



Authorization Guide

/ ForgeRock Access Management 7.0.2

Latest update: 7.0.2

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

Copyright © 2011-2021 ForgeRock AS.

Abstract

Guide to working with authorization in ForgeRock® Access Management (AM). ForgeRock Access Management provides intelligent authentication, authorization, federation, and single sign-on functionality.



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

Overview	iv
1. About Authorization and Policy Decisions	1
Protecting Resources	1
Reaching Policy Decisions	2
2. Configuring Policies	4
Policies (REST)	17
3. Configuring Resource Types	46
Resource Types (REST)	51
4. Configuring Policy Sets	59
Policy Sets (REST)	61
Policy Set Application Types (REST)	70
5. Importing and Exporting Policies	73
Exporting to XACML	75
Importing from XACML	78
6. Requesting Authorization from AM	82
Requesting Policy Decisions Using REST	82
7. Transactional Authorization	95
8. Dynamic OAuth 2.0 Authorization	112
OAuth 2.0 Scopes Policy Script API Functionality	117
9. Customizing Policy Evaluation With a Plug-In	119
About the Sample Plugin	119
Building the Sample Plugin	120
Adding Custom Policy Implementations to Existing Policy Sets	121
Trying the Sample Subject and Environment Conditions	123
Trying the Sample Resource Attributes	124
Extending the ssoadm Classpath	126
10. Scripting a Policy Condition	127
Preparing AM to use Scripted Policy Conditions	127
Trying the Default Policy Condition Script	133
Policy Condition Script API Functionality	135
Glossary	140

Overview

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

Quick Start

 <p>About Authorization</p> <p>Learn how AM determines whether to allow users to access protected resources by using a set of authorization policies.</p>	 <p>Create Policies</p> <p>Define resources, and protect them by creating authorization policies, all by using the AM console or REST endpoints.</p>
 <p>What is Transactional Authorization?</p> <p>Use transactional authorization to improve security by requiring your users to perform additional actions when trying to access certain resources protected by an AM policy.</p>	 <p>Dynamic OAuth 2.0 Scopes</p> <p>Learn how to grant OAuth 2.0 scopes dynamically with AM's authorization engine.</p>

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

About Authorization and Policy Decisions

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.

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.

Related Information:

- "*Dynamic OAuth 2.0 Authorization*"

Protecting Resources

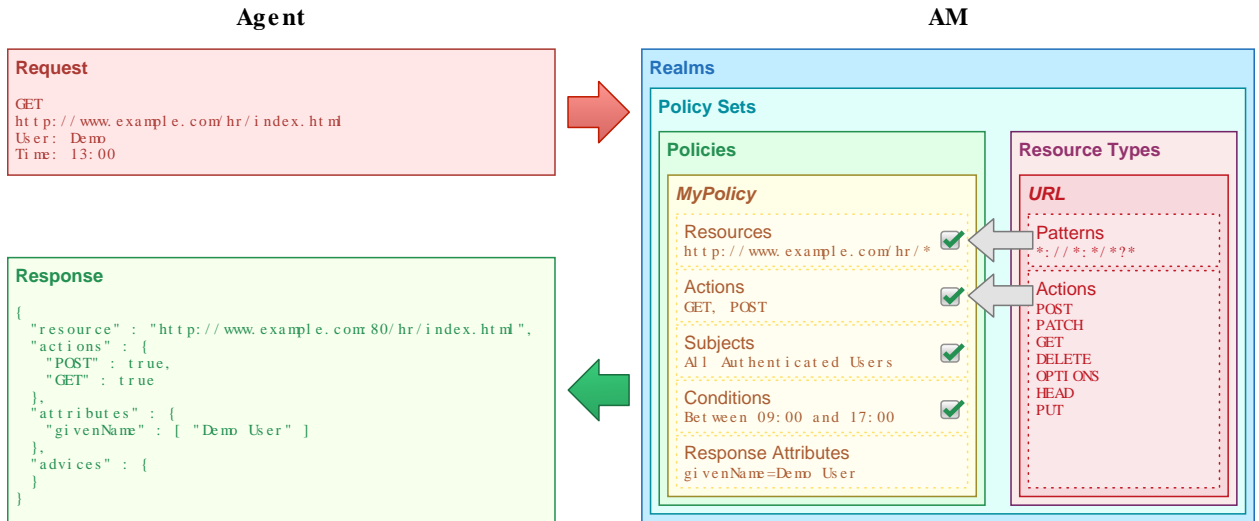
When you configure policy sets to protect resources, AM acts as the *policy decision point* (PDP), whereas AM web and Java agents act as *policy enforcement points* (PEP). In other words, an agent or other PEP takes responsibility only for enforcing a policy decision rendered by AM. When you configured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

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**.

+ *Example*

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 website developer can offer a profile page, but the website 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 website. You might need to agree with website 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 website, or allow users already logged in to their desktops to visit protected sites without having to type their credentials again.

Chapter 2

Configuring Policies

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

Authorization *policies* let AM determine whether to grant a subject access to a resource.

A policy defines the following:

resources

The resource definitions constrain specific 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 lets users 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.

+ *About Resource Types and Policy Sets*

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

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

See "*Configuring Resource Types*".

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

See "*Configuring Policy Sets*".

You can manage policies using the AM console or the REST API:

- "To Configure a Policy (console)"
- "Policies (REST)"

To Configure a Policy (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 ▼

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

New Policy Help

Name

Resource Type ▼

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

Resources ▼

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 "[Dynamic OAuth 2.0 Authorization](#)".

- 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.

