REST API

This guide provides general information about AM's REST APIs.

Get started

Learn about ForgeRock® Common REST and how AM supports it.

API Explorer

Access the online AM REST API reference through the AM admin UI.

REST endpoints

Discover the REST endpoints AM exposes.

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.

REST in AM

Representational State Transfer (REST) is an architectural style that sets certain constraints for designing and building large-scale distributed hypermedia systems.

As an architectural style, REST has very broad applications. The designs of both HTTP 1.1 and URIs follow RESTful principles. The World Wide Web is no doubt the largest and best

known REST application. Many other web services also follow the REST architectural style. Examples include OAuth 2.0, OpenID Connect 1.0, and User-Managed Access (UMA).

The ForgeRock Common REST API applies RESTful principles to define common verbs for HTTP-based APIs. Most native AM REST APIs use the common REST verbs. In contrast, OAuth 2.0, OpenID Connect 1.0 and UMA APIs follow their respective standards.

See Configure identities and realms over REST for examples of how to use the REST API in AM.

NOTE

The ForgeRock Common REST interface stability is classified as evolving.

About ForgeRock Common REST

ForgeRock® Common REST is a common REST API framework. It works across the ForgeRock platform to provide common ways to access web resources and collections of resources. Adapt the examples in this section to your resources and deployment.

NOTE

This page describes the full Common REST framework. Some platform component products do not implement all Common REST behaviors exactly as described. For details, refer to the product-specific examples and reference information.

Common REST resources

Servers generally return JSON-format resources, though resource formats can depend on the implementation.

Resources in collections can be found by their unique identifiers (IDs). IDs are exposed in the resource URIs. For example, if a server has a user collection under `/users`, then you can access a user at `/users/user-id`. The ID is also the value of the `_id` field of the resource.

Resources are versioned using revision numbers. A revision is specified in the resource's `_rev` field. Revisions make it possible to figure out whether to apply changes without resource locking and without distributed transactions.

Common REST verbs

The Common REST APIs use the following verbs, sometimes referred to collectively as `CRUDPAQ`. For details and HTTP-based examples of each, follow the links to the sections for each verb.

Create

Add a new resource.

This verb maps to HTTP PUT or HTTP POST.

For details, see Create.

Read

Retrieve a single resource.

This verb maps to HTTP GET.

For details, see Read.

Update

Replace an existing resource.

This verb maps to HTTP PUT.

For details, see Update.

Delete

Remove an existing resource.

This verb maps to HTTP DELETE.

For details, see Delete.

Patch

Modify part of an existing resource.

This verb maps to HTTP PATCH.

For details, see Patch.

Action

Perform a predefined action.

This verb maps to HTTP POST.

For details, see Action.

Query

Search a collection of resources.

This verb maps to HTTP GET.

For details, see Query.

Common REST parameters

Common REST reserved query string parameter names start with an underscore, `_`. Reserved query string parameters include, but are not limited to, the following names:

- `_action`
- `_api`
- `_crestapi`
- `_fields`
- `_mimeType`
- `_pageSize`
- `_pagedResultsCookie`
- `_pagedResultsOffset`
- `_prettyPrint`
- `_queryExpression`
- `_queryFilter`
- `_queryId`
- `_sortKeys`
- `_totalPagedResultsPolicy`

NOTE

Some parameter values are not safe for URLs, so URL-encode parameter values as necessary.

Continue reading for details about how to use each parameter.

Common REST extension points

The *action* verb is the main vehicle for extensions. For example, to create a new user with HTTP POST rather than HTTP PUT, you might use `/users?_action=create`. A server can define additional actions. For example, `/tasks/1?_action=cancel`.

A server can define *stored queries* to call by ID. For example, `/groups?_queryId=hasDeletedMembers`. Stored queries can call for additional parameters. The parameters are also passed in the query string. Which parameters are valid depends on the stored query.

Common REST API documentation

Common REST APIs often depend at least in part on runtime configuration. Many Common REST endpoints therefore serve *API descriptors* at runtime. An API descriptor documents the actual API as it is configured.

Use the following query string parameters to retrieve API descriptors:

_api
: Serves an API descriptor that complies with the OpenAPI specification.

  This API descriptor represents the API accessible over HTTP. It is suitable for use with popular tools such as Swagger UI.

_crestapi
: Serves a native Common REST API descriptor.

  This API descriptor provides a compact representation that is not dependent on the transport protocol. It requires a client that understands Common REST, as it omits many Common REST defaults.

NOTE

Consider limiting access to API descriptors in production environments in order to avoid unnecessary traffic.

To provide documentation in production environments, see To publish OpenAPI documentation instead.

To publish OpenAPI documentation

In production systems, developers expect stable, well-documented APIs. Rather than retrieving API descriptors at runtime through Common REST, prepare final versions, and publish them alongside the software in production.

Use the OpenAPI-compliant descriptors to provide API reference documentation for your developers:

1. Configure the software to produce production-ready APIs.

   In other words, configure the software as for production so that the APIs match exactly.

2. Retrieve the OpenAPI-compliant descriptor.

   The following command saves the descriptor to a file, `myapi.json` :

   ```
   $ curl -o myapi.json endpoint?_api
   ```

3. If necessary, edit the descriptor.

   For example, add security definitions to describe the API protection.

4. Publish the descriptor using a tool such as Swagger UI.

Create

There are two ways to create a resource, HTTP POST or HTTP PUT.

To create a resource using POST, perform an HTTP POST with the query string parameter `_action=create`, and the JSON resource as a payload. Accept a JSON response. The server creates the identifier if not specified:

```
POST /users?_action=create HTTP/1.1
Host: example.com
Accept: application/json
Content-Length: ...
Content-Type: application/json
{ JSON resource }
```

To create a resource using PUT, perform an HTTP PUT including the case-sensitive identifier for the resource in the URL path, and the JSON resource as a payload. Use the `If-None-Match: *` header. Accept a JSON response:

```
PUT /users/some-id HTTP/1.1
Host: example.com
Accept: application/json
Content-Length: ...
Content-Type: application/json
If-None-Match: *
{ JSON resource }
```

The `_id` and content of the resource depend on the server implementation. The server is not required to use the `_id` that the client provides. The server response to the request indicates the resource location as the value of the `Location` header.

