User-Managed Access (UMA) Guide

ForgeRock Access Management 5

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

Abstract

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

Table of Contents

Preface

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

OpenAM 13 added support for the UMA 1.0.1 specification.

Note

Support for UMA 1.0.1 is deprecated in this release, and will be removed in a future version of ForgeRock Access Management. Features and functionality will be upgraded to support upcoming UMA standards.

For more information on deprecation, see Appendix A, "*Release Levels and Interface Stability*" in the *Release Notes*.

This guide is written for anyone who wants to set up ForgeRock Access Management for user-managed access features.

About ForgeRock Identity Platform™ Software

ForgeRock Identity Platform™ is the only offering for access management, identity management, user-managed access, directory services, and an identity gateway, designed and built as a single, unified platform.

The platform includes the following components that extend what is available in open source projects to provide fully featured, enterprise-ready software:

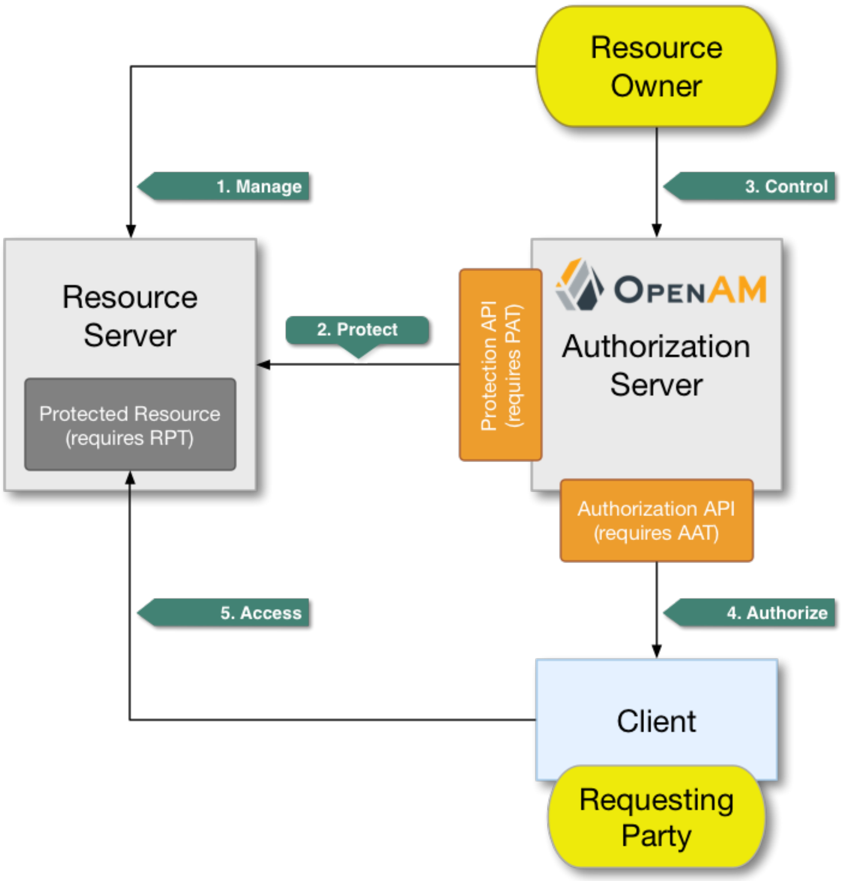
- ForgeRock Access Management (AM)
- ForgeRock Identity Management (IDM)
- ForgeRock Directory Services (DS)
- ForgeRock Identity Gateway (IG)

Chapter 1
Introducing UMA

This chapter provides with an overview of what is UMA and how it integrates with OpenAM.

UMA defines a workflow for allowing resource owners to manage access to their protected resources by creating authorization policies on a centralized authorization server, such as OpenAM.

Figure 1.1. The Role in the UMA Workflow



The actions that form the UMA workflow are as follows:

1. Manage

The resource owner manages their resources on the resource server.

2. Protect

The resource owner links their resource server and chosen authorization server, for example an OpenAM instance.

The authorization server provides a protection API so that the resource server can register sets of resources. Use of the protection API requires a protection API token (PAT) - an OAuth 2.0 token with a specific scope.

For more information, see Section 2.3, "Managing UMA Resource Sets".

3. Control

The resource owner controls who has access to their registered resources by creating policies on the authorization server.

For more information, see Section 2.5, "Managing UMA Policies".

4. Authorize

The client, acting on behalf of the requesting party, uses the authorization server's authorization API to acquire a requesting party token (RPT). The requesting party or client may need further interaction with the authorization server at this point, for example to supply identity claims. Use of the authorization API requires an authorization API token (AAT) - an OAuth 2.0 token with a specific scope.

For more information, see Section 3.2, "Accessing UMA Protected Resources".

5. Access

The client presents the RPT to the resource server, which verifies its validity with the authorization server and, if both valid and containing sufficient permissions, returns the protected resource to the requesting party.

For more information, see Section 3.2, "Accessing UMA Protected Resources".

Chapter 2
Implementing UMA

This chapter explains how to set up OpenAM as an authorization server in the UMA workflow.

To configure OpenAM as part of the UMA workflow perform the following steps:

1. Configure an UMA provider service and the UMA stores using the AM console:
    - Section 2.1, "Configuring the UMA Provider Service", to configure a UMA provider service.
    - Section 2.2, "Configuring UMA Stores", to configure the UMA stores.
2. Register resource sets, apply labels to resources, and manage UMA policies using REST APIs:
    - Section 2.3, "Managing UMA Resource Sets", to register a resource set to be protected by OpenAM.
    - Section 2.4, "Managing UMA Labels", to apply labels to the resources and locate them easily.
    - Section 2.5, "Managing UMA Policies", to manage policies to protect your resources.

2.1. Configuring the UMA Provider Service

To enable OpenAM to act as an authorization server in the UMA workflow, you must create an UMA Provider service.

Procedure 2.1. To Configure the UMA Provider Service

1. In the AM console, select Realms > *`Realm Name`* > Dashboard > Configure OAuth Provider > Configure User Managed Access.
2. On the Configure UMA page, select the Realm for the provider service.
3. If necessary, adjust the lifetimes for authorization codes, access tokens, and refresh tokens.
4. Select Issue Refresh Tokens unless you do not want the authorization service to supply a refresh token when returning an access token.
5. 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.

6. If you have a custom scope validator implementation, put it on the OpenAM 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 Section 4.1, "Customizing OAuth 2.0 Scope Handling" in the *OAuth 2.0 Guide*.

7. Click Create to save your changes. OpenAM creates the following:

   - An UMA provider service.
   - An OAuth2 provider service that supports OpenID Connect.
   - A policy to protect the OAuth2 authorization endpoints.

   Warning

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

8. 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 Section 5.2.1, "UMA Provider".

   To complete the configuration, click Save Changes.

2.2. Configuring UMA Stores

OpenAM stores information about registered resource sets, and also audit information generated when users manage access to their protected resources. OpenAM provides a default 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 Section 1.1.1, "AM Console Responsiveness" in the *Setup and Maintenance Guide* and Section 1.1.2, "The AM Console Search Feature" in the *Setup and Maintenance Guide*.

Procedure 2.2. To Configure the UMA Resource Sets Store

Resource Sets Store properties are inherited from the defaults. For more information about inherited properties, see Section 2.3.1, "Configuring Servers" in the *Reference*

1. Log in to the AM console as an OpenAM 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 Section 5.2.2, "UMA Properties".
   - Save your work.

Procedure 2.3. To Configure UMA Audit Storage

UMA Audit Store properties are inherited from the defaults. For more information about inherited properties, see Section 2.3.1, "Configuring Servers" in the *Reference*

1. Log in to the AM console as an OpenAM 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-rs,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 Section 5.2.2, "UMA Properties".
   - Save your work.

2.3. 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.

OpenAM supports optional *system* labels when registering resource sets to help resource owners organize their resources. For information on labelling resources, see Section 2.4, "Managing UMA Labels".