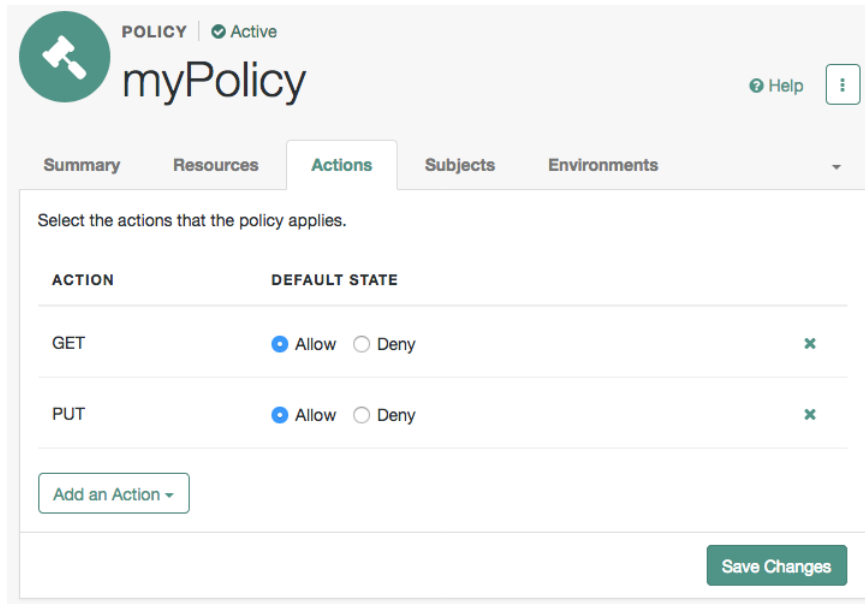
Tip

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.

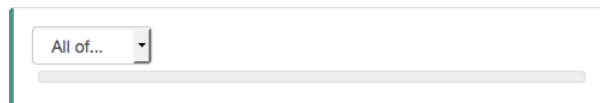
Allowing or Denying the Action for the Resource



- 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 screenshot displays the 'Subjects' tab of a policy configuration interface. At the top, there are tabs for 'Resources', 'Actions', 'Subjects', and 'Environments'. Below the tabs, a text prompt reads 'Specify the subject conditions to which the policy applies.' The main area contains a series of nested boxes representing logical conditions. The outermost box has a dropdown menu set to 'All of...'. Inside this box, there are two sub-sections: 'Type' with 'Users & Groups' selected, and 'User Subjects' with 'Employees' selected. A second box is nested inside the first, with a dropdown set to 'Not...'. This second box contains two sub-sections: 'Type' with 'Users & Groups' selected, and 'User Subjects' with 'Partners' selected. A third box is nested inside the second, with a dropdown set to 'Any of...'. This third box contains two sub-sections: 'Type' with 'Users & Groups' selected, and 'User Subjects' with 'Customers' selected. At the bottom of the interface, there are two buttons: '+ Add a Subject Condition' and '+ Add a Logical Operator'. A 'Save Changes' button is located at the bottom right.

+ *Subject Condition Types*

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.

+ Environment Condition Types**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 periods (.) 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 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=admins.`

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 policy condition. At the top, there is a 'Type' dropdown menu with the selected value 'Time (day, date, time, and timezone)'. Below this, there are seven input fields: 'Start Time' (09:00), 'End Time' (17:30), 'Start Day' (Monday), 'End Day' (Friday), 'Start Date' (2015:12:16), 'End Date' (YYYY:MM:DD), and 'Time Zone' (GMT). Each field has a small circular icon to its right, likely for clearing or refreshing the value. A checkmark is visible in the top right corner of the form.

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 "*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.

Policies (REST)

AM provides REST APIs for the following:

- "Querying Policies"
- "Reading a Specific Policy"
- "Creating Policies"
- "Updating Policies"
- "Deleting Policies"
- "Copying and Moving Policies"
- "Managing Environment Condition Types"
- "Managing Subject Condition Types"
- "Managing Subject Attributes"
- "Managing Decision Combiners"

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`).

+ Explained Example

```
{
  "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 `mypolicysset`.

"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 "Understanding Session Termination" in the *Security 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`.

Before making a REST API call to request 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 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`.

You must also 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" in the *Authentication and Single Sign-On Guide*.

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/realms/root/realms/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" in the *Getting Started with REST*.

+ *Supported `_queryFilter` Fields and Operators*

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

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" in the *Setup 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 labeled 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/realms/root/realms/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/realm/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 *"Dynamic OAuth 2.0 Authorization"*.

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 \br/>--header "Content-Type: application/json" \br/>--header "iPlanetDirectoryPro: AQIC5..." \br/>--header "Accept-API-Version: resource=1.0" \br/>--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.

+ 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.

+ *JSON Input Data for Copying or Moving Multiple Policies*

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

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

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

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

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

```

$ 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"
}

```

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" in the *Getting Started with REST*.

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" in the *Getting Started with REST*.

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 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 store. For more information, see "Identity Stores" in the *Setup 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" in the *Getting Started with REST*.

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"
}
```

Chapter 3

Configuring Resource Types

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

+ *Where Do Resource Types Fit in Authorization?*

For AM to decide if a user can access a resource, it requires a *policy*. When you configure a policy, you also configure a resource (or a pattern to match several resources) that the policy will apply to, and the actions that the policy allow or deny.

Resource types are templates that you can define once and reuse in several policies. For example, you could create a template that always PUT and POST operations from your internal network.

The default resource types that AM provides are enough for most environments.

+ *Default Resource Types*

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 `*://*:*/**?`. The resource supports the `GRANT` action, which can be allowed or denied.

Configure resource types using the AM console or the REST API:

- "To Configure a Resource Type (console)"
- "Resource Types (REST)"

To Configure a Resource Type (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.

