



User-Managed Access (UMA) 2.0 Guide

/ ForgeRock Access Management 6

Latest update: 6.0.0.7

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

Copyright © 2011-2018 ForgeRock AS.

Abstract

Guide to configuring and using UMA features in ForgeRock® Access Management.



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

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

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

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

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

DejaVu Fonts

Bitstream Vera Fonts Copyright

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

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

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

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

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

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

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

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

Arev Fonts Copyright

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

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

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

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

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

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

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

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

FontAwesome Copyright

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

This Font Software is licensed under the SIL Open Font License, Version 1.1. This license is available with a FAQ at: <http://scripts.sil.org/OFL>.

Table of Contents

Preface	iv
1. Introducing UMA 2.0	1
1.1. UMA 2.0 Specifications	1
1.2. UMA 2.0 Actors and Actions	2
1.3. UMA 2.0 Process Flow	4
1.4. UMA 2.0 Example Use Case	6
2. Configuring an UMA 2.0 Demo System	8
2.1. Before You Begin	8
2.2. UMA Setup Procedures	8
2.3. Register Resource Sets and Policies	12
2.4. Test the UMA Grant Flow	16
2.5. Additional Functionality for UMA End Users	22
3. Using UMA 2.0	28
3.1. Discovering UMA Configuration	28
3.2. Configuring UMA Stores	30
3.3. Managing UMA Resource Sets	32
3.4. Managing UMA Labels	37
3.5. Managing UMA 2.0 Policies	40
4. Customizing UMA	48
4.1. Resource Set Registration Extension Point	48
4.2. Permission Request Extension Point	49
4.3. Authorization Request Extension Point	49
4.4. Resource Sharing Extension Point	50
5. Reference	52
5.1. UMA Supported Standards	52
5.2. UMA Configuration Reference	52
A. Getting Support	60
A.1. Accessing Documentation Online	60
A.2. Using the ForgeRock.org Site	60
A.3. Getting Support and Contacting ForgeRock	61
Glossary	62

Preface

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 6 adds support for 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.

This guide is written for anyone who wants to set up Access Management for UMA 2.0 features.

About ForgeRock Identity Platform™ Software

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

Chapter 1

Introducing UMA 2.0

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

UMA 2.0 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. The authorization server 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.

1.1. UMA 2.0 Specifications

AM supports the UMA 2.0 protocol, which is defined by two specifications issued by the Kantara Initiative:

- 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 a requesting party token (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 (OIDC) ID token¹.

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.

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

¹The OIDC ID token is a signed and optionally encrypted JSON Web Token (JWT).

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).

1.2. UMA 2.0 Actors and Actions

UMA 2.0 protocol introduces actors and their interactions in its protocol. This section provides a brief description to help you better understand the model.

1.2.1. UMA 2.0 Actors

UMA 2.0 uses the OAuth 2.0 actors in slightly extended ways and introduces one new actor:

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 6 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, 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 (JWTs) to provide claim tokens and its policy engine for UMA resource management.

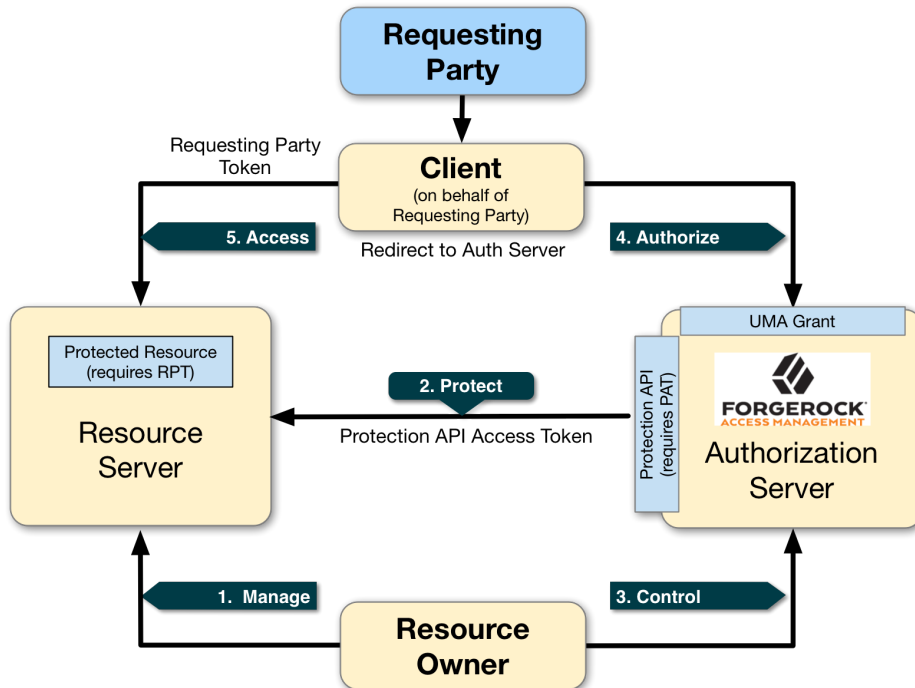
requesting party

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.

1.2.2. UMA 2.0 Actions

UMA 2.0 defines the following actions in the workflow as illustrated in "Actors and Actions in the UMA 2.0 Workflow":

Actors and Actions in the UMA 2.0 Workflow



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 they are loosely coupled, 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 ticket, and token inspection endpoints to bind the resource server and authorization server.

The API endpoints are protected by a protection API access token (PAT)—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 "Managing UMA Resource Sets".

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 a requesting party token (RPT).

For more information, see "Managing UMA 2.0 Policies".

4. Authorize

The client, acting on behalf of the requesting party, uses the authorization server's UMA Grant Flow to acquire a requesting party token (RPT), 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 want². 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. The RPT is a claims token with time-limited scoped permissions.

1.3. UMA 2.0 Process Flow

The UMA 2.0 process largely involves the UMA 2.0 Grant flow, in which a requesting party obtains a 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 (as shown in the sequence diagram), when needed for policy creation, and at resource access attempt.

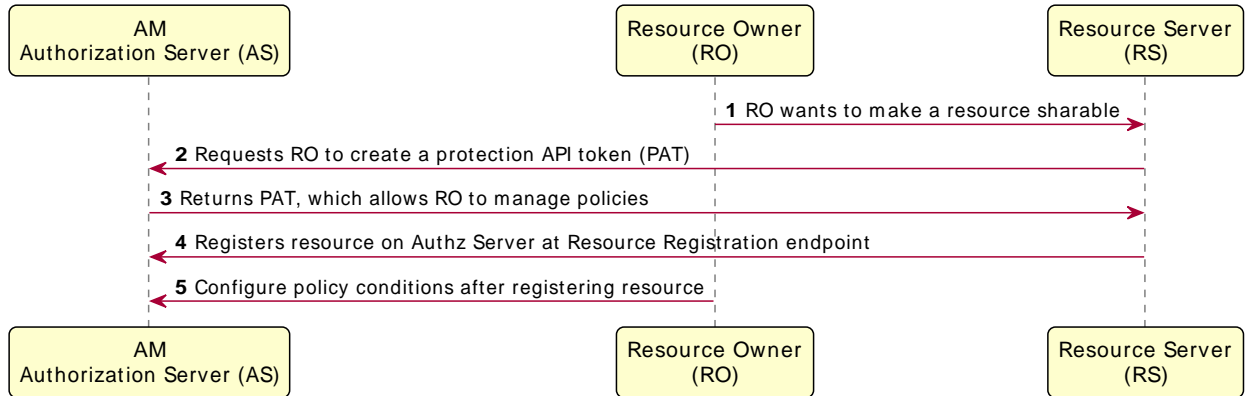
See the section, "Considerations Regarding Resource Registration Timing and Mechanism" in the UMA Implementer's Guide for information.

