



User-Managed Access (UMA) 2.0 Guide

/ ForgeRock Access Management 7.1.4

Latest update: 7.1.4

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 configuring and using User-Managed Access (UMA) 2.0 features 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.

© Copyright 2010-2020 ForgeRock, Inc. All rights reserved. ForgeRock is a registered trademark of ForgeRock, Inc. Other marks appearing herein may be trademarks of their respective owners.

This product or document is protected by copyright and distributed under licenses restricting its use, copying, and distribution. No part of this product or document may be reproduced in any form by any means without prior written authorization of ForgeRock and its licensors, if any.

DOCUMENTATION IS PROVIDED "AS IS" AND ALL EXPRESSED OR IMPLIED CONDITIONS, REPRESENTATIONS, AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

DejaVu Fonts

Bitstream Vera Fonts Copyright

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

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

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

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

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

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

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

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

Arev Fonts Copyright

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

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

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

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

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

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

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

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

FontAwesome Copyright

Copyright (c) 2017 by Dave Gandy, <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. First Steps With UMA	1
2. The UMA Guide Example	5
The Postman Collection	7
Manual Steps	8
What's Next?	17
3. AM as the UMA Authorization Server	19
Supported Specifications	19
Deployment Considerations	20
UMA Discovery	20
4. Configuring UMA Stores	21
Preparing External UMA Data Stores	21
Configuring UMA Stores in AM	28
5. Configuring the UMA Actors	31
6. Registering and Protecting Resources	33
How to Manage UMA Resources	33
How to Manage UMA Policies	39
7. Managing UMA Labels	52
How to Manage UMA User and Favorite Labels	53
8. The UMA Grant Flow	58
9. Extending UMA	66
Resource Registration Extension Point	66
Permission Request Extension Point	67
Authorization Request Extension Point	67
Resource Sharing Extension Point	68
10. UMA Endpoints	70
/uma/resource_set	70
/uma/permission_request	71
/json/users/{user}/uma/policies	72
/json/users/{user}/oauth2/resources/labels	74
/json/users/{user}/uma/pendingrequests	75
/uma/.well-known/uma2-configuration	78
11. Reference	81
UMA Configuration Reference	81
Glossary	87

Overview

This guide covers configuration, concepts and procedures for working with the User-Managed Access (UMA) 2.0 features in ForgeRock Access Management.

ForgeRock Access Management supports the User-Managed Access (UMA) 2.0 Grant for OAuth 2.0 Authorization and Federated Authorization for User-Managed Access (UMA) 2.0 specifications. Both specifications define UMA 2.0.

Quick Start

 <p>About UMA</p> <p>Learn about UMA 2.0, an OAuth 2.0 extension that let users manage access to their resources.</p>	 <p>The UMA Guide Example</p> <p>Read about a real-life UMA example, and go through different procedures to configure AM and to run the different UMA flows.</p>
 <p>The UMA Grant Flow</p> <p>Extend OAuth 2.0 and OpenID Connect with the UMA grant flow.</p>	 <p>Extension Points</p> <p>Customize UMA services using the extension points Access Management provides, when built-in functionality does not fit your deployment.</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

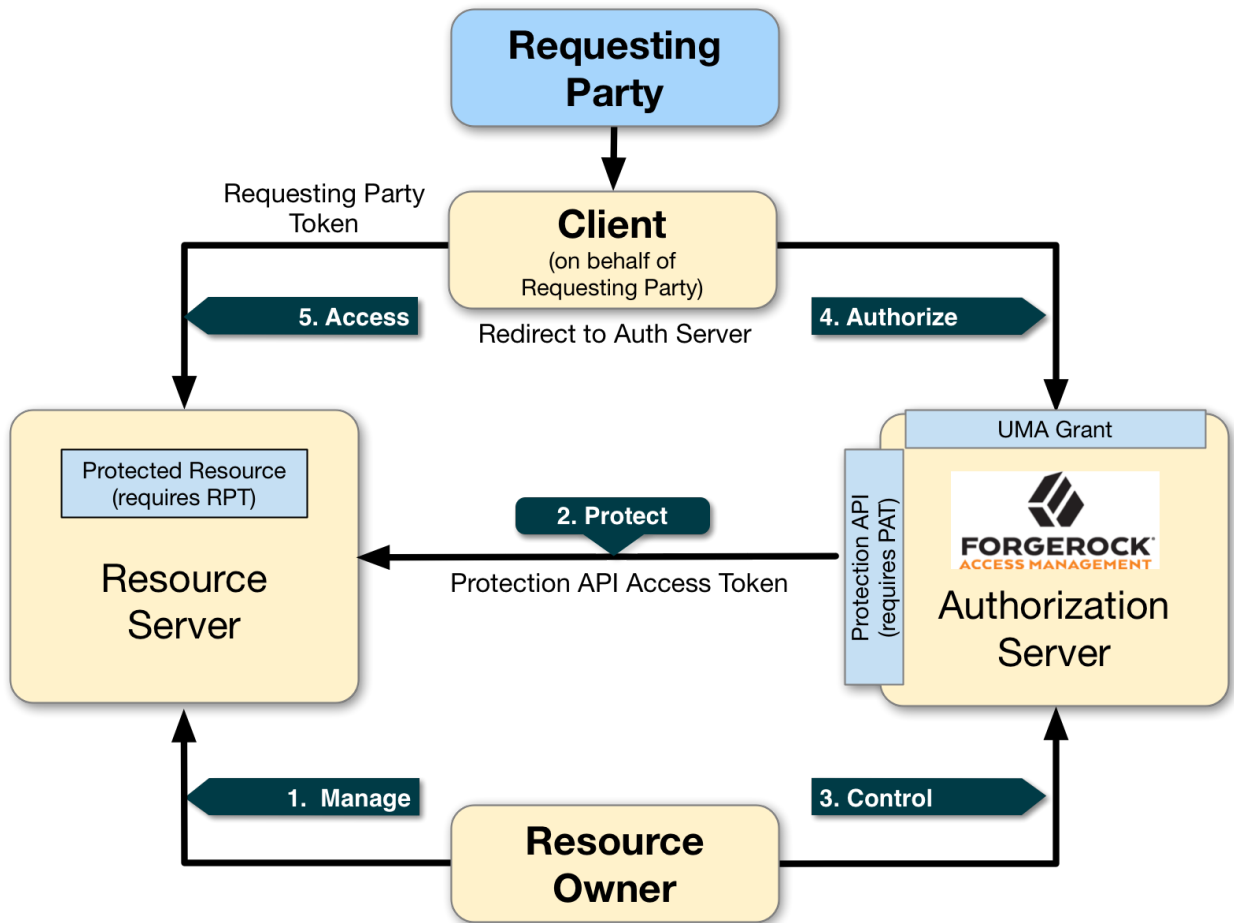
First Steps With UMA

UMA 2.0 is a lightweight access control protocol that defines a centralized workflow to let an entity (user or corporation) manage access to their resources.

It extends the OAuth 2.0 protocol and gives resource owners granular management of their protected resources by creating authorization policies on a centralized authorization server, such as AM.

UMA 2.0 uses the OAuth 2.0 actors in extended ways and introduces the requesting party as a new actor:

Actors and Actions in the UMA 2.0 Workflow



+ UMA Actors Explained

Resource Owner

The resource owner is a user or legal entity that is capable of granting access to a protected resource.

Client

The client is an application that is capable of making requests with the resource owner's authorization and on the requesting party's behalf.

Resource Server

The resource server hosts resources on a resource owner's behalf, and is capable of accepting and responding to requests for protected resources.

You can configure ForgeRock Identity Gateway 6 or later as an UMA resource server. For more information, see *ForgeRock Identity Gateway 7.1 Gateway Guide*.

Authorization Server

The authorization server protects resources hosted on a resource server on behalf of resource owners.

You can set up AM to fully function as an authorization server in an UMA 2.0 deployment. AM provides an UMA provider service, an UMA grant type handler, and endpoints for resource registration, permission ticket generation, and UMA token introspection. AM also uses its OAuth Provider Service to generate OIDC ID tokens, and to provide claim tokens and its policy engine for UMA resource management.

Requesting Party

The requesting party is a user or legal entity that uses a client to access a protected resource. The requesting party may or may not be the same as the resource owner. This actor is specific to the UMA protocol.

+ *UMA Actions Explained*

1. Manage

The resource owner manages their resources on the resource server.

2. Protect

The authorization server and the resource server are loosely coupled elements in an UMA deployment. Because of this, the authorization server can onboard multiple resource servers in any domain. To onboard multiple resource servers, the authorization server exposes a protection API that provides resource registration, permission tickets, and token inspection endpoints to bind the resource server and authorization server.

The API endpoints are protected by a PAT (*Protection API Token*) —an OAuth 2.0 token with a specific scope of `uma_protection`—which establishes a trust relationship between the two components.

For more information, see `"/uma/resource_set"`.

3. Control

The resource owner controls who has access to their registered resources by creating policies on the authorization server. This allows the resource owner to grant consent asynchronously,

rather than at resource request time. As a result, the requesting party can access data using an RPT (*Requesting party token*).

For more information, see `"/json/users/{user}/uma/policies"`.

4. Authorize

The client, acting on behalf of the requesting party, uses the authorization server's UMA Grant Flow to acquire an RPT (*Requesting party token*), which is a token unique to the requesting party, client, authorization server, resource server, and resource owner. The requesting party and the resource owner can interact with their applications at any time they want. In some cases, the requesting party and the resource owner may be the same entity.

This interaction allows for party-to-party data sharing and access authorization delegation. The resource owner can grant consent by policy using the authorization server, rather than issue a token at runtime; thus, allowing for the asynchronous granting of consent.

5. Access

The client presents the RPT to the resource server, which verifies its validity with the authorization server. If the token is valid and contains the sufficient permissions, the resource server returns the protected resource to the requesting party.

ForgeRock provides an example use case that will help you configure and demo the UMA flow.

Once you are familiar with the actors and the flow, read the rest of the topics to learn how to configure UMA in production environments, and the functionality that the AM REST APIs have to offer.

For additional UMA use cases, see:

- [Case Studies](#)
- [The Kantara Initiative](#)

Chapter 2

The UMA Guide Example

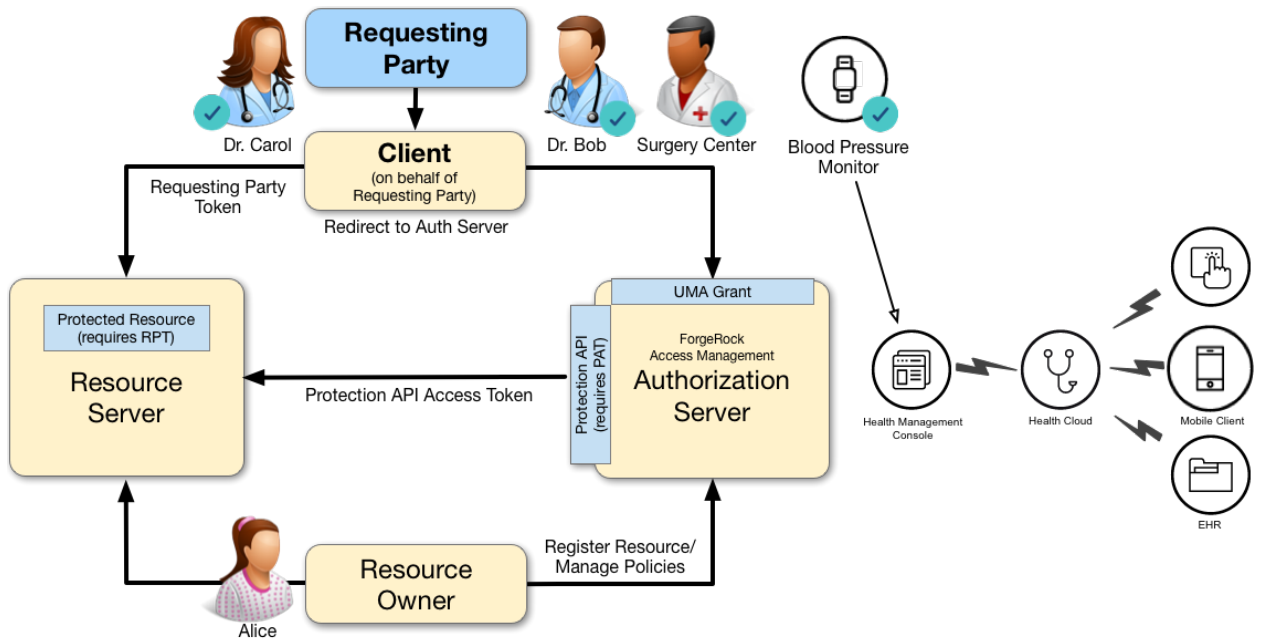
A resource owner, Alice, is a patient who plans to undergo a medical procedure at a surgery center. Dr. Bob is a specialist surgeon who needs read access (for example, `view` scope) to Alice's electronic health records in order to operate, and write access (for example, `comment` scope) in order to add new entries related to the surgery. These records are a resource whose contents have built up over time and to which Alice's regular physician, Dr. Carol, has access already.

Alice, or some party representing Alice, registers her medical health records and sets up permissions using authorization policies, allowing Dr. Bob and Dr. Carol access to her health data. On an online healthcare application, Alice can easily grant consent by clicking a "Share" button to her data, or decline access by clicking a "Deny" button.

UMA also solves managed consent for IoT deployments. For example, Alice will need to be monitored after her operation. Dr. Bob prescribes a smart medical device for Alice, such as a clinical-grade blood pressure monitor, which must be registered by the resource server to place it under the authorization server's protection. The blood pressure monitor sends data to a server that aggregates and transmits the data to external devices, allowing Dr. Bob and Dr. Carol access to Alice's data on their tablets or mobile apps.

AM supports a one-to-many policy that can be shared with many entities, not just targeting a single requesting party. Thus, Alice is able to share her data with Dr. Bob, Dr. Carol, as well as with the clinical and operational employees at the surgery center.

UMA 2.0 Use Case Health App



In this example, you will learn how to:

- Create an example configuration in AM.
- Register and protect a resource as the resource owner.
- Request access to the resource as the requesting party, by partaking in the UMA grant flow.

Important

The example will not give you insight on the more technical aspects of UMA. It is only meant to demo the concepts and the flows.

Read the rest of the topics on this guide to learn how to configure UMA in production environments, and for more in-depth explanations regarding how to use the REST APIs.

Decide How to Proceed with the Example

You can configure UMA manually by following the steps in a series of procedures, and then run the flow examples using your preferred REST client, or you can use the ForgeRock UMA Postman collection:



The UMA Postman Collection

Download a Postman collection to configure AM and to run the example scenarios.

If this is your first time with UMA, you may want to read through the manual steps as you run the collection to better understand the calls being made.



Manual Steps

Configure AM and run the example scenarios manually.

The Postman Collection

ForgeRock provides an UMA Postman collection to help configure AM for the example, and to run each of the steps in the procedures contained within. It also contains REST calls to the supporting UMA endpoints that you can run as part of the example.

1. Download and install Postman.
2. Download the ForgeRock UMA Postman Collection.
3. Import the collection in Postman:
 - a. Go to File > Import ... > Upload Files.
 - b. Select the collection you downloaded, and click Open. Then, click Import.
4. Configure the collection's variables to suit your environment:
 - a. In Postman, on the Collections tab, select the ForgeRock UMA Collection. Click on the ... button, and then on Edit.

The Edit Collection page appears.

- b. Click on the Variables tab, and change at least the value of the following variables:
 - `URL_base`
 - `admin_password`
- c. Click Update to save your changes.

You are ready to start running the collection.

The collection is divided into the following folders:

- `Prerequisites`, containing REST calls equivalent to the configuration tasks in "Configuring AM for UMA".

If you run the calls in this folder, do not perform the equivalent manual steps to avoid object conflicts.

- **UMA Registration Flow**, containing the REST calls in "Registering and Protecting an UMA Resource".
- **UMA Grant Flow**, containing the REST calls in "Requesting Access to a Resource".
- **Managing UMA Resources**, containing REST calls related to the "/uma/resource_set" endpoint.
- **Managing UMA User Labels**, containing REST calls related to the "How to Manage UMA User and Favorite Labels" endpoint.
- **Managing UMA Policies**, containing REST calls related to the "/json/users/{user}/uma/policies" endpoint.

Manual Steps

The high-level steps to run the example manually are as follows:

- Configure AM and the UMA actors.
- Register and protect an UMA resource as the resource owner.
- Request access to an UMA resource as the requesting party.

Configuring AM for UMA

To configure the example UMA deployment to test the procedures in this documentation, perform the following tasks:

Task	Description
Create an OAuth 2.0 provider service with a basic configuration.	"To Create a Basic OAuth 2.0 Provider"
Create a UMA provider service with a basic configuration.	"To Create a Basic UMA Provider Service"
Create an UMA client.	"To Create an Example UMA Client"
Create an UMA resource server client.	"To Create an Example UMA Resource Server Client"
Create an OAuth2/OpenID Connect provider.	"To Create a Basic OAuth 2.0 Provider"
Create a resource owner.	"To Create an Example UMA Resource Owner"
Create a requesting party.	"To Create an Example Requesting Party"

To Create a Basic OAuth 2.0 Provider

This procedure shows you how to configure a basic OAuth 2.0 provider suitable for the example.

