a value in the `TxId` array in the `environment` object:

```
{
  "resources" : ["http://www.example.com:8000/index.html"],
  "application" : "iPlanetAMWebAgentService",
  "subject" : {
    "ssoToken" : "AQIC5w...*AJTMQAA*"
  },
  "environment": {
    "TxId": ["7b8bfd4c-60fe-4271-928d-d09b94496f84"]
  }
}
```

14. AM verifies the transaction was authorized and that the transaction token is in the `COMPLETED` state.
15. If the transaction was completed successfully, authorization continues. The transaction token is marked for deletion, so that it cannot be used to grant more than a single access.
16. As the authentication required to complete the transaction was successful, AM returns the result of the policy reevaluation. For example, the following response grants the `POST` and `GET` actions to the resource `http://www.example.com:8000/index.html`:

```
{
  "resource": "http://www.example.com:8000/index.html",
  "actions": {
    "POST": true,
    "GET": true
  },
  "attributes": {},
  "advices": {},
  "ttl": 0
}
```

Important

Successful transactional authorization responses set the time-to-live (`ttl`) value to zero to ensure that the policy decision is not cached and cannot be used more than once.

ForgeRock agents prior to version 5 do not support a time-to-live value of zero and cannot be used for transactional authorization.

17. The user is able to access the protected resource once. Additional attempts to access a resource protected with a policy containing a transactional environment condition require a new transaction to be completed.

Using Transactional Authorization

This section demonstrates how to set up transactional authorization to send push notifications to a user's mobile device to authorize access to a protected resource.

Transactional Authorization Prerequisite Tasks

Before attempting this demonstration you must perform the following prerequisite tasks:

- Create an authentication chain containing the [ForgeRock Authenticator \(PUSH\) Registration](#) authentication module. Log in to that chain as the [demo](#) user and register a mobile device using the ForgeRock Authenticator application.

See Procedure 4.5: "To Create an Authentication Chain for Push Registration and Passwordless Authentication" and "Registering the ForgeRock Authenticator for Multi-Factor Authentication" in the *Authentication and Single Sign-On Guide*.

- Set up the Push Notification Service in AM with valid credentials.

For information on provisioning the credentials required by the Push Notification Service, see [How To Configure Service Credentials \(Push Auth, Docker\) in Backstage](#) in the *ForgeRock Knowledge Base*.

For detailed information about Push Notification Service properties, see "Push Notification Service".

- Perform at least one of the following steps:
 - To use the AM console for the demonstration, set up a web agent to protect web resources. See "[First Steps](#)" in the *Quick Start Guide*.
 - To use the AM REST API for the demonstration, create a user account that has read access to the policy endpoints. By default, users do not have permissions to access the policy evaluation endpoints directly. To allow access to the policy REST endpoints, follow the steps in "To Allow a User to Evaluate Policies".

After completing the prerequisite tasks, proceed to the steps outlined in "To Prepare AM for Transactional Authorization with Push Notifications".

Preparing AM for Transactional Authorization with Push Notifications

Perform the following steps to set up an authorization policy with a transaction environment condition, which requires users to respond to a push notification message on their registered mobile device to authorize access to a protected resource.

Ensure you have completed the steps outlined in "Transactional Authorization Prerequisite Tasks" before proceeding.

To Prepare AM for Transactional Authorization with Push Notifications

1. Add a ForgeRock Authenticator (Push) authentication module:
 - a. Log in as an AM administrator, for example, [amadmin](#).

- b. Navigate to Realms > Top Level Realm > Authentication > Modules, and then select Add Module.
- c. On the New Module page, name the module `pushAuth`, select ForgeRock Authenticator (Push) as the module type, and then select Create.
- d. (Optional) Alter the Login Message. For example:

```
Authorize {{user}} at {{issuer}}?
```

- e. Select Save Changes
2. Add the module to an authentication chain:
 - a. Navigate to Realms > Top Level Realm > Authentication > Chains, and then select Add Chain.
 - b. On the Add Chain page, name the chain `pushAuthChain`, and then select Create.
 - c. On the Edit Chain tab, select Add a Module.
 - d. On the New Module dialog, from the Select Module drop-down list, select the push module you created in the earlier step, for example, `pushAuth`, from the Select Criteria drop-down list, select Required, and then select OK.
 - e. On the Edit Chain tab, select Save Changes.
 3. Create an authorization policy as described in "To Configure a Policy" in the *Quick Start Guide*

Note

Ensure that there are not multiple policies which match the protected resource that is being authorized. If multiple policies have a matching subject or environment conditions with the protected resource, it may be incorrectly authorized.

Make the following changes to the policy:

- a. Navigate to Realms > Top Level Realm > Authorization > Policy Sets, and then select Default Policy Set.
- b. On the Default Policy Set page, select `Authenticated users can get Apache HTTP home page`.
- c. On the Environments tab, select Add an Environment Condition, and then select Transaction.
- d. From the Authentication Strategy drop-down list, select Authenticate to Chain.
- e. In the Strategy Specifier field, enter the name of the push authorization chain created earlier, for example, `pushAuthChain`.

Note

The value entered must exactly match the name of the chain. The value is not validated by the UI, and an incorrect value will cause the authorization to fail.

- f. Select the checkmark icon, and then select Save Changes.

The resulting policy will resemble the following image:

Transaction Environment Condition in a Policy

The screenshot displays the 'POLICY | Active' configuration page. The main title is 'Authenticated users can get Apache HTTP home page'. Below the title are tabs for 'Summary', 'Resources', 'Actions', 'Subjects', and 'Environments'. The 'Summary' tab is selected, showing four sections: 'Resources' with the URL 'http://www.example.com:8000/*', 'Actions' with 'POST: Allowed' and 'GET: Allowed', 'Response Attributes' with a '+ Add Response Attributes' button, and 'Subjects' with a JSON configuration:

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

. The 'Environments' section is also visible, showing a JSON configuration:

```
{
  "type": "Transaction",
  "authenticationStrategy": "Authentic",
  "strategySpecifier": "pushAuthChain"
}
```

When completed, choose one of the following options for the next step:

- To use the AM console for the demonstration, proceed to the steps outlined in "Using Transactional Authorization with the AM Console".
- To use the AM REST API for the demonstration, proceed to the steps outlined in "Using Transactional Authorization with the REST APIs".

Using Transactional Authorization with the AM Console

This section describes how to use the AM console to perform a transactional authorization that sends a push notification to a mobile device.

Ensure you have completed the steps outlined in "To Prepare AM for Transactional Authorization with Push Notifications" before proceeding.

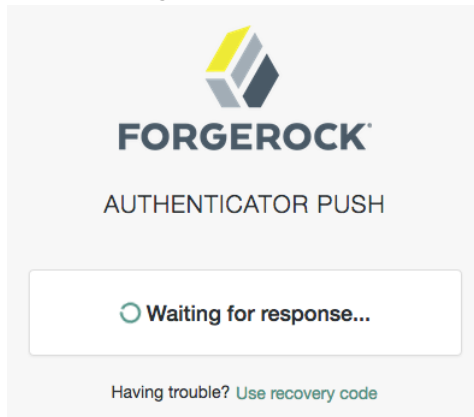
To Perform Transactional Authorization with the AM Console

1. In a web browser, navigate to a URL that is protected by the policy you edited in "To Prepare AM for Transactional Authorization with Push Notifications", such as <http://www.example.com:8000/index.html>

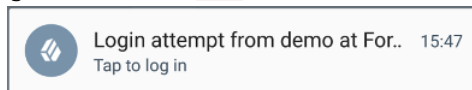
The web agent will redirect the browser to the AM login screen.

