REST API Reference

/ ForgeRock Identity Management 7.1

Latest update: 7.1.0

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

Abstract

Guide to creating and managing objects in ForgeRock® Identity Management.

Table of Contents

Overview

This reference describes the ForgeRock Common REST API. See "Common REST and IDM" for information specific to the IDM implementation of Common REST.

Quick Start

Start Here Learn about the Common REST interface in the ForgeRock Platform and the specifics of REST in IDM.	API Explorer Access the online IDM REST API reference through the Admin UI.
REST API Structure Understand RESTful syntax with respect to the IDM REST API.	REST Endpoints Discover the REST endpoints IDM 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.

The ForgeRock Common REST API works across the platform to provide common ways to access web resources and collections of resources.

Chapter 1
REST and IDM

Representational State Transfer (REST) is a software architecture style for exposing resources, using the technologies and protocols of the World Wide Web. REST describes how distributed data objects, or resources, can be defined and addressed.

IDM provides a RESTful API for accessing managed objects, system objects, workflows, and the system configuration.

- "About ForgeRock Common REST"
- "Common REST and IDM"

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.

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 `HttpURLConnection.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.

Common REST and IDM

IDM implements the Common REST API as described in the previous section, with the exception of the following elements:

- IDM provides limited support for the `in` expression clause. You can use this clause for queries on singleton string properties, not arrays. `in` query expressions are not supported through the Admin UI.

- The PATCH `transform` action is supported only on the `config` endpoint. Note that this is an optional action and not implemented everywhere across the ForgeRock Identity Platform.

- Common REST supports PATCH operations by list element index, as shown in the example in "Patch Operation: Remove". IDM *does not support* PATCH by list element index. So, for PATCH operations, you cannot use an ordinal when adding or removing list items.

  You can add an item using the special hyphen index, which designates that the element should be added to the end of the list. To remove specific items from a list, you must specify the *value* to be removed, for example:

```
[
    {
        "operation" : "remove",
        "field" : "/phoneNumber/",
        "value" : "202-555-0185"
    }
]
```

Note

When you remove items in this way, if the list contains two or more items with the same value, they are all removed.

- If `_fields` is left blank (null), the server returns all default values. In IDM, this excludes relationships and virtual fields. To include these fields in the output, add `"returnByDefault" : true` in the applicable schema.

  IDM also implements wild-card (`*`) handling with the `_fields` parameter. So, a value of `_fields=*_ref` will return all relationship fields associated with an object. A value of `_fields=*_ref/*` will return all the fields within each relationship.

- IDM does not implement the `ESTIMATE` total paged results policy. The `totalPagedResults` is either the exact total result count (`_totalPagedResultsPolicy=EXACT`) or result counting is disabled (`_totalPagedResultsPolicy=NONE`). For more information, see "Page Query Results" in the *Object Modeling Guide*.

Chapter 2
REST API Explorer

IDM includes an API Explorer, an implementation of the *OpenAPI Initiative Specification*, also known as Swagger.

To access the API Explorer, log in to the Admin UI, select the question mark in the upper right corner, and choose API Explorer from the drop-down menu.

Note

If the API Explorer does not appear, you might need to enable it in your `resolver/boot.properties` file by setting the `openidm.apidescriptor.enabled` property to `true`.

The API Explorer covers most of the endpoints provided with a default IDM installation.

Each endpoint lists supported HTTP methods, such as POST and GET. When custom actions are available, the API Explorer lists them as *HTTP Method* `/path/to/endpoint?_action=something`.

To see how this works, navigate to the `User` endpoint, select List Operations, and choose the GET option associated with the `/managed/user#_query_id_query-all` endpoint.

In this case, the defaults are set, and all you need to do is select the `Try it out!` button. The output you see includes:

- The REST call, in the form of the **curl** command.
- The request URL, which specifies the endpoint and associated parameters.
- The response body, which contains the data that you requested.
- The HTTP response code; if everything works, this should be `200`.
- Response headers.

Tip

If you see a `401 Access Denied` code in the response body, your session may have timed out, and you'll have to log in to the Admin UI again.

For details on common ForgeRock REST parameters, see "About ForgeRock Common REST".

You'll see examples of REST calls throughout this documentation set. You can try these calls with the API Explorer.

You can also generate an OpenAPI-compliant descriptor of the REST API to provide API reference documentation specific to your deployment. The following command saves the API descriptor of the managed/user endpoint to a file named `my-openidm-api.json`:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
--output "my-openidm-api.json" \
"http://localhost:8080/openidm/managed/user?_api"
```

For information about publishing reference documentation using the API descriptor, see "To Publish OpenAPI Documentation".

Chapter 3 REST API Versioning

ForgeRock 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 ForgeRock introduces a change that is not backwards-compatible, and that affects clients that use the feature.

If there is more than one version of the API, you must select the version by setting a version header that specifies which version of the *resource* is requested. For information about the supported resource versions, see the "*REST API Explorer*" in the Admin UI. To ensure that your clients are always compatible with a newer IDM version, you should always include resource versions in your REST calls.

Specifying the API Version in REST Calls

HTTP requests can optionally include the `Accept-API-Version` header with the value of the resource version, such as `resource=2.0`. If no `Accept-API-Version` header is included, the *latest* resource version is invoked by the HTTP request.

The following call requests version `2.0` of the specified resource:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Content-Type: application/json" \
--header "Accept-API-Version: resource=2.0" \
--request POST \
--data '{
  "url":"https://www.forgerock.com/favicon.ico",
  "method":"GET"
}' \
"http://localhost:8080/openidm/external/rest?_action=call"
```

Specifying the API Version in Scripts

You can specify a resource version in JavaScript and Groovy scripts using the fourth (*additional parameters*) argument. If present, the `Accept-API-Version` parameter is applied to the actual REST request. Any other parameters are set as Additional Parameters on the request.

The following examples request specific resource versions:

REST with Inline Javascript

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Content-Type: application/json" \
--request POST \
--data '{
  "type":"text/javascript",
  "source":"openidm.action(\"external/rest\", \"call\", {\"url\": \"https://www.forgerock.com/
favicon.ico\", \"method\": \"GET\"}, {\"Accept-API-Version\": \"resource=1.0\"});"
}' \
"http://localhost:8080/openidm/script?_action=eval"
```

Standalone Javascript

```
openidm.action("external/rest", "call",
 {"url": "https://www.forgerock.com", "method": "GET"},
 {"Accept-API-Version": "resource=1.0"});
```

Chapter 4
REST API Structure

URI Scheme

The URI scheme for accessing a managed object follows this convention, assuming the IDM web application was deployed at `/openidm`.

