



Getting Started with REST

/ ForgeRock Access Management 7.1.4

Latest update: 7.1.4

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

Copyright © 2019-2021 ForgeRock AS.

Abstract

Introduction to using the REST interfaces provided by Access Management. ForgeRock Access Management. ForgeRock Access Management provides intelligent authentication, authorization, federation, and single sign-on functionality.



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

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

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

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

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

DejaVu Fonts

Bitstream Vera Fonts Copyright

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

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

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

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

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

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

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

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

Arev Fonts Copyright

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

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

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

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

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

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

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

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

FontAwesome Copyright

Copyright (c) 2017 by Dave Gandy, <https://fontawesome.com/>.

This Font Software is licensed under the SIL Open Font License, Version 1.1. See <https://opensource.org/licenses/OFL-1.1>.




Table of Contents

| | |
|--|----|
| Overview | iv |
| 1. Introducing REST in AM | 1 |
| About ForgeRock Common REST | 1 |
| 2. REST API Explorer | 20 |
| 3. REST API Versioning | 25 |
| Specifying REST API Versions | 25 |
| Supported REST API Versions | 26 |
| Configuring the Default REST API Version | 26 |
| Versioning Messages | 27 |
| 4. Specifying Realms in REST API Calls | 29 |
| 5. Authenticating to AM Using REST | 30 |
| 6. REST API Endpoints | 31 |
| 7. REST API Auditing | 33 |
| Common Audit Logging of REST API Calls | 33 |
| Legacy Logging of REST API Calls | 33 |
| Glossary | 35 |

Overview

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

Quick Start

| | | |
|---|--|--|
|  Getting Started with AM's REST APIs Learn about the ForgeRock® Common REST interface in the ForgeRock Identity Platform® and the specifics of REST in AM. |  API Explorer Access the online Access Management REST API reference through the AM console. |  REST Endpoints Discover the REST endpoints AM exposes. |
|---|--|--|

About ForgeRock Identity Platform™ Software

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

Chapter 1

Introducing 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 that access web resources and collections of web resources.

Interface Stability: *Evolving*¹

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.)

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 section describes the full Common REST framework. Some platform component products do not implement all Common REST behaviors exactly as described in this section. For details, refer to the product-specific examples and reference information in other sections of this documentation set.

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.

¹ For information on the endpoints deprecated or removed in previous versions, and their current equivalents, see the following <https://backstage.forgerock.com/knowledge/kb> article.

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 as described in the following steps:

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

In other words, the software should be configured as in production so that the APIs are identical to what developers see in production.

2. Retrieve the OpenAPI-compliant descriptor.

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

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

3. (Optional) If necessary, edit the descriptor.

For example, you might want to add security definitions to describe how the API is protected.

If you make any changes, then also consider using a source control system to manage your versions of the API descriptor.

4. Publish the descriptor using a tool such as [Swagger UI](#).

You can customize Swagger UI for your organization as described in the documentation for the tool.

Create

There are two ways to create a resource, either with an HTTP POST or with an 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 create request indicates the resource location as the value of the `Location` header.

If you include the `If-None-Match` header, its value must be `*`. In this case, the request creates the object if it does not exist, and fails if the object does exist. If you include the `If-None-Match` header with any value other than `*`, 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

You can use the following parameters:

`_prettyPrint=true`

Format the body of the response.

`_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.

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

You can use the following parameters:

`_prettyPrint=true`

Format the body of the response.

`_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 for example.

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.

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

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 be retained. 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.

Parameters

You can use the following parameters:

`_prettyPrint=true`

Format the body of the response.

`_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.

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

You can use the following parameters:

`_prettyPrint=true`

Format the body of the response.

`_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.

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:

```
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 several different **operations**. The following sections show each of these operations, along with options for the **field** and **value**:

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. Examples of a single-valued field include: 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" ]
}
```

Note that 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 field value and the target field value are configured as 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. It is equivalent to performing 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 in the same fashion 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 `URLConnection.setRequestMethod("PATCH")` throws `ProtocolException`.