If you include the `If-None-Match` header, you must use `If-None-Match: *`. In this case, the request creates the object if it does not exist, and fails if the object does exist. If you include any value other `If-None-Match: *`, the server returns an HTTP 400 Bad Request error. For example, creating an object with `If-None-Match: revision` returns a bad request error.

If you do not include `If-None-Match: *`, the request creates the object if it does not exist, and *updates* the object if it does exist.

Parameters

_fields=field[,field…]

Return only the specified fields in the body of the response.

The `field` values are JSON pointers. For example if the resource is `{"parent":{"child":"value"}}`, `parent/child` refers to the `"child":"value"`.

If the `field` is left blank, the server returns all default values.

_prettyPrint=true
Format the body of the response.

Read

To retrieve a single resource, perform an HTTP GET on the resource by its case-sensitive identifier (`_id`), and accept a JSON response:

```
GET /users/some-id HTTP/1.1
Host: example.com
Accept: application/json
```

Parameters
_fields=field[,field…]
Return only the specified fields in the body of the response.

The `field` values are JSON pointers. For example if the resource is `{"parent":{"child":"value"}}`, `parent/child` refers to the `"child":"value"`.

If the `field` is left blank, the server returns all default values.

_mimeType=mime-type
Some resources have fields whose values are multi-media resources, such as a profile photo.

If the feature is enabled for the endpoint, you can read a single field that is a multi-media resource by specifying the *field* and *mime-type*.

In this case, the content type of the field value returned matches the *mime-type* that you specify, and the body of the response is the multi-media resource.

Do not use the `Accept` header in this case. For example, `Accept: image/png` does not work. Use the `_mimeType` query string parameter instead.

_prettyPrint=true
Format the body of the response.

Update

To update a resource, perform an HTTP PUT including the case-sensitive identifier (`_id`) as the final element of the path to the resource, and the JSON resource as the payload. Use the `If-Match: _rev` header to check that you are actually updating the version

you modified. Use `If-Match: *` if the version does not matter. Accept a JSON response:

```
PUT /users/some-id HTTP/1.1
Host: example.com
Accept: application/json
Content-Length: ...
Content-Type: application/json
If-Match: _rev
{ JSON resource }
```

When updating a resource, include all the attributes to retain. Omitting an attribute in the resource amounts to deleting the attribute unless it is not under the control of your application. Attributes not under the control of your application include private and read-only attributes. In addition, virtual attributes and relationship references might not be under the control of your application.

NOTE

Product-specific implementations may differ. Not all products use the payload to replace the state of the resource in its entirety. For example, attributes that are omitted from the request payload to AM will not be deleted. Instead, you need to specify the attribute and set the value to an empty array to delete the attribute from the resource.

For more information, see the product-specific examples and reference information.

Parameters

_fields=field[,field…]

Return only the specified fields in the body of the response.

The `field` values are JSON pointers. For example if the resource is `{"parent":{"child":"value"}}`, `parent/child` refers to the `"child":"value"`.

If the `field` is left blank, the server returns all default values.

_prettyPrint=true

Format the body of the response.

Delete

To delete a single resource, perform an HTTP DELETE by its case-sensitive identifier (`\_id`) and accept a JSON response:

```
DELETE /users/some-id HTTP/1.1
Host: example.com
Accept: application/json
```

Parameters

_fields=field[,field…]

Return only the specified fields in the body of the response.

The `field` values are JSON pointers. For example if the resource is `{"parent":{"child":"value"}}`, `parent/child` refers to the `"child":"value"`.

If the `field` is left blank, the server returns all default values.

_prettyPrint=true

Format the body of the response.

Patch

To patch a resource, send an HTTP PATCH request with the following parameters:

- `operation`
- `field`
- `value`
- `from` (optional with copy and move operations)

You can include these parameters in the payload for a PATCH request, or in a JSON PATCH file. If successful, you'll see a JSON response similar to the following:

```
PATCH /users/some-id HTTP/1.1
Host: example.com
Accept: application/json
Content-Length: ...
Content-Type: application/json
If-Match: _rev
{ JSON array of patch operations }
```

PATCH operations apply to three types of targets:

- **single-valued**, such as an object, string, boolean, or number.
- **list semantics array**, where the elements are ordered, and duplicates are allowed.
- **set semantics array**, where the elements are not ordered, and duplicates are not allowed.

ForgeRock PATCH supports multiple `operations`:

Patch operation: add

The `add` operation ensures that the target field contains the value provided, creating parent fields as necessary.

If the target field is single-valued, then the value you include in the PATCH replaces the value of the target. A single-valued field is an `object`, `string`, `boolean`, or `number`.

An `add` operation has different results on two standard types of arrays:

- **List semantic arrays**: you can run any of these `add` operations on that type of array:
  - If you `add` an array of values, the PATCH operation appends it to the existing list of values.
  - If you `add` a single value, specify an ordinal element in the target array, or use the `{-}` special index to add that value to the end of the list.
- **Set semantic arrays**: The value included in the patch is merged with the existing set of values. Any duplicates within the array are removed.

As an example, start with the following list semantic array resource:

```
{
    "fruits" : [ "orange", "apple" ]
}
```

The following add operation includes the pineapple to the end of the list of fruits, as indicated by the `-` at the end of the `fruits` array.

```
{
    "operation" : "add",
    "field" : "/fruits/-",
    "value" : "pineapple"
}
```

The following is the resulting resource:

```
{
    "fruits" : [ "orange", "apple", "pineapple" ]
}
```

You can add only one array element one at a time, as per the corresponding JSON Patch specification. If you add an array of elements, for example:

```
{
    "operation" : "add",
    "field" : "/fruits/-",
    "value" : ["pineapple", "mango"]
}
```

The resulting resource would have the following invalid JSON structure:

```
{
    "fruits" : [ "orange", "apple", ["pineapple", "mango"]]
}
```

Patch operation: copy

The copy operation takes one or more existing values from the source field. It then adds those same values on the target field. Once the values are known, it is equivalent to performing an `add` operation on the target field.

The following `copy` operation takes the value from a field named `mail`, and then runs a `replace` operation on the target field, `another_mail`.

```
[
  {
     "operation":"copy",
     "from":"mail",
     "field":"another_mail"
  }
]
```

If the source and target field values are arrays, the result depends on whether the array has list semantics or set semantics, as described in Patch operation: add.