OpenAM provides two REST endpoints for managing resource sets, as described in the sections below:

- Section 2.3.1, "UMA Resource Set Endpoint for Resource Servers"
- Section 2.3.2, "UMA Resource Set Endpoint for Users"

2.3.1. UMA Resource Set Endpoint for Resource Servers

OpenAM provides the `/oauth2/resource_set` REST endpoint, as described 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 Procedure 2.4, "To Acquire a Protection API Token".

After acquiring a PAT, use the `/oauth2/resource_set` REST endpoint for the following operations:

- Procedure 2.5, "To Register an UMA Resource Set"
- Procedure 2.6, "To List Registered UMA Resource Sets"
- Procedure 2.7, "To Read an UMA Resource Set"
- Procedure 2.8, "To Update an UMA Resource Set"
- Procedure 2.9, "To Delete an UMA Resource Set"

Procedure 2.4. To Acquire a Protection API Token

You must have first registered 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 Section 2.2, "Registering OAuth 2.0 Clients With the Authorization Service" in the *OAuth 2.0 Guide*.

After a suitable OAuth 2.0 client is configured, perform the following steps to acquire a PAT:

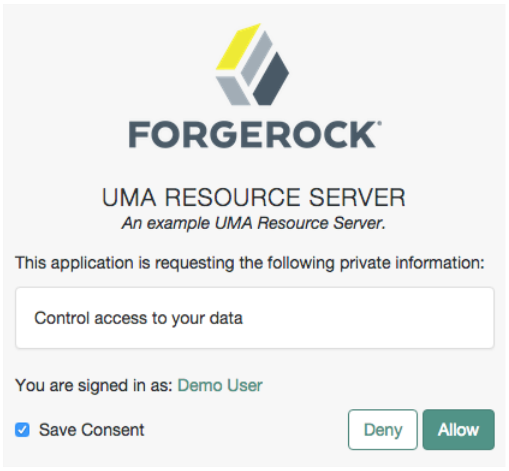
1. Direct the resource owner to the authorization server to obtain a PAT. The URL should specify the client name registered above, the redirect URI, and request the `uma_protection` scope, as shown in the example below:

   ```
   https://openam.example.com:8443/openam/oauth2/authorize?client_id=UMA-Resource-
   Server&redirect_uri=http://openam.example.com:8080&response_type=code&scope=uma_protection
   ```

   This example uses the OAuth 2.0 code grant, however the UMA resource server can use any of the OAuth 2.0 grants to obtain the access token.

2. After logging in, the consent screen asks the resource owner to allow or deny the requested scopes.

Figure 2.1. Consent Screen Presented to the Requesting Party


FORGEROCK
UMA RESOURCE SERVER
An example UMA Resource Server.
This application is requesting the following private information:
Control access to your data
You are signed in as: Demo User
Save Consent
Deny
Allow

3. If the resource owner allows access, they are sent to the configured redirection URL, which will have a `code` query string parameter added, which is used to request the PAT.

4. Create a POST request to the `/oauth2/access_token` endpoint, with the client credentials registered earlier, a grant type of `authorization_code`, a redirect URL, and the value of the `code` query string parameter returned in the previous step, as shown below:

```
$ curl \
 --request POST \
 --data 'client_id=UMA-Resource-Server' \
 --data 'client_secret=password' \
 --data 'grant_type=authorization_code' \
 --data 'code=c1bb2b94-038b-4ab2-beb1-a1ee14790c6b' \
 --data 'redirect_uri=http%3A%2F%2Fopenam.example.com%3A8080' \
 http://openam.example.com:8080/openam/oauth2/realms/root/access_token

{
 "scope": "uma_protection read",
 "expires_in": 599,
 "token_type": "Bearer",
 "refresh_token": "f9873041-885a-4522-836c-9fa71aaad3e4",
 "access_token": "983e1d96-20a7-437c-8432-cfde52076714"
}
```

The value returned in `access_token` is the PAT bearer token, used in the following procedures.

Procedure 2.5. To Register an UMA Resource Set

To register a resource set, the resource server must first acquire a PAT token, as described in Procedure 2.4, "To Acquire a Protection API Token".

Once you have the PAT bearer token, you can access the `/oauth2/resource_set` endpoint to register resources, as shown in the following steps.

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

  The following example uses a PAT bearer token to register a photo album resource set and a pair of system labels:

```
$ curl \
 --request POST \
 --header "Content-Type: application/json" \
 --header "Authorization: Bearer 515d6551-6512-5279-98b6-c0ef3f03a723" \
 --data \
 '{
   "name" : "Photo Album",
   "icon_uri" : "http://www.example.com/icons/flower.png",
   "scopes" : [
     "http://photoz.example.com/dev/scopes/view",
     "http://photoz.example.com/dev/scopes/all"
   ],
   "labels" : [
     "3D",
     "VIP"
   ],
  "type" : "http://www.example.com/rsets/photoalbum"
 }' \
 https://openam.example.com:8443/openam/oauth2/realms/root/resource_set/

{
 "_id": "43225628-4c5b-4206-b7cc-5164da81decd0",
 "user_access_policy_uri":
   "https://openam.example.com:8443/openam/XUI/#uma/share/43225628-4c5b-4206-b7cc-5164da81decd0/"
}
```

  The resource owner can then visit the user access policy URI in order to manage access to the resource set.

Procedure 2.6. To List Registered UMA Resource Sets

To list registered resource sets, you must first acquire a PAT token, as described in Procedure 2.4, "To Acquire a Protection API Token".

Once you have the PAT token, you can access the `/oauth2/resource_set` endpoint to list resource sets, as shown below:

- Create a GET request to the 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:

```
$ curl \
 --header "Authorization: Bearer 515d6551-6512-5279-98b6-c0ef3f03a723" \
 https://openam.example.com:8443/openam/oauth2/realms/root/resource_set

[
 "43225628-4c5b-4206-b7cc-5164da81decd0",
 "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:

- Procedure 2.7, "To Read an UMA Resource Set"
- Procedure 2.8, "To Update an UMA Resource Set"
- Procedure 2.9, "To Delete an UMA Resource Set"

Procedure 2.7. To Read an UMA Resource Set

To read a resource set, you must first acquire a PAT token, as described in Procedure 2.4, "To Acquire a Protection API Token".

Once you have the PAT token, you can access the `/oauth2/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/oauth2/realms/root/resource_set/`*`resource_set_ID`*.

  The following example uses a PAT bearer token and a resource set ID to read a specific resource set:

```
$ curl \
 --header "Authorization: Bearer 515d6551-6512-5279-98b6-c0ef3f03a723" \
 https://openam.example.com:8443/openam/oauth2/realms/root/resource_set/43225628-4c5b-4206-b7cc
-5164da81decd0

 {
  "scopes": [
    "http://photoz.example.com/dev/scopes/view",
    "http://photoz.example.com/dev/scopes/all"
  ],
  "_id": "43225628-4c5b-4206-b7cc-5164da81decd0",
  "name": "Photo Album",
  "icon_uri": "http://www.example.com/icons/flower.png",
  "type": "http://www.example.com/rsets/photoalbum",
  "user_access_policy_uri":
    "https://openam.example.com:8443/openam/XUI/#uma/share/43225628-4c5b-4206-b7cc-5164da81decd0"
}
```

On success, an HTTP 200 OK status code is returned, as well as a header containing the current ETag value, for example: `W/"123401234"`. Use this ETag value when updating a resource set. See Procedure 2.8, "To Update an UMA Resource Set".

Tip

Add the `-i` option to curl commands to show the returned headers. For example:

```
$ curl -i \
 --header "Authorization: Bearer 515d6551-4512-4279-98b6-c0ef3f03a722" \
https://openam.example.com:8443/openam/
oauth2\
/resource_set/43225628-4c5b-4206-b7cc-5164da81decd0
HTTP/1.1 200 OK
 ETag: W/"123401234"
 Date: Tue, 10 Feb 2015 11:57:35 GMT
 Accept-Ranges: bytes
 Server: Restlet-Framework/2.1.7
 Vary: Accept-Charset, Accept-Encoding, Accept-Language, Accept
 Content-Type: application/json;charset=UTF-8
 Transfer-Encoding: chunked

 {
  "scopes": [
    "http://photoz.example.com/dev/scopes/view",
    "http://photoz.example.com/dev/scopes/all"
  ],
  "_id": "myPhotoAlbum001",
  "name": "Photo Album",
  "icon_uri": "http://www.example.com/icons/flower.png",
  "type": "http://www.example.com/rsets/photoalbum",
  "user_access_policy_uri":
     "https://openam.example.com:8443/openam/XUI/#uma
           /share/43225628-4c5b-4206-b7cc-5164da81decd0"
 }
```

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"
}
```

Procedure 2.8. To Update an UMA Resource Set

To update a resource set, you must first acquire a PAT token, as described in Procedure 2.4, "To Acquire a Protection API Token".