```
/openidm/managed/type/id
```

Similar schemes exist for URIs associated with all but system objects. For more information, see "Configure Access Control in `access.json`" in the *Authentication and Authorization Guide*.

The URI scheme for accessing a system object follows this convention:

```
/openidm/system/resource-name/type/id
```

An example of a system object in an LDAP directory might be:

```
/openidm/system/ldap/account/07b46858-56eb-457c-b935-cfe6ddf769c7
```

Important

For LDAP resources, you should *not* map the LDAP `dn` to the IDM `uidAttribute` (`_id`). The attribute that is used for the `_id` should be immutable. You should therefore map the LDAP `entryUUID` operational attribute to the IDM `_id`, as shown in the following excerpt of the provisioner configuration file:

```
...
"uidAttribute" : "entryUUID",
...
```

Object Identifiers

Every managed and system object has an identifier (expressed as *id* in the URI scheme) that is used to address the object through the REST API. The REST API allows for client-generated and server-generated identifiers, through PUT and POST methods. The default server-generated identifier type is a UUID. If you create an object by using `POST`, a server-assigned ID is generated in the form of a UUID. If you create an object by using PUT, the client assigns the ID in whatever format you specify.

Most of the examples in this guide use client-assigned IDs, as it makes the examples easier to read.

Content Negotiation

The REST API fully supports negotiation of content representation through the `Accept` HTTP header. Currently, the supported content type is JSON. When you send a JSON payload, you must include the following header:

```
Accept: application/json
```

In a REST call (using the **curl** command, for example), you would include the following option to specify the noted header:

```
--header "Content-Type: application/json"
```

You can also specify the default UTF-8 character set as follows:

```
--header "Content-Type: application/json;charset=utf-8"
```

The `application/json` content type is not needed when the REST call does not send a JSON payload.

Conditional Operations

The REST API supports conditional operations through the use of the `ETag`, `If-Match` and `If-None-Match` HTTP headers. The use of HTTP conditional operations is the basis of IDM's optimistic concurrency control system. Clients should make requests conditional in order to prevent inadvertent modification of the wrong version of an object. For *managed objects*, if no conditional header is specified, a default of `If-Match: *` is applied.

REST API Conditional Operations

HTTP Header	Operation	Description
`If-Match: <rev>`	PUT	Update the object if the <rev> matches the revision level of the object.
`If-Match: *`	PUT	Update the object regardless of revision level
`If-None-Match: <rev>`		Bad request
`If-None-Match: *`	PUT	Create; fails if the object already exists
When the conditional operations `If-Match`, `If-None-Match` are not used	PUT	Upsert; attempts a create, and then an update; if both attempts fail, return an error

Chapter 5
REST Endpoints and Sample Commands

This chapter describes the REST endpoints and provides a number of sample commands that show the interaction with the REST interface.

Server Configuration

IDM stores configuration objects in the repository, and exposes them under the context path `/openidm/config`. Single instance configuration objects are exposed under `/openidm/config/`*`object-name`*.

Multiple instance configuration objects are exposed under `/openidm/config/object-name/instance-name`. The following table outlines these configuration objects and how they can be accessed through the REST interface.

URI	HTTP Operation	Description
/openidm/config	GET	Returns a list of configuration objects
/openidm/config/access	GET	Returns the current access configuration
/openidm/config/audit	GET	Returns the current audit configuration
/openidm/config/provisioner.openicf/*provisioner-name*	GET	Returns the configuration of the specified connector
/openidm/config/selfservice/*function*	GET	Returns the configuration of the specified self-service feature, `registration`, `reset`, or `username`
/openidm/config/router	PUT	Changes the router configuration. Modifications are provided with the `--data` option, in JSON format.
/openidm/config/*object*	PATCH	Changes one or more fields of the specified configuration object. Modifications are provided as a JSON array of patch operations.
/openidm/config/*object*	DELETE	Deletes the specified configuration object.
/openidm/config/*object*?_queryFilter=*query*	GET	Queries the specified configuration object. You cannot create custom predefined queries to query the configuration.

IDM supports REST operations to create, read, update, query, and delete configuration objects.

For command-line examples of managing the configuration over REST, see "Configure the Server Over REST" in the *Setup Guide*.

One entry is returned for each configuration object. To obtain additional information on the configuration object, include its `pid` or `_id` in the URL. The following example displays configuration information on the `sync` object, based on a deployment using the `sync-with-csv` sample:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/config/sync"
{
  "_id": "sync",
  "mappings": [
    {
      "name": "systemCsvfileAccounts_managedUser",
      "source": "system/csvfile/account",
      "target": "managed/user",
      "correlationQuery": {
        "type": "text/javascript",
        "source": "var query = {'_queryId' : 'for-userName', 'uid' : source.name};query;"
      },
      "properties": [
        {
          "source": "email",
          "target": "mail"
        },
        {
          "source": "firstname",
          "target": "givenName"
        },
        {
          "source": "lastname",
          "target": "sn"
        },
        {
          "source": "description",
          "target": "description"
        },
        {
          "source": "_id",
          "target": "_id"
        },
        {
          "source": "name",
          "target": "userName"
        },
        {
          "default": "Passw0rd",
          "target": "password"
        },
        {
          "source": "mobileTelephoneNumber",
          "target": "telephoneNumber"
        },
        {
          "source": "roles",
          "transform": {
            "type": "text/javascript",
            "source": "var _ = require('lib/lodash'); _.map(source.split(','), function(role)
            { return {'_ref': 'internal/role/' + role} });"
          },
          "target": "authzRoles"
        }
      ],

...
```

Managed Users

User objects are stored in the repository and are exposed under the context path `/managed/user`. Many examples of REST calls related to this context path exist throughout this document. The following table lists available functionality associated with the `/managed/user` context path.