Patch operation: increment

The `increment` operation changes the value or values of the target field by the amount you specify. The value that you include must be one number, and may be positive or negative. The value of the target field must accept numbers. The following `increment` operation adds `1000` to the target value of `/user/payment`.

```
[
  {
    "operation" : "increment",
    "field" : "/user/payment",
```

```
    "value" : "1000"
  }
]
```

Since the `value` of the `increment` is a single number, arrays do not apply.

Patch operation: move

The move operation removes existing values on the source field. It then adds those same values on the target field. This is equivalent to a `remove` operation on the source, followed by an `add` operation with the same values, on the target.

The following `move` operation is equivalent to a `remove` operation on the source field, `surname`, followed by a `replace` operation on the target field value, `lastName`. If the target field does not exist, it is created:

```
[
  {
     "operation":"move",
     "from":"surname",
     "field":"lastName"
  }
]
```

To apply a `move` operation on an array, you need a compatible single-value, list semantic array, or set semantic array on both the source and the target. For details, see the criteria described in Patch operation: add.

Patch operation: remove

The `remove` operation ensures that the target field no longer contains the value provided. If the remove operation does not include a value, the operation removes the field. The following `remove` deletes the value of the `phoneNumber`, along with the field.

```
[
  {
    "operation" : "remove",
    "field" : "phoneNumber"
  }
]
```

If the object has more than one `phoneNumber`, those values are stored as an array.

A `remove` operation has different results on two standard types of arrays:

- **List semantic arrays**: A `remove` operation deletes the specified element in the array. For example, the following operation removes the first phone number, based on its array index (zero-based):

```
[
    {
        "operation" : "remove",
        "field" : "/phoneNumber/0"
    }
]
```

- **Set semantic arrays**: The list of values included in a patch are removed from the existing array.

Patch operation: replace

The `replace` operation removes any existing value(s) of the targeted field, and replaces them with the provided value(s). It is essentially equivalent to a `remove` followed by a `add` operation. If the arrays are used, the criteria is based on Patch operation: add. However, indexed updates are not allowed, even when the target is an array.

The following `replace` operation removes the existing `telephoneNumber` value for the user, and then adds the new value of `+1 408 555 9999`.

```
[
  {
    "operation" : "replace",
    "field" : "/telephoneNumber",
    "value" : "+1 408 555 9999"
  }
]
```

A PATCH replace operation on a list semantic array works as a PATCH remove operation. The following example demonstrates how the effect of both operations. Start with the following resource:

```
{
    "fruits" : [ "apple", "orange", "kiwi", "lime" ],
}
```

Apply the following operations on that resource:

```
[
  {
```

```
    "operation" : "remove",
    "field" : "/fruits/0",
    "value" : ""
  },
  {
    "operation" : "replace",
    "field" : "/fruits/1",
    "value" : "pineapple"
  }
]
```

The PATCH operations are applied sequentially. The `remove` operation removes the first member of that resource, based on its array index, (`fruits/0`), with the following result:

```
[
  {
    "fruits" : [ "orange", "kiwi", "lime" ],
  }
]
```

The second PATCH operation, a `replace` , is applied on the second member (`fruits/1`) of the intermediate resource, with the following result:

```
[
  {
    "fruits" : [ "orange", "pineapple", "lime" ],
  }
]
```

Patch operation: transform

The `transform` operation changes the value of a field based on a script, or some other data transformation command. The following `transform` operation takes the value from the field named `/objects` , and applies the `something.js` script as shown:

```
[
  {
    "operation" : "transform",
    "field" : "/objects",
    "value" : {
      "script" : {
        "type" : "text/javascript",
```

```
        "file" : "something.js"
      }
    }
  }
]
```

Patch operation limitations

Some HTTP client libraries do not support the HTTP PATCH operation. Make sure that the library you use supports HTTP PATCH before using this REST operation.

For example, the Java Development Kit HTTP client does not support PATCH as a valid HTTP method. Instead, the method `HttpURLConnection.setRequestMethod("PATCH")` throws `ProtocolException`.

Parameters

_fields=field[,field…]

Return only the specified fields in the body of the response.

The `field` values are JSON pointers. For example if the resource is `{"parent":{"child":"value"}}`, `parent/child` refers to the `"child":"value"`.

If the `field` is left blank, the server returns all default values.

_prettyPrint=true

Format the body of the response.

Action

Actions are a means of extending Common REST APIs and are defined by the resource provider, so the actions you can use depend on the implementation.

The standard action indicated by `_action=create` is described in Create.

Parameters

In addition to these parameters, specific action implementations have their own parameters:

_fields=field[,field…]

Return only the specified fields in the body of the response.

The `field` values are JSON pointers. For example if the resource is `{"parent":{"child":"value"}}`, `parent/child` refers to the `"child":"value"`.

If the `field` is left blank, the server returns all default values.

_prettyPrint=true

Format the body of the response.

Query

To query a resource collection (or resource container), perform an HTTP GET, and accept a JSON response, including either a `_queryExpression`, `_queryFilter`, or `_queryId` parameter. The parameters cannot be used together:

```
GET /users?_queryFilter=true HTTP/1.1
Host: example.com
Accept: application/json
```

The server returns the result as a JSON object including a `"results"` array, and other fields that depend on the parameters.

Parameters

_countOnly=true

Return a count of query results without returning the resources.

This parameter requires protocol version 2.2 or later.

_fields=field[,field…]

Return only the specified fields in the body of the response.

The `field` values are JSON pointers. For example if the resource is `{"parent":{"child":"value"}}`, `parent/child` refers to the `"child":"value"`.

If the `field` is left blank, the server returns all default values.

_queryFilter=filter-expression

Query filters request that the server return entries that match the filter expression. You must URL-escape the filter expression.

The string representation is summarized as follows. Continue reading for additional explanation:

```
Expr           = OrExpr
OrExpr         = AndExpr ( 'or' AndExpr ) *
AndExpr        = NotExpr ( 'and' NotExpr ) *
NotExpr        = '!' PrimaryExpr | PrimaryExpr
PrimaryExpr    = '(' Expr ')' | ComparisonExpr | PresenceExpr |
LiteralExpr
ComparisonExpr = Pointer OpName JsonValue
PresenceExpr   = Pointer 'pr'
LiteralExpr    = 'true' | 'false'
Pointer        = JSON pointer
OpName         = 'eq' |  # equal to
                 'co' |  # contains
```

```
                  'sw' |  # starts with
                  'lt' |  # less than
                  'le' |  # less than or equal to
                  'gt' |  # greater than
                  'ge' |  # greater than or equal to
                  STRING  # extended operator
JsonValue       = NUMBER | BOOLEAN | '"' UTF8STRING '"'
STRING          = ASCII string not containing white-space
UTF8STRING      = UTF-8 string possibly containing white-space
```

JsonValue components of filter expressions follow RFC 7159: The JavaScript Object Notation (JSON) Data Interchange Format. In particular, as described in section 7 of the RFC, the escape character in strings is the backslash character. For example, to match the identifier `test\`, use `_id eq 'test\\'`. In the JSON resource, the `\` is escaped the same way: `"_id":"test\\"`.