Once you have the PAT token, you can access the `/oauth2/resource_set` endpoint to update resources, as shown below:

- Create a PUT request to the resource_set endpoint, including the PAT bearer token in a header named `Authorization`, and any new or changed parameters.

  The only difference between creating a resource set and updating one is the presence of an `If-Match` header when updating. This should contain the value of the ETag header returned when creating, updating, or reading a resource set.

> **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/oauth2/realms/root/resource_set/`*`resource_set_ID`*.

The following example uses a PAT bearer token, a resource set ID and an If-Match header to update a specific resource set:

```
$ curl \
 --request PUT \
 --header "Authorization: Bearer 515d6551-6512-5279-98b6-c0ef3f03a723" \
 --header "If-Match: "123401234"" \
 --data \
 '{
   "name" : "Photo Album 2.0",
   "icon_uri" : "http://www.example.com/icons/camera.png",
   "scopes" : [
     "http://photoz.example.com/dev/scopes/view",
     "http://photoz.example.com/dev/scopes/edit",
     "http://photoz.example.com/dev/scopes/all"
   ],
   "type" : "http://www.example.com/rsets/photoalbum"
 }' \
 https://openam.example.com:8443/openam/oauth2/realms/root/resource_set/43225628-4c5b-4206-b7cc
-5164da81decd0

 {
  "_id": "43225628-4c5b-4206-b7cc-5164da81decd0",
  "user_access_policy_uri":
     "https://openam.example.com:8443/openam/XUI/#uma/share/43225628-4c5b-4206-b7cc-5164da81decd0"
}
```

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"
}
```

If the `If-Match` header is missing, or does not match the current version of the resource set, an HTTP 412 Precondition Failed status code is returned, as follows:

```
{
 "error": "precondition_failed"
}
```

Procedure 2.9. To Delete an UMA Resource Set

To delete a resource set, you must first acquire a PAT token, as described in Procedure 2.4, "To Acquire a Protection API Token".

Once you have the PAT token, you can access the `/oauth2/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`.

  Add an `If-Match` header containing the value of the ETag header returned when creating, updating, or reading a resource set.

  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/oauth2/realms/root/resource_set/`*`resource_set_ID`*.

  The following example uses a PAT bearer token, a resource set ID and an If-Match header to delete a specific resource set:

```
$ curl \
 --request DELETE \
 --header "Authorization: Bearer 515d6551-6512-5279-98b6-c0ef3f03a723" \
 --header "If-Match: "123401234"" \
 https://openam.example.com:8443/openam/oauth2/realms/root/resource_set/43225628-4c5b-4206-b7cc
-5164da81decd0
  {}
```

  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"
}
```

  If the `If-Match` header is missing, or does not match the current version of the resource set, an HTTP 412 Precondition Failed status code is returned, as follows:

```
{
 "error": "precondition_failed"
}
```

2.3.2. UMA Resource Set Endpoint for Users

OpenAM provides the `/json/users/username/oauth2/resources/sets` REST endpoint for managing resource sets belonging to a user.

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

Procedure 2.10. To Manage Resource Sets for a User by using REST

1. To query resource sets for a user, create a GET request including `_queryFilter=resourceOwnerId eq "username"` in the query string. The query string should be URL-encoded, as shown below:

```
$ curl \
 --header "iPlanetDirectoryPro: AQIC5wM2LY4S...Q4MTE4NTA2*" \
 https://openam.example.com:8443/json/users/demo/oauth2/resources/sets?_queryFilter=resourceOwnerId+eq
+%22demo%22
{
 "result": [
   {
     "scopes": [
       "View Photos",
       "Edit Photos"
     ],
     "_id": "46a3392f-1d2f-4643-953f-d51ecdf141d47",
     "resourceServer": "UMA-Resource-Server",
     "labels": [],
     "name": "My Nature Photos",
     "icon_uri": "http://www.example.com/icons/flower.png",
     "resourceOwnerId": "demo",
     "type": "Photo Album"
   }
 ],
 "resultCount": 1,
 "pagedResultsCookie": null,
 "totalPagedResultsPolicy": "NONE",
 "totalPagedResults": -1,
 "remainingPagedResults": 0
}
```

   On success, an HTTP 200 OK status code is returned, as well as a JSON representation of the resource sets assigned to the specified user.

2. To read a specific resource set for a user, create a GET request including the ID of the resource set in the URL, as shown below:

```
$ curl \
 --header "iPlanetDirectoryPro: AQIC5wM2LY4S...Q4MTE4NTA2*" \
 https://openam.example.com:8443/json/users/demo/oauth2/resources/sets/46a3392f-1d2f-4643-953f-
d51ecdf141d47
 {
 "scopes": [
   "View Photos",
   "Edit Photos"
 ],
 "_id": "46a3392f-1d2f-4643-953f-d51ecdf141d47",
 "resourceServer": "UMA-Resource-Server",
 "labels": [],
 "name": "My Nature Photos",
 "icon_uri": "http://www.example.com/icons/flower.png",
 "resourceOwnerId": "demo",
 "type": "Photo Album"
}
```

On success, an HTTP 200 OK status code is returned, as well as a JSON representation of the specified resource set.

3. To update the user labels assigned to a resource set for a user, create a PUT request including the ID of the resource set in the URL, the full JSON representation of the resource set, and the additional user label IDs in the `labels` array in the body of the JSON data, as shown below:

```
$ curl \
 --header "iPlanetDirectoryPro: AQIC5wM2LY4S...Q4MTE4NTA2*" \
 --data \
 '{
   "scopes": [
     "View Photos",
     "Edit Photos"
   ],
   "_id": "46a3392f-1d2f-4643-953f-d51ecdf141d47",
   "resourceServer": "UMA-Resource-Server",
   "labels": ["257ee30a-b989-4fe6-9e70-a87a050f6a4a4"],
   "name": "My Nature Photos",
   "icon_uri": "http://www.example.com/icons/flower.png",
   "resourceOwnerId": "demo",
   "type": "Photo Album"
  }' \
  https://openam.example.com:8443/json/users/demo/oauth2/resources/sets/46a3392f-1d2f-4643-953f-
d51ecdf141d47
{
 "scopes": [
   "View Photos",
   "Edit Photos"
 ],
 "_id": "46a3392f-1d2f-4643-953f-d51ecdf141d47",
 "resourceServer": "UMA-Resource-Server",
 "labels": [
    "257ee30a-b989-4fe6-9e70-a87a050f6a4a4"
 ],
 "name": "My Nature Photos",
 "icon_uri": "http://www.example.com/icons/flower.png",
 "resourceOwnerId": "demo",
 "type": "Photo Album"
}
```

On success, an HTTP 200 OK status code is returned, as well as a JSON representation of the updated resource set.

> **Note**
>
> Only the `labels` field can be updated by using PUT. All other fields are read-only but must still be included in the JSON body of the request.

OpenAM also provides a read-only endpoint for viewing a user's resource sets, and if available policy definitions. For more information, see Section 3.3, "OAuth 2.0 Resource Set Endpoint" in the *OAuth 2.0 Guide*.