1. In the AM console, go to Realms > Top Level Realm > Services.
2. Create an OAuth 2.0 provider, or configure it if one is already created.
 - If an OAuth 2.0 provider is already created, click on it.
 - If there is no OAuth 2.0 provider, add one:
 1. Click Add a Service.
 2. On the drop-down menu, select the OAuth2 Provider service. Then, click the Create button without filling any other field.

The OAuth 2.0 provider page appears.

3. Go to the Advanced tab.
4. Ensure that the following fields are configured:
 - **Grant Types:** At least, `UMA` and `Resource Owner Password Credentials` must be configured.
 - **Response Type Plugins:** At least, `id_token|org.forgerock.openidconnect.IdTokenResponseTypeHandler` and `token|org.forgerock.oauth2.core.TokenResponseTypeHandler` must be configured.

To Create a Basic UMA Provider Service

- In the AM console, go to Realms > Top Level Realm > Services, and add an UMA Provider service.

The default configuration suitable for the example. For production environments, see "*Configuring the UMA Actors*".

To Create an Example UMA Client

Create a profile for the example UMA client in AM. Since this client will partake in the UMA grant, which requires an ID token, you need to configure OpenID Connect properties on it, too:

1. In the AM console, go to Realms > Top Level Realm > Applications > OAuth 2.0 > Clients.
2. Click Add Client, and enter the following values:
 - **Client ID :** `UmaClient`
 - **Client secret:** `password`

- **Redirection URIs:** `redirection URI`. For this example, leave it blank.
 - **Scope(s):** `view openid comment download`
You will need to enter `view`, press Enter, and then enter `openid`, and so on.
 - **Default Scope(s):** For this example, leave it blank.
3. Click Create. The page for the client appears.
 4. In the Advanced tab, enter the following values:
 - **Grant Types:** `UMA, Resource Owner Password Credentials`
You will need to enter `UMA`, press Enter, and then enter `Resource Owner Password Credentials`.

This example uses the Resource Owner Password Credentials grant, but you can use any grant type except the Client Credentials one. This is because this client acts on behalf of the requesting party, which needs to authenticate with their identities as part of the flow.
 - **Token Endpoint Authentication Method:** `client secret post`

Confidential OpenID Connect clients can use several methods to authenticate, but this example uses `client secret post` for clarity. For more information, see "*OpenID Connect Client Authentication*" in the *OpenID Connect 1.0 Guide*.
 5. Save your changes.

To Create an Example UMA Resource Server Client

Create a profile for the example resource server client in AM. This client only needs to obtain PATs (*Protection API access tokens*) and therefore, it does not require any OpenID Connect-related configuration.

1. In the AM console, go to Realms > Top Level Realm > Applications > OAuth 2.0 > Clients.
2. Click Add Client, and enter the following values:
 - **Client ID:** `Uma-Resource-Server`
 - **Client secret:** `password`
 - **Redirection URIs:** `redirection URI`. For this example, leave it blank.
 - **Scope(s):** `uma_protection`
 - **Default Scope(s):** For this example, leave it blank.
3. Click Create. The page for the client appears.
4. In the Advanced tab, enter the following values:

- **Grant Types:** Resource Owner Password Credentials

This example uses the Resource Owner Password Credentials grant, but you can use any grant type except the Client Credentials one. This is because the resource owner must authenticate to the resource server.

5. Save your changes.

To Create an Example UMA Resource Owner

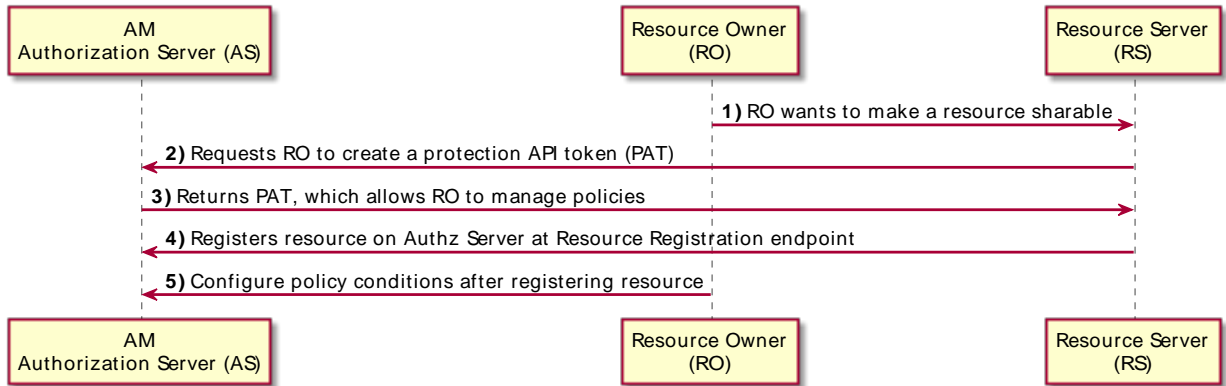
1. In the AM console, go to Realms > Top Level Realm > Identities.
2. Click New, and create a new requesting party. This example uses the following values:
 - **ID:** alice
 - **First Name:** Alice
 - **Last Name:** Resource-Owner
 - **Full Name:** Alice Resource-Owner
 - **Password:** Ch4ng31t
3. Click Create.

To Create an Example Requesting Party

1. In the AM console, go to Realms > Top Level Realm > Identities.
2. Click New, and create a new requesting party. This example uses the following values:
 - **ID:** bob
 - **First Name:** Bob
 - **Last Name:** Requesting-Party
 - **Full Name:** Bob Requesting-Party
 - **Password:** Ch4ng31t
3. Click Create.

Registering and Protecting an UMA Resource

Alice, the resource owner, has been provided with the latest results of their medical procedure. To share the results with their doctors, Alice registers them online in their surgery's resource server, and lets Dr. Bob view them and comment on them:



+ *Flow Explained*

- A resource owner wants to make a resource sharable and sends a request to the resource server (labeled **1** in the diagram).
- The resource server requires the resource owner to acquire a PAT (*Protection API access token*) on the authorization server (**2**).
- The authorization server returns a PAT, which lets the resource owner register resources and manage policies (**3**).
- The resource server registers the resource on the authorization server at the resource registration endpoint (**4**).
- The resource owner creates a policy after registering the resource (**5**).

Perform the steps in the following procedure to register an example resource and protect it as Alice:

To Register and Protect a Resource

1. Alice's client application requests a PAT (*Protection API Token*) on their behalf. The client application will use it as an authorization token to register Alice's medical records, which already are stored in their surgery's resource server, in AM.

The following example uses the Resource Owner Password Credentials grant:


```
$ curl \
--request POST \
--data 'grant_type=password' \
--data 'scope=uma_protection' \
--data 'username=alice' \
--data 'password=Ch4ng31t' \
--data 'client_id=UMA-Resource-Server' \
--data 'client_secret=password' \
https://openam.example.com:8443/openam/oauth2/realms/root/access_token
{
  "access_token": "057ad16f-7dba-4049-9f34-e609d230d43a",
  "refresh_token": "340f82a4-9aa9-471c-ac42-f0ca1809c82b",
  "scope": "uma_protection",
  "token_type": "Bearer",
  "expires_in": 4999
}
```

The value returned in the `access_token` object is the PAT.

- Alice registers their medical records in AM as an UMA resource. Their client application must use the PAT token obtained before as authorization:

```
$ curl -X POST \
--header 'authorization: Bearer 057ad16f-7dba-4049-9f34-e609d230d43a' \ ❶
--header 'cache-control: no-cache' \
--header 'content-type: application/json' \
--data '{
  "resource_scopes": [
    "view", "comment", "download" ❷
  ],
  "name": "my resource 106",
  "type": "type"
}' \
https://openam.example.com:8443/openam/uma/realms/root/resource_set
{
  "_id": "0d7790de-9066-4bb6-8e81-25b6f9d0b8853", ❸
  "user_access_policy_uri": "https://openam.example.com:8443/openam/XUI/?realm=#uma/
share/0d7790de-9066-4bb6-8e81-25b6f9d0b8853"
}
```

- Use the PAT previously acquired on behalf of Alice.
- A set of actions or permissions that the resource supports. Since UMA is an OAuth 2.0/OpenID Connect extension, the permissions are in the form of scopes.
- The value returned in the `_id` property is the ID of the new UMA resource.

The resource is ready. Now Alice must create a policy to share it with Dr. Bob, specifying which actions Dr. Bob can do on the resource.

- Alice logs in to AM, the UMA authorization sever:

```
$ curl \
--request POST \
--header "Content-Type: application/json" \
--header "X-OpenAM-Username: alice" \
--header "X-OpenAM-Password: Ch4ng31t" \
--header "Accept-API-Version: resource=2.0, protocol=1.0" \
'https://openam.example.com:8443/openam/json/realms/root/realms/alpha/authenticate'
{
  "tokenId": "AQIC5wM2LY4S...Q4MTE4NTA2*",
  "successUrl": "/openam/console",
  "realm": "/"
}
```

The value returned in the `tokenId` object is Alice's session token.

- Using their client application, Alice creates a policy to share the newly created resource with Dr. Bob:

```
$ curl \
--request PUT \
--header "Accept-API-Version: resource=1.0" \
--header "Cache-Control: no-cache" \
--header "Content-Type: application/json" \
--header "iPlanetDirectoryPro: AQIC5wM2LY4S...Q4MTE4NTA2*" \
--header "If-None-Match: *" \
--data '{
  "policyId": "0d7790de-9066-4bb6-8e81-25b6f9d0b8853",
  "permissions":
  [
    {
      "subject": "b@ob",
      "scopes": [
        "view",
        "comment"
      ]
    }
  ]
}' \
https://openam.example.com:8443/openam/json/realms/root/users/alice/uma/policies/0d7790de-9066-4bb6-8e81-25b6f9d0b8853
{
  "_id": "0d7790de-9066-4bb6-8e81-25b6f9d0b8853",
  "_rev": "-1985914901"
}
```

- Use Alice's session token. Administrative users such as `amAdmin` cannot create UMA resource policies on behalf of a resource owner.
- Use the ID of the UMA resource.
- A subset of the permissions or scopes available for this resource. The requesting party, `"subject": "bob"`, can only view or comment on the resource, but not download it.
- Use the ID of the UMA resource, appended at the end of the URL.

The resource is now shared with Dr. Bob, who has permission to **view** and **comment** on Alice's records.

Requesting Access to a Resource

When the requesting party, Dr. Bob, needs to access Alice's records, he uses an application in their computer. The application makes a call to AM to request permission to see Alice's records. Once granted, the application can show that permission token to the server that stores Alice's records.

This, in a nutshell, is the UMA Grant flow.

Perform the steps in the following procedure to request access to Alice's records as Dr. Bob's client application:

To Request Access to a Resource

1. Acquire a PAT (*Protection API Token*) on behalf of Alice. We will use it later to authorize a permission ticket for Dr. Bob.

The following example uses the Resource Owner Password Credentials grant:

```
$ curl \
--request POST \
--data 'grant_type=password' \
--data 'scope=uma_protection' \
--data 'username=alice' \
--data 'password=Ch4ng31t' \
--data 'client_id=UMA-Resource-Server' \
--data 'client_secret=password' \
https://openam.example.com:8443/openam/oauth2/realms/root/access_token
{
  "access_token": "057ad16f-7dba-4049-9f34-e609d230d43a",
  "refresh_token": "340f82a4-9aa9-471c-ac42-f0ca1809c82b",
  "scope": "uma_protection",
  "token_type": "Bearer",
  "expires_in": 4999
}
```

The value returned in **access_token** is the PAT.

2. Create a permission ticket for Dr. Bob's client application.

Permission tickets let requesting parties request access to a particular resource. In this case, Dr. Bob's client application needs a permission ticket to access Alice's medical records.

```
$ curl -X POST \
--header 'authorization: Bearer 057ad16f-7dba-4049-9f34-e609d230d43a' \ ❶
--header 'cache-control: no-cache' \
--header 'content-type: application/json' \
--data '[
  {
    "resource_id" : "ef4d750e-3831-483b-b395-c6f059b5e15d0", ❷
    "resource_scopes" : ["download"]
  }
]' \
https://openam.example.com:8443/openam/uma/realms/root/permission_request
{
  "ticket": "eyJ0eXAiOiJ...XPeJi3E" ❸
}
```

- ❶ Use the PAT previously acquired on behalf of Alice.
- ❷ The ID of the protected resource. This is the ID of the resource created in "To Register an UMA Resource (REST)".
- ❸ The permission ticket, which links Alice's identity to a resource, and to the action Dr. Bob's requesting to do.

3. Gather information about the requesting party.

To request access to Alice's records, Dr. Bob and their client application must prove their identity to the authorization server, AM. AM will issue them an ID token with their identity's details.

```
$ curl -X POST \
--data 'client_id=UmaClient' \
--data 'client_secret=password' \
--data 'grant_type=password' \
--data 'scope=openid' \
--data 'username=bob' \
--data 'password=Ch4ng31t' \
https://openam.example.com:8443/openam/oauth2/realms/root/access_token
{
  "access_token": "f09f55e5-5e9c-48fe-aeaa-d377de88e8e6",
  "refresh_token": "ee2d35f6-5819-4734-8b3e-9af77a545563",
  "scope": "openid",
  "id_token": "eyJ0eXA...FBznEB5A",
  "token_type": "Bearer",
  "expires_in": 4999
}
```

The value of the `id_token` object, which is an OpenID Connect ID token, is the UMA claim token that provides information about Dr. Bob.

4. Request an RPT (*Requesting party token*) for Dr. Bob so that they have access to Alice's records.

```
$ curl -X POST \
--data 'client_id=UmaClient' \
--data 'client_secret=password' \
--data 'grant_type=urn:ietf:params:oauth:grant-type:uma-ticket' \
--data 'ticket=eyJ0eXAiOiJ...XPeJi3E' \ ❶
--data 'claim_token=eyJ0eXA...FBznEB5A' \
--data 'claim_token_format=http://openid.net❷/specs/openid-connect-core-1_0.html#IDToken' \
https://openam.example.com:8443/openam/oauth2/realms/root/access_token
{
  "access_token": "Aw4a92ZoKsjadWKw2d4Rmcjv7DM",
  "token_type": "Bearer",
  "expires_in": 3599
}
```

- ❶ Use the permission ticket previously obtained.
- ❷ Use the claim token previously obtained.

The value of the `access_token` object is the RPT.

Presenting the RPT, Dr. Bob's client application can now access Alice's records in their surgery's resource server.

Tip

Permission tickets have a lifetime of 180 seconds by default. AM will return a message similar to the following if the ticket has expired:

```
{
  "error_description": "The provided access grant is invalid, expired, or revoked.", "error":
  "invalid_grant"
}
```

Obtain a new permission ticket and try requesting the RPT again.

What's Next?

Now you can use the knowledge you gained by running through the example to develop and configure your own UMA environment.

The following table shows the different tasks you need to perform to do so, and the associated documentation:

Task	Resources
Learn about AM as the authorization service, and configure the UMA actors.	<ul style="list-style-type: none">"<i>AM as the UMA Authorization Server</i>""<i>Configuring UMA Stores</i>""<i>Configuring the UMA Actors</i>"

Task	Resources
Learn more about resources, resource labels, and authorization policies, and their specific REST endpoints.	<ul style="list-style-type: none">• <i>"Registering and Protecting Resources"</i>• <i>"Managing UMA Labels"</i>• <i>"UMA Endpoints"</i>
Discover how to extend the default UMA implementation to suit your environment's needs.	<ul style="list-style-type: none">• <i>"Extending UMA"</i>

Chapter 3

AM as the UMA Authorization Server

In the role of the User-Managed Access (UMA) authorization server, AM grants delegated consent to a *requesting party* on behalf of the resource owner to authorize who and what can get access to their data and for how long.

Tip

Before configuring UMA in your environment, ensure you are familiar with the OAuth 2.0 standards and AM's implementation of OAuth 2.0.

Supported Specifications

AM supports the following UMA grants and specifications:

- User-Managed Access (UMA) 2.0 Grant for OAuth 2.0 Authorization

This specification defines an OAuth 2.0 extension grant, allowing a party-to-party authorization mechanism where entities in a requesting party role can access protected resources authorized by the resource owner using authorization policies. The specification also defines how a resource owner can configure an authorization server with authorization grant rules to run asynchronously with the resource server using an RPT versus granting consent at runtime.

Note

The User-Managed Access (UMA) 2.0 Grant for OAuth 2.0 Authorization specification also discusses the use of the authorization server's claims interaction endpoint for interactive claims gathering during the UMA grant flow. AM does not currently support interactive claims gathering. Claims gathering is accomplished by having the client acquire an OpenID Connect ID token.

The specification also discusses the optional issuance of a persisted claims token (PCT), which is a correlation handle issued by the authorization server, representing a set of claims collected during one authorization process to be used in later ones. AM does not currently support PCTs, because AM uses an OIDC ID token for its claims.

For more information, see "*The UMA Grant Flow*".

- Federated Authorization for User-Managed Access (UMA) 2.0

This specification defines the loosely coupled federation of the authorization process by means of multiple resource servers in different domains that communicate with the centralized authorization

server and acts on behalf of a resource owner. The authorization server can reside locally or in another domain from the resource server(s).