When using a query filter in a URL, the filter expression is part of a query string parameter. A query string parameter must be URL encoded, as described in RFC 3986: Uniform Resource Identifier (URI): Generic Syntax. For example, white space, double quotes ("), parentheses, and exclamation characters must be URL encoded in HTTP query strings. The following rules apply to URL query components:

```
query       = *( pchar / "/" / "?" )
pchar       = unreserved / pct-encoded / sub-delims / ":" / "@"
unreserved  = ALPHA / DIGIT / "-" / "." / "_" / "~"
pct-encoded = "%" HEXDIG HEXDIG
sub-delims  = "!" / "$" / "&amp;" / "'" / "(" / ")"
                 / "*" / "+" / "," / ";" / "="
```

`ALPHA`, `DIGIT`, and `HEXDIG` are core rules of RFC 5234: Augmented BNF for Syntax Specifications:

```
ALPHA       =  %x41-5A / %x61-7A   ; A-Z / a-z
DIGIT       =  %x30-39             ; 0-9
HEXDIG      =  DIGIT / "A" / "B" / "C" / "D" / "E" / "F"
```

As a result, a backslash escape character in a *JsonValue* component is percent-encoded in the URL query string parameter as `%5C`. To encode the query filter expression `_id eq 'test\\'`, use `_id+eq+'test%5C%5C'`, for example.

A simple filter expression can represent a comparison, presence, or a literal value.

For comparison expressions, use *`json-pointer comparator json-value`*, where the *comparator* is one of the following:

```
eq (equals)
co (contains)
sw (starts with)
lt (less than)
le (less than or equal to)
gt (greater than)
ge (greater than or equal to)
```

For presence, use *`json-pointer pr`* to match resources where the JSON pointer is present, and the value it points to is not `null`.

Literal values include `true` (match anything) and `false` (match nothing).

Complex expressions employ `and`, `or`, and `!` (not), with parentheses, (*`expression`*), to group expressions.

`_queryId=identifier`

Specify a query by its identifier.

Specific queries can take their own query string parameter arguments, which depend on the implementation.

`_pagedResultsCookie=string`

The string is an opaque cookie used by the server to keep track of the position in the search results. The server returns the cookie in the JSON response as the value of `pagedResultsCookie`.

In the request `_pageSize` must also be set and non-zero. You receive the cookie value from the provider on the first request, and then supply the cookie value in subsequent requests until the server returns a `null` cookie, meaning the final page of results has been returned.

The `_pagedResultsCookie` parameter is supported when used with the `_queryFilter` parameter. The `_pagedResultsCookie` parameter is not guaranteed to work with the `_queryExpression` or `_queryId` parameters.

The `_pagedResultsCookie` and `_pagedResultsOffset` parameters are mutually exclusive, and not to be used together.

`_pagedResultsOffset=integer`

When `_pageSize` is non-zero, use this as an index in the result set indicating the first page to return.

The `_pagedResultsCookie` and `_pagedResultsOffset` parameters are mutually exclusive, and not to be used together.

`_pageSize=integer`

Return query results in pages of this size. After the initial request, use `_pagedResultsCookie` or `_pageResultsOffset` to page through the results.

_prettyPrint=true

Format the body of the response.

_totalPagedResultsPolicy=string

When a `_pageSize` is specified, and non-zero, the server calculates the `"totalPagedResults"`, in accordance with the `totalPagedResultsPolicy`, and provides the value as part of the response.

The `"totalPagedResults"` is either an estimate of the total number of paged results (`_totalPagedResultsPolicy=ESTIMATE`), or the exact total result count (`_totalPagedResultsPolicy=EXACT`). If no count policy is specified in the query, or if `_totalPagedResultsPolicy=NONE`, result counting is disabled, and the server returns value of -1 for `"totalPagedResults"`.

_sortKeys=(|-)__field__[,(|-)field…]

Sort the resources returned based on the specified field(s), either in `+` (ascending, default) order, or in `-` (descending) order.

Because ascending order is the default, including the `` `+` `` character in the query is unnecessary. If you do include the `` `+` `` character, it must be URL-encoded as `%2B`, for example:

```
http://localhost:8080/api/users?
_queryFilter=true&_sortKeys=%2Bname/givenName
```

The `_sortKeys` parameter is not supported for predefined queries (`_queryId`).

HTTP status codes

When working with a Common REST API over HTTP, client applications should expect at least these HTTP status codes. Not all servers necessarily return all status codes identified here:

200 OK

The request was successful and a resource returned, depending on the request.

201 Created

The request succeeded and the resource was created.

204 No Content

The action request succeeded, and there was no content to return.

304 Not Modified

The read request included an `If-None-Match` header, and the value of the header matched the revision value of the resource.

400 Bad Request
The request was malformed.

401 Unauthorized
The request requires user authentication.

403 Forbidden
Access was forbidden during an operation on a resource.

404 Not Found
The specified resource could not be found, perhaps because it does not exist.

405 Method Not Allowed
The HTTP method is not allowed for the requested resource.

406 Not Acceptable
The request contains parameters that are not acceptable, such as a resource or protocol version that is not available.

409 Conflict
The request would have resulted in a conflict with the current state of the resource.

410 Gone
The requested resource is no longer available, and will not become available again. This can happen when resources expire for example.

412 Precondition Failed
The resource's current version does not match the version provided.

415 Unsupported Media Type
The request is in a format not supported by the requested resource for the requested method.

428 Precondition Required
The resource requires a version, but no version was supplied in the request.

500 Internal Server Error
The server encountered an unexpected condition that prevented it from fulfilling the request.

501 Not Implemented
The resource does not support the functionality required to fulfill the request.

503 Service Unavailable
The requested resource was temporarily unavailable. The service may have been disabled, for example.

Online REST API reference

AM provides an online AM REST API reference that can be accessed through the AM admin UI. The *API Explorer* displays the REST API endpoints that allow client applications to access AM's services.