+ 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+t5P1P+ISyQhVlp1E22A8=&action=get` is equivalent to the query string `?action=get&subject=SPBnfm+t5P1P+ISyQhVlp1E22A8=`.

- 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.

- 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.

Resource Types (REST)

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.

+ Explained Example

```
{
  "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`.

Before making a REST API call to request 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 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`.

You must also 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" in the *Authentication and Single Sign-On Guide*.

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" in the *Getting Started with REST*.

+ 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>)

Field	Supported Operators
actions	Equals (eq), Contains (co), Starts with (sw)

Reading a Specific Resource Type

To read 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/realms/root/realms/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/realms/root/realms/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 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.

Chapter 4

Configuring Policy Sets

Policy Sets are associated with a set of resource types, and contain one or more policies based upon the template it provides. They are used as templates for policies protecting Web sites, Web applications, or other resources.

+ *Where Do Policy Sets Fit in Authorization?*

For AM to decide if a user can access a resource, it requires a *policy*.

Policy sets are a way to group up policies that protect applications or sites that have similar characteristics (for example, they all use the same *resource type*) so that you do not need to configure some of the parameters in each policy.

There is a template for policy sets too, called application types. You can modify these using REST.

AM includes two policy sets that are used for web and Java agents, and for dynamic OAuth 2.0 policies.

Policy sets have templates, called application types. There are two application types defined by default, which correspond to the default policy sets. You can only configure these using REST, but for most use cases, you can use the default application types.

+ *Default Policy Sets*

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.

When creating or editing policy sets, consider the following points:

- By default, web and Java agents request policy decisions in the Top Level Realm (/) from the default policy set, `iPlanetAMWebAgentService`. When the realm and policy set differ for your web or Java agent, you can specify the realm and policy set in the agent profile. AM then directs requests from the agent to the specified realm and policy set, so this is backwards compatible with existing web and Java agents.

You can configure the realm and policy set the agent uses in the agent profile. See the *ForgeRock Web Agents User Guide*, or the *ForgeRock Java Agents User Guide* for more information.

- 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.
- AM creates a policy set containing a policy representing the resources and identities specified by a resource owner using UMA 2.0 to share their registered resources.

These policies appear in the AM administration console as read-only, and cannot be edited by administrative users such as `amAdmin`. They can, however, be viewed and deleted.

Manage policy sets using the AM console or the REST API:

- "To Configure a Policy Set (console)"
- "Policy Sets (REST)"

To Configure a Policy Set (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. If creating a new policy, enter an ID for the policy set. This is a required parameter.

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

3. (Optional) If creating a new policy, 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`).

- 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.

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

Policy Sets (REST)

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`).

+ Explained Example

```
{
  "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 "Policies (REST)" 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 "Policies (REST)" 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`.

Before making a REST API call to request 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 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`.

You must also 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" in the *Authentication and Single Sign-On Guide*.

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" in the *Getting Started with REST*.

+ 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": "mypolicysset"  
}
```

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
```

Policy Set Application Types (REST)

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`).

+ Explained Example

```
{
  "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" in the *Getting Started with REST*.

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"
}
```

Chapter 5

Importing and Exporting Policies

You can import and export policies to and from files.

You can use these files to back up 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.

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.

Importing and exporting JSON:

- "To Export Policies in JSON Format (ssoadm)"
- "To Import Policies in JSON Format (ssoadm)"

Importing and exporting XACML:

- "Exporting to XACML"
- "Importing from XACML"

To Export Policies in JSON Format (ssoadm)

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

```
$ ssoadm \  
policy-export \  
--realm "/" \  
--servername "https://openam.example.com:8443/openam" \  
--jsonfile "myPolicies.json" \  
--adminid uid=amAdmin,ou=openam,dc=openam,dc=forgerock,dc=org \  
--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 (ssoadm)

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

```
$ ssoadm \  
policy-import \  
--realm "/myRealm" \  
--servername "https://openam.example.com:8443/openam" \  
--jsonfile "myPolicies.json" \  
--adminid uid=amAdmin,ou=People,dc=openam,dc=forgerock,dc=org \  
--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*.

Exporting to XACML

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.

+ Policy Sets to XACML Mappings

AM	XACML
Realm:<timestamp>(yyyy.MM.dd.HH.mm.ss.SSS)	PolicySet ID
Current Time (yyyy.MM.dd.HH.mm.ss.SSS)	Version
Deny Overrides	Policy Combining Algorithm ID
No targets defined	Target

When exporting AM policies to XACML 3.0 policy sets, AM maps its policies to XACML 3.0 policy elements.

+ Policies to XACML Mappings

AM Policy	XACML Policy
Policy Name	Policy ID
Description	Description
Current Time (yyyy.MM.dd.HH.mm.ss.SSS)	Version
xacml rule target	entitlement excluded resource names
Rule Deny Overrides	Rule Combining Algorithm ID
Any of: <ul style="list-style-type: none"> • Entitlement Subject • Resource Names • Policy Set Names • Action Values 	Target

AM Policy	XACML Policy
Any of: <ul style="list-style-type: none"> • Policy Set Name • Entitlement Name • Privilege Created By • Privilege Modified By • Privilege Creation Date • Privilege Last Modification Date 	Variable Definitions
Single Level Permit/Deny Actions converted to Policy Rules	Rules

Note

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

You can export policies to XACML in the following ways:

- "To Export Policies in XACML Format (console)"
- "To Export Policies in XACML Format (REST)"
- "To Export Policies in XACML Format with Search Filters (REST)"
- "To Export Policies in XACML Format (ssoadm)"

To Export Policies in XACML Format (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 Export Policies in XACML Format (REST)

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

Tip

You can filter your XACML exports using query search filters. See "To Export Policies in XACML Format with Search Filters (REST)".