2.4. 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 Section 2.4.1, "UMA Labels Endpoint for Users" and Procedure 3.5, "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 Procedure 2.5, "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 Procedure 3.6, "To Mark a Resource as a Favorite".

2.4.1. UMA Labels Endpoint for Users

OpenAM 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:

- Procedure 2.11, "To Create User Labels by Using REST"
- Procedure 2.12, "To Query User Labels by Using REST"
- Procedure 2.13, "To Delete User Labels by using REST"

Procedure 2.11. 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 "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?
_action=create
{
 "_id": "db2161c0-167e-4195-a832-92b2f578c96e3",
 "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 Procedure 2.10, "To Manage Resource Sets for a User by using REST".

Procedure 2.12. 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
}
```

Procedure 2.13. 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-
d51ecdf141d44
{
 "_id": "46a3392f-1d2f-4643-953f-d51ecdf141d44",
 "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.

2.5. Managing UMA Policies

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

OpenAM 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 Section 2.3, "Managing UMA Resource Sets".

Once a resource set is registered to the user, use the `/json/users/{user}/uma/policies/` REST endpoint for the following operations:

- Procedure 2.14, "To Create an UMA Policy"
- Procedure 2.15, "To Read an UMA Policy"
- Procedure 2.16, "To Update an UMA Policy"
- Procedure 2.17, "To Delete an UMA Policy"
- Procedure 2.18, "To Query UMA Policies"

Procedure 2.14. 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.

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 POST 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.

  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 create a policy to share a resource set belonging to user *demo* with two subjects, with different scopes for each:

```
$ curl \
 --request POST \
 --header "Content-Type: application/json" \
 --header "iPlanetDirectoryPro: AQIC5wM2LY4S...Q4MTE4NTA2*" \
 --data \
 '{
   "policyId": "43225628-4c5b-4206-b7cc-5164da81decd0",
   "permissions":
  [
   {
    "subject": "user.1",
     "scopes": ["http://photoz.example.com/dev/scopes/view"]
   },
   {
     "subject": "user.2",
      "scopes": [
         "http://photoz.example.com/dev/scopes/view",
         "http://photoz.example.com/dev/scopes/all"
      ]
   }
  ]
 }' \
https://openam.example.com:8443/openam/json/realms/root/users/demo/uma/policies?_action=create
{}
```

  On success, an HTTP 201 Created status code is returned, with an empty JSON body as the response.

  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'."
}
```

Procedure 2.15. 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/43225628-4c5b-4206-b7cc-5164da81decd0
{
 "policyId": "43225628-4c5b-4206-b7cc-5164da81decd0",
 "name": "Photo Album",
 "permissions": [
    {
      "subject": "user.1",
      "scopes": [
         "http://photoz.example.com/dev/scopes/view"
      ]
    },
    {
      "subject": "user.2",
      "scopes": [
         "http://photoz.example.com/dev/scopes/view",
         "http://photoz.example.com/dev/scopes/all"
      ]
    }
 ]
}
```

  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"
}
```

Procedure 2.16. 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 `43225628-4c5b-4206-b7cc-5164da81decd0` belonging to user *`demo`* with a new scope for one of the subjects:

```
$ curl \
 --request PUT \
 --header "iPlanetDirectoryPro: AQIC5wM2LY4S...Q4MTE4NTA2*" \
 --data \
 '{
   "policyId": "43225628-4c5b-4206-b7cc-5164da81decd0",
   "permissions":
   [
     {
        "subject": "user.1",
        "scopes": [
           "http://photoz.example.com/dev/scopes/view",
           "http://photoz.example.com/dev/scopes/all"
        ]
     },
     {
        "subject": "user.2",
        "scopes": [
          "http://photoz.example.com/dev/scopes/view",
        "  http://photoz.example.com/dev/scopes/all"
        ]
     }
   ]
 }' \
 https://openam.example.com:8443/openam/json/realms/root/users/demo
\
/uma/policies/43225628-4c5b-4206-b7cc-5164da81decd0
{}
```

On success, an HTTP 204 Empty 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"
}
```

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."
}
```

Procedure 2.17. 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 `43225628-4c5b-4206-b7cc-5164da81decd0` belonging to user *demo*:

  ```
  $ curl \
   --request DELETE \
   --header "iPlanetDirectoryPro: AQIC5wM2LY4S...Q4MTE4NTA2*" \
   https://openam.example.com:8443/openam/json/realms/root/realms/root/users/demo\
   /uma/policies/43225628-4c5b-4206-b7cc-5164da81decd0
  {}
  ```

  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"
  }
  ```

Procedure 2.18. 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 a using 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 "user.1"
> ```

When URL-encoded becomes:

`resourceServer+eq+%22UMA-Resource-Server%22+AND+permissions%2Fsubject+eq+%22user.1%22`

The following example uses an SSO token to query the policies belonging to user *demo*, which have a subject `user.1` in the permissions:

```
$ curl \
 --header "iPlanetDirectoryPro: AQIC5wM2LY4S...Q4MTE4NTA2*" \
 --get \
 --data-urlencode '_sortKeys=policyId,name' \
 --data-urlencode '_pageSize=1' \
 --data-urlencode '_pagedResultsOffset=0' \
 --data-urlencode \
 '_queryFilter=permissions/subject eq "user.1"' \
 https://openam.example.com:8443/openam/json/realms/root/users/demo/uma/policies
{
 "result": [
   {
      "policyId": "52645907-e20b-4351-8e0c-523ebe0d44710",
      "name": "Photo Album",
      "permissions": [
         {
             "subject": "user.1",
             "scopes": [
                 "http://photoz.example.com/dev/scopes/view"
             ]
         },
         {
              "subject": "user.2",
              "scopes": [
                "http://photoz.example.com/dev/scopes/all",
                "http://photoz.example.com/dev/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 3
Using UMA

Follow this chapter for information about using the UMA features that OpenAM provides for administrators, developers, and end users.

3.1. Discovering UMA Configuration

OpenAM 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/realms/root/.well-known/uma-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 OpenAM. For more information on creating these services, see Section 2.1, "Configuring the OAuth 2.0 Authorization Service" in the *OAuth 2.0 Guide* and Section 2.1, "Configuring 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 `http://www.example.com/.well-known/uma-configuration`, to the well-known URIs in OpenAM's space: `http://www.example.com/openam/uma/realms/root/realms/subrealm/.well-known/uma-configuration`.

Note

OpenAM 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 Section 2.3, "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 configuration discovery endpoint for a subrealm named `subrealm` in the top-level realm:

```
$ curl \
 --request GET \
 https://openam.example.com:8443/openam/uma/realms/root/realms/subrealm/.well-known/uma-configuration
{
 "version": "1.0",
 "issuer": "openam.example.com",
 "pat_profiles_supported": [
   "bearer"
 ],
 "aat_profiles_supported": [
   "bearer"
 ],
 "rpt_profiles_supported": [
   "bearer"
 ],
 "pat_grant_types_supported": [
   "authorization_code"
 ],
 "aat_grant_types_supported": [
   "authorization_code"
 ],
 "token_endpoint": "https://openam.example.com:8443/openam/oauth2/realms/root/realms/subrealm/
access_token",
 "authorization_endpoint": "https://openam.example.com:8443/openam/oauth2/realms/root/realms/subrealm/
authorize",
 "introspection_endpoint": "https://openam.example.com:8443/openam/oauth2/realms/root/realms/subrealm/
introspect",
 "resource_set_registration_endpoint": "https://openam.example.com:8443/openam/oauth2/realms/root/realms/
subrealm/resource_set",
 "permission_registration_endpoint": "https://openam.example.com:8443/openam/uma/realms/root/realms/
subrealm/permission_request",
 "rpt_endpoint": "https://openam.example.com:8443/openam/uma/realms/root/realms/subrealm/authz_request",
 "dynamic_client_endpoint": "https://openam.example.com:8443/openam/oauth2/realms/root/realms/subrealm/
connect/register"
}
```

The JSON object returned includes the following configuration information:

`version`

The supported UMA core protocol version.