Tip

See the complete list of supported OpenID Connect in the *Reference* and OAuth 2.0 in the *Reference* standards.

Deployment Considerations

The UMA 2.0 process largely involves the UMA 2.0 Grant flow, in which a requesting party obtains an RPT to access the resource, and resource registration which can occur at various stages through the UMA process by the resource owner. These stages could occur at initial resource creation, when needed for policy creation, and at resource access attempt.

See the section, "Considerations Regarding Resource Registration Timing and Mechanism" in the *UMA Implementer's Guide* for information.

AM stores UMA-related data, such as resources, audit information, and labels in the configuration store by default, but this information may grow very large in environments with many users, or in environments where users own many resources.

In production environments, configure at least one external UMA store to hold UMA information.

For more information, see "*Configuring UMA Stores*".

UMA Discovery

In order to let relying parties or clients discover the URL of the UMA provider and its configuration for an end user, AM exposes the following REST endpoints:

- `/.well-known/webfinger` in the *OpenID Connect 1.0 Guide*
- `/.uma/.well-known/uma2-configuration`

Discovery relies on *WebFinger*, a protocol to discover information about people and other entities using standard HTTP methods. *WebFinger* uses *Well-Known URIs*, which defines the path prefix `/.well-known/` for the URLs defined by OpenID Connect Discovery.

Just like they do for OpenID Connect flows, relying parties need to find the right *host:port/deployment-uri* combination to locate the well-known endpoints. You must manage the redirection to AM using your proxies, load balancers, and others, such that a request to `http://www.example.com/.well-known/webfinger` reaches, for example, `https://openam.example.com:8443/openam/.well-known/webfinger`.

Chapter 4

Configuring UMA Stores

AM stores information about registered resources, audit information generated when users manage access to their protected resources, pending requests, and resource labels. AM stores these items in the configuration store by default.

For *demo and test purposes*, storing UMA information in the configuration store is sufficient and you do not need to take any additional action. To configure UMA for test purposes, see "*The UMA Guide Example*".

For *production environments*, configure at least one external store to hold UMA information. Configure more stores to separate UMA objects in high-load deployments when data tuning requirements differ.

The tasks to configure UMA stores are:

 <p>Prepare DS</p> <p>Prepare one or more DS instances to hold UMA data.</p>	 <p>Configure UMA Stores in AM</p> <p>Configure the newly-prepared DS instances in AM.</p>
---	--

Preparing External UMA Data Stores

This section explains how to prepare DS instances as external UMA data stores. You can create separate DS instances to store the following UMA-related data:

- Resources
- Resource labels
- UMA audit messages
- Pending requests

The procedure for preparing external DS instances for use by AM is similar for each of the different UMA-related data types. The steps to perform are as follows:

1. If you have not done so yet, install Directory Services.

2. As an administrative user, for example, `uid=admin`:
 - a. Create a backend and base DN entry in the external store.
 - b. Create a user account with the minimum required privileges for AM to bind to the directory server with, and access necessary data.
 - c. Apply the relevant schema to the directory. Each type of data requires a specific set of schema LDIF files to be applied.

+ LDIF Files for UMA Data Stores

Data Store	LDIF Files
Resources	<code>/path/to/tomcat/webapps/openam/WEB-INF/template/ldif/opensj/opensj_uma_resource_sets.ldif</code>
Resource labels	<code>/path/to/tomcat/webapps/openam/WEB-INF/template/ldif/opensj/opensj_uma_labels_schema.ldif</code> <code>/path/to/tomcat/webapps/openam/WEB-INF/template/ldif/opensj/opensj_uma_resource_set_labels.ldif</code>
UMA audit messages	<code>/path/to/tomcat/webapps/openam/WEB-INF/template/ldif/opensj/opensj_uma_audit.ldif</code>
Pending requests	<code>/path/to/tomcat/webapps/openam/WEB-INF/template/ldif/opensj/opensj_uma_pending_requests.ldif</code>

Tip

You can choose to use a single external DS instance with multiple backend configurations to store each of the UMA-related data types. Alter the steps in the sections below to apply the backends, schema, and administrative accounts to a single DS instance.

This section details how to set up a DS instance to store UMA resources. Repeat the procedures in the sections below with the relevant schema files to set up a DS instance to store other types of UMA-related data.

Note

Example commands throughout this section use example values for user IDs and port numbers. When running similar commands, be sure to use appropriate values for your deployment.

Installing and Configuring Directory Services for UMA Data

The following instructions show how to download, install, and set up the Directory Services server.

To Install and Configure Directory Services for UMA Data

1. Download and install Directory Services.
 - a. Generate a DS deployment key unless you already have one:

```
$ /path/to/openssl/dskeymgr create-deployment-key --deploymentKeyPassword password
```

Save the deployment key and its deployment password. Keep the key and the password safe, and keep the password secret. Use the same deployment key and password for all the servers in the same environment. For example, if you plan to deploy replication.

Replication is not covered in this example.

- b. DS does not include a UMA profile; create an example DS UMA server by providing the parameters in a single **setup** command. For example:

```
$ /path/to/openssl/setup \  
--instancePath '/path/to/openssl' \  
--serverId uma-resource-server \  
--deploymentKey deployment_key \  
--deploymentKeyPassword deployment_key_password \  
--rootUserDN uid=admin \  
--rootUserPassword str0ngAdm1nPa55word \  
--hostname uma-rs.example.com \  
--adminConnectorPort 4444 \  
--ldapPort 1389 \  
--enableStartTls \  
--ldapsPort 1636 \  
--httpsPort 8443 \  
--acceptLicense
```

For additional options for the **setup** command, see *setup - install OpenDJ server* in the *Directory Services 7.1 Tools Reference*.

DS 7 or later does not start automatically after installation; **do not** start it until after you have created a backend and added the required schemas at the end of this section.

2. When the install has completed, create backend for UMA resource data, named `umaRsStore`, and prepare for a base DN of `dc=uma-resources,dc=example,dc=com`:

```
$ /path/to/openssl/bin/dsconfig create-backend \
--hostname 'uma-rs.example.com' \
--port 4444 \
--usePkcs12TrustStore /path/to/openssl/config/keystore \
--trustStorePasswordFile /path/to/openssl/config/keystore.pin \
--backend-name umaRsStore \
--set base-dn:dc=uma-resources,dc=example,dc=com \
--set enabled:true \
--type je \
--bindDN uid=admin \
--bindPassword strongAdminPa55word \
--offline
The JE Backend was created successfully
```

3. Share the UMA store certificate with the AM container to prepare for TLS/LDAPS. UMA stores should communicate over secure connections for security reasons.

DS 7 or later is configured to require secure connections by default; therefore, share its certificate with the AM container before continuing.

+ *Sharing the DS Certificate with AM*

1. Export the DS server certificate:

```
$ keytool -exportcert \
-keystore /path/to/openssl/config/keystore \
-storepass $(cat /path/to/openssl/config/keystore.pin) \
-alias ssl-key-pair \
-rfc \
-file ds-cert.pem*
```

The default DS server certificate only has the hostname you supplied at setup time, and `localhost`, as the value of the `SubjectAlternativeName` attribute; however, certificate hostname validation is strict. Ensure that the certificate matches the hostname (or the FQDN) of the DS server before continuing.

Copy the `ds-cert.pem` file to an accessible location on the AM host.

2. Import the DS certificate into the AM truststore:

```
$ keytool \
-importcert \
-file ds-cert.pem \
-keystore /path/to/openam/security/kestores/truststore
```

For more information on configuring AM's truststore, see "Preparing a Truststore" in the *Installation Guide*.

Creating an UMA Store Base DN

Create a base DN that AM uses to store UMA directory server data by performing the following steps:

To Create an UMA Base DN

1. Create an LDIF file to add the base DN to the UMA store, and save the file as `add-uma-base-dn.ldif`:

```
dn: dc=uma-resources,dc=example,dc=com
changetype:add
objectClass: top
objectClass: domain
dc: uma-resources
```

2. Apply the LDIF file to the DS instance by using the **ldapmodify** command:

```
$ /path/to/openssl/bin/ldapmodify \
--hostname 'uma-rs.example.com' \
--port 1636 \
--useSsl \
--usePkcs12TrustStore /path/to/openssl/config/keystore \
--trustStorePasswordFile /path/to/openssl/config/keystore.pin \
--bindDN uid=admin \
--bindPassword str0ngAdm1nPa55word \
--continueOnError \
--offline \
add-uma-base-dn.ldif
# ADD operation successful for DN dc=uma-resources,dc=example,dc=com
```

Tip

If you are having trouble with the LDIF file, remove any line feeds in the ACI attributes.

Creating an UMA Store Bind Account

As a best practice, the use of the root `uid=admin` is not recommended for accessing data on the directory server. Instead, you can create a new service account, the UMA store bind account, with limited access and fewer privileges.

Create a bind account that AM uses to access the UMA directory server data by performing the following steps:

To Create and Configure an UMA Store Bind Account

1. Create an LDIF file to add the bind account to the UMA store, and save the file as `add-uma-bind-account.ldif`.

When AM connects as the bind account to store the UMA-related data, it requires read, write, persistent search, and server-side sorting access privileges. You add these privileges by setting access control instructions (ACIs) on the base distinguished name (DN) entry you created in the previous step (for example, `dc=uma-resources,dc=example,dc=com`).

The following is an example of a suitable `add-uma-bind-account.ldif` LDIF file:

```
dn: ou=admins,dc=uma-resources,dc=example,dc=com
objectclass: top
objectclass: organizationalUnit
ou: admins

dn: uid=am-uma-bind-account,ou=admins,dc=uma-resources,dc=example,dc=com
objectclass: top
objectclass: person
objectclass: organizationalPerson
objectclass: inetOrgPerson
cn: am-uma-bind-account
sn: am-uma-bind-account
uid: am-uma-bind-account
userPassword: 5up35tr0ng
aci: (targetattr="*)(version 3.0; acl "Allow CRUDQ operations"; allow (search, read, write, add, delete)(userdn = "ldap:///uid=am-uma-bind-account,ou=admins,dc=uma-resources,dc=example,dc=com");)
aci: (targetcontrol="2.16.840.1.113730.3.4.3")(version 3.0; acl "Allow persistent search"; allow (search, read)(userdn = "ldap:///uid=am-uma-bind-account,ou=admins,dc=uma-resources,dc=example,dc=com");)
aci: (targetcontrol="1.2.840.113556.1.4.473")(version 3.0; acl "Allow server-side sorting"; allow (read)(userdn = "ldap:///uid=am-uma-bind-account,ou=admins,dc=uma-resources,dc=example,dc=com");)
```

2. Apply the LDIF file to the DS instance by using the **ldapmodify** command:

```
$ /path/to/openssl/bin/ldapmodify \
--hostname 'uma-rs.example.com' \
--port 1636 \
--useSsl \
--usePkcs12TrustStore /path/to/openssl/config/keystore \
--trustStorePasswordFile /path/to/openssl/config/keystore.pin \
--bindDN uid=admin \
--bindPassword str0ngAdm1nPa55word \
--continueOnError \
--offline \
add-uma-bind-account.ldif
# ADD operation successful for DN uid=am-uma-bind-account,ou=admins,dc=uma-resources,dc=example,dc=com
```

Tip

If you are having trouble with the LDIF file, remove any line feeds in the ACI attributes.

The `am-uma-bind-account` account can now connect to the Directory Services instance. Note that you must use the full distinguished name of the account when binding, for example `uid=am-uma-bind-account,ou=admins,dc=uma-resources,dc=example,dc=com`, with the configured password, for example `5up35tr0ng`.

Creating a Schema for UMA-related Data

Once the DS installation is complete and the instance is operational, as an administrative user such as `uid=admin`, import the schema files required for UMA-related data. For information on the required schema files, see LDIF Files for UMA Data Stores.

To Apply the Schema for UMA-Related Data

Note

This procedure makes use of LDIF files that are shipped with the AM server.

1. Replace the `@SM_CONFIG_ROOT_SUFFIX@` variable in the LDIF files with the base DN you configured in the directory server. For example, `dc=uma-resources,dc=example,dc=com`.

Depending on the UMA-related data type you will store in this directory server, replace the variable in the following LDIF files:

- `/path/to/tomcat/webapps/openam/WEB-INF/template/ldif/opensj/opensj_uma_audit.ldif`
- `/path/to/tomcat/webapps/openam/WEB-INF/template/ldif/opensj/opensj_uma_resource_sets.ldif`
- `/path/to/tomcat/webapps/openam/WEB-INF/template/ldif/opensj/opensj_uma_resource_set_labels.ldif`
- `/path/to/tomcat/webapps/openam/WEB-INF/template/ldif/opensj/opensj_uma_pending_requests.ldif`

Note

If you have previously edited the files to replace the variables, replace those root entry values with the new UMA directory server root entries. For example, replace occurrences of `dc=cts,dc=example,dc=com` with `dc=uma-resources,dc=example,dc=com`.

2. Apply the required LDIF files to the store by using the `ldapmodify` command. The following applies the schema required for storing resources:

```
$ /path/to/opensj/bin/ldapmodify \  
--hostname 'uma-rs.example.com' \  
--port 1636 \  
--useSsl \  
--usePkcs12TrustStore /path/to/opensj/config/keystore \  
--trustStorePasswordFile /path/to/opensj/config/keystore.pin \  
--continueOnError \  
--bindDN uid=admin \  
--bindPassword str0ngAdm1nPa55word \  
--offline \  
/path/to/tomcat/webapps/openam/WEB-INF/template/ldif/opensj/opensj_uma_resource_sets.ldif  
# ADD operation successful for DN ou=resource_sets,dc=uma-resources,dc=example,dc=com
```

For more information on the required LDIF files, see LDIF Files for UMA Data Stores.

3. After applying all the required LDIF files, start the DS server.

```
$ /path/to/opensj/bin/start-ds
```

The DS instance is now ready to store UMA resources. To set up additional DS instances for other UMA-related data repeat the steps above, specifying the relevant schema LDIF files.

To configure AM to use the external DS instances to store UMA-related data, see "Configuring UMA Stores in AM".

Configuring UMA Stores in AM

UMA stores can only be configured at the server level so that all realms in the environment can access them. The procedures in this section show you how to configure the stores across all instances in your environment.

- "To Configure an UMA Resource Store"
- "To Configure an UMA Audit Store"
- "To Configure an UMA Pending Requests Store"
- "To Configure an UMA Resource Labels Store"

To Configure an UMA Resource Store

Resource Store properties are inherited from the defaults. For more information about inherited properties, see "Configuring Servers" in the *Reference*

1. In the AM console, go to Configure > Server Defaults > UMA > UMA Resource Store.
 - In the Store Mode field, choose External Token Store.
 - In the Root Suffix field, enter the base DN of the store. For example, `dc=uma-resources,dc=example,dc=com`.
 - Save your work.
2. Go to Configure > Server Defaults > UMA > External UMA Resource Store Configuration.
 - Enter the properties for the store. For information about the available settings, see "UMA Properties" in the *Reference*.
 - Save your work.

To Configure an UMA Audit Store

UMA Audit Store properties are inherited from the defaults. For more information about inherited properties, see "Configuring Servers" in the *Reference*

1. In the AM console, go to Configure > Server Defaults > UMA > UMA Audit Store.

- From the Store Mode drop-down list, choose External Token Store.
 - In the Root Suffix field, enter the base DN of the store. For example, `dc=uma-audit,dc=example,dc=com`.
 - Save your work.
2. Go to Configure > Server Defaults > UMA > External UMA Audit Store Configuration.
 - Enter the properties for the store. For information about the available settings, see "UMA Properties" in the *Reference*.
 - Save your work.

To Configure an UMA Pending Requests Store

UMA Pending Requests Store properties are inherited from the defaults. For more information about inherited properties, see "Configuring Servers" in the *Reference*

1. Go to Configure > Server Defaults > UMA > Pending Requests Store.
 - From the Store Mode drop-down list, choose External Token Store.
 - In the Root Suffix field, enter the base DN of the store. For example, `dc=uma-pending,dc=example,dc=com`.
 - Save your work.
2. Go to Configure > Server Defaults > UMA > External Pending Requests Store Configuration.
 - Enter the properties for the store. For information about the available settings, see "UMA Properties" in the *Reference*.
 - Save your work.

To Configure an UMA Resource Labels Store

UMA Resource Labels Store properties are inherited from the defaults. For more information about inherited properties, see "Configuring Servers" in the *Reference*

1. In the `am.console`, go to Configure > Server Defaults > UMA > UMA Resource Labels Store.
 - From the Store Mode drop-down list, choose External Token Store.
 - In the Root Suffix field, enter the base DN of the store. For example, `dc=uma-labels,dc=example,dc=com`.
 - Save your work.

2. Go to Configure > Server Defaults > UMA > External UMA Resource Labels Store Configuration.
 - Enter the properties for the store. For information about the available settings, see "UMA Properties" in the *Reference*.
 - Save your work.

Chapter 5

Configuring the UMA Actors

To allow UMA flows in your environment, you must configure the different UMA actors first. You may be familiar with some already, like the OAuth 2.0 provider and the OAuth 2.0 clients.

Even though the UMA provider is one of the actors, the role in AM is split between the OAuth2 Provider Service and the UMA provider Service, as you will see next.