- 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 in XACML Format with Search Filters (REST)

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)

Element	Description
	<ul style="list-style-type: none"> > Greater Than or Equal To (numerical)
Attribute Value	The matching value. Asterisk wildcards are supported.

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"
```

To Export Policies in XACML Format (ssoadm)

- Use the `ssoadm list-xacml` command:

```
$ ssoadm \
list-xacml \
--realm "/" \
--adminid uid=amAdmin,ou=People,dc=openam,dc=forgerock,dc=org \
--password-file /tmp/pwd.txt

<?xml version="1.0" encoding="UTF-8"?>
<PolicySet
....

Policy definitions were returned under realm, /.
```

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

Importing from XACML

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 XACML policies in the following ways:

- "To Import Policies in XACML Format (console)"

- "To Import XACML Policies (REST)"
- "To Import Policies in XACML Format (ssoadm)"

To Import Policies in XACML Format (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 Import XACML Policies (REST)

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

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"
```

To Import Policies in XACML Format (ssoadm)

- Use the `ssoadm create-xacml` command:

```
$ ssoadm \
  create-xacml \
  --realm "/" \
  --adminid uid=amAdmin,ou=People,dc=openam,dc=forgerock,dc=org \
  --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*.

Chapter 6

Requesting Authorization from AM

Once you have configured AM to determine whether to grant or deny access based on the policies you created, you must configure your policy enforcement points (PEP) to use AM.

The ForgeRock Identity Platform provides the following PEPs:

- Web Agents and Java Agents, which are add-on components installed on the web server or container serving your applications. They are tightly integrated with AM, and serve exclusively as policy enforcement points.

For more information, see the *ForgeRock Web Agents User Guide*, or the *ForgeRock Java Agents User Guide*.

- ForgeRock Identity Gateway, which is a high-performance reverse proxy server that can also function as a policy enforcement point.

For more information, see the *ForgeRock Identity Gateway Getting Started Guide*.

The ForgeRock Identity Platform PEP's intercept inbound client requests to access a resource in your web site or application. Then, based on internal rules, they may defer the request to AM for policy evaluation. Since they are tightly integrated with AM, you do not need to add additional code to request policy evaluation or manage advices.

We recommend that you use the ForgeRock Identity Platform PEP's. However, you can code your own and make REST calls to AM to request policy evaluation.

Related information:

- "Requesting Policy Decisions Using REST"

Requesting Policy Decisions Using REST

You can request policy decisions from AM by using the REST API 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.

AM provides the `/json/realms/root/policies` endpoint to request policy evaluation.

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`.

Before making a REST API call to request 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" in the *Authentication and Single Sign-On Guide*.

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 "Policy Sets (REST)".

