Configuration Reference

/ ForgeRock Microgateway 1.0.2

Latest update: 1.0.2

Abstract

Reference documentation for ForgeRock® Microgateway.

Table of Contents

Preface

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.

1. About This Guide

This guide describes in detail the configuration options for Microgateway. It is for Microgateway designers, developers, and administrators.

For API specifications, see the appropriate Javadoc.

Some of the following third-party tools are used in the examples in this guide:

- **curl**: https://curl.haxx.se
- **HTTPie**: https://httpie.org
- **jq**: https://stedolan.github.io/jq/
- **keytool**: https://docs.oracle.com/en/java/javase/11/tools/keytool.html

2. Reserved Routes

By default, Microgateway reserves all paths starting with `/openig` for administrative use, and only local client applications can access resources exposed under `/openig`.

To change the base for administrative routes, edit `admin.json`. For more information, see "*AdminHttpApplication (`admin.json`)*".

3. Reserved Field Names

Microgateway reserves all configuration field names that contain only alphanumeric characters.

If you must define your own field names, for example, in custom decorators, use names with dots, `.`, or dashes, `-`. Examples include `my-decorator` and `com.example.myDecorator`.

4. Field Value Conventions

Microgateway configuration uses JSON notation.

This reference uses the following terms when referring to values of configuration object fields:

array

JSON array.

boolean

Either `true` or `false`.

configuration token

Configuration tokens introduce variables into the server configuration. They can take values from Java system properties, environment variables, JSON and Java properties files held in specified directories, and from properties configured in routes.

For more information, see "*JSON Evaluation*".

duration

A duration is a lapse of time expressed in English, such as `23 hours 59 minutes and 59 seconds`. Durations are not case sensitive, and negative durations are not supported. The following units can be used in durations:

- `indefinite`, `infinity`, `undefined`, `unlimited`: unlimited duration
- `zero`, `disabled`: zero-length duration
- `days`, `day`, `d`: days
- `hours`, `hour`, `h`: hours
- `minutes`, `minute`, `min`, `m`: minutes
- `seconds`, `second`, `sec`, `s`: seconds
- `milliseconds`, `millisecond`, `millisec`, `millis`, `milli`, `ms`: milliseconds
- `microseconds`, `microsecond`, `microsec`, `micros`, `micro`, `us`, `µs`: microseconds
- `nanoseconds`, `nanosecond`, `nanosec`, `nanos`, `nano`, `ns`: nanoseconds

expression

See "*Expressions*".

configuration expression

Expression evaluated at configuration time, when routes are loaded.

Configuration expressions can refer to the system heap properties, the built-in functions listed in "*Functions*", the `${env['variable']}`, and `${system['property']}`. Because configuration expressions are evaluated before any requests are made, they cannot refer to the runtime properties, `request`, `response`, or `context`. For more information, see "*Expressions*".

runtime expression

Expression evaluated at runtime, for each request and response.

Runtime expressions can refer to the same information as configuration expressions, plus the following objects:

- `attributes`: org.forgerock.services.context.AttributesContext Map<String, Object>, obtained from `AttributesContext.getAttributes()`. For information, see "*AttributesContext*".
- `context`: org.forgerock.services.context.Context object.
- `contexts`: map<string, context> object. For information, see "*Contexts*".
- `request`: org.forgerock.http.protocol.Request object. For information, see "*Request*".
- `response`: org.forgerock.http.protocol.Response object, available only when the expression is intended to be evaluated on the response flow. For information, see "*Response*".
- `session`: org.forgerock.http.session.Session object, available only when the expression is intended to be evaluated for both request and response flow. For information, see "*SessionContext*".

instant

An instantaneous point on the timeline, as a Java type. For more information, see Class Instant.

lvalue-expression

Expression yielding an object whose value is to be set.

number

JSON number.

object

JSON object where the content depends on the object's type.

pattern

A regular expression according to the rules for the Java Pattern class.

pattern-template

Template for referencing capturing groups in a pattern by using `$n`, where *n* is the index number of the capturing group starting from zero.

For example, if the pattern is `"\w+\s*=\s*(\w)+"`, the pattern-template is `"$1"`, and the text to match is `"key = value"`, the pattern-template yields `"value"`.

reference

References an object in the following ways:

- An inline configuration object, where the name is optional.
- A configuration expression that is a string or contains variable elements that evaluate to a string, where the string is the name of an object declared in the heap.

  For example, the following `temporaryStorage` object takes the value of the system property `storage.ref`, which must a be string equivalent to the name of an object defined in the heap:

```
{
  "temporaryStorage": "${system['storage.ref']}"
}
```

secret-id

String that references a secret managed the ForgeRock Commons Secrets Service, as described in "Secrets".

The secret ID must conform to the following regex pattern: `Pattern.compile("[a-zA-Z0-9]+(\\.[a-zA-Z0-9]+)*");`

string

JSON string.

url

String representation for a resource available via the Internet. For more information, see RFC 1738.

5. 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.

5.1. 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.

5.2. 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".

5.3. 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.

5.4. 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.

5.5. 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.

5.6. 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.

5.7. 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.

By specifying both a single *field* and also the *mime-type* for the response content, you can read a single field value that is a multi-media resource.

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.

5.8. 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.

5.9. 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.

5.10. 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`:

5.10.1. 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"]]
}
```

5.10.2. 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".

5.10.3. 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.

5.10.4. 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".

5.10.5. 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.

5.10.6. 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" ],
  }
]
```

5.10.7. 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"
      }
    }
  }
]
```

5.10.8. 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.

5.11. 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.

5.12. 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.

5.13. 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.

Part 1

Required Configuration

The configuration of Microgateway is split into the following parts:

- AdminHttpApplication: the entry point for administrative requests
- GatewayHttpApplication: the entry point for gateway requests

Chapter 1.1

AdminHttpApplication (`admin.json`)

1.1.1. Description

The AdminHttpApplication serves requests on the administrative route, such as the creation of routes and the collection of monitoring information. The administrative route and its subroutes are reserved for administration endpoints.

The configuration is loaded from a JSON-encoded configuration file, expected by default at `/path/to/microservices/identity-gateway/config/admin.json`.

1.1.1.1. Default Objects

Microgateway creates the following objects by default in `admin.json`:

`AuditService`

Records no audit events. The default AuditService is `NoOpAuditService`. For more information, see "*NoOpAuditService*".

`CaptureDecorator`

Captures requests and response messages. The default CaptureDecorator is named `capture`. For more information, see "*CaptureDecorator*".

`ClientHandler`

Communicates with third-party services. For more information, see "*ClientHandler*".

`ForgeRockClientHandler`

Sends ForgeRock Common Audit transaction IDs when communicating with protected applications. The default ForgeRockClientHandler is a Chain, composed of a TransactionIdOutboundFilter and a ClientHandler.

`ReverseProxyHandler`

Communicates with third-party services. For more information, see "*ReverseProxyHandler*".

`ScheduledExecutorService`

Specifies the number of threads in a pool.

SecretsService

Manages a store of secrets from files, system properties, and environment variables, by using ForgeRock Commons Secrets Service. The default SecretsService is a SystemAndEnvSecretStore with the default configuration. For more information, see "Secrets".

TemporaryStorage

Manages temporary buffers. To change the default values, add a TemporaryStorage object named `TemporaryStorage`, and use non-default values. For more information, see "*TemporaryStorage*".

TimerDecorator

Records time spent within filters and handlers. The default TimerDecorator is named `timer`. For more information, see "*TimerDecorator*".

TransactionIdOutboundFilter

Inserts the ID of a transaction into the header of a request.

1.1.1.2. Provided Objects

Microgateway creates the following objects when a filter with the name of the object is declared in `admin.json`:

"ApiProtectionFilter"

The default filter used to protect administrative APIs on reserved routes. To override this filter, declare a different filter with the same name in `admin.json`.

By default, only the loopback address can access reserved routes.

For information about reserved routes, see "Reserved Routes".

"MetricsProtectionFilter"

A filter used to protect the monitoring endpoints.

By default, the Prometheus Scrape Endpoint and Common REST Monitoring Endpoint are open and accessible. No special credentials or privileges are required to access the monitoring endpoints.

To protect the monitoring endpoints, declare a different filter with the same name in `admin.json`.

1.1.2. Usage

```
{
   "heap": [ configuration object, ... ],
   "connectors": configuration object,
   "mode": enumeration,
   "prefix": configuration expression<string>,
   "properties": JSON object,
   "secrets": configuration object,
   "temporaryDirectory": configuration expression<string>,
   "temporaryStorage": TemporaryStorage reference,
   "preserveOriginalQueryString": configuration expression<boolean>
}
```

1.1.3. Properties

`"connectors"`: *configuration object, required*

This property can be configured as one or more ports on which the Microgateway is connected, or any other connection supported by Vertx. For more information, see Vertx HttpServerOptions.

When more than one port is defined, the Microgateway is connected to each port.

`mode`: *operating mode, optional*

Set the Microgateway mode to `development` or `production`. The value is not case-sensitive.

- **Development mode (mutable mode)**

  In development mode, by default all endpoints are open and accessible.

  Use development mode to evaluate or demo Microgateway, or to develop configurations on a single instance. This mode is not suitable for production.

- **Production mode (immutable mode)**

  In production mode, the `/routes` endpoint is not exposed or accessible.

  By default, other endpoints, such as `/share` and `api/info` are exposed to the loopback address only. To change the default protection for specific endpoints, configure an ApiProtectionFilter in `admin.json` and add it to the Microgateway configuration.

If `mode` is not set, the provided configuration token `ig.run.mode` can be resolved at startup to define the run mode.

Default: `production`

`"heap"`: *array of configuration objects, optional*

The heap object configuration, described in "*Heap Objects*".

You can omit an empty array.

"prefix": *configuration expression<string>, optional*

The base of the route for administration requests. This route and its subroutes are reserved for administration endpoints.

Default: `openig`

"properties": *JSON object, optional*

Configuration parameters declared as property variables for use in the configuration. See also "*Properties*".

Default: none

"secrets": *configuration object, optional*

An object that configures an inline array of one or more secret stores, as defined in "*secrets*".

"temporaryDirectory": *configuration expression<string>, optional*

Directory containing temporary storage files.

Set this property to store temporary files in a different directory, for example:

```
{
   "temporaryDirectory": "/path/to/my-temporary-directory"
}
```

Default: `/path/to/microservices/identity-gateway/tmp` (on Windows, `%appdata%\OpenIG\tmp`)

"temporaryStorage": *TemporaryStorage reference, optional*

Cache content during processing based on this TemporaryStorage configuration.

Define the TemporaryStorage in one of the following ways:

- An inline TemporaryStorage configuration object.
- The name of a TemporaryStorage object defined in the heap.
- A configuration expression that evaluates to the name of a TemporaryStorage object defined in the heap.

Default: use the heap object named TemporaryStorage.

See also reference and "*TemporaryStorage*".

"preserveOriginalQueryString": *configuration expression<boolean>, optional*

Process query strings in URLs, by applying or not applying a decode/encode process to the whole query string.

The following characters are disallowed in query string URL components: `"`, `{`, `}`, `<`, `>`, ` ` (space), and `|`. For more information about which query strings characters require encoding, see Uniform Resource Identifier (URI): Generic Syntax.

- `true`: Preserve query strings as they are presented.

  Select this option if the query string must not change during processing, for example, in signature verification.

  If a query string contains a disallowed character, the request produces a `400 Bad Request`.

- `false`: Tolerate disallowed characters in query string URL components, by applying a decode/encode process to the whole query string.

  Select this option when a user agent or client produces query searches with disallowed characters. Microgateway transparently encodes the disallowed characters before forwarding requests to the protected application.

  Characters in query strings are transformed as follows:

  - Allowed characters are not changed.

    For example, `sep=a` is not changed.

  - Percent-encoded values are re-encoded when the decoded value is an allowed character.

    For example, `sep=%27` is changed to `sep='`, because `'` is an allowed character.

  - Percent-encoded values are not changed when the decoded value is a disallowed character.

    For example, `sep=%22` is not changed, because `"` is a disallowed character.

  - Disallowed characters are encoded.

    For example, `sep="`, is changed to `sep=%22`, because `"` is a disallowed character.

Default: `false`

1.1.4. Example Configuration files

1.1.4.1. Default Configuration

When your configuration does not include an `admin.json` file, the following configuration is provided by default.

```
{
  "prefix": "openig",
  "connectors": [
    {
      "port": 8080
    }
  ]
}
```

1.1.5. More Information

org.forgerock.openig.http.AdminHttpApplication

Chapter 1.2

GatewayHttpApplication (`config.json`)

1.2.1. Description

The GatewayHttpApplication is the entry point for all incoming gateway requests. It is responsible for initializing a heap of objects, described in "*Heap Objects*", and providing the main Handler that receives all the incoming requests. The configuration is loaded from a JSON-encoded configuration file, expected by default at `/path/to/microservices/identity-gateway/config/config.json`.

If you provide a `config.json`, the Microgateway configuration is loaded from that file. If there is no file, the default configuration is loaded. For the default configuration, and the example `config.json` used in many of the examples in the documentation, see the Examples section of this page.

1.2.1.1. Routes Endpoint

The endpoint is defined by the presence and content of `config.json`, as follows:

- When `config.json` is not provided, the routes endpoint includes the name of the main router in the default configuration, `_router`.
- When `config.json` is provided with an unnamed main router, the routes endpoint includes the main router name `router-handler`.
- When `config.json` is provided with a named main router, the routes endpoint includes the provided name or the transformed, URL-friendly name.

1.2.1.2. Default Objects

Microgateway creates the following objects by default in `config.json`:

`BaseUriDecorator`

Overrides the scheme, host, and port of the existing request URI. The default BaseUriDecorator is named `baseURI`. For more information, see "*BaseUriDecorator*".

`AuditService`

Records no audit events. The default AuditService is `NoOpAuditService`. For more information, see "*NoOpAuditService*".

CaptureDecorator

Captures requests and response messages. The default CaptureDecorator is named `capture`. For more information, see "*CaptureDecorator*".

ClientHandler

Communicates with third-party services. For more information, see "*ClientHandler*".

ForgeRockClientHandler

Sends ForgeRock Common Audit transaction IDs when communicating with protected applications. The default ForgeRockClientHandler is a Chain, composed of a TransactionIdOutboundFilter and a ClientHandler.

ReverseProxyHandler

Communicates with third-party services. For more information, see "*ReverseProxyHandler*".

ScheduledExecutorService

Specifies the number of threads in a pool.

SecretsService

Manages a store of secrets from files, system properties, and environment variables, by using ForgeRock Commons Secrets Service. The default SecretsService is a SystemAndEnvSecretStore with the default configuration. For more information, see "Secrets".

TemporaryStorage

Manages temporary buffers. To change the default values, add a TemporaryStorage object named `TemporaryStorage`, and use non-default values. For more information, see "*TemporaryStorage*".

TimerDecorator

Records time spent within filters and handlers. The default TimerDecorator is named `timer`. For more information, see "*TimerDecorator*".

TransactionIdOutboundFilter

Inserts the ID of a transaction into the header of a request.

1.2.1.3. Sessions

When the heap is configured with a JwtSession object named `Session`, the object is used as the default session producer. Stateless sessions are created for all requests.

When a JwtSession is not configured for a request, stateful sessions are created by the container where Microgateway runs. Session information is stored in an Microgateway cookie called by default `IG_SESSIONID`.

1.2.2. Usage

```
{
  "handler": Handler reference or inline Handler declaration,
  "heap": [ configuration object, ... ],
  "properties": JSON object,
  "secrets": configuration object,
  "temporaryStorage": TemporaryStorage reference
}
```

1.2.3. Properties

`"handler"`: *Handler reference, required*

Dispatch all requests to this handler.

Provide either the name of a Handler object defined in the heap, or an inline Handler configuration object.

See also "Handlers".

`"heap"`: *array of configuration objects, optional*

The heap object configuration, described in "*Heap Objects*".

You can omit an empty array. If you only have one object in the heap, you can inline it as the handler value.

`"properties"`: *JSON object, optional*

Configuration parameters declared as property variables for use in the configuration. See also "*Properties*".

Default: none

`"secrets"`: *configuration object, optional*

An object that configures an inline array of one or more secret stores, as defined in "*secrets*".

`"temporaryStorage"`: *TemporaryStorage reference, optional*

Cache content during processing based on this TemporaryStorage configuration.

Provide either the name of a TemporaryStorage object defined in the heap, or an inline TemporaryStorage configuration object.

Default: use the heap object named TemporaryStorage. Otherwise use an internally-created TemporaryStorage object that is named TemporaryStorage and that uses default settings for a TemporaryStorage object.

See also "*TemporaryStorage*".

1.2.4. More Information

org.forgerock.openig.http.GatewayHttpApplication

Chapter 1.3

Heap Objects

1.3.1. Description

A *heaplet* creates and initializes an object that is stored in a heap. A heaplet can retrieve objects it depends on from the heap.

A *heap* is a collection of associated objects created and initialized by heaplet objects. All configurable objects in Microgateway are heap objects.

The heap configuration is included as an object in `admin.json` and `config.json`.

1.3.2. Usage

```
[
    {
        "name": string,
        "type": string,
        "config": {
            object-specific configuration
        }
    },
    ...
]
```

1.3.3. Properties

`"name"`: *string, required except for inline objects*

The unique name to give the heap object in the heap. This name is used to resolve the heap object, for example, when another heap object names a heap object dependency.

`"type"`: *string, required*

The class name of the object to be created. To determine the type name, see the object's documentation in this reference.

`"config"`: *object, required*

The configuration that is specific to the heap object being created.

If all the fields are optional and the configuration uses only default settings, you can omit the config field instead of including an empty config object as the field value.

1.3.4. More Information

org.forgerock.openig.heap.Heap

Chapter 1.4
Configuration Settings

1.4.1. Description

Filters, handlers, and other objects whose configuration settings are defined by strings, integers, or booleans, can alternatively be defined by expressions that match the expected type.

Expressions can retrieve the values for configuration settings from system properties or environment variables. When Microgateway starts up or when a route is reloaded, the expressions are evaluated. If you change the value of a system property or environment variable and then restart Microgateway or reload the route, the configuration settings are updated with the new values.

If a configuration setting is required and the expression returns `null`, an error occurs when Microgateway starts up or when the route is reloaded. If the configuration setting is optional, there is no error.

For information about expressions, see "*Expressions*".

Part 2

Handlers

Handler objects process a request and context, and return a response. The way the response is created depends on the type of handler.

This part describes the handlers provided in Microgateway.

Chapter 2.1

Chain

2.1.1. Description

Dispatches a request and context to an ordered list of filters, and then finally to a handler.

Filters process the incoming request and context, pass it on to the next filter, and then to the handler. After the handler produces a response, the filters process the outgoing response and context as it makes its way to the client. Note that the same filter can process both the incoming request and the outgoing response but most filters do one or the other.

A Chain can be placed in a configuration anywhere that a handler can be placed.

Unlike ChainOfFilters, Chain finishes by dispatching the request to a handler. For more information, see "*ChainOfFilters*".

2.1.2. Usage

```
{
     "name": string,
     "type": "Chain",
     "config": {
         "filters": [ Filter reference, ... ],
         "handler": Handler reference
     }
}
```

2.1.3. Properties

`"filters"`: *array of filter references, required*

An array of names of filter objects defined in the heap, and inline filter configuration objects.

The chain dispatches the request to these filters in the order they appear in the array.

See also "Filters".

`"handler"`: *Handler reference, required*

Either the name of a handler object defined in the heap, or an inline handler configuration object.

The chain dispatches to this handler after the request has traversed all of the specified filters.

See also "Handlers".

2.1.4. Example

```
{
     "name": "LoginChain",
     "type": "Chain",
     "config": {
         "filters": [ "LoginFilter" ],
         "handler": "ReverseProxyHandler"
     }
}
```

2.1.5. More Information

org.forgerock.openig.filter.ChainHandlerHeaplet

Chapter 2.2
ClientHandler

2.2.1. Description

Creates a response to a request by forwarding the request to a third-party services accessible through HTTP, and reconstructing the response from the received bytes.

A third-party service is one that Microgateway calls for data, such as an HTTP API or AM, or one to which Microgateway submits data, such as Elasticsearch or Splunk from an audit event handler.

If Microgateway fails to connect to the third-party service, the ClientHandler propagates the error along the chain.

To relay requests to protected applications, use a "*ReverseProxyHandler*", to manage connection failures differently.

2.2.2. Usage

```
{
  "name": string,
  "type": "ClientHandler",
  "config": {
    "connections": number,
    "disableReuseConnection": boolean,
    "hostnameVerifier": configuration expression<enumeration>,
    "soTimeout": duration string,
    "connectionTimeout": duration string,
    "connectionTimeToLive": duration string,
    "numberOfWorkers": number,
    "proxy": Server reference,
    "systemProxy": boolean,
    "temporaryStorage": string,
    "tls": TlsOptions reference,
    "asyncBehavior": enumeration,
    "retries": object,
    "websocket": object
  }
}
```

2.2.3. Properties

`"connections"`: *number, optional*

The maximum number of connections in the HTTP client connection pool.

Default: 64

`"connectionTimeout"`: *duration string, optional*

Amount of time to wait to establish a connection, expressed as a duration

Default: 10 seconds

For information about supported formats for `duration`, see duration.

`"connectionTimeToLive"`: *duration string, optional*

Amount of time before a reusable pooled connection expires.

Set this property to expire reusable pooled connections after a fixed duration. For example, to prevent the reuse of connections set this property in routes for applications where the IP address (baseURI) is not stable or can change.

Default: Unlimited

For information about supported formats for `duration`, see duration.

`"disableReuseConnection"`: *boolean, optional*

Whether to disable connection reuse.

Default: `false`

`"hostnameVerifier"`: *configuration expression<enumeration>, optional*

Way to handle hostname verification for outgoing SSL connections. Use one of the following values:

- `ALLOW_ALL`: Allow a certificate issued by a trusted CA for any hostname or domain to be accepted for a connection to any domain.

  Caution

  This setting allows a certificate issued for one company to be accepted as a valid certificate for another company.

  To prevent the compromise of TLS connections, use this setting in development mode only. In production, use `STRICT`.

- `STRICT`: Match the hostname either as the value of the the first CN, or any of the subject-alt names.

A wildcard can occur in the CN, and in any of the subject-alt names. Wildcards match one domain level, so `*.example.com` matches `www.example.com` but not `some.host.example.com`.

Default: `STRICT`

`"numberOfWorkers"`: *number, optional*

The number of worker threads dedicated to processing outgoing requests.

Increasing the value of this attribute can be useful in deployments where a high number of simultaneous connections remain open, waiting for protected applications to respond.

Default: One thread per CPU available to the JVM.

`"proxy"`: *Server reference, optional*

A proxy server to which requests can be submitted. Use this property to relay requests to other parts of the network. For example, use it to submit requests from an internal network to the internet.

If both `proxy` and `systemProxy` are defined, `proxy` takes precedence.

```
"proxy" : {
  "uri": configuration expression<uri string>,
  "username": configuration expression<string>,
  "passwordSecretId": configuration expression<secret-id>,
  "secretsProvider": SecretsProvider reference
}
```

`uri`: *configuration expression<uri string>, required*

URI of a server to use as a proxy for outgoing requests.

The result of the expression must be a string that represents a valid URI, but is not a real `java.net.URI` object.

`username`: *configuration expression<string>, required if the proxy requires authentication*

Username to access the proxy server.

`"passwordSecretId"`: *configuration expression<secret-id>, required if the proxy requires authentication*

The secret ID of the password to access the proxy server.

For information about supported formats for `secret-id`, see secret-id.

`"secretsProvider"`: *SecretsProvider reference, optional*

The SecretsProvider to query for the proxy's password. For more information, see "*SecretsProvider*".

Default: The route's default secret service. For more information, see "*secrets*".

In the following example, the ClientHandler passes outgoing requests to the proxy server, which requires authentication:

```
"handler": {
  "type": "ClientHandler",
  "config": {
     "proxy": {
        "uri": "http://proxy.example.com:3128",
        "username": "proxyuser",
        "passwordSecretId": "myproxy.secret.id",
        "secretsProvider": "SystemAndEnvSecretStore"
     }
  }
}
```

`"soTimeout"`: *duration string, optional*

Socket timeout, after which stalled connections are destroyed, expressed as a duration

Default: 10 seconds

For information about supported formats for `duration`, see duration.

`"systemProxy"`: *boolean, optional*

Submit outgoing requests to a system-defined proxy, set by the following system properties or their HTTPS equivalents:

- `http.proxyHost`, the host name of the proxy server.
- `http.proxyPort`, the port number of the proxy server. The default is `80`.
- `http.nonProxyHosts`, a list of hosts that should be reached directly, bypassing the proxy.

This property can't be used with a proxy that requires a username and password. Use the property `proxy` instead.

If both `proxy` and `systemProxy` are defined, `proxy` takes precedence.

For more information, see *Java Networking and Proxies* .

Default: False.

`"temporaryStorage"`: *string, optional*

Specifies the heap object to use for temporary buffer storage.

Default: The temporary storage object named `TemporaryStorage`, declared in the top-level heap.

`tls`: *TlsOptions reference, optional*

Configure options for connections to TLS-protected endpoints, based on a "*TlsOptions*" configuration. Define a TlsOptions object inline or in the heap.

Default: Connections to TLS-protected endpoints are not configured.

`"asyncBehavior"`: *enumeration, optional*

Specifies how the HTTP client behaves for asynchronous responses. Set the value to `streaming` or `non_streaming` (not case-sensitive):

- `streaming`: Responses are processed as soon as all headers are received. The entity content is downloaded in a background thread.

  Streaming mode reduces latency and is mandatory for Server-Sent Events (SSE) and the support of very large files (bigger than 2 GB). If thread starvation occurs, consider increasing `numberOfWorkers`, the number of worker threads dedicated to processing outgoing requests.

- `non_streaming`: Responses are processed when the entity content is entirely available.

  Non-streaming mode does not support SSE or very large files. However, it has higher latency, and does not cause thread starvation.

Default: `non_streaming`

`"retries"`: *object, optional*

Enable and configure retry for requests.

When a runtime error occurs while executing the request to the remote server, Microgateway schedules a new execution of the request after the specified delay, until the allowed number of retries is reached or the execution succeeds.

`"enabled"`: *boolean, optional*

Enable retries.

Default: `true`

`"executor"`: *ScheduledExecutorService reference, optional*

The ScheduledExecutorService to use for scheduling delayed execution of the request.

Default: `ScheduledExecutorService`.

See also "*ScheduledExecutorService*".

`"count"`: *number, optional*

The maximum number of retries to perform.

After this threshold is passed and if the request is still not successful, then the ClientHandler propagates the failure.

Default: `5` retries.

`"delay"`: *duration, optional*

The delay to wait before retrying the request.

After a failure to send the request, if the number of retries is below the threshold, a new attempt is scheduled with the executor service after this delay.

Default: `10 seconds`.

For information about supported formats for `duration`, see duration.

The following example configures the handler to retry the request only once, after a 1-minute delay:

```
{
  "retries": {
    "count": 1,
    "delay": "1 minute"
  }
}
```

The following example configures the handler to retry the request at most 20 times, every second:

```
{
  "retries": {
    "count": 20,
    "delay": "1 second"
  }
}
```

The following example configures the handler to retry the request 5 times, every 10 seconds (default values), with a dedicated executor:

```
{
  "retries": {
    "executor": {
      "type": "ScheduledExecutorService",
      "config": {
        "corePoolSize": 20
      }
    }
  }
}
```

`"websocket"`: *object, optional*

Enable upgrade from HTTP or HTTPS protocol to WebSocket protocol.

Important

When Microgateway is running in the Jetty application container, it cannot proxy WebSocket traffic.