URI	HTTP Operation	Description
/openidm/managed/user?_queryFilter=true&_fields=_id	GET	Lists the IDs of all the managed users in the repository.
/openidm/managed/user?_queryFilter=true	GET	Lists all info for the managed users in the repository.
/openidm/managed/user?_queryFilter=*filter*	GET	Queries the managed user object with the defined filter.
/openidm/managed/user/*_id*	GET	Returns the JSON representation of a specific user.
/openidm/managed/user/*_id*	PUT	Creates a new user.
/openidm/managed/user/*_id*	PUT	Updates a user entry (replaces the entire entry).
/openidm/managed/user?_action=create	POST	Creates a new user.
/openidm/managed/user?_action=patch&_queryId=for-userName&uid=*userName*	POST	Updates a user (can be used to replace the value of one or more existing attributes).
/openidm/managed/user/*_id*	PATCH	Updates specified fields of a user entry.
/openidm/managed/user/*_id*	DELETE	Deletes a user entry.

For a number of sample commands that show how to manage users over REST, see "Managed Users" in the *Object Modeling Guide*.

Managed Organizations

Organizations are exposed under the context path `/managed/organization`. The following table lists the REST commands associated with managed organizations.

URI	HTTP Operation	Description
/openidm/managed/organization?_queryFilter=true&_fields=_id	GET	Lists the IDs of all managed organizations.
/openidm/managed/organization?_queryFilter=*filter*	GET	Queries managed organizations with the defined filter.
/openidm/managed/organization/*_id*	GET	Returns the JSON representation of a specific organization.
/openidm/managed/organization/*_id*	PUT	Creates an organization with a user-defined ID.
/openidm/managed/organization/*_id*	PUT	Updates an organization (replaces the entire object).

URI	HTTP Operation	Description
/openidm/managed/organization?_action=create	POST	Creates a new organization with a system-generated ID.
/openidm/managed/organization/*_id*	DELETE	Deletes an organization.

For a number of sample commands that show how to manage organizations over REST, see "Managed Organizations" in the *Object Modeling Guide*.

System Objects

System objects, that is, objects that are stored in remote systems, are exposed under the `/openidm/system` context. IDM provides access to system objects over REST, as listed in the following table:

URI	HTTP Operation	Description
/openidm/system?_action=*action-name*	POST	`_action=availableConnectors` returns a list of the connectors that are available in `openidm/connectors` or in `openidm/bundle`. `_action=createCoreConfig` takes the supplied connector reference (`connectorRef`) and adds the configuration properties required for that connector. This generates a core connector configuration that you can use to create a full configuration with the `createFullConfig` action. `_action=createFullConfig` generates a complete connector configuration, using the configuration properties from the `createCoreConfig` action, and retrieving the object types and operation options from the resource, to complete the configuration. `_action=test` returns a list of all remote systems, with their status, and supported object types. `_action=testConfig` validates the connector configuration provided in the POST body. `_action=liveSync` triggers a liveSync operation on the specified source object.

URI	HTTP Operation	Description
		`_action=authenticate` authenticates to the specified system with the credentials provided.
/openidm/system/*system-name*?_action=*action-name*	POST	`_action=test` tests the status of the specified system.
/openidm/system/*system-name*/*system-object*?_action=*action-name*	POST	`_action=liveSync` triggers a liveSync operation on the specified system object. `_action=script` runs the specified script on the system object. `_action=authenticate` authenticates to the specified system object, with the provided credentials. `_action=create` creates a new system object.
/openidm/system/*system-name*/*system-object*?_queryId=query-all-ids	GET	Lists all IDs related to the specified system object, such as users, and groups.
/openidm/system/*system-name*/*system-object*?_queryFilter=*filter*	GET	Lists the item(s) associated with the query filter.
/openidm/system/*system-name*/*system-object*/*id*	PUT	Creates a system object, or updates the system object, if it exists (replaces the entire object).
/openidm/system/*system-name*/*system-object*/*id*	PATCH	Updates the specified fields of a system object.
/openidm/system/*system-name*/*system-object*/*id*	DELETE	Deletes a system object.

Note

When you create a system object with a PUT request (that is, specifying a client-assigned ID), you should specify the ID in the URL only and not in the JSON payload. If you specify a different ID in the URL and in the JSON payload, the request will fail, with an error similar to the following:

```
{
    "code":500,
    "reason":"Internal Server Error",
    "message":"The uid attribute is not single value attribute."
}
```

A `POST` request with a `patch` action is not currently supported on system objects. To patch a system object, you must send a `PATCH` request.

Returns a list of the available connector configurations:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
"http://localhost:8080/openidm/system?_action=availableConnectors"
```

Returns a list of remote systems, and their status:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
"http://localhost:8080/openidm/system?_action=test"
[
  {
    "name": "ldap",
    "enabled": true,
    "config": "config/provisioner.openicf/ldap",
    "connectorRef": {
      "bundleVersion": "[1.4.0.0,1.6.0.0)",
      "bundleName": "org.forgerock.openicf.connectors.ldap-connector",
      "connectorName": "org.identityconnectors.ldap.LdapConnector"
    },
    "displayName": "LDAP Connector",
    "objectTypes": [
      "__ALL__",
      "account",
      "group"
    ],
    "ok": true
  }
]
```

Two options for running a liveSync operation on a specified system object:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
"http://localhost:8080/openidm/system?_action=liveSync&source=system/ldap/account"
{
  "connectorData": {
    "nativeType": "integer",
    "syncToken": 0
  },
  "_rev": "00000000a92657c7",
  "_id": "SYSTEMLDAPACCOUNT"
}
```

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
"http://localhost:8080/openidm/system/ldap/account?_action=liveSync"
{
  "connectorData": {
    "nativeType": "integer",
    "syncToken": 0
  },
  "_rev": "00000000a92657c7",
  "_id": "SYSTEMLDAPACCOUNT"
}
```

Run a script on a system object:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
"http://localhost:8080/openidm/system/ldap/account?_action=script&_scriptId=addUser"
```

Authenticate to a system object

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--request POST \
--data '{
  "username" : "bjensen",
  "password" : "Passw0rd"
}' \
"http://localhost:8080/openidm/system/ldap/account?_action=authenticate"
{
  "_id": "fc252fd9-b982-3ed6-b42a-c76d2546312c"
}
```

Creating a new system object

```
curl \
--header "Content-Type: application/json" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "X-OpenIDM-Username: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--data '{
  "cn": "James Smith",
  "dn": "uid=jsmith,ou=people,dc=example,dc=com",
  "uid": "jsmith",
  "sn": "Smith",
  "givenName":"James",
  "mail": "jsmith@example.com",
  "description": "Created by IDM REST"}' \
--request POST \
"http://localhost:8080/openidm/system/ldap/account?_action=create"
{
  "telephoneNumber": null,
  "description": "Created by IDM REST",
  "mail": "jsmith@example.com",
  "givenName": "James",
  "cn": "James Smith",
  "dn": "uid=jsmith,ou=people,dc=example,dc=com",
  "uid": "jsmith",
  "ldapGroups": [],
  "sn": "Smith",
  "_id": "07b46858-56eb-457c-b935-cfe6ddf769c7"
}
```

Rename a system object

You can rename a system object simply by supplying a new naming attribute value in a PUT request. The PUT request replaces the entire object. The naming attribute depends on the external resource.