The following sequence diagram outlines a successful registration of a protected resource:

²In some cases, the requesting party and the resource owner may be the same entity.

UMA 2.0 Protecting a Resource Flow

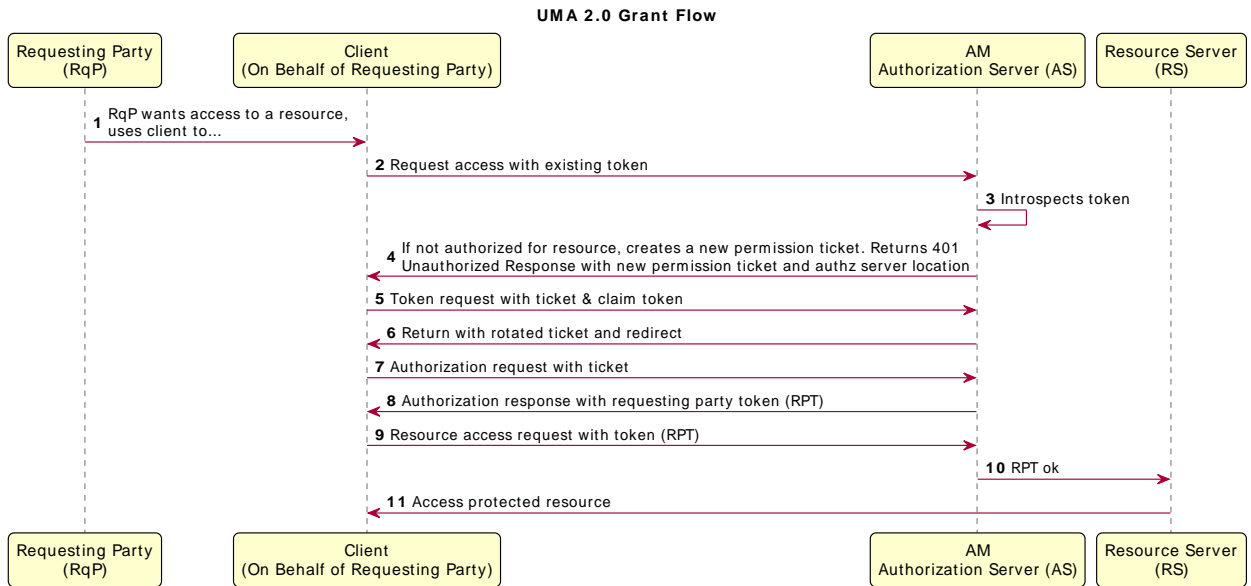
UMA 2.0 Flow for Protecting a Resource



- 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 an protection API access token (PAT) on the authorization server (**2**).
- The authorization server returns a PAT, which allows the resource owner to 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**).

The following sequence diagram outlines a successful UMA 2.0 grant flow where the client accesses the protected resource:

UMA 2.0 Grant Flow Process



- A requesting party, using a client application, requests access to an UMA-protected resource (labeled **1** and **2** in the diagram above).
- The authorization server checks the existing token (**3**) and determines that the resource owner does not have the correct privileges to access the resource. The authorization 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 set, and the user-created policy, and if successful returns an RPT (**7** and **8**).
- The client presents the RPT to the authorization 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 requesting party (**11**).

1.4. UMA 2.0 Example Use Case

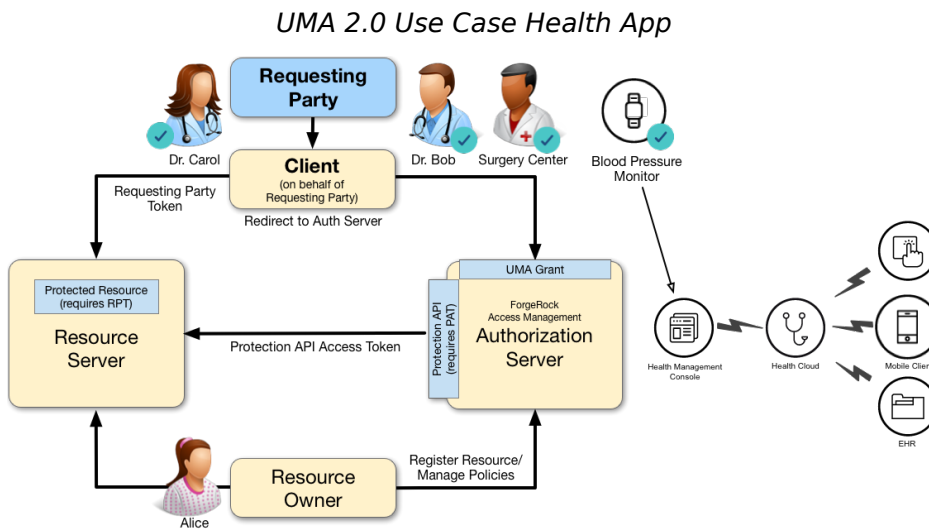
A resource owner, Alice, is a patient who plans to undergo a medical procedure at a surgery center. Dr. Bob is a specialist surgeon who needs read access (i.e., *read* scope) to Alice's electronic health records in order to operate, and write access (i.e., *write* scope) in order to add new entries related

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.



To view other Case Studies, see the [Kantara Initiative](#).

Chapter 2

Configuring an UMA 2.0 Demo System

This chapter explains how to set up a simple demo UMA 2.0 system with AM as an authorization server. The demo uses a single AM instance as authorization server, UMA client, and resource server agent. You will enter REST requests to mimic the actions of an UMA client, resource owner, authorization server, and requesting party.

2.1. Before You Begin

Before you configure an UMA 2.0 deployment, do the following:

- Install an AM instance. For this example, install AM default configuration with the embedded directory server. For instructions, see "First Steps" in the *Quick Start Guide*.
- (Optional). Install and configure ForgeRock Identity Gateway as an UMA resource server. For this example, the procedures uses `uri://-*-` as a sample resource server URI. See the *ForgeRock Identity Gateway Guide*.

2.2. UMA Setup Procedures

To set up an UMA 2.0 deployment, set up the components required for an UMA 2.0 deployment on AM:

UMA 2.0 Setup Procedures

Task	Description
Create an UMA provider service	"To Create the UMA Provider Service"
Create an UMA client agent	"To Create an UMA Client Agent"
Create an UMA resource server agent	"To Create an UMA Resource Server Agent"
Create an OAuth2/OIDC server	"To Create an OAuth2/OpenID Connect Service"
Create a resource owner.	"To Create an UMA Resource Owner"
Create a requesting party.	"To Create a Requesting Party"

To Create the UMA Provider Service

1. Log in to the AM console as an administrator.
2. In the AM console, select Realms > *Realm Name* > Dashboard > Configure OAuth Provider > Configure User Managed Access.

Important

Some IDM deployments use AM's UMA Provider Service to help support privacy use cases, such as the General Data Protection Regulation (GDPR). If you are configuring the UMA Provider Service as part of an IDM deployment, perform the following steps:

1. In the AM console, navigate to Realms > *Realm Name* > Services and add an UMA Provider service.
For information about the available attributes, see "UMA Provider".
2. Save your changes.