```
{
  "websocket": {
    "enabled": boolean,
    "connectionTimeout": duration string,
    "soTimeout": duration string,
    "numberOfSelectors": number,
    "tls": TlsOptions reference
  }
}
```

For more information, see The WebSocket Protocol, RFC6455.

`"enabled"`: *boolean, optional*

Enable upgrade from HTTP protocol and HTTPS protocol to WebSocket protocol.

Default: `false`

`"connectionTimeout"`: *duration string, optional*

The maximum time allowed to establish a WebSocket connection.

Default: The value of handler's main `connectionTimeout`.

For information about supported formats for `duration`, see duration.

`"soTimeout"`: *duration string, optional*

The time after which stalled connections are destroyed.

Tip

If there can be long delays between messages, consider increasing this value. Alternatively, keep the connection active by using WebSocket ping messages in your application.

Default: The value of handler's main `soTimeout`.

For information about supported formats for `duration`, see duration.

`"numberOfSelectors"`: *number, optional*

The maximum number of worker threads.

In deployments with a high number of simultaneous connections, consider increasing the value of this property.

Default: `2`

`tls`: *TlsOptions reference, optional*

Configure options for connections to TLS-protected endpoints, based on a "*TlsOptions*" configuration. Define a TlsOptions object inline or in the heap.

Default: Use TlsOptions defined for the handler

2.2.4. Example

The following object configures a `ClientHandler` named `Client`:

```
{
  "name": "Client",
  "type": "ClientHandler",
  "config": {
    "hostnameVerifier": "STRICT",
    "tls": {
      "type": "TlsOptions",
      "config": {
        "sslContextAlgorithm": "TLSv1.2",
        "keyManager": {
          "type": "KeyManager",
          "config": {
            "keystore": {
              "type": "KeyStore",
              "config": {
                "url": "file://${env['HOME']}/keystore.jks",
                "passwordSecretId": "keymanager.keystore.secret.id",
                "secretsProvider": "SystemAndEnvSecretStore"
              }
            },
            "passwordSecretId": "keymanager.secret.id",
            "secretsProvider": "SystemAndEnvSecretStore"
          }
        },
        "trustManager": {
          "type": "TrustManager",
          "config": {
            "keystore": {
              "type": "KeyStore",
              "config": {
                "url": "file://${env['HOME']}/truststore.jks",
                "passwordSecretId": "trustmanager.keystore.secret.id",
                "secretsProvider": "SystemAndEnvSecretStore"
              }
            }
          }
        }
      }
    }
  }
}
```

2.2.5. More Information

org.forgerock.openig.handler.ClientHandler

Chapter 2.3

DesKeyGenHandler

2.3.1. Description

Generates a DES key for use with AM.

2.3.2. Usage

```
{
    "name": string,
    "type": "DesKeyGenHandler"
}
```

2.3.3. More Information

org.forgerock.openig.handler.DesKeyGenHandler

Chapter 2.4
DispatchHandler

2.4.1. Description

When a request is handled, the first condition in the list of conditions is evaluated. If the condition expression yields `true`, the request is dispatched to the associated handler with no further processing. Otherwise, the next condition in the list is evaluated.

2.4.2. Usage

```
{
  "name": string,
  "type": "DispatchHandler",
  "config": {
    "bindings": [
      {
        "condition": runtime expression<boolean>,
        "handler": Handler reference,
        "baseURI": runtime expression<uri string>,
      }, ...
    ]
  }
}
```

2.4.3. Properties

`"bindings"`: *array of objects, required*

A list of bindings of conditions and associated handlers to dispatch to.

`"condition"`: *runtime expression<boolean>, optional*

An inline expression to define a condition based on the request, context, or Microgateway runtime environment, such as system properties or environment variables.

Conditions are defined using Microgateway expressions, as described in "*Expressions*", and are evaluated as follows:

- If the condition evaluates to `true`, the request is dispatched to the associated handler.
- If the condition evaluates to `false`, the next condition in the list is evaluated.

- If no condition is specified, the request is dispatched unconditionally to the associated handler.

Default: No condition is specified.

`"handler"`: *Handler reference, required*

A handler to dispatch the request to if the associated condition yields `true`, or if there is no associated condition.

Provide either the name of a handler object defined in the heap, or an inline handler configuration object.

See also "Handlers".

`"baseURI"`: *runtime expression<uri string>, optional*

A base URI that overrides the existing request URI. Only scheme, host, and port are used in the supplied URI.

The result of the expression must be a string that represents a valid URI, but is not a real `java.net.URI` object. For example, it would be incorrect to use `${request.uri}`, which is not a String but a MutableUri.

In the following example, the binding condition looks up the hostname of the request. If it finds a match, the value is used for the `baseURI`. Otherwise, the default value is used:

```
{
  "properties": {
    "uris": {
      "app1.example.com": {
        "baseURI": "http://backend1:8080/"
      },
      "app2.example.com": {
        "baseURI": "http://backend2:8080/"
      },
      "default": {
        "baseURI": "http://backend3:8080/"
      }
    }
  },
  "handler": {
    "type": "DispatchHandler",
    "config": {
      "bindings": [
        {
          "condition": "${not empty uris[contexts.router.originalUri.host]}",
          "baseURI": "${uris[contexts.router.originalUri.host].baseURI}",
          "handler": "ReverseProxyHandler"
        },
        {
          "baseURI": "${uris['default'].baseURI}",
          "handler": "ReverseProxyHandler"
        }
      ]
    }
  }
}
```

Default: No change to the base URI

2.4.4. More Information

org.forgerock.openig.handler.DispatchHandler

"*Expressions*"

Chapter 2.5

ReverseProxyHandler

2.5.1. Description

Relays requests to protected applications.

If the request is to upload or download a large file to or from the protected application, consider setting `asyncBehavior` to `streaming`, and increasing the value of `soTimeout`.

If Microgateway fails to connect to the protected application, the ReverseProxyHandler does not propagate the error along the chain. Instead, it changes the runtime exception into a `502 Bad Gateway` response.

To submit requests to third-party services such as AM, Elasticsearch or Splunk from an audit event handler, or HTTP APIs, use a "*ClientHandler*", to manage connection failures differently.

2.5.2. Usage

```
{
  "name": string,
  "type": "ReverseProxyHandler",
  "config": {
    "connections": number,
    "disableReuseConnection": boolean,
    "hostnameVerifier": configuration expression<enumeration>,
    "soTimeout": duration string,
    "connectionTimeout": duration string,
    "connectionTimeToLive": duration string,
    "numberOfWorkers": number,
    "proxy": Server reference,
    "systemProxy": boolean,
    "temporaryStorage": string,
    "tls": TlsOptions reference,
    "asyncBehavior": enumeration,
    "retries": object,
    "websocket": object
  }
}
```

2.5.3. Properties

`"connections"`: *number, optional*

The maximum number of connections in the HTTP client connection pool.

Default: 64

`"connectionTimeout"`: *duration string, optional*

Amount of time to wait to establish a connection, expressed as a duration

Default: 10 seconds

For information about supported formats for `duration`, see duration.

`"connectionTimeToLive"`: *duration string, optional*

Amount of time before a reusable pooled connection expires.

Set this property to expire reusable pooled connections after a fixed duration. For example, to prevent the reuse of connections set this property in routes for applications where the IP address (baseURI) is not stable or can change.

Default: Unlimited

For information about supported formats for `duration`, see duration.

`"disableReuseConnection"`: *boolean, optional*

Whether to disable connection reuse.

Default: `false`

`"hostnameVerifier"`: *configuration expression<enumeration>, optional*

Way to handle hostname verification for outgoing SSL connections. Use one of the following values:

- `ALLOW_ALL`: Allow a certificate issued by a trusted CA for any hostname or domain to be accepted for a connection to any domain.

  Caution

  This setting allows a certificate issued for one company to be accepted as a valid certificate for another company.

  To prevent the compromise of TLS connections, use this setting in development mode only. In production, use `STRICT`.

- `STRICT`: Match the hostname either as the value of the the first CN, or any of the subject-alt names.

A wildcard can occur in the CN, and in any of the subject-alt names. Wildcards match one domain level, so `*.example.com` matches `www.example.com` but not `some.host.example.com`.

Default: `STRICT`

`"numberOfWorkers"`: *number, optional*

The number of worker threads dedicated to processing outgoing requests.

Increasing the value of this attribute can be useful in deployments where a high number of simultaneous connections remain open, waiting for protected applications to respond.

Default: One thread per CPU available to the JVM.

`"proxy"`: *Server reference, optional*

A proxy server to which requests can be submitted. Use this property to relay requests to other parts of the network, for example, to submit requests from an internal network to the internet.

If both `proxy` and `systemProxy` are defined, `proxy` takes precedence.

`uri`: *configuration expression<uri string>, required*

URI of a server to use as a proxy for outgoing requests.

The result of the expression must be a string that represents a valid URI, but is not a real `java.net.URI` object.

`username`: *string, required if the proxy requires authentication*

Username to access the proxy server.

`"passwordSecretId"`: *configuration expression<secret-id>, required if the proxy requires authentication*

The secret ID of the password to access the proxy server.

For information about supported formats for `secret-id`, see secret-id.

In the following example, the ReverseProxyHandler passes outgoing requests to the proxy server, which requires authentication:

```
"handler": {
  "type": "ReverseProxyHandler",
  "config": {
     "proxy": {
        "uri": "http://proxy.example.com:3128",
        "username": "proxyuser",
        "passwordSecretId": "myproxy.secret.id"
     }
  }
}
```

`"soTimeout"`: *duration string, optional*

Socket timeout, after which stalled connections are destroyed, expressed as a duration

> **Tip**
>
> If `SocketTimeoutException` errors occur in the logs when you try to upload or download large files, consider increasing `soTimeout`.

Default: 10 seconds

For information about supported formats for `duration`, see duration.

`"systemProxy"`: *boolean, optional*

Submit outgoing requests to a system-defined proxy, set by the following system properties or their HTTPS equivalents:

- `http.proxyHost`, the host name of the proxy server.
- `http.proxyPort`, the port number of the proxy server. The default is `80`.
- `http.nonProxyHosts`, a list of hosts that should be reached directly, bypassing the proxy.

This property can't be used with a proxy that requires a username and password. Use the property `proxy` instead.

If both `proxy` and `systemProxy` are defined, `proxy` takes precedence.

For more information, see *Java Networking and Proxies* .

Default: False.

`"temporaryStorage"`: *string, optional*

Specifies the heap object to use for temporary buffer storage.

Default: The temporary storage object named `TemporaryStorage`, declared in the top-level heap.

`tls`: *TlsOptions reference, optional*

Configure options for connections to TLS-protected endpoints, based on a "*TlsOptions*" configuration. Define a TlsOptions object inline or in the heap.

Default: Connections to TLS-protected endpoints are not configured.

`"asyncBehavior"`: *enumeration, optional*

Specifies how the HTTP client behaves for asynchronous responses. Set the value to `streaming` or `non_streaming` (not case-sensitive):

- `streaming`: Responses are processed as soon as all headers are received. The entity content is downloaded in a background thread.

  Streaming mode reduces latency and is mandatory for Server-Sent Events (SSE) and the support of very large files (bigger than 2 GB). If thread starvation occurs, consider increasing `numberOfWorkers`, the number of worker threads dedicated to processing outgoing requests.

- `non_streaming`: Responses are processed when the entity content is entirely available.

  Non-streaming mode does not support SSE or very large files. However, it has higher latency, and does not cause thread starvation.

> **Tip**
>
> If timeout errors occur in the logs, consider setting `soTimeout` to limit the timeout, and setting `asyncBehavior` to `non_streaming`.

Default: `non_streaming`

`"retries"`: *object, optional*

Enable and configure retry for requests.

When a runtime error occurs while executing the request to the remote server, Microgateway schedules a new execution of the request after the specified delay, until the allowed number of retries is reached or the execution succeeds.

`"enabled"`: *boolean, optional*

Enable retries.

Default: `true`

`"executor"`: *ScheduledExecutorService reference, optional*

The ScheduledExecutorService to use for scheduling delayed execution of the request.

Default: `ScheduledExecutorService`.

See also "*ScheduledExecutorService*".

`"count"`: *number, optional*

The maximum number of retries to perform.

After this threshold is passed and if the request is still not successful, then the ReverseProxyHandler returns a `502 Bad Gateway` response.

Default: `5` retries.

`"delay"`: ***duration, optional***

The delay to wait before retrying the request.

After a failure to send the request, if the number of retries is below the threshold, a new attempt is scheduled with the executor service after this delay.

Default: `10 seconds`.

For information about supported formats for `duration`, see duration.

The following example configures the handler to retry the request only once, after a 1-minute delay:

```
{
  "retries": {
    "count": 1,
    "delay": "1 minute"
  }
}
```

The following example configures the handler to retry the request at most 20 times, every second:

```
{
  "retries": {
    "count": 20,
    "delay": "1 second"
  }
}
```

The following example configures the handler to retry the request 5 times, every 10 seconds (default values), with a dedicated executor:

```
{
  "retries": {
    "executor": {
      "type": "ScheduledExecutorService",
      "config": {
        "corePoolSize": 20
      }
    }
  }
}
```

`"websocket"`: ***object, optional***

Enable upgrade from HTTP or HTTPS protocol to WebSocket protocol.

Important

When Microgateway is running in the Jetty application container, it cannot proxy WebSocket traffic.

```
{
  "websocket": {
    "enabled": boolean,
    "connectionTimeout": duration string,
    "soTimeout": duration string,
    "numberOfSelectors": number,
    "tls": TlsOptions reference
  }
}
```

For more information, see The WebSocket Protocol, RFC6455.

`"enabled"`: *boolean, optional*

Enable upgrade from HTTP protocol and HTTPS protocol to WebSocket protocol.

Default: `false`

`"connectionTimeout"`: *duration string, optional*

The maximum time allowed to establish a WebSocket connection.

Default: The value of handler's main `connectionTimeout`.

For information about supported formats for `duration`, see duration.

`"soTimeout"`: *duration string, optional*

The time after which stalled connections are destroyed.

Tip

If there can be long delays between messages, consider increasing this value. Alternatively, keep the connection active by using WebSocket ping messages in your application.

Default: The value of handler's main `soTimeout`.

For information about supported formats for `duration`, see duration.

`"numberOfSelectors"`: *number, optional*

The maximum number of worker threads.

In deployments with a high number of simultaneous connections, consider increasing the value of this property.

Default: `2`

`tls`: *TlsOptions reference, optional*

Configure options for connections to TLS-protected endpoints, based on a "*TlsOptions*" configuration. Define a TlsOptions object inline or in the heap.

Default: Use TlsOptions defined for the handler

2.5.4. More Information

org.forgerock.openig.handler.ReverseProxyHandler

Chapter 2.6

ResourceHandler

Serves static content from a directory.

2.6.1. Usage

```
{
  "name": string,
  "type": "ResourceHandler",
  "config": {
    "directories": [ configuration expression<string>, ... ],
    "basePath": configuration expression<string>,
    "welcomePages": [ configuration expression<string>, ... ]
  }
}
```

2.6.2. Properties

`"directories"`: *array of configuration expressions<string>, required*

A list of one or more directories in which to search for static content.

When multiple directories are specified in an array, the directories are searched in the listed order.

`"basePath"`: *configuration expression<string>, required if the route is not `/`*

The base path of the incoming request for static content.

To specify no base path, leave this property out of the configuration, or specify it as `"basePath": ""` or `"basePath": "/"`.

Default: `""`.

`"welcomePages"`: *array of configuration expressions<string>, optional*

A set of static content to serve from one of the specified directories when no specific resource is requested.

When multiple sets of static content are specified in an array, the sets are searched for in the listed order. The first set that is found is used.

Default: Empty list

2.6.3. Example

The following example serves requests to `http://microgateway.example.com:8080` with the static file `index.html` from `/path/to/static/pages/`:

```
{
  "name": "StaticWebsite",
  "type": "ResourceHandler",
  "config": {
    "directories": ["/path/to/static/pages"],
    "welcomePages": ["index.html"]
  }
}
```

When the `basePath` is `/website`, the example serves requests to `http://microgateway.example.com:8080/website`:

```
{
  "name": "StaticWebsite",
  "type": "ResourceHandler",
  "config": {
    "directories": ["/path/to/static/pages"],
    "basePath": "/website",
    "welcomePages": ["index.html"]
  }
}
```

2.6.4. More Information

org.forgerock.openig.handler.resources.ResourceHandler

org.forgerock.http.protocol.Entity

Chapter 2.7
Route

2.7.1. Description

Routes are configuration files that you add to Microgateway to manage requests. They are flat files in JSON format. You can add routes in the following ways:

- Manually into the filesystem.
- Through Common REST commands.

Every route must call a handler to process requests and produce responses to requests.

When a route has a condition, it can handle only requests that meet the condition. When a route has no condition, it can handle any request.

Routes inherit settings from their parent configuration. This means that you can configure global objects in the `config.json` heap, for example, and then reference the objects by name in any other Microgateway configuration.

2.7.2. Usage

```
{
  "handler": Handler reference or inline Handler declaration,
  "heap": [ configuration object, ... ],
  "condition": runtime expression<boolean>,
  "name": string,
  "secrets": configuration object,
  "session": JwtSession object,
  "auditService": AuditService object,
  "globalDecorators": configuration object,
  "decorator name": decorator configuration object
}
```

2.7.3. Properties

`"handler"`: *Handler reference, required*

For this route, dispatch the request to this handler.

Provide either the name of a Handler object defined in the heap, or an inline Handler configuration object.

See also "Handlers".

`"heap"`: *array of configuration objects, optional*

Heap object configuration for objects local to this route.

Objects referenced but not defined here are inherited from the parent.

You can omit an empty array. If you only have one object in the heap, you can inline it as the handler value.

See also "*Heap Objects*".

`"condition"`: *runtime expression<boolean>, optional*

An inline expression to define a condition based on the request, context, or Microgateway runtime environment, such as system properties or environment variables.

Conditions are defined using Microgateway expressions, as described in "*Expressions*", and are evaluated as follows:

- If the condition evaluates to `true`, the request is dispatched to the route.
- If the condition evaluates to `false`, the condition for the next route in the configuration is evaluated.
- If no condition is specified, the request is dispatched unconditionally to the route.

An external request can never match a condition that uses the reserved administrative route. Therefore, routes that use these conditions are effectively ignored. For example, if `/openig` is the administrative route, a route with the following condition would effectively be ignored: `${matches(request.uri.path, '^/openig/my/path')}`.

Default: No condition is specified.

`"name"`: *string, optional*

Name for the route.

The Router uses the `name` property to order the routes in the configuration. If the route does not have a `name` property, the Router uses the route ID.

The route ID is managed as follows:

- When you add a route manually to the routes folder, the route ID is the value of the `_id` field. If there is no `_id` field, the route ID is the filename of the added route.
- When you add a route through the Common REST endpoint, the route ID is the value of the mandatory `_id` field.

Caution

The filename of a route cannot be `default.json`, and the route's `name` property and route ID cannot be `default`.

Default: route ID

`"secrets"`: *configuration object, optional*

An object that configures an inline array of one or more secret stores, as defined in "*secrets*".

`"session"`: *JwtSession object, optional*

Stateless session implementation for this route. Define a JwtSession object inline or in the heap.

When a request enters the route, Microgateway builds a new session object for the route. The session content is available to the route's downstream handlers and filters. Session content available in the ascending configuration (a parent route or `config.jso`) is not available in the new session.

When the response exits the route, the session content is serialized as a secure JWT that is encrypted and signed, and the resulting JWT string is placed in a cookie. Session information set inside the route is no longer available. The `session` references the previous session object.

Default: Do not change the session storage implementation.

`"auditService"`: *AuditService object, optional*

Provide an audit service for the route. Provide either the name of an AuditService object defined in the heap, or an inline AuditService configuration object.

Default: No auditing of a configuration. The NoOpAuditService provides an empty audit service to the top-level heap and its child routes.

`"globalDecorators"`: *one or more decorations, optional*

Decorate all compatible objects in the route with one or more decorators referred to by the decorator name, and provide the configuration as described in "Decorators":

```
"globalDecorators":  {
  "decorator name": "decoration configuration"
  ...
}
```

The following object decorates all compatible objects in the route with a capture and timer decorator:

```
"globalDecorators":  {
  "capture": "all",
  "timer": true
}
```

Default: No decoration.

"*decorator name*": *decoration, optional*

Decorate the main handler of this route with a decorator referred to by the decorator name, and provide the configuration as described in "Decorators".

Default: No decoration.

2.7.4. Metrics

This section describes the metrics that Route publishes to the Prometheus Scrape Endpoint and the Common REST Monitoring Endpoint.

2.7.4.1. Route Metrics at the Prometheus Scrape Endpoint

Route metrics at the Prometheus Scrape Endpoint have the following labels:

- `name`: Route name, for example, `My Route`.

  If the router was declared with a default handler, then its metrics are published through the route named `default`.

- `route`: Route identifier, for example, `my-route`.

- `router`: Fully qualified name of the router, for example, `gateway.main-router`.

The following table summarizes the recorded metrics:

Name	Type [a]	Description
`ig_route_request_active`	Gauge	Number of requests being processed.
`ig_route_request_total`	Counter	Number of requests processed by the router or route since it was deployed.
`ig_route_response_error_total`	Counter	Number of responses that threw an exception.
`ig_route_response_null_total`	Counter	Number of responses that were not handled by Microgateway.
`ig_route_response_status_total`	Counter	Number of responses by HTTP status code family. The `family` label depends on the HTTP status code: • Informational (1xx) • Successful (2xx) • Redirection (3xx)

Name	Type [a]	Description
		• Client_error (4xx) • Server_error (5xx) • Unknown (status code >= 600)
`ig_route_response_time`	Summary	A summary of response time observations.

[a]As described in "*Monitoring Types*"

For more information about the the Prometheus Scrape Endpoint, see "Prometheus Scrape Endpoint".

2.7.4.2. Route Metrics at the Common REST Monitoring Endpoint

Route metrics at the Common REST Monitoring Endpoint are published with an `_id` in the following pattern:

- *`heap.router-name.`*`route.`*`route-name.metric`*

The following table summarizes the recorded metrics:

Name	Type [a]	Description
`request`	Counter	Number of requests processed by the router or route since it was deployed.
`request.active`	Gauge	Number of requests being processed by the router or route at this moment.
`response.error`	Counter	Number of responses that threw an exception.
`response.null`	Counter	Number of responses that were not handled by Microgateway.
`response.status.client_error`	Counter	Number of responses with an HTTP status code `400`-`499`, indicating client error.
`response.status.informational`	Counter	Number of responses with an HTTP status code `100`-`199`, indicating that they are provisional responses.
`response.status.redirection`	Counter	Number of responses with an HTTP status code `300`-`399`, indicating a redirect.
`response.status.server_error`	Counter	Number of responses with an HTTP status code `500`-`599`, indicating server error.
`response.status.successful`	Counter	Number of responses with an HTTP status code `200`-`299`, indicating success.
`response.status.unknown`	Counter	Number of responses with an HTTP status code `600`-`699`, indicating that a request failed completely and was not executed.

Name	Type [a]	Description
`response.time`	Timer	Time-series summary statistics.

[a] As described in "*Monitoring Types*"

For more information about the the Common REST Monitoring Endpoint, see "Common REST Monitoring Endpoint".

Chapter 2.8

Router

2.8.1. Description

A Router is a handler that performs the following tasks:

- Defines the routes directory and loads routes into the configuration.
- Depending on the scanning interval, periodically scans the routes directory and updates the Microgateway configuration when routes are added, removed, or changed. The router updates the Microgateway configuration without needing to restart Microgateway or access the route.
- Manages an internal list of routes, where routes are ordered lexicographically by route name. If a route is not named, then the route ID is used instead. For more information, see "*Route*".
- Routes requests to the first route in the internal list of routes, whose condition is satisfied.

  Because the list of routes is ordered lexicographically by route name, name your routes with this in mind:

  - If a request satisfies the condition of more than one route, it is routed to the first route in the list whose condition is met.
  - Even if the request matches a later route in the list, it might never reach that route.

  If a request does not satisfy the condition of any route, it is routed to the default handler if one is configured.

The router does not have to know about specific routes in advance - you can configure the router first and then add routes while Microgateway is running.

2.8.2. Usage

```
{
    "name": "Router",
    "type": "Router",
    "config": {
        "defaultHandler": Handler reference,
        "directory": expression,
        "scanInterval": duration string or integer
    }
}
```

An alternative value for type is RouterHandler.

2.8.3. Properties

`"defaultHandler"`: *Handler reference, optional*

Handler to use when a request does not satisfy the condition of any route.

Provide either the name of a handler object defined in the heap, or an inline handler configuration object.

Default: If no default route is set either here or in the route configurations, Microgateway aborts the request with an internal error.

See also "Handlers".

`"directory"`: *expression, optional*

Directory from which to load route configuration files.

Default: The default directory for route configuration files, at `/path/to/microservices/identity-gateway` (on Windows, `%appdata%\OpenIG`).

With the following example, route configuration files are loaded from `/path/to/safe/routes` instead of from the default directory:

```
{
  "type": "Router",
  "config": {
    "directory": "/path/to/safe/routes",
  }
}
```

Important

If you define multiple routers, configure `directory` so that the routers load route configuration files from different directories.

An infinite route-loading sequence is triggered when a router starts a route that, directly or indirectly, starts another router, which then loads route configuration files from the same directory.

See also "*Expressions*".

`"scanInterval"`: *duration string or integer, optional*

Time interval at which Microgateway scans the specified directory for changes to routes. When a route is added, removed, or changed, the router updates the Microgateway configuration without needing to restart Microgateway or access the route.

When an integer is used for the `scanInterval`, the time unit is seconds.

To load routes at startup only, and prevent changes to the configuration if the routes are changed, set the scan interval to `disabled`.

Default: 10 seconds

For information about supported formats for `duration`, see duration.

2.8.4. Metrics

This section describes the metrics that Router publishes to the Prometheus Scrape Endpoint and the Common REST Monitoring Endpoint.

2.8.4.1. Router Metrics at the Prometheus Scrape Endpoint

Router metrics at the Prometheus Scrape Endpoint have the following labels:

- `fully_qualified_name`: Fully qualified name of the router, for example, `gateway.main-router`.
- `heap`: Name of the heap in which this router is declared, for example, `gateway`.
- `name`: Simple name declared in router configuration, for example, `main-router`.

The following table summarizes the recorded metrics:

Name	Type [a]	Description
`ig_router_deployed_routes`	`Gauge`	Number of routes deployed in the configuration.

[a]As described in "*Monitoring Types*"

For more information about the the Prometheus Scrape Endpoint, see "Prometheus Scrape Endpoint".

2.8.4.2. Router Metrics at the Common REST Monitoring Endpoint

Router metrics at the Common REST Monitoring Endpoint are JSON objects, with the following form:

- *`[heap name].[router name]`*`.deployed-routes`

The following table summarizes the recorded metrics:

Name	Type [a]	Description
`deployed-routes`	`Gauge`	Number of routes deployed in the configuration.

[a]As described in "*Monitoring Types*"

For more information about the the Common REST Monitoring Endpoint, see "Common REST Monitoring Endpoint".

2.8.5. More Information

org.forgerock.openig.handler.router.RouterHandler

Chapter 2.9
ScriptableHandler

2.9.1. Description

Creates a response to a request by executing a script.

Scripts must return either a Promise<Response, NeverThrowsException> or a Response.

This section describes the usage of ScriptableHandler. For information about script properties, available global objects, and automatically imported classes, see "*Scripts*".

2.9.2. Usage

```
{
    "name": string,
    "type": "ScriptableHandler",
    "config": {
        "type": string,
        "file": expression,                    // Use either "file"
        "source": string or array of strings   // or "source", but not both
        "args": object,
        "clientHandler": Handler reference
    }
}
```

2.9.3. Properties

For information about properties for ScriptableHandler, see "*Scripts*".

2.9.4. More Information

org.forgerock.openig.handler.ScriptableHandler

Chapter 2.10

SequenceHandler

2.10.1. Description

Processes a request through a sequence of handlers. This allows multi-request processing such as retrieving a form, extracting form content (for example, nonce) and submitting in a subsequent request. Each `handler` in the `bindings` is dispatched to in order; the binding `postcondition` determines if the sequence should continue.

2.10.2. Usage

```
{
    "name": string,
    "type": "SequenceHandler",
    "config": {
        "bindings": [
            {
                "handler": Handler reference,
                "postcondition": runtime expression<boolean>
            }
        ]
    }
}
```

2.10.3. Properties

`"bindings"`: *array of objects, required*

A list of bindings of handler and postcondition to determine that sequence continues.

`"handler"`: *Handler reference, required*

Dispatch to this handler.

Either the name of the handler heap object to dispatch to, or an inline Handler configuration object.

See also "Handlers".

`"postcondition"`: *runtime expression<boolean>, optional*

If the expression evaluates to `true`, the sequence continues. If a condition is not specified, the sequence continues unconditionally.

Default: No condition is specified.

See also "*Expressions*".

2.10.4. More Information

org.forgerock.openig.handler.SequenceHandler

Chapter 2.11

StaticResponseHandler

2.11.1. Description

Creates a response to a request statically or based on something in the context.

2.11.2. Usage

```
{
     "name": string,
     "type": "StaticResponseHandler",
     "config": {
         "status": number,
         "reason": string,
         "version": string,
         "headers": {
             name: [ runtime expression<string>, ... ], ...
         },
         "entity": runtime expression<string>
     }
}
```

2.11.3. Properties

`"status"`: *number, required*

The response status code (for example, 200).

`"reason"`: *string, optional*

The response status reason (for example, `"OK"`).

`"version"`: *string, optional*

Protocol version. Default: `"HTTP/1.1"`.

`"headers"`: *array of objects, optional*

The `name` specifies the header name. Its value is an array of runtime expressions to evaluate as header values.

`"entity"`: *runtime expression<string>, optional*

The message entity to include in the request.

If present, it must conform to the `Content-Type` header and set the content length header automatically.

See also "*Expressions*".

2.11.4. Example

```
{
     "name": "ErrorHandler",
     "type":"StaticResponseHandler",
     "config": {
        "status": 500,
        "reason": "Error",
        "entity": "<html>
                   <h2>Epic #FAIL</h2>
                   </html>",
        "headers": {
          "content-type": [
            "text/html"
          ]
        }
     }
}
```

2.11.5. More Information

org.forgerock.openig.handler.StaticResponseHandler

Part 3
Filters

Filter objects intercept requests and responses during processing, and change them as follows:

- Leave the request, response, and contexts unchanged. For example, the filter can simply can log the context as it passes through the filter.
- In the request flow, change any aspect of the request (such as the URL, headers, or entity), or replace the request with a new Request object.
- In the response flow, change any aspect of the response (such as the status, headers, or entity), or return a new Response instance

This part describes the filters provided in Microgateway.

Chapter 3.1

AssignmentFilter

3.1.1. Description

Verifies that a specified condition is met. If the condition is met or if no condition is specified, the value is assigned to the target. Values can be assigned before the request is handled and after the response is handled.

3.1.2. Usage

```
{
  "name": string,
  "type": "AssignmentFilter",
  "config": {
    "onRequest": [
      {
        "condition": runtime expression<boolean>,
        "target": lvalue-expression,
        "value": runtime expression
      }, ...
    ],
    "onResponse": [
      {
        "condition": runtime expression<boolean>,
        "target": lvalue-expression,
        "value": runtime expression
      }, ...
    ]
  }
}
```

3.1.3. Properties

`"onRequest"`: *array of objects, optional*

Defines a list of assignment bindings to evaluate before the request is handled.

`"onResponse"`: *array of objects, optional*

Defines a list of assignment bindings to evaluate after the response is handled.

`"condition"`: *runtime expression<boolean>, optional*

If the expression evaluates `true`, the value is assigned to the target. If no condition is specified, the value is assigned to the target unconditionally.

Default: No condition is specified.

See also "*Expressions*".

`"target"`: *lvalue-expression, required*

Expression that yields the target object whose value is to be set.

See also "*Expressions*".

`"value"`: *runtime expression, optional*

The value to be set in the target. The value can be a string, information from the context, or even a whole map of information.

See also "*Expressions*".

3.1.4. Example

3.1.4.1. Adding Info To a Session

The following example assigns a value to a session. Add the filter to a route to prevent Microgateway from clearing up empty JWTSession cookies:

```
{
  "type": "AssignmentFilter",
  "config": {
    "onRequest": [{
      "target": "${session.authUsername}",
      "value": "I am root"
    }]
  }
}
```

3.1.4.2. Capturing and Storing Login Credentials

The following example captures credentials and stores them in the Microgateway session during a login request. Notice that the credentials are captured on the request but are not marked as valid until the response returns a positive 302. The credentials could then be used to log a user in to a different application:

```
{
  "name": "PortalLoginCaptureFilter",
  "type": "AssignmentFilter",
  "config": {
    "onRequest": [
      {
        "target": "${session.authUsername}",
        "value": "${request.form['username'][0]}"
      },
      {
        "target": "${session.authPassword}",
        "value": "${request.form['password'][0]}"
      },
      {
        "comment": "Authentication has not yet been confirmed.",
        "target": "${session.authConfirmed}",
        "value": "${false}"
      }
    ],
    "onResponse": [
      {
        "condition": "${response.status.code == 302}",
        "target": "${session.authConfirmed}",
        "value": "${true}"
      }
    ]
  }
}
```

3.1.5. More Information

org.forgerock.openig.filter.AssignmentFilter

Chapter 3.2

CapturedUserPasswordFilter

3.2.1. Description

This filter makes an AM password available to Microgateway in the following steps:

- Checks for the presence of the SessionInfoContext context, at `${contexts.amSession}`.
  - If the context is not present, or if `sunIdentityUserPassword` is `null`, the CapturedUserPasswordFilter collects session info and properties from AM.
  - If the context is present and `sunIdentityUserPassword` is not `null`, the CapturedUserPasswordFilter uses that value for the password.
- The CapturedUserPasswordFilter decrypts the password and stores it in the CapturedUserPasswordContext, at `${contexts.capturedPassword}`.

Supported with AM 5 and later versions, and with AM 6 and later versions when the `AES` keyType is used to decrypt the password.

3.2.2. Usage

```
{
  "name": string,
   "type": "CapturedUserPasswordFilter",
   "config": {
     "amService": AmService reference,
     "keySecretId": configuration expression<secret-id>,
     "keyType": configuration expression<string>,
     "secretsProvider": SecretsProvider reference,
     "ssoToken": runtime expression<string>
   }
}
```

3.2.3. Properties

`"amService"`: *AmService reference, required*

The AmService heap object to use for the following properties:

- `agent`, the credentials of an Microgateway agent in AM. When the agent is authenticated, the token can be used for tasks such as getting the user's profile, making policy evaluations, and connecting to the AM notification endpoint.
- `url`, the URL of an AM service to use for session token validation and authentication.
- `amHandler`, the handler to use when communicating with AM to validate the token in the incoming request.
- `realm`: Realm of the Microgateway agent in AM.
- `version`: The version of the AM server.

This filter is compatible with AM version 5.5 or higher. If `version` is not set, the default version is AM 5 and an error is thrown.

See also, "*AmService*".

`"keySecretId"`: *configuration expression<secret-id>, required*

The secret ID for the key required decrypt the AM password.

For information about supported formats for `secret-id`, see secret-id.

`"keyType"`: *configuration expression<enumeration>, required*

Algorithm to decrypt the AM password. Use one of the following values:

- `DES` for DES/ECB/NoPadding
- `AES` AES for JWT-based AES_128_CBC_HMAC_SHA_256 encryption, available from AM 6.

  For more information, see AES_128_CBC_HMAC_SHA_256 in the IETF *JSON Web Algorithms*.

Default: `DES`

`"secretsProvider"`: *SecretsProvider reference, optional*

The SecretsProvider object to query for the JWT session signing or encryption keys. For more information, see "*SecretsProvider*".

Default: The route's default secret service. For more information, see "*secrets*".

`"ssoToken"`: *runtime expression<string>, required*

Location of the AM SSO token.

Default: `${request.cookies['AmService-ssoTokenHeader'][0].value}`, where `AmService-ssoTokenHeader` is the name of the header or cookie where the AmService expects to find SSO tokens.

3.2.4. More Information

org.forgerock.openig.openam.CapturedUserPasswordFilter

org.forgerock.openig.openam.CapturedUserPasswordContext

"*CapturedUserPasswordContext*"

"*SessionInfoFilter*"

Chapter 3.3
ClientCredentialsOAuth2ClientFilter

3.3.1. Description

Authenticates OAuth 2.0 clients by using the client's OAuth 2.0 credentials to obtain an access_token from an authorization server, and injecting the access_token into the inbound request as a Bearer Authorization header.

The filter obtains the client's access_token by using the `client_credentials` grant type, where the credentials are sent with the `client_secret_basic` method. The filter refreshes the access_token as required.

Use this filter in a service-to-service context, where services need to access resources protected by OAuth 2.0.

3.3.2. Usage