2. Log in to AM as user `demo` with password `changeit`.

AM will display the authenticator push page:

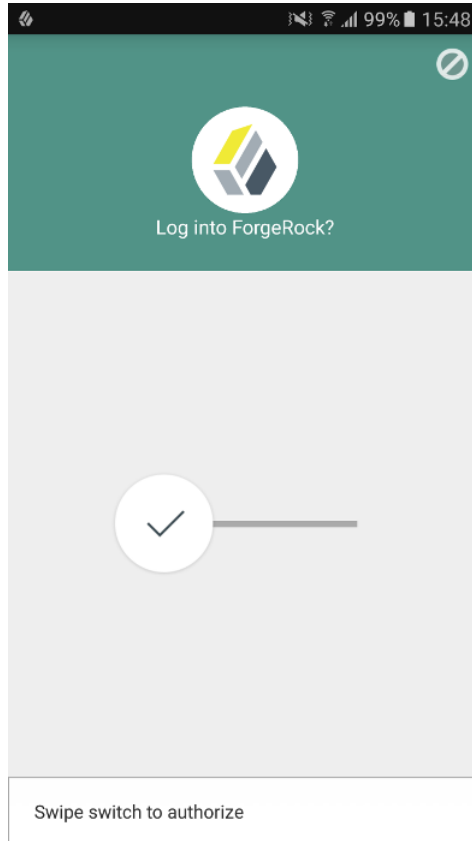


The mobile device that was registered to the `demo` user will receive a push notification message:



3. On the registered mobile device, tap the notification.

The ForgeRock Authenticator app will open. Swipe the switch to authorize the access attempt.



After authorizing the request in the ForgeRock Authenticator app, the authenticator push page in the web browser redirects to the requested resource, completing the transactional authorization.

Note that refreshing the protected page in the web browser at this point starts a new transactional authorization flow, and send a new push notification.

Using Transactional Authorization with the REST APIs

This section describes how to use the AM REST API to perform a transactional authorization that sends a push notification to a mobile device.

Ensure you have completed the steps outlined in "To Prepare AM for Transactional Authorization with Push Notifications" before proceeding.

To Perform Transactional Authorization with the AM REST API

1. Obtain a session token from AM for user `demo` with password `changeit`:

```
$ 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": "/"
}
```

2. Request a policy evaluation with the `tokenId` from the previous step as the subject, and a resource URL that is protected by the policy you edited in "To Prepare AM for Transactional Authorization with Push Notifications", such as `http://www.example.com:8000/index.html`.

Note

The request requires authentication as a user with the privileges to access the policy endpoints, for example by specifying the SSO token ID in the `iPlanetDirectoryPro` cookie. See "Authentication and Logout using REST".

```
$ curl \
--cookie "iPlanetDirectoryPro=AQIC5wM2L...zEAAA.*" \
--request POST \
--header "Content-Type: application/json" \
--header "Accept-API-Version: resource=2.0" \
--data '{
  "resources" : ["http://www.example.com:8000/index.html"],
  "subject" : {
    "ssoToken" : "<AQIC5w...NTcy*"
  }
}' \
https://openam.example.com:8443/openam/json/policies/?_action=evaluate
{
  "resource": "http://www.example.com:8000/index.html",
  "actions": {},
  "attributes": {},
  "advices": {
    "TransactionConditionAdvice": [
      "9dae2c80-fe7a-4a36-b57b-4fb1271b0687"
    ]
  },
  "ttl": 0
}
```

AM returns an empty `actions` element, and a transaction ID in the `TransactionConditionAdvice` property, because a transactional authorization is required to access the resource.

3. Initiate authentication, and include the transaction ID in the composite advice. Note that the steps used for performing a transactional authorization are identical to performing a session upgrade. See "Session Upgrade" in the *Authentication and Single Sign-On Guide*.

The transaction ID returned in the previous step must be returned as composite advice query parameters, wrapped in URL-encoded XML. The XML format is as follows:

```
<Advices>
  <AttributeValuePair>
    <Attribute name="TransactionConditionAdvice"/>
    <Value>Transaction Id</Value>
  </AttributeValuePair>
</Advices>
```

Use the SSO token of the **demo** user for this request.

Note that the following **curl** command URL-encodes the XML values, and the **-G** parameter appends them as query string parameters to the URL:

```
$ curl -get \
--cookie "iPlanetDirectoryPro=AQIC5w...NTcy*" \ \
--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="TransactionConditionAdvice"/>
    <Value>9dae2c80-fe7a-4a36-b57b-4fb1271b0687</Value>
  </AttributeValuePair>
</Advices>' \
'https://openam.example.com:8443/openam/json/realms/root/authenticate'
{
  "authId": "eyJ0eXAiOi...WLxJ-1d6ovYKHQ",
  "template": "",
  "stage": "AuthenticatorPush3",
  "header": "Authenticator Push",
  "callbacks": [
    {
      "type": "PollingWaitCallback",
      "output": [
        {
          "name": "waitTime",
          "value": "10000"
        }
      ]
    },
    {
      "type": "ConfirmationCallback",
      "output": [
        {
          "name": "prompt",
          "value": ""
        },
        {
          "name": "messageType",
          "value": 0
        }
      ]
    }
  ]
}
```

```

    },
    {
      "name": "options",
      "value": [
        "Use Emergency Code"
      ]
    },
    {
      "name": "optionType",
      "value": -1
    },
    {
      "name": "defaultOption",
      "value": 0
    }
  ],
  "input": [
    {
      "name": "IDToken2",
      "value": 100
    }
  ]
}
]
}
}

```

At this point, the mobile device that was registered to the `demo` user will receive a push notification message, that they should authorize in the ForgeRock Authenticator app.

4. Ensure that the time specified in the `waitTime` property in the callbacks has passed, in this case at least 10 seconds, and then complete and return the requested callbacks.

The value of the `authId` property must also be returned, as well as the URL-encoded transaction ID.

Use the SSO token of the `demo` user for this request.

Note

In this example, the required XML parameters have been URL-encoded and added to the URL. The `curl` command is not able to use the `--data-urlencode` option for query-string parameters and also send a JSON payload.

```

$ curl \
--request POST \
--header "Accept-API-Version: resource=2.0, protocol=1.0" \
--header "Content-Type: application/json" \
--data '{
  "authId": "eyJ0eXAiOiJK...u4WvZmiI",
  "callbacks": [
    {
      "type": "PollingWaitCallback",
      "output": [
        {

```

```

        "name": "waitTime",
        "value": "10000"
    },
    {
        "name": "message",
        "value": "Waiting for response..."
    }
]
}
]
}' 'https://openam.example.com:8443/openam/json/realms/root/authenticate?
authIndexType=service&authIndexValue=Example'
#authenticate-indextype

#authenticate-xml
curl -get \
--cookie "iPlanetDirectoryPro=AQIC5w..NTcy*" \ \
--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="TransactionConditionAdvice"/>
  <Value>9dae2c80-fe7a-4a36-b57b-4fb1271b0687</Value>
</AttributeValuePair>
</Advices>' \
'https://openam.example.com:8443/openam/json/realms/root/authenticate'
#authenticate-xml

cat << EOF
#authenticate-xml-expected
{
  "authId": "eyJ0eXAiOi...WLxJ-1d6ovYKHQ",
  "template": "",
  "stage": "AuthenticatorPush3",
  "header": "Authenticator Push",
  "callbacks": [
    {
      "type": "PollingWaitCallback",
      "output": [
        {
          "name": "waitTime",
          "value": "10000"
        }
      ]
    },
    {
      "type": "ConfirmationCallback",
      "output": [
        {
          "name": "prompt",
          "value": ""
        },
        {
          "name": "messageType",
          "value": 0
        }
      ]
    }
  ]
}