The following example renames an object on an LDAP server, by changing the DN of the LDAP object (effectively performing a modDN operation on that object). The example renames the user created in the previous example:

```
curl \
--header "Content-Type: application/json" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "X-OpenIDM-Username: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "If-Match: *" \
--data '{
  "cn": "James Smith",
  "dn": "uid=jimmysmith,ou=people,dc=example,dc=com",
  "uid": "jimmysmith",
  "sn": "Smith",
  "givenName": "James",
  "mail": "jsmith@example.com"}' \
--request PUT \
"http://localhost:8080/openidm/system/ldap/account/07b46858-56eb-457c-b935-cfe6ddf769c7"
{
  "mail": "jsmith@example.com",
  "cn": "James Smith",
  "sn": "Smith",
  "dn": "uid=jimmysmith,ou=people,dc=example,dc=com",
  "ldapGroups": [],
  "telephoneNumber": null,
  "description": "Created by IDM REST",
  "givenName": "James",
  "uid": "jimmysmith",
  "_id": "07b46858-56eb-457c-b935-cfe6ddf769c7"
}
```

Lists the IDs associated with a specific system object:

```
curl \
--header "X-OpenIDM-Password: openidm-admin" \
--header "X-OpenIDM-Username: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/system/ldap/account?_queryId=query-all-ids"
{
  "remainingPagedResults": -1,
  "pagedResultsCookie": null,
  "resultCount": 3,
  "result": [
    {
      "dn": "uid=jdoe,ou=People,dc=example,dc=com",
      "_id": "1ff2e78f-4c4c-300c-b8f7-c2ab160061e0"
    },
    {
      "dn": "uid=bjensen,ou=People,dc=example,dc=com",
      "_id": "fc252fd9-b982-3ed6-b42a-c76d2546312c"
    },
    {
      "dn": "uid=jimmysmith,ou=people,dc=example,dc=com",
      "_id": "07b46858-56eb-457c-b935-cfe6ddf769c7"
    }
  ]
}
```

Internal Objects

You can manage the following internal objects over REST:

URI	HTTP Operation	Description
/openidm/internal/role?_queryFilter=true	GET	Lists all internal roles.
/openidm/internal/user?_queryFilter=true	GET	Lists internal users.
/openidm/internal/user/*username*	PUT	Adds a new internal user, or changes the password of an existing internal user.
/openidm/internal/user/*username*	PATCH	Adds or removes roles of an internal user.
/openidm/internal/role?_queryFilter=true&_fields=_id	GET	Lists internal roles.
/openidm/internal/role/*role-id*?_fields*,authzMembers	GET	Lists internal and managed users with the specified internal role.

Schedules

Use the scheduler service in the *Schedules Guide* to manage and monitor scheduled jobs.

You can access the scheduler service over REST, as indicated in the following table:

URI	HTTP Operation	Description
/openidm/scheduler?_action=validateQuartzCronExpression	POST	Validates a cron expression.
/openidm/scheduler/job/*id*	PUT	Creates or updates a schedule with the specified ID.
	GET	Obtains the details of the specified schedule.
	POST with ?_action=trigger *API V2 only*	Manually triggers the specified schedule.
	POST with ?_action=pause *API V2 only*	Suspends the specified schedule.
	POST with ?_action=resume *API V2 only*	Resumes the specified schedule.
	DELETE	Deletes the specified schedule.
/openidm/scheduler/job?_action=create	POST	Creates a schedule with a system-generated ID.
/openidm/scheduler/job?_queryFilter=*query*	GET	Queries the existing defined schedules.
/openidm/scheduler/job?_action=listCurrentlyExecutingJobs	POST	Returns a list of the jobs that are currently running.
/openidm/scheduler/job?_action=pauseJobs	POST	Suspends all scheduled jobs.
/openidm/scheduler/job?_action=resumeJobs	POST	Resumes all suspended scheduled jobs.
/openidm/scheduler/trigger?_queryFilter=*query*	GET	Queries the existing triggers.
/openidm/scheduler/trigger/*id*	GET	Obtains the details of the specified trigger.
/openidm/scheduler/acquiredTriggers	GET	Returns an array of the triggers that have been acquired, per node.
/openidm/scheduler/waitingTriggers	GET	Returns an array of the triggers that have not yet been acquired.

Scanning Tasks

The task scanning mechanism lets you perform a batch scan for a specified date, on a scheduled interval, and then execute a task when this date is reached.

IDM provides REST access to the task scanner, as listed in the following table:

URI	HTTP Operation	Description
/openidm/taskscanner	GET	Lists all the scanning tasks, past and present.
/openidm/taskscanner/*id*	GET	Lists details of the given task.
/openidm/taskscanner?_action=execute&name=*name*	POST	Triggers the specified task scan run.
/openidm/taskscanner/*id*?_action=cancel	POST	Cancels the specified task scan run.

Log Entries

You can interact with the audit logs over REST, as shown in the following table. Queries on the audit endpoint must use `queryFilter` syntax.

URI	HTTP Operation	Description
/openidm/audit/recon?_queryFilter=true	GET	Displays the reconciliation audit log.
/openidm/audit/recon/*id*	GET	Reads a specific reconciliation audit log entry.
/openidm/audit/recon/*id*	PUT	Creates a reconciliation audit log entry.
/openidm/audit/recon?_queryFilter=/reconId+eq +"*reconId*"	GET	Queries the audit log for a particular reconciliation operation.
/openidm/audit/recon?_queryFilter=/reconId+eq +"*reconId*"+and+situation+eq+"*situation*"	GET	Queries the reconciliation audit log for a specific reconciliation situation.
/openidm/audit/sync?_queryFilter=true	GET	Displays the synchronization audit log.
/openidm/audit/sync/*id*	GET	Reads a specific synchronization audit log entry.
/openidm/audit/sync/*id*	PUT	Creates a synchronization audit log entry.
/openidm/audit/activity?_queryFilter=true	GET	Displays the activity log.
/openidm/audit/activity/*id*	GET	Returns activity information for a specific action.
/openidm/audit/activity/*id*	PUT	Creates an activity audit log entry.
/openidm/audit/activity? _queryFilter=transactionId=*id*	GET	Queries the activity log for all actions resulting from a specific transaction.