```
{
  "name": string,
  "type": "ClientCredentialsOAuth2ClientFilter",
  "config": {
    "clientId": configuration expression<sting>,
    "clientSecretId": configuration expression<secret-id>,
    "secretsProvider": SecretsProvider reference,
    "tokenEndpoint": configuration expression<url>,
    "scopes": [ configuration expression<string>, ... ],
    "handler": Handler reference or inline Handler declaration
  }
}
```

3.3.3. Properties

`"clientId"`: *configuration expression<string>, required*

The ID of the OAuth 2.0 client registered with the authorization server.

`"clientSecretId"`: *configuration expression<secret-id>, required*

The ID to use when querying the `secretsProvider` for the client secret.

`"secretsProvider"`: *SecretsProvider reference, required*

The "*SecretsProvider*" to use to resolve queried secrets, such as passwords and cryptographic keys. Provide either the name of a SecretsProvider object defined in the heap, or specify a SecretsProvider object inline.

`"tokenEndpoint"`: *configuration expression<url>, required*

The URL to the authorization server's OAuth 2.0 token endpoint.

`"scopes"`: *array of configuration expression<string>, optional*

Array of scope strings to request from the authorization server.

Default: Empty, request no scopes.

`"handler"`: *Handler reference or inline Handler declaration, optional*

The Handler to use to access the authorization server's OAuth 2.0 token endpoint. Provide either the name of a handler object defined in the heap, or specify a handler object inline.

Default: ClientHandler

3.3.4. Log Level

To facilitate debugging secrets for this filter, in `logback.xml` add a logger defined by the fully qualified package name of the secrets API backend. The following line in `logback.xml` sets the log level to `ALL`:

```
<logger name="org.forgerock.secrets.oauth2" level="ALL">
```

3.3.5. More Information

org.forgerock.openig.filter.oauth2.client.ClientCredentialsOAuth2ClientFilterHeaplet

org.forgerock.openig.filter.oauth2.OAuth2ResourceServerFilterHeaplet

The OAuth 2.0 Authorization Framework

OAuth 2.0 Bearer Token Usage

Chapter 3.4
ConditionalFilter

3.4.1. Description

Verifies that a specified condition is met. If the condition is met, the request is dispatched to a delegate Filter. Otherwise, the delegate Filter is skipped.

Use `ConditionalFilter` to easily use or skip a Filter depending on whether a condition is met. To easily use or skip a set of Filters, use a `ChainOfFilters` as the delegate Filter and define a set of Filters. For information, see "*ChainOfFilters*".

3.4.2. Usage

```
{
    "type": "ConditionalFilter",
    "config": {
        "condition": runtime expression<boolean>,
        "delegate": filter reference
   }
}
```

3.4.3. Properties

`"condition"`: *runtime expression<boolean>, required*

If the expression evaluates to `true`, the request is dispatched to the delegate Filter. Otherwise the delegate Filter is skipped.

See also "Expressions".

`"delegate"`: *filter reference, required*

Filter to treat the request if the condition expression evaluates as `true`.

Provide an inline Filter configuration object, or the name of a Filter object defined in the heap.

See also "Filters".

3.4.4. Example

The following example tests whether a request finishes with `.js` or `.jpg`:

```
{
  "type": "Chain",
  "config": {
    "filters": [{
      "type": "ConditionalFilter",
      "config": {
        "condition": "${not matches ((request.uri.path, '.js$') or (request.uri.path, '.jpg$'))}",
        "delegate": "mySingleSignOnFilter"
      }
    }],
    "handler": "ReverseProxyHandler"
  }
}
```

If the request is to access a .js file or .jpg file, it skips the delegate SingleSignOnFilter filter declared in the heap, and passes straight to the ReverseProxyHandler.

If the request is to access another type of resource, it must pass through the delegate SingleSignOnFilter for authentication with AM before it can pass to the ReverseProxyHandler.

3.4.5. More Information

org.forgerock.http.filter.ConditionalFilter

Chapter 3.5

ConditionEnforcementFilter

3.5.1. Description

Verifies that a specified condition is met. If the condition is met, the request continues to be executed. Otherwise, the request is referred to a failure handler, or Microgateway returns 403 Forbidden and the request is stopped.

3.5.2. Usage

```
{
    "type": "ConditionEnforcementFilter",
    "config": {
        "condition": runtime expression<boolean>,
        "failureHandler": handler reference
    }
}
```

3.5.3. Properties

`"condition"`: *runtime expression<boolean>, required*

If the expression evaluates to `true`, the request continues to be executed.

See also "Expressions".

`"failureHandler"`: *handler reference, optional*

Handler to treat the request if the condition expression evaluates as `false`.

Provide an inline handler configuration object, or the name of a handler object that is defined in the heap.

See also "Handlers".

Default: HTTP 403 Forbidden, the request stops being executed.

3.5.4. Example

The following example tests whether a request contains a session username. If it does, the request continues to be executed. Otherwise, the request is dispatched to the `ConditionFailedHandler` failure handler.

```
{
    "name": "UsernameEnforcementFilter",
    "type": "ConditionEnforcementFilter",
    "config": {
        "condition": "${not empty (session.username)}",
        "failureHandler": "ConditionFailedHandler"
    }
}
```

3.5.5. More Information

org.forgerock.openig.filter.ConditionEnforcementFilter

Chapter 3.6

ChainOfFilters

3.6.1. Description

Dispatches a request to an ordered list of filters. Use this filter to assemble a list of filters into a single filter that you can then use in different places in the configuration.

A ChainOfFilters can be placed in a configuration anywhere that a filter can be placed.

Unlike `Chain`, `ChainOfFilters` does not finish by dispatching the request to a handler. For more information, see "*Chain*".

3.6.2. Usage

```
{
    "name": string,
    "type": "ChainOfFilters",
    "config": {
        "filters": [ Filter reference, ... ]
    }
}
```

3.6.3. Properties

`"filters"`: *array of filter references, required*

An array of names of filter objects defined in the heap, and inline filter configuration objects.

The chain dispatches the request to these filters in the order they appear in the array.

See also "Filters".

3.6.4. Example

```
{
    "name": "MyChainOfFilters",
    "type": "ChainOfFilters",
    "config": {
        "filters": [ "Filter1", "Filter2" ]
   }
}
```

3.6.5. More Information

org.forgerock.openig.filter.ChainFilterHeaplet

Chapter 3.7
CookieFilter

3.7.1. Description

Manages, suppresses, and relays cookies as follows:

- **Manage**, to store cookies from the protected application in the Microgateway session, and include them in later requests.

  For requests with a `Cookie` header, managed cookies are removed so that protected applications don't see them.

  For responses with a `Set-Cookie` header, managed cookies are removed and then added in a `Cookie` header to the next request that goes through that filter.

  Manage is the default action, and a common choice to manage cookies originating from the protected application.

- **Suppress**, to remove cookies from the request and response. Use this option to hide domain cookies, such as `iPlanetDirectoryPro`, that are used by Microgateway but are not usually used by protected applications.

- **Relay**, to transmit cookies freely from the user agent to the remote server, and vice versa.

If a cookie does not appear in one of the three action parameters, then the default action is performed, controlled by setting the `defaultAction` parameter. If unspecified, the default action is to manage all cookies. In the event a cookie appears in more than one configuration parameter, then it will be selected in the order of precedence: managed, suppressed, relayed.

3.7.2. Usage

```
{
     "name": string,
     "type": "CookieFilter",
     "config": {
         "managed": [ string, ... ],
         "suppressed": [ string, ... ],
         "relayed": [ string, ... ],
         "defaultAction": string
     }
}
```

3.7.3. Properties

`"managed"`: *array of strings, optional*

A list of the names of cookies to be managed.

`"suppressed"`: *array of strings, optional*

A list of the names of cookies to be suppressed.

`"relayed"`: *array of strings, optional*

A list of the names of cookies to be relayed.

`"defaultAction"`: *enumeration, optional*

Action to perform for cookies that do not match an action set. Set to `"MANAGE"`, `"RELAY"`, or `"SUPPRESS"`. Default: `"MANAGE"`.

3.7.4. More Information

org.forgerock.openig.filter.CookieFilter

Chapter 3.8

CrossDomainSingleSignOnFilter

3.8.1. Description

When Microgateway and AM are running in the same domain, the SingleSignOnFilter can be used for SSO. When Microgateway and AM are running in different domains, AM cookies are not visible to Microgateway because of the same-origin policy. The CrossDomainSingleSignOnFilter provides a mechanism to push tokens issued by AM to Microgateway running in a different domain.

When this filter processes a request, it injects the CDSSO token, the session user ID, and the full claims set into the "*CdSsoContext*". Should an error occur during authentication, the error details are captured in a "*CdSsoFailureContext*".

Supported with AM 5.5 and later versions.

3.8.1.1. WebSocket Notifications for Sessions

When WebSocket notifications are set up for sessions, Microgateway receives a notification from AM when a user logs out of AM, or when the AM session is modified, closed, or times out. Microgateway then evicts entries that are related to the event from the `sessionCache`.

3.8.2. Usage

```
{
  "name": string,
  "type": "CrossDomainSingleSignOnFilter",
  "config": {
    "amService": AmService reference,
    "authCookie": object,
    "redirectEndpoint": runtime expression<uri string>,
    "failureHandler": Handler reference,
    "verificationSecretId": configuration expression<secret-id>,
    "secretsProvider": SecretsProvider reference
  }
}
```

3.8.3. Properties

`"amService"`: *AmService reference, required*

The AmService heap object to use for the following properties:

- `agent`, the credentials of an Microgateway agent in AM. When the agent is authenticated, the token can be used for tasks such as getting the user's profile, making policy evaluations, and connecting to the AM notification endpoint.
- `url`, the URL of an AM service to use for session token validation and authentication.
- `ssoTokenHeader`, the name of the cookie that contains the session token created by AM.
- `amHandler`, the handler to use when communicating with AM to validate the token in the incoming request.
- `realm`: Realm of the Microgateway agent in AM.
- `version`: The version of the AM server.

See also, "*AmService*".

`"authCookie"`: *object, optional*

The configuration of the cookie used to store the authentication.

```
{
  "cookie": {
    "name": configuration expression<string>,
    "domain": configuration expression<string>,
    "httpOnly": configuration expression<boolean>,
    "path": configuration expression<string>,
    "sameSite": configuration expression<enumeration>,
    "secure": configuration expression<boolean>
  }
}
```

`"name"`: *configuration expression<string>, optional*

Name of the cookie containing the authentication token from AM.

For security, change the default name of cookies.

Default: `ig-token-cookie`

`"domain"`: *configuration expression<string>, optional*

Domain to which the cookie applies.

Set a domain only if the user-agent is able to re-emit cookies on that domain on its next hop. For example, to re-emit a cookie on the domain `example.com`, the user-agent must be able to access that domain on its next hop.

Default: The fully qualified hostname of the user-agent's next hop.

`"httpOnly"`: *configuration expression<boolean>, optional*

Flag to mitigate the risk of client-side scripts accessing protected cookies.

Default: `true`

`"path"`: *configuration expression<string>, optional*

Path protected by this authentication.

Set a path only if the user-agent is able to re-emit cookies on the path. For example, to re-emit a cookie on the path `/home/cdsso`, the user-agent must be able to access that path on its next hop.

Default: The path of the request that got the `Set-Cookie` in its response.

`"sameSite"`: *configuration expression<enumeration>, optional*

Manage the circumstances in which a cookie is sent to the server. Use one of the following options to reduce the risk of cross-site request forgery (CSRF) attacks:

- `STRICT`: Send the cookie only if the request was initiated from the cookie domain.
- `LAX`: Send the cookie only with GET requests in a first-party context, where the URL in the address bar matches the cookie domain.

The value is not case-sensitive.

Default: Null; send the cookie whenever a request is made to the cookie domain.

`"secure"`: *configuration expression<boolean>, optional*

Flag to limit the scope of the cookie to secure channels.

Set this flag only if the user-agent is able to re-emit cookies over HTTPS on its next hop. For example, to re-emit a cookie with the `secure` flag, the user-agent must be connected to its next hop by HTTPS.

Default: `false`

`"redirectEndpoint"`: *runtime expression<uri string>, required*

The URI to the endpoint for CDSSO.

This URI must also be configured in AM exactly as it is configured here. For more information about configuring policy agents in AM, see Implementing Cross-Domain Single Sign-On in the Access Management *Authentication and Single Sign-On Guide*..

To make sure that the redirect is routed back to this filter, where the authentication can be validated, include the endpoint in the route condition in one of the following ways:

- As a sub-path of the condition path.

  For example, use the following route condition with the following endpoint:

  ```
  "condition": "${matches(request.uri.path, '^/home/cdsso')}"
  ```

  ```
  "redirectEndpoint": "/home/cdsso/callback"
  ```

- To match the route condition.

  For example, use the following route condition with the following endpoint:

  ```
  "condition": "${matches(request.uri.path, '^/home/cdsso')}"
  ```

  ```
  "redirectEndpoint": "/home/cdsso"
  ```

  Note

  With this route condition, all POST requests on the condition path are treated as AM CDSSO callbacks. Any POST requests that aren't the result of an AM CDSSO callback will fail.

- As a specific path that is not related to the condition path.

  To make sure that the redirect is routed back to this filter, include the redirectEndpoint as a path in the filter condition.

  For example, use the following route condition with the following endpoint:

  ```
  "condition": "${matches(request.uri.path, '^/home/cdsso/redirect') || matches(request.uri.path, '^/ig/cdssoRedirectUri')}"
  ```

  ```
  "redirectEndpoint": "/ig/cdssoRedirectUri"
  ```

`"failureHandler"`: *Handler reference, optional*

Failure handler to be invoked should an error occur during authentication.

Should an error occur during authentication, a "*CdSsoFailureContext*" is populated with details of the error and any associated `Throwable`. This is available to the failure handler so that it can respond appropriately.

Be aware that the failure handler does not itself play a role in user authentication. It is only invoked if there is a problem that prevents user authentication from taking place.

A number of circumstances may cause the failure handler to be invoked, including:

- The redirect endpoint is invalid.
- The redirect endpoint is invoked without a valid CDSSO token.
- The redirect endpoint is invoked inappropriately.

- An error was reported by AM during authentication.

Should no failure handler be configured then the default failure handler is used.

See also"Handlers".

Default: HTTP 200 Ok. The response entity contains details of the error.

`"verificationSecretId"`: *configuration expression<secret-id>, required to verify the signature of signed tokens*

The secret ID for the secret to verify the signature of signed tokens. If not configured, Microgateway does not verify the signature of signed tokens.

If the signed token contains a `kid`, by priority the secret store uses the `kid` to try to find and resolve a matching secret.

`"secretsProvider"`: *SecretsProvider reference, required to verify the signature of signed JWTs*

The "*SecretsProvider*" to use to resolve queried secrets, such as passwords and cryptographic keys. Provide either the name of a SecretsProvider object defined in the heap, or specify a SecretsProvider object inline.

3.8.4. More Information

org.forgerock.openig.openam.SingleSignOnFilter

"*CdSsoContext*"

"*CdSsoFailureContext*"

"*SsoTokenContext*"

Chapter 3.9

CryptoHeaderFilter

3.9.1. Description

Encrypts or decrypts headers in a request or response.

3.9.2. Usage

```
{
    "name": string,
    "type": "CryptoHeaderFilter",
    "config": {
        "messageType": string,
        "operation": string,
        "key": expression,
        "algorithm": string,
        "charset": string,
        "keyType": string,
        "headers": [ string, ... ]
    }
}
```

3.9.3. Properties

`"messageType"`: *string, required*

Indicates the type of message whose headers to encrypt or decrypt.

Must be one of: `"REQUEST"`, `"RESPONSE"`.

`"operation"`: *string, required*

Indicates whether to encrypt or decrypt.

Must be one of: `"ENCRYPT"`, `"DECRYPT"`.

`"key"`: *expression, required*

Base64 encoded key value.

See also "*Expressions*".

`"algorithm"`: *string, optional*

Algorithm used for encryption and decryption.

Default: `AES/ECB/PKCS5Padding`

`"charset"`: *string, optional*

The name of the charset used to encrypt or decrypt values, as described in Class Charset.

Default: `UTF-8`

`"keyType"`: *string, optional*

Algorithm name for the secret key.

Default: `AES`

`"headers"`: *array of strings, optional*

The names of header fields to encrypt or decrypt.

Default: Do not encrypt or decrypt any headers

3.9.4. Example

```
{
    "name": "DecryptReplayPasswordFilter",
    "type": "CryptoHeaderFilter",
    "config": {
        "messageType": "REQUEST",
        "operation": "DECRYPT",
        "algorithm": "DES/ECB/NoPadding",
        "keyType": "DES",
        "key": "oqdP3DJdE1Q=",
        "headers": [
            "replaypassword"
        ]
    }
}
```

3.9.5. More Information

org.forgerock.openig.filter.CryptoHeaderFilter

Chapter 3.10

DateHeaderFilter

Inserts the server date in an HTTP Date header on the response, if the Date header is not present.

3.10.1. Usage

```
{
  "type": "DateHeaderFilter"
}
```

3.10.2. Properties

There are no configuration properties for this filter.

3.10.3. Example

The following example includes a DateHeaderFilter in a chain:

```
{
  "condition": "...",
  "handler": {
    "type": "Chain",
    "config": {
      "filters": [{
          ...
        },
        {
          "type": "DateHeaderFilter"
        }
      ],
      "handler": {
        "name": "StaticResponseHandler-1",
        ...
      }
    }
  }
}
```

3.10.4. More Information

For information about Date format, see rfc7231 - Date.

This filter is also available to support Financial-Grade API, for information, see Financial-grade API - Part 1: Read-Only API Security Profile

org.forgerock.openig.filter.DateHeaderFilter

Chapter 3.11

EntityExtractFilter

3.11.1. Description

Extracts regular expression patterns from a message entity. The extraction results are stored in a "target" object. For a given matched pattern, as described in "*Patterns*", the value stored in the object is either the result of applying its associated pattern template (if specified) or the match result itself otherwise.

3.11.2. Usage

```
{
    "name": string,
    "type": "EntityExtractFilter",
    "config": {
        "messageType": string,
        "charset": string,
        "target": lvalue-expression,
        "bindings": [
            {
                "key": string,
                "pattern": pattern,
                "template": pattern-template
            }, ...
        ]
    }
}
```

3.11.3. Properties

`"messageType"`: *string, required*

The message type to extract patterns from.

Must be one of: `REQUEST`, `RESPONSE`.

`"charset"`: *string, optional*

Overrides the character set encoding specified in message.

Default: the message encoding is used.

`"target"`: *lvalue-expression, required*

Expression that yields the target object that contains the extraction results.

The bindings determine what type of object is stored in the target location.

The object stored in the target location is a Map<String, String>. You can then access its content with `${target.key}` or `${target['key']}`.

See also "*Expressions*".

`"key"`: *string, required*

Name of element in target object to contain an extraction result.

`"pattern"`: *pattern, required*

The regular expression pattern to find in the entity.

See also "*Patterns*".

`"template"`: *pattern-template, optional*

The template to apply to the pattern and store in the named target element.

Default: store the match result itself.

See also "*Patterns*".

3.11.4. Examples

Extracts a nonce from the response, which is typically a login page, and sets its value in the attributes context to be used by the downstream filter posting the login form. The nonce value would be accessed using the following expression: `${attributes.extract.wpLoginToken}`.

The pattern finds all matches in the HTTP body of the form `wpLogintoken value="abc"`. Setting the template to `$1` assigns the value `abc` to `attributes.extract.wpLoginToken`:

```
{
    "name": "WikiNoncePageExtract",
    "type": "EntityExtractFilter",
    "config": {
        "messageType": "response",
        "target": "${attributes.extract}",
        "bindings": [
            {
                "key": "wpLoginToken",
                "pattern": "wpLoginToken\"\\s.*value=\"(.*)\"",
                "template": "$1"
            }
        ]
    }
}
```

The following example reads the response looking for the AM login page. When found, it sets `isLoginPage = true` to be used in a SwitchFilter to post the login credentials:

```
{
    "name": "FindLoginPage",
    "type": "EntityExtractFilter",
    "config": {
        "messageType": "response",
        "target": "${attributes.extract}",
        "bindings": [
            {
                "key": "isLoginPage",
                "pattern": "OpenAM\s\(Login\)",
                "template": "true"
            }
        ]
    }
}
```

3.11.5. More Information

org.forgerock.openig.filter.EntityExtractFilter

Chapter 3.12

FapiInteractionIdFilter

Tracks the interaction ID of requests, according to the Financial-grade API (FAPI) WG, as follows:

- If a FAPI header is provided in a client request, includes the interaction ID in the `x-fapi-interaction-id` property of the response header.
- If a FAPI header is not provided in the request, includes a new Universally Unique Identifier (UUID) in the `x-fapi-interaction-id` property of the response header.
- Adds the value of `x-fapi-interaction-id` to the log.

3.12.1. Usage

```
{
  "type": "FapiInteractionIdFilter"
}
```

3.12.2. Properties

There are no configuration properties for this filter.

3.12.3. More Information

org.forgerock.openig.filter.finance.FapiInteractionIdFilter

Financial-grade API - Part 1: Read-Only API Security Profile

Chapter 3.13
FileAttributesFilter

3.13.1. Description

Retrieves and exposes a record from a delimiter-separated file. Lookup of the record is performed using a specified `key`, whose `value` is derived from an expression. The resulting record is exposed in an object whose location is specified by the `target` expression. If a matching record cannot be found, then the resulting object is empty.

The retrieval of the record is performed lazily; it does not occur until the first attempt to access a value in the `target`. This defers the overhead of file operations and text processing until a value is first required. This also means that the value expression is not evaluated until the object is first accessed.

3.13.2. Usage

```
{
     "name": string,
     "type": "FileAttributesFilter",
     "config": {
         "file": expression,
         "charset": string,
         "separator": string,
         "header": boolean,
         "fields": [ string, ... ],
         "target": lvalue-expression,
         "key": string,
         "value": runtime expression<string>
     }
}
```

3.13.3. Properties

`"file"`: *expression, required*

The file containing the record to be read.

See also "*Expressions*".

`"charset"`: *string, optional*

The character set in which the file is encoded.

Default: `"UTF-8"`.

`"separator"`: *separator identifier string, optional*

The separator character, which is one of the following:

`COLON`

Unix-style colon-separated values, with backslash as the escape character.

`COMMA`

Comma-separated values, with support for quoted literal strings.

`TAB`

Tab separated values, with support for quoted literal strings.

Default: `COMMA`

`"header"`: *boolean, optional*

The setting to treat or not treat the first row of the file as a header row.

When the first row of the file is treated as a header row, the data in that row is disregarded and cannot be returned by a lookup operation.

Default: `true`.

`"fields"`: *array of strings, optional*

A list of keys in the order they appear in a record.

If `fields` is not set, the keys are assigned automatically by the column numbers of the file.

`"target"`: *lvalue-expression, required*

Expression that yields the target object to contain the record.

The target object is a `Map<String, String>`, where the fields are the keys. For example, if the target is `${attributes.file}` and the record has a `username` field and a `password` field mentioned in the fields list, Then you can access the user name as `${attributes.file.username}` and the password as `${attributes.file.password}`.

See also "*Expressions*".

`"key"`: *string, required*

The key used for the lookup operation.

`"value"`: *runtime expression<string>, required*

The value to be looked-up in the file.

See also "*Expressions*".

3.13.4. More Information

org.forgerock.openig.filter.FileAttributesFilter

Chapter 3.14

HeaderFilter

3.14.1. Description

Removes headers from and adds headers to request and response messages. Headers are added to any existing headers in the message. To replace a header, remove the header and then add it again.

3.14.2. Usage

```
{
  "name": string,
  "type": "HeaderFilter",
  "config": {
    "messageType": enumeration,
    "remove": [ string, ... ],
    "add": {
      name: [ runtime expression<string>, ... ], ...
    }
  }
}
```

3.14.3. Properties

`"messageType"`: *enumeration, required*

Indicates the type of message to filter headers for. Must be one of: `"REQUEST"`, `"RESPONSE"`.

`"remove"`: *array of strings, optional*

The names of header fields to remove from the message.

`"add"`: *object, optional*

Header fields to add to the message. The header name is specified by `name`. The header values are specified by an array of runtime expressions that evaluate to strings.

3.14.4. Examples

3.14.4.1. Replacing Host Header On An Incoming Request

The following example replaces the host header on the incoming request with the value `myhost.com`:

```
{
  "name": "ReplaceHostFilter",
  "type": "HeaderFilter",
  "config": {
    "messageType": "REQUEST",
    "remove": [ "host" ],
    "add": {
      "host": [ "myhost.com" ]
    }
  }
}
```

3.14.4.2. Adding a Header to a Response

The following example adds a `Set-Cookie` header to the response:

```
{
  "name": "SetCookieFilter",
  "type": "HeaderFilter",
  "config": {
    "messageType": "RESPONSE",
    "add": {
      "Set-Cookie": [ "mysession=12345" ]
    }
  }
}
```

3.14.4.3. Adding Headers to a Request

The following example adds the headers `custom1` and `custom2` to the request:

```
{
  "name": "SetCustomHeaders",
  "type": "HeaderFilter",
  "config": {
    "messageType": "REQUEST",
    "add": {
      "custom1": [ "12345", "6789" ],
      "custom2": [ "abcd" ]
    }
  }
}
```

3.14.4.4. Adding a Token Value to a Response

The following example adds the value of session's policy enforcement token to the `pef_sso_token` header in the response:

```
{
  "type": "HeaderFilter",
  "config": {
    "messageType": "RESPONSE",
    "add": {
      "pef_sso_token": ["${session.pef_token}"]
    }
  }
}
```

3.14.5. More Information

org.forgerock.openig.filter.HeaderFilter

Chapter 3.15

HttpBasicAuthenticationClientFilter

3.15.1. Description

Authenticate clients according to HTTP Basic Authentication protocol, using the client's credentials.

Use this filter in a service-to-service context, where services need to access resources protected by HTTP Basic Authentication.

3.15.2. Usage

```
{
  "type": "HttpBasicAuthenticationClientFilter",
  "config": {
    "username": configuration expression<string>,
    "passwordSecretId": configuration expression<secret-id>,
    "secretsProvider": SecretsProvider reference,
    "urlEncodeCredentials": configuration expression<boolean>
  }
}
```

3.15.3. Properties

`"username"`: *configuration expression<string>, required*

The username of the client to authenticate.

`"passwordSecretId"`: *configuration expression<string>, required*

The secret ID required to obtain the client password.

For information about supported formats for `secret-id`, see secret-id.

`"secretsProvider"`: *SecretsProvider reference, required*

The "*SecretsProvider*" to use to obtain the `passwordSecretId`. Provide either the name of a SecretsProvider object defined in the heap, or specify a SecretsProvider object inline.

`"urlEncodeCredentials"`: *configuration expression<boolean>, optional*

Set to `true` to URL-encoded credentials before base64-encoding them.

Default: `false`

3.15.4. More Information

org.forgerock.openig.filter.HttpBasicAuthenticationClientFilter

Chapter 3.16

HttpBasicAuthFilter

3.16.1. Description

Authenticate clients by providing the client credentials as a basic authorization header in the request. The credentials are base64-encoded.

This filter performs authentication through the HTTP Basic authentication scheme, described in RFC 2617.

Use this filter primarily for password replay scenarios, where the password is stored externally in clear text.

If challenged for authentication via a `401 Unauthorized` status code by the server, this filter retries the request with credentials attached. After an HTTP authentication challenge is issued from the remote server, all subsequent requests to that remote server that pass through the filter include the user credentials.

If authentication fails (including the case where no credentials are yielded from expressions), then processing is diverted to the specified authentication failure handler.

3.16.2. Usage

```
{
    "name": string,
    "type": "HttpBasicAuthFilter",
    "config": {
        "username": runtime expression<string>,
        "password": runtime expression<string>,
        "failureHandler": Handler reference,
        "cacheHeader": boolean
    }
}
```

3.16.3. Properties

`"username"`: *runtime expression<string>, required*

The username to supply during authentication.

See also "*Expressions*".

`"password"`: *runtime expression<string>, required*

The password to supply during authentication.

See also "*Expressions*".

`"failureHandler"`: *Handler reference, required*

Dispatch to this Handler if authentication fails.

Provide either the name of a Handler object defined in the heap, or an inline Handler configuration object.

See also "Handlers".

`"cacheHeader"`: *boolean, optional*

Whether or not to cache credentials in the session after the first successful authentication, and then replay those credentials for subsequent authentications in the same session.

With `"cacheHeader": false`, the filter generates the header for each request. This is useful, for example, when users change their passwords during a browser session.

Default: `true`

3.16.4. Example

```
{
    "name": "TomcatAuthenticator",
    "type": "HttpBasicAuthFilter",
    "config": {
        "username": "tomcat",
        "password": "tomcat",
        "failureHandler": "TomcatAuthFailureHandler",
        "cacheHeader": false
    }
}
```

3.16.5. More Information

org.forgerock.openig.filter.HttpBasicAuthFilter

Chapter 3.17

JwtBuilderFilter

3.17.1. Description

This filter collects data at runtime, packs it in a JSON Web Token (JWT), and places the resulting JWT into the "*JwtBuilderContext*". Use this filter with a HeaderFilter for a flexible way to pass identity or other runtime information to the protected application.

Configure JwtBuilderFilter to create an unsigned JWT, a signed JWT, or a signed then encrypted JWT:

- Sign the JWT so that an application can validate the authenticity of the claims/data. The JWT can be signed with a shared secret or private key, and verified with a shared secret or corresponding public key.
- Encrypt the JWT to reduce the risk of a data breach.

To help with development, the sample app includes a `/jwt` endpoint to display the JWT, verify its signature, and decrypt the JWT.

3.17.2. Usage

```
{
  "name": string,
  "type": "JwtBuilderFilter",
  "config": {
    "template": map or runtime expression<map>,
    "secretsProvider": SecretsProvider reference,
    "signature": object
  }
}
```

3.17.3. Properties

`"template"`: *map or runtime expression<map>, required*

A map of information taken from the request or associated contexts in Microgateway.

If this property is a map, the structure must have the format `Map<String, Object>`. For example,

```
"template": {
  "name": "${contexts.userProfile.commonName}",
  "email": "${contexts.userProfile.rawInfo.mail[0]}",
  "address": "${contexts.userProfile.rawInfo.postalAddress[0]}",
  "phone": "${contexts.userProfile.rawInfo.telephoneNumber[0]}"
}
```

If this property is an expression, its evaluation must give an object of type `Map<String, Object>`. For example,

```
"template": "${contexts.attributes}"
```

See also "*Expressions*".

`"secretsProvider"`: *SecretsProvider reference, optional*

The SecretsProvider object to query for JWT signing or encryption keys. For more information, see "*SecretsProvider*".

Default: The route's default secret service. For more information, see "*secrets*".

`"signature"`: *object, optional*

A JWT signature to allow the authenticity of the claims/data to be validated. A signed JWT can be encrypted.