"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 *"Authenticating (REST)"* in the *Authentication and Single Sign-On Guide*.

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 "Policies (REST)".

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 "Policy Sets (REST)".

"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, *"Authenticating (REST)"* in the *Authentication and Single Sign-On 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 <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.

Chapter 7

Transactional Authorization

Transactional authorization makes use of the "*Session Upgrade*" in the *Sessions Guide* functionality, applying it while authorization is in progress. It 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.

Transactional authorization is implemented as a new environment condition type that can be added to authorization policies.

+ *Example*

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	✎ ✕
	/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

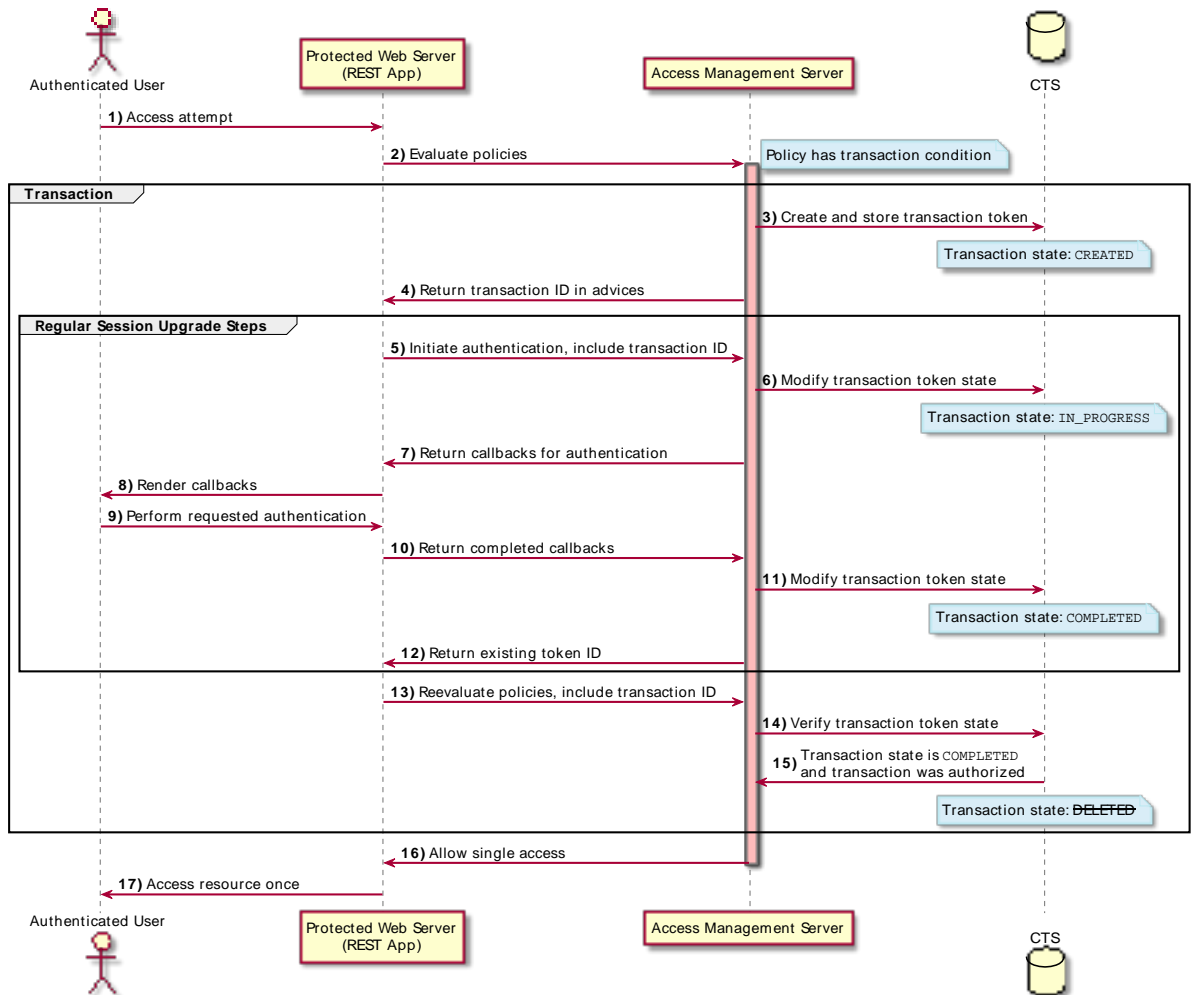
Related Information:

"About Authorization and Policy Decisions"

+ *How Does Transactional Authorization Work?*

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" in the *Reference*.

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 Using REST".

- 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.transactionauth.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 reevaluate 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.

To configure transactional authorization, first configure the chains or trees you need for session upgrade, and then configure your policies. The following procedures demonstrate an example that uses push notifications:

Prerequisites

- 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 9.4: "To Create a 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" in the *Reference*.

- 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 the *ForgeRock Web Agents User 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".

To Prepare 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.

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 box, 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 "*Configuring Policies*"

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.

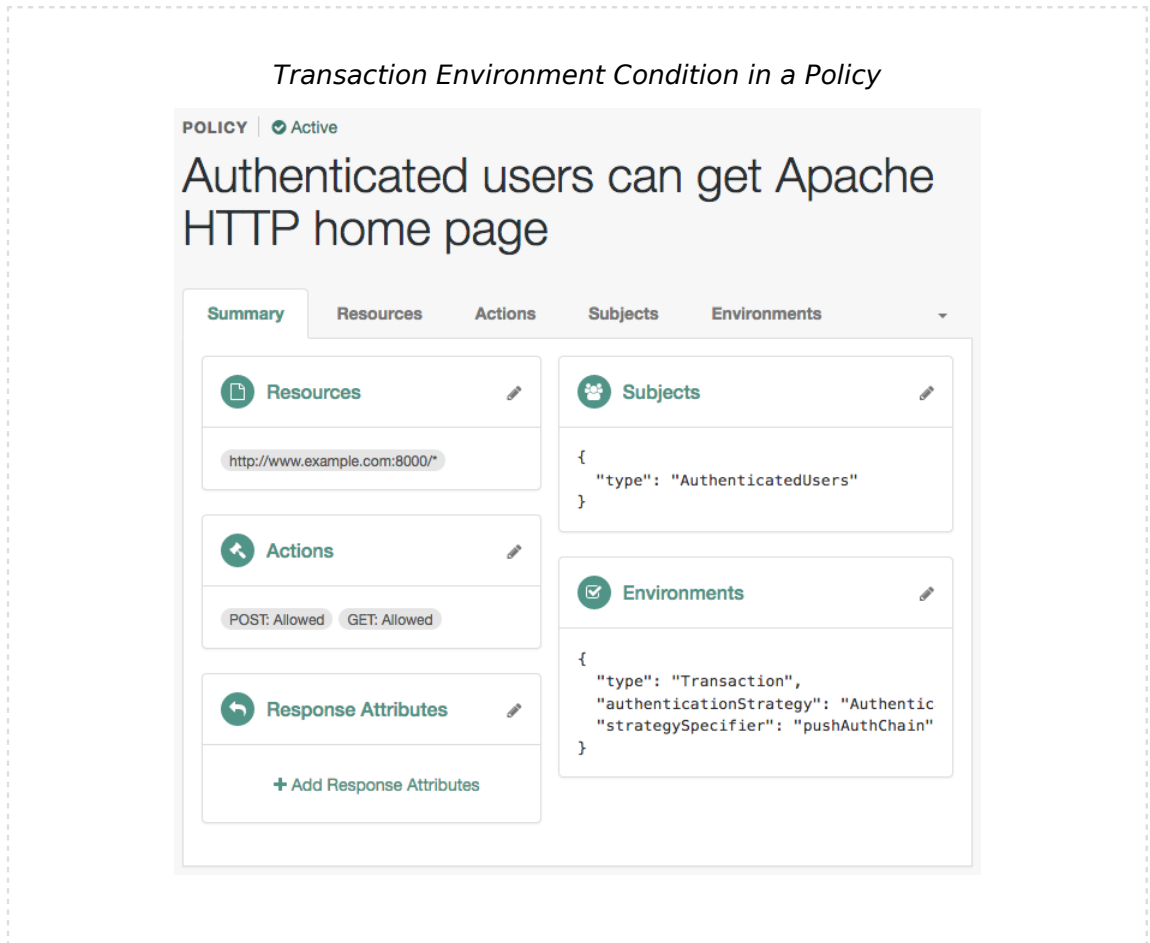
- e. In the Strategy Specifier field, enter the name of the push authorization chain created earlier, for example, `pushAuthChain`.

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:*



4. Choose one of the following options to demo transactional authorization:

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

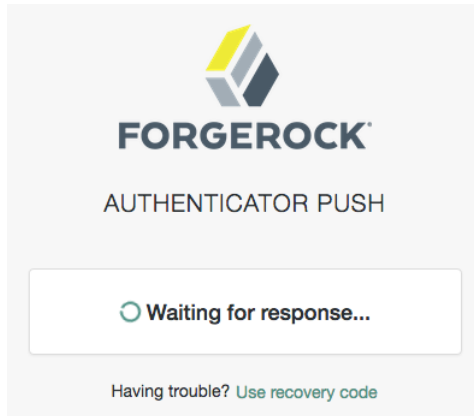
To Perform Transactional Authorization with the AM Console

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>.

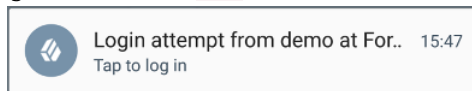
The web agent will redirect the browser to the AM login screen.

2. Log in to AM as user `demo` with password `Ch4ng31t`.

AM will display the authenticator push page:

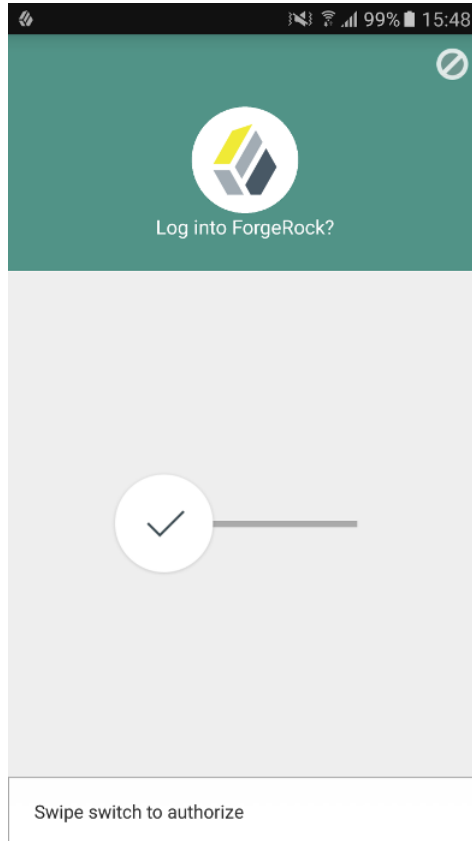


The mobile device that was registered to the `demo` user will receive a push notification message:



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.

To Perform Transactional Authorization with the AM REST API

1. Obtain a session token from AM for user `demo` with password `Ch4ng31t:`