URI	HTTP Operation	Description
/openidm/audit/access?_queryFilter=true	GET	Displays the full list of auditable actions.
/openidm/audit/access/*id*	GET	Displays information on the specific audit item.
/openidm/audit/access/*id*	PUT	Creates an access audit log entry.
/openidm/audit/authentication?_queryFilter=true	GET	Displays a complete list of authentication attempts, successful and unsuccessful.
/openidm/audit/authentication?_queryFilter=/principal+eq+"*principal*"	GET	Displays the authentication attempts by a specified user.
/openidm/audit?_action=availableHandlers	POST	Returns a list of audit event handlers.
openidm/audit/config?_queryFilter=true	GET	Lists changes made to the configuration.

Reconciliation Operations

You can interact with the reconciliation engine over REST, as shown in the following table:

URI	HTTP Operation	Description
/openidm/recon	GET	Lists all reconciliation runs, including those in progress. Inspect the `state` property to see the reconciliation status.
/openidm/recon?_action=recon&mapping=*mapping-name*	POST	Launches a reconciliation run with the specified mapping.
/openidm/recon?_action=reconById&mapping=*mapping-name*&id=*id*	POST	Restricts the reconciliation run to the specified ID.
/openidm/recon/*id*?_action=cancel	POST	Cancels the specified reconciliation run.

The following example runs a reconciliation for the mapping in the *Synchronization Guide* `systemHrdb_managedUser`:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
"http://localhost:8080/openidm/recon?_action=recon&mapping=systemHrdb_managedUser"
```

Synchronization Service

You can interact with the synchronization service over REST, as shown in the following table:

URI	HTTP Operation	Description
/openidm/sync?_action=getLinkedResources&resourceName=*resource*	POST	Provides a list of linked resources for the specified resource.
/openidm/sync/mappings?_queryFilter=true	GET	Returns a list of all configured mappings, in the order in which they will be processed.
/openidm/sync/queue?_queryFilter=*filter*	GET	Lists the queued synchronization events in the *Synchronization Guide*, based on the specified filter.
/openidm/sync/queue/*eventID*	DELETE	Deletes a queued synchronization event, based on its ID.

For example:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--request POST \
"http://localhost:8080/openidm/sync?_action=getLinkedResources&resourceName=managed/
user/42f8a60e-2019-4110-a10d-7231c3578e2b"

[
  {
    "resourceName": "system/ldap/account/03496258-1c5e-40a0-8744-badc2500f262",
    "content": {
      "uid": "joe.smith1",
      "mail": "joe.smith@example.com",
      "sn": "Smith",
      "givenName": "Joe",
      "employeeType": [],
      "dn": "uid=joe.smith1,ou=People,dc=example,dc=com",
      "ldapGroups": [],
      "cn": "Joe Smith",
      "kbaInfo": [],
      "aliasList": [],
      "objectClass": [
        "top",
        "inetOrgPerson",
        "organizationalPerson",
        "person"
      ],
      "_id": "03496258-1c5e-40a0-8744-badc2500f262"
    },
    "linkQualifier": "default",
    "linkType": "systemLdapAccounts_managedUser"
  }
]
```

Scripts

You can interact with the script service over REST, as shown in the following table:

URI	HTTP Operation	Description
/openidm/script?_action=compile	POST	Compiles a script, to validate that it can be executed. Note that this action compiles a script, but does not execute it. A successful compilation returns `true`. An unsuccessful compilation returns the reason for the failure.
/openidm/script?_action=eval	POST	Executes a script and returns the result, if any.

The following example compiles, but does not execute, the script provided in the JSON payload:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--request POST \
--data '{
  "type": "text/javascript",
  "source": "source.mail ? source.mail.toLowerCase() : null"
}' \
"http://localhost:8080/openidm/script?_action=compile"
True
```

The following example executes the script referenced in the `file` parameter, with the provided input:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--request POST \
--data '{
  "type": "text/javascript",
  "file": "script/autoPurgeAuditRecon.js",
  "globals": {
    "input": {
      "mappings": ["%"],
      "purgeType": "purgeByNumOfRecordsToKeep",
      "numOfRecons": 1
    }
  }
}' \
"http://localhost:8080/openidm/script?_action=eval"
"Must choose to either purge by expired or number of recons to keep"
```

Privileges

Privileges are a part of internal roles, and can be created or modified using the REST calls specified in "Internal Objects". Additionally, `openidm/privilege` can be used for getting information about privileges on a resource as they apply to the authenticated user.

URI	HTTP Operation	Description
/openidm/privilege?_action=listPrivileges	POST	Lists an array of privilege paths for the authenticated user, with additional detail required by the Admin UI.
/openidm/privilege/*resource*	GET	Lists the privileges for the logged in user associated with the given resource path.
/openidm/privilege/*resource*/*guid*	GET	Lists the privileges for the logged in user associated with the specified object.

Updates

You can interact with the updates engine over REST, as shown in the following table:

URI	HTTP Operation	Description
/openidm/maintenance/update?_action=available	POST	Lists update archives in the *`project-dir`*`/openidm/bin/update/` directory.
/openidm/maintenance/update?_action=preview&archive=*patch*.zip	POST	Lists file states of the current installation, relative to the *patch*.zip archive, using checksums.
openidm/maintenance/update?_action=listMigrations&archive=*patch*.zip	POST	Gets a list of repository migrations for a given update type.
/openidm/maintenance/update?_action=getLicense&archive=*patch*.zip	POST	Retrieves the license from the *patch*.zip archive.
/openidm/maintenance/update?_action=listRepoUpdates&archive=*patch*.zip	POST	Gets a list of repository update archives; use the *path* in the output for the endpoint with repo files.
/openidm/maintenance/update/archives/*patch*.zip/*path*?_field=contents&_mimeType=text/plain	POST	Get files for the specific repository update, defined in the *path*.
/openidm/maintenance?_action=enable	POST	Activates maintenance mode; you should first run the commands in Pause All Scheduled Jobs.
/openidm/maintenance?_action=disable	POST	Disables maintenance mode; you can then re-enable scheduled tasks in the *Schedules Guide*.