```
{
  "signature": {
    "secretId":  configuration expression<secret-id>,
    "algorithm": configuration expression<string>,
    "encryption": object
  }
}
```

`"secretId"`: *configuration expression<secret-id>, required if `signature` is used*

The secret ID of the key used to sign the JWT.

For information about supported formats for `secret-id`, see secret-id.

`"algorithm"`: *expression<string>, optional*

The algorithm with which to sign the JWT.

The following algorithms are supported but not necessarily tested in Microgateway:

- Algorithms described in *Cryptographic Algorithms for Digital Signatures and MACs* .

  For RSASSA-PSS, you must install Bouncy Castle. For information, see *The Legion of the Bouncy Castle* .

- From Microgateway 6.1, `Ed25519` described in *CFRG Elliptic Curve Diffie-Hellman (ECDH) and Signatures* .

Default: RS256

`"encryption"`: *object, optional*

Encrypt the JWT.

```
{
  "encryption": {
    "secretId": configuration expression<secret-id>,
    "algorithm": configuration expression<string>,
    "method": configuration expression<enumeration>
  }
}
```

`"secretId"`: *configuration expression<secret-id>, optional*

The secret ID of the key used to encrypt the JWT. The value is mapped to key `aliases` in "*KeyStoreSecretStore*".

For information about supported formats for `secret-id`, see secret-id.

`"algorithm"`: *expression<string>, required*

The algorithm used to encrypt the JWT.

For information about available algorithms, see *"alg" (Algorithm) Header Parameter Values for JWE* .

`"method"`: *configuration expression<enumeration>, required*

The method used to encrypt the JWT.

For information about available methods, see *"enc" (Encryption Algorithm) Header Parameter Values for JWE* .

3.17.4. More Information

org.forgerock.openig.filter.JwtBuilderFilter

org.forgerock.openig.filter.JwtBuilderContext

Chapter 3.18

LocationHeaderFilter

3.18.1. Description

For a response that generates a redirect to the proxied application, this filter rewrites the Location header on the response to redirect the user to Microgateway.

3.18.2. Usage

```
{
    "name": string,
    "type": "LocationHeaderFilter",
    "config": {
        "baseURI": runtime expression<uri string>
    }
}
```

An alternative value for type is RedirectFilter.

3.18.3. Properties

`"baseURI"`: *runtime expression<uri string>, optional*

The base URI of the Microgateway instance. This is used to rewrite the Location header on the response.

The result of the expression must be a string that represents a valid URI, but is not a real `java.net.URI` object. For example, it would be incorrect to use `${request.uri}`, which is not a String but a MutableUri.

Default: Redirect to the original URI specified in the request.

See also "*Expressions*".

3.18.4. Example

In the following example, Microgateway listens on `https://microgateway.example.com:443` and the application it protects listens on `http://app.example.com:8081`. The filter rewrites redirects that would

normally take the user to locations under `http://app.example.com:8081` to go instead to locations under `https://microgateway.example.com:443`.

```
{
     "name": "LocationRewriter",
     "type": "LocationHeaderFilter",
     "config": {
         "baseURI": "https://microgateway.example.com:443/"
      }
}
```

3.18.5. More Information

org.forgerock.openig.filter.LocationHeaderFilter

Chapter 3.19

OAuth2ClientFilter

3.19.1. Description

The OAuth2ClientFilter uses OAuth 2.0 delegated authorization to authenticate end users. The filter can act as an OpenID Connect relying party or as an OAuth 2.0 client.

OAuth2ClientFilter performs the following tasks:

- Allows the user to select an authorization server from one or more static client registrations, or by discovery and dynamic registration.

  In static client registration, authorization servers are provided by "*Issuer*", and registrations are provided by "*ClientRegistration*".

- Redirects the user through the authentication and authorization steps of an OAuth 2.0 authorization code grant, which results in the authorization server returning an access_token to the filter.

- When an authorization grant succeeds, injects the access_token data into a configurable target in the context so that subsequent filters and handlers can access the access_token. Subsequent requests can use the access_token without authenticating again.

- When an authorization grant fails, invokes a failureHandler.

3.19.2. Service URIs

Service URIs are constructed from the `clientEndpoint`. Task that the OAuth2ClientFilter performs is determined by the service URI, as follows:

clientEndpoint*/login/?discovery=*user-input*&goto=*url

Discover and register dynamically with the end user's OpenID Provider or with the client registration endpoint as described in RFC 7591, using the value of *user-input*.

After successful registration, redirect the end user to the provider for authentication and authorization consent. Then redirect the user-agent back to the callback client endpoint.

Supported with OpenAM 13 and later versions, and AM 5 and later versions.

clientEndpoint`/login?registration=`*clientId*`&issuer=`*issuerName*`&goto=`*url*

Redirect the end user for authorization with the specified *registration*, defined by the ClientRegistration's `clientId` and `issuerName`. For information, see "*ClientRegistration*".

The provider corresponding to the registration then authenticates the end user and obtains authorization consent before redirecting the user-agent back to the callback client endpoint.

If the whole process is successful, the filter saves the authorization state in the session and redirects the user-agent to the specified URL.

clientEndpoint`/logout?goto=`*url*

Remove the authorization state for the end user, and redirect to the specified URL.

If no goto URL is specified in the request, use the `defaultLogoutGoto`.

clientEndpoint`/callback`

Handle the callback from the OAuth 2.0 authorization server, that occurs as part of the authorization process.

If the callback is handled successfully, the filter saves the authorization state in the context at the specified target location and redirects to the URL during login.

Other request URIs

Restore the authorization state in the specified target location, and call the next filter or handler in the chain.

3.19.3. Usage

```
{
  "name": string,
  "type": "OAuth2ClientFilter",
  "config": {
    "clientEndpoint": runtime expression<uri sting>,
    "failureHandler": Handler reference,
    "loginHandler": Handler reference,
    "registrations": [ ClientRegistration reference(s) ],
    "metadata": dynamic registration client metadata object,
    "cacheExpiration": configuration expression<duration>,
    "executor": executor service reference,
    "target": configuration expression<lvalue-expression>,
    "defaultLoginGoto": runtime expression<uri string>,
    "defaultLogoutGoto": runtime expression<uri string>,
    "requireHttps": configuration expression<boolean>,
    "requireLogin": configuration expression<boolean>,
    "issuerRepository": Issuer repository reference,
    "discoveryHandler": Handler reference,
    "discoverySecretId": configuration expression<secret-id>,
    "tokenEndpointAuthMethod": configuration expression<enumeration>,
    "tokenEndpointAuthSigningAlg": configuration expression<string>,
    "secretsProvider": SecretsProvider reference
  }
}
```

3.19.4. Properties

`"clientEndpoint"`: *runtime expression<uri string>, required*

The URI to the client endpoint.

So that routes can accept redirects from the authorization server to the callback endpoint, the `clientEndpoint` must be the same as the route condition or a sub path of the route condition. For example:

- The same as the route condition:

```
"condition": "${matches(request.uri.path, '^/discovery')}"
```

```
"clientEndpoint": "/discovery"
```

- As a sub path of the route condition:

```
"condition": "${matches(request.uri.path, '^/home/id_token')}"
```

```
"clientEndpoint": "/home/id_token/sub-path"
```

The service URIs are constructed from the `clientEndpoint`. For example, when `clientEndpoint` is `openid`, the service URIs are `/openid/login`, `/openid/logout`, and `/openid/callback`. These endpoints are implicitly reserved, and attempts to access them directly can cause undefined errors.

The result of the expression must be a string that represents a valid URI, but is not a real `java.net.URI` object. For example, it would be incorrect to use `${request.uri}`, which is not a String but a MutableUri.

See also "*Expressions*".

`"failureHandler"`: *handler reference, required*

Provide an inline handler configuration object, or the name of a handler object that is defined in the heap.

When the OAuth 2.0 Resource Server denies access to a resource, the failure handler can be invoked only if the error response contains a WWW-Authenticate header (meaning that there was a problem with the OAuth 2.0 exchange). All other responses are forwarded to the user-agent without invoking the failure handler.

If the value of the WWW-Authenticate header is `invalid_token`, the OAuth2ClientFilter tries to refresh the access_token:

- If the token is refreshed, the OAuth2ClientFilter tries again to access the protected resource.
- If the token is not refreshed, or if the second attempt to access the protected resource fails, the OAuth2ClientFilter invokes the failure handler.

When the failure handler is invoked, the target in the context can be populated with information such as the exception, client registration, and error. The failure object in the target is a simple map, similar to the following example:

```
{
    "client_registration": "ClientRegistration name string",
    "error": {
        "realm": "optional string",
        "scope": [ "optional scope string (required by the client)", ... ],
        "error": "optional string",
        "error_description": "optional string",
        "error_uri": "optional string"
    },
    "access_token": "string",
    "id_token": "string",
    "token_type": "Bearer",
    "expires_in": "number",
    "scope": [ "optional scope string", ... ],
    "client_endpoint": "URL string",
    "exception": exception
}
```

In the failure object, the following fields are not always present. Their presence depends on when the failure occurs:

- "access_token"
- "id_token"

- "token_type"
- "expires_in"
- "scope"
- "client_endpoint"

See also "Handlers".

`"loginHandler"`: *Handler reference, required if there are zero or multiple client registrations, optional if there is one client registration*

Use this Handler when the user must choose an authorization server. When `registrations` contains only one client registration, this Handler is optional but is displayed if specified.

Provide either the name of a Handler object defined in the heap, or an inline Handler configuration object.

See also "Handlers".

`"registrations"`: *Array of ClientRegistration references or inline ClientRegistration declarations, optional*

List of client registrations that authenticate Microgateway to the authorization server. The list must contain all client registrations that are to be used by the client filter.

The value represents a static client registration with an authorization server, as described in "*ClientRegistration*".

`"metadata"`: *client metadata object, required for dynamic client registration and ignored otherwise*

This object holds client metadata as described in *OpenID Connect Dynamic Client Registration 1.0* , and optionally a list of scopes. See that document for additional details and a full list of fields.

This object can also hold client metadata as described in RFC 7591, *OAuth 2.0 Dynamic Client Registration Protocol* . See that RFC for additional details.

The following partial list of metadata fields is not exhaustive, but includes metadata that is useful with AM as OpenID Provider:

`"redirect_uris"`: *array of URI strings, required*

The array of redirection URIs to use when dynamically registering this client.

One of the registered values **must** match the `clientEndpoint`.

`"client_name"`: *string, optional*

Name of the client to present to the end user.

`"scope"`: *space separated string, optional*

Space separated string of scopes to request of the OpenID Provider, for example:

```
"scope": "openid profile"
```

This property is available for dynamic client registration with AM 5.5 and later versions, or with authorization servers that support RFC 7591, *OAuth 2.0 Dynamic Client Registration Protocol*

Use both `scope` and `scopes` to dynamically register with a wider range of identity providers.

`"scopes"`: *array of strings, optional*

Array of scope strings to request of the OpenID Provider, for example:

```
"scopes": [
  "openid",
  "profile",
  "email"
]
```

This property is available for dynamic client registration with AM 5.5 and earlier versions only.

Use both `scope` and `scopes` to dynamically register with a wider range of identity providers.

`"cacheExpiration"`: *configuration expression<duration>, optional*

Duration for which to cache user-info resources.

Microgateway lazily fetches user info from the OpenID provider. In other words, Microgateway only fetches the information when a downstream Filter or Handler uses the user info. Caching allows Microgateway to avoid repeated calls to OpenID providers when reusing the information over a short period.

Default: 10 minutes

Set this to disabled or zero to disable caching. When caching is disabled, user info is still lazily fetched.

For information about supported formats for `duration`, see duration.

`"executor"`: *executor service reference, optional*

An executor service to schedule the execution of tasks, such as the eviction of entries in the OpenID Connect user information cache.

Default: `ScheduledExecutorService`

See also "*ScheduledExecutorService*".

`"target"`: *configuration expression<lvalue-expression>, optional*

An expression that yields the target object. Downstream filters and handlers can use data in the target to enrich the existing request or create a new request.

When the `target` is `openid`, the following information can be provided in `${attributes.openid}`:

- `access_token`: The value of the OAuth 2.0 access_token
- `client_endpoint`: The URL to the client endpoint
- `client_registration`: The client ID of the OAuth 2.0 client that enables Microgateway to communicate as an OAuth 2.0 relying party with an authorization server
- `expires_in`: Number of milliseconds until the token expires
- `id_token_claims`: The claims used in the token
- `scope`: The scopes associated with the token
- `token_type`: The type or authentication token
- `user_info`: The profile attributes of an authenticated user

Data is provided to the target as follows:

- If the authorization process completes successfully, the OAuth2ClientFilter injects the authorization state data into the target. In the following example, a downstream StaticRequestFilter retrieves the username and password from the target to log the user in to the sample application.

```
{
  "type": "StaticRequestFilter",
  "config": {
    "method": "POST",
    "uri": "http://app.example.com:8081/login",
    "form": {
      "username": [
        "${attributes.openid.user_info.sub}"
      ],
       "password": [
        "${attributes.openid.user_info.family_name}"
      ]
    }
  }
}
```

- If the failure handler is invoked, the target can be populated with information such as the exception, client registration, and error, as described in "failureHandler" in this reference page.

Default: `${attributes.openid}`

See also "*Expressions*".

`"defaultLoginGoto"`: *runtime expression<uri string>, optional*

After successful authentication and authorization, if the user accesses the `clientEndpoint/login` endpoint without providing a landing page URL in the `goto` parameter, the request is redirected to this URI.

The result of the expression must be a string that represents a valid URI, but is not a real `java.net.URI` object. For example, it would be incorrect to use `${request.uri}`, which is not a String but a MutableUri.

Default: return an empty page.

See also "*Expressions*".

`"defaultLogoutGoto"`: *runtime expression<uri string>, optional*

If the user accesses the `clientEndpoint/logout` endpoint without providing a goto URL, the request is redirected to this URI.

The result of the expression must be a string that represents a valid URI, but is not a real `java.net.URI` object. For example, it would be incorrect to use `${request.uri}`, which is not a String but a MutableUri.

Default: return an empty page.

See also "*Expressions*".

`"requireHttps"`: *configuration expression<boolean>, optional*

Whether to require that original target URI of the request (`originalUri` in "*UriRouterContext*") uses the HTTPS scheme.

If the request received by the web container is not using HTTPS, the request is rejected.

Default: true.

`"requireLogin"`: *configuration expression<boolean>, optional*

Whether to require authentication for all incoming requests.

Default: true.

`"issuerRepository"`: *Issuer repository reference, optional*

A repository of OAuth 2.0 issuers, built from discovered issuers and the Microgateway configuration.

Provide the name of an IssuerRepository object defined in the heap.

Default: Look up an issuer repository named `IssuerRepository` in the heap. If none is explicitly defined, then a default one named `IssuerRepository` is created in the current route.

See also "*IssuerRepository*".

`"discoveryHandler"`: *Handler reference, optional*

Use this property for discovery and dynamic registration of OpenID Connect clients.

Provide either the name of a Handler object defined in the heap, or an inline Handler configuration object. Usually set this to the name of a ClientHandler configured in the heap, or a chain that ends in a ClientHandler.

Default: The default ClientHandler.

See also "Handlers", "*ClientHandler*".

`"discoverySecretId"`: *configuration expression<secret-id>, required for discovery and dynamic registration*

Use this property for discovery and dynamic registration of OAuth 2.0 clients.

Specifies the secret ID of the secret that is used to sign a JWT before the JWT is sent to the authorization server.

If `discoverySecretId` is used, then the `tokenEndpointAuthMethod` is always `private_key_jwt`.

For information about supported formats for `secret-id`, see secret-id.

`"tokenEndpointAuthMethod"`: *configuration expression<enumeration>, optional*

Use this property for discovery and dynamic registration of OAuth 2.0 clients.

The authentication method with which a client authenticates to the authorization server or OpenID provider at the token endpoint. For information about client authentication methods, see OpenID Client Authentication. The following client authentication methods are allowed:

- `client_secret_basic`: Clients that have received a `client_secret` value from the authorization server authenticate with the authorization server by using the HTTP Basic authentication scheme, as in the following example:

```
POST /oauth2/token HTTP/1.1
Host: as.example.com
Authorization: Basic ....
Content-Type: application/x-www-form-urlencoded

grant_type=authorization_code&
code=...
```

- `client_secret_post`: Clients that have received a `client_secret` value from the authorization server authenticate with the authorization server by including the client credentials in the request body, as in the following example:

```
POST /oauth2/token HTTP/1.1
Host: as.example.com
Content-Type: application/x-www-form-urlencoded

grant_type=authorization_code&
client_id=...&
client_secret=...&
code=...
```

- `private_key_jwt`: Clients send a signed JSON Web Token (JWT) to the authorization server. Microgateway builds and signs the JWT, and prepares the request as in the following example:

  ```
  POST /token HTTP/1.1
  Host: as.example.com
  Content-Type: application/x-www-form-urlencoded

  grant_type=authorization_code&
  code=...&
  client_id=<clientregistration_id>&
  client_assertion_type=urn%3Aietf%3Aparams%3Aoauth%3Aclient-assertion-type%3Ajwt-bearer&
  client_assertion=PHNhbWxwOl ... ZT
  ```

  If the authorization server doesn't support `private_key_jwt`, a dynamic registration falls back on the method returned by the authorization server, for example, `client_secret_basic` or `client_secret_post`.

  If `tokenEndpointAuthSigningAlg` is not configured, the `RS256` signing algorithm is used for `private_key_jwt`.

Consider these points for identity providers:

- Some providers accept more than one authentication method.
- If a provider strictly enforces how the client must authenticate, align the authentication method with the provider.
- If a provider doesn't support the authentication method, the provider sends an HTTP 400 Bad Request response with an `invalid_client` error message, according to RFC 6749 *The OAuth 2.0 Authorization Framework*, section 5.2 .
- If the authentication method is invalid, the provider sends an `IllegalArgumentException`.

Default:

- If `discoverySecretId` is used, then the `tokenEndpointAuthMethod` is always `private_key_jwt`.
- Otherwise, `client_secret_basic`

`"tokenEndpointAuthSigningAlg"`: *configuration expression<string>, optional*

The JSON Web Algorithm (JWA) used to sign the JWT that is used to authenticate the client at the token endpoint. The property is used when `private_key_jwt` is used for authentication.

If the authorization server sends a notification to use a different algorithm to sign the JWT, that algorithm is used.

Default:

- If `discoverySecretId` is used, then the `tokenEndpointAuthSigningAlg` is `RS256`.
- Otherwise, not used.

`"secretsProvider"`: *SecretsProvider reference, required*

The "*SecretsProvider*" to use to resolve queried secrets, such as passwords and cryptographic keys. Provide either the name of a SecretsProvider object defined in the heap, or specify a SecretsProvider object inline.

3.19.5. More Information

org.forgerock.openig.filter.oauth2.client.OAuth2ClientFilter

"*Issuer*", "*ClientRegistration*"

The OAuth 2.0 Authorization Framework

OAuth 2.0 Bearer Token Usage

OpenID Connect site, in particular the list of standard OpenID Connect 1.0 scope values

Chapter 3.20

OAuth2ResourceServerFilter

3.20.1. Description

This filter validates a request containing an OAuth 2.0 access_token. The filter expects an OAuth 2.0 token from the HTTP Authorization header of the request, such as the following example header, where the OAuth 2.0 access_token is `1fc...ec9`:

```
Authorization: Bearer 1fc...ec9
```

The filter performs the following tasks:

- Extracts the access_token from the request header.
- Uses the configured access_token resolver to resolve the access_token against an authorization server, and validate the token claims.
- Checks that the token has the scopes required by the filter configuration.
- Injects the access_token info into the "*OAuth2Context*".

The following errors can occur during access_token validation:

Errors During Token Validation

Error	Response from the filter to the user-agent
Combination of the filter configuration and access_token result in an invalid request to the authorization server.	`HTTP 400 Bad Request`
There is no access_token in the request header.	`HTTP 401 Unauthorized WWW-Authenticate: Bearer realm="Microgateway"`
The access_token isn't valid, for example, because it has expired.	`HTTP 401 Unauthorized`
The access_token doesn't have all of the scopes required in the OAuth2ResourceServerFilter configuration.	`HTTP 403 Forbidden`

3.20.2. Usage

```
{
  "name": string,
  "type": "OAuth2ResourceServerFilter",
  "config": {
    "accessTokenResolver": AccessTokenResolver reference,
    "cache": object,
    "executor": executor,
    "requireHttps": configuration expression<boolean>,
    "realm": string,
    "scopes": [ runtime expression<string>, ... ] or object
  }
}
```

An alternative value for type is OAuth2RSFilter.

3.20.3. Properties

`"accessTokenResolver"`: *AccessTokenResolver reference, required*

Resolves an access_token against an authorization server. Configure one of the following access_token resolvers:

- "*TokenIntrospectionAccessTokenResolver*"
- "*StatelessAccessTokenResolver*"
- "*OpenAmAccessTokenResolver*"
- "*ConfirmationKeyVerifierAccessTokenResolver*"
- "*ScriptableAccessTokenResolver*"

`"cache"`: *object, optional*

Configuration of caching for OAuth 2.0 access_tokens. By default, access_tokens are not cached.

For an alternative way of caching of OAuth 2.0 access_tokens, configure "*CacheAccessTokenResolver*".

When an access_token is cached, Microgateway can reuse the token information without repeatedly asking the authorization server to verify the access_token. When caching is disabled, Microgateway must ask the authorization server to verify the access_token for each request.

The following example caches access_tokens for these times:

- One hour, if the access_token doesn't provide a valid expiry value.
- The duration specified by the token expiry value, when the token expiry value is shorter than one day.

- One day, when the token expiry value is longer than one day.

```
"cache": {
  "enabled": true,
  "defaultTimeout": "1 hour",
  "maxTimeout": "1 day"
}
```

`enabled`: *boolean, optional*

Enable or disable caching.

Default: `false`

`defaultTimeout`: *configuration expression<duration string>, optional*

The duration for which to cache an OAuth 2.0 access_token if it doesn't provide a valid expiry value.

If an access_token provides an expiry value that falls *before* the current time plus the `maxTimeout`, Microgateway uses the token expiry value.

Default: `1 minute`

`maxTimeout`: *configuration expression<duration string>, optional*

The maximum duration for which to cache OAuth 2.0 access_tokens.

If an access_token provides an expiry value that falls *after* the current time plus the `maxTimeout`, Microgateway uses the `maxTimeout`.

The duration cannot be `zero` or `unlimited`.

For information about supported formats for `duration`, see duration.

`"executor"`: *executor, optional*

An executor service to schedule the execution of tasks, such as the eviction of entries in the access_token cache.

Default: `ScheduledExecutorService`

See also "*ScheduledExecutorService*".

`"requireHttps"`: *configuration expression<boolean>, optional*

Whether to require that original target URI of the request (`originalUri` in "*UriRouterContext*") uses the HTTPS scheme.

If the request received by the web container is not using HTTPS, the request is rejected.

Default: true.

`"realm"`: *string, optional*

HTTP authentication realm to include in the WWW-Authenticate response header field when returning an HTTP 401 Unauthorized status to a user-agent that need to authenticate.

Default: Microgateway

`"scopes"`: *array of runtime expression<string> or ScriptableResourceAccess object, required*

A list of one of more scopes that the OAuth 2.0 access_token must have.

array of runtime expression<string>, required if ScriptableResourceAccess is not used

A string, array of strings, runtime expression<string>, or array of runtime expression<string> to represent one or more scopes; mutually exclusive with `ScriptableResourceAccess`.

ScriptableResourceAccess, required if "array of runtime expression<string>" is not used

A script that produces a list of one or more required scopes; mutually exclusive with "array of runtime expression<string>".

The script evaluates each request dynamically and returns the scopes that request needs to access the protected resource. The script must return a Promise<Set, ResponseException> or a Set<String>.

For information about the properties of ScriptableResourceAccess, see "*Scripts*". For an example of how to use this property, see the examples section on this page.

```
{
  "name": ScriptableResourceAccess-1,
  "type": "ScriptableResourceAccess",
  "config": {
    "type": string,
    "file": expression,                     // Use either "file"
    "source": string or array of strings,   // or "source", but not both.
    "args": object,
    "clientHandler": Handler reference
  }
}
```

3.20.4. More information

org.forgerock.openig.filter.oauth2.OAuth2ResourceServerFilterHeaplet

org.forgerock.http.oauth2.OAuth2Context

org.forgerock.http.oauth2.AccessTokenInfo

"*OAuth2Context*"

"*ConfirmationKeyVerifierAccessTokenResolver*"

"*OpenAmAccessTokenResolver*"

"*TokenIntrospectionAccessTokenResolver*"

"*StatelessAccessTokenResolver*"

"*ScriptableAccessTokenResolver*"

The OAuth 2.0 Authorization Framework

OAuth 2.0 Bearer Token Usage

Chapter 3.21

PasswordReplayFilter

3.21.1. Description

For requests directed to a login page, this filter extracts credentials, and replays them.

Requests that are not directed to the login page are passed along to the next filter or handler in the chain.

The PasswordReplayFilter does not retry failed authentication attempts.

3.21.2. Usage

```
{
  "name": string,
  "type": "PasswordReplayFilter",
  "config": {
    "request": request configuration object,
    "loginPage": runtime expression<boolean>,
    "loginPageContentMarker": pattern,
    "credentials": Filter reference,
    "headerDecryption": crypto configuration object,
    "loginPageExtractions": [ extract configuration object, ... ]
  }
}
```

3.21.3. Properties

`"request"`: *request configuration object, required*

The request that replays the credentials. The JSON object of `request` is the `config` content of a "*StaticRequestFilter*".

`"method"`: *string, required*

The HTTP method to be performed on the resource such as `GET` or `POST`.

`"uri"`: *uri string, required*

The fully qualified URI of the resource to access, such as `http://www.example.com/login`.

`"entity"`: *expression, optional*

The entity body to include in the request.

When the `method` is set to `POST`, this setting is mutually exclusive with `form`.

See also "*Expressions*".

`"form"`: *object, optional*

A form to include in the request.

The `param` specifies the form parameter name. Its value is an array of expressions to evaluate as form field values.

When the `method` is set to `POST`, this setting is mutually exclusive with `entity`.

`"headers"`: *object, optional*

Header fields to set in the request.

The `name` specifies the header name. Its value is an array of expressions to evaluate as header values.

`"version"`: *string, optional*

The HTTP protocol version.

Default: `"HTTP/1.1"`.

`"loginPage"`: *runtime expression<boolean>, required unless `loginPageContentMarker` is defined*

When the expression evaluates to `true`, direct the request to a login page, extract credentials, and replay them.

When `false`, pass the request unchanged to the next filter or handler in the chain.

The following example expression resolves to `true` when the request is an HTTP GET, and the request URI path is `/login`:

```
${matches(request.uri.path, '/login') and (request.method == 'GET')}
```

`"loginPageContentMarker"`: *pattern, required unless `loginPage` is defined*

A pattern that matches when a response entity is that of a login page.

See also "*Patterns*".

`"credentials"`: *Filter reference, optional*

Filter that injects credentials, making them available for replay. Consider using a `FileAttributesFilter` or a `SqlAttributesFilter`.

When this is not specified, credentials must be made available to the request by other means.

See also "Filters".

`"headerDecryption"`: *crypto configuration object, optional*

Object to decrypt request headers that contain credentials to replay.

The crypto configuration object has the following fields:

`"key"`: *expression, required*

Base64 encoded key value.

See also "*Expressions*".

`"algorithm"`: *string, optional*

Algorithm used for decryption.

Use the same algorithm that is used to send the encrypted credentials.

Default: `AES/ECB/PKCS5Padding`

`"keyType"`: *string, optional*

Algorithm name for the secret key.

Default: `AES`

`"headers"`: *array of strings, optional*

The names of header fields to decrypt.

Default: Do not decrypt any headers.

`"loginPageExtractions"`: *extract configuration array, optional*

Object to extract values from the login page entity.

The extract configuration array is a series of configuration objects. To extract multiple values, use multiple extract configuration objects. Each object has the following fields:

`"name"`: *string, required*

Name of the field where the extracted value is put.

The names are mapped into `attributes.extracted`.

For example, if the name is `nonce`, the value can be obtained with the expression `${attributes.extracted.nonce}`.

The name `isLoginPage` is reserved to hold a boolean that indicates whether the response entity is a login page.

`"pattern"`: *pattern, required*

The regular expression pattern to find in the entity.

The pattern must contain one capturing group. (If it contains more than one, only the value matching the first group is placed into `attributes.extracted`.)

For example, suppose the login page entity contains a nonce required to authenticate, and the nonce in the page looks like `nonce='n-0S6_WzA2Mj'`. To extract `n-0S6_WzA2Mj`, set `"pattern": " nonce='(.*)'"`.

See also "*Patterns*".

3.21.4. Examples

The following example authenticates requests using static credentials when the request URI path is `/login`. This PasswordReplayFilter example does not include any mechanism for remembering when authentication has already been successful, it simply replays the authentication every time that the request URI path is `/login`:

```
{
  "handler": {
    "type": "Chain",
    "config": {
      "filters": [{
        "type": "PasswordReplayFilter",
        "config": {
          "loginPage": "${request.uri.path == '/login'}",
          "request": {
            "method": "POST",
            "uri": "https://www.example.com:8444/login",
            "form": {
              "username": [
                "MY_USERNAME"
              ],
              "password": [
                "MY_PASSWORD"
              ]
            }
          }
        }
      }],
      "handler": "ReverseProxyHandler"
    }
  }
}
```

3.21.5. More Information

org.forgerock.openig.filter.PasswordReplayFilterHeaplet

Chapter 3.22
PolicyEnforcementFilter

3.22.1. Description

This filter requests policy decisions from AM, which allows or denies the request based on the request context, the request URI, and the AM policies.

Supported with AM 5 and later versions.

3.22.1.1. Processing Requests After a Policy Decision

After a policy decision, Microgateway continues to process requests as follows:

- If the request is allowed, processing continues.
- If the request is denied with advices, Microgateway checks whether it can respond to the advices. If Microgateway can respond, it sends a redirect and information about how to meet the conditions in the advices.

  By default, the request is redirected to AM. If the SingleSignOnFilter property `loginEndpoint` is configured, the request is redirected to that endpoint.
- If the request is denied without advice, or if Microgateway cannot respond to the advice, Microgateway forwards the request to a `failureHandler` declared in the `PolicyEnforcementFilter`. If there is no `failureHandler`, Microgateway returns a 403 Forbidden.
- If an error occurs during the process, Microgateway returns 500 Internal Server Error.

3.22.1.2. Using Advices From Policy Decisions

Attributes and advices returned by the policy decision are stored in the `policyDecision` context. For information, see "*PolicyDecisionContext*".

Microgateway responds to the following advice types from AM:

- `AuthLevel`: The minimum authentication level at which a user-agent must authenticate to access a resource.
- `AuthenticateToService`: The name of an authorization chain or service to which a user-agent must authenticate to access a resource.

- `AuthenticateToRealm`: The name of a realm to which a user-agent must authenticate to access a resource.
- `AuthScheme`: The name of an authentication module to which a user-agent must authenticate to access a resource, the policy set name, and the authentication timeout.
- `Transaction`: The additional actions that a user-agent must perform before having a one-time access to the protected resource. Supported with AM 5.5 and later versions.

3.22.1.3. Notes on Configuring Policies in AM

In the AM policy, remember to configure the `Resources` parameter with the URI of the protected application.

The request URI from Microgateway must match the `Resources` parameter defined in the AM policy. If the URI of the incoming request is changed before it enters the policy filter (for example, by rebasing or scripting), remember to change the `Resources` parameter in AM policy accordingly.

3.22.1.4. WebSocket Notifications for Policy Changes

When WebSocket notifications are set up for changes to policies, Microgateway receives a notification from AM when a policy decision is created, deleted, or updated.

3.22.2. Usage

```
{
  "name": string,
  "type": "PolicyEnforcementFilter",
  "config": {
    "amService": AmService reference,
    "pepRealm": string,
    "ssoTokenSubject": runtime expression<string>,
    "jwtSubject": runtime expression<string>,
    "claimsSubject": map or runtime expression<map>,
    "cache": object,
    "application": runtime expression<string>,
    "environment": map or runtime expression<map>,
    "failureHandler": handler reference,
    "resourceUriProvider": ResourceUriProvider reference
  }
}
```

3.22.3. Properties

`"amService"`: *AmService reference, required*

The AmService object to use for the following properties:

- `agent`, the credentials of an Microgateway agent in AM. When the agent is authenticated, the token can be used for tasks such as getting the user's profile, making policy evaluations, and connecting to the AM notification endpoint.
- `realm`: Realm of the Microgateway agent in AM.
- `url`, the URL of an AM service to use for session token validation and authentication.
- `ssoTokenHeader`, the name of the HTTP header that provides the session token created by AM
- `amHandler`, the handler to use when requesting policy decisions from AM.
- `version`: The version of the AM server.

See also, "*AmService*".

`"pepRealm"`: *string, optional*

The AM realm where the policy set is located.

Default: The realm declared for `amService`.

`"ssoTokenSubject"`: *runtime expression<string>, required if neither of the following properties are present:* *`"jwtSubject"`*, *`"claimsSubject"`*

The AM SSO or CDSSO token ID string for the subject making the request to the protected resource.

`ssoTokenSubject` can take the following values:

- `${contexts.ssoToken.value}`, when the SingleSignOnFilter is used for authentication
- `${contexts.ssoToken.value}` or `${contexts.cdsso.token}`, when the CrossDomainSingleSignOnFilter is used for authentication

When no SSO is involved (API protection), then `ssoTokenSubject` usually points to a header value (`${request.headers.iPlanetDirectoryPro[0]}`).

See also "*Expressions*".

`"jwtSubject"`: *runtime expression<string>, required if neither of the following properties are present:* *`"ssoTokenSubject"`*, *`"claimsSubject"`*

The JWT string for the subject making the request to the protected resource.

To use the raw id_token (base64, not decoded) returned by the OpenID Connect Provider during authentication, place an `OAuth2ClientFilter` filter before the PEP filter, and then use `${attributes.openid.id_token}` as the expression value.

See also "*OAuth2ClientFilter*" and "*Expressions*".

`"claimsSubject"`: *map or runtime expression<map>, required if neither of the following properties are present: `"jwtSubject"`, `"ssoTokenSubject"`*

A representation of JWT claims for the subject.

The claim `"sub"` must be specified. Other claims are optional.

When this property is an expression, its evaluation must give an object of type `Map<String, Object>`.