3. On the Configure UMA page, select the Realm for the provider service.
4. (Optional) If necessary, adjust the lifetimes for authorization codes, access tokens, and refresh tokens.
5. (Optional) Select Issue Refresh Tokens unless you do not want the authorization service to supply a refresh token when returning an access token.
6. (Optional) Select Issue Refresh Tokens on Refreshing Access Tokens if you want the authorization service to supply a new refresh token when refreshing an access token.
7. (Optional) If you have a custom scope validator implementation, put it on the AM classpath, for example `/path/to/tomcat/webapps/openam/WEB-INF/lib/`, and specify the class name in the Scope Implementation Class field. For an example, see "Customizing OAuth 2.0 Scope Handling" in the *OAuth 2.0 Guide*.
8. Click Create to save your changes. AM creates the following:
 - An UMA provider service.
 - An OAuth2 provider service that supports OpenID Connect.

Warning

If an UMA or OAuth 2.0 provider service already exists, it will be overwritten with the new values.

9. To access the provider service configuration in the AM console, browse to Realms > *Realm Name* > Services, and then click UMA Provider.

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

10. To complete the configuration, click Save Changes.

To Create an UMA Client Agent

Create a profile for the UMA client agent in AM for OAuth 2.0 and UMA 2.0:

1. Log in to the AM console as an administrator.
2. In the top level realm, select Applications > OAuth 2.0.
3. 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): **read openid**

Note

You will need to enter **read**, press Enter, and then enter **openid**.

- Default Scope(s): For this example, leave it blank.

To Create an UMA Resource Server Agent

Create a profile for the resource server agent in AM for OAuth 2.0 and UMA 2.0:

1. Log in to the AM console as an administrator.
2. In the top level realm, select Applications > OAuth 2.0.
3. 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.

To Create an OAuth2/OpenID Connect Service

1. Log in to the AM console as an administrator.
2. In the top level realm, click Configure OAuth Provider > Configure OpenID Connect. Accept the defaults, and click Create.

To Create an UMA Resource Owner

1. Log in to the AM console as an administrator.
2. In the top level realm, click Identities.
3. 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: **password**
 - Password (confirm): **password**
 - User Status: **Active**
4. Click OK to save the settings.

To Create a Requesting Party

1. Log in to the AM console as an administrator.
2. In the top level realm, click Identities.
3. 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: **password**

- Password (confirm): `password`
 - User Status: `Active`
4. Click OK to save the settings.

2.3. Register Resource Sets and Policies

The resource owner must acquire a protection API access token (PAT) to register a resource set and create UMA policies. The general flow to protect a resource can be seen in "UMA 2.0 Protecting a Resource Flow".

Resource registration can occur at three different stages: at initial resource creation, when needed for policy creation, and at resource access attempt. The resource registration process is the same regardless of when it is run.

To register a resource set and create UMA policies, perform the following procedures:

To Acquire a Protection API Token (PAT)

First, register an OAuth 2.0 client with a name, such as *UMA-Resource-Server* and a client password, such as *password*. Ensure that `uma_protection` is in the list of available scopes in the client, and a redirection URI is configured. See "Registering OAuth 2.0 Clients With the Authorization Service" in the *OAuth 2.0 Guide*.

This example uses the OAuth 2.0 Resource Owner Password Credentials grant type; however, the UMA resource server can use any of the OAuth 2.0 grants, except the client credentials grant, to obtain the access token.

After a suitable OAuth 2.0 client is configured, perform the following steps to acquire a PAT on behalf of the resource owner, `alice`:

- Create a POST request to the `/oauth2/access_token` endpoint. The following example uses the Resource Owner Password Credentials grant type:


```
$ curl \
--request POST \
--data 'grant_type=password' \
--data 'scope=uma_protection' \
--data 'username=alice' \
--data 'password=password' \
--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-e609d230d431a",
  "refresh_token": "340f82a4-9aa9-471c-ac42-f0ca1809c82b",
  "scope": "uma_protection",
  "token_type": "Bearer",
  "expires_in": 4999
}
```

- 1 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.

Register an UMA Resource Set

After acquiring a PAT, the resource owner can now register a resource set on the authorization server.

- Send a POST request to the `/uma/resource_set` endpoint to register the resource, `my_resource_106`. Make sure to use the `resource_scopes` attribute to define your scopes and the PAT as your resource owner's bearer token:

```

$ curl -X POST
\
--header 'authorization: Bearer 057ad16f-7dba-4049-9f34-e609d230d143a' \
--header 'cache-control: no-cache'
\
--header 'content-type: application/json'
\
--data '{
  "resource_scopes": [
    "view", "comment", "download"
  ],
  "name": "my resource 106",
  "type": "type",
  "uri": "http://rs.example.com/alice/myresource106"
}' \
https://openam.example.com:8443/openam/uma/resource_set
{
  "_id": "0d7790de-9066-4bb6-8e81-25b6f9d0b88523",
  "user_access_policy_uri": "https://openam.example.com:8443/openam/XUI/?realm=#uma/share/0d7790de-9066-4bb6-8e81-25b6f9d0b8853"
}

```

- ❶ Use the PAT Bearer Token previously acquired on behalf of the resource owner. See "To Acquire a Protection API Token (PAT)".
- ❷ The value returned in the `_id` property is the ID of the created UMA resource set.

To Create an UMA Policy

To create a policy, the resource owner must be logged in to the authorization server and have an SSO token issued to them, and must also have the resource set ID to be protected. This information is used when creating policies.

For more information on authenticating to AM to obtain an SSO token for a user, see "Using Authentication" in the *Authentication and Single Sign-On Guide*.

Note

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

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

Note

The SSO token must have been issued to the resource owner. In this example, the resource owner is `alice`. The following is an example of how to obtain the SSO token for a user:

```

$ curl \
--request POST \
--header "Content-Type: application/json" \
--header "X-OpenAM-Username: alice" \
--header "X-OpenAM-Password: password" \
--header "Accept-API-Version: resource=2.0, protocol=1.0" \
--data "{}" \
https://openam.example.com:8443/openam/json/realms/root/authenticate
{
  "tokenId": "AQIC5wM2LY4S...Q4MTE4NTA2*1",
  "successUrl": "/openam/console",
  "realm": "/"
}
    
```

- 1 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.

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

```

$ curl -X PUT \
--header 'Accept-API-Version: resource=1.0' \
--header 'cache-control: no-cache' \
--header 'content-type: application/json' \
--header "iPlanetDirectoryPro: AQIC5wM2LY4S...Q4MTE4NTA12*" \
--header "If-None-Match: *" \
--data '{
  "policyId": "0d7790de-9066-4bb6-8e81-25b6f9d0b8853",
  "permissions":
  [
    {
      "subject": "bob",
      "scopes": [
        "view",
        "comment"
      ]
    }
  ]
}' \
https://openam.example.com:8443/openam/json/users/alice/uma/policies/0d7790de-9066-4bb6-8e81-25b6f9d0b8853
{
  "_id": "0d7790de-9066-4bb6-8e81-25b6f9d0b8853",
  "_rev": "-1985914901"
}
    
```

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

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

If the permissions are not correct, an HTTP 400 Bad Request status code is returned, for example:

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

2.4. Test the UMA Grant Flow

The UMA grant flow grants a requesting party access token (RPT) to the requesting party to allow access to a resource. The general UMA grant flow can be seen in "UMA 2.0 Grant Flow Process".

To issue an RPT to a requesting party, run the following procedures:

UMA 2.0 Grant Flow

Task	Description
Create a permission ticket	"Create a Permission Ticket"
Gather claims to create a claim token	"Gather Claims"
Obtain an RPT	"Obtain a Requesting Party Token (RPT)"

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 a requesting party access token (RPT). A new permission ticket must be used for each attempt to acquire an RPT.