CAUTION

The API Explorer is enabled by default. For security reasons, it is strongly recommended that you disable it in production environments.

To disable the API Explorer, go to **Configure > Global Services > REST APIs**, and select **Disabled** in the **API Descriptors** drop-down list.

The key features of the API Explorer are the following:

- **API versioning**. The API Explorer displays the different API versions available depending on your deployment.

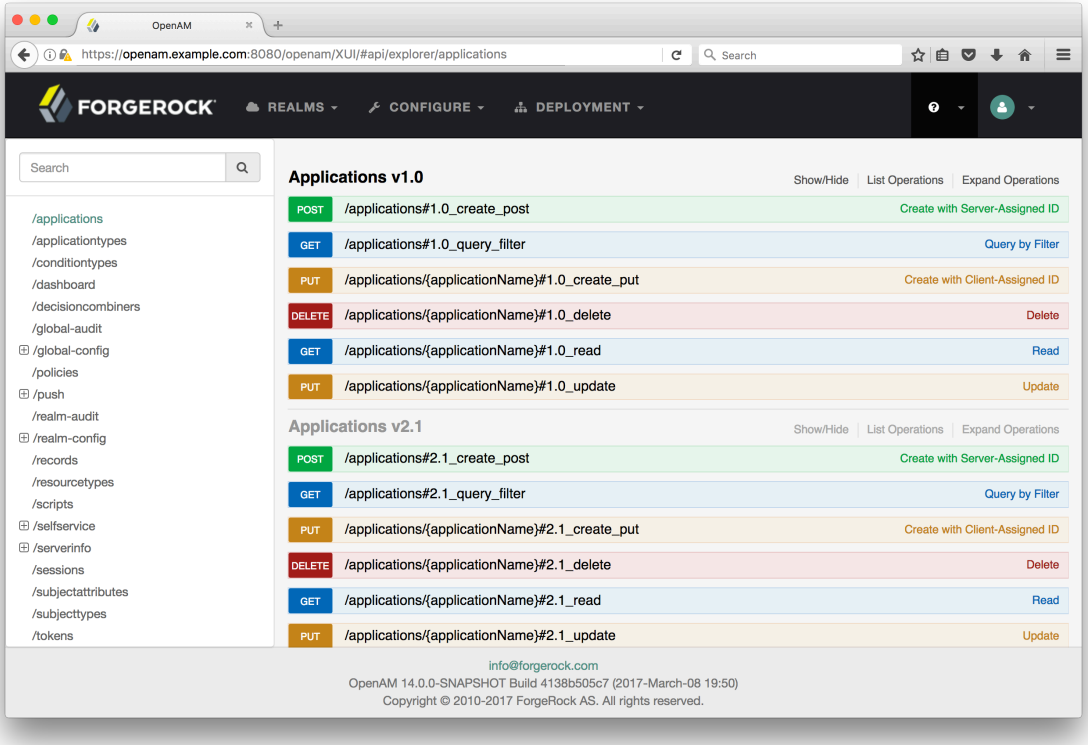

Figure 1. API Explorer

- **Detailed information**. The API Explorer provides an **Expand Operations** button for each available CRUDPAQ method. **Expand Operations** displays implementation notes, successful response class, headers, parameters, and response messages with examples. For example, you can populate the **requestPayload** field with an example value. If you select **Model**, you can view the schema for each parameter, as follows:

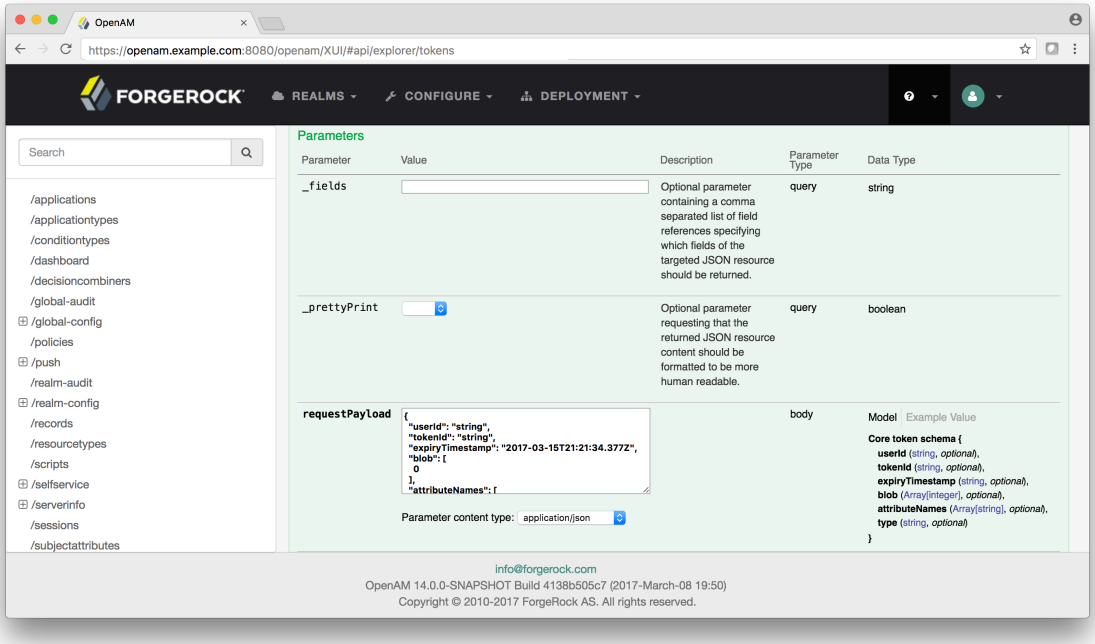

Figure 2. API Explorer Request Payload

- **Try It Out**. The API Explorer also provides a **Try It Out** feature that lets you send a sample request to the endpoint, and view the possible responses.