`issuer`

The URI of the issuing authorization server.

`pat_profiles_supported`

The supported OAuth token types used for issuing Protection API Tokens (PATs).

`aat_profiles_supported`

The supported OAuth token types used for issuing Authorization API Tokens (AATs).

`rpt_profiles_supported`

The supported Requesting Party Token (RPT) profiles.

`pat_grant_types_supported`

The supported OAuth grant types used for issuing PATs.

`aat_grant_types_supported`

The supported OAuth grant types used for issuing AATs.

`token_endpoint`

The URI to request a PAT or AAT.

`authorization_endpoint`

The URI to request authorization for issuing a PAT or AAT.

`introspection_endpoint`

The URI to introspect an RPT.

For more information, see Section 3.1, "OAuth 2.0 Client and Resource Server Endpoints" in the *OAuth 2.0 Guide*.

`resource_set_registration_endpoint`

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

For more information, see Section 2.3, "Managing UMA Resource Sets".

`permission_registration_endpoint`

The URI for a resource server to register a requested permission.

For more information, see Procedure 3.1, "To Register an UMA Permission Request".

`rpt_endpoint`

The URI for the client to request authorization data.

For more information, see Procedure 3.3, "To Acquire a Requesting Party Token".

`dynamic_client_endpoint`

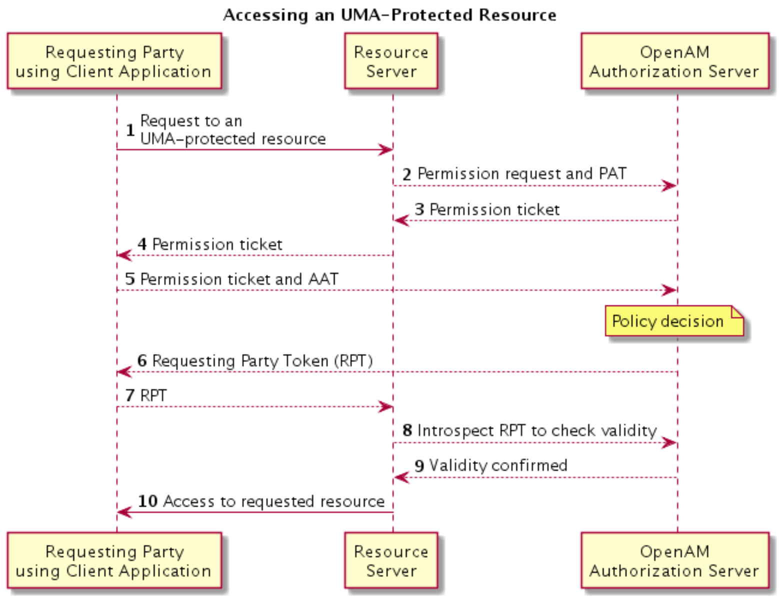
The URI for registering a dynamic client.

3.2. Accessing UMA Protected Resources

To access an UMA-protected resource, a client must provide the resource server with a Requesting Party Token (RPT) obtained from OpenAM, which is acting as the authorization server.

In order to obtain access to an UMA-protected resource, the following actions take place:

Figure 3.1. UMA RPT Token Flow


Accessing an UMA-Protected Resource
Requesting Party using Client Application
Resource Server
OpenAM Authorization Server
1 Request to an UMA-protected resource
2 Permission request and PAT
3 Permission ticket
4 Permission ticket
5 Permission ticket and AAT
Policy decision
6 Requesting Party Token (RPT)
7 RPT
8 Introspect RPT to check validity
9 Validity confirmed
10 Access to requested resource
Requesting Party using Client Application
Resource Server
OpenAM Authorization Server

- A requesting party, using a client application, requests access to an UMA-protected resource (labeled **1** in the diagram above).
- The resource server registers a permission request with OpenAM on behalf of the client (**2**), which contains the ID of the resource set to access, and the requested scopes. A permission ticket is returned (**3**), which the resource server provides to the client (**4**).

  For more information about registering permission requests, see Procedure 3.1, "To Register an UMA Permission Request".
- The client uses the permission ticket, and an Authorization API Token (AAT) to acquire an RPT from OpenAM (**5**).

  For more information about acquiring an RPT, see Procedure 3.3, "To Acquire a Requesting Party Token".
- OpenAM 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 (**6**).
- The client presents the RPT to the resource server (**7**) which must verify the token is valid using the OpenAM introspection endpoint (**8**).

  For more information about the introspection endpoint, see Section 3.1, "OAuth 2.0 Client and Resource Server Endpoints" in the *OAuth 2.0 Guide*.

If the RPT is confirmed to be valid, and non-expired (**9**) the resource server can return the protected resource to the requesting party (**10**).

OpenAM exposes the following endpoints for managing UMA workflow and accessing protected resources:

`/uma/permission_request`

For registering permission requests. For more information, see Procedure 3.1, "To Register an UMA Permission Request".

`/uma/authz_request`

For acquiring requesting party tokens. For more information, see Procedure 3.3, "To Acquire a Requesting Party Token".

Procedure 3.1. To Register an UMA Permission Request

OpenAM provides the `/uma/permission_request` REST endpoint for a resource server to register an access request on behalf of a client.

To register a permission request, the resource server must first acquire a PAT token, as described in Procedure 2.4, "To Acquire a Protection API Token".

Once you have the PAT bearer token, you can access the `/uma/permission_request` endpoint to register a permission request, as shown below:

- Create a POST request to the permission_request endpoint, including the PAT bearer token in a header named `Authorization`:

```
$ curl \
 --request POST \
 --header "Content-Type: application/json" \
 --header "Authorization: Bearer 515d6551-6512-5279-98b6-c0ef3f03a723" \
 --data \
 '{
   "resource_set_id" : "43225628-4c5b-4206-b7cc-5164da81decd0",
   "scopes" : [
     "http://photoz.example.com/dev/scopes/view",
     "http://photoz.example.com/dev/scopes/all"
   ]
 }' \
 https://openam.example.com:8443/openam/uma/permission_request
{
 "ticket": "dc630c21-7d55-45bf-958d-24d624441138"
}
```

  On success, an HTTP 201 Created status code is returned, as well as a `ticket` property in the JSON-formatted body, which can be used by the client to acquire a requesting party token. For more information, see Procedure 3.3, "To Acquire a Requesting Party Token".

  If the resource set does not allow the requested scopes, an error is returned, as follows:

```
{
 "error_description": "Requested scopes are not in allowed scopes for resource set.",
 "error": "invalid_scope"
}
```

Procedure 3.2. To Acquire an Authorization API Token

You must have first registered an OAuth 2.0 client with a name, such as *`UMA-Client`* and a client password, such as *`password`*. Ensure that `uma_authorization` is in the list of available scopes in the client, and a redirection URI is configured. See Section 2.2, "Registering OAuth 2.0 Clients With the Authorization Service" in the *OAuth 2.0 Guide*.

After a suitable OAuth 2.0 client is configured, perform the following steps to acquire an AAT:

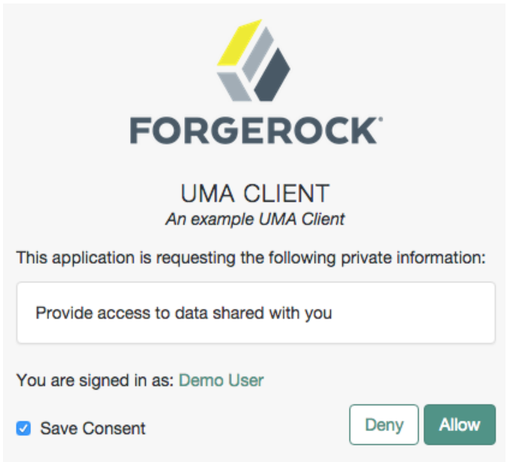
1. Direct the requesting party to the authorization server to obtain an AAT. The URL should specify the client name registered above, the redirect URI, and request the `uma_authorization` scope, as shown in the example below:

   `https://openam.example.com:8443/openam/oauth2/realms/root/authorize?client_id=UMA-Client&redirect_uri=http://openam.example.com:8080&response_type=code&scope=uma_authorization`

   This example uses the OAuth 2.0 code grant, however the UMA client can use any of the OAuth 2.0 grants to obtain the access token.