Parameters

You can use the following parameters. Other parameters might depend on the specific action implementation:

_prettyPrint=true

Format the body of the response.

`_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.

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

You can use the following parameters. Other parameters might depend on the specific action implementation:

`_prettyPrint=true`

Format the body of the response.

`_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.

Query

To query a resource collection (or resource container if you prefer to think of it that way), perform an HTTP GET and accept a JSON response, including at least a `_queryExpression`, `_queryFilter`, or `_queryId` parameter. These 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 related to the query string parameters that you specify.

Parameters

You can use the following parameters:

`_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, be aware that 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 need URL encoding 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     = "!" / "$" / "&" / "'" / "(" / ")"
                / "*" / "+" / "," / ";" / "="
```

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.
- 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 that 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 when used with the `_queryExpression` and `_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.

`_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 `+`, it must be URL-encoded as `%2B`, for example:

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

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

`_prettyPrint=true`

Format the body of the response.

`_fields=field[, field...]`

Return only the specified fields in each element of the "results" array in 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.

HTTP Status Codes

When working with a Common REST API over HTTP, client applications should expect at least the following 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.

Chapter 2

REST API Explorer

AM provides an online AM REST API reference that can be accessed through the AM console. It displays the REST API endpoints that let client applications access AM's services.

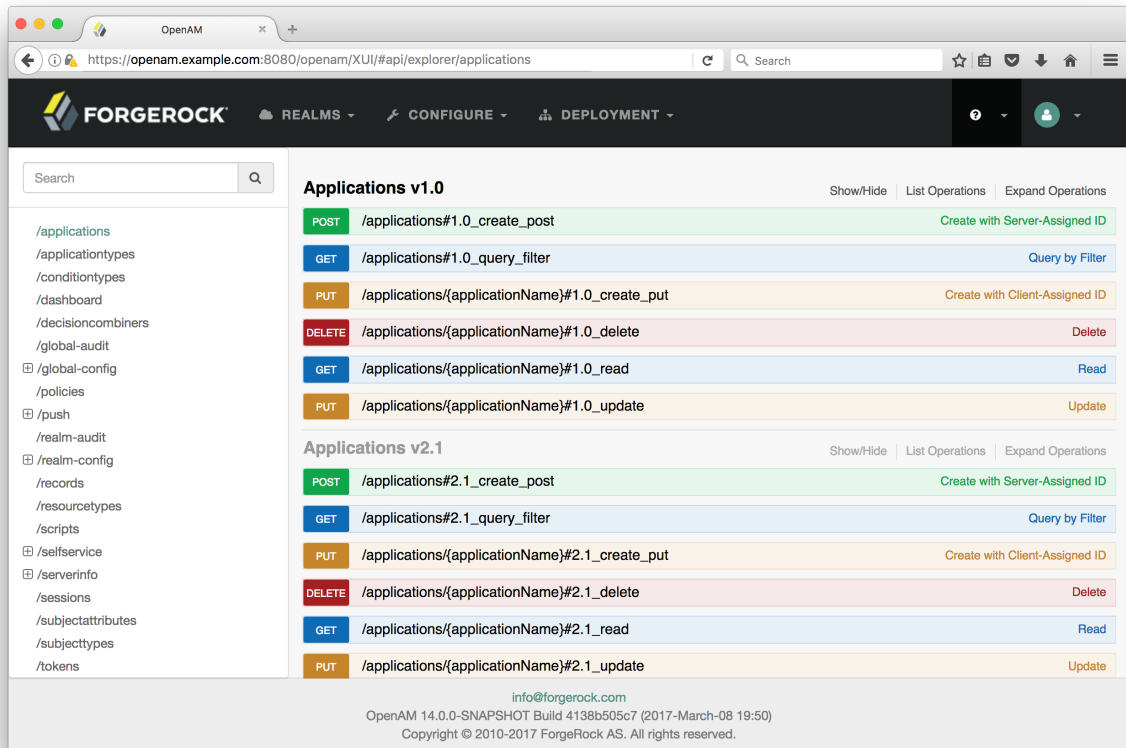
Caution

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:

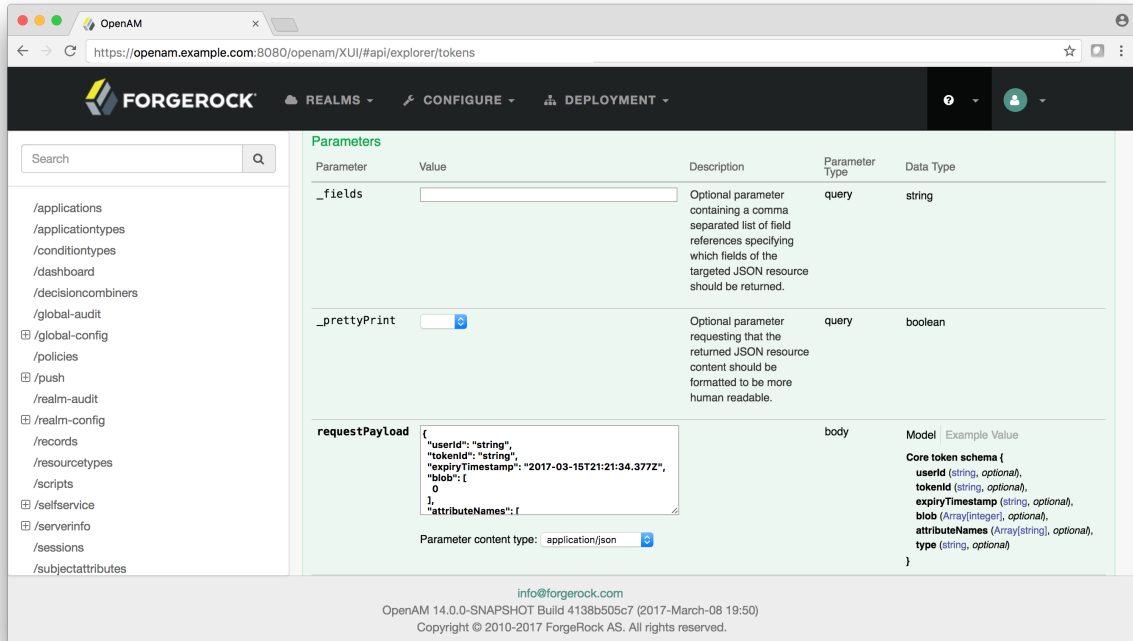
- **API Versioning.** The API Explorer displays the different API versions available depending on your deployment.

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, the `requestPayload` field can be populated with an example value. Also, if you select `Model`, you can view the schema for each parameter, as seen below:

API Explorer Request Payload



The screenshot shows the OpenAM REST API Explorer interface. The browser address bar displays `https://openam.example.com:8080/openam/XUI/#api/explorer/tokens`. The interface includes a sidebar with a search bar and a list of API endpoints. The main area displays the parameters for the selected endpoint.

Parameters

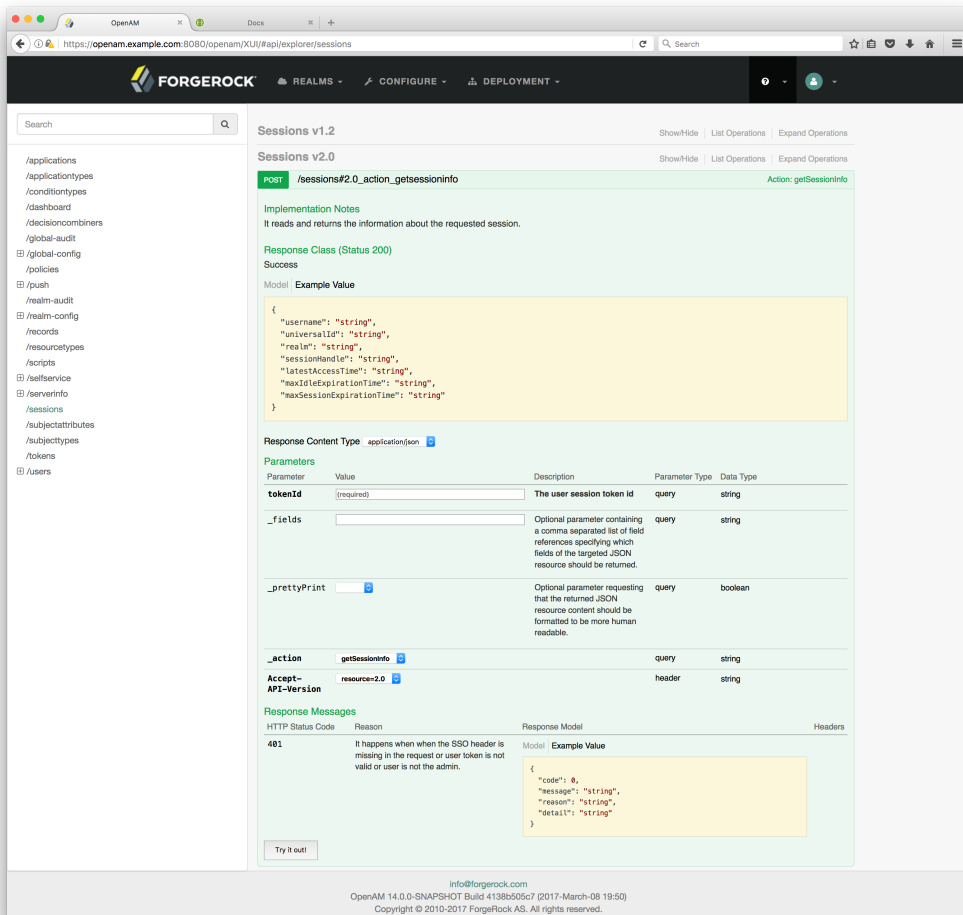
| Parameter | Value | Description | Parameter Type | Data Type |