```
$ curl \
--request POST \
--header "Content-Type: application/json" \
--header "X-OpenAM-Username: demo" \
--header "X-OpenAM-Password: Ch4ng31t" \
--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": "/"
}
```

- 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 "Authenticating (REST)" in the *Authentication and Single Sign-On Guide*.

```
$ 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*"
  },
  "application": "iPlanetAMWebAgentService"
}' \
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
}
```

Tip

Enter the name of your policy set in the `application` parameter if you are not creating policies in the default, `iPlanetAMWebAgentService`.

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 *Sessions 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 \
--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=%3CAdvices%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%2FAdvices%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. Reevaluate 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.

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 8

Dynamic OAuth 2.0 Authorization

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.

+ *Example Use Case*

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 their 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.

+ *How Does Dynamic OAuth 2.0 Authorization Work?*

Granting or Denying OAuth2 2.0 Scopes Dynamically

OAuth 2.0 Client

Request

```
$ curl \
--request POST \
--user "myClient:forgerock" \
--data "grant_type=password&username=demo&
password=forgerock&scope=profile+email" \
https://openam.example.com:8443/openam/oauth2/access_token
```

Response

```
{
  "access_token": "bBHPAs6ycNWCbspk-IW6Cnu8uk0",
  "scope": "profile",
  "token_type": "Bearer",
  "expires_in": 3599
}
```

AM

Realms

Policy Sets

Policies

MyPolicy

Resources	profile	<input checked="" type="checkbox"/>
Actions	ALLOW	<input checked="" type="checkbox"/>
Subjects	All Authenticated Users	<input checked="" type="checkbox"/>
Conditions	Between 09:00 and 17:00	<input checked="" type="checkbox"/>

Resource Types

OAuth2 Scope

Patterns
Actions
GRANT

When issuing access tokens, AM deduces consent status based on a 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 "*OAuth 2.0 Grant Flows*" in the *OAuth 2.0 Guide* and "*OpenID Connect Grant Flows*" in the *OpenID Connect 1.0 Guide*.

OAuth 2.0 authorization is implemented by using the **OAuth2 Scope** resource type, which must always be configured in the **Default OAuth2 Scopes** policy set.

Related information:

- "Configuring Policies"

Writing policies for OAuth 2.0 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.

The following procedures set up an OpenID Connect environment and demonstrate how OAuth 2.0 authorization works:

- "To Configure Access Management for the Examples"
- "To Test OAuth 2.0 Policies in a Non-Interactive Flow"
- "To Test OAuth 2.0 Policies in an Interactive OpenID Connect Flow"

To Configure Access Management for the Examples

1. Configure an instance of the OAuth 2.0 provider. See "Authorization Server Configuration" in the *OAuth 2.0 Guide*.

Ensure that:

- Use Policy Engine for Scope Decisions is enabled
 - Response Type Plugins is configured for `token` and `id_token`
 - Grant Types is configured for `Resource Owner Password Credentials` and `Implicit`
2. 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`
 3. Navigate to Realms > *Realm Name* > Authorization > Policy Sets, and select Default OAuth2 Scopes Policy Set.
 4. 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.

5. On the Actions tab, add the `GRANT` action, and set the default state to `Allow`. Save your changes.
6. On the Subjects tab, specify the subject condition `All of... Authenticated Users`. Save your changes.
7. (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 `Ch4ng31t`.

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 cannot therefore 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 `Ch4ng31t`
 - The requested scopes, `email` and `profile`