```

```

        "name": "options",
        "value": [
            "Use Emergency Code"
        ]
    },
    {
        "name": "optionType",
        "value": -1
    },
    {
        "name": "defaultOption",
        "value": 0
    }
],
"input": [
    {
        "name": "IDToken2",
        "value": 100
    }
]
}
]
}
}
#authenticate-xml-expected
EOF

#authenticate-indextype
curl \
--cookie "iPlanetDirectoryPro=AQIC5w...NTcy*" \ \
--request POST \
--header "Content-Type: application/json" \
--header "Accept-API-Version: resource=2.0, protocol=1.0" \
--data '{
    "authId":"eyJ0eXAiOi...WLxJ-1d6ovYKHQ",
    "template":"","
    "stage":"AuthenticatorPush3",
    "header":"Authenticator Push",
    "callbacks":[
        {
            "type":"PollingWaitCallback",
            "output":[
                {
                    "name":"waitTime",
                    "value":"10000"
                }
            ]
        },
        {
            "type":"ConfirmationCallback",
            "output":[
                {
                    "name":"prompt",
                    "value":""
                },
                {
                    "name":"messageType",
                    "value":0
                }
            ]
        }
    ]
}'

```

```

    {
      "name": "options",
      "value": [
        "Use Emergency Code"
      ]
    },
    {
      "name": "optionType",
      "value": -1
    },
    {
      "name": "defaultOption",
      "value": 0
    }
  ],
  "input": [
    {
      "name": "IDToken2",
      "value": 100
    }
  ]
}
]
}' \
"https://openam.example.com:8443/openam/json/realms/root/authenticate\
?authIndexType=composite_advice\
&authIndexValue=%3CAdvises%3E%0A\
%3CAttributeValuePair%3E%0A%3CAttribute%20name%3D\
%22TransactionConditionAdvice%22%2F%3E%0A\
%3CValue%3E9dae2c80-fe7a-4a36-b57b-4fb1271b0687\
%3C%2FValue%3E%0A%3C%2FAttributeValuePair\
%3E%0A%3C%2FAdvises%3E"
{
  "tokenId": "AQIC5w...NTcy*",
  "successUrl": "http://www.example.com:8000/index.html",
  "realm": "/"
}

```

If the callbacks were correctly completed, and the push notification was responded to in the ForgeRock Authenticator app, AM returns the original `tokenId` value.

If the push notification has not yet been responded to in the ForgeRock Authenticator app, AM will return the required callbacks again, as in the previous step. Wait until the amount of time specified in the `waitTime` element has passed and retry the request until the `tokenId` returns.

5. Re-evaluate the policy, including the transaction ID as the value of a `TxId` property in the `environment` element:

```
$ curl \
--cookie "iPlanetDirectoryPro=AQIC5wM2L...zEAAA.*" \ \
--request POST \
--header "Content-Type: application/json" \
--header "Accept-API-Version: resource=1.0" \
--data '{
  "resources" : ["http://www.example.com:8000/index.html"],
  "subject" : {
    "ssoToken" : "AQIC5w...NTcy*"
  },
  "environment": {
    "TxId": ["9dae2c80-fe7a-4a36-b57b-4fb1271b0687"]
  }
}' \
"https://openam.example.com:8443/openam/json/policies/?_action=evaluate"
{
  "resource": "http://www.example.com:8000/index.html",
  "actions": {
    "POST": true,
    "GET": true
  },
  "attributes": {
  },
  "advices": {
  },
  "ttl": 0
}
```

As the authentication required by the transaction was successful, the second policy evaluation returns the **POST** and **GET** actions as defined in the policy.

Notice that the time-to-live (**ttl**) value of the policy evaluation result is set to **0**, meaning that the policy must not be cached. The policy only allows a single access to the resource, which must be managed by the policy enforcement point.

Note that performing the policy evaluation again with the same subject and resource at this point starts a new transactional authorization flow, requiring each of the steps above to be repeated in order to access the protected resource each time.

Chapter 4

Customizing Authorization

This chapter describes how to customize policy evaluation by writing a policy plug-in and by scripting a customized policy condition.

Customizing Policy Evaluation With a Plug-In

AM policies let you restrict access to resources based both on identity and group membership, and also on a range of conditions including session age, authentication chain or module used, authentication level, realm, session properties, IP address and DNS name, user profile content, resource environment, date, day, time of day, and time zone. Yet, some deployments require further distinctions for policy evaluation. This section explains how to customize policy evaluation for deployments with particular requirements not met by built-in AM functionality.

This section shows how to build and use a custom policy plugin that implements a custom subject condition, a custom environment condition, and a custom resource attribute.

About the Sample Plugin

The AM policy framework lets you build plugins that extend subject conditions, environment conditions, and resource attributes.

For information on downloading and building AM sample source code, see [How do I access and build the sample code provided for OpenAM 12.x, 13.x and AM \(All versions\)?](#) in the *Knowledge Base*.

Get a local clone so that you can try the sample on your system. In the sources, you find the following files under the `/path/to/openam-samples-external/policy-evaluation-plugin` directory:

`pom.xml`

Apache Maven project file for the module

This file specifies how to build the sample policy evaluation plugin, and also specifies its dependencies on AM components.

`src/main/java/org/forgerock/openam/examples/SampleAttributeType.java`

Extends the `com.sun.identity.entitlement.ResourceAttribute` interface, and shows an implementation of a resource attribute provider to send an attribute with the response.

`src/main/java/org/forgerock/openam/examples/SampleConditionType.java`

Extends the `com.sun.identity.entitlement.EntitlementCondition` interface, and shows an implementation of a condition that is the length of the user name.

A condition influences whether the policy applies for a given access request. If the condition is fulfilled, then AM includes the policy in the set of policies to evaluate in order to respond to a policy decision request.

`src/main/java/org/forgerock/openam/examples/SampleSubjectType.java`

Extends the `com.sun.identity.entitlement.EntitlementSubject` interface, and shows an implementation that defines a user to whom the policy applies.

A subject, like a condition, influences whether the policy applies. If the subject matches in the context of a given access request, then the policy applies.

`src/main/java/org/forgerock/openam/examples/SampleEntitlementModule.java`

`src/main/resources/META-INF/services/org.forgerock.openam.entitlement.EntitlementModule`

These files serve to register the plugin with AM.

The Java class, `SampleEntitlementModule`, implements the `org.forgerock.openam.entitlement.EntitlementModule` interface. In the sample, this class registers `SampleAttribute`, `SampleCondition`, and `SampleSubject`.

The services file, `org.forgerock.openam.entitlement.EntitlementModule`, holds the fully qualified class name of the `EntitlementModule` that registers the custom implementations. In this case, `org.forgerock.openam.entitlement.EntitlementModule`.

For an explanation of service loading, see the `ServiceLoader` API specification.

Building the Sample Plugin

Follow the steps in this procedure to build the sample plugin:

To Build the Sample Plugin

1. If you have not already done so, download and build the samples.

For information on downloading and building AM sample source code, see *How do I access and build the sample code provided for OpenAM 12.x, 13.x and AM (All versions)?* in the *Knowledge Base*.

2. When the build is complete, copy the `policy-evaluation-plugin-6.5.5.jar` file to the `WEB-INF/lib` directory where you deployed AM:

```
$ cp target/*.jar /path/to/tomcat/webapps/openam/WEB-INF/lib/
```

3. Edit the `/path/to/tomcat/webapps/openam/XUI/locales/en/translation.json` file to update the user interface to include the custom subject and environment conditions:

- a. Locate the line that contains the following text:

```
"subjectTypes": {
```

- b. Insert the following text after the line you located in the previous step:

```
"SampleSubject": {  
  "title": "Sample Subject",  
  "props": {  
    "name": "Name"  
  }  
},
```

- c. Locate the line that contains the following text:

```
"conditionTypes": {
```

- d. Insert the following text after the line you located in the previous step:

```
"SampleCondition": {  
  "title": "Sample Condition",  
  "props": {  
    "nameLength": "Minimum username length"  
  }  
},
```

4. If you require additional translations under `/path/to/tomcat/webapps/openam/XUI/locales`, modify other `translation.json` files as needed.
5. Clear your browser's cache and restart your browser.

Clearing the cache and refreshing the browser is required when you modify the `translation.json` file.