URI	**HTTP Operation**	**Description**
/openidm/maintenance?_action=status	POST	Returns current maintenance mode information.
/openidm/maintenance/update?_action=update&archive=*patch*.zip	POST	Starts an update with the *patch*.zip archive.
/openidm/maintenance/update?_action=installed	POST	Retrieves a summary of all installed updates.
/openidm/maintenance/update?_action=restart	POST	Restarts IDM.
/openidm/maintenance/update?_action=lastUpdateId	POST	Returns the `_id` value of the last successful update.
/openidm/maintenance/update?_action=markComplete&updateId=*id_string*	POST	For an update with `PENDING_REPO_UPDATES` for one or more repositories, mark as complete. Replace *id_string* with the value of `_id` for the update archive.
/openidm/maintenance/update/log/*_id*	GET	Gets information about an update, by *_id* (status, dates, file action taken).
/openidm/maintenance/update/log/?_queryFilter=true	GET	Gets information about all past updates, by repository.

Update Status Message

Status	**Description**
IN_PROGRESS	Update has started, not yet complete.
PENDING_REPO_UPDATES	Update is complete, updates to the repository are pending.
COMPLETE	Update is complete.
FAILED	Update failed, not yet reverted.

File Upload

IDM supports a generic file upload service at the `file` endpoint. Files are uploaded either to the filesystem or to the repository. For information about configuring this service, and for command-line examples, see "Upload Files to the Server" in the *Object Modeling Guide*.

IDM provides REST access to the file upload service, as listed in the following table:

URI	**HTTP Operation**	**Description**
/openidm/file/*handler*/	PUT	Uploads a file to the specified file *handler*. The file handler is either the repository or the filesystem and the context path is

URI	HTTP Operation	Description
		configured in the `conf/file-handler.json` file.
/openidm/file/*handler*/*filename*	GET	Returns the file content in a base 64-encoded string within the returned JSON object.
/openidm/file/*handler*/*filename*?_fields=content&_mimeType=*mimeType*"	GET	Returns the file content with the specified MIME type.
/openidm/file/*handler*/*filenamemimeType*"	DELETE	Deletes an uploaded file.

Bulk Import

The bulk import service lets you import large numbers of entries from a CSV file into the IDM repository. You can import any managed object type, but you will generally use this service to import user entries. The following table shows the endpoints used by the bulk import service:

URI	HTTP Operation	Description
/openidm/csv/template&_resourceCollection=managed/user	GET	Generates a CSV header row that you can use as a template for the import. You can safely remove generated columns for properties that are not required. Set the query parameters `_fields=header` and `_mimeType=text/csv` to download the header file.
/upload/csv/*resourceCollection*	POST	Uploads the file specified by the `--form` (`-F`) parameter to the specified resource collection. `?uniqueProperty=propertyName` is required. Generally, for `managed/user` objects, the `uniqueProperty` is `userName`. You can specify multiple comma-delimited values here to identify unique records; for example, `?uniqueProperty=firstName,lastName`. Example.
/openidm/csv/metadata/?_action=cleanupList	POST	Lists the import UUIDs that have error records or temporary records. These can be cleaned up to free up database space. If you clean up error records, you will no longer be able to download a CSV of failed import records.
/openidm/csv/metadata/*importUUID*?_action=cleanup	POST	Cleans up temporary import records for the specified import UUID. To also clean up error records, set the query parameter `?deleteErrorRecords=true`.

URI	HTTP Operation	Description
/openidm/csv/metadata/*importUUID*?_action=cancel	POST	Cancels the specified in-progress import.
/openidm/csv/metadata/*importUUID*	DELETE	Deletes the specified import record. This does not affect the data that was imported.
/openidm/csv/metadata?_queryFilter	GET	Queries bulk imports.
/openidm/csv/metadata/*importUUID*	GET	Reads the specified import record.
/export/csvImportFailures/*importUUID*	GET	Downloads a CSV file of failed import records. Returns 404 if there were no failures for the specified import UUID.

Server State

You can access information about the current state of the IDM instance through the `info` endpoint, as shown in the following table:

URI	HTTP Operation	Description
/openidm/info/features?_queryFilter=true	GET	Queries the available features in the server configuration.
/openidm/info/login	GET	Provides authentication and authorization details for the current user.
/openidm/info/ping	GET	Lists the current server state. Possible states are `STARTING`, `ACTIVE_READY`, `ACTIVE_NOT_READY`, and `STOPPING`.
/openidm/info/uiconfig	GET	Provides the UI configuration of this IDM instance. The language parameter returned is the user's preferred language, based on the `Accept-Language` header included in the request. If `Accept-Language` is not specified in the request, it returns the language set in `conf/ui-configuration.json`.
/openidm/info/version	GET	Provides the software version of this IDM instance.

Social Identity Providers

URI	HTTP Operation	Description
/openidm/identityProviders	GET	Returns JSON details for all configured social identity providers.

URI	HTTP Operation	Description
/openidm/authentication	GET	Returns JSON details for all configured social identity providers, if the `SOCIAL_PROVIDERS` module is enabled.
/openidm/managed/*social identity provider*	multiple	Supports access to social identity provider information.
/openidm/managed/user/*social identity provider*	GET	Supports a list of users associated with a specific social identity provider.
/openidm/managed/user/*User UUID*/idps	multiple	Supports management of social identity providers by UUID.

Workflows

Workflow objects are exposed under the `/openidm/workflow` context path. IDM provides access to the workflow module over REST, as listed in the following table:

URI	HTTP Operation	Description
/openidm/workflow/execution?_action=message	POST	Invokes a message event. When the `processVariables` field is *non-null* the message is sent synchronously; otherwise, asynchronously.
/openidm/workflow/execution?_action=signal	POST	Invokes a signal event. When the `processVariables` field is *non-null* the message is sent synchronously; otherwise, asynchronously.
/openidm/workflow/execution?_action=trigger	POST	Triggers an execution asynchronously; for example, to continue a process instance that is waiting at a *Receive Task*. When the `transientVariables` field is *non-null* the trigger is sent synchronously; otherwise, asynchronously.
/openidm/workflow/execution?_queryFilter=true	GET	Queries the executions. Only supports `_queryFilter=true`.
/openidm/workflow/execution?_queryId=filtered-query&*filter*	GET	Returns a list of executions, based on the specified query filter. + *Filter Parameters* • `executionId` • `executionParentId` • `processDefinitionCategory`