Tip

To set up AM as an example UMA provider, resource server, and client, see "*The UMA Guide Example*" instead.

The OAuth 2.0/OpenID Connect Provider

As an extension of the OAuth 2.0 and OpenID Connect specifications, the AM authorization server is in charge of providing protection API access tokens, requesting party access tokens, and ID tokens to the UMA clients.

To configure the OAuth 2.0/OpenID Connect provider, see:

- "*Authorization Server Configuration*" in the *OAuth 2.0 Guide*
- "*OpenID Provider Configuration*" in the *OpenID Connect 1.0 Guide*

The UMA Provider

Configure the UMA provider by realm to expose UMA-related endpoints, and to configure UMA-related properties that are not exposed in the OAuth 2.0 provider.

The service's defaults are suitable for most situations and strike a good balance between security and ease of use.

Perform the steps in the following procedure to configure the service:

- In the AM console, go to Realms > Top Level Realm > Services, and add an UMA Provider service.

The UMA Provider page appears.

For information about the available attributes, see the "*UMA Provider*" in the *Reference*.

The Resource Server

You need a server to let the end user register their resources and share them. The resource server can be an AM instance, a third-party service, or Identity Gateway

Regardless of where the resource server is, it needs an UMA client registered in the AM instance configured as the UMA provider.

UMA Clients

Configure OAuth 2.0 clients to work as a resource server agent, a requesting party, and a resource owner.

Special Scopes

- The `uma_protection` Scope.

Clients requiring a protection API access token (PAT) must be configured with the `uma_protection` scope. This scope tells AM that the token is a PAT, and not a regular access token.

- The `openid` Scope.

Clients performing the UMA grant require the `openid` scope, since AM will provide the claims that UMA requires inside an ID token.

For more information about registering clients, see see "*Client Registration*" in the *OAuth 2.0 Guide*.

Chapter 6

Registering and Protecting Resources

Resource owners register their resources in the UMA provider, and protect them with authorization policies:

- Resource registration can occur at three different stages: at initial resource creation, when needed for policy creation, and at resource access attempt.

The process is the same regardless of when it is run.

- Policy creation can occur after resource creation or at resource access attempt. The process is the same regardless of when it is run, and the requesting party always needs to partake in the UMA grant flow to gain access to the resources.

Important

Only the resource owner can create a policy to protect a resource. Administrative users, such as `amAdmin`, cannot create policies on behalf of a resource owner.

See the following sections to learn how to register and protect resources with the AM UI and the REST APIs:

- "How to Manage UMA Resources"
- "How to Manage UMA Policies"

How to Manage UMA Resources

UMA resource servers register resources with the resource owner's chosen authorization server. Registered resources can then be protected, and are available for user-created policies.

AM supports optional *system* labels when registering resources to help resource owners with organization. For information on labeling resources, see "*Managing UMA Labels*".

AM provides the UMA `resource_set` REST endpoint, as documented in the *OAuth 2.0 Resource Registration* specification, to let UMA resource servers register and manage resources.

The endpoint requires a protection API token (PAT), which is an OAuth 2.0 access token with a scope of `uma_protection`. A resource server must acquire a PAT in order to use the resource set endpoint.

To Acquire a PAT

This example assumes that a confidential client called *UMA-Resource-Server* is registered in AM with, at least, the following configuration:

- **Client Secret:** `password`
- **Scopes:** `uma_protection`
- **Grant Types:** `Resource Owner Password Credentials`

This example uses the Resource Owner Password Credentials grant, but you can use any grant type except the Client Credentials one to obtain the PAT.

The example also assumes that an identity for the resource owner, `alice`, exists in AM.

Perform the following steps to acquire a PAT on behalf of the resource owner>:

- Send a POST request to the OAuth 2.0 `access_token` endpoint. The following example uses the Resource Owner Password Credentials grant:

```
$ curl \
--request POST \
--data 'grant_type=password' \
--data 'scope=uma_protection' \
--data 'username=alice' \
--data 'password=Ch4ng31t' \
--data 'client_id=UMA-Resource-Server' \
--data 'client_secret=password' \
https://openam.example.com:8443/openam/oauth2/realms/root/access_token
{
  "access_token": "057ad16f-7dba-4049-9f34-e609d230d43a",
  "refresh_token": "340f82a4-9aa9-471c-ac42-f0ca1809c82b",
  "scope": "uma_protection",
  "token_type": "Bearer",
  "expires_in": 4999
}
```

The value returned in `access_token` is the Protection API Token, or PAT Bearer token.

Note

To use the Resource Owner Password Credentials grant type, as described in RFC 6749, the default authentication chain in the relevant realm must allow authentication using only a username and password, for example by using a `DataStore` module. Attempting to use the Resource Owner Password Credentials grant type with a chain that requires any additional input returns an HTTP 500 Server Error message.

After acquiring a PAT, use the UMA `resource_set` REST endpoint for the following operations:

- "To Register an UMA Resource (REST)"

- "To List Registered UMA Resources (REST)"
- "To Read an UMA Resource (REST)"
- "To Update an UMA Resource (REST)"
- "To Delete an UMA Resource (REST)"

To Register an UMA Resource (REST)

- Create a POST request to the UMA `resource_set` endpoint, including the PAT bearer token in an Authorization header.

The following example uses a resource owner's PAT bearer token to register a photo album resource and a pair of system labels in a realm named `subrealm`:

```
$ curl \
--request POST \
--header "Content-Type: application/json" \
--header "Authorization: Bearer 515d6551-6512-5279-98b6-c0ef3f03a723" \
--header "Accept-API-Version: resource=1.0" \
--data \
'{
  "name" : "Photo Album",
  "icon_uri" : "http://photoz.example.com/icons/flower.png",
  "resource_scopes" : [
    "edit",
    "view",
    "http://photoz.example.com/dev/scopes/print"
  ],
  "labels" : [
    "3D",
    "VIP"
  ],
  "type" : "http://photoz.example.com/dev/rtypes/photoalbum"
}' \
https://openam.example.com:8443/openam/uma/realms/root/resource_set
{
  "_id": "126615ba-b7fd-4660-b281-bae81aa45f7c0",
  "user_access_policy_uri": "https://openam.example.com:8443/openam/XUI/?realm=#uma/share/126615ba-b7fd-4660-b281-bae81aa45f7c0"
}
```

To List Registered UMA Resources (REST)

- Create a GET request to the UMA `resource_set` endpoint, including the PAT bearer token in an Authorization header.

The following example uses a PAT bearer token to list the registered resources in a realm named `subrealm`:

```
$ curl \
--header "Authorization: Bearer 515d6551-6512-5279-98b6-c0ef3f03a723" \
--header "Accept-API-Version: resource=1.0" \
https://openam.example.com:8443/openam/uma/realms/root/resource_set
{
  "126615ba-b7fd-4660-b281-bae81aa45f7c0",
  "3a2fe6d5-67c8-4a5a-83fb-09734f1dd5b10",
  "8ed24623-fcb5-46b8-9a64-18ee1b9b7d5d0"
}
```

On success, an array of the registered resource IDs is returned. Use the ID to identify a resource in the following procedures:

- "To Read an UMA Resource (REST)"
- "To Update an UMA Resource (REST)"
- "To Delete an UMA Resource (REST)"

To Read an UMA Resource (REST)

- Create a GET request to the UMA `resource_set` endpoint, including the PAT bearer token in an Authorization header.

Note

You must provide the ID of the resource to read, specified at the end of the request, as follows: https://openam.example.com:8443/openam/uma/realms/root/resource_set/resource_set_ID.

The following example uses a PAT bearer token and a resource ID to read a specific resource in a realm named `subrealm`:


```
$ curl \
--header "Authorization: Bearer 515d6551-6512-5279-98b6-c0ef3f03a723" \
https://openam.example.com:8443/openam/uma/realms/root/resource_set/126615ba-b7fd-4660-b281-
bae81aa45f7c0
{
  "resource_scopes": [
    "read",
    "view",
    "http://photoz.example.com/dev/scopes/print"
  ],
  "name": "Photo Album",
  "id": "126615ba-b7fd-4660-b281-bae81aa45f7c0",
  "type": "https://www.example.com/rsets/photoalbum",
  "icon_uri": "http://www.example.com/icons/flower.png",
  "labels": [
    "VIP",
    "3D"
  ],
  "user_access_policy_uri":
    "https://openam.example.com:8443/openam/XUI/?realm=#uma/share/126615ba-b7fd-4660-b281-
bae81aa45f7c0"
}
```

On success, AM returns an HTTP 200 OK status code as well as a representation of the resource in the JSON body of the response.

If the resource ID does not exist, AM returns an HTTP 404 Not Found status code, as follows:

```
{
  "error": "not_found",
  "error_description":
    "Resource set corresponding to id: 43225628-4c5b-4206-b7cc-5164da81decd0 not found"
}
```

To Update an UMA Resource (REST)

This example updates the UMA policy for user **bob** with the **delete** scope. The registered user for UMA in this example is **alice** who has permission to update the policy using their own SSO token in the header.

1. Before you can read or update a resource, you have to acquire a PAT token on behalf of the resource owner.
2. Add the new policy action **delete** to the appropriate resource type.

```
$ curl \
'http://openam.example.com:8080/openam/json/resourcetypes/630a0d-4d86-4b7e-848-3bf7dda7d220' \
--request PUT \
--header 'Accept-API-Version: protocol=1.0,resource=1.0' \
--header 'Content-Type: application/json' \
--header "Authorization: Bearer 515d6551-6512-5279-98b6-c0ef3f03a723"
--data-raw '{"uuid":"63d10a0d-4d86-4b7e-8548-3bf70dda7d220", \
"description":"Dynamically created resource type for the UMA resource set. \
Used to find all Policy Engine Policies that make up an UMA Policy", \
"actions": \
  {"download":true, \
   "view":true, \
   "comment":true, \
   "delete":true}, \
}'
```

3. Send a PUT request to the UMA `resource_set` endpoint. Include the following:

- A PAT bearer token in a header named `Authorization`
- Any new or changed parameters in the existing values.
- The ID of the resource to update, specified at the end of the request. Example: `https://openam.example.com:8080/openam/uma/realms/root/resource_set/resource_set_ID`.

The following example uses a PAT bearer token and a resource ID add the `delete` scope to a realm resource:

```
$ curl \
--request PUT \
--header "Content-Type: application/json" \
--header "Authorization: Bearer 515d6551-6512-5279-98b6-c0ef3f03a723" \
--header "Accept-API-Version: resource=1.0" \
--header "If-Match: *" \
--data \
'{
  "policyId": "5c322250-a39b-455e-8413-33c3f8a876e00",
  "permissions":
  [
    {
      "subject": "bob",
      "scopes": [
        "view",
        "comment",
        "download",
        "delete"
      ]
    }
  ]
}' \
http://openam.example.com:8080/openam/json/realms/root/users/alice/uma/policies/5c322250-a39b-455e-8413-33c3f8a876e00
{
  "_id": "63d10a0d-4d86-4b7e-8548-3bf70dda7d220",
  "_rev": "-92235058",
```

```
"policyId": "63d10a0d-4d86-4b7e-8548-3bf70dda7d220",
"permissions": [
  {
    "subject": "bob",
    "scopes": [
      "download",
      "delete",
      "view",
      "comment"
    ]
  }
]
}
```

To Delete an UMA Resource (REST)

- Create a DELETE request to the UMA `resource_set` endpoint, including the PAT bearer token in a header named `Authorization`.

Provide the ID of the resource to delete, specified at the end of the request as follows: https://openam.example.com:8443/openam/uma/realms/root/resource_set/resource_set_ID

```
$ curl \
--request DELETE \
--header "Authorization: Bearer 515d6551-6512-5279-98b6-c0ef3f03a723" \
--header "Accept-API-Version: resource=1.0" \
https://openam.example.com:8443/openam/uma/realms/root/resource_set/126615ba-b7fd-4660-b281-
bae81aa45f7c0
{}
```

On success, AM returns an HTTP 204 No Content status code as well as an empty response body.

If the resource ID does not exist, AM returns an HTTP 404 Not Found status code, as follows:

```
{
  "error": "not_found",
  "error_description":
    "Resource set corresponding to id: 43225628-4c5b-4206-b7cc-5164da81decd0 not found"
}
```

How to Manage UMA Policies

UMA authorization servers must manage the resource owner's authorization policies, so that registered resources can be protected.

The `/json/users/{user}/uma/policies/` REST endpoint lets users create and manage authorization policies.

Important

UMA policies are designed to be user-managed, and not manipulated by AM administrators, using the standard **policy** endpoints. If AM administrators manipulate UMA policies directly, they risk breaking the integrity of the UMA security model.

Managing UMA policies with the REST endpoint requires that a resource is registered to the user in the URL. For information on registering resource sets, see `"/uma/resource_set"`.

- "To Create an UMA Policy (REST)"
- "To Read an UMA Policy (REST)"
- "To Update an UMA Policy (REST)"
- "To Delete an UMA Policy (REST)"
- "To Query UMA Policies (REST)"
- "To Manage Policies (UI)"

To Create an UMA Policy (REST)

To create a policy, the resource owner must be logged in to the authorization server, have an SSO token issued to them, and must also have the resource ID to be protected.

Note

Only the resource owner can create a policy to protect a resource. Administrator users, such as `amAdmin`, cannot create policies on behalf of a resource owner.

1. Log in as the resource owner to obtain an SSO token:

```
$ curl \
--request POST \
--header "Content-Type: application/json" \
--header "X-OpenAM-Username: alice" \
--header "X-OpenAM-Password: Ch4ng3!t" \
--header "Accept-API-Version: resource=2.0, protocol=1.0" \
'https://openam.example.com:8443/openam/json/realms/root/realms/alpha/authenticate'
{
  "tokenId": "AQIC5wM2LY4S...Q4MTE4NTA2*",
  "successUrl": "/openam/console",
  "realm": "/"
}
```

The value returned in the `tokenId` element is the SSO token of the resource owner, *Alice*. Use this value as the contents of the `iPlanetDirectoryPro` cookie when creating the policy below.

2. Send an HTTP PUT request to the UMA policy endpoint or to the user policy endpoint. Include the SSO token in a header based on the configured session cookie name (default: `iPlanetDirectoryPro`), and the resource ID as the value of `policyId` in the body, and also in the URI.

Example 1: UMA Policy Endpoint

This example uses the policy owner's SSO token (`iPlanetDirectoryPro` cookie). The command below creates a policy to share a resource belonging to user `alice` with user `bob`:

```
$ curl \
--request PUT \
--header "Accept-API-Version: resource=1.0" \
--header "Cache-Control: no-cache" \
--header "Content-Type: application/json" \
--header "iPlanetDirectoryPro: AQIC5wM2LY4S...Q4MTE4NTA2*" \
--header "If-None-Match: *" \
--data '{
  "policyId": "0d7790de-9066-4bb6-8e81-25b6f9d0b8853",
  "permissions":
  [
    {
      "subject": "bob",
      "scopes": [
        "view",
        "comment"
      ]
    }
  ]
}' \
https://openam.example.com:8443/openam/json/realms/root/users/alice/uma/
policies/0d7790de-9066-4bb6-8e81-25b6f9d0b8853
{
  "_id": "0d7790de-9066-4bb6-8e81-25b6f9d0b8853",
  "_rev": "-1985914901"
}
```

- 1 Specify the SSO token of the resource owner. Administrative users such as `amAdmin` cannot create UMA resource policies on behalf of a resource owner.
- 2 Specify the ID of the registered resource that this policy will protect. The same resource ID must also be included in the URI of the REST call. See "To Register an UMA Resource (REST)".
- 3 Note that the returned `_id` value of the new policy set is identical to the ID of the registered resource.

On success, AM returns an HTTP 201 Created status code with the ID of the created policy.

If the permissions are not correct, AM returns an HTTP 400 Bad Request status. For example:

```
{
  "code": 400,
  "reason": "Bad Request",
  "message": "Invalid UMA policy permission. Missing required attribute, 'subject'."
}
```

Example 2: User Policy Endpoint

Using the user policy endpoint gives more control over how you define the policy. For example, you have more flexibility in using resource owner names or policy names that include special characters.

You can use this endpoint if you have implemented your own UI or other interface for creating UMA policies. You'll have to manually manage any policies you create in this way.

```
$ curl \
--request PUT \
--header "-API-Version: protocol=1.0,resource=1.0" \
--header "Content-Type: application/json" \
--header "iPlanetDirectoryPro: AQIC5wM2LY4S...Q4MTE4NTA2*" \
--header "If-None-Match: *" \
--data '{
  "resources":[
    "uma://02f08d5e-396e-4d90-b151-0a09ff836c021"
  ],
  "applicationName":"umaresourceserver",
  "actionValues":{
    "read":true
  },
  "subject":{
    "type":"AuthenticatedUsers"
  }
}' \
https://openam.example.com:8443/openam/json/users/alice/policies/test123
```

To Read an UMA Policy (REST)

To read a policy, the resource owner or an administrator user must be logged in to the authorization server and have an SSO token issued to them. The policy ID to read must also be known.

Tip

The ID used for a policy is always identical to the ID of the resource it protects.