```
"claimsSubject": "${attributes.openid.id_token_claims}"
```

When this property is a map, the structure must have the format `Map<String, Object>`. The value is evaluated as an expression.

```
"claimsSubject": {
  "sub": "${attributes.subject_identifier}",
  "iss": "openam.example.com"
}
```

`"application"`: *runtime expression<string>, optional*

The ID of the AM policy set to use when requesting policy decisions.

Default: `iPlanetAMWebAgentService`, provided by AM's default policy set

`cache`: *object, optional*

Enable and configure caching of policy decisions from AM, based on *Caffeine*. For more information, see the GitHub entry, *Caffeine* .

```
{
  "cache": {
    "enabled": configuration expression<boolean>,
    "defaultTimeout": configuration expression<duration>,
    "executor": executor service reference,
    "maximumSize": configuration expression<number>,
    "maximumTimeToCache": configuration expression<duration>,
    "onNotificationDisconnection": configuration expression<enumeration>,
  }
}
```

Default: Policy decisions are not cached.

Note

Policy decisions that contain advices are never cached.

The following code example caches AM policy decisions without advices for these times:

- One hour, when the policy decision doesn't provide a `ttl` value.

- The duration specified by the `ttl`, when `ttl` is shorter than one day.
- One day, when `ttl` is longer than one day.

```
"cache": {
  "enabled": true,
  "defaultTimeout": "1 hour",
  "maximumTimeToCache": "1 day"
}
```

For information about supported formats for `duration`, see duration.

`enabled`: *configuration expression<boolean>, optional*

Enable or disable caching of policy decisions.

When a policy decision is cached, Microgateway can reuse the policy decision without repeatedly asking AM for a new policy decision. When caching is disabled, Microgateway must ask AM to make a decision for each request.

Default: `false`

`defaultTimeout`: *configuration expression<duration>, optional*

The default duration for which to cache AM policy decisions.

If an AM policy decision provides a valid `ttl` value to specify the time until which the policy decision remains valid, Microgateway uses that value or the `maxTimeout`.

Default: `1 minute`

For information about supported formats for `duration`, see duration.

`executor`: *executor service reference, optional*

An executor service to schedule the execution of tasks, such as the eviction of entries in the cache.

Default: `ForkJoinPool.commonPool()`

`"maximumSize"`: *configuration expression<number>, optional*

The maximum number of entries the cache can contain.

Default: Unlimited/unbound.

`maximumTimeToCache`: *configuration expression<duration>, optional*

The maximum duration for which to cache AM policy decisions.

If the `ttl` value provided by the AM policy decision is after the current time plus the `maximumTimeToCache`, Microgateway uses the `maximumTimeToCache`.

The duration cannot be `zero` or `unlimited`.

For information about supported formats for `duration`, see duration.

`onNotificationDisconnection`: *configuration expression<enumeration>, optional*

The strategy to manage the cache when the WebSocket notification service is disconnected, and Microgateway receives no notifications for AM events, such as policy changes. If the cache is not cleared it can become outdated, and Microgateway can enforce outdated policy decisions.

Cached entries that expire naturally while the notification service is disconnected are removed from the cache.

Use one of the following values:

- `NEVER_CLEAR`
  - When the notification service is disconnected:
    - Continue to use the existing cache.
    - Deny access for requests that are not cached, but do not update the cache with these requests.
  - When the notification service is reconnected:
    - Continue to use the existing cache.
    - Query AM for incoming requests that are not found in the cache, and update the cache with these requests.
- `CLEAR_ON_DISCONNECT`
  - When the notification service is disconnected:
    - Clear the cache.
    - Deny access to all requests, but do not update the cache with these requests.
  - When the notification service is reconnected:
    - Query AM for all requests that are not found in the cache. (Because the cache was cleared, the cache is empty after reconnection.)
    - Update the cache with these requests.
- `CLEAR_ON_RECONNECT`

- When the notification service is disconnected:
  - Continue to use the existing cache.
  - Deny access for requests that are not cached, but do not update the cache with these requests.
- When the notification service is reconnected:
  - Query AM for all requests that are not found in the cache. (Because the cache was cleared, the cache is empty after reconnection.)
  - Update the cache with these requests.

Default: `CLEAR_ON_DISCONNECT`

`"environment"`: *map or runtime expression<map>, optional*

A list of strings to forward to AM with the request for a policy decision, that represent the environment (or context) of the request. Forward any HTTP header or any value that the AM policy definition can use.

AM can use the environment conditions to set the circumstances under which a policy applies. For example, environment conditions can specify that the policy applies only during working hours or only when accessing from a specific IP address.

If this property is a map, the structure must have the format `Map<String, List<String>>`. In the following example, the `PolicyEnforcementFilter` forwards standard and non-standard request headers, an ID token, and the IP address of the subject making the request:

```
"environment": {
  "H-Via": "${request.headers['Via']}",
  "H-X-Forwarded-For": "${request.headers['X-Forwarded-For']}",
  "H-myHeader": "${request.headers['myHeader']}",
  "id_token": [
    "${attributes.openid.id_token}"
  ],
    "IP": [
      "${contexts.client.remoteAddress}"
  ]
}
```

If this property is an expression, its evaluation must give an object of type `Map<String, List<String>>`.

```
"environment": "${attributes.my_environment}"
```

`"failureHandler"`: *handler reference, optional*

Handler to treat requests when they are denied by the policy decision.

In the following example, the `failureHandler` is a chain with a scriptable filter. If there are some advices with the policy decision, the script recovers the advices for processing. Otherwise, it passes the request to the `StaticResponseHandler` to display a message.

```
"failureHandler":{
  "type": "Chain",
  "config": {
    "filters": [
      {
        "type": "ScriptableFilter",
        "config": {
          "type": "application/x-groovy",
          "source": [
            "if (contexts.policyDecision.advices['MyCustomAdvice'] != null) {",
            "  return handleCustomAdvice(context, request)",
            "} else {",
            "  return next.handle(context, request)",
            "}"
          ]
        }
      }
    ],
    "handler": {
      "type": "StaticResponseHandler",
      "config": {
        "status": 403,
        "entity": "Restricted area. You do not have sufficient privileges."
      }
    }
  }
}
```

Provide an inline handler configuration object, or the name of a handler object that is defined in the heap.

See also "Handlers".

Default: HTTP 403 Forbidden, the request stops being executed.

`"resourceUriProvider"`: *ResourceUriProvider reference, optional*

Return the resource URL to include in policy decision requests to AM. Configure one of the following ResourceUriProviders inline or in the heap:

- `RequestResourceUriProvider`
- `ScriptableResourceUriProvider`

Default: `RequestResourceUriProvider`

`RequestResourceUriProvider`

Return the resource URL to include in policy decision requests to AM by using the original URI of the request or the `baseURI` of the route.

```
"resourceUriProvider": {
  "type": "RequestResourceUriProvider",
  "config": {
    "useOriginalUri": runtime expression<boolean>,
    "includeQueryParams": runtime expression<boolean>
  }
}
```

`useOriginalUri`: *runtime expression<boolean>, optional*

In policy decision requests to AM, use the original URI of the request as the resource URL.

- `true`: Use the original URI of the request as the resource URL when requesting policy decisions from AM.
- `false`: Use the `baseURI` of the route as the resource URL when requesting policy decisions from AM.

Default: `false`

`includeQueryParams`: *runtime expression<boolean>, optional*

Include query parameters in the resource URL when requesting policy decisions from AM:

- `true`: Include query parameters in the resource URL. For example, use the following URL with a query parameter: `http://microgateway.example.com:8080/login?demo=capture`.
- `false`: Exclude query parameters from the resource URL. For example, exclude the query parameter from the previous example: `http://microgateway.example.com:8080/login`.

  For AM policies that specify resource URLs without query parameters, use this option to reduce the amount of cached information. For information about resource URLs with query parameters, see AM's Specifying Resource Patterns with Wildcards.

Default: `true`

`ScriptableResourceUriProvider`

Use a script to return the resource URL to include in policy decision requests to AM. The result of the script must be a string that represents the resource to be used in the policy decision request sent to AM.

```
"resourceUriProvider": {
  "type": "ScriptableResourceUriProvider",
  "config": {
    "type": string,
    "file": expression,                    // Use either "file"
    "source": string or array of strings,  // or "source", but not both.
    "args": object,
    "clientHandler": Handler reference
  }
}
```

For information about these properties, see "*Scripts*".

The following example script replaces existing query parameters with a single parameter:

```
"resourceUriProvider": {
  "type": "ScriptableResourceUriProvider",
  "config": {
    "type": "application/x-groovy",
    "source": [
      "request.uri.setQuery('fromIG=true')",
      "return request.uri.toASCIIString()"
    ]
  }
}
```

3.22.4. More Information

org.forgerock.openig.openam.PolicyEnforcementFilter

org.forgerock.openig.openam.PolicyDecisionContext

"*PolicyDecisionContext*"

AM Authorization Guide

Chapter 3.23

ScriptableFilter

3.23.1. Description

Processes requests and responses by using a Groovy script.

When a ScriptableFilter processes a request, it can execute `return next.handle(context, request)` to call the next filter or handler in the current chain and return the value from the call. Actions on the response must be performed in the Promise's callback methods.

Scripts must return a Promise<Response, NeverThrowsException> or a Response.

This section describes the usage of ScriptableFilter, and refers to the following sections of the documentation:

- For information about script properties, available global objects, and automatically imported classes, see "*Scripts*".

3.23.2. Usage

```
{
    "name": string,
    "type": "ScriptableFilter",
    "config": {
        "type": string,
        "file": expression,                     // Use either "file"
        "source": string or array of strings,   // or "source", but not both.
        "args": object,
        "clientHandler": Handler reference
    }
}
```

3.23.3. Properties

For information about properties for ScriptableFilter, see "*Scripts*".

3.23.4. Examples

See the following examples of scriptable filters:

- For an example scriptable filter that recovers policy advices from AM, see the `failureHandler` property of "*PolicyEnforcementFilter*".

3.23.5. More Information

"*Scripts*"

org.forgerock.openig.filter.ScriptableFilter

Chapter 3.24

SessionInfoFilter

3.24.1. Description

This filter calls the AM endpoint for session information, and makes the data available as a new context to downstream Microgateway filters and handlers. For information, see "*SessionInfoContext*".

Supported with AM 5 and later versions.

For more information about AM sessions, see Using Sessions in the AM *Authentication and Single Sign-On Guide*.

3.24.1.1. WebSocket Notifications for Sessions

When WebSocket notifications are set up for sessions, Microgateway receives a notification from AM when a user logs out of AM, or when the AM session is modified, closed, or times out. Microgateway then evicts entries that are related to the event from the `sessionCache`.

3.24.2. Usage

```
{
  "name": string,
  "type": "SessionInfoFilter",
  "config": {
    "amService": AmService reference,
    "ssoToken": runtime expression<string>
  }
}
```

3.24.3. Properties

`"amService"`: *AmService reference, required*

The AmService heap object to use for the following properties:

- `agent`, the credentials of an Microgateway agent in AM. When the agent is authenticated, the token can be used for tasks such as getting the user's profile, making policy evaluations, and connecting to the AM notification endpoint.

- `url`, the URL of an AM service to use for session token validation and authentication.
- `amHandler`, the handler to use when communicating with AM to validate the token in the incoming request.
- `realm`: Realm of the Microgateway agent in AM.
- `version`: The version of the AM server.
- `sessionProperties`, the list of user session properties to retrieve from AM.

  The following properties are retrieved:

  - When `sessionProperties` in AmService is configured, listed session properties with a value.
  - When `sessionProperties` in AmService is not configured, all session properties with a value.
  - Properties with a value that are required by Microgateway but not specified by `sessionProperties` in AmService. For example, when the session cache is enabled, session properties related to the cache are automatically retrieved.

  Properties with a value are returned, properties with a null value are not returned.

This filter is compatible with AM version 5 or higher.

See also, "*AmService*".

`"ssoToken"`: *runtime expression<string>, optional*

Location of the AM SSO token. For example, with `${request.headers['mySsoToken'].values[0]}` the SSO token is the first value of the mySsoToken header in the request.

Default: `${request.cookies['AmService-ssoTokenHeader'][0].value}`, where `AmService-ssoTokenHeader` is the name of the header or cookie where the AmService expects to find SSO tokens.

3.24.4. More Information

org.forgerock.openig.openam.SessionInfoFilter

org.forgerock.openig.openam.SessionInfoContext

"*SessionInfoContext*"

AM Authorization Guide

Chapter 3.25
SingleSignOnFilter

3.25.1. Description

This filter tests for the presence and validity of an SSO token in the cookie header of a request:

- If an SSO token is present, the filter calls AM to validate the SSO token. If the SSO token is valid, the request continues along the chain. The token value and additional information are stored in the `ssoToken` context. For information, see "*SsoTokenContext*".
- If the SSO token is not present, or is empty or invalid, Microgateway checks the `goto` query parameter for the presence of a marker parameter:
  - If the marker parameter is present, Microgateway fails the request because the cookie domain is incorrectly configured.
  - If the marker parameter is not present, Microgateway redirects the user-agent for authentication to the AM login page or another provided login page.

Supported with AM 5 and later versions.

Tip

To prevent issues with performance when accessing large resources, such as .jpg and .js files, consider using the SingleSignOnFilter with the following options:

- The `sessionCache`, so that Microgateway can reuse session token information without repeatedly asking AM to verify the session token.
- A ConditionalFilter, so that requests to access large resources skip the SingleSignOnFilter. For an example configuration, see the example in "*ConditionalFilter*".

Note

Each time the SingleSignOnFilter validates an SSO token on AM, the idle timeout for the AM stateful session is reset. As a result, validating a session token triggers a write operation to the AM Core Token Service token store.

For information about token validation in AM, see Using Sessions in the AM *Authentication and Single Sign-On Guide*.

3.25.1.1. WebSocket Notifications for Sessions

When WebSocket notifications are set up for sessions, Microgateway receives a notification from AM when a user logs out of AM, or when the AM session is modified, closed, or times out. Microgateway then evicts entries that are related to the event from the `sessionCache`.

3.25.2. Usage

```
{
  "name": string,
  "type": "SingleSignOnFilter",
    "config": {
      "amService": AmService reference,
      "realm": string,
      "defaultLogoutLandingPage": configuration expression<url>,
      "loginEndpoint": runtime expression<url>,
      "logoutExpression": runtime expression<boolean>
  }
}
```

3.25.3. Properties

`"amService"`: *AmService reference, required*

An AmService object to use for the following properties:

- `agent`, the credentials of an Microgateway agent in AM. When the agent is authenticated, the token can be used for tasks such as getting the user's profile, making policy evaluations, and connecting to the AM notification endpoint.
- `realm`: Realm of the Microgateway agent in AM.
- `url`, the URL of an AM service to use for session token validation and authentication when `loginEndpoint` is not specified.
- `ssoTokenHeader`, the name of the cookie that contains the session token created by AM.
- `amHandler`, the handler to use when communicating with AM to validate the token in the incoming request.
- `sessionCache`, the configuration of a cache for session information from AM.
- `version`: The version of the AM server.

See also, "*AmService*".

`"realm"`: *string, optional*

The AM realm where the user is authenticated.

Default: The realm declared for `amService`.

`"defaultLogoutLandingPage"`: *configuration expression<url>, optional*

The URL to which a request is redirected after the user logs out of AM.

If this property is not an absolute URL, the request is redirected to the Microgateway domain name.

This parameter is effective only when `logoutEndpoint` is specified.

Default: None. Processing continues.

`"loginEndpoint"`: *runtime expression<url>, optional*

The URL of a service instance for the following tasks:

- Manage authentication and the location to which the request is redirected after authentication.
- Process policy advices after an AM policy decision denies a request with supported advices. The PolicyEnforcementFilter redirects the request to this URL, with information about how to meet the conditions in the advices.

  For examples of different advice types, and the conditions that cause AM to return advices, see Policy Decision Advice in the AM *Authorization Guide*. For information about supported advice types in Microgateway, see "Using Advices From Policy Decisions".

Default: The value of `url` in `amService`

Authentication can be performed in the following ways:

- Directly through AM, with optional authentication parameters in the query string, such as `service`, `module`, and `realm`. For a list of authentication parameters that you can include in the query string, see Authentication Parameters in the AM *Authentication and Single Sign-On Guide*.

  The value must include a redirect with a `goto` parameter.

  The following example uses AM as the authentication service, and includes the `service` authentication parameter:

  ```
  "loginEndpoint": "https://openam.example.com/openam?
  service=TwoFactor&goto=${urlEncodeQueryParameterNameOrValue(contexts.router.originalUri)}"
  ```

- Through the URL of another application, with optional authentication parameters in the query string, such as `service`, `module`, and `realm`. The application must create a session with an AM instance to set an SSO token and return the request to the redirect location.

  The value can optionally include a redirect with a `goto` parameter or different parameter name.

  The following example uses an authentication service that is not AM, and includes a redirect parameter:

  ```
  "loginEndpoint": "https://authservice.example.com/auth?
  redirect=${urlEncodeQueryParameterNameOrValue(contexts.router.originalUri)}"
  ```

  When using this option, review the cookie domains to make sure that cookies set by the authentication server are properly conveyed to the Microgateway instance.

`"logoutExpression"` *runtime expression<boolean>, optional*

An expression to define a condition for logout, based on the request. If the expression evaluates to `true`, the AM authentication token for the end user is revoked.

If a `defaultLogoutLandingPage` is specified, the request is redirected to that page. Otherwise, the request continues to be processed.

The following example expressions can be used to trigger revocation of the end user token:

- The request URI contains `/logout`:

  ```
  ${matches(request.uri, '/logout')}
  ```

- The request path ends in `/logout`:

  ```
  ${matches(request.uri.path, '^/logout')}
  ```

- The request query includes the `logOff=true` query parameter:

  ```
  ${matches(request.uri.query, 'logOff=true')}
  ```

Default: Logout is not managed by this filter.

3.25.4. More Information

org.forgerock.openig.openam.SingleSignOnFilter

org.forgerock.openig.openam.SsoTokenContext

"*SsoTokenContext*"

Chapter 3.26

SqlAttributesFilter

3.26.1. Description

Executes a SQL query through a prepared statement and exposes its first result. Parameters in the prepared statement are derived from expressions. The query result is exposed in an object whose location is specified by the `target` expression. If the query yields no result, then the resulting object is empty.

The execution of the query is performed lazily; it does not occur until the first attempt to access a value in the target. This defers the overhead of connection pool, network and database query processing until a value is first required. This also means that the parameters expressions is not evaluated until the object is first accessed.

3.26.2. Usage

```
{
     "name": string,
     "type": "SqlAttributesFilter",
     "config": {
         "dataSource": JdbcDataSource reference,
         "preparedStatement": string,
         "parameters": [ runtime expression<string>, ... ],
         "target": lvalue-expression
     }
}
```

3.26.3. Properties

`"dataSource"`: *JdbcDataSource reference, required*

The JdbcDataSource to use for connections. Configure JdbcDataSource as described in "*JdbcDataSource*".

`"preparedStatement"`: *string, required*

The parameterized SQL query to execute, with `?` parameter placeholders.

`"parameters"`: *array of runtime expressions<string>, optional*

The parameters to evaluate and include in the execution of the prepared statement.

See also "*Expressions*".

`"target"`: *lvalue-expression, required*

Expression that yields the target object that will contain the query results.

See also "*Expressions*".

3.26.4. Example

Using the user's session ID from a cookie, query the database to find the user logged in and set the profile attributes in the attributes context:

```
{
        "name": "SqlAttributesFilter",
        "type": "SqlAttributesFilter",
        "config": {
              "target": "${attributes.sql}",
              "dataSource": "java:comp/env/jdbc/mysql",
              "preparedStatement": "SELECT f.value AS 'first', l.value AS
                'last', u.mail AS 'email', GROUP_CONCAT(CAST(r.rid AS CHAR)) AS
                'roles'
                FROM sessions s
                INNER JOIN users u
                ON ( u.uid = s.uid AND u.status = 1 )
                LEFT OUTER JOIN profile_values f
                ON ( f.uid = u.uid AND f.fid = 1 )
                LEFT OUTER JOIN profile_values l
                ON ( l.uid = u.uid AND l.fid = 2 )
                LEFT OUTER JOIN users_roles r
                ON ( r.uid = u.uid )
                WHERE (s.sid = ? AND s.uid <> 0) GROUP BY s.sid;",
              "parameters": [ "${request.cookies
                [keyMatch(request.cookies,'JSESSION1234')]
                [0].value}" ]
         }
 }
```

Lines are folded for readability in this example. In your JSON, keep the values for `"preparedStatement"` and `"parameters"` on one line.

3.26.5. More Information

org.forgerock.openig.filter.SqlAttributesFilter

Chapter 3.27
StaticRequestFilter

3.27.1. Description

Creates a new request, replacing any existing request. The request can include an entity specified in the `entity` parameter. Alternatively, the request can include a form, specified in the `form` parameter, which is included in an entity encoded in `application/x-www-form-urlencoded` format if request method is `POST`, or otherwise as (additional) query parameters in the URI. The `form` and `entity` parameters cannot be used together when the `method` is set to `POST`.

3.27.2. Usage

```
{
     "name": string,
     "type": "StaticRequestFilter",
     "config": {
         "method": string,
         "uri": runtime expression<uri string>,
         "version": string,
         "headers": {
             configuration expression<string>: [ runtime expression<string>, ... ], ...
         },
         "form": {
             configuration expression<string>: [ runtime expression<string>, ... ], ...
         },
         "entity": runtime expression<string>
     }
}
```

3.27.3. Properties

`"method"`: *string, required*

The HTTP method to be performed on the resource (for example, `"GET"`).

`"uri"`: *runtime expression<uri string>, required*

The fully-qualified URI of the resource to access (for example, `"http://www.example.com/resource.txt"`).

The result of the expression must be a string that represents a valid URI, but is not a real `java.net.URI` object. For example, it would be incorrect to use `${request.uri}`, which is not a String but a MutableUri.

`"version"`: *string, optional*

Protocol version. Default: `"HTTP/1.1"`.

`"headers"`: *object, optional*

Header fields to set in the request, with the format `name: [ value, ... ]`.

The `name` field of `headers` specifies the header name. It can be defined by a configuration expression or string. If the configuration expressions for multiple `name` resolve to the same final string, multiple values are associated with the `name`.

The `value` field of `headers` is an array of runtime expressions to evaluate as header values.

In the following example, the name of the header is the value of the configuration time, system variable defined in `cookieHeaderName`. The value of the header is the runtime value stored in `contexts.ssoToken.value`:

```
"headers": {
  "${application['header1Name']}": [
    "${application['header1Value'}"
  ]
```

`"form"`: *object, optional*

A form to include in the request, with the format `param: [ value, ... ]`.

The `param` field of `form` specifies the name of the form parameter. It can be defined by a configuration expression or string. If the configuration expressions for multiple `param` resolve to the same final string, multiple values are associated with the `param`.

The `value` field of `form` is an array of runtime expressions to evaluate as form field values.

When the `method` is set to `POST`, this setting is mutually exclusive with the `entity` setting.

In the following example, the names of the field parameters and the values are hardcoded in the form:

```
"form": {
  "username": [
    "demo"
  ],
  "password": [
    "changeit"
  ]
```

In the following example, the names of the field parameters are hardcoded. The values take the first value of `username` and `password` provided in the session:

```
"form": {
  "username": [
    "${session.username[0]}"
  ],
  "password": [
    "${session.password[0]}"
  ]
```

In the following example, the name of the first field param take the value of the expression `${application['formName']}` when it is evaluated at startup. The values take the first value of `username` and `password` provided in the session:

```
"form": {
  "${application['formName']}": [
    "${session.username[0]}"
  ],
  "${application['formPassword']}": [
    "${session.password[0]}"
  ]
```

`"entity"`: *runtime expression<string>, optional*

The entity body to include in the request.

This setting is mutually exclusive with the `form` setting when the `method` is set to `POST`.

See also "*Expressions*".

3.27.4. More Information

org.forgerock.openig.filter.StaticRequestFilter

Chapter 3.28
SwitchFilter

3.28.1. Description

Verifies that a specified condition is met. If the condition is met or no condition is specified, the request is diverted to the associated handler, with no further processing by the switch filter.

3.28.2. Usage

```
{
    "name": string,
    "type": "SwitchFilter",
    "config": {
        "onRequest": [
            {
                "condition": runtime expression<boolean>,
                "handler": Handler reference,
            },
            ...
        ],
        "onResponse": [
            {
                "condition": runtime expression<boolean>,
                "handler": Handler reference,
            },
            ...
        ]
    }
}
```

3.28.3. Properties

`"onRequest"`: *array of objects, optional*

Conditions to test (and handler to dispatch to, if `true`) before the request is handled.

`"onResponse"`: *array of objects, optional*

Conditions to test (and handler to dispatch to, if `true`) after the response is handled.

`"condition"`: *runtime expression<boolean>, optional*

If the expression evaluates to `true`, the request is dispatched to the handler. If no condition is specified, the request is dispatched to the handler unconditionally.

Default: No condition is specified.

See also "*Expressions*".

`"handler"`: *Handler reference, required*

Dispatch to this handler if the condition yields `true`.

Provide either the name of a Handler object defined in the heap, or an inline Handler configuration object.

See also "Handlers".

3.28.4. Example

This example intercepts the response if it is equal to 200 and executes the LoginRequestHandler. This filter might be used in a login flow where the request for the login page must go through to the target, but the response should be intercepted in order to send the login form to the application. This is typical for scenarios where there is a hidden value or cookie returned in the login page, which must be sent in the login form:

```
{
    "name": "SwitchFilter",
    "type": "SwitchFilter",
    "config": {
        "onResponse": [
            {
                "condition": "${response.status.code == 200}",
                "handler": "LoginRequestHandler"
            }
        ]
    }
}
```

3.28.5. More Information

org.forgerock.openig.filter.SwitchFilter

Chapter 3.29

TokenTransformationFilter

3.29.1. Description

This filter transforms a token issued by AM to another token type.

The TokenTransformationFilter makes the result of the token transformation available to downstream handlers in the `sts` context. For information, see "*StsContext*".

The current implementation uses REST Security Token Service (STS) APIs to transform an OpenID Connect ID Token (`id_token`) into a SAML 2.0 assertion. The subject confirmation method is Bearer, as described in *Profiles for the OASIS Security Assertion Markup Language (SAML) V2.0* .

The TokenTransformationFilter makes the result of the token transformation available to downstream handlers in the `issuedToken` property of the `${contexts.sts}` context.

The TokenTransformationFilter configuration references a REST STS instance that must be set up in AM before the TokenTransformationFilter can be used. The REST STS instance exposes a preconfigured transformation under a specific REST endpoint. For information about setting up a REST STS instance, see the AM documentation.

Errors that occur during the token transformation cause a error response to be returned to the client and an error message to be logged for the Microgateway administrator.

Supported with OpenAM 13.5, and AM 5 and later versions.

3.29.2. Usage

```
{
    "name": "string",
    "type": "TokenTransformationFilter",
    "config": {
        "amService": AmService reference,
        "idToken": runtime expression<string>,
        "instance": "expression"
    }
}
```

3.29.3. Properties

`"amService"`: *AmService reference, required*

The AmService heap object to use for the following properties:

- `agent`, the credentials of an Microgateway agent in AM, to authenticate Microgateway as an AM REST STS client, and to communicate WebSocket notifications from AM to Microgateway. This credentials are evaluated when the route is initialized
- `url`, the URL of an AM service to use for session token validation and authentication. Authentication and REST STS requests are made to this service.
- `realm`, the AM realm containing the following information:
  - The AM application that can make the REST STS request and whose credentials are the username and password.
  - The STS instance described by the instance field.
- `ssoTokenHeader`, the name of the HTTP header that provides the SSO token for the REST STS client subject.
- `amHandler`, the handler to use for authentication and STS requests to AM.

See also, "*AmService*".

`"idToken"`: *runtime expression<string>, required*

The value of the OpenID Connect ID token. The expected value is a string that is the JWT encoded `id_token`.

See also "*Expressions*".

`"instance"`: *expression, required*

An expression evaluating to the name of the REST STS instance.

This expression is evaluated when the route is initialized, so the expression cannot refer to `request` or `contexts`.

See also "*Expressions*".

3.29.4. Example

The following example shows a configuration for a TokenTransformationFilter:

```
{
  "type": "TokenTransformationFilter",
  "config": {
    "amService": "MyAmService",
    "idToken": "${attributes.openid.id_token}",
    "instance": "openig"
  }
}
```

3.29.5. More Information

org.forgerock.openig.openam.TokenTransformationFilter

org.forgerock.openig.openam.StsContext

"*StsContext*"

Chapter 3.30

UmaFilter

3.30.1. Description

This filter acts as a policy enforcement point, protecting access as a User-Managed Access (UMA) resource server. Specifically, this filter ensures that a request for protected resources includes a valid requesting party token with appropriate scopes before allowing the response to flow back to the requesting party.

UMA 2.0 is supported with AM 5.5 and later versions. UMA 1.0 is supported with AM 5.1 and later versions.

3.30.2. Usage

```
{
    "type": "UmaFilter",
    "config": {
        "protectionApiHandler": Handler reference,
        "umaService": UmaService reference,
        "realm": string
    }
}
```

3.30.3. Properties

`"protectionApiHandler"`: *Handler reference, required*

The handler to use when interacting with the UMA authorization server for token introspection and permission requests, such as a ClientHandler capable of making an HTTPS connection to the server.

For details, see "Handlers".

`"umaService"`: *UmaService reference, required*

The UmaService to use when protecting resources.

For details, see "*UmaService*".

`"realm"`: *string, optional*

The UMA realm set in the response to a request for a protected resource that does not include a requesting party token enabling access to the resource.

Default: `uma`

3.30.4. More Information

User-Managed Access (UMA) Profile of OAuth 2.0

org.forgerock.openig.uma.UmaResourceServerFilter

Chapter 3.31

UserProfileFilter

3.31.1. Description

This filter queries AM to retrieve the profile attributes of an user identified by their `username`.

Only profile attributes that are enabled in AM can be returned by the query. The `roles` field is not returned.

The data is made available to downstream Microgateway filters and handlers, in the context "*UserProfileContext*".

Supported with AM 5 and later versions.

3.31.2. Usage

```
{
  "name": string,
  "type": "UserProfileFilter",
  "config": {
    "username": configuration expression<string>,
    "userProfileService": UserProfileService reference
  }
}
```

3.31.3. Properties

`"username"`: *configuration expression<string>, required*

The username of an AM subject. This filter retrieves profile attributes for the subject.

`"userProfileService"`: *UserProfileService reference, required*

The service to retrieve profile attributes from AM, for the subject identified by `username`.