6. Restart AM or the container in which it runs.

Adding Custom Policy Implementations to Existing Policy Sets

In order to use your custom policy in existing policy sets, you must update the policy sets. Note that you cannot update a policy set that already has policies configured. When there are already policies configured for a policy set, you must instead first delete the policies, and then update the policy set.

Update the `iPlanetAMWebAgentService` policy set in the top level realm of a fresh installation. First, authenticate to AM as the `amadmin` user:

```
$ curl \
--request POST \
--header "Content-Type: application/json" \
--header "X-OpenAM-Username: amadmin" \
--header "X-OpenAM-Password: password" \
--header "Accept-API-Version: resource=2.0, protocol=1.0" \
'https://openam.example.com:8443/openam/json/realms/root/authenticate'
{
  "tokenId": "AQIC5wM2...",
  "successUrl": "/openam/console",
  "realm": "/"
}
```

Then update the `iPlanetAMWebAgentService` policy set by adding the `SampleSubject` subject condition and the `SampleCondition` environment condition:

```
$ curl \
--request PUT \
--header "iPlanetDirectoryPro: AQIC5wM2..." \
--header "Content-Type: application/json" \
--header "Accept-API-Version: resource=1.0" \
--data '{
"name": "iPlanetAMWebAgentService",
"conditions": [
  "LEAuthLevel",
  "Script",
  "AuthenticateToService",
  "SimpleTime",
  "AMIdentityMembership",
  "OR",
  "IPv6",
  "IPv4",
  "SessionProperty",
  "AuthScheme",
  "AuthLevel",
  "NOT",
  "AuthenticateToRealm",
  "AND",
  "ResourceEnvIP",
  "LDAPFilter",
  "OAuth2Scope",
  "Session",
  "SampleCondition"
],
"subjects": [
  "NOT",
  "OR",
  "JwtClaim",
  "AuthenticatedUsers",
  "AND",
  "Identity",
  "NONE",
  "SampleSubject"
],
"applicationType": "iPlanetAMWebAgentService",
"entitlementCombiner": "DenyOverride"
}' https://openam.example.com:8443/openam/json/realms/root/applications/iPlanetAMWebAgentService
```

Trying the Sample Subject and Environment Conditions

Using the AM console, add a policy to the `iPlanetAMWebAgentService` policy set in the top level realm that allows HTTP GET access for URLs based on the template `http://www.example.com:80/*` and uses the custom subject and environment conditions.

Create the policy with the following properties:

Sample Policy Properties

Property	Value
Name	Sample Policy
Resource Type	URL
Resources	Use the <code>*://*:*/*</code> resource template to specify the resource <code>http://www.example.com:80/*</code> .
Actions	Allow GET
Subject Conditions	Add a subject condition of type <code>Sample Subject</code> and a name of <code>demo</code> so that the <code>demo</code> user is the only user who can access the resource.
Environment Conditions	Add an environment condition of type <code>Sample Condition</code> and a minimum username length of <code>4</code> so that only users with a username length of <code>4</code> characters or greater can access the resource.

With the policy in place, authenticate both as a user who can request policy decisions and also as a user trying to access a resource. Both of these calls return `tokenId` values for use in the policy decision request.

```
$ curl \
--request POST \
--header "Content-Type: application/json" \
--header "X-OpenAM-Username: amadmin" \
--header "X-OpenAM-Password: password" \
--header "Accept-API-Version: resource=2.0, protocol=1.0" \
'https://openam.example.com:8443/openam/json/realms/root/authenticate'
{
  "tokenId": "AQIC5wM2...",
  "successUrl": "/openam/console",
  "realm": "/"
}
$ 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": "/"
}
```

Use the administrator `tokenId` as the header of the policy decision request, and the user `tokenId` as the subject `ssoToken` value.

```
$ curl \
--request POST \
--header "Content-Type: application/json" \
--header "Accept-API-Version: resource=2.1" \
--header "iPlanetDirectoryPro: AQIC5wM2LY4Sfcw..." \
--data '{
  "subject":{
    "ssoToken":"AQIC5wM2LY4Sfcw..."
  },
  "resources":[
    "http://www.example.com:80/index.html"
  ],
  "application":"iPlanetAMWebAgentService"
}' \
"https://openam.example.com:8443/openam/json/realms/root/policies?_action=evaluate"
{
  "resource": "http://www.example.com:80/index.html",
  "actions": {
    "GET": true
  },
  "attributes": {},
  "advices": {}
}
```

Notice that the actions returned from the policy evaluation call are set in accordance with the policy.

Trying the Sample Resource Attributes

The sample custom policy plugin can have AM return an attribute with the policy decision. In order to make this work, list the resource type for the `URL` resource type to obtain its UUID, and then update your policy to return a `test` attribute:

```
$ curl \
--request GET \
--header "iPlanetDirectoryPro: AQIC5wM2..." \
--header "Accept-API-Version: resource=1.0" \
https://openam.example.com:8443/openam/json/realms/root/resourcetypes?_queryFilter=name%20eq%20%22URL%22
{
  "result":[
    {
      "uuid":"URL-resource-type-UUID",
      "name":"URL",
      "description":"The built-in URL Resource Type available policies.",
      "patterns":["*://*:*/*", "*://*:*/*?*" ],
      ...
    }
  ],
  "resultCount":1,
  "pagedResultsCookie":null,
  "totalPagedResultsPolicy":"NONE",
  "totalPagedResults":-1,f
  "remainingPagedResults":0
}
```

When you now request the same policy decision as before, AM returns the `test` attribute that you configured in the policy.

```
$ curl \
--request POST \
--header "Content-Type: application/json" \
--header "Accept-API-Version: resource=2.1" \
--header "iPlanetDirectoryPro: AQIC5wM2LY4Sfcw..." \
--data '{
  "subject":{
    "ssoToken":"AQIC5wM2LY4Sfcy..."
  },
  "resources":[
    "http://www.example.com:80/index.html"
  ],
  "application":"iPlanetAMWebAgentService"
}' \
"https://openam.example.com:8443/openam/json/realms/root/policies?_action=evaluate"
{
  "resource": "http://www.example.com/profile",
  "actions": {
    "GET": true
  },
  "attributes": {
    "test": [
      "sample"
    ]
  },
  "advices": {}
}
```

Extending the ssoadm Classpath

After customizing your AM deployment to use policy evaluation plugins, inform **ssoadm** users to add the jar file containing the plugins to the classpath before running policy management subcommands.

To add a jar file to the **ssoadm** classpath, set the `CLASSPATH` environment variable before running the **ssoadm** command.

```
$ export CLASSPATH=/path/to/jarfile:$CLASSPATH
$ ssoadm ...
```

Scripting a Policy Condition

This section demonstrates how to use the sample policy condition script as part of an authorization policy. To examine the contents of the sample policy condition script in the AM console browse to Realms > Top Level Realm > Scripts, and then select *Scripted Policy Condition*.

The default policy condition script demonstrates how to access a user's profile information, use that information in HTTP calls, and make a policy decision based on the outcome.

For general information about scripting in AM, see "*About Scripting*".

For information about APIs available for use when scripting policy conditions, see the following sections:

- "Global Scripting API Functionality"
- "Authorization API Functionality"

Preparing

AM requires a small amount of configuration before trying the default policy condition script. The default policy condition script requires that the subject of the policy has an address in their profile. The script compares the address to the country in the resource URL and to the country from which the request originated, as determined by an external GeoIP web service. The `demo` user also requires access to evaluate policies.

The procedures in this section are:

- "To Add an Address to the Demo User"
- "To Allow a User to Evaluate Policies"
- "To Create a Policy that Uses the Default Policy Condition Script"
- "To Enable Message-level Logging for Policy Evaluation"

To Add an Address to the Demo User

In this procedure, add an address value to the `demo` user's profile. The default policy condition script uses the address when performing policy evaluation.