1. Log in as the resource owner to obtain an SSO token:

```
$ curl \
--request POST \
--header "Content-Type: application/json" \
--header "X-OpenAM-Username: alice" \
--header "X-OpenAM-Password: Ch4ng31t" \
--header "Accept-API-Version: resource=2.0, protocol=1.0" \
'https://openam.example.com:8443/openam/json/realms/root/realms/alpha/authenticate'
{
  "tokenId":"AQIC5wM2LY4S...Q4MTE4NTA2*",
  "successUrl":"/openam/console",
  "realm":"/"
}
```

The value returned in the `tokenId` element is the SSO token of the resource owner, *Alice*. Use this value as the contents of the `iPlanetDirectoryPro` cookie in the next step.

2. Create a GET request to the UMA `policies` endpoint, including the SSO token in a header based on the configured session cookie name (default: `iPlanetDirectoryPro`), and the resource ID as part of the URL.

Note

The SSO token must have been issued to the user specified in the URL, or to an administrative user such as `amAdmin`. In this example, the user is `demo`.

The following example uses an SSO token to read a specific policy with ID `43225628-4c5b-4206-b7cc-5164da81decd0` belonging to user `demo`:

```
$ curl \
--header "iPlanetDirectoryPro: AQIC5wM2LY4S...Q4MTE4NTA2*" \
--header "Accept-API-Version: resource=1.0" \
https://openam.example.com:8443/openam/json/realms/root/users/demo\
/uma/policies/0d7790de-9066-4bb6-8e81-25b6f9d0b8853
{
  "_id": "0d7790de-9066-4bb6-8e81-25b6f9d0b8853",
  "_rev": "1444644662",
  "policyId": "0d7790de-9066-4bb6-8e81-25b6f9d0b8853",
  "name": "Photo Album",
  "permissions": [
    {
      "subject": "bob",
      "scopes": [
        "view",
        "comment"
      ]
    }
  ]
}
```

On success, AM returns an HTTP 200 OK status code with a JSON body representing the policy.

If the policy ID does not exist, an HTTP 404 Not Found status code is returned, as follows:

```
{
  "code": 404,
  "reason": "Not Found",
  "message": "UMA Policy not found, 43225628-4c5b-4206-b7cc-5164da81decd0"
}
```

To Update an UMA Policy (REST)

To update a policy, the resource owner or an administrator user must be logged in to the authorization server and have an SSO token issued to them. The policy ID to read must also be known.

Tip

The ID used for a policy is always identical to the ID of the resource it protects.

1. Log in as the resource owner to obtain an SSO token:

```
$ curl \
--request POST \
--header "Content-Type: application/json" \
--header "X-OpenAM-Username: alice" \
--header "X-OpenAM-Password: Ch4ng31t" \
--header "Accept-API-Version: resource=2.0, protocol=1.0" \
'https://openam.example.com:8443/openam/json/realms/root/realms/alpha/authenticate'
{
  "tokenId": "AQIC5wM2LY4S...Q4MTE4NTA2*",
  "successUrl": "/openam/console",
  "realm": "/"
}
```

The value returned in the `tokenId` element is the SSO token of the resource owner, *Alice*. Use this value as the contents of the `iPlanetDirectoryPro` cookie in the next step.

2. Create a PUT request to the UMA `policies` endpoint, including the SSO token in a header based on the configured session cookie name (default: `iPlanetDirectoryPro`), and the resource ID as both the value of `policyId` in the body and also as part of the URL.

Note

The SSO token must have been issued to the user specified in the URL. In this example, the user is `demo`.

The following example uses an SSO token to update a policy with ID `0d7790de-9066-4bb6-8e81-25b6f9d0b8853` belonging to user *demo* with an additional subject, *chris*:

```
$ curl \
--request PUT \
--header "iPlanetDirectoryPro: AQIC5wM2LY4S...Q4MTE4NTA2*" \
--header "Content-Type: application/json" \
--header "If-Match: *" \
--header "Accept-API-Version: resource=1.0" \
--data \
'{
  "policyId": "0d7790de-9066-4bb6-8e81-25b6f9d0b8853",
  "permissions":
  [
    {
      "subject": "bob",
      "scopes": [
        "view",
        "comment"
      ]
    },
    {
      "subject": "chris",
      "scopes": [
```



```

        "comment"
      ]
    }
  ]
}' \
https://openam.example.com:8443/openam/json/realms/root/users/demo\
/uma/policies/0d7790de-9066-4bb6-8e81-25b6f9d0b8853
{
  "_id": "0d7790de-9066-4bb6-8e81-25b6f9d0b8853",
  "_rev": "-1844449592",
  "policyId": "0d7790de-9066-4bb6-8e81-25b6f9d0b8853",
  "permissions": [
    {
      "subject": "bob",
      "scopes": [
        "view",
        "comment"
      ]
    },
    {
      "subject": "chris",
      "scopes": [
        "view"
      ]
    }
  ]
}

```

On success, AM returns an HTTP 200 OK status code with a JSON representation of the policy in the body as the response.

If the policy ID does not exist, an HTTP 404 Not Found status code is returned, as follows:

```

{
  "code": 404,
  "reason": "Not Found",
  "message": "UMA Policy not found, 43225628-4c5b-4206-b7cc-5164da81decd0"
}

```

If the permissions are not correct, AM returns an HTTP 400 Bad Request status code. For example:

```

{
  "code": 400,
  "reason": "Bad Request",
  "message": "Invalid UMA policy permission. Missing required attribute, 'subject'."
}

```

If the policy ID in the URL does not match the policy ID used in the sent JSON body, AM returns an HTTP 400 Bad Request status code. For example:

```

{
  "code": 400,
  "reason": "Bad Request",
  "message": "Policy ID does not match policy ID in the body."
}

```

To Delete an UMA Policy (REST)

To delete a policy, the resource owner or an administrator user must be logged in to the authorization server and have an SSO token issued to them. The policy ID to read must also be known.

Tip

The ID used for a policy is always identical to the ID of the resource it protects.

1. Log in as the resource owner to obtain an SSO token:

```
$ curl \
--request POST \
--header "Content-Type: application/json" \
--header "X-OpenAM-Username: alice" \
--header "X-OpenAM-Password: Ch4ng31t" \
--header "Accept-API-Version: resource=2.0, protocol=1.0" \
'https://openam.example.com:8443/openam/json/realms/root/realms/alpha/authenticate'
{
  "tokenId": "AQIC5wM2LY4S...Q4MTE4NTA2*",
  "successUrl": "/openam/console",
  "realm": "/"
}
```

The value returned in the `tokenId` element is the SSO token of the resource owner, *Alice*. Use this value as the contents of the `iPlanetDirectoryPro` cookie in the next step.

2. Create a DELETE request to the UMA `policies` endpoint, including the SSO token in a header based on the configured session cookie name (default: `iPlanetDirectoryPro`), and the resource ID as part of the URL.

Note

The SSO token must have been issued to the user specified in the URL. In this example, the user is `demo`.

The following example uses an SSO token to delete a policy with ID `0d7790de-9066-4bb6-8e81-25b6f9d0b8853` belonging to user `demo`:

```
$ curl \
--request DELETE \
--header "iPlanetDirectoryPro: AQIC5wM2LY4S...Q4MTE4NTA2*" \
--header "Accept-API-Version: resource=1.0" \
https://openam.example.com:8443/openam/json/realms/root/realms/root/users/demo\
/json/policies/0d7790de-9066-4bb6-8e81-25b6f9d0b8853
{}
```

On success, AM returns an HTTP 200 OK status code with an empty JSON body as the response.

If the policy ID does not exist, an HTTP 404 Not Found status code is returned, as follows:

```
{
  "code": 404,
  "reason": "Not Found",
  "message": "UMA Policy not found, 43225628-4c5b-4206-b7cc-5164da81decd0"
}
```

To Query UMA Policies (REST)

To query policies, the resource owner or an administrator user must be logged in to the authorization server and have an SSO token issued to them.

1. Log in as the resource owner to obtain an SSO token:

```
$ curl \
--request POST \
--header "Content-Type: application/json" \
--header "X-OpenAM-Username: alice" \
--header "X-OpenAM-Password: Ch4ng31t" \
--header "Accept-API-Version: resource=2.0, protocol=1.0" \
'https://openam.example.com:8443/openam/json/realms/root/realms/alpha/authenticate'
{
  "tokenId": "AQIC5wM2LY4S...Q4MTE4NTA2*",
  "successUrl": "/openam/console",
  "realm": "/"
}
```

The value returned in the `tokenId` element is the SSO token of the resource owner, *Alice*. Use this value as the contents of the `iPlanetDirectoryPro` cookie in the next step.

2. Create a GET request to the UMA `policies` endpoint, including the SSO token in a header based on the configured session cookie name (default: `iPlanetDirectoryPro`).

Note

The SSO token must have been issued to the user specified in the URL, or to an administrative user such as `amAdmin`.

In this example, the user is `demo`.

Use the following query string parameters to affect the returned results:

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

Sort the results returned, where *field* represents a field in the JSON policy objects returned.

For UMA policies, only the `policyId` and `name` fields can be sorted.

Optionally, use the `+` prefix to sort in ascending order (the default), or `-` to sort in descending order.

`_pageSize=integer`

Limit the number of results returned.

`_pagedResultsOffset=integer`

Start the returned results from the specified index.

`_queryFilter`

The `_queryFilter` parameter can take `true` to match every policy, `false` to match no policies, or a filter of the following form to match field values: `field operator value` where `field` represents the field name, `operator` is the operator code, `value` is the value to match, and the entire filter is URL-encoded. Only the equals (`eq`) operator is supported by the `/uma/policies` endpoint.

The `field` value can take the following values:

- `resourceServer` - the resource server that created the resource.
- `permissions/subject` - the list of subjects that are assigned scopes in the policy.

Filters can be composed of multiple expressions by using a boolean operator `AND`, and by using parentheses, `(expression)`, to group expressions.

Note

You must URL-encode the filter expression in `_queryFilter=filter`. So, for example, the following filter:

```
resourceServer eq "UMA-Resource-Server" AND permissions/subject eq "bob"
```

When URL-encoded becomes:

```
resourceServer+eq+%22UMA-Resource-Server%22+AND+permissions%2Fsubject+eq+%22bob%22
```

The following example uses an SSO token to query the policies belonging to user `demo`, which have a subject `bob` in the permissions:

```
$ curl \
--header "iPlanetDirectoryPro: AQIC5wM2LY4S...Q4MTE4NTA2*" \
--header "Accept-API-Version: resource=1.0" \
--get \
--data-urlencode '_sortKeys=policyId,name' \
--data-urlencode '_pageSize=1' \
--data-urlencode '_pagedResultsOffset=0' \
--data-urlencode \
'_queryFilter=permissions/subject eq "bob"' \
https://openam.example.com:8443/openam/json/realms/root/users/demo/uma/policies
{
  "result": [
    {
      "_id": "0d7790de-9066-4bb6-8e81-25b6f9d0b8853",
```

```
{
  "policyId": "0d7790de-9066-4bb6-8e81-25b6f9d0b8853",
  "name": "Photo Album",
  "permissions": [
    {
      "subject": "bob",
      "scopes": [
        "view",
        "comment"
      ]
    },
    {
      "subject": "chris",
      "scopes": [
        "view"
      ]
    }
  ]
},
"resultCount": 1,
"pagedResultsCookie": null,
"remainingPagedResults": 0
}
```

On success, AM returns an HTTP 200 OK status code with a JSON body representing the policies that match the query.

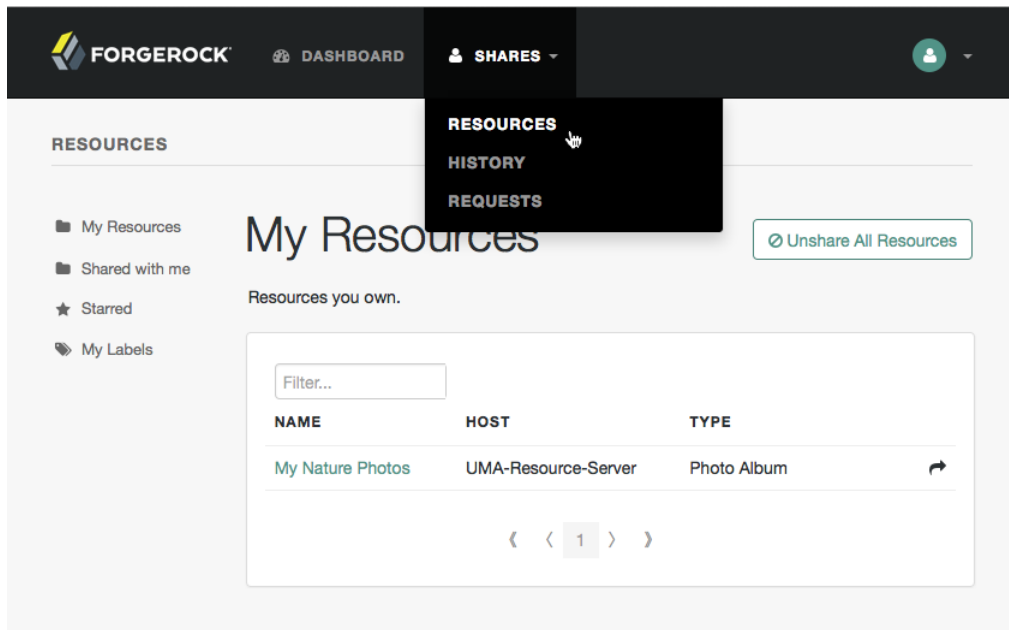
If the query is not formatted correctly, for example, an incorrect field is used in the `_queryFilter`, AM returns an HTTP 500 Server Error as follows:

```
{
  "code": 500,
  "reason": "Internal Server Error",
  "message": "'/badField' not queryable"
}
```

To Manage Policies (UI)

1. Log in to AM. Your user profile page appears.
2. On the Shares menu, click Resources. A list of the resources you own appears.

The Resources Page when Logged In

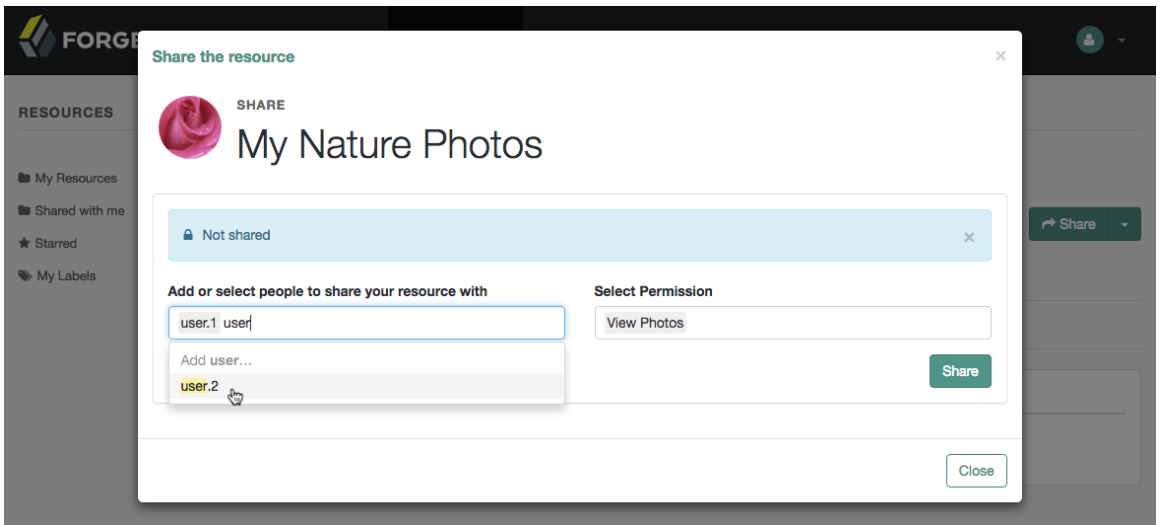


3. To share a resource, click the name of the resource to open the resource details page, and then click the Share button.

On the Share the resource form:

- a. Enter the username of the user with whom to share the resource.
- b. From the Select Permission drop-down list, choose the permissions to assign to the user for the selected resource.
- c. Click Share.

Sharing an UMA Resource



- d. Repeat these steps to share the resources with additional users.

Note

AM creates a policy set containing a policy representing the resources and identities specified by the resource owner sharing their resources.

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

4. When finished, click Close.

Chapter 7

Managing UMA Labels

Apply labels to resources to help organize and locate them more easily. Resources can have multiple labels applied to them, and labels can apply to multiple resources.

Resources support three types of labels:

User Labels

- Managed by the resource owner after the resource has been registered to them.
- Can be created and deleted. Deleting a label does not delete the resources to which it was applied.
- Support nested hierarchies. Separate levels of the hierarchy with forward slashes (/) when creating a label. For example, `Top Level/Second Level/My Label`.
- Are only visible to the user who created them.

You can manage user labels by using the AM console, or by using a REST interface. For more information, see "How to Manage UMA User and Favorite Labels" and "To Label Resources Using (UI)".

System Labels

- Created by the resource server when registering a resource.
- Cannot be deleted.
- Do not support a hierarchy of levels.
- Are only visible to the owner of the resource.

Note

Each resource is automatically assigned a system label containing the name of the resource server that registered it, as well as a system label that lets users add the resource to a list of favorites.

For information on creating system labels, see "To Register an UMA Resource (REST)".