|-----------------------------|---|--|----------------|---|
| <code>_fields</code> | <input type="text"/> | Optional parameter containing a comma separated list of field references specifying which fields of the targeted JSON resource should be returned. | query | string |
| <code>_prettyPrint</code> | <input type="checkbox"/> | Optional parameter requesting that the returned JSON resource content should be formatted to be more human readable. | query | boolean |
| <code>requestPayload</code> | <pre>{ "userid": "string", "tokenId": "string", "expiryTimestamp": "2017-03-15T21:21:34.377Z", "blob": { 0 }, "attributeNames": f }</pre> | | body | Model Example Value Core token schema { userid (string, optional), tokenId (string, optional), expiryTimestamp (string, optional), blob (Array<integer>, optional), attributeNames (Array<string>, optional), type (string, optional) } |

Parameter content type: `application/json`

info@forgerock.com
 OpenAM 14.0.0-SNAPSHOT Build 4138b505c7 (2017-March-08 19:50)
 Copyright © 2010-2017 ForgeRock AS. All rights reserved.

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

API Explorer Detailed Information



The screenshot displays the ForgeRock REST API Explorer interface. The left sidebar shows a tree view of API endpoints, with `/sessions` selected. The main panel shows the details for the `POST /sessions/v2.0/action_getsessioninfo` endpoint. The `Implementation Notes` section states: "It reads and returns the information about the requested session." The `Response Class (Status 200)` section shows a success response with a JSON model example:

```
{
  "username": "string",
  "universalId": "string",
  "realm": "string",
  "sessionHandle": "string",
  "lastAccessTime": "string",
  "maxIdleExpirationTime": "string",
  "maxSessionExpirationTime": "string"
}
```

The `Response Content Type` is `application/json`. The `Parameters` section lists the following:

| Parameter | Value | Description | Parameter Type | Data Type |
|---------------------------------|-----------------------------|--|----------------|-----------|
| <code>tokenId</code> | (required) | The user session token id | query | string |
| <code>_fields</code> | | Optional parameter containing a comma separated list of field references specifying which fields of the targeted JSON resource should be returned. | query | string |
| <code>_prettyPrint</code> | <input type="checkbox"/> | Optional parameter requesting that the returned JSON resource content should be formatted to be more human readable. | query | boolean |
| <code>_action</code> | <code>getSessionInfo</code> | | query | string |
| <code>Accept-API-Version</code> | <code>resource=2.0</code> | | header | string |

The `Response Messages` section shows an error response for status 401:

| HTTP Status Code | Reason | Response Model | Headers |
|------------------|--|---------------------|---------|
| 401 | It happens when when the SSO header is missing in the request or user token is not valid or user is not the admin. | Model Example Value | |

```
{
  "code": 0,
  "message": "string",
  "reason": "string",
  "detail": "string"
}
```

At the bottom of the interface, there is a "Try it out!" button and footer information: "info@forgerock.com", "OpenAM 14.0.0-SNAPSHOT Build 4138b050c7 (2017-March-08 19:50)", and "Copyright © 2010-2017 ForgeRock AS. All rights reserved."

Note the following when using the *Try It Out* feature in the API Explorer:

- The example payload values are auto-generated, and whilst they may be the correct data type, their value may not be correct for the API to function correctly. Check the *Model* tab for a description of the value required, 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 if you need to target a different realm.

For more information, see "*Specifying Realms in REST API Calls*".

To Access the API Explorer

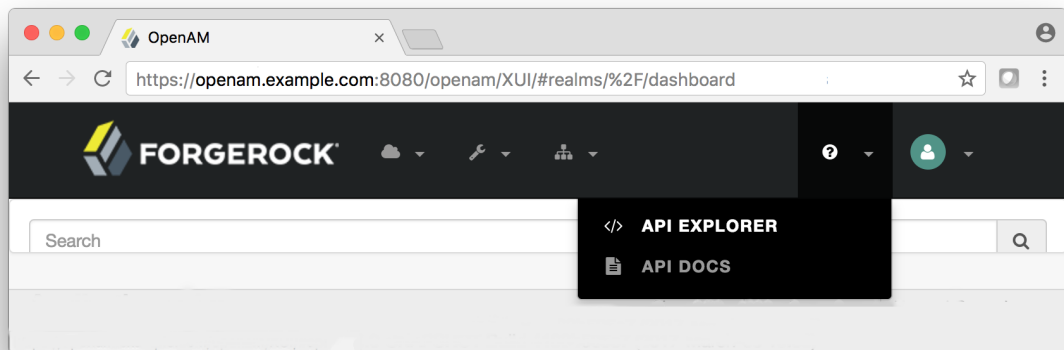
- From the AM console, you can access the API Explorer in one of two ways:

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 then click API Explorer.

API Explorer



Chapter 3

REST API Versioning

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`

For more information about the filter, see "Cross-Site Request Forgery (CSRF) Protection" in the *Security Guide*.

Specifying 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 does not specify explicit version information. For more information, see "Configuring the Default REST API Version".

Supported REST API Versions

For information about the supported protocol and resource versions available in AM, see the [API Explorer](#) available in the AM console.

The *AM Release Notes* section, [Changes](#), describes any breaking changes between API versions.

Configuring the Default REST API Version

Configure how AM behaves to REST calls that are not presenting API versions:

Configure Versioning Behavior

1. Log in as AM administrator, `amAdmin`.
2. Click Configure > Global Services, and then click REST APIs.
3. In Default Version, select the required response to a REST API request that does not 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 may not be the first that was released, as APIs versions become deprecated or unsupported. Refer to [Deprecated](#).

None

No version will be used. When a REST client application calls a REST API without specifying the version, AM returns an error and the request fails.

4. (Optional) Optionally, enable **Warning Header** to include warning messages in the headers of responses to requests.
5. Save your work.

Versioning 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 not specified 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: Ch4ng3!t" \
--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...TU30Q*",
  "successUrl": "/openam/console"
}
```