- Send a POST request to the `/uma/permission_request`:

```

curl -X POST
\
--header 'authorization: Bearer 057ad16f-7dba-4049-9f34-e609d230d143a' \
--header 'cache-control: no-cache'
\
--header 'content-type: application/json'
\
--data '[
  {
    "resource_id" : "ef4d750e-3831-483b-b395-c6f059b5e15d200",
    "resource_scopes" : ["download"]
  }
]' \
https://openam.example.com:8443/openam/uma/permission_request
{
  "ticket": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJhdWQiOiJodHRwOi8vb3BlbmFtLmRlZmF1bHQuZXhhbXBsZS5jb206ODAvb3BlbmFtL29hdXRoMiIsImZcyI6Imh0dHA6Ly9vcGVuYW0uZGVmYXVsZC5leGFtcGxlLmNvbTo4MCM9vcGVuYW0vb2F1dGgyIiwiaXQiOiJAsImV4cCI6MTUwNzE0ODI2MiwiZGlkIjoiaWZ0TUtYzZmMDU5YjVlMTVhMiIsImZvcmlkcm9jayI6eyJzaWciOiIzUyLG0k1uN0N0dVlIZVpQVVtrMGZlajJmL2JAVDBYfWI4dH1h0EQmIn19.vi6RxLMxL5MiB1uK60P1yE2HIhWnF7CDFgz4agM5YAQ" ❸
}

```

- ❶ Use the PAT Bearer Token previously acquired on behalf of the resource owner. See "To Acquire a Protection API Token (PAT)".
- ❷ Specify the ID of the registered resource set for which this permission ticket will maintain permission state information. See "Register an UMA Resource Set".
- ❸ The value returned in the `ticket` property is the permission ticket, which is used to obtain an RPT. See "Obtain a Requesting Party Token (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 navigating to `Realms > Realm Name > Services > UMA Provider`, and editing the `Permission Ticket Lifetime (seconds)` property.

Gather Claims

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

- Send a POST request to the `/oauth2/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 \
  \
  --header 'authorization: Basic VW1hQ2xpZW50nBhc3N3b3Jk'❶ \
  --header 'cache-control: no-cache' \
  \
  --header 'content-type: application/x-www-form-urlencoded' \
  \
  --data 'grant_type=password' \
  --data 'scope=openid' \
  --data 'username=bob' \
  --data 'password=password' \
https://openam.example.com:8443/openam/oauth2/access_token
{
  "access_token": "f09f55e5-5e9c-48fe-aeaa-d377de88e8e6",
  "refresh_token": "ee2d35f6-5819-4734-8b3e-9af77a545563",
  "scope": "openid",
  "id_token": "eyJ0eXAiOiJKV1QiLCJraWQiOiJpdj082T3ZWdJereStXZ3JINNVp0VUaW9MdDA
9IiwiaWxnb3JpdGkiOiJmNTYifQ.eyJhdF9oYXNoIjoiMmV0dGpUQU0JmLWdKeDFUZR
EdFlIdyIsInN1YiI6ImJvYiIsImF1ZG0VJmJyY2tpbmdJZCI6IjBmZjAxNzQ0LW
Y5ZDMtNGFhMi04NmU4LTcyNTJlYTUyMWQ4Yi04MDIiLCJpc3MiOiJodHRwOi8v
3BlmFtLmRlZmFlbHQuZXBhbnB5Sj5jb206ODAvb3BlbmFtL29hdXRoMiIsInRv
a2VuTmFtZSI6ImlkX3Rva2VuIiwiaXVkiOiVW1hQ2xpZW50IiwidXBkYXRlZlF9
hdCI6IjE1MDcwODc5MDc1MDc1MDc1MDc1MDc1MDc1MDc1MDc1MDc1MDc1MDc1MDc1
E1MDcwODgyNjcsInJlYXxtIjoiYiIsImV4cCI6MTUwNzE0ODI2NywidG9rZW5Ue
XB1IjoiaS1dUVG9rZW4iLCJpYXQiOiJlMDcwODgyNjcyNjcyNjcyNjcyNjcyNjcy
QwWxhv6CvqNEBk6_kDpat_u-gfwKZZRwmdvouLa9dNmGr33EQJY4LiFCUHA_AKn
048G3W6czBupC4TJESK4rDwKr20XPQVzTrGNk25ix5Dw_BdngW-YJfnXmLSByKt
ZmaYT7FGS0bdGMedtkypVpM6uZAQcc9JEm0L0CHF-l57NH88p0P7Mky9gszLGI
3cFEMHONTBB6-yroJyU-vaDjzWeCX3uGVqY8K3nUMqMguRSIkqrLtabyX4CuKtN
JlthJUEnX4mZFBnf7un7Qs3nx3BUAzclKjL2H1FNyD2a-zx3Y0J9Wm4kj_SlGtG
FBznEB5A",❷
  "token_type": "Bearer",
  "expires_in": 4999
}
```

- ❶ Authorize using the credentials of the UMA client. Use the client ID and client secret, separated by a colon character, for example `UmaClient:password`. Base64-encode the result. See "To Create an UMA Client Agent".
- ❷ The value returned in the `id_token` property is the claim token, which is used to obtain an RPT. See "Obtain a Requesting Party Token (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.

Obtain a Requesting Party Token (RPT)

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

1. Send a POST request to the `/oauth2/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 \
  --header 'authorization: Basic VW1hQ2xpZW50Nhc3N3b3Jk'❶ \
  --header 'cache-control: no-cache' \
  --header 'content-type: application/x-www-form-urlencoded' \
  --data 'grant_type=urn:ietf:params:oauth:grant-type:uma-ticket' \
  --data 'ticket=eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJhdWQiOiJodHRwOi8vb3BlbmFtLmRlZmF1bHQzXhhbXBsZS5jb206ODAvb3BlbmFtL29hdXRoMiIsImZyI6Imh0dHA6Ly9vcGVuYW0uZGVmYXVsdC5leG$'❷ \
  --data 'scope=download' \
  --data 'claim_token=eyJ0eXAiOiJKV1QiLCJraWQiOiJiL082T3ZwdjEreStXZ3JlNlVpOVdUaW9mMDDA9iIiwiaWxnIjoiaUlyMyNTYifQ.eyJhdF9oYXNoIjoieWExdHlyZFBtTYWwGfLlU5EUTc4ZyIsInN1YiI6ImJvYiIsImF1ZG$'❸ \
  --data 'claim_token_format=http://openid.net/specs/openid-connect-core-1_0.html#IDToken' \
  https://openam.example.com:8443/openam/oauth2/access_token
{
  "ticket": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJhdWQiOiJodHRwOi8vb3BlbmFtLmRlZmF1bHQzXhhbXBsZS5jb206ODAvb3BlbmFtL29hdXRoMiIsImZyI6Imh0dHA6Ly9vcGVuYW0uZGVmYXVsdC5leGfTcGxLmNvbTo4MCM9vcGVuYW0vb2F1dGgyIiwiaXQiOiJEsImV4cCI6MTUwNzE0DEZNSwidGlkiJoiNTkzNmExOGMtOGE2OC00YTIxLTLiOGQtZWViNzIyOGEmMDY3MCIzImZvcmdlcm9jayI6eyJzaWciOiIq2pMUEVVM3hYm5RbU8-YFXZ0d9aTj10jNnLlRrc2FKI3UpXnokIn19.a9JpfsXS09CNvr2BRtLD0-t4ZCnmrHr2h3auXPeJi3E",
  "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 Authorize using the credentials of the UMA client. Use the username and password, separated by a colon character, for example `UmaClient:password`, and base64-encode the result. See "To Create a UMA Client Agent".
- ❷ Specify the permission ticket acquired earlier. See "Create a Permission Ticket".
- ❸ Specify the claim token acquired earlier. See "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:

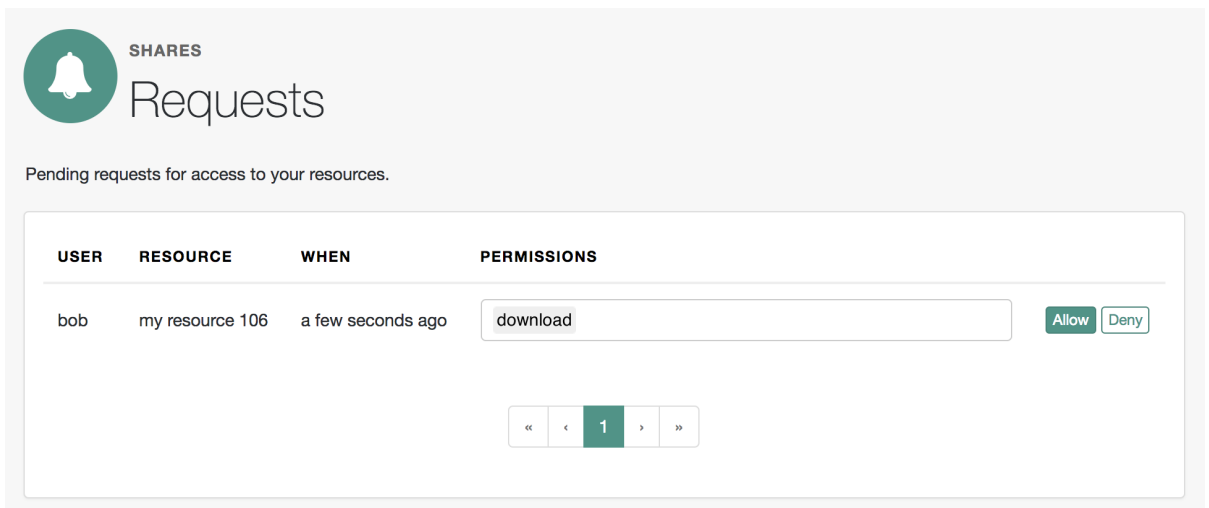
```
{
  "error_description": "The provided access grant is invalid, expired, or revoked.", "error":
  "invalid_grant"
}
```

If the ticket has expired, obtain another by repeating the steps in "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.

- The resource owner, Alice, logs into AM to view the access request. She clicks Shares > Requests, and clicks Allow to grant read 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 "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
\
--header 'authorization: Basic VW1hQ2xpZW50nBhc3N3b3Jk' ❶ \
--header 'cache-control: no-cache'
\
--header 'content-type: application/x-www-form-urlencoded'
\
--data 'grant_type=urn:ietf:params:oauth:grant-type:uma-ticket'
\
--data 'ticket=eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJhdWQiOiJodHRwOi8vb3BlbmFt
LmRlZmF1bHQyZG90b206ODAvb3BlbmFtL29hdXRoMiIsImZcyI6Imh0d
HA6Ly9vcGVuYW0uZGVmYXVsdC5leG$' ❷
--data 'scope=download'
\
--data 'claim_token=eyJ0eXAiOiJKV1QiLCJraWQiOiJiL082T3ZwdjEreStXZ3JINVVpOVdUaw9Mdda
9IiwiYXNjIjoiaWMyNTYifQ.eyJhdF9oYXNoIjoieWExdHlyZFdBTTlyWwG1fLU5
EUTc4ZyIsInN1YiI6ImJvYiIsImF1ZG$' ❸
--data 'claim_token_format=http://openid.net/specs/openid-connect-core-1_0.html#IDToken' \
https://openam.example.com:8443/openam/oauth2/access_token
{
  "access_token": "Aw4a92ZoKsjadWKw2d4Rmcjv7DM",
  "token_type": "Bearer",
  "expires_in": 3599
}
    
```