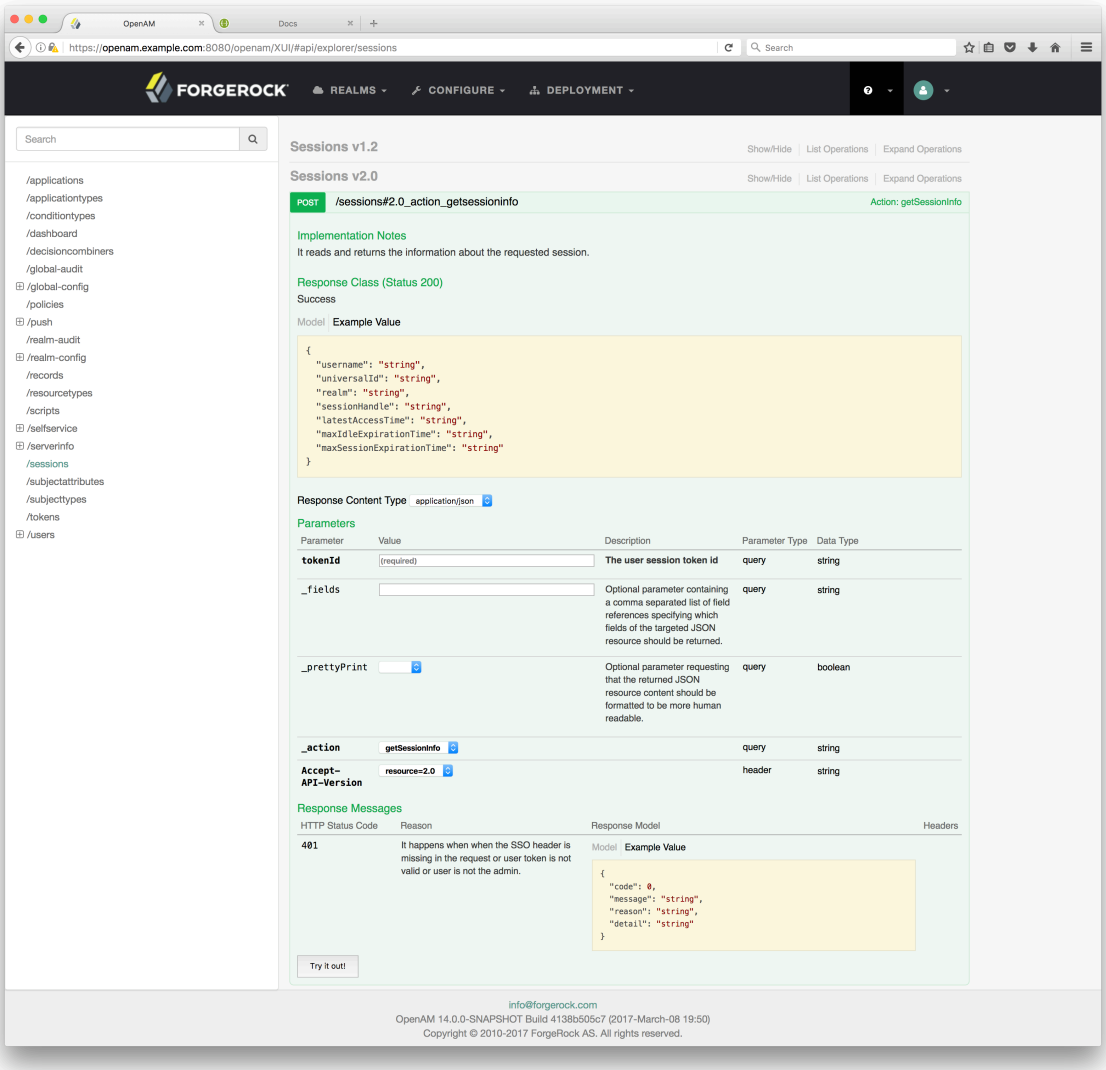

Figure 3. API Explorer Detailed Information

Note the following when using the **Try It Out** feature:

- The example payload values are auto-generated, and while they may be the correct data type, their value may not be correct for the API to function correctly. See the **Model** tab for a description of the required value, and replace the example values before sending the REST request to AM.
- Endpoints in the API Explorer are hard-coded to point to the top-level realm. You must adjust either the domain, or the path in the request, to target a different realm.

  For more information, see Specify realms in REST API calls.

Access the API Explorer

1. From the AM admin UI, you can access the API Explorer in one of two ways:

   Point your browser to the following URL:

   ```
   https://openam.example.com:8443/openam/ui-admin/#api/explorer/applications
   ```

   You can also click the help icon in the top-right corner, and click **API Explorer**.

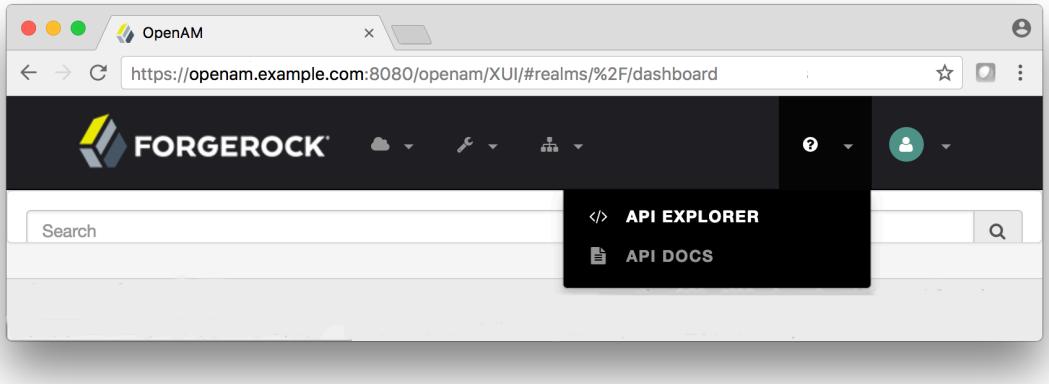

Figure 4. API Explorer

REST API versions

REST API features are assigned version numbers.

Providing version numbers in the REST API helps ensure compatibility between releases. The version number of a feature increases when AM introduces a non-backwards-

compatible change that affects clients making use of the feature.

AM provides versions for the following aspects of the REST API:

resource
: Any changes to the structure or syntax of a returned response will incur a *resource* version change. For example, changing `errorMessage` to `message` in a JSON response.

protocol
: Any changes to the methods used to make REST API calls will incur a *protocol* version change. For example, changing `_action` to `$action` in the required parameters of an API feature.

IMPORTANT

To ensure your clients are always compatible with a newer version of AM, you should always include resource versions in your REST calls.

Moreover, AM includes a CSRF filter for all the endpoints under `/json` that requires that all requests other than GET, HEAD, or OPTIONS include, at least, one of the following headers

- `X-Requested-With`
- `Accept-API-Version`

Learn more in Protect against CSRF attacks.