1. Log in as an AM administrator, for example `amadmin`.
2. Select Realms > Top Level Realm > Identities.
3. On the Identities tab, select the `demo` user.
4. In Home Address, enter a valid address. For example:

```
201 Mission St, Suite 2900, San Francisco, CA 94105
```

5. Select Save Changes.

To Allow a User to Evaluate Policies

In this procedure, add a user to a group and assign the privilege required to perform policy evaluations.

1. Log in as an AM administrator, for example `amadmin`.
2. Select Realms > Top Level Realm > Identities.

3. Select Add Identity, enter an ID for the identity, such as `restPolicyUser`, complete the required fields, and then select Create.
4. Return to Realms > Top Level Realm > Identities. On the Groups tab, select Add Group, enter an ID for the group, such as `policyEval`, and then select Create.
5. Return to Realms > Top Level Realm > Identities.
 - a. Select the user you created, for example, `restPolicyUser`.
 - b. Select the Groups tab.
 - c. In the Name box, select the group created in step 3, for example `policyEval`.
 - d. Select Save Changes.
6. Select Realms > Top Level Realm > Identities > Groups.
7. Select the group created in step 3, for example `policyEval`.
8. On the Privileges tab, select `Policy Admin`.
9. Select Save Changes.

To Create a Policy that Uses the Default Policy Condition Script

In this procedure, create a policy that uses the default policy condition script. Policy evaluations can then be performed to test the script functionality.

1. Log in as an AM administrator, for example `amadmin`.
2. Select Realms > Top Level Realm > Authorization > Policy Sets.
3. On the Policy Sets page, select `Default Policy Set`.
4. On the Default Policy Set page, select Add a Policy.
5. Define the policy as follows:
 - a. Enter a name for the policy.
 - b. Define resources to which the policy applies:
 - i. Select `URL` from the Resource Type drop down list.
 - ii. Select the resource pattern `*://*:*/*` from the Resources drop down list.
 - iii. Select Add.

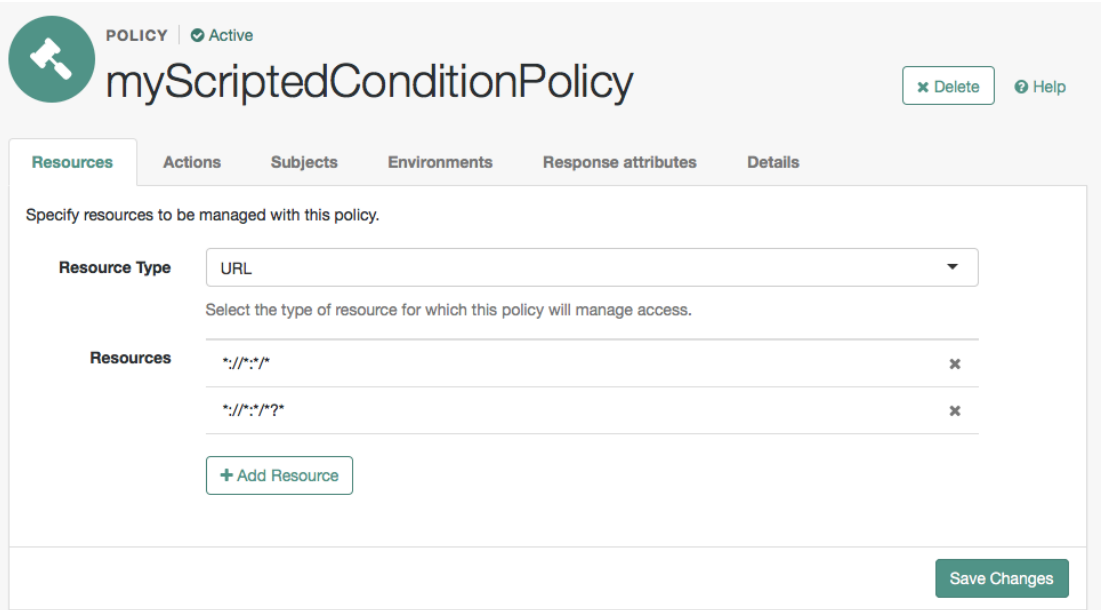
The `*://*:*/*` resource appears in the Resources field.
 - iv. Select Add Resource to add a second resource to the policy.

- v. Select the resource pattern `*://*:*/?*` from the Resources drop down list.
- vi. Select Add.

The `*://*:*/?*` resource appears along with the `*://*:*/*` resource in the Resources field.

- vii. Select Create to create the policy.

The Resources tab appears as follows:



- c. Specify actions to which the policy applies:
 - i. Select the Actions tab.
 - ii. Select GET from the Add an Action drop down list.
 - iii. The GET action appears in the list of actions. The default state for the GET action is Allow.

The Actions tab appears as follows:

The screenshot shows the configuration page for a policy named "myScriptedConditionPolicy". At the top left, there is a green circular icon with a white hammer and the text "POLICY | Active". To the right of the icon is the policy name "myScriptedConditionPolicy". Further right are two buttons: "Delete" with a red 'x' icon and "Help" with a question mark icon. Below the header is a navigation bar with tabs: "Resources", "Actions" (which is selected and highlighted in green), "Subjects", "Environments", "Response attributes", and "Details". The main content area has a heading "Select the actions that the policy applies." followed by a table. The table has two columns: "ACTION" and "DEFAULT STATE". There is one row with the action "GET" and the default state "Allow" (selected with a blue radio button) and "Deny" (unselected with a white radio button). To the right of the "Deny" option is a red 'x' icon. Below the table is a button "Add an Action" with a downward arrow. At the bottom right of the main content area is a green "Save Changes" button.

ACTION	DEFAULT STATE
GET	<input checked="" type="radio"/> Allow <input type="radio"/> Deny ✕

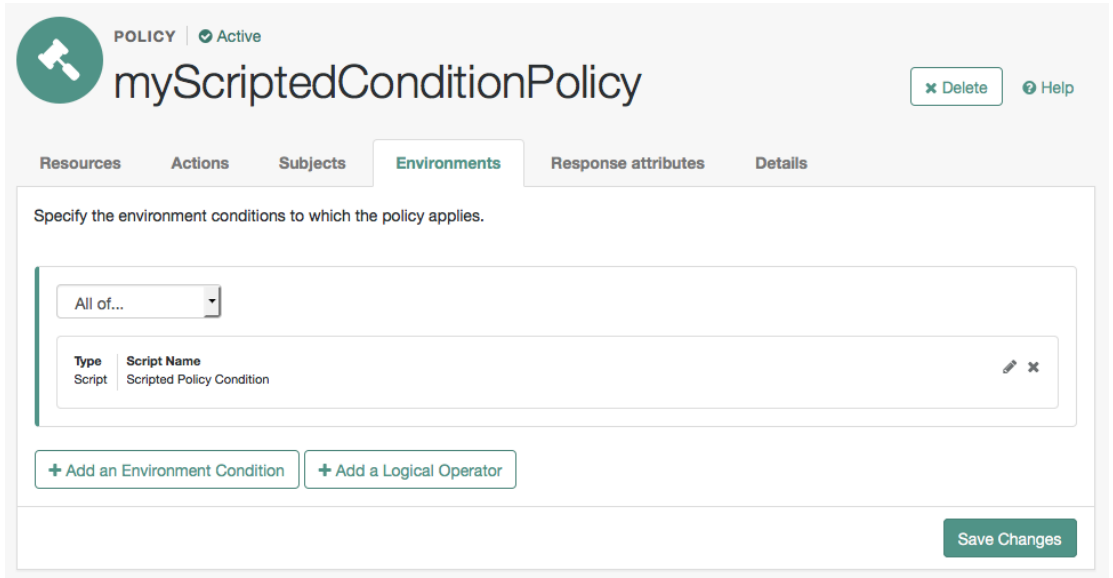
- iv. Select Save Changes.
- d. Configure identities to which the policy applies:
 - i. Select the Subjects tab.
 - ii. Select the edit icon—the pencil.
 - iii. Select Authenticated Users from the Type drop down list.
 - iv. Select the OK icon—the check mark.