If the default REST API version behavior is set to **None**, and a REST API call does not include the **Accept-API-Version** header, or does not specify a **resource** version, then a **400 Bad Request** status code is returned, as shown below:

```
$ curl \
--header "Content-Type: application/json" \
--header "Accept-API-Version: protocol=1.0" \
https://openam.example.com:8443/openam/json/realms/root/serverinfo/*
{
  "code":400,
  "reason":"Bad Request",
  "message":"No requested version specified and behavior set to NONE."
}
```

If a REST API call does include the `Accept-API-Version` header, but the specified `resource` or `protocol` version does not exist in AM, then a `404 Not Found` status code is returned, as shown below:

```
$ 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":"Accept-API-Version: Requested version \"999.0\" does not match any routes."
}
```

Tip

For more information on setting the default REST API version behavior, see "Specifying REST API Versions".

Chapter 4

Specifying 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, the REST endpoint would be:

```
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 realm named `alpha` within the Top Level Realm, 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=*
```

Chapter 5

Authenticating to AM Using REST

To authenticate to AM using 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/alpha/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 *"Authenticating (REST)"* in the *Authentication and Single Sign-On Guide*.

Chapter 6

REST API Endpoints

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

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

How to use the AM REST APIs to authenticate to AM.

Policies (REST) in the *Authorization Guide*, Policy Sets (REST) in the *Authorization Guide*, Resource Types (REST) in the *Authorization Guide*, and Policy Set Application Types (REST) in the *Authorization Guide*

How to use the AM REST APIs for policy management.

Requesting Policy Decisions Using REST in the *Authorization Guide*

How to use the AM REST APIs for requesting authorization decisions from AM.

OAuth 2.0 Endpoints in the *OAuth 2.0 Guide*

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 and Supporting REST Endpoints in the *OAuth 2.0 Guide*

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

OpenID Connect 1.0 Endpoints in the *OpenID Connect 1.0 Guide*

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

"Retrieving Forgotten Usernames" in the *User Self-Service Guide*, "Resetting Forgotten Passwords" in the *User Self-Service Guide*, and "Registering Users" in the *User Self-Service Guide*

How to use the AM REST APIs for user self-registration and forgotten password reset.

Configuring Realms (REST) in the *Setup Guide*

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

Managing Scripts (REST) in the *Getting Started with Scripting*

How to use the AM REST APIs to manage AM scripts.

Recording Troubleshooting Information in the *Maintenance Guide*

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

Consuming REST STS Instances in the *Security Token Service (STS) Guide* and "*Querying, Validating, and Canceling Tokens*" in the *Security Token Service (STS) Guide*

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.

Chapter 7

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" in the *Security Guide*.