- ❶ Authorize using the same credentials of the UMA client as the first request for an RPT.
- ❷ 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 "Create a Permission Ticket".
- ❸ Specify the same claim token as the first request for an RPT.

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

You have successfully configured and tested an UMA 2.0 example demo deployment.

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/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"  
}
```

2.5. Additional Functionality for UMA End Users

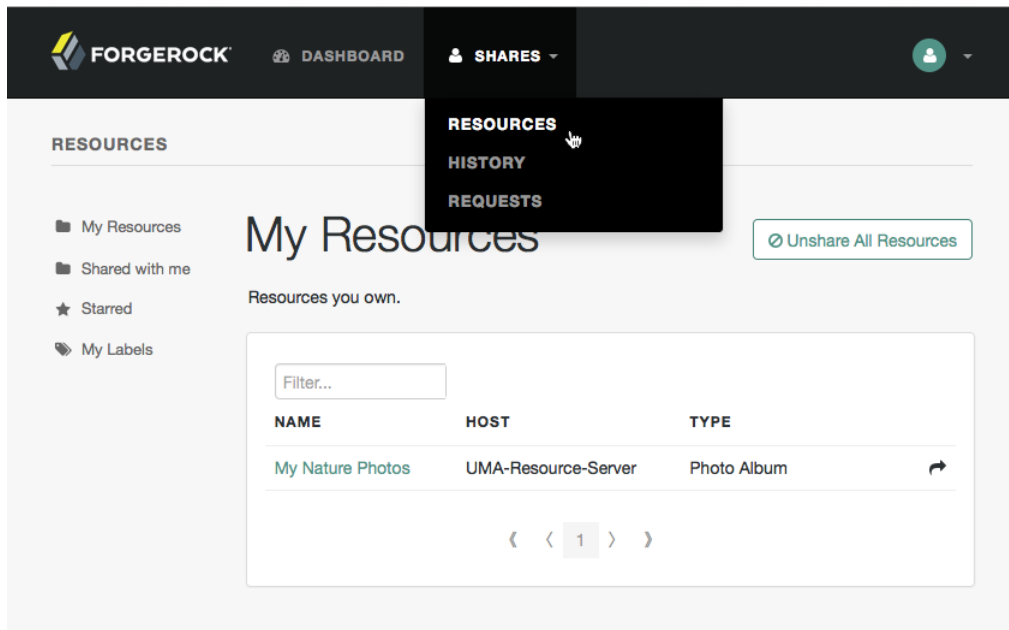
The functionality covered is described in the following procedures:

- "To Share UMA Resources"
- "To Apply User Labels to a Resource"
- "To Mark a Resource as a Favorite"
- "To View and Manage Pending Access Requests"

To Share UMA Resources

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

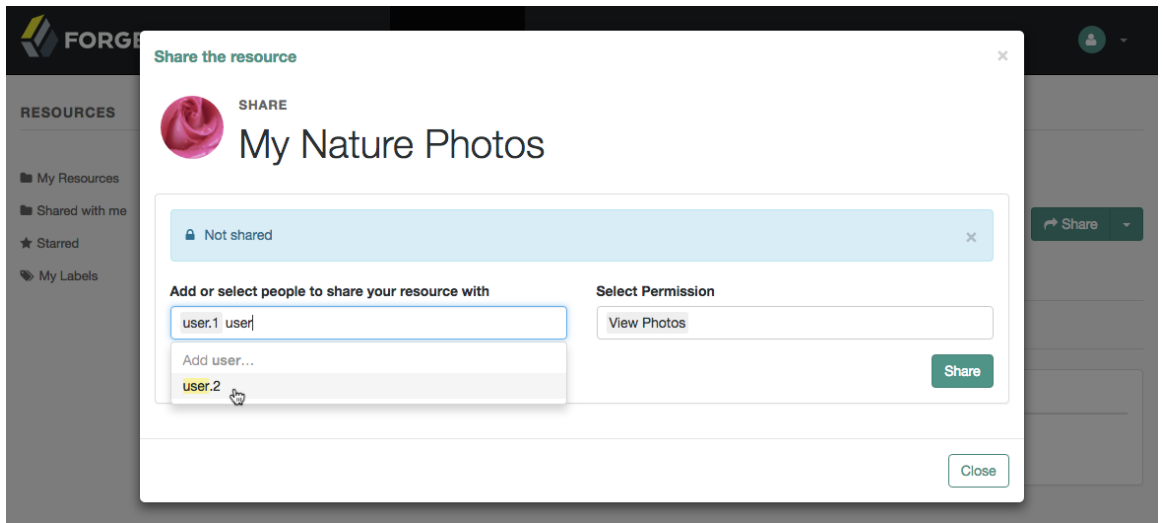


3. To share a resource, click the name of the resource to open the resource details page, and then click the Share button.

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.

4. When finished, click Close.

To Apply User Labels to a Resource

To apply labels to a resource:

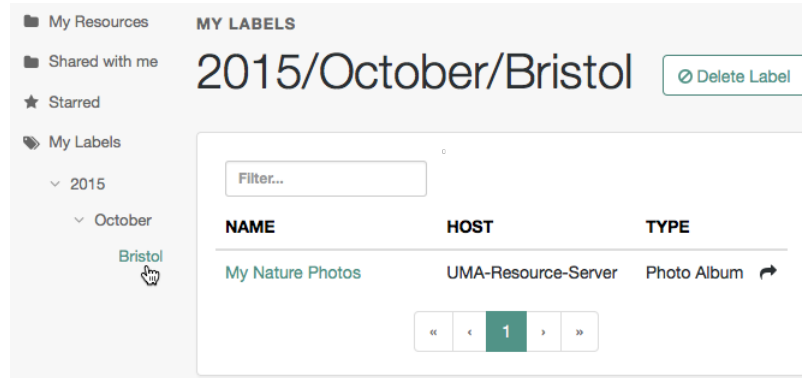
1. Log in to AM as a user. The profile page is displayed.
2. Navigate to Shares > Resources > My Resources, and then click the name of the resource to add labels to.
3. On the resource details page, click Edit Labels.

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`:

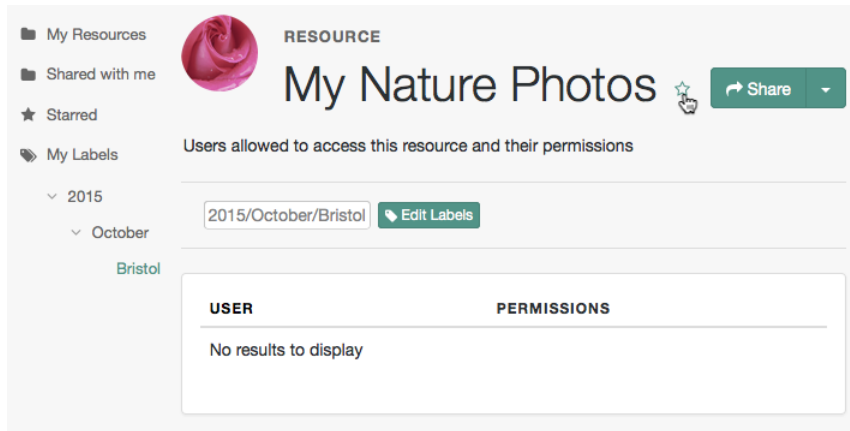


- Click an existing label, and then press **Delete** or **Backspace** to delete the label from the resource.
4. When you have finished editing labels you can:
- Click the checkmark button to save any changes made.
 - Click the X button to cancel any changes made.

To Mark a Resource as a Favorite

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

1. Log in to AM as a user. The profile page is displayed.
2. Navigate 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.

To View and Manage Pending Access Requests

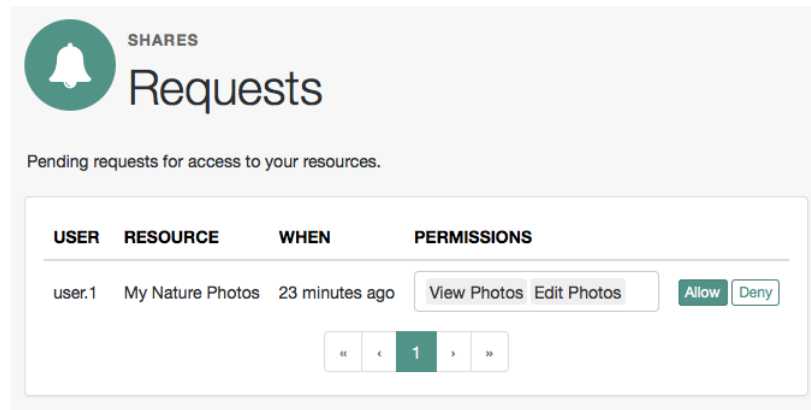
AM supports an UMA workflow in which a user can request access to a resource that has not been explicitly shared with them. The resource owner receives a notification of the request and can choose to allow or deny access.

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

1. Login to AM as the resource owner, and then navigate 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, and 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, navigate to Shares > History.

Chapter 3

Using UMA 2.0

This chapter shows how to use UMA 2.0 features that AM provides for administrators and developers.

3.1. Discovering UMA Configuration

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

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.

To use the endpoint, you must first create both an OAuth 2.0 Provider service, and an UMA Provider service in AM. For more information on creating these services, see "Configuring the OAuth 2.0 Authorization Service" in the *OAuth 2.0 Guide* and "To Create the UMA Provider Service".

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: `https://www.example.com:8080/openam/uma/realms/root/realms/subrealm/.well-known/uma2-configuration`.

Note

AM supports a provider service that allows a realm to 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 *OpenID Connect 1.0 Guide*.

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/.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/connect/jwk_uri",
"dynamic_client_endpoint": "https://openam.example.com:8443/openam/oauth2/realms/root/register",
"token_endpoint": "https://openam.example.com:8443/openam/oauth2/realms/root/access_token",
"authorization_endpoint": "https://openam.example.com:8443/openam/oauth2/realms/root/authorize",
"revocation_endpoint": "https://openam.example.com:8443/openam/oauth2/realms/root/token/revoke",
"introspection_endpoint": "https://openam.example.com:8443/openam/oauth2/realms/root/introspect",
"resource_registration_endpoint": "https://openam.example.com:8443/openam/uma/realms/root/resource_set",
"permission_endpoint": "https://openam.example.com:8443/openam/uma/realms/root/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 "OAuth 2.0 Client and Resource Server Endpoints" in the *OAuth 2.0 Guide*.

resource_registration_endpoint

The URI for a resource server to register a resource set.

For more information, see "Managing UMA Resource Sets".

dynamic_client_endpoint

The URI for registering a dynamic client.

3.2. Configuring UMA Stores

AM stores information about registered resource sets, audit information generated when users manage access to their protected resources, pending requests, and resource set labels. AM provides a default store, which uses the embedded data store, or you can configure external stores to maintain this information.