The Subjects tab appears as follows:

The screenshot shows the configuration page for a policy named "myScriptedConditionPolicy". At the top left, there is a green circular icon with a white hammer and pickaxe, followed by the text "POLICY | Active". The policy name "myScriptedConditionPolicy" is displayed in a large font. To the right of the name are two buttons: "Delete" (with a red 'x' icon) and "Help" (with a question mark icon). Below the header is a navigation bar with tabs: "Resources", "Actions", "Subjects" (which is highlighted in green), "Environments", "Response attributes", and "Details". The main content area is titled "Specify the subject conditions to which the policy applies." It features a large white box with a light blue border. Inside this box, there is a dropdown menu currently set to "All of...". Below the dropdown is a list of subject conditions. One condition is visible: "Type: Authenticated Users", with a small edit icon and a close icon (red 'x') to its right. At the bottom of the white box are two buttons: "+ Add a Subject Condition" and "+ Add a Logical Operator". Below the white box is a large green "Save Changes" button.

- v. Select Save Changes.
- e. Configure environments in which the policy applies:
 - i. Select the Environments tab.
 - ii. Select Add an Environment Condition.
 - iii. Select Script from the Type drop down list.
 - iv. Select Scripted Policy Condition from the Script Name drop down list.
 - v. Select the OK icon—the check mark.

The Environments tab appears as follows:



vi. Select Save Changes.

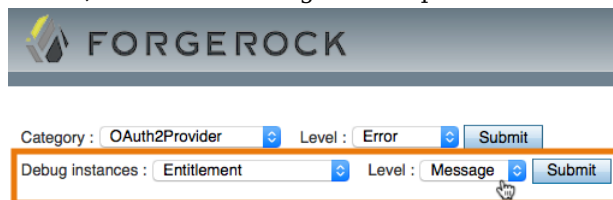
f. No additional configuration is required in the Response Attributes or Details tabs.

To Enable Message-level Logging for Policy Evaluation

The default policy condition script writes to the debug logs at the `message` level. Message-level debug logging is not enabled for policy evaluation by default.

This section shows how to enable message-level debug logging for policy evaluation, so that logger output from the default policy condition script can be viewed in the `Entitlement` debug log.

1. Log in as an AM administrator, for example `amadmin`.
2. Visit the `Debug.jsp` page, for example: <https://openam.example.com:8443/openam/Debug.jsp>.
3. From the Debug instances drop-down list, select `Entitlement`.
4. From the Level drop-down list, choose the debug level required. In this example, select `Message`.



5. Select Submit, and on the summary page that appears, select Confirm.

Message-level debug logging is now enabled for policy evaluation.

Trying the Default Policy Condition Script

This section demonstrates using a policy that contains the default policy condition script.

To evaluate against a policy, you must first obtain an SSO token for the subject performing the evaluation, in this case the `demo` user. You can then make a call to the `policies?action=evaluate` endpoint, including some environment information, which the policy uses to make an authorization decision.

To Evaluate a Policy

1. Obtain an SSO token for the `demo` 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": "/"
}
```

2. Obtain an SSO token for the user who has the privilege required to evaluate policies. For example, `restPolicyUser`.

```
$ curl \
--request POST \
--header "Content-Type: application/json" \
--header "X-OpenAM-Username: restPolicyUser" \
--header "X-OpenAM-Password: myStrongPassword" \
--header "Accept-API-Version: resource=2.0, protocol=1.0" \
'https://openam.example.com:8443/openam/json/realms/root/authenticate'
{
  "tokenId": "AQIC8aF...TA10Q*",
  "successUrl": "/openam/console",
  "realm": "/"
}
```

3. Send an evaluation request to the `policies` endpoint, providing the SSO token of the `restPolicyUser` user as the value of the `iPlanetDirectoryPro` header.

In the JSON data, set the `subject` object to the SSO token of the `demo` user. In the `resources` object, include a URL that resides on a server in the same country as the address set for the `demo` user. In the `environment` object, include an IP address that is also based in the same country as the user and the resource.

The example below uses the URL of a web site and an IP address located in the United States:

```
$ curl \
--request POST \
--header "Content-Type: application/json" \
--header "iPlanetDirectoryPro: AQIC8aF...TA10Q*" \
--data '{
  "resources":[
    "https://www.us-site.com:8443/index.html"
  ],
  "application":"iPlanetAMWebAgentService",
  "subject":{
    "ssoToken":"AQIC5wM...TU30Q*"
  },
  "environment":{
    "IP":[
      "38.99.39.210"
    ]
  }
}' \
"https://openam.example.com:8443/openam/json/realms/root/policies?_action=evaluate"
{
  "advices":{},
  "ttl":9223372036854775807,
  "resource":"https://www.us-site.com:8443/index.html",
  "actions":{
    "POST":true,
    "GET":true
  },
  "attributes":{
    "countryOfOrigin":[
      "United States"
    ]
  }
}
```

If the country in the subject's profile matches the country determined from the source IP in the environment and the country determined from the resource URL, then AM returns a list of actions available. The script will also add an attribute to the response called `countryOfOrigin` with the country as the value.

If the countries do not match, no actions are returned. In the following example, the resource URL is based in France, while the IP and user's address in the profile are based in the United States:

```

$ curl \
--request POST \
--header "Content-Type: application/json" \
--header "iPlanetDirectoryPro: AQIC8aF...TA10Q*" \
--data '{
  "resources": [
    "https://www.france-site.com:8443/index.html"
  ],
  "application": "iPlanetAMWebAgentService",
  "subject": {
    "ssoToken": "AQIC5wM...TU30Q*"
  },
  "environment": {
    "IP": [
      "38.99.39.210"
    ]
  }
}' \
"https://openam.example.com:8443/openam/json/realms/root/policies?_action=evaluate"
{
  "advices": {},
  "ttl": 9223372036854775807,
  "resource": "https://www.france-site.com:8443/index.html",
  "actions": {},
  "attributes": {}
}

```

Chapter 5

Reference

This reference section covers settings and the scripting API relating to authorization in AM.

Global Service Properties

The following sections document AM services with configuration properties that affect AM authorization:

- "Push Notification Service"
- "Policy Configuration"
- "Transaction Authentication Service"

Push Notification Service

amster service name: `PushNotification`

Realm Defaults

The following settings appear on the **Realm Defaults** tab:

SNS Access Key ID

Amazon Simple Notification Service Access Key ID. For more information, see [Setting up access for Amazon SNS](#).

For example, you might set this property to: `AKIAIOSFODNN7EXAMPLE`

amster attribute: `accessKey`

SNS Access Key Secret

Amazon Simple Notification Service Access Key Secret. For more information, see [Setting up access for Amazon SNS](#).

For example, you might set this property to: `wJalrXUtnFEMI/K7MDENG/bPxrFiCYEXAMPLEKEY`

amster attribute: `secret`

SNS Endpoint for APNS

The Simple Notification Service endpoint in Amazon Resource Name format, used to send push messages to the Apple Push Notification Service (APNS).

For example, you might set this property to: *arn:aws:sns:us-east-1:1234567890:app/APNS/production*

amster attribute: `appleEndpoint`

SNS Endpoint for GCM

The Simple Notification Service endpoint in Amazon Resource Name format, used to send push messages over Google Cloud Messaging (GCM).

For example, you might set this property to: *arn:aws:sns:us-east-1:1234567890:app/GCM/production*

amster attribute: `googleEndpoint`

SNS Client Region

Region of your registered Amazon Simple Notification Service client. For more information, see <https://docs.aws.amazon.com/general/latest/gr/rande.html>.

The possible values for this property are:

- `us-gov-west-1`
- `us-east-1`
- `us-west-1`
- `us-west-2`
- `eu-west-1`
- `eu-central-1`
- `ap-southeast-1`
- `ap-southeast-2`
- `ap-northeast-1`
- `ap-northeast-2`
- `sa-east-1`
- `cn-north-1`

Default value: `us-east-1`

amster attribute: `region`

Message Transport Delegate Factory

The fully qualified class name of the factory responsible for creating the `PushNotificationDelegate`. The class must implement `org.forgerock.openam.services.push.PushNotificationDelegate`.

Default value: `org.forgerock.openam.services.push.sns.SnsHttpDelegateFactory`

amster attribute: `delegateFactory`

Response Cache Duration

The minimum lifetime to keep unanswered message records in the message dispatcher cache, in seconds. To keep unanswered message records indefinitely, set this property to `0`. Should be tuned so that it is applicable to the use case of this service. For example, the ForgeRock Authenticator (Push) authentication module has a default timeout of 120 seconds.

Default value: `120`

amster attribute: `mdDuration`

Response Cache Concurrency

Level of concurrency to use when accessing the message dispatcher cache. Defaults to `16`, and must be greater than `0`. Choose a value to accommodate as many threads as will ever concurrently access the message dispatcher cache.

Default value: `16`

amster attribute: `mdConcurrency`

Response Cache Size

Maximum size of the message dispatcher cache, in number of records. If set to `0` the cache can grow indefinitely. If the number of records that need to be stored exceeds this maximum, then older items in the cache will be removed to make space.

Default value: `10000`

amster attribute: `mdCacheSize`

Policy Configuration

amster service name: `PolicyConfiguration`

Global Attributes

The following settings appear on the **Global Attributes** tab:

Resource Comparator

AM uses resource comparators to match resources specified in policy rules. When setting comparators on the command line, separate fields with `|` characters.

Default value:

```
serviceType=iPlanetAMWebAgentService|class=com.sun.identity.policy.plugins.HttpURLResourceName|
wildcard=*|oneLevelWildcard=-*|delimiter=/|caseSensitive=false
serviceType=sunIdentityServerDiscoveryService|
class=com.sun.identity.policy.plugins.PrefixResourceName|wildcard=*|oneLevelWildcard=-*|delimiter=;|
caseSensitive=false
```

amster attribute: `resourceComparators`

Continue Evaluation on Deny Decision

If no, then AM stops evaluating policy as soon as it reaches a deny decision.

Default value: `false`

amster attribute: `continueEvaluationOnDeny`

Realm Alias Referrals

If yes, then AM allows creation of policies for HTTP and HTTPS resources whose FQDN matches the DNS alias for the realm even when no referral policy exists.

Default value: `false`

amster attribute: `realmAliasReferrals`

Realm Defaults

The following settings appear on the **Realm Defaults** tab:

Primary LDAP Server

Configuration directory server host:port that AM searches for policy information.

Format: `local AM server name | hostname:port`

Multiple entries must be prefixed by local server name. Make sure to place the multiple entries on a single line and separate the hostname:port URLs with a space.

For example, `openam.example.com|opendj.example.com:1389 opendj.example.com:2389`

Default value: `openam.example.com:50389`

amster attribute: `ldapServer`

LDAP Users Base DN

Base DN for LDAP Users subject searches.

Default value: `dc=openam,dc=forgerock,dc=org`

amster attribute: `usersBaseDn`

LDAP Bind DN

Bind DN to connect to the directory server for policy information.

Default value: `cn=Directory Manager`

amster attribute: `bindDn`

LDAP Bind Password

Bind password to connect to the directory server for policy information.

amster attribute: `bindPassword`

LDAP Organization Search Filter

Search filter to match organization entries.

Default value: `(objectclass=sunismangedorganization)`

amster attribute: `realmSearchFilter`

LDAP Users Search Filter

Search filter to match user entries.

Default value: `(objectclass=inetorgperson)`

amster attribute: `usersSearchFilter`

LDAP Users Search Scope

Search scope to find user entries.

The possible values for this property are:

- SCOPE_BASE
- SCOPE_ONE
- SCOPE_SUB

Default value: SCOPE_SUB

amster attribute: usersSearchScope

LDAP Users Search Attribute

Naming attribute for user entries.

Default value: uid

amster attribute: usersSearchAttribute

Maximum Results Returned from Search

Search limit for LDAP searches.

Default value: 100

amster attribute: maximumSearchResults

Search Timeout

Time after which AM returns an error for an incomplete search, in seconds.

Default value: 5

amster attribute: searchTimeout

LDAP SSL/TLS

If enabled, AM connects securely to the directory server. This requires that you install the directory server certificate.

Default value: false

amster attribute: sslEnabled

LDAP Connection Pool Minimum Size

Minimum number of connections in the pool.

Default value: 1

amster attribute: `connectionPoolMinimumSize`

LDAP Connection Pool Maximum Size

Maximum number of connections in the pool.

Default value: `10`

amster attribute: `connectionPoolMaximumSize`

Heartbeat Interval

Specifies how often should AM send a heartbeat request to the directory.

Use this option in case a firewall/loadbalancer can close idle connections, since the heartbeat requests will ensure that the connections won't become idle.

Default value: `10`

amster attribute: `policyHeartbeatInterval`

Heartbeat Unit

Defines the time unit corresponding to the Heartbeat Interval setting.

Use this option in case a firewall/loadbalancer can close idle connections, since the heartbeat requests will ensure that the connections won't become idle.

The possible values for this property are:

- `SECONDS`. second
- `MINUTES`. minute
- `HOURS`. hour

Default value: `SECONDS`

amster attribute: `policyHeartbeatTimeUnit`

Subjects Result Time to Live

Maximum time that AM caches a subject result for evaluating policy requests, in minutes. A value of `0` prevents AM from caching subject evaluations for policy decisions.

Default value: `10`

amster attribute: `subjectsResultTTL`

User Alias

If enabled, AM can evaluate policy for remote users aliased to local users.

Default value: `false`

amster attribute: `userAliasEnabled`

Check resources exist when Resource Server is updated

Check all registered resources exist when updating Resource Server.

Policy Set will check each registered Resource Types one by one against config datastore if enabled. Consider disabling this option if you have large number of Resource Types registered to a Policy Set.

Default value: `true`

amster attribute: `checkIfResourceTypeExists`

Transaction Authentication Service

amster service name: `TransactionAuthentication`

Realm Defaults

The following settings appear on the **Realm Defaults** tab:

Time to Live

The number of seconds within which the transaction must be completed.

Default value: `180`

amster attribute: `timeToLive`

Authorization API Functionality

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

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>	Describe the environment passed from the client making the authorization request. For example, the following shows a simple <code>environment</code> map with a single entry: <pre> "environment": { "IP": ["127.0.0.1"] } </pre>
<code>resourceURI</code>	<code>String</code>	Specify the URI of the resource to which authorization is being requested.
<code>username</code>	<code>String</code>	Specify the user ID of the subject that is requesting authorization.

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>	<code>Attribute Name</code> (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>	<code>Attribute Name</code> (type: <code>String</code>) <code>Attribute Values</code> (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>	<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 the subject of the authorization request.

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

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)	String	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

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>ttl</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>

Appendix A. About the REST API

This appendix shows how to use the RESTful interfaces for direct integration between web client applications and ForgeRock Access Management.

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.)

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.

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.

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".

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.

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.

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.

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.

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.

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.

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.

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`:

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 value included in the patch is 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" ]
}
```

Note that you can add only one array element one at a time, as per the corresponding JSON Patch specification. If you add an array of elements, for example:

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

The resulting resource would have the following invalid JSON structure:

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

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".

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.

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".

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.

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" ],
  }
]
```

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"
      }
    }
  }
]
```


Patch Operation Limitations

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

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

Parameters

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

`_prettyPrint=true`

Format the body of the response.

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

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

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

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

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.

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.

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.

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.

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.

Supported REST API Versions

For information about the supported protocol and resource versions available in AM, see the API Explorer in the *Development Guide* available in the AM console.

The *AM Release Notes* section, "Changes and Deprecated Functionality" in the *Release Notes* describes the differences between API versions.