URI	HTTP Operation	Description
		• `processDefinitionId` • `processDefinitionKey` • `processDefinitionName` • `processDefinitionVersion` • `processInstanceBusinessKey` • `processInstanceId` • `activityId` • `signalName` • `messageName` • `startedBefore` • `startedAfter` • `startedBy` • `processVariableName` • `processVariableValue` • `processVariableValueType` • `processVariableOperator` • `variableName` • `variableValue` • `variableValueType` • `variableOperator`
/openidm/workflow/job?_queryFilter=true	GET	Queries jobs. Only supports `_queryFilter=true`.
/openidm/workflow/job?_queryId=filtered-query&*filter*	GET	Returns a list of jobs, based on the specified query filter. You can't combine `timersOnly=true` with `messagesOnly=true` in the same query. + *Filter Parameters* • `jobId`

URI	HTTP Operation	Description
		• `processDefinitionId` • `processInstanceId` • `executionId` • `timersOnly` • `messagesOnly` • `withException` • `exceptionMessage` • `dueAfter` • `dueBefore`
/openidm/workflow/job/deadletter?_queryFilter=true	GET	Queries dead-letter jobs. Only supports `_queryFilter=true`.
/openidm/workflow/job/deadletter?_queryId=filtered-query&*filter*	GET	Returns a list of dead-letter jobs, based on the specified query filter. You can't combine `timersOnly=true` with `messagesOnly=true` in the same query. + *Filter Parameters* • `jobId` • `processDefinitionId` • `processInstanceId` • `executionId` • `timersOnly` • `messagesOnly` • `withException` • `exceptionMessage` • `dueAfter` • `dueBefore`
/openidm/workflow/job/deadletter/*id*?_action=execute	POST	Executes a dead-letter job. If successful, runs as a normal job.

URI	HTTP Operation	Description
/openidm/workflow/job/deadletter/*id*?_action=stacktrace	POST	Displays the stacktrace for a dead-letter job that triggered an exception.
/openidm/workflow/job/deadletter/*id*	DELETE	Deletes a dead-letter job.
/openidm/workflow/job/deadletter/*id*	GET	Reads a dead-letter job.
/openidm/workflow/job/*id*?_action=execute	POST	Forces the synchronous execution of a job, even if it is suspended.
/openidm/workflow/job/*id*?_action=stacktrace	POST	Displays the stacktrace for a job that triggered an exception.
/openidm/workflow/job/*id*	DELETE	Deletes a job.
/openidm/workflow/job/*id*	GET	Reads a job.
/openidm/workflow/model?_action=validate_bpmn	POST	Validates a BPMN 2.0 XML file.
/openidm/workflow/model	POST	Creates a new model. Omitting the `bpmnXML` field generates a template. Also see the `deploy` action.
/openidm/workflow/model?_queryFilter=*query*	GET	Queries the existing models. `bpmnXML` and `resourceMap` fields are omitted from results by default.
openidm/workflow/model/*id*?_action=deploy	POST	Deploys a model and creates associated process definitions. Existing process definition IDs will be returned if duplicate model detected. See `workflow/processdefinition` endpoints.
/openidm/workflow/model/*id*?_action=list_deployments	POST	Lists process definition IDs for model deployments.
/openidm/workflow/model/*id*	DELETE	Deletes a model.
/openidm/workflow/model/*id*	GET	Reads a model.
/openidm/workflow/model/*id*	PUT	Updates a model.
/openidm/workflow/processdefinition?_queryFilter=true	GET	Queries the process definitions. Only supports `_queryFilter=true`. Use the `READ` endpoint to get form-related fields.
/openidm/workflow/processdefinition?_queryId=filtered-query&*filter*	GET	Returns a list of workflow process definitions, based on the specified query filter. + *Filter Parameters* • `version` • `deploymentId` • `category`

URI	HTTP Operation	Description
		• `key` • `name` • `processDefinitionResourceName`
/openidm/workflow/processdefinition/*id*	DELETE	Deletes a process definition.
/openidm/workflow/processdefinition/*id*	GET	Reads a process definition with form-related fields included.
/openidm/workflow/processdefinition/*procdefid*/taskdefinition?_queryFilter=true	GET	Queries the task definitions. Only supports `_queryFilter=true`.
/openidm/workflow/processdefinition/*procdefid*/taskdefinition/*id*	GET	Reads a task definition.
/openidm/workflow/processinstance	POST	Creates a process instance. JSON request object must contain either the `_processDefinitionId` or `_key` fields, but not both. Additional JSON fields will be passed to the create-operation and utilized if applicable.
/openidm/workflow/processinstance?_queryFilter=true	GET	Queries the process instances. Only supports `_queryFilter=true`.
/openidm/workflow/processinstance?_queryId=filtered-query&*filter*	GET	Returns a list of workflow process instances, based on the specified query filter. + *Filter parameters* • `processDefinitionId` • `processDefinitionKey` • `processInstanceBusinessKey` • `processInstanceId` • `superProcessInstanceId` • `finished` • `unfinished` • `involvedUserId` • `startUserId` • `startedAfter`

URI	HTTP Operation	Description
		• `startedBefore`
/openidm/workflow/processinstance/history?_queryFilter=true	GET	Queries the process instance history. Only supports `_queryFilter=true`.
/openidm/workflow/processinstance/history?_queryId=filtered-query&*filter*	GET	Returns a list of process instance history, based on the specified query filter. + *Filter parameters* • `processDefinitionId` • `processDefinitionKey` • `processInstanceBusinessKey` • `processInstanceId` • `superProcessInstanceId` • `finished` • `unfinished` • `involvedUserId` • `startUserId` • `startedAfter` • `startedBefore`
/openidm/workflow/processinstance/history/*id*	DELETE	Deletes process instance history.
/openidm/workflow/processinstance/history/*id*	GET	Reads process instance history. `diagram` field returned when defined and requested in `_fields` parameter.
/openidm/workflow/processinstance/*id*?_action=migrate	POST	Migrates a process instance to a different process definition. To simulate the migration first, see the action `validateMigration`. + *JSON Request Payload Model*