Favorite Labels

Each user can assign the built-in *star* label to a resource to mark it as a favorite.

You can manage favorite labels by using the AM console, or by using a REST interface. For more information, see "How to Manage UMA User and Favorite Labels" and "To Label Resources as Favorites with the UI".

How to Manage UMA User and Favorite Labels

AM provides the `/json/users/username/oauth2/resources/labels` REST endpoint to let users manage user labels. It also provides built-in user pages in the UI.

When using the REST endpoint, specify the `username` in the URL, and provide the SSO token of that user in the `iPlanetDirectoryPro` header.

- "To Create User Labels (REST)"
- "To Query User Labels (REST)"
- "To Delete User Labels (REST)"
- "To Label Resources Using (UI)"
- "To Label Resources as Favorites with the UI"

To Create User Labels (REST)

1. Log in as the resource owner to obtain an SSO token:

```
$ curl \
--request POST \
--header "Content-Type: application/json" \
--header "X-OpenAM-Username: alice" \
--header "X-OpenAM-Password: Ch4ng31t" \
--header "Accept-API-Version: resource=2.0, protocol=1.0" \
'https://openam.example.com:8443/openam/json/realms/root/realms/alpha/authenticate'
{
  "tokenId": "AQIC5wM2LY4S...Q4MTE4NTA2*",
  "successUrl": "/openam/console",
  "realm": "/"
}
```

The value returned in the `tokenId` element is the SSO token of the resource owner, *Alice*. Use this value as the contents of the `iPlanetDirectoryPro` cookie in the next step.

2. To create a new user label, create a POST request with the name of the new user label and the type, `USER`, as shown below:

```
$ curl \
--request POST \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--header "iPlanetDirectoryPro: AQIC5wM2LY4S...Q4MTE4NTA2*" \
--data \
'{
  "name" : "New Resource Label",
  "type" : "USER"
}' \
https://openam.example.com:8443/openam/json/realms/root/users/demo/oauth2/resources/Labels
{
  "_id": "db2161c0-167e-4195-a832-92b2f578c96e3",
  "_rev": "-785293115",
  "name": "New Resource Set Label",
  "type": "USER"
}
```

Tip

If you set the `type` object as `STAR`, you create a favorite label.

On success, AM returns an HTTP 201 Created status code as well as the unique identifier of the new user label in the `_id` property in the JSON-formatted body. Note that the user label is not yet associated with a resource. To apply the new label to a resource, see "To Update an UMA Resource (REST)".

To Query User Labels (REST)

1. Log in as the resource owner to obtain an SSO token:

```
$ curl \
--request POST \
--header "Content-Type: application/json" \
--header "X-OpenAM-Username: alice" \
--header "X-OpenAM-Password: Ch4ng31t" \
--header "Accept-API-Version: resource=2.0, protocol=1.0" \
'https://openam.example.com:8443/openam/json/realms/root/realms/alpha/authenticate'
{
  "tokenId": "AQIC5wM2LY4S...Q4MTE4NTA2*",
  "successUrl": "/openam/console",
  "realm": "/"
}
```

The value returned in the `tokenId` element is the SSO token of the resource owner, *Alice*. Use this value as the contents of the `iPlanetDirectoryPro` cookie in the next step.

2. To query the labels belonging to a user, create a GET request including `_queryFilter=true` in the query string, as shown below:

```
$ curl \
--header "iPlanetDirectoryPro: AQIC5wM2LY4S...Q4MTE4NTA2*" \
--header "Accept-API-Version: resource=1.0" \
https://openam.example.com:8443/json/realms/root/users/demo/oauth2/resources/Labels?_queryFilter=true
```

```
{
  "result": [
    {
      "_id": "46a3392f-1d2f-4643-953f-d51ecd141d44",
      "name": "2015/October/Bristol",
      "type": "USER"
    },
    {
      "_id": "60b785c2-9510-40f5-85e3-9837ac272f1b1",
      "name": "Top Level/Second Level/My Label",
      "type": "USER"
    },
    {
      "_id": "ed5fad66-c873-4b80-93bb-92656eb06deb0",
      "name": "starred",
      "type": "STAR"
    },
    {
      "_id": "db2161c0-167e-4195-a832-92b2f578c96e3",
      "name": "New Resource Set Label",
      "type": "USER"
    }
  ],
  "resultCount": 4,
  "pagedResultsCookie": null,
  "totalPagedResultsPolicy": "NONE",
  "totalPagedResults": -1,
  "remainingPagedResults": -1
}
```

To Delete User Labels (REST)

1. Log in as the resource owner to obtain an SSO token:

```
$ curl \
--request POST \
--header "Content-Type: application/json" \
--header "X-OpenAM-Username: alice" \
--header "X-OpenAM-Password: Ch4ng3!t" \
--header "Accept-API-Version: resource=2.0, protocol=1.0" \
'https://openam.example.com:8443/openam/json/realms/root/realms/alpha/authenticate'
{
  "tokenId": "AQIC5wM2LY4S...Q4MTE4NTA2*",
  "successUrl": "/openam/console",
  "realm": "/"
}
```

The value returned in the `tokenId` element is the SSO token of the resource owner, *Alice*. Use this value as the contents of the `iPlanetDirectoryPro` cookie in the next step.

2. To delete a user label belonging to a user, create a DELETE request including the ID of the user label to delete in the URL, as shown below:

```
$ curl \
--request DELETE \
--header "iPlanetDirectoryPro: AQIC5wM2LY4S...Q4MTE4NTA2*" \
--header "Accept-API-Version: resource=1.0" \
https://openam.example.com:8443/json/realms/root/users/demo/oauth2/resources/
labels/46a3392f-1d2f-4643-953f-d51ecdf141d44
{
  "_id": "46a3392f-1d2f-4643-953f-d51ecdf141d44",
  "name": "2015/October/Bristol",
  "type": "USER"
}
```

On success, AM returns an HTTP 200 OK status code, as well as a JSON representation of the user label that was removed.

To Label Resources Using (UI)

To apply labels to a resource using the UI:

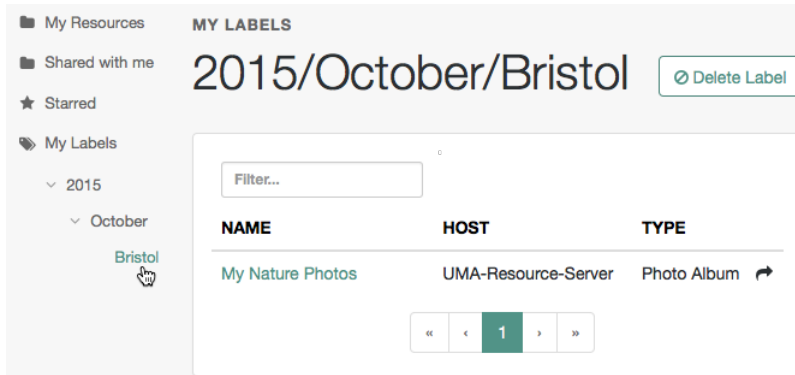
1. Log in to AM as a user. The profile page is displayed.
2. Go to Shares > Resources > My Resources, and then click the name of the resource to add labels to.
3. On the resource details page, click Edit Labels.

In the edit box that appears, you can:

- Enter the label you want to add to the resource, and then press **Enter**.

If you enter a label containing forward slash (/) characters, a hierarchy of each component of the label is created. The resource only appears in the last component of the hierarchy.

For example, the screenshot below shows the result of the label: **2015/October/Bristol**:



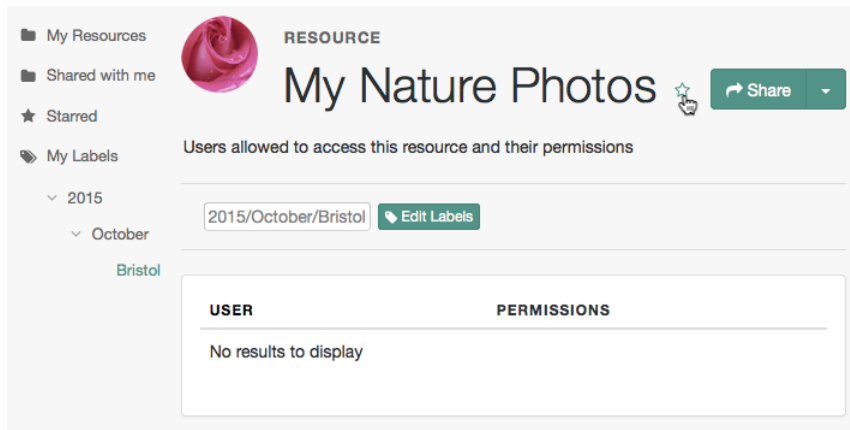
- Click an existing label, and then press **Delete** or **Backspace** to delete the label from the resource.

4. When you have finished editing labels you can:
 - Click the checkmark button to save any changes made.
 - Click the X button to cancel any changes made.

To Label Resources as Favorites with the UI

Mark resources as favorites to have them appear on the Starred page.

1. Log in to AM as the resource owner user. The profile page is displayed.
2. Go to Shares > Resources > My Resources, and then click the name of the resource to add to the list of favorites.
3. On the resource details page, click the star icon, as shown below:



To view the list of favorite resources, click Starred.

Chapter 8

The UMA Grant Flow

Endpoints

- `/oauth2/authorize` in the *OAuth 2.0 Guide*
- `/oauth2/access_token` in the *OAuth 2.0 Guide*
- `/uma/permission_request`

The UMA grant flow issues an RPT (*Requesting party token*) to the requesting party to allow access to a resource.

The implementation in AM covers the following scenarios:

- The requesting party wants to perform an action over a resource (for example, downloading it), and the resource owner has already granted them that permission.

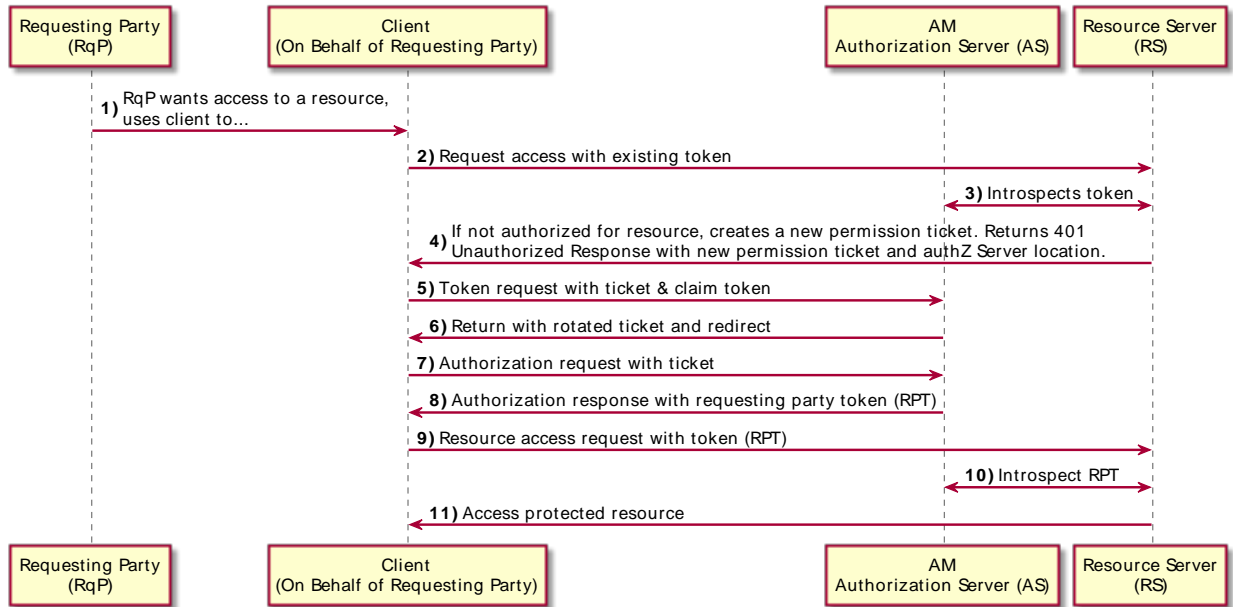
During the UMA grant flow, AM issues the requesting party an RPT (*Requesting party token*) that they can present to the resource server to access the resource.

- The requesting party wants to perform an action over a resource (for example, downloading it), and the resource owner has not granted them that permission.

During the UMA grant flow, AM denies access to the resource, and sends the resource owner a request for permission on behalf of the requesting party.

If the resource owner grants their permission (which happens asynchronously), the requesting party requests a new permission ticket and applies for another RPT.

The following diagram shows the first scenario:



+ Flow Explained

- A *requesting party*, using a client application, requests access to an UMA-protected resource (labeled 1 and 2 in the diagram above).
- The *resource server* checks the existing token (3) and determines that the *requesting party* does not have the correct privileges to access the resource. The *resource server* returns a permission ticket (4) to the client.
- The client uses the permission ticket and a claim token to send an RPT from AM (5 and 6).
- AM makes a policy decision using the requested scopes, the scopes permitted in the registered resource, and the user-created policy, and if successful returns an RPT (7 and 8).
- The client presents the RPT to the *resource server* (9), which must verify the token is valid using the AM introspection endpoint (10). If the RPT is confirmed to be valid and non-expired (10), the *resource server* can return the protected resource to the client for access by the requesting party (11).

Perform the steps in the following procedures to issue an RPT to a requesting party, and have it rejected as in the second scenario. The resource owner will then grant permission over the resource, and the requesting party will retry the flow again:

To Acquire a PAT

This example assumes that a confidential client called *UMA-Resource-Server* is registered in AM with, at least, the following configuration:

- **Client Secret:** `password`
- **Scopes:** `uma_protection`
- **Grant Types:** `Resource Owner Password Credentials`

This example uses the Resource Owner Password Credentials grant, but you can use any grant type except the Client Credentials one to obtain the PAT.

The example also assumes that an identity for the resource owner, `alice`, exists in AM.

Perform the following steps to acquire a PAT on behalf of the resource owner>:

- Create a POST request to the OAuth 2.0 `access_token` endpoint. The following example uses the Resource Owner Password Credentials grant:

```
$ curl \
--request POST \
--data 'grant_type=password' \
--data 'scope=uma_protection' \
--data 'username=alice' \
--data 'password=Ch4ng31t' \
--data 'client_id=UMA-Resource-Server' \
--data 'client_secret=password' \
https://openam.example.com:8443/openam/oauth2/realms/root/access_token
{
  "access_token": "057ad16f-7dba-4049-9f34-e609d230d43a",
  "refresh_token": "340f82a4-9aa9-471c-ac42-f0ca1809c82b",
  "scope": "uma_protection",
  "token_type": "Bearer",
  "expires_in": 4999
}
```

The value returned in `access_token` is the Protection API Token, or PAT Bearer token.

Note

To use the Resource Owner Password Credentials grant type, as described in RFC 6749, the default authentication chain in the relevant realm must allow authentication using only a username and password,

for example by using a `DataStore` module. Attempting to use the Resource Owner Password Credentials grant type with a chain that requires any additional input returns an HTTP `500 Server Error` message.

To Create a Permission Ticket

When the resource server receives a request for access to a resource, it contacts the authorization server to acquire a permission ticket. The permission ticket associates a request for a particular resource with the corresponding scopes. The PAT bearer token of the resource owner is used to map the request to the correct identity.

The permission ticket and the claim token are used to obtain an RPT (*Requesting party token*). A new permission ticket must be used for each attempt to acquire an RPT.

This example assumes that a confidential client called *UmaClient* is registered in AM with, at least, the following configuration:

- **Client Secret:** `password`
- **Scopes:** `openid, download`
- **Grant Types:** `Resource Owner Password Credentials, UMA`

This example uses the Resource Owner Password Credentials grant, but you can use any grant type, except the Client Credentials one.

- **Token Endpoint Authentication Method:** `client secret post`

Confidential OpenID Connect clients can use several methods to authenticate, but this example uses `client secret post` for clarity. For more information, see "*OpenID Connect Client Authentication*" in the *OpenID Connect 1.0 Guide*.

The example also assumes that an identity for the resource owner, `bob`, exists in AM.

- Send a POST request to the UMA `permission_request` endpoint:

```
$ curl -X POST \  
--header 'authorization: Bearer 057ad16f-7dba-4049-9f34-e609d230d43a' \  
--header 'cache-control: no-cache' \  
--header 'content-type: application/json' \  
--data '{  
  {  
    "resource_id" : "ef4d750e-3831-483b-b395-c6f059b5e15d0",  
    "resource_scopes" : ["download"]  
  }  
}' \  
https://openam.example.com:8443/openam/uma/realms/root/permission_request  
{  
  "ticket": "eyJ0eXAiOiJ...XPeJi3E"  
}
```

- 1 Use the PAT Bearer Token previously acquired on behalf of the resource owner.
- 2 Specify the ID of the registered resource for which this permission ticket will maintain permission state information. See "To Register an UMA Resource (REST)".
- 3 The value returned in the `ticket` property is the permission ticket, which is used to obtain an RPT. See "To Obtain an RPT".

Note

The default lifetime for an UMA permission ticket is 120 seconds. Attempting to obtain a requesting party token after the permission ticket has expired will fail with an error message as follows:

```
{
  "error_description": "The provided access grant is invalid, expired, or revoked.", "error":
  "invalid_grant"
}
```

You can alter the default lifetime of a permission ticket by going to Realms > *Realm Name* > Services > UMA Provider, and editing the Permission Ticket Lifetime (seconds) property.