```
$ curl --request POST \  
--data "grant_type=password" \  
--data "username=demo" \  
--data "password=Ch4ng31t" \  
--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 can therefore 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:443/callback`

For example:

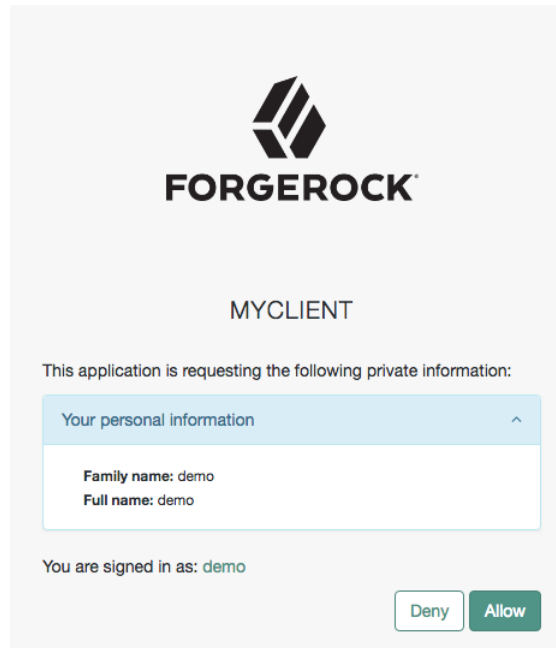
```
https://openam.example.com:8443/openam/oauth2/authorize?nonce=123&state=456&scope=openid+email+profile&response_type=id_token&client_id=myClient&redirect_uri=https://www.example.com:443/callback
```

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.

OAuth 2.0 Scopes Policy Script API Functionality

In addition to the functionality provided by "Accessing HTTP Services" in the *Getting Started with Scripting* and "Debug Logging" in the *Getting Started with Scripting*, scripted OAuth 2.0 policy condition scripts can access some environment properties relating to the authorization request.

This information can then be returned as needed in the response to an authorization request:

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 an <code>environment</code> map with a single entry: <pre> "environment": { "clientId": ["MyOAuth2Client"] } </pre>

The following JavaScript writes the ID of the OAuth 2.0 client to the debug log, and then authorizes the request:

```

logger.message("Client ID: " + environment.get("clientId"));
authorized=true;
  
```

Chapter 9

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 AM \(All versions\)?](#) in the *Knowledge Base*.

Get a local clone so that you can try the sample on your system. You will find the relevant files under the `/path/to/openam-samples-external/policy-evaluation-plugin` directory.

+ Files in the Sample

`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 AM \(All versions\)?](#) in the *Knowledge Base*.

2. When the build is complete, copy the `policy-evaluation-plugin-7.0.2.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 4 so that only users with a username length of 4 characters or greater can access the resource.