Specify REST API versions

You can specify which version of the REST API to use by adding an `Accept-API-Version` header to the request. The following example requests *resource* version 2.0 and *protocol* version 1.0:

```
$ curl \
--request POST \
--header "Content-Type: application/json" \
--header "X-OpenAM-Username: demo" \
--header "X-OpenAM-Password: Ch4ng31t" \
--header "Accept-API-Version: resource=2.0, protocol=1.0" \
'https://openam.example.com:8443/openam/json/realms/root/realms/alpha/authenticate'
```

You can configure the default behavior AM will take when a REST call doesn't specify explicit version information. Learn more in Configure versioning behavior.

Supported REST API versions

Find information about the supported protocol and resource versions in the Online REST API reference available in the AM admin UI.

The *AM Release Notes* page, Changes, describes any breaking changes between API versions.

Configure versioning behavior

Configure how AM handles REST calls that don't present an API version:

1. Log in as AM administrator, `amAdmin` .
2. Click **Configure > Global Services**, and click **REST APIs**.
3. In **Default Version**, select the required response to a REST API request that doesn't specify an explicit version:

   The available options for default API version behavior are as follows:

   Latest
   : The latest available supported version of the API is used.

     This is the preset default for new installations of AM.

   Oldest
   : The oldest available supported version of the API is used.

     This is the preset default for upgraded AM instances.

     NOTE

     The oldest supported version might not be the first that was released, as API versions become deprecated or unsupported.

     Learn more in Deprecated in the Release notes.

   None
   : No version will be used. When a client application calls a REST API without specifying the version, AM returns an error, and the request fails.
4. Optionally, enable `Warning Header` to include warning messages in the headers of responses to requests.
5. Save your work.

Version messages

AM provides REST API version messages in the JSON response to a REST API call. You can also configure AM to return version messages in the response headers.

Messages include:

- Details of the REST API versions used to service a REST API call.
- Warning messages if REST API version information is unspecified or is incorrect in a REST API call.

The `resource` and `protocol` version used to service a REST API call are returned in the `Content-API-Version` header, as shown below:

```
$ curl \
-i \
--request POST \
--header "Content-Type: application/json" \
--header "X-OpenAM-Username: demo" \
--header "X-OpenAM-Password: Ch4ng31t" \
--header "Accept-API-Version: resource=2.0, protocol=1.0" \
'https://openam.example.com:8443/openam/json/realms/root/realms/alpha/authenticate'
HTTP/1.1 200 OK
Content-API-Version: protocol=1.0,resource=2.0
Server: Restlet-Framework/2.1.7
Content-Type: application/json;charset=UTF-8

{
    "tokenId":"AQIC5wM…TU3OQ*",
    "successUrl":"/openam/console"
}
```

If the default REST API version behavior is set to `None` , and a REST API call doesn't include the `Accept-API-Version` header, or doesn't specify a `resource` version, then a `404 Not Found` status code is returned. For example:

```
$ curl \
--header "Content-Type: application/json" \
--header "Accept-API-Version: protocol=1.0" \
https://openam.example.com:8443/openam/json/realms/root/serverinfo/*
{
  "code": 404,
  "reason": "Not Found",
```

```
  "message": "Resource '*' not found"
}
```

If a REST API call does include the `Accept-API-Version` header, but the specified `resource` or `protocol` version doesn't exist in AM, then a `404 Not Found` status code is returned. For example:

```
$ curl \
--header "Content-Type: application/json" \
--header "Accept-API-Version: protocol=1.0, resource=999.0" \
https://openam.example.com:8443/openam/json/realms/root/serverinfo
/*
{
  "code": 404,
  "reason": "Not Found",
  "message": "Resource '*' not found"
}
```

TIP

Find information on setting the default REST API version behavior in Specify REST API versions.

Specify realms in REST API calls

Realms can be specified in the following ways when making a REST API call to AM:

DNS alias

When making a REST API call, the DNS alias of a realm can be specified in the subdomain and domain name components of the REST endpoint.

To list all users in the Top Level Realm use the DNS alias of the AM instance, for example:

```
https://openam.example.com:8443/openam/json/users?_queryId=*
```

To list all users in a realm with DNS alias `suppliers.example.com` the REST endpoint would be:

```
https://suppliers.example.com:8443/openam/json/users?_queryId=*
```

Path

When making a REST API call, specify the realm in the path component of the endpoint. You must specify the entire hierarchy of the realm, starting at the Top Level Realm. Prefix each realm in the hierarchy with the `realms/` keyword. For example, `/realms/root/realms/customers/realms/europe`.

+ To authenticate a user in the Top Level Realm, use the `root` keyword. For example:

+ https://openam.example.com:8443/openam/json/realms/root/authenticate

+ To authenticate a user in a subrealm named `alpha`, the REST endpoint would be:

+ https://openam.example.com:8443/openam/json/realms/root/realms/alpha/authenticate

+ If realms are specified using both the DNS alias and path methods, the path is used to determine the realm.

+ For example, the following REST endpoint returns users in a realm named `bravo`, not the realm with DNS alias `suppliers.example.com`:

```
https://suppliers.example.com:8443/openam/json/realms/root/realms/
bravo/users?_queryId=*
```

Authenticate to AM over REST

To authenticate to AM over REST, make an HTTP POST request to the `json/authenticate` endpoint. You must specify the entire hierarchy of the realm, starting at the Top Level Realm. Prefix each realm in the hierarchy with the `realms/` keyword. For example, `/realms/root/realms/customers/realms/europe`.

For authentication journeys where providing a user name and password is enough, you can log in to AM using a **`curl`** command similar to the following:

```
$ curl \
--request POST \
--header "Content-Type: application/json" \
--header "X-OpenAM-Username: demo" \
--header "X-OpenAM-Password: Ch4ng31t" \
--header "Accept-API-Version: resource=2.0, protocol=1.0" \
'https://openam.example.com:8443/openam/json/realms/root/realms/al
pha/authenticate'
{
    "tokenId": "AQIC5w…NTcy*",
    "successUrl": "/openam/console",
```

```
    "realm":"/alpha"
}
```

Note that the user name and password are sent in headers; this zero page login mechanism works only for name/password authentication.

AM returns a `tokenID` object that applications can present as a cookie value for other operations that require authentication. This object is called a session token.