URI	HTTP Operation	Description
		`{ "processDefinitionId": "string", "variables": {}, "fromActivityIdMap": { "123": { "toActivityId": "string", "localVariables": {} } } }`
/openidm/workflow/processinstance/*id*?_action=validateMigration	POST	Simulates a process instance migration (`migrate` action).
/openidm/workflow/processinstance/*id*	DELETE	Deletes a process instance.
/openidm/workflow/processinstance/*id*	GET	Reads a process instance. `diagram` field returned when defined and requested in `_fields` parameter.
/openidm/workflow/taskinstance?_queryFilter=true	GET	Queries the task instances. Only supports `_queryFilter=true`.
/openidm/workflow/taskinstance?_queryId=filtered-query&*filter*	GET	Returns a list of task instances, based on the specified query filter. + *Filter parameters* • `executionId` • `processDefinitionId` • `processDefinitionKey` • `processInstanceId` • `assignee` • `taskCandidateGroup` • `taskCandidateUser` • `name` • `owner` • `description` • `priority`

URI	HTTP Operation	Description
		• `unassigned`
/openidm/workflow/taskinstance?_queryId=unassignedTaskQuery	GET	Queries unassigned task instances for which the authenticated user is authorized to assign.
/openidm/workflow/taskinstance/history?_queryFilter=true	GET	Queries the task instance history. Only supports `_queryFilter=true`.
/openidm/workflow/taskinstance/history?_queryId=filtered-query&*filter*	GET	Returns a list of task instance history, based on the specified query filter. + *Filter parameters* • `executionId` • `processDefinitionId` • `processDefinitionKey` • `processInstanceId` • `assignee` • `taskCandidateGroup` • `taskCandidateUser` • `taskId` • `taskName` • `owner` • `description` • `finished` • `unfinished` • `processFinished` • `processUnfinished` • `priority` • `deleteReason`
/openidm/workflow/taskinstance/history/*id*	GET	Reads a task instance history.
/openidm/workflow/taskinstance/*id*?_action=claim	POST	Assigns a task to a `userId`.

URI	HTTP Operation	Description
/openidm/workflow/taskinstance/*id*?_action=complete	POST	To complete a task, supply the required task parameters, as specified in the BPMN 2.0 XML file.
/openidm/workflow/taskinstance/*id*	DELETE	Deletes a task instance.
/openidm/workflow/taskinstance/*id*	GET	Reads a task instance.
/openidm/workflow/taskinstance/*id*	PUT	Updates a task instance. Must include one or more *supported* fields in JSON payload: • `assignee` • `description` • `name` • `owner` • `category` • `dueDate` • `priority`

The following examples list the defined workflows. For a workflow to appear in this list, the corresponding workflow definition must be in the `openidm/workflow` directory:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/workflow/processdefinition?_queryId=query-all-ids"
```

Depending on the defined workflows, the output will be something like the following:

```
{
  "result": [
    {
      "_id": "contractorOnboarding:1:5"
    },
    {
      "_id": "contractorOnboarding:2:9"
    }
  ],
  "resultCount": 2,
  "pagedResultsCookie": null,
  "totalPagedResultsPolicy": "NONE",
  "totalPagedResults": -1,
  "remainingPagedResults": -1
}
```

The following example invokes a workflow named "myWorkflow". The `foo` parameter is given the value `bar` in the workflow invocation:

```
curl \
--header "Content-Type: application/json" \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
--data '{
  "_key":"contractorOnboarding",
  "foo":"bar"
}' \
"http://localhost:8080/openidm/workflow/processinstance?_action=create"
```

IDM Glossary

correlation query	A correlation query specifies an expression that matches existing entries in a source repository to one or more entries in a target repository. A correlation query might be built with a script, but it is not the same as a correlation script. For more information, see "*Correlating Source Objects With Existing Target Objects*" in the *Synchronization Guide*.
correlation script	A correlation script matches existing entries in a source repository, and returns the IDs of one or more matching entries on a target repository. While it skips the intermediate step associated with a `correlation query`, a correlation script can be relatively complex, based on the operations of the script.
entitlement	An entitlement is a collection of attributes that can be added to a user entry via roles. As such, it is a specialized type of `assignment`. A user or device with an entitlement gets access rights to specified resources. An entitlement is a property of a managed object.
JCE	Java Cryptographic Extension, which is part of the Java Cryptography Architecture, provides a framework for encryption, key generation, and digital signatures.
JSON	JavaScript Object Notation, a lightweight data interchange format based on a subset of JavaScript syntax. For more information, see the JSON site.
JSON Pointer	A JSON Pointer defines a string syntax for identifying a specific value within a JSON document. For information about JSON Pointer syntax, see the JSON Pointer RFC.

JWT	JSON Web Token. As noted in the JSON Web Token draft IETF Memo, "JSON Web Token (JWT) is a compact URL-safe means of representing claims to be transferred between two parties." For IDM, the JWT is associated with the `JWT_SESSION` authentication module.
managed object	An object that represents the identity-related data managed by IDM. Managed objects are configurable, JSON-based data structures that IDM stores in its pluggable repository. The default configuration of a managed object is that of a user, but you can define any kind of managed object, for example, groups or roles.
mapping	A policy that is defined between a source object and a target object during reconciliation or synchronization. A mapping can also define a trigger for validation, customization, filtering, and transformation of source and target objects.
OSGi	A module system and service platform for the Java programming language that implements a complete and dynamic component model. For more information, see What is OSGi? Currently, only the Apache Felix container is supported.
reconciliation	During reconciliation, comparisons are made between managed objects and objects on source or target systems. Reconciliation can result in one or more specified actions, including, but not limited to, synchronization.
resource	An external system, database, directory server, or other source of identity data to be managed and audited by the identity management system.
REST	Representational State Transfer. A software architecture style for exposing resources, using the technologies and protocols of the World Wide Web. REST describes how distributed data objects, or resources, can be defined and addressed.
role	IDM distinguishes between two distinct role types - provisioning roles and authorization roles. For more information, see "Managed Roles" in the *Object Modeling Guide*.
source object	In the context of reconciliation, a source object is a data object on the source system, that IDM scans before attempting to find a corresponding object on the target system. Depending on the defined mapping, IDM then adjusts the object on the target system (target object).
synchronization	The synchronization process creates, updates, or deletes objects on a target system, based on the defined mappings from the source system. Synchronization can be scheduled or on demand.

system object	A pluggable representation of an object on an external system. For example, a user entry that is stored in an external LDAP directory is represented as a system object in IDM for the period during which IDM requires access to that entry. System objects follow the same RESTful resource-based design principles as managed objects.
target object	In the context of reconciliation, a target object is a data object on the target system, that IDM scans after locating its corresponding object on the source system. Depending on the defined mapping, IDM then adjusts the target object to match the corresponding source object.