To Gather Claims

The authorization server must gather claims from the requesting party to create a claim token.

- Send a POST request to the OAuth 2.0 `access_token` endpoint. The value returned in the `id_token` property is the claim token required to obtain an RPT, along with the permission ticket acquired earlier:

```
$ curl -X POST \
--data 'client_id=UmaClient' \
--data 'client_secret=password' \
--data 'grant_type=password' \
--data 'scope=openid' \
--data 'username=bob' \
--data 'password=Ch4ng31t' \
https://openam.example.com:8443/openam/oauth2/realms/root/access_token
{
  "access_token": "f09f55e5-5e9c-48fe-aeaa-d377de88e8e6",
  "refresh_token": "ee2d35f6-5819-4734-8b3e-9af77a545563",
  "scope": "openid",
  "id_token": "eyJ0eXA...FBznEB5A",
  "token_type": "Bearer",
  "expires_in": 4999
}
```

- 1 The value returned in the `id_token` property is the claim token, which is used to obtain an RPT. See "To Obtain an RPT".

Note

To use the Resource Owner Password Credentials grant type, as described in RFC 6749, the default authentication chain in the relevant realm must allow authentication using only a username and password,

for example by using a [DataStore](#) module. Attempting to use the Resource Owner Password Credentials grant type with a chain that requires any additional input returns an HTTP 500 Server Error message.

To Obtain an RPT

The requesting party makes a request using the permission ticket and the claim token, in exchange for a RPT.

1. Send a POST request to the OAuth 2.0 `access_token` endpoint. Make sure to include the permission ticket, `ticket`, and the `claim_token`. The following example results in an error description, indicating that "The client is not authorised to access the requested resource set." The authorization server sends a request to the resource owner to allow or deny access to the requesting party.

```
$ curl -X POST \
--data 'client_id=UmaClient' \
--data 'client_secret=password' \
--data 'grant_type=urn:ietf:params:oauth:grant-type:uma-ticket' \
--data 'ticket=eyJ0eXAiOiJ...XPeJi3E' \ ❶
--data 'claim_token=eyJ0eXA...FBznEB5A' \
--data 'claim_token_format=http://openid.net/specs/openid-connect-core-1_0.html#IDToken' \
https://openam.example.com:8443/openam/oauth2/realms/root/access_token
{
  "ticket": "eyJ0eXAiOiJ...XPeJi3E",
  "error_description": "The client is not authorised to access the requested resource set. A request
has been submitted to the resource owner requesting access to the resource",
  "error": "request_submitted"
}
```

- ❶ Specify the permission ticket acquired earlier. See "To Create a Permission Ticket".
- ❷ Specify the claim token acquired earlier. See "To Gather Claims".

Note

The default lifetime for an UMA permission ticket is 120 seconds. Attempting to obtain a requesting party token after the permission ticket has expired will fail with an error message as follows:

```
{
  "error_description": "The provided access grant is invalid, expired, or revoked.", "error":
  "invalid_grant"
}
```

If the ticket has expired, obtain another by repeating the steps in "To Create a Permission Ticket".

You can alter the default lifetime of a permission ticket by navigating to Realms > *Realm Name* > Services > UMA Provider, and editing the Permission Ticket Lifetime (seconds) property.

2. The resource owner, Alice, logs into AM to view the access request. Alice clicks Shares > Requests, and clicks Allow to grant download access to Bob, the requesting party.

Consent Screen Presented to the Resource Owner

SHARES Requests

Pending requests for access to your resources.

USER	RESOURCE	WHEN	PERMISSIONS
bob	my resource 106	a few seconds ago	download

« < 1 > »

- Because each permission token can only be used once, request a new permission token by performing the steps in "To Create a Permission Ticket".
- Resubmit the previous POST request for the RPT, with the new permission ticket obtained in the previous step and the original claim token:

```
curl -X POST \
--data 'client_id=UmaClient' \
--data 'client_secret=password' \
--data 'grant_type=urn:ietf:params:oauth:grant-type:uma-ticket' \
--data 'ticket=eyJ0efBiOiJ...XPeJc2A' \ ❶
--data 'claim_token=eyJ0eXA...FBznEB5A' \
--data 'claim_token_format=http://openid.net/specs/openid-connect-core-1_0.html#IDToken' \
https://openam.example.com:8443/openam/oauth2/realms/root/access_token
{
  "access_token": "Aw4a92ZoKsjadWKw2d4Rmcjv7DM",
  "token_type": "Bearer",
  "expires_in": 3599
}
```

- ❶ Specify a refreshed permission ticket acquired earlier, otherwise you will receive a response such as: **The provided access grant is invalid, expired, or revoked.** See "To Create a Permission Ticket".
- ❷ Specify the same claim token as the first request for an RPT.

The `access_token` is the RPT, which lets the requesting party access the resource through a client.

5. (Optional) You can use the `/oauth2/introspect` endpoint to inspect the properties of the RPT. Use the PAT issued to the resource owner for authenticating to the authorization server, and specify the RPT token in a query parameter named `token`, as follows:

```
$ curl --header 'authorization: Bearer 057ad16f-7dba-4049-9f34-e609d230d43a' \  
'https://openam.example.com:8443/openam/oauth2/realms/root/introspect? \  
token=Aw4a92ZoKsjadWKw2d4Rmcjv7DM' \  
{ \  
  "active": true, \  
  "permissions": [ \  
    { \  
      "resource_id": "ef4d750e-3831-483b-b395-c6f059b5e15d0", \  
      "resource_scopes": [ \  
        "download" \  
      ], \  
      "exp": 1522334692 \  
    } \  
  ], \  
  "token_type": "access_token", \  
  "exp": 1522334692, \  
  "iss": "https://openam.example.com:8443/openam/oauth2" \  
}
```

Chapter 9

Extending UMA

AM exposes extension points that enable you to extend UMA services when built-in functionality does not fit your deployment.

AM provides a number of extension points for extending the UMA workflow that are provided as filters and that are dynamically loaded by using the Java [ServiceLoader](#) framework during the UMA workflow.

The extension points available are described in the sections below:

- "Resource Registration Extension Point"
- "Permission Request Extension Point"
- "Authorization Request Extension Point"
- "Resource Sharing Extension Point"

Resource Registration Extension Point

AM provides the [ResourceRegistrationFilter](#) extension point, which can be used to extend UMA resource registration functionality.

Resource Registration Extension Methods

Method	Parameters	Description
beforeResourceRegistration	<i>resourceSet</i> (type: ResourceSetDescription)	Invoked before a resource is registered in the backend. Changes made to the <i>resourceSet</i> object at this stage <i>will</i> be persisted.
afterResourceRegistration	<i>resourceSet</i> (type: ResourceSetDescription)	Invoked after a resource is registered in the backend. Changes made to the <i>resourceSet</i> object at this stage <i>will not</i> be persisted.

Permission Request Extension Point

AM provides the `PermissionRequestFilter` extension point, which can be used to extend UMA permission request functionality.

Permission Request Extension Methods

Method	Parameters	Description
<code>onPermissionRequest</code>	<p><code>resourceSet</code> (type: <code>ResourceSetDescription</code>)</p> <p><code>requestedScopes</code> (type: <code>Set<String></code>)</p> <p><code>requestingClientId</code> (type: <code>String</code>)</p>	Invoked before a permission request is created.

Authorization Request Extension Point

AM provides the `RequestAuthorizationFilter` extension point, which can be used to extend UMA authorization functionality.

Authorization Request Extension Methods

Method	Parameters	Description
<code>beforeAuthorization</code>	<p><code>permissionTicket</code> (type: <code>PermissionTicket</code>)</p> <p><code>requestingParty</code> (type: <code>Subject</code>)</p> <p><code>resourceOwner</code> (type: <code>Subject</code>)</p> <p><code>requestedScope</code> (type: <code>Set<String></code>)</p>	<p>Invoked before authorization of a request is attempted.</p> <p>Throws <code>UmaException</code> if authorization of the request should not be attempted.</p>
<code>afterSuccessfulAuthorization</code>	<p><code>permissionTicket</code> (type: <code>PermissionTicket</code>)</p> <p><code>requestingParty</code> (type: <code>Subject</code>)</p> <p><code>resourceOwner</code> (type: <code>Subject</code>)</p> <p><code>requestedScope</code> (type: <code>Set<String></code>)</p> <p><code>grantedScope</code> (type: <code>Set<String></code>)</p>	Invoked after a successful request authorization attempt.
<code>afterFailedAuthorization</code>	<p><code>permissionTicket</code> (type: <code>PermissionTicket</code>)</p> <p><code>requestingParty</code> (type: <code>Subject</code>)</p>	Invoked after a failed request authorization attempt.

Method	Parameters	Description
	<i>resourceOwner</i> (type: Subject) <i>requestedScope</i> (type: Set<String>)	

Resource Sharing Extension Point

AM provides the **ResourceDelegationFilter** extension point, which can be used to extend UMA resource sharing functionality.

Resource Sharing Extension Methods

Method	Parameters	Description
beforeResourceShared	<i>umaPolicy</i> (type: UmaPolicy)	Invoked before creating a sharing policy for a resource. Changes to the <i>umaPolicy</i> object at this stage <i>will</i> be persisted. Throws ResourceException if a sharing policy for the resource should not be created.
afterResourceShared	<i>umaPolicy</i> (type: UmaPolicy)	Invoked after creating a sharing policy for a resource. Changes to the <i>umaPolicy</i> object at this stage <i>will not</i> be persisted.
beforeResourceSharedModification	<i>currentUmaPolicy</i> (type: UmaPolicy) <i>updatedUmaPolicy</i> (type: UmaPolicy)	Invoked before altering the sharing policy of a resource. Changes to the <i>updatedUmaPolicy</i> object at this stage <i>will</i> be persisted. Throws ResourceException if the sharing policy of the resource should not be modified.
onResourceSharedDeletion	<i>umaPolicy</i> (type: UmaPolicy)	Invoked before deleting the sharing policy of a resource.

Method	Parameters	Description
		Throws <code>ResourceException</code> if the sharing policy of the resource should not be deleted.
<code>beforeQueryResourceSets</code>	<p><code>userId</code> (type: <code>String</code>)</p> <p><code>queryFilter</code> (type: <code>QueryFilter<JsonPointer></code>)</p>	<p>Invoked before querying the resources owned or shared with a user.</p> <p>The <code>userId</code> parameter provides the ID of the user making the query request.</p> <p>The <code>queryFilter</code> parameter provides the incoming request query filter.</p> <p>Returns a <code>QueryFilter</code> that can be used to return the user's resources.</p>

Chapter 10

UMA Endpoints

When acting as an UMA server, AM exposes the following endpoints. Click on them for more information about what they can do, and how to use them:

Endpoint	Description
/uma/resource_set	Lets users register and manage UMA resources (Federated Authorization for User-Managed Access (UMA) 2.0 endpoint).
/uma/permission_request	Returns permission tickets during the UMA grant flow (Federated Authorization for User-Managed Access (UMA) 2.0 endpoint).
/json/users/{user}/uma/policies	Creates and manages UMA-related authorization policies.
/json/users/{user}/oauth2/resources/labels	Creates, queries, and deletes UMA user labels.
/json/users/{user}/uma/pendingrequests	Manages UMA resource pending requests.
/uma/.well-known/uma2-configuration	Exposes provider configuration during UMA discovery.

Tip

For examples of most of the calls to the endpoints, see "The Postman Collection".

/uma/resource_set

UMA resource registration endpoint, as defined in the Federated Authorization for User-Managed Access (UMA) 2.0 specification.

Use this endpoint to register, read, delete, edit, and list resources for a particular resource owner.

+ Supported HTTP Methods

Action	HTTP Method
Register	POST
Read	GET

Action	HTTP Method
Update	PUT
Delete	DELETE
List	GET

You must compose the path to the token endpoint addressing the specific realm where the token will be issued. For example, https://openam.example.com:8443/openam/uma/realms/root/realms/subrealm1/resource_set.

The resource registration endpoint does not support any parameters. To authenticate to the endpoint, send an **Authorization: Bearer** header with the PAT of the resource owner.

To create and update resources, add their description to the body of the call as a JSON document that follows the UMA 2.0 specification. For example:

```
{
  "resource_scopes": [
    "view", "comment", "download"
  ],
  "name": "My Resource Name",
  "description": "An example resource stored in resourceserver.example.com",
  "type": "https://resourceserver.example.com/resources/",
  "icon_uri": "https://resourceserver.example.com/resources/resources.png"
}
```

The **resource_scopes** object is the only required object, and indicates the scopes that can be requested for the resource. Scope descriptions are not supported.

When reading, updating, and deleting a resource, you must include the resource ID in the URL. For example:

```
$ curl \
--header "Authorization: Bearer 515d6551-6512-5279-98b6-c0ef3f03a723" \
https://openam.example.com:8443/openam/uma/realms/root/resource_set/126615ba-b7fd-4660-b281-bae81aa45f7c0
```

For examples of the different REST calls, see "How to Manage UMA Resources".

/uma/permission_request

UMA permission endpoint, as defined in the Federated Authorization for User-Managed Access (UMA) 2.0 specification.

Use this endpoint to request permission tickets to the authorization server during the UMA grant flow.

+ *Supported HTTP Methods*

Action	HTTP Method
Request	POST

You must compose the path to the token endpoint addressing the specific realm where the token will be issued. For example, https://openam.example.com:8443/openam/uma/realms/root/realms/subrealm1/permission_request.

The permission request endpoint does not support any parameters. To authenticate to the endpoint, send an **Authorization: Bearer** header with the PAT of the resource owner.

To request a ticket, send an HTTP POST call to the endpoint specifying the resource and the scope that the permission ticket applies to in body of the call as a JSON document that follows the UMA 2.0 specification. For example:

```
$ curl -X POST \
--header 'authorization: Bearer 057ad16f-7dba-4049-9f34-e609d230d43a' \
--header 'cache-control: no-cache' \
--header 'content-type: application/json' \
--data '[
  {
    "resource_id" : "ef4d750e-3831-483b-b395-c6f059b5e15d0",
    "resource_scopes" : ["download"]
  }
]' \
https://openam.example.com:8443/openam/uma/realms/root/permission_request
{
  "ticket": "eyJ0eXAiOiJ...XPeJi3E"
}
```

Both of the objects in the JSON body are required. To obtain the resource ID, query the `/uma/resource_set` endpoint.

Tip

The default lifetime for a permission ticket is 120 seconds. Alter it by going to Realms > *Realm Name* > Services > UMA Provider, and editing the Permission Ticket Lifetime (seconds) property.

For an example of requesting a permission ticket in the flow, see *"The UMA Grant Flow"*.

/json/users/{user}/uma/policies

AM-specific endpoint used to create, delete, read, update, and query UMA policies.

+ *Supported HTTP Methods*

Action	HTTP Method
Create	PUT

Action	HTTP Method
Read	GET
Update	PUT
Delete	DELETE
Query	GET

Tip

Use the AM API Explorer for detailed information about this endpoint, and to test it against your deployed AM instance.

In the AM console, click the Help icon, and then go to API Explorer > users > {user} > uma > policies.

You must compose the path to the token endpoint addressing the specific realm where the token will be issued. For example, <https://openam.example.com:8443/openam/json/realms/root/realms/subrealm1/users/{user}/uma/policies>.

The policies endpoint does not support any parameters. To authenticate to the endpoint, send the SSO token of the resource owner as the value of the `iplanetDirectoryPro` header.

To create or update a policy, make an HTTP PUT call to the endpoint, adding the description of the policy as a JSON document in the body. For example:

```
{
  "policyId": "UMA_resource_ID_12345678",
  "permissions":
  [
    {
      "subject": "requesting_party_identity",
      "scopes": [
        "view",
        "comment",
        "download"
      ]
    }
  ]
}
```

All the objects in the JSON are mandatory.

The value of the `policyID` object is an UMA resource ID. To obtain it, query the `/uma/resource_set` endpoint.

The value of the `subject` object is the username or identity associated with the requesting party. In other words, the person, device, or client that the policy grants permission to.

The value of the `scopes` object is an array of permissions or scopes that are granted to the `subject`. They must be in concordance with the scopes supported by the resource that the policy protects.

For examples of using this endpoint, see "How to Manage UMA Policies".

/json/users/{user}/oauth2/resources/labels

AM-specific endpoint used to create, delete, and query UMA user and star labels.

+ Supported HTTP Methods

Action	HTTP Method
Create	POST
Query	GET
Delete	DELETE

Tip

Use the AM API Explorer for detailed information about this endpoint, and to test it against your deployed AM instance.

In the AM console, click the Help icon, and then go to API Explorer > users > {user} > oauth2 > resources > labels.

You must compose the path to the token endpoint addressing the specific realm where the token will be issued. For example, <https://openam.example.com:8443/openam/json/realms/root/realms/subrealm1/users/{user}/oauth2/resources/labels>.

The labels endpoint does not support any parameters. To authenticate to the endpoint, send the SSO token of the resource owner as the value of the `iplanetDirectoryPro` header.