Tip

If you cannot find the attribute you are looking for, click on the dropdown button on the left-hand side of the tabs or use the Search box. For more information, see "AM Console Responsiveness" in the *Setup and Maintenance Guide* and "The AM Console Search Feature" in the *Setup and Maintenance Guide*.

To Configure the External UMA Resource Sets Store

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

1. Log in to the AM console as an administrator, for example `amadmin`.
2. Navigate to Deployment > Servers > *Server Name* > UMA > Resource Sets Store.
 - Unlock the Store Mode property and choose External Token Store.
 - Unlock the Root Suffix property and enter the base DN of the store. For example `dc=uma-rs,dc=example,dc=com`.
 - Save your work.
3. Navigate to Deployment > Servers > *Server Name* > UMA > External Resource Sets Store Configuration.

- Enter the properties for the store. For information about the available settings, see "UMA Properties".
- Save your work.

To Configure the External 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. Log in to the AM console as an administrator, for example `amadmin`.
2. Navigate to Deployment > Servers > *Server Name* > UMA > UMA Audit Store.
 - Unlock the Store Mode property and choose External Token Store.
 - Unlock the Root Suffix property and enter the base DN of the store. For example `dc=uma-audit,dc=example,dc=com`.
 - Save your work.
3. Navigate to Deployment > Servers > *Server Name* > UMA > External UMA Audit Store Configuration.
 - Enter the properties for the store. For information about the available settings, see "UMA Properties".
 - Save your work.

To Configure the External 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. Navigate to Deployment > Servers > *Server Name* > UMA > Pending Requests Store.
 - Unlock the Store Mode property and choose External Token Store.
 - Unlock the Root Suffix property and enter the base DN of the store. For example `dc=uma-pending,dc=example,dc=com`.
 - Save your work.
2. Navigate to Deployment > Servers > *Server Name* > UMA > External Pending Requests Store Configuration.
 - Enter the properties for the store. For information about the available settings, see "UMA Properties".

- Save your work.

To Configure the External UMA Resource Sets Labels Store

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

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

Navigate to Deployment > Servers > *Server Name* > UMA > UMA Resource Set Labels Store.

- Unlock the Store Mode property and choose External Token Store.
- Unlock the Root Suffix property and enter the base DN of the store. For example `dc=uma-labels,dc=example,dc=com`.
- Save your work.

2. Navigate to Deployment > Servers > *Server Name* > UMA > External Resource Set Labels Store Configuration.

- Enter the properties for the store. For information about the available settings, see "UMA Properties".
- Save your work.

3.3. Managing UMA Resource Sets

UMA resource servers register resource sets 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 resource sets to help resource owners organize their resources. For information on labelling resources, see "Managing UMA Labels".

AM provides the `/uma/resource_set` REST endpoint, as documented in the *OAuth 2.0 Resource Set Registration* specification, to allow UMA resource servers to register and manage resource sets.

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. For more information, see "To Acquire a Protection API Token (PAT)".

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

- "To Register an UMA Resource Set"
- "To List Registered UMA Resource Sets"

- "To Read an UMA Resource Set"
- "To Update an UMA Resource Set"
- "To Delete an UMA Resource Set"

To Register an UMA Resource Set

To register a resource set, the resource server must first acquire a PAT token on behalf of the resource owner, as described in "To Acquire a Protection API Token (PAT)".

Once you have the PAT bearer token, you can access the `/uma/resource_set` endpoint to register resources, as shown in the following steps.

- 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 set 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" \
--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 Resource Sets

To list the resource sets registered to a user, you must first acquire a PAT token as that user, as described in "To Acquire a Protection API Token (PAT)".

Once you have the PAT token, you can access the `/uma/resource_set` endpoint to list resource sets, as shown below:

- 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 resource sets 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",
  "3a2fe6d5-67c8-4a5a-83fb-09734f1dd5b10",
  "8ed24623-fcb5-46b8-9a64-18ee1b9b7d5d0"
}
```