With the policy in place, authenticate both as a user who can request policy decisions and also as a user trying to access a resource. 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: Ch4ng31t" \
--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":"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: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 ...
```

Chapter 10

Scripting a Policy Condition

You can use scripts to tailor the actions AM does as part of policy evaluation to your environment.

AM includes a sample policy condition script that demonstrates how to access a user's profile information, use that information in HTTP calls, and make a policy decision based on the outcome.

To examine the contents of the sample policy condition script in the AM console, navigate to Realms > Top Level Realm > Scripts, and then select *Scripted Policy Condition*.

Related information:

- [Getting Started with Scripting](#)
- ["Policy Condition Script API Functionality"](#)

Preparing AM to use Scripted Policy Conditions

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.

Tasks:

- ["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 Debug Logging for Scripted Policy Conditions"](#)

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.

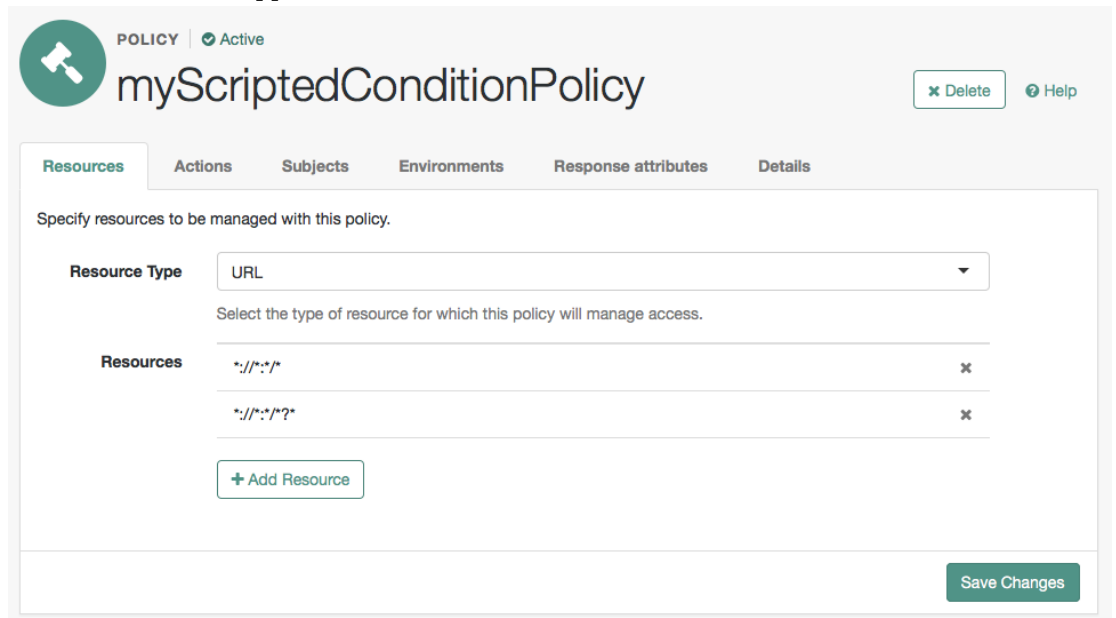
The ***://*:*/*** resource appears in the Resources field.

- 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:



The screenshot shows the configuration page for a policy named "myScriptedConditionPolicy". The page has a header with a logo, the policy name, and a status indicator "POLICY | Active". There are "Delete" and "Help" buttons in the top right. Below the header is a navigation bar with tabs: "Resources" (selected), "Actions", "Subjects", "Environments", "Response attributes", and "Details". The main content area is titled "Specify resources to be managed with this policy." and contains a form with the following elements:

- Resource Type:** A dropdown menu with "URL" selected.
- Resources:** A list of resource patterns. Two patterns are visible: ***://*:*/*** and ***://*:*/?***. Each pattern has a small "x" icon to its right, indicating it can be removed.
- + Add Resource:** A button to add a new resource.
- Save Changes:** A button at the bottom right of the form.

- 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 'Actions' tab for a policy named 'myScriptedConditionPolicy'. The policy is currently 'Active'. The interface includes a header with the policy name and status, navigation tabs for Resources, Actions, Subjects, Environments, Response attributes, and Details, and a table for selecting actions and their default states.

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

Buttons: Add an Action, Save Changes, Delete, Help

- 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 'POLICY | Active' configuration page for 'myScriptedConditionPolicy'. The 'Subjects' tab is selected, and the instruction 'Specify the subject conditions to which the policy applies.' is displayed. A dropdown menu is set to 'All of...'. A single subject condition is added: 'Type: Authenticated Users'. Below the list are buttons for '+ Add a Subject Condition' and '+ Add a Logical Operator'. A 'Save Changes' button is located at the bottom right.

- 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:

The screenshot shows the 'Environments' tab of a policy configuration page. At the top left, there is a green circular icon with a white hammer and the text 'POLICY | Active'. The main title is 'myScriptedConditionPolicy'. On the top right, there are 'Delete' and 'Help' buttons. Below the title, there are tabs for 'Resources', 'Actions', 'Subjects', 'Environments' (selected), 'Response attributes', and 'Details'. The main content area is titled 'Specify the environment conditions to which the policy applies.' It features a dropdown menu set to 'All of...'. Below this is a table with two columns: 'Type' and 'Script Name'. The table contains one row with 'Script' and 'Scripted Policy Condition'. To the right of the table are edit and delete icons. At the bottom of the main area are two buttons: '+ Add an Environment Condition' and '+ Add a Logical Operator'. A 'Save Changes' button is located at the bottom right of the entire form.

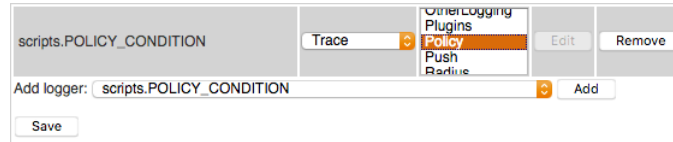
- vi. Select Save Changes.
- f. No additional configuration is required in the Response Attributes or Details tabs.

To Enable Debug Logging for Scripted Policy Conditions

This procedure shows how to enable trace-level debug logging for scripted policy conditions, so that logger output from the default policy condition script is recorded.

1. Ensure that the script containing the debug output you want to capture has executed at least once.
This will create the required logger that can be configured in the next steps.
2. Log in as the AM administrator, `amAdmin`.
3. Visit the `Logback.jsp` page, for example: `https://openam.example.com:8443/openam/Logback.jsp`.
4. In the Add logger drop-down list, select the `scripts.POLICY_CONDITION` class, and then click Add.
5. Locate the newly added `scripts.POLICY_CONDITION` class in the Logger list, and then click Edit.
 - a. From the Level drop-down list, choose the debug level required. In this example, select `Trace`.
 - b. From the Appender list, choose the location for the debug output. In this example, select `Policy`.

The result will resemble the following:



- At the bottom of the `Logback.jsp` page, click **Save**.

Trace-level debug logging is now enabled for scripted policy conditions, with script output appearing in the `/path/to/openam/var/debug/Policy` debug log file.

Changes made in the `Logback.jsp` page are not persisted after rebooting AM or the container in which it runs. Repeat the steps above to debug scripted decision nodes.

For more information on configuring debug logging, see "*Debug Logging*" in the *Maintenance Guide*.

Trying 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

- 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: Ch4ng31t" \
--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": "/"
}
```

- 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": {}
}
```

Policy Condition Script API Functionality

In addition to the functionality provided by "Accessing HTTP Services" in the *Getting Started with Scripting* and "Debug Logging" in the *Getting Started with Scripting*, scripted policy condition scripts can access the authorization state of a request, the information pertaining a session, and the user profile's data.

This information can then be returned as needed in the response to an authorization request.

Accessing Authorization State

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

Authorization State Objects

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

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/XUI/?realm=alpha#login/
Host	198.51.100.1
HostName	openam.example.com
Locale	en_US
Organization	dc=openam,dc=forgerock,dc=org
Principal	uid=amAdmin,ou=People,dc=openam,dc=forgerock,dc=org
Principals	amAdmin
Service	ldapService
successURL	/openam/console
sun.am.UniversalIdentifier	uid=amAdmin,ou=People,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>

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.