```
"userProfileService": {
  "type": "UserProfileService",
  "config": {
    "amService": AmService reference,
    "cache": object,
    "profileAttributes": array of runtime expression<string>,
    "realm": configuration expression<string>
  }
}
```

`"amService"`: *AmService reference, required*

The AmService heap object to use for the following properties:

- `agent`, the credentials of an Microgateway agent in AM. When the agent is authenticated, the token can be used for tasks such as getting the user's profile, making policy evaluations, and connecting to the AM notification endpoint.
- `url`: URL of the AM server where the user is authenticated.
- `amHandler`: Handler to use when communicating with AM to fetch the requested user's profile.
- `realm`: Realm of the Microgateway agent in AM.
- `version`: The version of the AM server.

`cache`: *object, optional*

Caching of AM user profiles, based on *Caffeine*. For more information, see the GitHub entry, *Caffeine* .

When caching is enabled, Microgateway can reuse cached profile attributes without repeatedly querying AM. When caching is disabled, Microgateway must query AM for each request, to retrieve the required user profile attributes.

Default: No cache.

`enabled`: *boolean, optional*

Enable or disable caching of user profiles. When `false`, the cache is disabled but the cache configuration is maintained.

Default: `true` when `cache` is configured

`executor`: *executor, optional*

An executor service to schedule the execution of tasks, such as the eviction of entries in the cache.

Default: `ForkJoinPool.commonPool()`

`"maximumSize"`: *configuration expression<number>, optional*

The maximum number of entries the cache can contain.

Default: Unlimited/unbound

`maximumTimeToCache`: *duration, required*

The maximum duration for which to cache user profiles.

The duration cannot be `zero`.

For information about supported formats for `duration`, see duration.

`profileAttributes`: *array of runtime expression<string>, optional*

List of one or more fields to return and store in UserProfileContext.

Field names are defined by the underlying repository in AM. When AM is installed with the default configuration, the repository is ForgeRock Directory Services.

The following convenience accessors are provided for commonly used fields:

- `cn`: Retrieved through `${contexts.userProfile.commonName}`
- `dn`: Retrieved through `${contexts.userProfile.distinguishedName}`
- `realm`: Retrieved through `${contexts.userProfile.realm}`
- `username`: Retrieved through `${contexts.userProfile.username}`

All other available fields can be retrieved through `${contexts.userProfile.rawInfo}` and `${contexts.userProfile.asJsonValue()}`.

When `profileAttributes` is configured, the specified fields **and** the following fields are returned: `username`, `_id`, and `_rev`.

Default: All available fields are returned.

`"realm"`: *configuration expression<string>, optional*

The AM realm where the subject is authenticated.

Default: The realm declared for `amService`.

3.31.4. More Information

org.forgerock.openig.openam.UserProfileFilter

org.forgerock.openig.tools.userprofile.UserProfileService

org.forgerock.openig.openam.UserProfileContext

"*UserProfileContext*"

AM Authorization Guide

Part 4

Decorators

Microgateway provides decorators to extend what objects can do.

Chapter 4.1

BaseUriDecorator

4.1.1. Description

Overrides the scheme, host, and port of the existing request URI, rebasing the URI and so making requests relative to a new base URI. Rebasing changes only the scheme, host, and port of the request URI. Rebasing does not affect the path, query string, or fragment.

4.1.2. Decorator Usage

```
{
    "name": string,
    "type": "BaseUriDecorator"
}
```

A BaseUriDecorator does not have configurable properties.

Microgateway creates a default BaseUriDecorator named baseURI at startup time in the top-level heap, so you can use baseURI as the decorator name without adding the decorator declaration explicitly.

4.1.3. Decorated Object Usage

```
{
    "name": string,
    "type": string,
    "config": object,
    decorator name: runtime expression<uri string>
}
```

`"name"`: string, required except for inline objects

The unique name of the object, just like an object that is not decorated

`"type"`: string, required

The class name of the decorated object, which must be either a Filter or a Handler.

See also "Filters" and "Handlers".

`"config"`: object, required unless empty

The configuration of the object, just like an object that is not decorated

***decorator name*: runtime expression<uri string>, required**

The scheme, host, and port of the new base URI. The port is optional when using the defaults (80 for HTTP, 443 for HTTPS).

The value of the string must not contain underscores, and must conform to the syntax specified in RFC 3986.

4.1.4. Examples

Add a custom decorator to the heap named myBaseUri:

```
{
    "name": "myBaseUri",
    "type": "BaseUriDecorator"
}
```

Set a Router's base URI to `https://www.example.com:8443`:

```
{
    "name": "Router",
    "type": "Router",
    "myBaseUri": "https://www.example.com:8443/"
}
```

4.1.5. More Information

org.forgerock.openig.decoration.baseuri.BaseUriDecorator

Chapter 4.2

CaptureDecorator

4.2.1. Description

Captures request and response messages for further analysis.

4.2.2. Decorator Usage

```
{
    "name": string,
    "type": "CaptureDecorator",
    "config": {
        "captureEntity": boolean,
        "captureContext": boolean,
        "maxEntityLength": number
    }
}
```

Captured information is written to SLF4J logs, and named in this format:

```
org.forgerock.openig.decoration.capture.CaptureDecorator.<decoratorName>.<decoratedObjectName>
```

If the decorated object is not named, the object path is used in the log.

The decorator configuration has these properties:

`"captureEntity"`: *boolean, optional*

Whether the message entity should be captured. The message entity is the body of the HTTP message, which can be a JSON document, XML, HTML, image, or other information.

The filter omits binary entities, instead writing a `[binary entity]` marker to the file.

Default: false

`"captureContext"`: *boolean, optional*

Whether the context should be captured as JSON. The context chain is used when processing the request inside Microgateway in the filters and handlers.

Default: false

`"maxEntityLength"`: *number, optional*

The maximum number of bytes that can be captured for an entity. This property is used when `captureEntity` is `true`.

If the captured entity is bigger than `maxEntityLength`, everything up to `maxEntityLength` is captured, and an `[entity truncated]` message is written in the log.

Set `maxEntityLength` to be big enough to allow capture of normal entities, but small enough to prevent excessive memory use or `OutOfMemoryError` errors. Setting `maxEntityLength` to 2 GB or more causes an exception at startup.

Default: 524 288 bytes (512 KB)

4.2.3. Decorated Object Usage

```
{
    "name": string,
    "type": string,
    "config": object,
    decorator name: capture point(s)
}
```

`"name"`: string, required except for inline objects

The unique name of the object, just like an object that is not decorated

`"type"`: string, required

The class name of the decorated object, which must be either a Filter or a Handler.

See also "Filters" and "Handlers".

`"config"`: object, required unless empty

The configuration of the object, just like an object that is not decorated

***decorator name*: capture point(s), optional**

The *decorator name* must match the name of the CaptureDecorator. For example, if the CaptureDecorator has `"name": "capture"`, then *decorator name* is capture.

The capture point(s) are either a single string, or an array of strings. The strings are documented here in lowercase, but are not case-sensitive:

`"all"`

Capture at all available capture points.

`"none"`

Disable capture.

If `none` is configured with other capture points, `none` takes precedence.

`"request"`

Capture the request as it enters the Filter or Handler.

`"filtered_request"`

Capture the request as it leaves the Filter.

Only applies to Filters.

`"response"`

Capture the response as it enters the Filter or leaves the Handler.

`"filtered_response"`

Capture the response as it leaves the Filter.

Only applies to Filters.

4.2.4. Examples

Decorator configured to log the entity:

```
{
    "name": "capture",
    "type": "CaptureDecorator",
    "config": {
        "captureEntity": true
    }
}
```

Decorator configured not to log the entity:

```
{
    "name": "capture",
    "type": "CaptureDecorator"
}
```

Decorator configured to log the context in JSON format, excluding the request and the response:

```
{
    "name": "capture",
    "type": "CaptureDecorator",
    "config": {
        "captureContext": true
    }
}
```

To capture requests and responses with the entity before sending the request and before returning the response, do so as in the following example:

```
{
    "heap": [
        {
            "name": "capture",
            "type": "CaptureDecorator",
            "config": {
                "captureEntity": true
            }
        },
        {
            "name": "ReverseProxyHandler",
            "type": "ReverseProxyHandler",
            "capture": [
                "request",
                "response"
            ]
        }
    ],
    "handler": "ReverseProxyHandler"
}
```

To capture all transformed requests and responses as they leave filters, decorate the Route as in the following example. This Route uses the default CaptureDecorator:

```
{
    "handler": {
        "type": "Chain",
        "config": {
            "filters": [
                {
                    "type": "HeaderFilter",
                    "config": {
                        "messageType": "REQUEST",
                        "add": {
                            "X-RequestHeader": [
                                "Capture at filtered_request point",
                                "And at filtered_response point"
                            ]
                        }
                    }
                },
                {
                    "type": "HeaderFilter",
                    "config": {
                        "messageType": "RESPONSE",
                        "add": {
                            "X-ResponseHeader": [
                                "Capture at filtered_response point"
                            ]
                        }
                    }
                }
            ],
            "handler": {
```

```
                "type": "StaticResponseHandler",
                "config": {
                    "status": 200,
                    "reason": "OK",
                    "entity": "<html><body><p>Hello, World!</p></body></html>"
                }
            }
        }
    },
    "capture": [
        "filtered_request",
        "filtered_response"
    ]
}
```

To capture the context as JSON, excluding the request and response, before sending the request and before returning the response, do so as in the following example:

```
{
    "heap": [
        {
            "name": "capture",
            "type": "CaptureDecorator",
            "config": {
                "captureContext": true
            }
        },
        {
            "name": "ReverseProxyHandler",
            "type": "ReverseProxyHandler",
            "capture": [
                "request",
                "response"
            ]
        }
    ],
    "handler": "ReverseProxyHandler"
}
```

4.2.5. More Information

org.forgerock.openig.decoration.capture.CaptureDecorator

Chapter 4.3
TimerDecorator

4.3.1. Description

Records time in milliseconds to process filters and handlers.

4.3.2. Decorator Usage

```
{
    "name": string,
    "type": "TimerDecorator",
    "config": {
        "timeUnit": string
    }
}
```

Microgateway configures a default TimerDecorator named `timer`. You can use `timer` as the decorator name without explicitly declaring a decorator named timer.

`"timeUnit"`: *duration string, optional*

Unit of time used in the decorator output. The unit of time can be any unit allowed in the `duration` field.

Default: `ms`

For information about supported formats for `duration`, see duration.

4.3.3. Decorated Object Usage

```
{
    "name": string,
    "type": string,
    "config": object,
    decorator name: boolean
}
```

`"name"`: string, required except for inline objects

The unique name of the object, just like an object that is not decorated.

`"type"`: string, required

The class name of the decorated object, which must be either a filter or a handler.

See also "Filters" and "Handlers".

`"config"`: object, required unless empty

The configuration of the object, just like an object that is not decorated.

***`decorator name`*: boolean, required**

Microgateway looks for the presence of the *decorator name* field for the TimerDecorator.

To activate the timer, set the value of the *decorator name* field to `true`.

To deactivate the TimerDecorator temporarily, set the value to `false`.

4.3.4. Timer Metrics At the Prometheus Scrape Endpoint

This section describes the timer metrics recorded at the Prometheus Scrape Endpoint. For more information about metrics, see "*Monitoring Endpoints*".

When Microgateway is set up as described in the documentation, the endpoint is http://microgateway.example.com:8080/openig/metrics/prometheus.

Each timer metric is labelled with the following fully qualified names:

- `decorated_object`
- `heap`
- `name` (decorator name)
- `route`
- `router`

Timer Metrics At the Prometheus Scrape Endpoint

Name	Type [a]	Description
`ig_timerdecorator_handler_elapsed_seconds`	`Summary`	Time to process the request and response in the decorated handler.
`ig_timerdecorator_filter_elapsed_seconds`	`Summary`	Time to process the request and response in the decorated filter **and** its downstream filters and handler.
`ig_timerdecorator_filter_internal_seconds`	`Summary`	Time to process the request and response in the decorated filter.

Name	Type [a]	Description
`ig_timerdecorator_filter_downstream_seconds`	`Summary`	Time to process the request and response in filters and handlers that are downstream of the decorated filter.

[a]As described in "*Monitoring Types*"

4.3.5. Timer Metrics At the Common REST Monitoring Endpoint

This section describes the metrics recorded at the the ForgeRock Common REST Monitoring Endpoint. For more information about metrics, see "*Monitoring Endpoints*".

When Microgateway is set up as described in the documentation, the endpoint is http://microgateway.example.com:8080/openig/metrics/api?_queryFilter=true.

Metrics are published with an `_id` in the following pattern:

- *`heap.router-name.route-name.decorator-name.object`*

Timer Metrics At the Common REST Monitoring Endpoint

Name	Type [a]	Description
`elapsed`	`Timer`	Time to process the request and response in the decorated handler, or in the decorated filter **and** its downstream filters and handler.
`internal`	`Timer`	Time to process the request and response in the decorated filter.
`downstream`	`Timer`	Time to process the request and response in filters and handlers that are downstream of the decorated filter.

[a]As described in "*Monitoring Types*"

4.3.6. Timer Metrics in SLF4J Logs

SLF4J logs are named in this format:

```
<className>.<decoratorName>.<decoratedObjectName>
```

If the decorated object is not named, the object path is used in the log.

When a route's top-level handler is decorated, the timer decorator records the elapsed time for operations traversing the whole route:

```
2018-09-04T12:16:08,994Z | INFO  | I/O dispatcher 17 | o.f.o.d.t.T.t.top-level-handler | @myroute |
 Elapsed time: 13 ms
```

When an individual handler in the route is decorated, the timer decorator records the elapsed time for operations traversing the handler:

```
2018-09-04T12:44:02,161Z | INFO  | http-nio-8080-exec-8 | o.f.o.d.t.T.t.StaticResponseHandler-1 | @myroute
 | Elapsed time: 1 ms
```

4.3.7. More Information

org.forgerock.openig.decoration.timer.TimerDecorator

Part 5

Audit Framework

Microgateway uses the ForgeRock common audit framework to record audit events, using an implementation that is common across the ForgeRock platform.

Audit logs use timestamps in UTC format (for example, 2018-07-18T08:48:00.160Z), a unified standard that is not affected by time changes for daylight savings. The timestamps format is not configurable.

Chapter 5.1
AuditService

5.1.1. Description

Configure the audit service, based on the ForgeRock common audit event framework:

- To record `access` audit events, configure AuditService inline in a route, or in the heap.
- To record custom audit events, configure AuditService in the heap, and refer to it from the route or subroutes.

5.1.2. Default Audit Service

By default, there is no auditing of a configuration. The NoOpAuditService provides an empty audit service to the top-level heap and its child routes.

To configure auditing for the whole configuration, define an AuditService object named `AuditService` in the top-level heap.

To prevent a route from inheriting the NoOpAuditService or a parent audit service, do one of the following:

- Define an AuditService object named `AuditService` in the route heap. No other configuration is required.
- Configure the Route property `auditService` with an inline audit service or a reference to an AuditService object defined in the route heap or a parent heap.

For more information, see "*NoOpAuditService*".

5.1.3. Usage

```
{
  "name": string,
  "type": "AuditService",
  "config": {
    "config": object,
    "eventHandlers": array,
    "topicsSchemasDirectory": configuration expression<string>
  }
}
```

5.1.4. Properties

`"config"`: *object, required*

Configures the audit service itself, rather than event handlers. If the configuration uses only default settings, you can omit the field instead of including an empty object as the field value.

The configuration object has the following fields:

`"handlerForQueries"`: *string, optional*

The name of the event handler to use when querying audit event messages over REST.

`"availableAuditEventHandlers"`: *array of strings, optional*

A list of fully qualified event handler class names for event handlers available to the audit service.

`"caseInsensitiveFields"`: *array of strings, optional*

A list of audit event fields to be considered as case-insensitive for filtering. The fields are referenced using JSON pointer syntax. The list can be `null` or empty.

Default: `/access/http/request/headers` and `/access/http/response/headers` fields are considered case-insensitive for filtering. All other fields are considered case-sensitive.

`"filterPolicies"`: *object, optional*

To prevent logging of sensitive data for an event, the Common Audit implementation uses a whitelist to specify which event fields appear in the logs. By default, only event fields that are whitelisted are included in the audit event logs.

`"field"`: *object, optional*

This property specifies non-whitelisted event fields to include in the logs, and whitelisted event fields to exclude from the logs.

If `includeIf` and `excludeIf` are specified for the same field, `excludeIf` takes precedence.

Audit event fields use JSON pointer notation, and are taken from the JSON schema for the audit event content.

Default: Include only whitelisted event fields in the logs.

`"includeIf"`: *array of strings, optional*

A list of non-whitelisted audit event fields to include in the logs. Specify the topic and the hierarchy to the field. Any child fields of the specified field are encompassed.

Important

Before you include non-whitelisted event fields in the logs, consider the impact on security. Including some headers, query parameters, or cookies in the logs could cause credentials or tokens to be logged, and allow anyone with access to the logs to impersonate the holder of these credentials or tokens.

`"excludeIf"`: ***array of strings, optional***

A list of whitelisted audit event fields to exclude from the logs. Specify the topic and the hierarchy to the field. Any child fields of the specified field are encompassed.

The following example excludes fields for the `access` topic:

```
{
  "field": {
    "excludeIf": [
      "/access/http/request/headers/host",
      "/access/http/request/path",
      "/access/server",
      "/access/response"
    ]
  }
}
```

`"eventHandlers"`: ***array of configuration objects, required***

An array of one or more audit event handler configuration objects to deal with audit events.

The configuration of the event handler depends on type of event handler. Microgateway supports the event handlers listed in "Audit Framework".

`"topicsSchemasDirectory"`: ***configuration expression<string>, optional***

Directory containing the JSON schema for the topic of a custom audit event. The schema defines which fields are included in the topic. For information about the syntax, see JSON Schema.

Default: `/path/to/microservices/identity-gateway/audit-schemas` (on Windows, `%appdata%\OpenIG\audit-schemas`)

5.1.5. Example

The following example audit service logs access event messages in a comma-separated variable file, named `/path/to/audit/logs/access.csv`:

```
{
  "name": "AuditService",
  "type": "AuditService",
  "config": {
    "config": {},
    "eventHandlers": [
      {
        "class": "org.forgerock.audit.handlers.csv.CsvAuditEventHandler",
        "config": {
          "name": "csv",
          "logDirectory": "/path/to/audit/logs",
          "topics": [
            "access"
          ]
        }
      }
    ]
  }
}
```

The following example route uses the audit service:

```
{
  "handler": "ReverseProxyHandler",
  "auditService": "AuditService"
}
```

5.1.6. More Information

"*NoOpAuditService*"

org.forgerock.audit.AuditService

Chapter 5.2

NoOpAuditService

5.2.1. Description

Provides an empty audit service to the top-level heap and its child routes. When an AuditService is not explicitly defined, there is by default no auditing of the configuration.

For information about how to override the default audit service, see "Default Audit Service".

5.2.2. Usage

```
{
  "name": "AuditService",
  "type": "NoOpAuditService"
 }
```

```
"auditService": "NoOpAuditService"
```

5.2.3. More Information

"*AuditService*"

org.forgerock.audit.NoOpAuditService

Chapter 5.3

CsvAuditEventHandler

5.3.1. Description

An audit event handler that responds to events by logging messages to files in comma-separated variable (CSV) format.

Declare the configuration in an audit service, as described in "*AuditService*".

Important

The CSV handler does not sanitize messages when writing to CSV log files.

Do not open CSV logs in spreadsheets and other applications that treat data as code.

5.3.2. Usage

```
{
  "class": "org.forgerock.audit.handlers.csv.CsvAuditEventHandler",
  "config": {
    "name": configuration expression<string>,
    "logDirectory": configuration expression<string>,
    "topics": [ configuration expression<string>, ... ],
    "enabled": configuration expression<boolean>,
    "formatting": {
      "quoteChar": configuration expression<single-character string>,
      "delimiterChar": configuration expression<single-character string>,
      "endOfLineSymbols": configuration expression<string>
    },
    "buffering": {
      "enabled": configuration expression<boolean>,
      "autoFlush": configuration expression<boolean>
    },
    "security": {
      "enabled": configuration expression<boolean>,
      "filename": configuration expression<string>,
      "password": configuration expression<string>,
      "signatureInterval": configuration expression<duration>
    },
    "fileRetention": {
      "maxDiskSpaceToUse": configuration expression<number>,
      "maxNumberOfHistoryFiles": configuration expression<number>,
      "minFreeSpaceRequired": configuration expression<number>
    },
    "fileRotation": {
```

```
      "rotationEnabled": configuration expression<boolean>,
      "maxFileSize": configuration expression<number>,
      "rotationFilePrefix": configuration expression<string>,
      "rotationFileSuffix": configuration expression<string>,
      "rotationInterval": configuration expression<duration>,
      "rotationTimes": [ configuration expression<duration>, ... ]
    },
      "rotationRetentionCheckInterval": configuration expression<duration>
  }
}
```

The values in this configuration object can use expressions as long as they resolve to the correct types for each field. For details about expressions, see "*Expressions*".

5.3.3. Configuration

The `"config"` object has the following properties:

`"name"`: *configuration expression<string>, required*

The name of the event handler.

`"logDirectory"`: *configuration expression<string>, required*

The file system directory where log files are written.

`"topics"`: *array of configuration expression<string>, required*

An array of one or more topics that this event handler intercepts. Microgateway can record the following audit event topics:

- `access`: Log access audit events. Access audit events occur at the system boundary, and include the arrival of the initial request and departure of the final response.

  To record `access` audit events, configure AuditService inline in a route, or in the heap.

- *customTopic*: Log custom audit events. To create a topic for a custom audit event, include a JSON schema for the topic in your Microgateway configuration.

  To record custom audit events, configure AuditService in the heap, and refer to it from the route or subroutes.

`"enabled"`: *configuration expression<boolean>, optional*

Whether this event handler is active.

Default: true

`"formatting"`: *object, optional*

Formatting settings for CSV log files.

The formatting object has the following fields:

`"quoteChar"`: *configuration expression<single-character string>, optional*

The character used to quote CSV entries.

Default: `"`

`"delimiterChar"`: *configuration expression<single-character string>, optional*

The character used to delimit CSV entries.

Default: `,`

`"endOfLineSymbols"`: *configuration expression<string>, optional*

The character or characters that separate a line.

Default: system-dependent line separator defined for the JVM

`"buffering"`: *object, optional*

Buffering settings for writing CSV log files. The default is for messages to be written to the log file for each event.

The buffering object has the following fields:

`"enabled"`: *configuration expression<boolean>, optional*

Whether log buffering is enabled.

Default: false

`"autoFlush"`: *configuration expression<boolean>, optional*

Whether events are automatically flushed after being written.

Default: true

`"security"`: *configuration expression<object>, optional*

Security settings for CSV log files. These settings govern tamper-evident logging, whereby messages are signed. By default tamper-evident logging is not enabled.

The security object has the following fields:

`"enabled"`: *configuration expression<boolean>, optional*

Whether tamper-evident logging is enabled.

Default: false

Tamper-evident logging depends on a specially prepared keystore. For details, see "Preparing a Keystore for Tamper-Evident Logs".

`"filename"`: *configuration expression<string>, required*

File system path to the keystore containing the private key for tamper-evident logging.

The keystore must be a keystore of type `JCEKS`. For details, see "Preparing a Keystore for Tamper-Evident Logs".

`"password"`: *configuration expression<string>, required*

The password for the keystore for tamper-evident logging.

This password is used for the keystore and for private keys. For details, see "Preparing a Keystore for Tamper-Evident Logs".

`"signatureInterval"`: *configuration expression<duration>, required*

The time interval after which to insert a signature in the CSV file. This duration must not be zero, and must not be unlimited.

For information about supported formats for `duration`, see duration.

`"fileRetention"`: *object, optional*

File retention settings for CSV log files.

The file retention object has the following fields:

`"maxDiskSpaceToUse"`: *configuration expression<number>, optional*

The maximum disk space in bytes the audit logs can occupy. A setting of 0 or less indicates that the policy is disabled.

Default: 0

`"maxNumberOfHistoryFiles"`: *configuration expression<number>, optional*

The maximum number of historical log files to retain. A setting of -1 disables pruning of old history files.

Default: 0

`"minFreeSpaceRequired"`: *configuration expression<number>, optional*

The minimum free space in bytes that the system must contain for logs to be written. A setting of 0 or less indicates that the policy is disabled.

Default: 0

"fileRotation": *object, optional*

File rotation settings for log files.

The file rotation object has the following fields:

"rotationEnabled": *configuration expression<boolean>, optional*

Whether file rotation is enabled for log files.

Default: false.

"maxFileSize": *configuration expression<number>, optional*

The maximum file size of an audit log file in bytes. A setting of 0 or less indicates that the policy is disabled.

Default: 0.

"rotationFilePrefix": *configuration expression<string>, optional*

The prefix to add to a log file on rotation.

This has an effect when time-based file rotation is enabled.

"rotationFileSuffix": *configuration expression<string>, optional*

The suffix to add to a log file on rotation, possibly expressed in SimpleDateFormat.

This has an effect when time-based file rotation is enabled.

Default: `-yyyy.MM.dd-HH.mm.ss`, where *yyyy* characters are replaced with the year, *MM* characters are replaced with the month, *dd* characters are replaced with the day, *HH* characters are replaced with the hour (00-23), *mm* characters are replaced with the minute (00-60), and *ss* characters are replaced with the second (00-60).

"rotationInterval": *configuration expression<duration>, optional*

The time interval after which to rotate log files. This duration must not be zero.

This has the effect of enabling time-based file rotation.

For information about supported formats for `duration`, see duration.

"rotationTimes": *array of configuration expression<duration>, optional*

The durations, counting from midnight, after which to rotate files.

The following example schedules rotation six and twelve hours after midnight:

```
"rotationTimes": [ "6 hours", "12 hours" ]
```

This has the effect of enabling time-based file rotation.

For information about supported formats for `duration`, see duration.

`"rotationRetentionCheckInterval"`: *configuration expression<duration>, optional*

The time interval after which to check file rotation and retention policies for updates.

Default: 5 seconds

For information about supported formats for `duration`, see duration.

5.3.4. Preparing a Keystore for Tamper-Evident Logs

Tamper-evident logging depends on a public key/private key pair and on a secret key that are stored together in a JCEKS keystore. Follow these steps to prepare the keystore:

1. Generate a key pair in the keystore.

   The CSV event handler expects a JCEKS-type keystore with a key alias of `Signature` for the signing key, where the key is generated with the `RSA` key algorithm and the `SHA256withRSA` signature algorithm:

   ```
   $ keytool \
    -genkeypair \
    -keyalg RSA \
    -sigalg SHA256withRSA \
    -alias "signature" \
    -dname "CN=microgateway.example.com,O=Example Corp,C=FR" \
    -keystore /path/to/audit-keystore \
    -storetype JCEKS \
    -storepass password \
    -keypass password
   ```

   Note

   Because KeyStore converts all characters in its key aliases to lower case, use only lowercase in alias definitions of a KeyStore.

2. Generate a secret key in the keystore.

   The CSV event handler expects a JCEKS-type keystore with a key alias of `Password` for the symmetric key, where the key is generated with the `HmacSHA256` key algorithm and 256-bit key size:

   ```
   $ keytool \
    -genseckey \
    -keyalg HmacSHA256 \
    -keysize 256 \
    -alias "password" \
    -keystore /path/to/audit-keystore \
    -storetype JCEKS \
    -storepass password \
    -keypass password
   ```

3. Verify the content of the keystore:

```
$ keytool \
 -list \
 -keystore /path/to/audit-keystore \
 -storetype JCEKS \
 -storepass password

Keystore type: JCEKS
Keystore provider: SunJCE

Your keystore contains 2 entries

signature, Nov 27, 2015, PrivateKeyEntry,
Certificate fingerprint (SHA1): 4D:CF:CC:29:...:8B:6E:68:D1
password, Nov 27, 2015, SecretKeyEntry,
```

5.3.5. Example

The following example configures a CSV audit event handler to write a log file, `/path/to/audit/logs/access.csv`, that is signed every 10 seconds to make it tamper-evident:

```
{
  "name": "csv",
  "topics": [
    "access"
  ],
  "logDirectory": "/path/to/audit/logs/",
  "security": {
    "enabled": "true",
    "filename": "/path/to/audit-keystore",
    "password": "password",
    "signatureInterval": "10 seconds"
  }
}
```

5.3.6. More Information

org.forgerock.audit.handlers.csv.CsvAuditEventHandler

Chapter 5.4

ElasticsearchAuditEventHandler

5.4.1. Description

An audit event handler that responds to events by logging messages in the Elasticsearch search and analytics engine. For information about downloading and installing Elasticsearch, see the Elasticsearch *Getting Started* document.

5.4.2. Usage

Configure the ElasticsearchAuditEventHandler within an "*AuditService*":

```
{
  "type": "AuditService",
  "config": {
    "config": {},
    "eventHandlers": [{
      "class": "org.forgerock.audit.handlers.elasticsearch.ElasticsearchAuditEventHandler",
      "config": {
        "name": configuration expression<string>,
        "topics": [ configuration expression<string>, ... ],
        "connection": {
          "host": configuration expression<string>,
          "port": configuration expression<number>,
          "useSSL": configuration expression<boolean>,
          "username": configuration expression<string>,
          "password": configuration expression<string>
        },
        "indexMapping": {
          "indexName": configuration expression<string>
        },
        "buffering": {
          "enabled": configuration expression<boolean>,
          "writeInterval": configuration expression<duration>,
          "maxSize": configuration expression<number>,
          "maxBatchedEvents": configuration expression<number>
        }
      }
    }
  }
}
```

The ElasticsearchAuditEventHandler relays audit events to Elasticsearch through the HTTP protocol, using a handler defined in a heap. The handler can be of any kind of handler, from a

simple ClientHandler to a complex Chain, composed of multiple filters and a final handler or ScriptableHandler.

Microgateway searches first for a handler named `ElasticsearchClientHandler`. If not found, Microgateway searches for a client handler named `AuditClientHandler`. If not found, Microgateway uses the route's default client handler, named `ClientHandler`.

The following example configures a ClientHandler named `ElasticsearchClientHandler`:

```
{
  "name": "ElasticsearchClientHandler",
  "type": ClientHandler,
  "config": {}
}
```

The following example configures a ScriptableHandler named `AuditClientHandler`:

```
{
  "name": "AuditClientHandler",
  "type": ScriptableHandler,
  "config": {}
 }
```

5.4.3. Properties

The `"config"` object has the following properties:

`"name"`: *configuration expression<string>, required*

The name of the event handler.

`"topics"`: *array of configuration expression<string>, required*

An array of one or more topics that this event handler intercepts. Microgateway can record the following audit event topics:

- `access`: Log access audit events. Access audit events occur at the system boundary, and include the arrival of the initial request and departure of the final response.

  To record `access` audit events, configure AuditService inline in a route, or in the heap.

- *customTopic*: Log custom audit events. To create a topic for a custom audit event, include a JSON schema for the topic in your Microgateway configuration.

  To record custom audit events, configure AuditService in the heap, and refer to it from the route or subroutes.

`"connection"`: *object, optional*

Connection settings for sending messages to Elasticsearch. If this object is not configured, it takes default values for its fields. This object has the following fields:

`"host"`: *configuration expression<string>, optional*

Hostname or IP address of Elasticsearch.

Default: `localhost`

`"port"`: *configuration expression<number>, optional*

The port used by Elasticsearch. The value must be between 0 and 65535.

Default: `9200`

`"useSSL"`: *configuration expression<boolean>, optional*

Setting to use or not use SSL/TLS to connect to Elasticsearch.

Default: `false`

`"username"`: *configuration expression<string>, optional*

Username when basic authentication is enabled through Elasticsearch Shield.

`"password"`: *configuration expression<string>, optional*

Password when basic authentication is enabled through Elasticsearch Shield.

`"indexMapping"`: *object, optional*

Defines how an audit event and its fields are stored and indexed.

`"indexName"`: *configuration expression<string>, optional*

The index name. Set this parameter if the default name `audit` conflicts with an existing Elasticsearch index.

Default: `audit`.

`"buffering"`: *object, optional*

Settings for buffering events and batch writes.

`"enabled"`: *configuration expression<boolean>, optional*

Setting to use or not use log buffering.

Default: false.

`"writeInterval"`: *configuration expression<duration>*

The interval at which to send buffered event messages to Elasticsearch. If buffering is enabled, this interval must be greater than 0.

Default: 1 second

For information about supported formats for `duration`, see duration.

`"maxBatchedEvents"`: *configuration expression<number>, optional*

The maximum number of event messages in a batch write to Elasticsearch for each `writeInterval`.

Default: 500

`"maxSize"`: *configuration expression<number>, optional*

The maximum number of event messages in the queue of buffered event messages.

Default: 10000

5.4.4. More Information

org.forgerock.audit.handlers.elasticsearch.ElasticsearchAuditEventHandler

Chapter 5.5

JdbcAuditEventHandler

5.5.1. Description

An audit event handler that responds to events by logging messages to an appropriately configured relational database table.

Declare the configuration in an audit service, as described in "*AuditService*".

5.5.2. Usage

```
{
  "class": "org.forgerock.audit.handlers.jdbc.JdbcAuditEventHandler",
  "config": {
    "name": configuration expression<string>,
    "topics": [ configuration expression<string>, ... ],
    "databaseType": configuration expression<string>,
    "enabled": configuration expression<boolean>,
    "buffering": {
      "enabled": configuration expression<boolean>,
      "writeInterval": configuration expression<duration>,
      "autoFlush": configuration expression<boolean>,
      "maxBatchedEvents": configuration expression<number>,
      "maxSize": configuration expression<number>,
      "writerThreads": configuration expression<number>
    },
    "connectionPool": {
      "driverClassName": configuration expression<string>,
      "dataSourceClassName": configuration expression<string>,
      "jdbcUrl": configuration expression<string>,
      "username": configuration expression<string>,
      "password": configuration expression<string>,
      "autoCommit": configuration expression<boolean>,
      "connectionTimeout": configuration expression<number>,
      "idleTimeout": configuration expression<number>,
      "maxLifetime": configuration expression<number>,
      "minIdle": configuration expression<number>,
      "maxPoolSize": configuration expression<number>,
      "poolName": configuration expression<string>
    },
    "tableMappings": [
      {
        "event": configuration expression<string>,
         "table": configuration expression<string>,
         "fieldToColumn": configuration expression<map>
      }
```

```
    ]
  }
}
```

The values in this configuration object can use expressions as long as they resolve to the correct types for each field. For details about expressions, see "*Expressions*".

5.5.3. Configuration

The `"config"` object has the following properties:

`"name"`: *configuration expression<string>, required*

The name of the event handler.

`"topics"`: *array of configuration expression<string>, required*

An array of one or more topics that this event handler intercepts. Microgateway can record the following audit event topics:

- `access`: Log access audit events. Access audit events occur at the system boundary, and include the arrival of the initial request and departure of the final response.

  To record `access` audit events, configure AuditService inline in a route, or in the heap.

- *customTopic*: Log custom audit events. To create a topic for a custom audit event, include a JSON schema for the topic in your Microgateway configuration.

  To record custom audit events, configure AuditService in the heap, and refer to it from the route or subroutes.

`"databaseType"`: *configuration expression<string>, required*

The database type name.

Built-in support is provided for `oracle`, `mysql`, and `h2`. Unrecognized database types rely on a GenericDatabaseStatementProvider.

`"enabled"`: *configuration expression<boolean>, optional*

Whether this event handler is active.

Default: true.

`"buffering"`: *object, optional*

Buffering settings for sending messages to the database. The default is for messages to be written to the log file for each event.

The buffering object has the following fields:

`"enabled"`: *configuration expression<boolean>, optional*

Whether log buffering is enabled.

Default: false.

`"writeInterval"`: *configuration expression<duration>, required*

The interval at which to send buffered event messages to the database.

This interval must be greater than 0 if buffering is enabled.

For information about supported formats for `duration`, see duration.

`"autoFlush"`: *configuration expression<boolean>, optional*

Whether the events are automatically flushed after being written.

Default: true.

`"maxBatchedEvents"`: *configuration expression<number>, optional*

The maximum number of event messages batched into a PreparedStatement.

Default: 100.

`"maxSize"`: : *configuration expression<number>, optional*

The maximum size of the queue of buffered event messages.

Default: 5000.

`"writerThreads"`: *configuration expression<number>, optional*

The number of threads to write buffered event messages to the database.

Default: 1.

`"connectionPool"`: *object, required*

Connection pool settings for sending messages to the database.

The connection pool object has the following fields:

`"driverClassName"`: *configuration expression<string>, optional*

The class name of the driver to use for the JDBC connection. For example, with MySQL Connector/J, the class name is `com.mysql.jdbc.Driver`.

`"dataSourceClassName"`: *configuration expression<string>, optional*

The class name of the data source for the database.

`"jdbcUrl"`: ***configuration expression<string>, required***

The JDBC URL to connect to the database.

`"username"`: ***configuration expression<string>, required***

The username identifier for the database user with access to write the messages.

`"password"`: ***configuration expression<number>, optional***

The password for the database user with access to write the messages.

`"autoCommit"`: ***configuration expression<boolean>, optional***

Whether to commit transactions automatically when writing messages.

Default: true.

`"connectionTimeout"`: ***configuration expression<number>, optional***

The number of milliseconds to wait for a connection from the pool before timing out.

Default: 30000.

`"idleTimeout"`: ***configuration expression<number>, optional***

The number of milliseconds to allow a database connection to remain idle before timing out.

Default: 600000.

`"maxLifetime"`: ***configuration expression<number>, optional***

The number of milliseconds to allow a database connection to remain in the pool.

Default: 1800000.

`"minIdle"`: ***configuration expression<number>, optional***

The minimum number of idle connections in the pool.

Default: 10.

`"maxPoolSize"`: ***configuration expression<number>, optional***

The maximum number of connections in the pool.

Default: 10.

`"poolName"`: ***configuration expression<string>, optional***

The name of the connection pool.

`"tableMappings"`: *array of objects, required*

Table mappings for directing event content to database table columns.

A table mappings object has the following fields:

`"event"`: *configuration expression<string>, required*

The audit event that the table mapping is for.

Set this to `access`.

`"table"`: *configuration expression<string>, required*

The name of the database table that corresponds to the mapping.

`"fieldToColumn"`: *configuration expression<map>, required*

Maps of names of audit event fields to database columns, where the keys and values are both strings.

Audit event fields use JSON pointer notation, and are taken from the JSON schema for the audit event content.

5.5.4. Example

The following example configures a JDBC audit event handler using a local MySQL database, writing to a table named `auditaccess`:

```
{
  "class": "org.forgerock.audit.handlers.jdbc.JdbcAuditEventHandler",
  "config": {
    "databaseType": "mysql",
    "name": "jdbc",
    "topics": [
      "access"
    ],
    "connectionPool": {
      "jdbcUrl": "jdbc:mysql://localhost:3306/audit?allowMultiQueries=true&characterEncoding=utf8",
      "username": "audit",
      "password": "audit"
    },
    "tableMappings": [
      {
        "event": "access",
        "table": "auditaccess",
        "fieldToColumn": {
          "_id": "id",
          "timestamp": "timestamp_",
          "eventName": "eventname",
          "transactionId": "transactionid",
          "userId": "userid",
          "trackingIds": "trackingids",
```

```
          "server/ip": "server_ip",
          "server/port": "server_port",
          "client/host": "client_host",
          "client/ip": "client_ip",
          "client/port": "client_port",
          "request/protocol": "request_protocol",
          "request/operation": "request_operation",
          "request/detail": "request_detail",
          "http/request/secure": "http_request_secure",
          "http/request/method": "http_request_method",
          "http/request/path": "http_request_path",
          "http/request/queryParameters": "http_request_queryparameters",
          "http/request/headers": "http_request_headers",
          "http/request/cookies": "http_request_cookies",
          "http/response/headers": "http_response_headers",
          "response/status": "response_status",
          "response/statusCode": "response_statuscode",
          "response/elapsedTime": "response_elapsedtime",
          "response/elapsedTimeUnits": "response_elapsedtimeunits"
        }
      }
    ]
  }
}
```

Examples including statements to create tables are provided in the JDBC handler library, `forgerock-audit-handler-jdbc-version.jar`, Unpack the library, then find the examples under the `db/` folder.

The JDBC handler library is in the `lib` directory.

5.5.5. More Information

org.forgerock.audit.handlers.jdbc.JdbcAuditEventHandler

Chapter 5.6

JmsAuditEventHandler

5.6.1. Description

The Java Message Service (JMS) is a Java API for sending asynchronous messages between clients. It wraps audit events in JMS messages and publishes them in a JMS broker, which then delivers the messages to the appropriate destination.

The JMS API architecture includes a *JMS provider* and *JMS clients*, and supports the *publish/subscribe* messaging pattern. For more information, see *Basic JMS API Concepts*.

The JMS audit event handler does not support queries. To support queries, also enable a second handler that supports queries.

The ForgeRock JMS audit event handler supports JMS communication, based on the following components:

- JMS message broker, to provide clients with connectivity, message storage, and message delivery functionality.
- JMS messages.
- Destinations, maintained by a message broker. A destination can be a JMS topic, using publish/subscribe to take the ForgeRock JSON for an audit event, wrap it into a JMS TextMessage, and send it to the broker.
- JMS clients, to produce and/or receive JMS messages.

Depending on the configuration, some or all of these components are included in JMS audit log messages.

Important

The example in this section is based on *Apache ActiveMQ*, but you can choose a different JMS message broker.

Make sure that the `.jar` files required by the JMS message broker are available in the Microgateway web container.

Declare the configuration in an audit service, as described in "*AuditService*".

5.6.2. Usage

```
{
  "type": "AuditService",
  "config": {
    "config": {},
    "eventHandlers": [
    {
      "class": "org.forgerock.audit.handlers.jms.JmsAuditEventHandler",
      "config": {
        "name": configuration expression<string>,
        "topics": [ configuration expression<string>, ... ],
        "deliveryMode": configuration expression<string>,
        "sessionMode": configuration expression<string>,
        "jndi": {
          "contextProperties": map,
          "topicName": configuration expression<string>,
          "connectionFactoryName": configuration expression<string>
        }
      }
    }]
  }
}
```

The values in this configuration object can use configuration expressions, as described in "Configuration and Runtime Expressions".

5.6.3. Configuration

For a full list of properties in the `"config"` object, see Configuration Properties for the JMS Audit Event Handler in the *IDM Integrator's Guide*.

`"name"`: *configuration expression<string>, required*

The name of the event handler.

`"topics"`: *array of configuration expression<string>, required*

An array of one or more topics that this event handler intercepts. Microgateway can record the following audit event topics:

- `access`: Log access audit events. Access audit events occur at the system boundary, and include the arrival of the initial request and departure of the final response.

  To record `access` audit events, configure AuditService inline in a route, or in the heap.

- *customTopic*: Log custom audit events. To create a topic for a custom audit event, include a JSON schema for the topic in your Microgateway configuration.

  To record custom audit events, configure AuditService in the heap, and refer to it from the route or subroutes.

"deliveryMode": *configuration expression<string>, required*

Delivery mode for messages from a JMS provider. Set to `PERSISTENT` or `NON_PERSISTENT`.

"sessionMode": *configuration expression<string>, required*

Acknowledgement mode in sessions without transactions. Set to `AUTO`, `CLIENT`, or `DUPS_OK`.

"contextProperties": *map, optional*

Settings with which to populate the initial context.

The following properties are required when ActiveMQ is used as the message broker:

- `java.naming.factory.initial`

  For example, `"org.apache.activemq.jndi.ActiveMQInitialContextFactory"`.

  To substitute a different JNDI message broker, change the JNDI context properties.

- `java.naming.provider.url`

  For example, `"tcp://127.0.0.1:61616"`.

  To configure the message broker on a remote system, substitute the associated IP address.

  To set up SSL, set up keystores and truststores, and change the value of the `java.naming.provider.url` to:

  ```
  ssl://127.0.0.1:61617?
  daemon=true&socket.enabledCipherSuites=SSL_RSA_WITH_RC4_128_SHA,SSL_DH_anon_WITH_3DES_EDE_CBC_SHA
  ```

- `topic.audit`

  For example, `"audit"`.

To use the JMS resources provided by your application server, leave this field empty. The values for `topicName` and `connectionFactoryName` are then JNDI names that depend on the configuration of your application server.

"topicName": *configuration expression<string>, required*

JNDI lookup name for the JMS topic.

For ActiveMQ, this property must be consistent with the value of `topic.audit` in `contextProperties`.

"connectionFactoryName": *configuration expression<string>, required*

JNDI lookup name for the JMS connection factory.

5.6.4. More Information

org.forgerock.audit.handlers.jms.JmsAuditEventHandler

Chapter 5.7

JsonAuditEventHandler

5.7.1. Description

The JSON audit event handler logs events as JSON objects to a set of JSON files. There is one file for each topic defined in `topics`, named with the format *`topic`*`.audit.json`.

The JsonAuditEventHandler is the preferred file-based audit event handler.

Declare the configuration in an audit service, as described in "*AuditService*".

5.7.2. Usage

```
{
  "type": "AuditService",
  "config": {
    "config": {},
    "eventHandlers": [
    {
      "class": "org.forgerock.audit.handlers.json.JsonAuditEventHandler",
      "config": {
        "name": configuration expression<string>,
        "topics": [ configuration expression<string>, ... ],
        "logDirectory": configuration expression<string>,
        "elasticsearchCompatible": configuration expression<boolean>,
        "fileRotation": {
          "rotationEnabled": configuration expression<boolean>,
          "maxFileSize": configuration expression<number>,
          "rotationFilePrefix": configuration expression<string>,
          "rotationFileSuffix": configuration expression<string>,
          "rotationInterval": configuration expression<duration>,
          "rotationTimes": [ configuration expression<duration>, ... ]
        },
        "fileRetention": {
          "maxNumberOfHistoryFiles": configuration expression<number>,
          "maxDiskSpaceToUse": configuration expression<number>,
          "minFreeSpaceRequired": configuration expression<number>,
          "rotationRetentionCheckInterval": configuration expression<duration>
        },
        "buffering": {
          "writeInterval": configuration expression<duration>,
          "maxSize": configuration expression<number>
        }
      }
    }]
  }
```

```
}
```

5.7.3. Configuration

`"name"`: *configuration expression<string>, required*

The event handler name. This property is used only to refer to the event handler, but is not used to name the generated log file.

`"topics"`: *array of configuration expression<string>, required*

An array of one or more topics that this event handler intercepts. Microgateway can record the following audit event topics:

- `access`: Log access audit events. Access audit events occur at the system boundary, and include the arrival of the initial request and departure of the final response.

  To record `access` audit events, configure AuditService inline in a route, or in the heap.

- *customTopic*: Log custom audit events. To create a topic for a custom audit event, include a JSON schema for the topic in your Microgateway configuration.

  To record custom audit events, configure AuditService in the heap, and refer to it from the route or subroutes.

`"logDirectory"`: *configuration expression<string>, required*

The file system directory where log files are written.

`elasticsearchCompatible`: *configuration expression<boolean>, optional*

Set to `true` to enable compatibility with ElasticSearch JSON format. For more information, see the ElasticSearch documentation.

Default: `false`

`"fileRotation"`: *object, optional*

File rotation settings for log files.

The file rotation object has the following fields:

`"rotationEnabled"`: *configuration expression<boolean>, optional*

Whether file rotation is enabled for log files.

Default: false.

`"maxFileSize"`: *configuration expression<number>, optional*

The maximum file size of an audit log file in bytes. A setting of 0 or less indicates that the policy is disabled.

Default: 0.

`"rotationFilePrefix"`: *configuration expression<string>, optional*

The prefix to add to a log file on rotation.

This has an effect when time-based file rotation is enabled.

`"rotationFileSuffix"`: *configuration expression<string>, optional*

The suffix to add to a log file on rotation, possibly expressed in SimpleDateFormat.

This has an effect when time-based file rotation is enabled.

Default: `-yyyy.MM.dd-HH.mm.ss`, where *yyyy* characters are replaced with the year, *MM* characters are replaced with the month, *dd* characters are replaced with the day, *HH* characters are replaced with the hour (00-23), *mm* characters are replaced with the minute (00-60), and *ss* characters are replaced with the second (00-60).

`"rotationInterval"`: *configuration expression<duration>, optional*

The time interval after which to rotate log files. This duration must not be zero.

This has the effect of enabling time-based file rotation.

For information about supported formats for `duration`, see duration.

`"rotationTimes"`: *array of configuration expression<duration>, optional*

The durations, counting from midnight, after which to rotate files.

The following example schedules rotation six and twelve hours after midnight:

```
"rotationTimes": [ "6 hours", "12 hours" ]
```

This has the effect of enabling time-based file rotation.

For information about supported formats for `duration`, see duration.

`"fileRetention"`: *object, optional*

File retention settings for log files.

The file retention object has the following fields:

`"maxNumberOfHistoryFiles"`: *configuration expression<number>, optional*

The maximum number of historical audit files that can be stored. If the number exceeds this maximum, older files are deleted. A value of `-1` disables purging of old log files.

Default: 0.

`"maxDiskSpaceToUse"`: *configuration expression<number>, optional*

The maximum disk space in bytes that can be used for audit files. If the audit files use more than this space, older files are deleted. A negative or zero value indicates that this policy is disabled, and historical audit files can use unlimited disk space.

`"minFreeSpaceRequired"`: *configuration expression<string>, optional*

The minimum free disk space in bytes required on the system that houses the audit files. If the free space drops below this minimum, older files are deleted. A negative or zero value indicates that this policy is disabled, and no minimum space requirements apply.

`"rotationRetentionCheckInterval"`: *configuration expression<string>, optional*

Interval at which to periodically check file rotation and retention policies. The interval must be a duration, for example, 5 seconds, 5 minutes, or 5 hours.

`"buffering"`: *object, optional*

Settings for buffering events and batch writes.

`"writeInterval"`: *configuration expression<duration>, optional*

The interval at which to send buffered event messages. If buffering is enabled, this interval must be greater than 0.

Default: 1 second

For information about supported formats for `duration`, see duration.

`"maxSize"`: *configuration expression<number>, optional*

The maximum number of event messages in the queue of buffered event messages.

Default: 10000

5.7.4. More Information

org.forgerock.audit.handlers.json.JsonAuditEventHandler

Chapter 5.8

JsonStdoutAuditEventHandler

5.8.1. Description

Logs events to JSON standard output (stdout).

Declare the configuration in an audit service, as described in "*AuditService*".

5.8.2. Usage

```
{
  "type": "AuditService",
  "config": {
    "config": {},
    "eventHandlers": [
    {
      "class": "org.forgerock.audit.handlers.json.stdout.JsonStdoutAuditEventHandler",
      "config": {
        "name": configuration expression<string>,
        "topics": [ configuration expression<string>, ... ],
        "elasticsearchCompatible": configuration expression<boolean>
      }
    }
  }
}
```

5.8.3. Configuration

`"name"`: *configuration expression<string>, required*

The name of the event handler.

`"topics"`: *array of configuration expression<string>, required*

An array of one or more topics that this event handler intercepts. Microgateway can record the following audit event topics:

- `access`: Log access audit events. Access audit events occur at the system boundary, and include the arrival of the initial request and departure of the final response.

  To record `access` audit events, configure AuditService inline in a route, or in the heap.

- *customTopic*: Log custom audit events. To create a topic for a custom audit event, include a JSON schema for the topic in your Microgateway configuration.

  To record custom audit events, configure AuditService in the heap, and refer to it from the route or subroutes.

`elasticsearchCompatible`: *configuration expression<boolean>, optional*

Set to `true` to enable compatibility with ElasticSearch JSON format. For more information, see the ElasticSearch documentation.

Default: `false`

5.8.4. More Information

org.forgerock.audit.handlers.json.stdout.JsonStdoutAuditEventHandler

Chapter 5.9

SyslogAuditEventHandler

5.9.1. Description

An audit event handler that responds to events by logging messages to the UNIX system log as governed by RFC 5424, *The Syslog Protocol*.

Declare the configuration in an audit service, as described in "*AuditService*".

5.9.2. Usage

```
{
  "class": "org.forgerock.audit.handlers.syslog.SyslogAuditEventHandler",
  "config": {
    "name": configuration expression<string>,
    "topics": [ configuration expression<string>, ... ],
    "protocol": configuration expression<string>,
    "host": configuration expression<string>,
    "port": configuration expression<number>,
    "connectTimeout": configuration expression<number>,
    "facility": configuration expression<string>,
    "buffering": {
        "enabled": configuration expression<boolean>,
        "maxSize": configuration expression<number>
    },
    "severityFieldMappings": [
      {
        "topic": configuration expression<string>,
        "field": configuration expression<string>,
        "valueMappings": {
          "field-value": object
        }
      }
    ]
  }
}
```

The values in this configuration object can use expressions as long as they resolve to the correct types for each field. For details about expressions, see "*Expressions*".

5.9.3. Configuration

The `"config"` object has the following properties:

`"name"`: *configuration expression<string>, required*

The name of the event handler.

`"topics"`: *array of configuration expression<string>, required*

An array of one or more topics that this event handler intercepts. Microgateway can record the following audit event topics:

- `access`: Log access audit events. Access audit events occur at the system boundary, and include the arrival of the initial request and departure of the final response.

  To record `access` audit events, configure AuditService inline in a route, or in the heap.

- *customTopic*: Log custom audit events. To create a topic for a custom audit event, include a JSON schema for the topic in your Microgateway configuration.

  To record custom audit events, configure AuditService in the heap, and refer to it from the route or subroutes.

`"protocol"`: *configuration expression<string>, required*

The transport protocol used to send event messages to the Syslog daemon.

Set this to `TCP` for Transmission Control Protocol, or to `UDP` for User Datagram Protocol.

`"host"`: *configuration expression<string>, required*

The hostname of the Syslog daemon to which to send event messages. The hostname must resolve to an IP address.

`"port"`: *configuration expression<number>, required*

The port of the Syslog daemon to which to send event messages.

The value must be between 0 and 65535.

`"connectTimeout"`: *configuration expression<number>, required when using TCP*

The number of milliseconds to wait for a connection before timing out.

`"facility"`: *configuration expression<string>, required*

The Syslog facility to use for event messages.

Set this to one of the following values:

`kern`

Kernel messages

`user`

User-level messages

`mail`

Mail system

`daemon`

System daemons

`auth`

Security/authorization messages

`syslog`

Messages generated internally by `syslogd`

`lpr`

Line printer subsystem

`news`

Network news subsystem

`uucp`

UUCP subsystem

`cron`

Clock daemon

`authpriv`

Security/authorization messages

`ftp`

FTP daemon

`ntp`

NTP subsystem

`logaudit`

Log audit

`logalert`

Log alert

`clockd`

Clock daemon

`local0`

Local use 0

`local1`

Local use 1

`local2`

Local use 2

`local3`

Local use 3

`local4`

Local use 4

`local5`

Local use 5

`local6`

Local use 6

`local7`

Local use 7

`"buffering"`: *object, optional*

Buffering settings for writing to the system log facility. The default is for messages to be written to the log for each event.

The buffering object has the following fields:

`"enabled"`: *configuration expression<boolean>, optional*

Whether log buffering is enabled.

Default: false.

`"maxSize"`: ***configuration expression<number>, optional***

The maximum number of buffered event messages.

Default: 5000.

`"severityFieldMappings"`: ***object, optional***

Severity field mappings set the correspondence between audit event fields and Syslog severity values.

The severity field mappings object has the following fields:

`"topic"`: ***configuration expression<string>, required***

The audit event topic to which the mapping applies.

Set this to a value configured in `topics`.

`"field"`: ***configuration expression<string>, required***

The audit event field to which the mapping applies.

Audit event fields use JSON pointer notation, and are taken from the JSON schema for the audit event content.

`"valueMappings"`: ***object, required***

The map of audit event values to Syslog severities, where both the keys and the values are strings.

Syslog severities are one of the following values:

`emergency`

System is unusable.

`alert`

Action must be taken immediately.

`critical`

Critical conditions.

`error`

Error conditions.

`warning`

Warning conditions.

notice

Normal but significant condition.

informational

Informational messages.

debug

Debug-level messages.

5.9.4. Example

The following example configures a Syslog audit event handler that writes to the system log daemon on `syslogd.example.com`, port `6514` over TCP with a timeout of 30 seconds. The facility is the first one for local use, and response status is mapped to Syslog informational messages:

```
{
  "class": "org.forgerock.audit.handlers.syslog.SyslogAuditEventHandler",
  "config": {
    "name": "MySyslogAuditEventHandler",
    "topics": ["access"],
    "protocol": "TCP",
    "host": "https://syslogd.example.com",
    "port": 6514,
    "connectTimeout": 30000,
    "facility": "local0",
    "severityFieldMappings": [
      {
        "topic": "access",
        "field": "response/status",
        "valueMappings": {
          "FAILED": "INFORMATIONAL",
          "SUCCESSFUL": "INFORMATIONAL"
        }
      }
    ]
  }
}
```

5.9.5. More Information

org.forgerock.audit.handlers.syslog.SyslogAuditEventHandler

Chapter 5.10
SplunkAuditEventHandler

5.10.1. Description

The Splunk audit event handler logs Microgateway events to a Splunk system.

5.10.2. Usage

Configure the SplunkAuditEventHandler within an "*AuditService*":

```
{
  "type": "AuditService",
  "config": {
    "config": {},
    "eventHandlers": [{
      "class": "org.forgerock.audit.handlers.splunk.SplunkAuditEventHandler",
      "config": {
        "name": configuration expression<string>,
        "topics": [ configuration expression<string>, ... ],
        "enabled": configuration expression<boolean>,
        "connection": {
          "useSSL": configuration expression<boolean>,
          "host": configuration expression<string>,
          "port": configuration expression<number>
        },
        "buffering": {
          "maxSize": configuration expression<number>,
          "writeInterval": configuration expression<duration>,
          "maxBatchedEvents": configuration expression<number>
        },
        "authzToken": configuration expression<string>
      }
    }]
  }
}
```

The SplunkAuditEventHandler relays audit events to Elasticsearch through the HTTP protocol, using a handler defined in a heap. The handler can be of any kind of handler, from a simple ClientHandler to a complex Chain, composed of multiple filters and a final handler or ScriptableHandler.

Microgateway searches first for a handler named `SplunkAuditEventHandler`. If not found, Microgateway searches for a client handler named `AuditClientHandler`. If not found, Microgateway uses the route's default client handler, named `ClientHandler`.

The following example configures a ClientHandler named `SplunkClientHandler`:

```
{
  "name": "SplunkClientHandler",
  "type": ClientHandler,
  "config": {}
}
```

The following example configures a ScriptableHandler named `AuditClientHandler`:

```
{

  "name": "AuditClientHandler",
  "type": ScriptableHandler,
  "config": {}
}
```

5.10.3. Configuration

`"name"`: *configuration expression<string>, required*

The name of the event handler.

`"topics"`: *array of configuration expression<string>, required*

An array of one or more topics that this event handler intercepts. Microgateway can record the following audit event topics:

- `access`: Log access audit events. Access audit events occur at the system boundary, and include the arrival of the initial request and departure of the final response.

  To record `access` audit events, configure AuditService inline in a route, or in the heap.

- *customTopic*: Log custom audit events. To create a topic for a custom audit event, include a JSON schema for the topic in your Microgateway configuration.

  To record custom audit events, configure AuditService in the heap, and refer to it from the route or subroutes.

`"enabled"`: *configuration expression<boolean>, required*

Specifies whether this audit event handler is enabled.

`"connection"`: *object, optional*

Connection settings for sending messages to the Splunk system. If this object is not configured, it takes default values for its fields. This object has the following fields:

`"useSSL"`: *configuration expression<boolean>, optional*

Specifies whether Microgateway should connect to the audit event handler instance over SSL.

Default: `false`

`"host"`: ***configuration expression<string>, optional***

Hostname or IP address of the Splunk system.

Default: `localhost`

`"port"`: ***configuration expression<number>, optional***

The dedicated Splunk port for HTTP input.

Before you install Splunk, make sure that this port is free. Otherwise, change the port number in Splunk and in the Microgateway routes that use Splunk.

Default: `8088`

`"buffering"`: ***object, optional***

Settings for buffering events and batch writes. If this object is not configured, it takes default values for its fields. This object has the following fields:

`"maxSize"`: ***configuration expression<number>, optional***

The maximum number of event messages in the queue of buffered event messages.

Default: 10000

`"maxBatchedEvents"`: ***configuration expression<number>, optional***

The maximum number of event messages in a batch write to this event handler for each `writeInterval`.

Default: 500

`"writeInterval"`: ***configuration expression<duration>, optional***

The delay after which the writer thread is scheduled to run after encountering an empty event buffer.

Default: 100 ms (units of 'ms' or 's' are recommended)

For information about supported formats for `duration`, see duration.

`"authzToken"`: ***configuration expression<string>, required***

The authorization token associated with the configured HTTP event collector.

5.10.4. More Information

org.forgerock.audit.handlers.splunk.SplunkAuditEventHandler

Part 6
Monitoring

This section describes monitoring endpoints exposed by Microgateway, and the metrics available at the endpoints.

Chapter 6.1
Monitoring Types

This section describes the data types used in monitoring:

Counter

Cumulative metric for a numerical value that only increases.

Gauge

Metric for a numerical value that can increase or decrease.

Summary

Metric that samples observations, providing a count of observations, sum total of observed amounts, average rate of events, and moving average rates across a sliding time window.

The Prometheus view does not provide time-based statistics, as rates can be calculated from the time-series data. Instead, the Prometheus view includes summary metrics whose names have the following suffixes or labels:

- `_count`: number of events recorded
- `_total`: sum of the amounts of events recorded
- `{quantile="0.5"}`: 50% at or below this value
- `{quantile="0.75"}`: 75% at or below this value
- `{quantile="0.95"}`: 95% at or below this value
- `{quantile="0.98"}`: 98% at or below this value
- `{quantile="0.99"}`: 99% at or below this value
- `{quantile="0.999"}`: 99.9% at or below this value

Timer

Metric combining time-series summary statistics.

Common REST views show summaries as JSON objects. JSON summaries have the following fields:

```
{
 "count": number,           // events recorded for this metric
 "max": number,             // maximum duration recorded
 "mean": number,            // total/count, or 0 if count is 0
 "min": number,             // minimum duration recorded for this metric
 "mean_rate": number,       // average rate
 "p50": number,             // 50% at or below this value
 "p75": number,             // 75% at or below this value
 "p95": number,             // 95% at or below this value
 "p98": number,             // 98% at or below this value
 "p99": number,             // 99% at or below this value
 "p999": number,            // 99.9% at or below this value
 "stddev": number,          // standard deviation of recorded durations
 "m15_rate": number,        // fifteen-minute average rate
 "m5_rate": number,         // five-minute average rate
 "m1_rate": number,         // one-minute average rate
 "duration_units": string,  // time unit used in durations
 "rate_units": string,      // event count unit and time unit used in rate
 "total": number            // sum of the durations of events recorded
}
```

Chapter 6.2
Monitoring Endpoints

This section describes the monitoring endpoints exposed in Microgateway.

6.2.1. Prometheus Scrape Endpoint

All ForgeRock products automatically expose a monitoring endpoint where Prometheus can scrape metrics, in a standard Prometheus format. For information about configuring Prometheus to scrape metrics, see the Prometheus web site.

When Microgateway is set up as described in the documentation, the endpoint is http://microgateway.example.com:8080/openig/metrics/prometheus.

For information about available metrics, see:

- "Route Metrics at the Prometheus Scrape Endpoint"
- "Router Metrics at the Prometheus Scrape Endpoint"
- "Timer Metrics At the Prometheus Scrape Endpoint"

6.2.2. Common REST Monitoring Endpoint

All ForgeRock products expose a monitoring endpoint where metrics are exposed as a JSON format monitoring resource.

When Microgateway is set up as described in the documentation, the endpoint is http://microgateway.example.com:8080/openig/metrics/api?_queryFilter=true.

For information about available metrics, see:

- "Route Metrics at the Common REST Monitoring Endpoint"
- "Router Metrics at the Common REST Monitoring Endpoint"
- "Timer Metrics At the Common REST Monitoring Endpoint"

Part 7

Throttling Filters and Policies

To protect applications from being overused by clients, use a throttling filter to limit how many requests clients can make in a defined time.

Chapter 7.1
ThrottlingFilter

7.1.1. Description

Limits the rate that requests pass through a filter. The maximum number of requests that a client is allowed to make in a defined time is called the *throttling rate*.

When the throttling rate is reached, Microgateway issues an HTTP status code 429 `Too Many Requests` and a `Retry-After` header, whose value is rounded up to the number of seconds to wait before trying the request again.