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

- "To Read an UMA Resource Set"
- "To Update an UMA Resource Set"
- "To Delete an UMA Resource Set"

To Read an UMA Resource Set

To read a resource set, you must first acquire a PAT token on behalf of the resource owner, as described in "To Acquire a Protection API Token (PAT)".

Once you have the PAT token, you can access the `/uma/resource_set` endpoint to read resources, as shown below:

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

Note

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

The following example uses a PAT bearer token and a resource set ID to read a specific resource set 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, an HTTP 200 OK status code is returned, as well as a representation of the resource set in the JSON body of the response.

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

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

To Update an UMA Resource Set

To read a resource set, you must first acquire a PAT token on behalf of the resource owner, as described in "To Acquire a Protection API Token (PAT)".

Once you have the PAT token, you can access the `/uma/resource_set` endpoint to update resources, as shown below:

- Create a PUT request to the `/uma/resource_set` endpoint, including the PAT bearer token in a header named `Authorization`, and any new or changed parameters as part of a complete replacement of the existing values.

Note

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

The following example uses a PAT bearer token and a resource set ID to update a specific resource set in a realm named `subrealm` with an additional `delete` resource scope:

```
$ curl \
--request PUT
\
--header "Authorization: Bearer 515d6551-6512-5279-98b6-c0ef3f03a723"
\
--data \
'{
  "name" : "Photo Album",
  "icon_uri" : "http://photoz.example.com/icons/flower.png",
  "resource_scopes" : [
    "delete",
    "edit",
    "view",
    "http://photoz.example.com/dev/scopes/print"
  ],
  "labels" : [
    "3D",
    "VIP"
  ],
  "type" : "http://photoz.example.com/dev/rsets/photoalbum"
}' \
https://openam.example.com:8443/openam/uma/realms/root/resource_set/126615ba-b7fd-4660-b281-
bae81aa45f7c0
{
  "_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"
}
```

On success, an HTTP 200 OK status code is returned, with the resource set ID, and a user access policy URI that the resource owner can visit in order to manage access to the resource set.

If the resource set ID is not found, an HTTP 404 Not Found status code is returned, as follows:

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

To Delete an UMA Resource Set

To delete a resource set, you must first acquire a PAT token, as described in "To Acquire a Protection API Token (PAT)".

Once you have the PAT token, you can access the `/uma/resource_set` endpoint to delete resources, as shown below:

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

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

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

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

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

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

3.4. 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 label:

User Labels

- Managed by the resource owner after the resource set 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 "UMA Labels Endpoint for Users" and "To Apply User Labels to a Resource".

System Labels

- Created by the resource server when registering a resource set.

- Cannot be deleted.
- Do not support a hierarchy of levels.
- Are only visible to the owner of the resource.

Note

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

For information on creating system labels, see "To Register an UMA Resource Set".

Favourite Labels

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

For more information, see "To Mark a Resource as a Favorite".

3.4.1. UMA Labels Endpoint for Users

AM provides the `/json/users/username/oauth2/resources/labels` REST endpoint to allow users to manage user labels.

Specify the *username* in the URL, and provide the SSO token of that user in the `iPlanetDirectoryPro` header.

Use the `/json/users/username/oauth2/resources/labels` REST endpoint for the following operations:

- "To Create User Labels by Using REST"
- "To Query User Labels by Using REST"
- "To Delete User Labels by using REST"

To Create User Labels by Using REST

- 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 Set 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"
}
```

On success, an HTTP 201 Created status code is returned, 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 set. To apply the new label to a resource set, see "To Update an UMA Resource Set".

To Query User Labels by Using REST

- 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*" \
https://openam.example.com:8443/json/realms/root/users/demo/oauth2/resources/labels?_queryFilter=true
{
  "result": [
    {
      "_id": "46a3392f-1d2f-4643-953f-d51ecdf141d44",
      "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 by using REST

- 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*" \
https://openam.example.com:8443/json/users/demo/oauth2/resources/labels/46a3392f-1d2f-4643-953f-d51ecdff141d44
{
  "_id": "46a3392f-1d2f-4643-953f-d51ecdff141d44",
  "name": "2015/October/Bristol",
  "type": "USER"
}

```

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

3.5. Managing UMA 2.0 Policies

UMA 2.0 authorization servers must manage the resource owner's authorization policies, so that registered resource sets can be protected.