In this example, AM uses the default authentication service configured for the realm. You can override the default by specifying authentication services and other options in the REST request. To support complex authentication journeys, such as multi-factor authentication, AM implements callback mechanisms.

For more information about how to authenticate, log out, and use AM session tokens, see Authenticate over REST.

REST API endpoints

REST API endpoints are discussed in detail in the following sections:

Authenticate over REST
: How to use the AM REST APIs to authenticate to AM.

Policies over REST*, *Policy sets over REST*, *Resource types over REST*, and *Policy set application types over REST
: How to use the AM REST APIs for policy management.

Request policy decisions over REST
: How to use the AM REST APIs for requesting authorization decisions from AM.

OAuth 2.0 endpoints
: How to use OAuth 2.0-specific endpoints to request access and refresh tokens, as well as introspecting and revoking them.

OAuth 2.0 administration REST endpoints
: How to use perform OAuth 2.0 administrative tasks, such as register, read, and delete clients.

OpenID Connect 1.0 endpoints
: How to use OpenID Connect-specific endpoints to retrieve information about an authenticated user, as well as validate ID tokens and check sessions.

Retrieve forgotten usernames*, *Reset forgotten passwords*, and *Register a user
: How to use the AM REST APIs for user self-registration and forgotten password reset.

Configure realms over REST

How to use the AM REST APIs for managing AM identities and realms.

Manage scripts (REST)
How to use the AM REST APIs to manage AM scripts.

Capture troubleshooting information
How to use the AM REST APIs to record information that can help you troubleshoot AM.

Consume REST STS instances* and *Query, validate, and cancel tokens
How to use the AM REST APIs to manage AM's Security Token Service, which lets you bridge identities across web and enterprise identity access management (IAM) systems through its token transformation process.

REST API auditing

AM supports two audit logging services: the Common REST-based Audit Logging service, and the legacy Logging service, which is based on a Java SDK.

Both audit facilities log AM REST API calls.

Common Audit Logging of REST API calls

AM logs information about all REST API calls to the `access` topic. For more information about AM audit topics, see Audit log topics.

Locate specific REST endpoints in the `http.path` log file property.

Legacy Logging of REST API calls

NOTE

This functionality is labeled as legacy.

AM logs information about REST API calls to two files:

amRest.access

Records accesses to a Common REST endpoint, regardless of whether the request successfully reached the endpoint through policy authorization.

An `amRest.access` example is as follows:

```
$ cat openam/var/audit/amRest.access
#Version: 1.0
```

```
#Fields: time Data LoginID ContextID IPAddr LogLevel Domain
LoggedBy MessageID ModuleName
NameID HostName
"2011-09-14 16:38:17"  /home/user/openam/var/audit/
"cn=dsameuser,ou=DSAME Users,o=openam"
aa307b2dcb721d4201 "Not Available" INFO o=openam
"cn=dsameuser,ou=DSAME Users,o=openam"
LOG-1 amRest.access "Not Available" 192.168.56.2
"2011-09-14 16:38:17" "Hello World" id=bjensen,ou=user,o=openam
8a4025a2b3af291d01 "Not Available"
INFO o=openam id=amadmin,ou=user,o=openam "Not Available"
amRest.access "Not Available"
192.168.56.2
```

amRest.authz

Records all Common REST authorization results regardless of success. If a request has an entry in the `amRest.access` log, but no corresponding entry in `amRest.authz`, then that endpoint was not protected by an authorization filter and therefore the request was granted access to the resource.

The `amRest.authz` file contains the `Data` field, which specifies the authorization decision, resource, and type of action performed on that resource. The `Data` field has the following syntax:

```
("GRANT"||"DENY") > "RESOURCE | ACTION"
```

- `GRANT >` is prepended to the entry if the request was allowed.
- `DENY >` is prepended to the entry if the request was not allowed.
- `RESOURCE` is *ResourceLocation* | *ResourceParameter*, where:
  - *ResourceLocation* is the endpoint location (for example, `subrealm/applicationtypes`).
  - *ResourceParameter* is the ID of the resource being touched (for example, `myApplicationType`) if applicable.

  Otherwise, this field is empty if touching the resource itself, such as in a query.
- `ACTION` is *ActionType* | *ActionParameter*, where:
  - *ActionType* is `CREATE||READ||UPDATE||DELETE||PATCH||ACTION||QUERY`.
  - *ActionParameter* is one of the following depending on the *ActionType*:
    - For `CREATE`: the new resource ID
    - For `READ`: empty

- For `UPDATE` : the revision of the resource to update
- For `DELETE` : the revision of the resource to delete
- For `PATCH` : the revision of the resource to patch
- For `ACTION` : the actual action performed (for example, "forgotPassword")
- For `QUERY` : the query ID if any

```
$ cat openam/var/audit/amRest.authz
#Version: 1.0
#Fields: time   Data   ContextID   LoginID   IPAddr   LogLevel   Domain
MessageID   LoggedBy   NameID
ModuleName     HostName
"2014-09-16 14:17:28"    /var/root/openam/var/audit/
7d3af9e799b6393301
"cn=dsameuser,ou=DSAME Users,dc=openam,dc=forgerock,dc=org" "Not
Available" INFO
dc=openam,dc=forgerock,dc=org   LOG-1   "cn=dsameuser,ou=DSAME
Users,dc=openam,dc=forgerock,dc=org"
"Not Available" amRest.authz     10.0.1.5
"2014-09-16 15:56:12"   GRANT >
sessions|ACTION|logout|AdminOnlyFilter   d3977a55a2ee18c201
id=amadmin,ou=user,dc=openam,dc=forgerock,dc=org "Not Available"
INFO   dc=openam,dc=forgerock,dc=org
OAuth2Provider-2   "cn=dsameuser,ou=DSAME
Users,dc=openam,dc=forgerock,dc=org"   "Not Available"
amRest.authz     127.0.0.1
"2014-09-16 15:56:40"    GRANT >
sessions|ACTION|logout|AdminOnlyFilter   eedbc205bf51780001
id=amadmin,ou=user,dc=openam,dc=forgerock,dc=org   "Not Available"
INFO dc=openam,dc=forgerock,dc=org
OAuth2Provider-2   "cn=dsameuser,ou=DSAME
Users,dc=openam,dc=forgerock,dc=org"   "Not Available"
amRest.authz     127.0.0.1
```

AM also provides additional information in its debug notifications for accesses to any endpoint, depending on the message type (error, warning or message) including realm, user, and result of the operation.