To create a label, make an HTTP POST call to the endpoint, adding the description of the label as a JSON document in the body. For example:

```
{
  "name" : "My Favorites",
  "type" : "STAR"
  "resourceSetIDs": [
    "UMA_resource_ID_1234567890",
    "UMA_resource_ID_0987654321"
  ]
}
```

The value of the `type` object can be `USER`, for user labels, and `STAR`, for star labels. For more information about the different types, see "[Managing UMA Labels](#)".

The `resourceSetIDs` object is an array of UMA resource IDs that the label applies to. It is not mandatory; if you do not add it, the label will be created without any resource associated, and you will need to update the `resource` to add it to the label, since the labels endpoint does not support updating labels.

For examples, see "How to Manage UMA User and Favorite Labels".

/json/users/{user}/uma/pendingrequests

AM-specific endpoint used to list, approve, or deny pending authorization requests on a resource.

+ Supported HTTP Methods

Action	HTTP Method
Approve, Approve All	POST
Deny, Deny All	POST
Query	GET

Tip

Use the AM API Explorer for detailed information about this endpoint, and to test it against your deployed AM instance.

In the AM console, click the Help icon, and then go to API Explorer > users > {user} > uma > pendingrequests.

You must compose the path to the token endpoint addressing the specific realm where the token will be issued. For example, <https://openam.example.com:8443/openam/json/realms/root/realms/subrealm1/users/{user}/oauth2/resources/labels>.

To authenticate to the endpoint, send the SSO token of the resource owner as the value of the `iPlanetDirectoryPro` header.

The endpoint supports the following actions:

approve

Approves the permission request specified in the endpoint's URL. It does not grant the permission requested. Instead, it grants the scopes sent as a JSON document in the body of the `approve` call. For example:

```
$ curl \
  --request POST \
  --header "iPlanetDirectoryPro: AQIC5wM2LY4S...Q4MTE4NTA2*" \
  --header "Accept-API-Version: resource=1.0" \
  --data {
    "scopes": [
      "comment"
    ]
  } \
  https://openam.example.com:8443/openam/json/realms/root/users/demo/uma/
  pendingrequests/0d7790de-9066-4bb6-8e81-25b6f9d0b8853?_action=approve
```

approveAll

Approves every pending permission request for the user, but does not grant the permissions requested. Instead, it grants the scopes sent as a JSON document in the body of the `approveAll` call. For example:

```
$ curl \
--request POST
--header "iPlanetDirectoryPro: AQIC5wM2LY4S...Q4MTE4NTA2*" \
--header "Accept-API-Version: resource=1.0" \
--data {
  "scopes": [
    "comment"
  ]
} \
https://openam.example.com:8443/openam/json/realms/root/users/demo/uma/pendingrequests?
_action=approveAll
```

deny

Denies the permission request specified in the endpoint's URL. For example:

```
$ curl \
--request POST
--header "iPlanetDirectoryPro: AQIC5wM2LY4S...Q4MTE4NTA2*" \
--header "Accept-API-Version: resource=1.0" \
https://openam.example.com:8443/openam/json/realms/root/users/demo/uma/
pendingrequests/0d7790de-9066-4bb6-8e81-25b6f9d0b8853?_action=deny
```

denyAll

Denies every pending permission request for the user. For example:

```
$ curl \
--request POST
--header "iPlanetDirectoryPro: AQIC5wM2LY4S...Q4MTE4NTA2*" \
--header "Accept-API-Version: resource=1.0" \
https://openam.example.com:8443/openam/json/realms/root/users/demo/uma/pendingrequests?_action=denyAll
```

AM returns an HTTP 200 message with an empty body if the request is successful, and an HTTP 500 message if not.

The endpoint also supports the `_queryFilter` parameter to request a list of pending permission requests for the user. Specify `true` to match every request, `false` to match no request. For example:


```
$ curl \
--request GET
--header "iPlanetDirectoryPro: AQIC5wM2LY4S...Q4MTE4NTA2*" \
--header "Accept-API-Version: resource=1.0" \
https://openam.example.com:8443/openam/json/realms/root/users/demo/uma/pendingrequests?_queryFilter=true
{
  "result": [
    {
      "_id": "0d3190ef-6901-654b-82pa-13a6h9d0b53452",
      "user": "bob",
      "resource": "My Resource Name",
      "when": 1607002810,
      "permissions": [
        "download"
      ]
    }
  ],
  "resultCount": 1,
  "pagedResultsCookie": "10d7790de",
  "totalPagedResultsPolicy": "EXACT",
  "totalPagedResults": 0,
  "remainingPagedResults": 0
}
```

AM also provides built-in user pages in the UI to view pending resource access requests:

To View and Manage Pending Access Requests

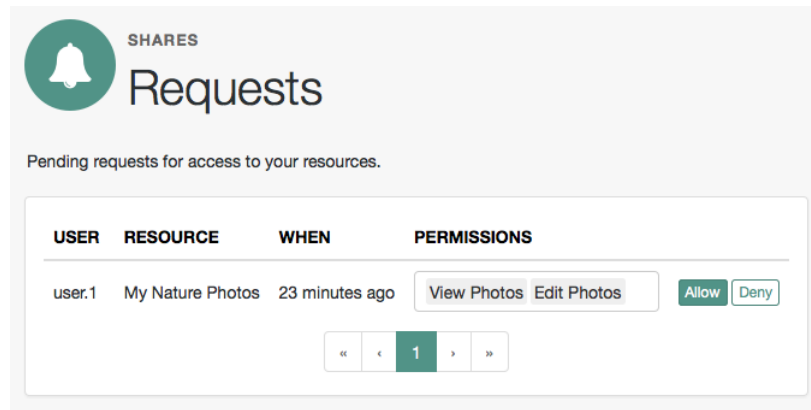
AM supports an UMA workflow in which a user can request access to a resource that has not been explicitly shared with them. The resource owner receives a notification of the request and can choose to allow or deny access.

Manage pending requests for access to resources by using the steps below:

1. Log in to AM as the resource owner, and then go to Shares > Requests.

The Requests page is displayed:

UMA Requests Screen Presented to the Resource Owner



2. Review the pending request, and take one of the following actions:

- Click Allow to approve the request.

Tip

You can remove permissions from the request by clicking the permission, then press either **Delete** or **Backspace**. Select the permission from the drop-down list to return it to the permissions granted to the resource owner.

The required UMA policy will be created, and optionally, the requesting party will be notified that they can now access the resource.

The requesting party can view a list of resources to which they have access by navigating to Shares > Resources > Shared with me.

- Click Deny to prevent the requesting party from accessing the resource. The pending request is removed, and the requesting party will not be notified.

3. After allowing or denying access to a resource, an entry is created in the History page.

To view a list of actions that have occurred, go to Shares > History.

/uma/.well-known/uma2-configuration

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

A resource server or client can perform an HTTP GET on </uma/.well-known/uma2-configuration> to retrieve a JSON object indicating the UMA configuration.

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`.

The following is an example of a GET request to the UMA 2.0 configuration discovery endpoint for a subrealm named `subrealm` in the Top Level Realm:

```
$ curl \
--request GET \
https://openam.example.com:8443/openam/uma/realms/root/realms/subrealm/.well-known/uma2-configuration
{
  "issuer": "https://openam.example.com:8443/openam/oauth2/subrealm",
  "grant_types_supported": [
    "urn:ietf:params:oauth:grant-type:saml2-bearer",
    "urn:ietf:params:oauth:grant-type:uma-ticket",
    "client_credentials",
    "password",
    "authorization_code",
    "urn:ietf:params:oauth:grant-type:device_code",
    "http://oauth.net/grant_type/device/1.0"
  ],
  "token_endpoint_auth_methods_supported": [
    "client_secret_post",
    "private_key_jwt",
    "client_secret_basic"
  ],
  "revocation_endpoint_auth_methods_supported": [
    "client_secret_post",
    "private_key_jwt",
    "client_secret_basic"
  ],
  "response_types_supported": [
    "code token id_token",
    "code",
    "code id_token",
    "device_code",
    "id_token",
    "code token",
    "token",
    "token id_token"
  ],
  "jwks_uri": "https://openam.example.com:8443/openam/oauth2/realms/root/realms/subrealm/connect/
jwk_uri",
  "dynamic_client_endpoint": "https://openam.example.com:8443/openam/oauth2/realms/root/realms/subrealm/
register",
  "token_endpoint": "https://openam.example.com:8443/openam/oauth2/realms/root/realms/subrealm/
access_token",
  "authorization_endpoint": "https://openam.example.com:8443/openam/oauth2/realms/root/realms/subrealm/
authorize",
  "revocation_endpoint": "https://openam.example.com:8443/openam/oauth2/realms/root/realms/subrealm/
token/revoke",
  "introspection_endpoint": "https://openam.example.com:8443/openam/oauth2/realms/root/realms/subrealm/
introspect",
  "resource_registration_endpoint": "https://openam.example.com:8443/openam/uma/realms/root/realms/
subrealm/resource_set",
  "permission_endpoint": "https://openam.example.com:8443/openam/uma/realms/root/realms/subrealm/
permission_request"
}
```

The JSON object returned includes the following configuration information:

issuer

The URI of the issuing authorization server.

grant_types_supported

The supported OAuth 2.0 grant types.

token_endpoint

The URI to request tokens.

authorization_endpoint

The URI to request authorization for issuing a token.

introspection_endpoint

The URI to introspect an RPT.

For more information, see `"/oauth2/introspect"` in the *OAuth 2.0 Guide*.

resource_registration_endpoint

The URI for a resource server to register a resource.

For more information, see `"/uma/resource_set"`.

dynamic_client_endpoint

The URI for registering a dynamic client.

Tip

Resource servers and clients need to be able to discover the UMA provider for a resource owner. You should consider redirecting requests to URIs at the server root, such as `https://www.example.com/.well-known/uma2-configuration`, to the well-known URIs in AM's space.

For example, if your UMA provider is in a subrealm named `subrealm`, you could map the following URI: `https://www.example.com:8080/openam/uma/realms/root/realms/subrealm/.well-known/uma2-configuration`.

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

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

Chapter 11

Reference

This reference section covers supported standards, settings and other information related to UMA. For the global services reference, see [Reference](#).

UMA Configuration Reference

This section covers reference for UMA global settings and UMA datastore server settings:

- To configure UMA global settings, go to [Configure > Global Settings > UMA Provider](#). For more information, see "UMA Provider" in the *Reference*.
- To configure UMA data store settings:
 - Go to [Configure > Server Defaults > UMA](#) to configure the settings for all your servers.
 - Go to [Deployment > Servers > Server Name > UMA](#) to configure the settings for one server.

For more information, see "UMA Properties".

UMA Properties

UMA server settings are inherited by default.

UMA Resource Store

The following settings appear on the UMA Resource Store tab:

Store Mode

Specifies the data store where AM stores UMA tokens. Possible values are:

- **Default Token Store**: AM stores UMA tokens in the configuration data store.
- **External Token Store**: AM stores UMA tokens in an external data store.

Root Suffix

Specifies the base DN for storage information in LDAP format, such as `dc=uma-resources,dc=example,dc=com`.

Max Connections

Specifies the maximum number of connections to the data store.

External UMA Resource Store Configuration

AM honors the following properties when **External Token Store** is selected under the Resource Sets Store tab:

SSL/TLS Enabled

When enabled, AM uses SSL or TLS to connect to the external data store. Make sure AM trusts the data store's certificate when using this option.

Connection String(s)

Specifies an ordered list of connection strings for external data stores. The format is **HOST:PORT[|SERVERID[|SITEID]]**, where **HOST:PORT** specify the FQDN and port of the data store, and **SERVERID** and **SITEID** are optional parameters that let you prioritize the particular connection when used by the specified node(s).

Multiple connection strings must be comma-separated, for example, **uma-ldap1.example.com:389|1|1,uma-ldap2.example.com:389|2|1**.

See the entry for Connection String(s) in "CTS Properties" in the *Reference* for more syntax examples.

Login Id

Specifies the username AM uses to authenticate to the data store. For example, **uid=am-uma-bind-account,ou=admins,dc=uma,dc=example,dc=com**. This user must be able to read and write to the root suffix of the data store.

Password

Specifies the password associated with the login ID property.

Heartbeat

Specifies, in seconds, how often AM should send a heartbeat request to the data store to ensure that the connection does not remain idle.

Default: **10**

UMA Audit Store

The following settings appear on the UMA Audit Store tab:

Store Mode

Specifies the data store where AM stores audit information generated when users access UMA resources. Possible values are:

- **Default Token Store:** AM stores UMA audit information in the configuration data store.
- **External Token Store:** AM stores UMA audit information in an external data store.

Root Suffix

Specifies the base DN for storage information in LDAP format, such as `dc=uma-audit,dc=example,dc=com`.

Max Connections

Specifies the maximum number of connections to the data store.

External UMA Audit Store Configuration

AM honors the following properties when **External Token Store** is selected under the UMA Audit Store tab:

SSL/TLS Enabled

When enabled, AM uses SSL or TLS to connect to the external data store. Make sure AM trusts the data store's certificate when using this option.

Connection String(s)

Specifies an ordered list of connection strings for external data stores. The format is `HOST:PORT[|SERVERID[|SITEID]]`, where `HOST:PORT` specify the FQDN and port of the data store, and `SERVERID` and `SITEID` are optional parameters that let you prioritize the particular connection when used by the specified node(s).

Multiple connection strings must be comma-separated, for example, `uma-ldap1.example.com:389|1|1,uma-ldap2.example.com:389|2|1`.

See the entry for Connection String(s) in "CTS Properties" in the *Reference* for more syntax examples.

Login Id

Specifies the username AM uses to authenticate to the data store. For example, `uid=am-uma-bind-account,ou=admins,dc=uma,dc=example,dc=com`. This user must be able to read and write to the root suffix of the data store.

Password

Specifies the password associated with the login ID property.

Heartbeat

Specifies, in seconds, how often AM should send a heartbeat request to the data store to ensure that the connection does not remain idle.

Default: 10

Pending Requests Store

The following settings appear on the Pending Requests Store tab:

Store Mode

Specifies the data store where AM stores pending requests to UMA resources. Possible values are:

- **Default Token Store**: AM stores UMA pending requests in the configuration data store.
- **External Token Store**: AM stores UMA pending requests in an external data store.

Root Suffix

Specifies the base DN for storage information in LDAP format, such as `dc=uma-pending,dc=forgerock,dc=com`.

Max Connections

Specifies the maximum number of connections to the data store.

External Pending Requests Store Configuration

AM honors the following properties when **External Token Store** is selected under the Pending Requests Store tab:

SSL/TLS Enabled

When enabled, AM uses SSL or TLS to connect to the external data store. Make sure AM trusts the data store's certificate when using this option.

Connection String(s)

Specifies an ordered list of connection strings for external data stores. The format is `HOST:PORT[SERVERID[|SITEID]]`, where `HOST:PORT` specify the FQDN and port of the data store, and `SERVERID` and `SITEID` are optional parameters that let you prioritize the particular connection when used by the specified node(s).

Multiple connection strings must be comma-separated, for example, `uma-ldap1.example.com:389|1|1,uma-ldap2.example.com:389|2|1`.

See the entry for Connection String(s) in "CTS Properties" in the *Reference* for more syntax examples.

Login Id

Specifies the username AM uses to authenticate to the data store. For example, `uid=am-uma-bind-account,ou=admins,dc=uma,dc=example,dc=com`. This user must be able to read and write to the root suffix of the data store.

Password

Specifies the password associated with the login ID property.

Heartbeat

Specifies, in seconds, how often AM should send a heartbeat request to the data store to ensure that the connection does not remain idle.

Default: `10`

UMA Resource Labels Store

The following settings appear on the UMA Resource Labels Store tab:

Store Mode

Specifies the data store where AM stores user-created labels used for organizing UMA resources. Possible values are:

- `Default Token Store`: AM stores user-created labels in the configuration data store.
- `External Token Store`: AM stores user-created labels in an external data store.

Root Suffix

Specifies the base DN for storage information in LDAP format, such as `dc=uma-resources-labels,dc=forgerock,dc=com`.

Max Connections

Specifies the maximum number of connections to the data store.

External UMA Resource Labels Store Configuration

AM honors the following properties when `External Token Store` is selected under the UMA Resource Labels Store tab.

SSL/TLS Enabled

When enabled, AM uses SSL or TLS to connect to the external data store. Make sure AM trusts the data store's certificate when using this option.

Connection String(s)

Specifies an ordered list of connection strings for external data stores. The format is `HOST:PORT[|SERVERID[|SITEID]]`, where `HOST:PORT` specify the FQDN and port of the data store, and `SERVERID` and `SITEID` are optional parameters that let you prioritize the particular connection when used by the specified node(s).

Multiple connection strings must be comma-separated, for example, `uma-ldap1.example.com:389|1|1,uma-ldap2.example.com:389|2|1`.

See the entry for Connection String(s) in "CTS Properties" in the *Reference* for more syntax examples.

Login Id

Specifies the username AM uses to authenticate to the data store. For example, `uid=am-uma-bind-account,ou=admins,dc=uma,dc=example,dc=com`. This user must be able to read and write to the root suffix of the data store.

Password

Specifies the password associated with the login ID property.

Heartbeat

Specifies, in seconds, how often AM should send a heartbeat request to the data store to ensure that the connection does not remain idle.

Default: `10`

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.