AM provides the `/json/users/{user}/uma/policies/` REST endpoint for creating and managing user-managed authorization policies.

Managing UMA policies requires that a resource set is registered to the user in the URL. For information on registering resource sets, see "Managing UMA Resource Sets".

Once a resource set is registered to the user, use the `/json/users/{user}/uma/policies/` REST endpoint for managing UMA resource policies.

To create an UMA policy, see "To Create an UMA Policy".

To Read an UMA Policy

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 set it protects.

- Create a GET request to the policies endpoint, including the SSO token in a header based on the configured session cookie name (default: `iPlanetDirectoryPro`), and the resource set 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*" \
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, an HTTP 200 OK status code is returned, 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

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 set it protects.

- Create a PUT request to the policies endpoint, including the SSO token in a header based on the configured session cookie name (default: `iPlanetDirectoryPro`), and the resource set 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, an HTTP 200 OK status code is returned, 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, an HTTP 400 Bad Request status code is returned, 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, an HTTP 400 Bad Request status code is returned, for example:

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

To Delete an UMA Policy

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 set it protects.

- Create a DELETE request to the policies endpoint, including the SSO token in a header based on the configured session cookie name (default: `iPlanetDirectoryPro`), and the resource set 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, an HTTP 200 OK status code is returned, 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

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.

- Create a GET request to the 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 set.
- `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, an HTTP 200 OK status code is returned, 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`, an HTTP 500 Server Error is returned, as follows:

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

Chapter 4

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

4.1. Resource Set Registration Extension Point

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

Resource Set Registration Extension Methods

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

4.2. 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.

4.3. 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>)	

4.4. 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 be</i> 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 be</i> 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 be</i> 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 resource sets 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 resource sets.</p>

Chapter 5

Reference

This reference section covers supported standards, settings and other information related to UMA.

5.1. UMA Supported Standards

This section covers information related to UMA support in AM:

User-Managed Access (UMA) 2.0

User-Managed Access (UMA) 2.0 is a protocol comprised of two specifications:

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

Federated Authorization for User-Managed Access (UMA) 2.0

5.2. UMA Configuration Reference

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

- To configure UMA global settings, navigate to Configure > Global Settings > UMA Provider. For more information, see "UMA Provider".
- To configure UMA data store settings:
 - Navigate to Configure > Server Defaults > UMA to configure the settings for all your servers.
 - Navigate to Deployment > Servers > *Server Name* > UMA to configure the settings for one server.

For more information, see "UMA Properties".

5.2.1. UMA Provider

amster service name: `uma`

5.2.1.1. Realm Defaults

The following settings appear on the **Realm Defaults** tab:

Permission Ticket Lifetime (seconds)

The maximum life of a permission ticket before it expires, in seconds.

Default value: 120

amster attribute: `permissionTicketLifetime`

Delete user policies when Resource Server is removed

Delete all user policies that relate to a Resource Server when removing the OAuth2 agent entry or removing the `uma_protection` scope from the OAuth2 agent.

Default value: `true`

amster attribute: `deletePoliciesOnDeleteRS`

Delete resource sets when Resource Server is removed

Delete all resource sets that relate to a Resource Server when removing the OAuth2 agent entry or removing the `uma_protection` scope from the OAuth2 agent.

Default value: `true`

amster attribute: `deleteResourceSetsOnDeleteRS`

Pending Requests Enabled

Specifies whether to use the Pending Requests subsystem that notifies the resource owner that an attempt to access a resource was made.

Default value: `true`

amster attribute: `pendingRequestsEnabled`

Email Resource Owner on Pending Request creation

Specifies whether to send an email to the Resource Owner when a Pending Request is created when a Requesting Party requests authorization to a resource.

Default value: `true`

amster attribute: `emailResourceOwnerOnPendingRequestCreation`

Email Requesting Party on Pending Request approval

Specifies whether to send an email to the Requesting Party when a Pending Request is approved by the Resource Owner.

Default value: `true`

amster attribute: `emailRequestingPartyOnPendingRequestApproval`

User profile preferred Locale attribute

User profile attribute storing the user's preferred locale.

Default value: `inetOrgPerson`

amster attribute: `userProfileLocaleAttribute`

Re-Sharing Mode

Specifies whether re-sharing is off or on implicitly for all users, allowing all users to re-share resource sets that have been shared with them.

The possible values for this property are:

- `OFF`
- `IMPLICIT`

Default value: `IMPLICIT`

amster attribute: `resharingMode`

Grant RPTs...

In UMA, scope comes from both the permission ticket and from the token request. An RPT is always granted when all scope matches, and is never granted when no scope matches. You can configure when RPTs are granted for partial match conditions here. For more information, see the UMA Grant Type specification section on Authorization Assessment and Results Determination.

Default value:

```
REQUEST_PARTIAL
REQUEST_NONE
TICKET_PARTIAL
```

amster attribute: `grantRptConditions`

5.2.2. UMA Properties

UMA server settings are inherited by default.

5.2.2.1. Resource Sets Store

The following settings appear on the Resource Sets 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 embedded 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-rs,dc=forgerock,dc=com`.

Max Connections

Specifies the maximum number of connections to the data store.

5.2.2.2. External Resource Sets 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. 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`

5.2.2.3. 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 embedded 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-rs,dc=forgerock,dc=com`.

Max Connections

Specifies the maximum number of connections to the data store.

5.2.2.4. 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. 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

5.2.2.5. 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 embedded 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-rs,dc=forgerock,dc=com`.

Max Connections

Specifies the maximum number of connections to the data store.

5.2.2.6. 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. 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`

5.2.2.7. UMA Resource Set Labels Store

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

Store Mode

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

- `Default Token Store`: AM stores user-created labels in the embedded 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-rs,dc=forgerock,dc=com`.

Max Connections

Specifies the maximum number of connections to the data store.

5.2.2.8. External Resource Set Labels Store Configuration

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

Appendix A. Getting Support

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

A.1. Accessing Documentation Online

ForgeRock publishes comprehensive documentation online:

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

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

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

A.2. Using the ForgeRock.org Site

The [ForgeRock.org](#) site has links to source code for ForgeRock open source software, as well as links to the ForgeRock forums and technical blogs.

If you are a *ForgeRock customer*, raise a support ticket instead of using the forums. ForgeRock support professionals will get in touch to help you.

A.3. Getting Support and Contacting ForgeRock

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

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

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 sessions	AM sessions for which AM returns session state to the client after each request, and require it to be passed in with the subsequent request. For browser-based clients, AM sets a cookie in the browser that contains the session information.

	<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 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 user data stores, and when different organizations use the same AM deployment.

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

Resource	Something a user can access over the network such as a web page. Defined as part of policies, these can include wildcards in order to match multiple actual resources.
Resource owner	In OAuth 2.0, entity who can authorize access to protected web resources, such as an end user.
Resource server	In OAuth 2.0, server hosting protected web resources, capable of handling access tokens to respond to requests for such resources.
Response attributes	Defined as part of policies, these allow AM to return additional information in the form of "attributes" with the response to a policy decision.
Role based access control (RBAC)	Access control that is based on whether a user has been granted a set of permissions (a role).
Security Assertion Markup Language (SAML)	Standard, XML-based language for exchanging authentication and authorization data between identity providers and service providers.
Service provider (SP)	Entity that consumes assertions about a principal (and provides a service that the principal is trying to access).
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>
User data 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.