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".

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.
```

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".

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=*
```

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*.

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 `dēmjø`, 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": "/"
}
```

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".

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"*.

- **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.

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.eyJvdiIjOiJ... ", 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": [ ❻
    {
      "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.

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": "0"
      }
    ]
  }
]
```

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 bXlDbGllbnQ6Zm9yZ2Vybn2Nr
```

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`

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"
}
```

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
  }
}

```

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*.

Windows Desktop SSO Requirements

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

```
$ curl \
--request POST \
--header "Content-Type: application/json" \
--header "Accept-API-Version: resource=2.0, protocol=1.0" \
--header "X-OpenAM-Username: demo" \
--header "X-OpenAM-Password: changeit" \
--header "Authorization: Basic ZGVtbzpjajGFuZ2VpdA==" \
'https://openam.example.com:8443/openam/json/realms/root/authenticate'
{
  "tokenId": "AQIC5w...NTcy*",
  "successUrl": "/openam/console",
  "realm": "/"
}
```

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.

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"
}
```

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.*
```

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.

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.

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.

Reference

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

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`

Appendix B. About Scripting

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

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.

+ *How to determine the JavaScript Engine Version?*

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"
```

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

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](#).

+ *How to determine the Groovy Engine Version?*

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:

```
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.

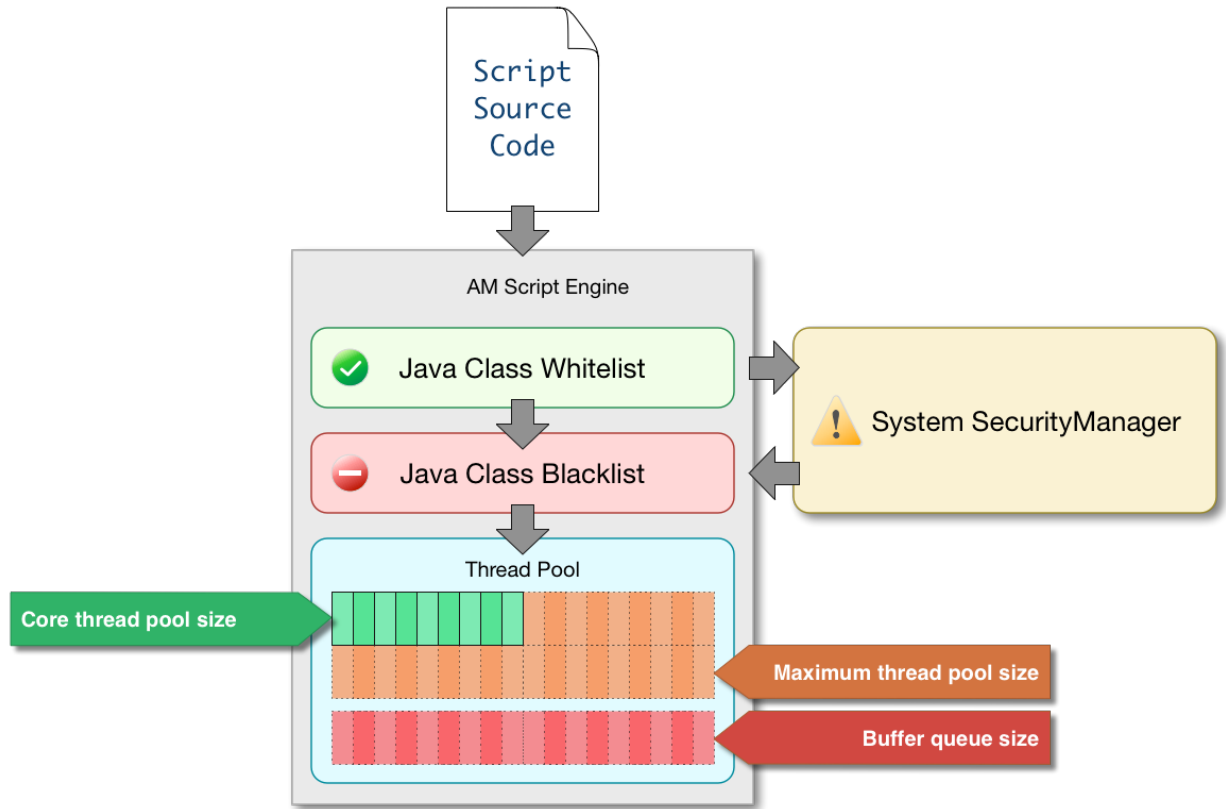
OIDC Claims

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.



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.

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".

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

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

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: String)	Void	Write <i>Error Message</i> to AM debug logs if ERROR level logging is enabled.

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

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.

Managing Scripts With the AM Console

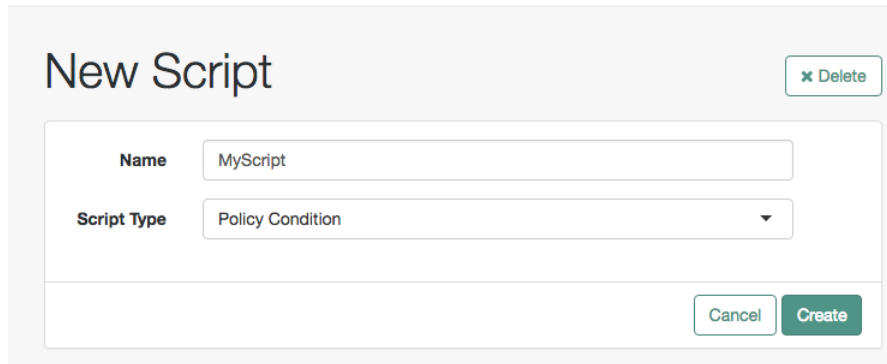
The following procedures describe how to create, modify, and delete scripts using the AM console:

- "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:




New Script ✕ Delete

Name

Script Type

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

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:

- a. Enter a description of the script.
- b. Choose the script language, either JavaScript or Groovy. Note that not every script type supports both languages.
- c. Enter the source code in the Script field.

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

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

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

- e. Save your changes.

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.

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": "dmFyIFNUNUQVJUX1RJ...",
  "language": "JAVASCRIPT",
  "context": "AUTHENTICATION_SERVER_SIDE",
  "createdBy": "id=dsameuser,ou=user,dc=openam,dc=forgerock,dc=org",
  "creationDate": 1433147666269,
  "lastModifiedBy": "id=dsameuser,ou=user,dc=openam,dc=forgerock,dc=org",
  "lastModifiedDate": 1433147666269
}
```

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

`_id`

The UUID that 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:

```
dmFyIGEGPSAxMjM7IA0KdmFyIGIyPSA0NTY7
```

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`.

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

```

Supported `_queryFilter` Fields and Operators

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

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": "Ly0qCiAqIFRoaxMg...",
  "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
}
```

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": "dmFyIGEGPSAxMjM7dmFyIGIyPSA0NTY7Cg==",
  "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": "dmFyIGEGPSAxMjM7dmFyIGIyPSA0NTY7ID1WQUxJREFUSU90IFNIT1VMRCBGQULMPQo=",
  "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"
    }
  ]
}
```

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

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
}
```

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
{}
```

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
```

Scripting

amster service name: `Scripting`

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`

Secondary Configurations

This service has the following Secondary Configurations.

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 C. 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 on ForgeRock's support offering, including support plans and service level agreements (SLAs), visit <https://www.forgerock.com/support>.

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

	<p>request. For browser-based clients, AM sets a cookie in the browser that contains the session information.</p> <p>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.</p>
Conditions	<p>Defined as part of policies, these determine the circumstances under which which a policy applies.</p> <p>Environmental conditions reflect circumstances like the client IP address, time of day, how the subject authenticated, or the authentication level achieved.</p> <p>Subject conditions reflect characteristics of the subject like whether the subject authenticated, the identity of the subject, or claims in the subject's JWT.</p>
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 <i>reference</i> 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.