2. After logging in, the consent screen asks the requesting party to allow or deny the requested scopes.

Figure 3.2. Consent Screen Presented to the Requesting Party



This application is requesting the following private information:

Provide access to data shared with you

You are signed in as: Demo User

☑ Save Consent

3. If the requesting party allows access, they are sent to the configured redirection URL, which will have a `code` query string parameter added, which is used to request the AAT.

4. Create a POST request to the `/oauth2/access_token` endpoint, with the client credentials registered earlier, a grant type of `authorization_code`, a redirect URL, and the value of the `code` query string parameter returned in the previous step, as shown below:

   ```
   $ curl \
    --request POST \
    --data 'client_id=UMA-Client' \
    --data 'client_secret=password' \
    --data 'grant_type=authorization_code' \
    --data 'code=2b911969-5b8e-4d07-bf34-612917a37c9d' \
    --data 'redirect_uri=http%3A%2F%2Fopenam.example.com%3A8080' \
    http://openam.example.com:8080/openam/oauth2/realms/root/access_token

   {
    "scope": "uma_authorization print",
    "expires_in": 599,
    "token_type": "Bearer",
    "refresh_token": "e77fac0e-0dc6-40c3-a600-3309451bd6ee",
    "access_token": "d47c2278-460b-41e8-bf98-a8a1206e2c58"
   }
   ```

   The value returned in `access_token` is the AAT bearer token, used in the following procedures.

Procedure 3.3. To Acquire a Requesting Party Token

OpenAM provides the `/uma/authz_request` REST endpoint for acquiring a Requesting Party Token (RPT).

The endpoint is protected - access requires a Authorization API Token (AAT) - an OAuth 2.0 token with a scope of `uma_authorization`. A client must acquire an AAT in order to use the authorization request endpoint. For more information, see Procedure 3.2, "To Acquire an Authorization API Token".

Once the client has an AAT bearer token, it can access the `/uma/authz_request` endpoint to acquire an RPT, as shown below:

- Create a POST request to the authz_request endpoint, including the AAT bearer token in a header named `Authorization`, and the permission token in the JSON body of the request, as follows:

  ```
  $ curl \
   --request POST \
   --header "Content-Type: application/json" \
   --header "Authorization: Bearer 3b08e99c-b09d-4a65-9780-ea0c9e1f0f52" \
   --data \
   '{
     "ticket": "dc630c21-7d55-45bf-958d-24d624441138"
   }' \
   https://openam.example.com:8443/openam/uma/authz_request
  {
   "rpt": "162d6137-68a4-4e8e-950d-edd834589eb73"
  }
  ```

  On success, an HTTP 201 Created status code is returned, as well as the `rpt` property in the JSON-formatted body.

If the resource owner has not shared the resource with the requesting party, an HTTP 403 Forbidden is returned. If OpenAM is configured to email the resource owner upon pending request creation as described in Section 5.2.1, "UMA Provider", the JSON body returned includes a message that the resource owner will be notified to allow or deny access to the resource, as shown below:

```
{
 "error": "request_submitted",
 "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"
}
```

For more information, see Procedure 3.7, "To View and Manage Pending Access Requests"

3.3. Functionality for UMA End Users

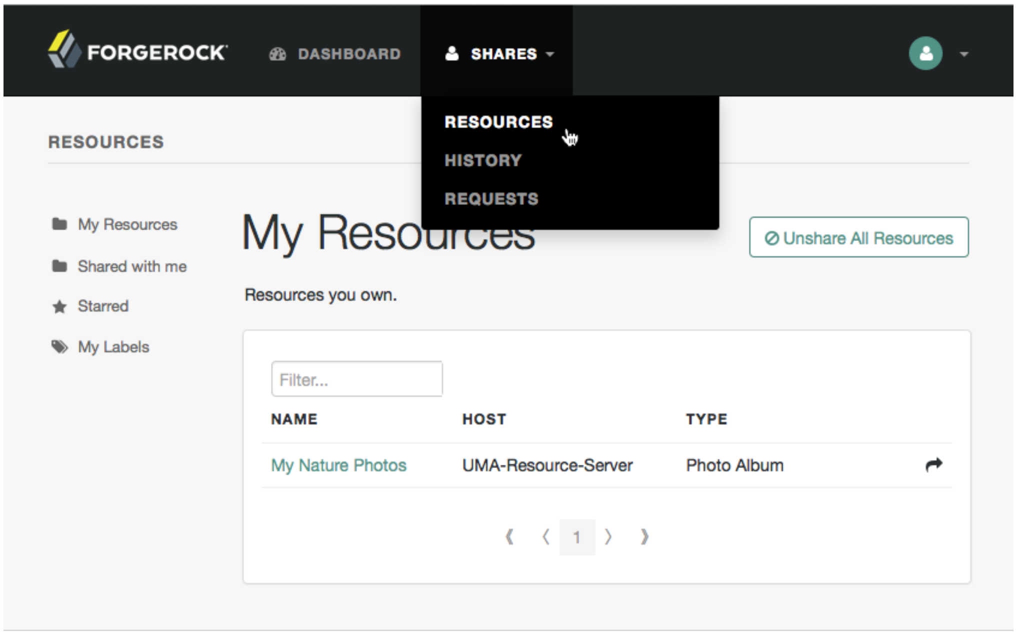
The functionality covered is described in the following procedures:

- Procedure 3.4, "To Share UMA Resources"
- Procedure 3.5, "To Apply User Labels to a Resource"
- Procedure 3.6, "To Mark a Resource as a Favorite"
- Procedure 3.7, "To View and Manage Pending Access Requests"

Procedure 3.4. To Share UMA Resources

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

Figure 3.3. The Resources Page when Logged In



My Resources

Resources you own.

NAME	HOST	TYPE
My Nature Photos	UMA-Resource-Server	Photo Album

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

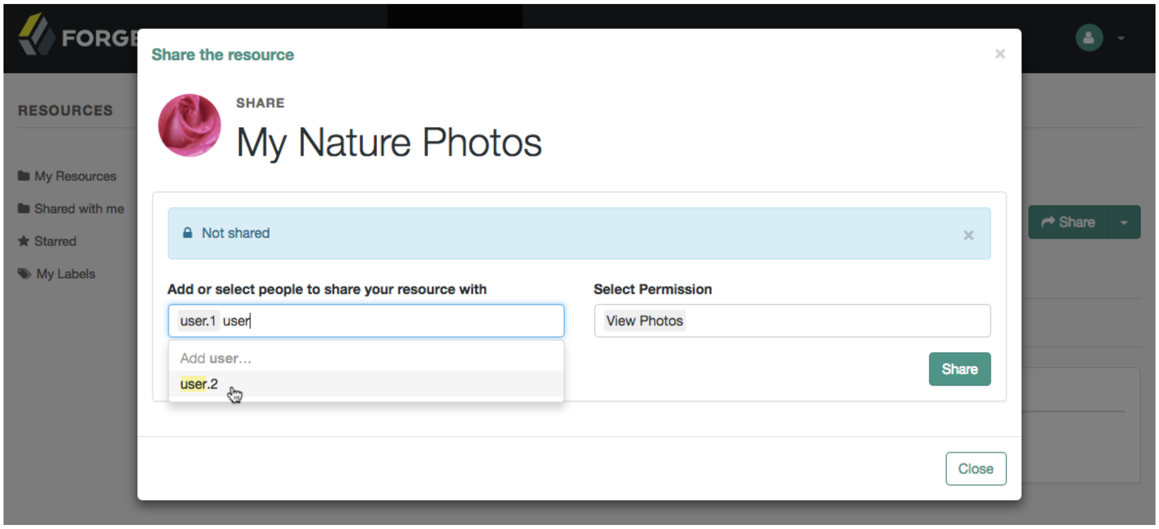
   On the Share the resource form:

   a. Enter the username of the user with whom to share the resource.

   b. In the Select Permission drop-down list, choose the permissions to assign to the user for the selected resource.

   c. Click Share.