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"
aa307b2dc721d4201 "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"
```

where

- "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 (e.g., subrealm/applicationtypes)
 - "ResourceParameter" is the ID of the resource being touched (e.g., 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 (e.g., "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.

Glossary

| | |
|---------------------|---|
| Access control | Control to grant or to deny access to a resource. |
| Account lockout | The act of making an account temporarily or permanently inactive after successive authentication failures. |
| Actions | Defined as part of policies, these verbs indicate what authorized identities can do to resources. |
| Advice | In the context of a policy decision denying access, a hint to the policy enforcement point about remedial action to take that could result in a decision allowing access. |
| Agent administrator | User having privileges only to read and write agent profile configuration information, typically created to delegate agent profile creation to the user installing a web or Java agent. |
| Agent authenticator | Entity with read-only access to multiple agent profiles defined in the same realm; allows an agent to read web service profiles. |
| Application | <p>In general terms, a service exposing protected resources.</p> <p>In the context of AM policies, the application is a template that constrains the policies that govern access to protected resources. An application can have zero or more policies.</p> |
| Application type | <p>Application types act as templates for creating policy applications.</p> <p>Application types define a preset list of actions and functional logic, such as policy lookup and resource comparator logic.</p> |

| | |
|---------------------------------------|--|
| | Application types also define the internal normalization, indexing logic, and comparator logic for applications. |
| Attribute-based access control (ABAC) | Access control that is based on attributes of a user, such as how old a user is or whether the user is a paying customer. |
| Authentication | The act of confirming the identity of a principal. |
| Authentication chaining | A series of authentication modules configured together which a principal must negotiate as configured in order to authenticate successfully. |
| Authentication level | Positive integer associated with an authentication module, usually used to require success with more stringent authentication measures when requesting resources requiring special protection. |
| Authentication module | AM authentication unit that handles one way of obtaining and verifying credentials. |
| Authorization | The act of determining whether to grant or to deny a principal access to a resource. |
| Authorization Server | In OAuth 2.0, issues access tokens to the client after authenticating a resource owner and confirming that the owner authorizes the client to access the protected resource. AM can play this role in the OAuth 2.0 authorization framework. |
| Auto-federation | Arrangement to federate a principal's identity automatically based on a common attribute value shared across the principal's profiles at different providers. |
| Bulk federation | Batch job permanently federating user profiles between a service provider and an identity provider based on a list of matched user identifiers that exist on both providers. |
| Circle of trust | Group of providers, including at least one identity provider, who have agreed to trust each other to participate in a SAML v2.0 provider federation. |
| Client | In OAuth 2.0, requests protected web resources on behalf of the resource owner given the owner's authorization. AM can play this role in the OAuth 2.0 authorization framework. |
| Client-based OAuth 2.0 tokens | After a successful OAuth 2.0 grant flow, AM returns a token to the client. This differs from CTS-based OAuth 2.0 tokens, where AM returns a <i>reference</i> to token to the client. |
| Client-based sessions | AM sessions for which AM returns session state to the client after each request, and require it to be passed in with the subsequent |

request. For browser-based clients, AM sets a cookie in the browser that contains the session information.

For browser-based clients, AM sets a cookie in the browser that contains the session state. When the browser transmits the cookie back to AM, AM decodes the session state from the cookie.

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 AM configuration data.

Cross-domain single sign-on (CDSSO)

AM capability allowing single sign-on across different DNS domains.

CTS-based OAuth 2.0 tokens

After a successful OAuth 2.0 grant flow, AM returns a *reference* to the token to the client, rather than the token itself. This differs from [client-based OAuth 2.0 tokens](#), where AM returns the entire token to the client.

CTS-based sessions

AM [sessions](#) that reside in the Core Token Service's token store. CTS-based sessions might also be cached in memory on one or more AM servers. AM tracks these sessions in order to handle events like logout and timeout, to permit session constraints, and to notify applications involved in SSO when a session ends.

Delegation

Granting users administrative privileges with AM.

Entitlement

Decision that defines which resource names can and cannot be accessed for a given identity in the context of a particular application, which actions are allowed and which are denied, and any related advice and attributes.

Extended metadata

Federation configuration information specific to AM.

Extensible Access Control Markup Language (XACML)

Standard, XML-based access control policy language, including a processing model for making authorization decisions based on policies.

Federation

Standardized means for aggregating identities, sharing authentication and authorization data information between trusted providers, and

| | |
|-----------------------------------|--|
| | allowing principals to access services across different providers without authenticating repeatedly. |
| Fedlet | Service provider application capable of participating in a circle of trust and allowing federation without installing all of AM on the service provider side; AM lets you create Java Fedlets. |