Figure 3.4. Sharing an UMA Resource



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

4. When finished, click Close.

Procedure 3.5. To Apply User Labels to a Resource

To apply labels to a resource:

1. Log in to OpenAM 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.

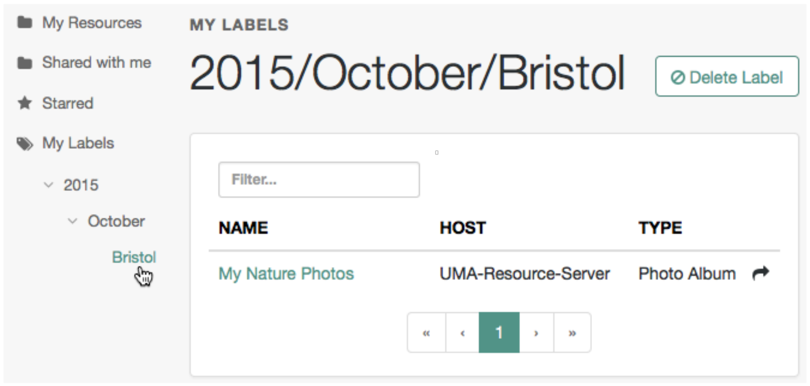
3. On the resource details page, click Edit Labels.

   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`:



NAME	HOST	TYPE
My Nature Photos	UMA-Resource-Server	Photo Album

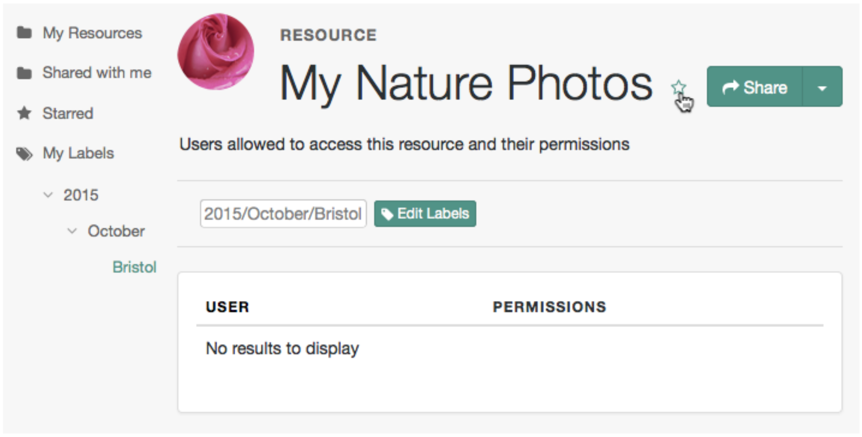
- Click an existing label, and then press **Delete** or **Backspace** to delete the label from the resource.

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.

Procedure 3.6. To Mark a Resource as a Favorite

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

1. Log in to OpenAM 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:



Users allowed to access this resource and their permissions

USER	PERMISSIONS
No results to display	

To view the list of favorite resources, click Starred.

Procedure 3.7. To View and Manage Pending Access Requests

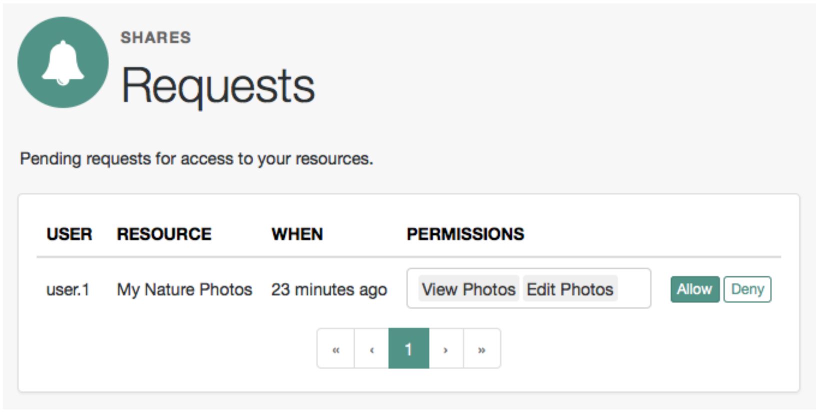
OpenAM supports an UMA workflow in which a user can request access to a resource that has not been explicitly shared with them. The resource owner receives a notification of the request and can choose to allow or deny access.

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

1. Login to OpenAM as the resource owner, and then navigate to Shares > Requests.

   The Requests page is displayed:

   Figure 3.5. UMA Requests Screen Presented to the Resource Owner

   SHARES
   Requests

   Pending requests for access to your resources.

USER	RESOURCE	WHEN	PERMISSIONS	
user.1	My Nature Photos	23 minutes ago	View Photos Edit Photos	Allow Deny

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 4
Customizing UMA

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

OpenAM 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:

- Section 4.1, "Resource Set Registration Extension Point"
- Section 4.2, "Permission Request Extension Point"
- Section 4.3, "Authorization Request Extension Point"
- Section 4.4, "Resource Sharing Extension Point"

4.1. Resource Set Registration Extension Point

OpenAM provides the `ResourceRegistrationFilter` extension point, which can be used to extend UMA resource set registration functionality.

Table 4.1. Resource Set Registration Extension Methods

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

4.2. Permission Request Extension Point

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

Table 4.2. Permission Request Extension Methods

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

4.3. Authorization Request Extension Point

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

Table 4.3. Authorization Request Extension Methods

Method	Parameters	Description
`beforeAuthorization`	*permissionTicket* (type: `PermissionTicket`) *requestingParty* (type: `Subject`) *resourceOwner* (type: `Subject`)	Invoked before authorization of a request is attempted. Throws `UmaException` if authorization of the request should not be attempted.
`afterAuthorization`	*isAuthorized* (type: `boolean`) *permissionTicket* (type: `PermissionTicket`) *requestingParty* (type: `Subject`) *resourceOwner* (type: `Subject`)	Invoked before authorization of a request is attempted. If the authorization request was successful, *isAuthorized* will be `true`.

4.4. Resource Sharing Extension Point

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

Table 4.4. Resource Sharing Extension Methods

Method	Parameters	Description
`beforeResourceShared`	*`umaPolicy`* (type: `UmaPolicy`)	Invoked before creating a sharing policy for a resource. Changes to the *`umaPolicy`* object at this stage *will* be persisted. Throws `ResourceException` if a sharing policy for the resource should not be created.
`afterResourceShared`	*`umaPolicy`* (type: `UmaPolicy`)	Invoked after creating a sharing policy for a resource. Changes to the *`umaPolicy`* object at this stage *will not* be persisted.
`beforeResourceSharedModification`	*`currentUmaPolicy`* (type: `UmaPolicy`) *`updatedUmaPolicy`* (type: `UmaPolicy`)	Invoked before altering the sharing policy of a resource. Changes to the *`updatedUmaPolicy`* object at this stage *will* be persisted. Throws `ResourceException` if the sharing policy of the resource should not be modified.
`onResourceSharedDeletion`	*`umaPolicy`* (type: `UmaPolicy`)	Invoked before deleting the sharing policy of a resource. Throws `ResourceException` if the sharing policy of the resource should not be deleted.
`beforeQueryResourceSets`	*`userId`* (type: `String`) *`queryFilter`* (type: `QueryFilter<JsonPointer>`)	Invoked before querying the resource sets owned or shared with a user. The *`userId`* parameter provides the ID of the user making the query request.

Method	Parameters	Description
		The *`queryFilter`* parameter provides the incoming request query filter. Returns a `QueryFilter` that can be used to return the user's resource sets.

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 OpenAM:

User-Managed Access (UMA) 1.0.1

User-Managed Access (UMA) Profile of OAuth 2.0, in which OpenAM can play the role of authorization server.

OAuth 2.0 Resource Set Registration, in which OpenAM plays the role of authorization server.

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 Section 5.2.1, "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 Section 5.2.2, "UMA Properties".

5.2.1. UMA Provider

ssoadm service name: `uma`

5.2.1.1. Realm Defaults

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

Requesting Party Token Lifetime (seconds)

The maximum life of a Requesting Party Token (RPT) before it expires, in seconds.

Default value: `3600`

ssoadm attribute: `rptLifetime`

Permission Ticket Lifetime (seconds)

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

Default value: `120`

ssoadm 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`

ssoadm 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`

ssoadm attribute: `deleteResourceSetsOnDeleteRS`

Email Resource Owner on Pending Request creation

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`

ssoadm attribute: `emailResourceOwnerOnPendingRequestCreation`

Email Requesting Party on Pending Request approval

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

Default value: `true`

ssoadm attribute: `emailRequestingPartyOnPendingRequestApproval`

User profile preferred Locale attribute

User profile attribute storing the user's preferred locale.

Default value: `inetOrgPerson`

ssoadm attribute: `userProfileLocaleAttribute`

Re-Sharing Mode

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`

ssoadm attribute: `resharingMode`

Require Trust Elevation

Determine if trust elevation is required and claims (such as an OpenID Connection ID token) need to be present in the authorization request. If not, the AAT is sufficient to determine the requesting party of the authorization request.

Default value: `true`

ssoadm attribute: `requireTrustElevation`

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 OpenAM stores UMA tokens. Possible values are:

- `Default Token Store`: OpenAM stores UMA tokens in the embedded data store.
- `External Token Store`: OpenAM 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

OpenAM honors the following properties when `External Token Store` is selected under the Resource Sets Store tab:

SSL/TLS Enabled

When enabled, OpenAM uses SSL or TLS to connect to the external data store. Make sure OpenAM 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 Section 2.3.1.5, "CTS Properties" in the *Reference* for more syntax examples.

Login Id

Specifies the username OpenAM 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 OpenAM 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 OpenAM stores audit information generated when users access UMA resources. Possible values are:

- `Default Token Store`: OpenAM stores UMA audit information in the embedded data store.
- `External Token Store`: OpenAM 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

OpenAM honors the following properties when `External Token Store` is selected under the UMA Audit Store tab:

SSL/TLS Enabled

When enabled, OpenAM uses SSL or TLS to connect to the external data store. Make sure OpenAM 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 Section 2.3.1.5, "CTS Properties" in the *Reference* for more syntax examples.

Login Id

Specifies the username OpenAM 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 OpenAM 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 OpenAM stores pending requests to UMA resources. Possible values are:

- `Default Token Store`: OpenAM stores UMA pending requests in the embedded data store.
- `External Token Store`: OpenAM 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

OpenAM honors the following properties when `External Token Store` is selected under the Pending Requests Store tab:

SSL/TLS Enabled

When enabled, OpenAM uses SSL or TLS to connect to the external data store. Make sure OpenAM 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 Section 2.3.1.5, "CTS Properties" in the *Reference* for more syntax examples.

Login Id

Specifies the username OpenAM 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 OpenAM 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 OpenAM stores user-created labels used for organizing UMA resource sets. Possible values are:

- `Default Token Store`: OpenAM stores user-created labels in the embedded data store.
- `External Token Store`: OpenAM 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

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

SSL/TLS Enabled

When enabled, OpenAM uses SSL or TLS to connect to the external data store. Make sure OpenAM 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 Section 2.3.1.5, "CTS Properties" in the *Reference* for more syntax examples.

Login Id

Specifies the username OpenAM 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 OpenAM 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 OpenAM 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.
- ForgeRock core 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.

Core documentation therefore follows a three-phase review process designed to eliminate errors:

- Product managers and software architects review project documentation design with respect to the readers' software lifecycle needs.
- Subject matter experts review proposed documentation changes for technical accuracy and completeness with respect to the corresponding software.
- Quality experts validate implemented documentation changes for technical accuracy, completeness in scope, and usability for the readership.

The review process helps to ensure that documentation published for a ForgeRock release is technically accurate and complete.

Fully reviewed, published core documentation is available at http://backstage.forgerock.com/. Use this documentation when working with a ForgeRock Identity Platform release.

A.2. Joining the ForgeRock Community

Visit the Community resource center where you can find information about each project, download trial builds, browse the resource catalog, ask and answer questions on the forums, find community events near you, and find the source code for open source software.

A.3. Getting Support and Contacting ForgeRock

ForgeRock provides support services, professional services, classes 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 subjects 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 policy agent profile configuration information, typically created to delegate policy agent profile creation to the user installing a policy 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	In general terms, a service exposing protected resources. In the context of OpenAM policies, the application is a template that constrains the policies that govern access to protected resources. An application can have zero or more policies.
Application type	Application types act as templates for creating policy applications. Application types define a preset list of actions and functional logic, such as policy lookup and resource comparator logic.

	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	OpenAM 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. OpenAM 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. OpenAM can play this role in the OAuth 2.0 authorization framework.
Conditions	Defined as part of policies, these determine the circumstances under which which a policy applies.
	Environmental conditions reflect circumstances like the client IP address, time of day, how the subject authenticated, or the authentication level achieved.

	Subject conditions reflect characteristics of the subject like whether the subject authenticated, the identity of the subject, or claims in the subject's JWT.
Configuration datastore	LDAP directory service holding OpenAM configuration data.
Cross-domain single sign-on (CDSSO)	OpenAM capability allowing single sign-on across different DNS domains.
Delegation	Granting users administrative privileges with OpenAM.
Entitlement	Decision that defines which resource names can and cannot be accessed for a given subject 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 OpenAM.
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 OpenAM on the service provider side; OpenAM lets you create Java Fedlets.
Hot swappable	Refers to configuration properties for which changes can take effect without restarting the container where OpenAM 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 EE policy agent	Java web application installed in a web container that acts as a policy agent, 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	Agent that intercepts requests for resources, directs principals to OpenAM for authentication, and enforces policy decisions from OpenAM.
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, OpenAM associates the Subject with the Principal.
Privilege	In the context of delegated administration, a set of administrative tasks that can be performed by specified subjects in a given realm.
Provider federation	Agreement among providers to participate in a circle of trust.
Realm	OpenAM 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 OpenAM 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 OpenAM 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).
Session	The interval that starts with the user authenticating through OpenAM and ends when the user logs out, or when their session is terminated. For browser-based clients, OpenAM manages user sessions across one or more applications by setting a session cookie. See also Stateful session and Stateless session.
Session high availability	Capability that lets any OpenAM 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 OpenAM after successful authentication. For a Stateful session, 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	Group of OpenAM servers configured the same way, accessed through a load balancer layer. The load balancer handles failover to provide service-level availability. Use sticky load balancing based on `amlbcookie` values to improve site performance. The load balancer can also be used to protect OpenAM services.
Standard metadata	Standard federation configuration information that you can share with other access management software.
Stateful session	An OpenAM session that resides in the Core Token Service's token store. Stateful sessions might also be cached in memory on one or

	more OpenAM servers. OpenAM tracks stateful 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.
Stateless session	An OpenAM session for which state information is encoded in OpenAM and stored on the client. The information from the session is not retained in the CTS token store. For browser-based clients, OpenAM sets a cookie in the browser that contains the session information.
Subject	Entity that requests access to a resource When a subject successfully authenticates, OpenAM associates the subject with the Principal that distinguishes it from other subjects. A subject can be associated with multiple principals.
User data store	Data storage service holding principals' profiles; underlying storage can be an LDAP directory service, a relational database, or a custom `IdRepo` implementation.
Web policy agent	Native library installed in a web server that acts as a policy agent with policies based on web page URLs.