| Hot swappable | Refers to configuration properties for which changes can take effect without restarting the container where AM runs. |
| Identity | Set of data that uniquely describes a person or a thing such as a device or an application. |
| Identity federation | Linking of a principal's identity across multiple providers. |
| Identity provider (IDP) | Entity that produces assertions about a principal (such as how and when a principal authenticated, or that the principal's profile has a specified attribute value). |
| Identity repository | Data store holding user profiles and group information; different identity repositories can be defined for different realms. |
| Java agent | Java web application installed in a web container that acts as a policy enforcement point, filtering requests to other applications in the container with policies based on application resource URLs. |
| Metadata | Federation configuration information for a provider. |
| Policy | Set of rules that define who is granted access to a protected resource when, how, and under what conditions. |
| Policy agent | Java, web, or custom agent that intercepts requests for resources, directs principals to AM for authentication, and enforces policy decisions from AM. |
| Policy Administration Point (PAP) | Entity that manages and stores policy definitions. |
| Policy Decision Point (PDP) | Entity that evaluates access rights and then issues authorization decisions. |
| Policy Enforcement Point (PEP) | Entity that intercepts a request for a resource and then enforces policy decisions from a PDP. |
| Policy Information Point (PIP) | Entity that provides extra information, such as user profile attributes that a PDP needs in order to make a decision. |
| Principal | Represents an entity that has been authenticated (such as a user, a device, or an application), and thus is distinguished from other entities. |

| | |
|---|---|
| | When a Subject successfully authenticates, AM associates the Subject with the Principal . |
| Privilege | In the context of delegated administration, a set of administrative tasks that can be performed by specified identities in a given realm. |
| Provider federation | Agreement among providers to participate in a circle of trust. |
| Realm | AM unit for organizing configuration and identity information. Realms can be used for example when different parts of an organization have different applications and identity stores, and when different organizations use the same AM deployment. Administrators can delegate realm administration. The administrator assigns administrative privileges to users, allowing them to perform administrative tasks within the realm. |
| Resource | Something a user can access over the network such as a web page. Defined as part of policies, these can include wildcards in order to match multiple actual resources. |
| Resource owner | In OAuth 2.0, entity who can authorize access to protected web resources, such as an end user. |
| Resource server | In OAuth 2.0, server hosting protected web resources, capable of handling access tokens to respond to requests for such resources. |
| Response attributes | Defined as part of policies, these allow AM to return additional information in the form of "attributes" with the response to a policy decision. |
| Role based access control (RBAC) | Access control that is based on whether a user has been granted a set of permissions (a role). |
| Security Assertion Markup Language (SAML) | Standard, XML-based language for exchanging authentication and authorization data between identity providers and service providers. |
| Service provider (SP) | Entity that consumes assertions about a principal (and provides a service that the principal is trying to access). |
| Authentication Session | The interval while the user or entity is authenticating to AM. |
| Session | The interval that starts after the user has authenticated and ends when the user logs out, or when their session is terminated. For browser-based clients, AM manages user sessions across one or more applications by setting a session cookie. See also CTS-based sessions and Client-based sessions. |

| | |
|---------------------------|---|
| Session high availability | Capability that lets any AM server in a clustered deployment access shared, persistent information about users' sessions from the CTS token store. The user does not need to log in again unless the entire deployment goes down. |
| Session token | Unique identifier issued by AM after successful authentication. For a CTS-based sessions , the session token is used to track a principal's session. |
| Single log out (SLO) | Capability allowing a principal to end a session once, thereby ending her session across multiple applications. |
| Single sign-on (SSO) | Capability allowing a principal to authenticate once and gain access to multiple applications without authenticating again. |
| Site | <p>Group of AM servers configured the same way, accessed through a load balancer layer. The load balancer handles failover to provide service-level availability.</p> <p>The load balancer can also be used to protect AM services.</p> |
| Standard metadata | Standard federation configuration information that you can share with other access management software. |
| Stateless Service | <p>Stateless services do not store any data locally to the service. When the service requires data to perform any action, it requests it from a data store. For example, a stateless authentication service stores session state for logged-in users in a database. This way, any server in the deployment can recover the session from the database and service requests for any user.</p> <p>All AM services are stateless unless otherwise specified. See also Client-based sessions and CTS-based sessions.</p> |
| Subject | <p>Entity that requests access to a resource</p> <p>When an identity successfully authenticates, AM associates the identity with the Principal that distinguishes it from other identities. An identity can be associated with multiple principals.</p> |
| Identity store | Data storage service holding principals' profiles; underlying storage can be an LDAP directory service or a custom IdRepo implementation. |
| Web Agent | Native library installed in a web server that acts as a policy enforcement point with policies based on web page URLs. |