```
GET http://microgateway.example.com:8080/home/throttle-scriptable HTTP/1.1

 . . .

HTTP/1.1 429 Too Many Requests
Retry-After: 10
```

7.1.2. Usage

```
{
    "name": string,
    "type": "ThrottlingFilter",
    "config": {
        "requestGroupingPolicy": runtime expression<string>,
        "throttlingRatePolicy": reference or inline declaration,  //Use either "throttlingRatePolicy"
        "rate": {                                                 //or "rate", but not both.
            "numberOfRequests": integer,
            "duration": duration string
        },
        "cleaningInterval": duration string,
        "executor": executor
    }
}
```

7.1.3. Properties

`"requestGroupingPolicy"`: *runtime expression<string>, optional*

An expression to identify the partition to use for the request. In many cases the partition identifies an individual client that sends requests, but it can also identify a group that sends requests. The expression can evaluate to the client IP address or user ID, or an OpenID Connect subject/issuer.

The value for this expression must not be null.

Default: Empty string; all requests share the same partition

See also "*Expressions*".

`"throttlingRatePolicy"`: *reference or inline declaration, required if `"rate"` is not used*

A reference to or inline declaration of a policy to apply for throttling rate. The following policies can be used:

- "*MappedThrottlingPolicy*"
- "*ScriptableThrottlingPolicy*"
- "*DefaultRateThrottlingPolicy*"

This value for this parameter must not be null.

`"rate"`: *rate object, required if `"throttlingRatePolicy"` is not used*

The throttling rate to apply to requests. The rate is calculated as the number of requests divided by the duration:

`"numberOfRequests"`: *integer, required*

The number of requests allowed through the filter in the time specified by `"duration"`.

`"duration"`: *duration string, required*

A time interval during which the number of requests passing through the filter is counted.

For information about supported formats for `duration`, see duration.

`"cleaningInterval"`: *duration, optional*

The time to wait before cleaning outdated partitions. The value must be more than zero but not more than one day.

`"executor"`: *executor, optional*

An executor service to schedule the execution of tasks, such as the clean up of partitions that are no longer used.

Default: `ScheduledExecutorService`

See also "*ScheduledExecutorService*".

7.1.4. More Information

org.forgerock.openig.filter.throttling.ThrottlingFilterHeaplet

Chapter 7.2

MappedThrottlingPolicy

7.2.1. Description

Maps different throttling rates to different groups of requests, according to the evaluation of `throttlingRateMapper`.

7.2.2. Usage

```
{
    "type": "ThrottlingFilter",
    "config": {
        "requestGroupingPolicy": expression,
        "throttlingRatePolicy": {
            "type": "MappedThrottlingPolicy",
            "config": {
                "throttlingRateMapper": runtime expression<string>,
                "throttlingRatesMapping": {
                    "mapping1": {
                        "numberOfRequests": integer,
                        "duration": duration string
                    },
                    "mapping2": {
                        "numberOfRequests": integer,
                        "duration": duration string
                    }
                },
                "defaultRate": {
                    "numberOfRequests": integer,
                    "duration": duration string
                }
            }
        }
    }
}
```

7.2.3. Properties

`"throttlingRateMapper"`: *runtime expression<string>, required*

An expression to categorize requests for mapping to a throttling rate in the `throttlingRatesMapping`.

If this parameter is null or does not match any specified mappings, the default throttling rate is applied.

`"throttlingRatesMapping"`: *object, required*

A map of throttling rate by request group. Requests are categorized into groups by the evaluation of the expression `"throttlingRateMapper"`.

`"mapping1"` and `"mapping2"`: *string, required*

The evaluation of the expression `"throttlingRateMapper"`.

The number of mappings is not limited to two.

`"numberOfRequests"`: *integer, required*

The number of requests allowed through the filter in the time specified by `"duration"`.

`"duration"`: *duration string, required*

A time interval during which the number of requests passing through the filter is counted.

For information about supported formats for `duration`, see duration.

`"defaultRate"`: *object, required*

The default throttling rate to apply if the evaluation of the expression `"throttlingRateMapper"` is null or is not mapped to a throttling rate.

`"numberOfRequests"`: *integer, required*

The number of requests allowed through the filter in the time specified by `"duration"`.

`"duration"`: *duration string, required*

A time interval during which the number of requests passing through the filter is counted.

For information about supported formats for `duration`, see duration.

7.2.4. More Information

org.forgerock.openig.filter.throttling.MappedThrottlingPolicyHeaplet

Chapter 7.3
ScriptableThrottlingPolicy

7.3.1. Description

Uses a script to look up the throttling rates to apply to groups of requests.

The script can store the mapping for the throttling rate in memory, and can use a more complex mapping mechanism than that used in the `MappedThrottlingPolicy`. For example, the script can map the throttling rate for a range of IP addresses. The script can also query an LDAP directory, an external database, or read the mapping from a file.

Scripts must return a Promise<ThrottlingRate, Exception> or a ThrottlingRate.

This section describes the usage of ScriptableThrottlingPolicy, and refers to the following sections of the documentation:

- For information about script properties, available global objects, and automatically imported classes, see "*Scripts*".

7.3.2. Usage

```
{
   "type": "ThrottlingFilter",
   "config": {
       "requestGroupingPolicy": expression,
       "throttlingRatePolicy": {
           "name": string,
           "type": "ScriptableThrottlingPolicy",
           "config": {
               "type": string,
               "file": string,                          // Use either "file"
               "source": string or array of strings,  // or "source", but not both
               "args": object
           }
       }
   }
}
```

7.3.3. Properties

For information about properties for ScriptableThrottlingPolicy, see "*Scripts*".

7.3.4. More Information

org.forgerock.openig.filter.throttling.ScriptableThrottlingPolicy.Heaplet

Chapter 7.4
DefaultRateThrottlingPolicy

7.4.1. Description

Provides a default throttling rate if the delegating throttling policy returns `null`.

7.4.2. Usage

```
{
   "type": "ThrottlingFilter",
   "config": {
       "requestGroupingPolicy": expression,
       "throttlingRatePolicy": {
           "type": "DefaultRateThrottlingPolicy",
           "config": {
               "delegateThrottlingRatePolicy": reference or inline declaration,
               "defaultRate": {
                   "numberOfRequests": integer,
                   "duration": duration string
               }
           }
       }
   }
}
```

7.4.3. Properties

`"delegateThrottlingRatePolicy"`: *reference, required*

The policy to which the default policy delegates the throttling rate. The `DefaultRateThrottlingPolicy` delegates management of throttling to the policy specified by `delegateThrottlingRatePolicy`.

If `delegateThrottlingRatePolicy` returns `null`, the `defaultRate` is used.

For information about policies to use, see "*MappedThrottlingPolicy*" and "*ScriptableThrottlingPolicy*".

`"defaultRate"`: *object, required*

The default throttling rate to apply if the delegating policy returns `null`.

`"numberOfRequests"`: *integer, required*

The number of requests allowed through the filter in the time specified by `"duration"`.

`"duration"`: *duration string, required*

A time interval during which the number of requests passing through the filter is counted.

For information about supported formats for `duration`, see duration.

7.4.4. More Information

org.forgerock.openig.filter.throttling.DefaultRateThrottlingPolicyHeaplet

Part 8

Miscellaneous Heap Objects

Chapter 8.1

AmService

8.1.1. Description

An AmService holds information about the configuration of an instance of AM. It is available to Microgateway filters that communicate with that instance.

8.1.2. Usage

```
{
  "name": string,
   "type": "AmService",
   "config": {
     "agent": object,
     "secretsProvider": SecretsProvider reference,
     "notifications": object,
     "realm": configuration expression<string>,
     "amHandler": handler reference,
     "sessionCache": object,
     "sessionProperties": [ configuration expression<string>, ... ],
     "ssoTokenHeader": configuration expression<string>,
     "url": configuration expression<uri string>,
     "version": configuration expression<string>
   }
}
```

8.1.3. Properties

`"agent"`: *object, required*

The credentials of an Microgateway agent in AM. When the agent is authenticated, the token can be used for tasks such as getting the user's profile, making policy evaluations, and connecting to the AM notification endpoint.

`"username"`: *configuration expression<string>, required*

Agent name.

`"passwordSecretId"`: *configuration expression<secret-id>, required*

The secret ID of the agent password.

For information about supported formats for `secret-id`, see secret-id.

`"secretsProvider"`: *SecretsProvider reference, optional*

The SecretsProvider object to query for the agent password. For more information, see "*SecretsProvider*".

Default: The route's default secret service. For more information, see "*secrets*".

`"realm"`: *configuration expression<string>, optional*

The realm of the Microgateway agent in AM.

Default: `/` (top level realm).

`"amHandler"`: *handler reference, optional*

The handler to use for communicating with AM. In production, use a ClientHandler that is capable of making an HTTPS connection to AM.

Tip

To facilitate auditing, configure this handler with a `ForgeRockClientHandler`, which sends a ForgeRock Common Audit transaction ID when it communicates with protected applications.

Alternatively, configure this handler as a chain containing a `TransactionIdOutboundFilter`, as in the following configuration:

```
"amHandler": {
  "type": "Chain",
  "config": {
    "handler": "MySecureClientHandler",
    "filters": [ "TransactionIdOutboundFilter" ]
  }
}
```

Default: `ForgeRockClientHandler`

See also "Handlers", "*ClientHandler*".

`"notifications"`: *object, optional*

Configure WebSocket notification service.

```
{
  "notifications": {
     "enabled": boolean,
     "reconnectDelay": object,
     "tls": TlsOptions reference
  }
}
```

`enabled`: *boolean, optional*

Enable or disable WebSocket notifications. Set to `false` to disable WebSocket notifications.

Default: `true`

`reconnectDelay`: *duration, optional*

The time between attempts to re-establish a lost WebSocket connection.

When a WebSocket connection is lost, Microgateway waits for this delay and then attempts to re-establish the connection. If subsequent attempts fail, Microgateway waits and tries again an unlimited number of times.

Default: `5 seconds`

For information about supported formats for `duration`, see duration.

`tls`: *TlsOptions reference, optional*

Configure options for WebSocket connections to TLS-protected endpoints. Define a TlsOptions object inline or in the heap.

For more information, see "*TlsOptions*".

Default: Connections for WebSocket connections to TLS-protected endpoints not configured.

`"url"`: *configuration expression<uri string>, required*

The URI of the AM server instance, for example: `https://openam.example.com/openam`.

`"sessionCache"`: *object, optional*

Supported in AM 5.5 when the user manually whitelists the `AmCtxId` session property, and in AM 6 and later versions without additional configuration in AM.

Enable and configure caching of session information from AM, based on *Caffeine*. For more information, see the GitHub entry, *Caffeine* .

When `sessionCache` is enabled, Microgateway can reuse session token information without repeatedly asking AM to verify the token. Each instance of AmService has an independent cache content. The cache is not shared with other AmService instances, either in the same or different routes, and is not distributed among clustered Microgateway instances.

When `sessionCache` is disabled, Microgateway must ask AM to verify the token for each request.

Microgateway evicts session info entries from the cache for the following reasons:

- AM cache timeout, based the whichever of the following events occur first:
  - `maxSessionExpirationTime` from SessionInfo

- `maxSessionTimeout` from the AmService configuration

When Microgateway evicts session info entries from the cache, the next time the token is presented, Microgateway must ask AM to verify the token.

- **If Websocket notifications are enabled**, AM session revocation, for example, when a user logs out of AM.

  When Websocket notifications are enabled, Microgateway evicts a cached token almost as soon as it is revoked on AM, and in this way stays synchronized with AM. Subsequent requests to Microgateway that present the revoked token are rejected.

  When Websocket notifications are disabled, the token remains in the cache after it is revoked on AM. Subsequent requests to Microgateway that present the revoked token are considered as valid, and can cause incorrect authentication and authorization decisions until its natural eviction from the cache.

```
{
  "sessionCache": {
    "enabled": configuration expression<boolean>,
    "executor": executor service reference,
    "maximumSize": configuration expression<number>,
    "maximumTimeToCache": configuration expression<duration>,
    "onNotificationDisconnection": configuration expression<enumeration>
  }
}
```

`enabled`: *configuration expression<boolean>, optional*

Enable caching.

Default: `false`

`executor`: *executor service reference, optional*

An executor service to schedule the execution of tasks, such as the eviction of entries in the cache.

Default: `ForkJoinPool.commonPool()`

`"maximumSize"`: *configuration expression<number>, optional*

The maximum number of entries the cache can contain.

Default: Unlimited/unbound.

`maximumTimeToCache`: *configuration expression<duration string>, optional*

The maximum duration for which to cache session info. Consider setting this duration to be less than the idle timeout of AM.

If `maximumTimeToCache` is longer than `maxSessionExpirationTime`, `maxSessionExpirationTime` is used.

Default: `maxSessionExpirationTime`, from SessionInfo.

For information about supported formats for `duration`, see duration.

`onNotificationDisconnection`: *configuration expression<enumeration>, optional*

The strategy to manage the cache when the WebSocket notification service is disconnected, and Microgateway receives no notifications for AM events, such as session revocation. If the cache is not cleared it can become outdated, and Microgateway can allow requests on revoked sessions.

Cached entries that expire naturally while the notification service is disconnected are removed from the cache.

Use one of the following values:

- `NEVER_CLEAR`
  - When the notification service is disconnected:
    - Continue to use the existing cache.
    - Deny access for requests that are not cached, but do not update the cache with these requests.
  - When the notification service is reconnected:
    - Continue to use the existing cache.
    - Query AM for incoming requests that are not found in the cache, and update the cache with these requests.
- `CLEAR_ON_DISCONNECT`
  - When the notification service is disconnected:
    - Clear the cache.
    - Deny access to all requests, but do not update the cache with these requests.
  - When the notification service is reconnected:
    - Query AM for all requests that are not found in the cache. (Because the cache was cleared, the cache is empty after reconnection.)
    - Update the cache with these requests.
- `CLEAR_ON_RECONNECT`

- When the notification service is disconnected:
  - Continue to use the existing cache.
  - Deny access for requests that are not cached, but do not update the cache with these requests.
- When the notification service is reconnected:
  - Query AM for all requests that are not found in the cache. (Because the cache was cleared, the cache is empty after reconnection.)
  - Update the cache with these requests.

Default: `CLEAR_ON_DISCONNECT`

`"sessionProperties"`: *array of configuration expression<string>, optional*

Supported with AM 6 and later versions.

The list of user session properties to retrieve from AM by the "*SessionInfoFilter*".

Default: All available session properties are retrieved from AM.

`"ssoTokenHeader"`: *configuration expression<string>, optional*

The name of the header or cookie where this AM server expects to find SSO tokens.

Default: `iPlanetDirectoryPro`

`"version"`: *configuration expression<string>, optional*

The version number of the AM server. Specify `5` or higher.

Default: AM 5.0.

8.1.4. Example

For examples where AmService is used, see the example routes at the end of "*OAuth2ResourceServerFilter*", "*PolicyEnforcementFilter*", "*SingleSignOnFilter*", "*TokenTransformationFilter*", and "*UserProfileFilter*".

8.1.5. More Information

org.forgerock.openig.tools.am.AmService

org.forgerock.openig.tools.session.SessionInfo

Chapter 8.2
ClientRegistration

8.2.1. Description

A ClientRegistration holds information about registration with an OAuth 2.0 authorization server or OpenID Provider.

The configuration includes the client credentials that are used to authenticate to the identity provider. The client credentials can be included directly in the configuration, or retrieved in some other way using an expression, described in "*Expressions*".

8.2.2. Usage

```
{
  "name": string,
  "type": "ClientRegistration",
  "config": {
    "clientId": expression,
    "clientSecretId": configuration expression<secret-id>,
    "issuer": Issuer reference,
    "registrationHandler": Handler reference,
    "scopes": [ expression, ...],
    "secretsProvider": SecretsProvider reference,
    "tokenEndpointAuthMethod": enumeration,
    "tokenEndpointAuthSigningAlg": string,
    "privateKeyJwtSecretId": configuration expression<secret-id>,
    "claims": map or runtime expression<map>,
    "jwtExpirationTimeout": duration
  }
}
```

8.2.3. Properties

The client registration configuration object properties are as follows:

`"name"`: *string, required*

A name for the client registration.

`"clientId"`: *expression, required*

The `client_id` obtained when registering with the authorization server.

See also "*Expressions*".

`"clientSecretId"`: *configuration expression<secret-id>, required if `tokenEndpointAuthMethod` is `client_secret_basic` or `client_secret_post`*

The secret ID of the client secret required to authenticate the client to the authorization server.

For information about supported formats for `secret-id`, see secret-id.

`"issuer"`: *Issuer reference, required*

The provider configuration to use for this client registration.

Provide either the name of a Issuer object defined in the heap, or an inline Issuer configuration object.

See also "*Issuer*".

`"registrationHandler"`: *Handler reference, optional*

Invoke this HTTP client handler to communicate with the authorization server.

Provide either the name of a Handler object defined in the heap, or an inline Handler configuration object.

Usually set this to the name of a ClientHandler configured in the heap, or a chain that ends in a ClientHandler.

Default: Microgateway uses the default ClientHandler.

See also "Handlers", "*ClientHandler*".

`"scopes"`: *array of expressions, optional*

OAuth 2.0 scopes to use with this client registration.

See also "*Expressions*".

`"secretsProvider"`: *SecretsProvider reference, optional*

The SecretsProvider object to query for the client secret. For more information, see "*SecretsProvider*".

Default: The route's default secret service. For more information, see "*secrets*".

`"tokenEndpointAuthMethod"`: *enumeration, optional*

The authentication method with which a client authenticates to the authorization server or OpenID provider at the token endpoint. For information about client authentication methods, see OpenID Client Authentication. The following client authentication methods are allowed:

- `client_secret_basic`: Clients that have received a `client_secret` value from the authorization server authenticate with the authorization server by using the HTTP Basic authentication scheme, as in the following example:

```
POST /oauth2/token HTTP/1.1
Host: as.example.com
Authorization: Basic ....
Content-Type: application/x-www-form-urlencoded

grant_type=authorization_code&
code=...
```

- `client_secret_post`: Clients that have received a `client_secret` value from the authorization server authenticate with the authorization server by including the client credentials in the request body, as in the following example:

```
POST /oauth2/token HTTP/1.1
Host: as.example.com
Content-Type: application/x-www-form-urlencoded

grant_type=authorization_code&
client_id=...&
client_secret=...&
code=...
```

- `private_key_jwt`: Clients send a signed JSON Web Token (JWT) to the authorization server. Microgateway builds and signs the JWT, and prepares the request as in the following example:

```
POST /token HTTP/1.1
Host: as.example.com
Content-Type: application/x-www-form-urlencoded

grant_type=authorization_code&
code=...&
client_id=<clientregistration_id>&
client_assertion_type=urn%3Aietf%3Aparams%3Aoauth%3Aclient-assertion-type%3Ajwt-bearer&
client_assertion=PHNhbWxwOl ... ZT
```

  If the authorization server doesn't support `private_key_jwt`, a dynamic registration falls back on the method returned by the authorization server, for example, `client_secret_basic` or `client_secret_post`.

  If `tokenEndpointAuthSigningAlg` is not configured, the `RS256` signing algorithm is used for `private_key_jwt`.

Consider these points for identity providers:

- Some providers accept more than one authentication method.
- If a provider strictly enforces how the client must authenticate, align the authentication method with the provider.
- If a provider doesn't support the authentication method, the provider sends an HTTP 400 Bad Request response with an `invalid_client` error message, according to RFC 6749 *The OAuth 2.0 Authorization Framework*, section 5.2 .
- If the authentication method is invalid, the provider sends an `IllegalArgumentException`.

Default: `client_secret_basic`

`"tokenEndpointAuthSigningAlg"`: *string, optional*

The JSON Web Algorithm (JWA) used to sign the JWT that is used to authenticate the client at the token endpoint. The property is used when `private_key_jwt` is used for authentication.

Use one of the following algorithms:

- `RS256`: RSA using SHA-256
- `ES256`: ECDSA with SHA-256 and NIST standard P-256 elliptic curve
- `ES384`: ECDSA with SHA-384 and NIST standard P-384 elliptic curve
- `ES512`: ECDSA with SHA-512 and NIST standard P-521 elliptic curve

Default: `RS256`

`"privateKeyJwtSecretId"`: *configuration expression<secret-id>, required when `private_key_jwt` is used for client authentication*

The secret ID of the key that is used to sign the JWT.

For information about supported formats for `secret-id`, see secret-id.

`"claims"`: *map or runtime expression<map>, optional*

When `private_key_jwt` is used for authentication, this property specifies the claims used in the authentication. If this property is a map, the structure must have the format `Map<String, Object>`.

The JWT can contain the following claim value and other optional claims, where claims that are not understood are ignored:

`"aud"`: *string or array of strings, optional*

The URI of the authorization server that is the intended audience of the token.

Default: URL of the authorization server token endpoint

In the following example, the claims include the value `aud`, which is the URI of the authorization server that is the audience of the token:

```
"claims": {
  "aud": "https://myapp.authentication.example.com"
}
```

If this property is an expression, its evaluation must yield an object of type `Map<String, Object>`. In the following example, `overrideAudience` is declared in the properties and then included in an expression in the claims declaration:

```
{
  "properties": {
    "overrideAudience": {
      "aud": "https://myapp.authentication.example.com"
    }
  }
}
```

```
"claims": "${overrideAudience}"
```

`"jwtExpirationTimeout"`: *duration, optional*

When `private_key_jwt` is used for authentication, this property specifies the duration for which the JWT is valid.

Default: 1 minute

For information about supported formats for `duration`, see duration.

8.2.4. Example

The following example shows a client registration for AM. In this example client credentials are replaced with `**********`. In the actual configuration either include the credentials and protect the configuration file or obtain the credentials from the environment in a safe way:

```
{
    "name": "registration",
    "type": "ClientRegistration",
    "config": {
        "clientId": "**********",
        "clientSecretId": "**********",
        "issuer": {
            "type": "Issuer",
            "config": {
                "wellKnownEndpoint": "http://openam.example.com:8088/openam/oauth2/.well-known/openid-
configuration"
            }
        },
        "secretsProvider": "mySystemAndEnvSecretStore",
        "scopes": [
          "openid",
          "profile",
          "email"
        ]
    }
}
```

8.2.5. More Information

org.forgerock.openig.filter.oauth2.client.ClientRegistration

"*Issuer*", "*OAuth2ClientFilter*"

The OAuth 2.0 Authorization Framework

OAuth 2.0 Bearer Token Usage

OpenID Connect

Chapter 8.3
Delegate

Delegates all method calls to a referenced handler, filter, or any object type.

Use a Delegate to decorate referenced objects differently when they are used multiple times in a configuration.

8.3.1. Usage

```
{
  "filter or handler": {
  "type": "Delegate",
  [decorator reference, ...],
  "config": {
    "delegate": [object reference] }
  }
}
```

8.3.2. More Information

org.forgerock.openig.decoration.DelegateHeaplet

Chapter 8.4

JwtSession

Configures settings for stateless sessions.

Session information is serialized as a secure JWT, that is encrypted and signed, and the resulting JWT string is placed in a cookie. The cookie contains the session attributes as JSON, and a marker for the session timeout.

Use JwtSession to configure stateless sessions as follows:

- Configure a JwtSession object named `Session` in the heap of `config.json`.

  Stateless sessions are created when a request traverses any route or subroute in the configuration. No routes can create stateful sessions.

- Configure a JwtSession object in the `session` property of a Route object.

  When a request enters the route, Microgateway builds a new session object for the route. Any child routes inherit the session. The session information is saved/persisted when the response exits the route. For more information, see "*Route*".

- Configure a JwtSession object in the `session` property of multiple sibling routes in the configuration, using an identical cookie name and cryptographic properties (`encryptionSecretId`, `signatureSecretId`, and `secretsProvider`). Sibling routes are in the same configuration, with no ascending hierarchy to each other.

  When a JwtSession object is declared in a route, the session content is available only within that route. With this configuration, sibling routes can read/write in the same session.

Consider the following points when you configure JwtSession:

- Only JSON-compatible types can be serialized into a JWT and included in a session cookie. Compatible types include primitive JSON structures, lists, arrays, and maps. For more information, see http://json.org.
- The maximum size of a JWT cookie is 4 KB. Because encryption adds overhead, limit the size of any JSON that you store in a cookie. Consider storing large data outside of the session.
- If an empty session is serialized, the supporting cookie is marked as expired and is effectively discarded.

  To prevent Microgateway from cleaning up empty session cookies, consider adding some information to the session context by using an AssignmentFilter. For an example, see "Adding Info To a Session".

- When HTTP clients perform multiple requests in a session that modify the content, the session information can become inconsistent.

8.4.1. Usage

```
{
  "name": string,
  "type": "JwtSession",
  "config": {
    "encryptionSecretId": configuration expression<secret-id>,
    "cookie": object,
    "sessionTimeout": duration,
    "persistentCookie": boolean,
    "secretsProvider": SecretsProvider reference,
    "signatureSecretId": configuration expression<secret-id>,
    "skewAllowance": configuration expression<duration>
  }
}
```

An alternative value for type is JwtSessionFactory.

8.4.2. Properties

`"encryptionSecretId"`: *configuration expression<secret-id>, optional*

The secret ID of the encryption key used to encrypt the JWT.

Default: Microgateway generates a random key/pair for the encryption. Consequently, Microgateway instances cannot share the JWT session.

For information about supported formats for `secret-id`, see secret-id.

`"cookie"` *object, optional*

The configuration of the cookie used to store the encrypted JWT.

Default: The cookie is treated as a host-based cookie.

```
{
  "cookie": {
    "name": configuration expression<string>,
    "domain": configuration expression<string>,
    "httpOnly": configuration expression<boolean>,
    "path": configuration expression<string>,
    "sameSite": configuration expression<enumeration>,
    "secure": configuration expression<boolean>
  }
}
```

`name` *configuration expression<string>, optional*

Name of the JWT cookie stored on the user-agent.

For security, change the default name of cookies.

Default: `openig-jwt-session`

`domain` *configuration expression<string>, optional*

Domain from which the JWT cookie can be accessed. When the domain is specified, a JWT cookie can be accessed from different hosts in that domain.

Set a domain only if the user-agent is able to re-emit cookies on that domain on its next hop. For example, to re-emit a cookie on the domain `.example.com`, the user-agent must be able to access that domain on its next hop.

Default: The fully qualified hostname of the user-agent's next hop.

`"httpOnly"`: *configuration expression<boolean>, optional*

Flag to mitigate the risk of client-side scripts accessing protected cookies.

Default: `true`

`"path"`: *configuration expression<string>, optional*

Path protected by this session.

Set a path only if the user-agent is able to re-emit cookies on the path. For example, to re-emit a cookie on the path `/home/cdsso`, the user-agent must be able to access that path on its next hop.

Default: The path of the request that got the `Set-Cookie` in its response.

`"sameSite"`: *configuration expression<enumeration>, optional*

Manage the circumstances in which a cookie is sent to the server. Use one of the following options to reduce the risk of cross-site request forgery (CSRF) attacks:

- `STRICT`: Send the cookie only if the request was initiated from the cookie domain.
- `LAX`: Send the cookie only with GET requests in a first-party context, where the URL in the address bar matches the cookie domain.

The value is not case-sensitive.

Default: Null; send the cookie whenever a request is made to the cookie domain.

`"secure"`: *configuration expression<boolean>, optional*

Flag to limit the scope of the cookie to secure channels.

Set this flag only if the user-agent is able to re-emit cookies over HTTPS on its next hop. For example, to re-emit a cookie with the `secure` flag, the user-agent must be connected to its next hop by HTTPS.

Default: `false`

`"sessionTimeout"` ***duration, optional***

The duration for which a JWT session is valid. If the supporting cookie is persistent, this property also defines the expiry of the cookie.

The value must be above zero. The maximum value is 3650 days (approximately 10 years). If you set a longer duration, Microgateway truncates the duration to 3650 days.

Default: 30 minutes

For information about supported formats for `duration`, see duration.

`"persistentCookie"` ***boolean, optional***

Whether or not the supporting cookie is persistent:

- `true`: the supporting cookie is a persistent cookie. Persistent cookies are re-emitted by the user-agent until their expiration date or until they are deleted.
- `false`: the supporting cookie is a session cookie. Microgateway does not specify an expiry date for session cookies. The user-agent is responsible for deleting them when it considers that the session is finished (for example, when the browser is closed).

Default: `false`

`"secretsProvider"`: ***SecretsProvider reference, optional***

The SecretsProvider object to query for the JWT session signing or encryption keys. For more information, see "*SecretsProvider*".

Default: The route's default secret service. For more information, see "*secrets*".

`"signatureSecretId"` ***configuration expression<secret-id>, optional***

The secret ID of the secret used to sign and verify JWTs. JWTs are signed with the HS256 algorithm.

`signatureSecretId` can be provided by a SystemAndEnvSecretStore, FileSystemSecretStore, or Base64EncodedSecretStore.

For information about supported formats for `secret-id`, see secret-id.

Default: The secret is generated randomly at Microgateway startup. If Microgateway is restarted, a new secret is generated, and signature validation for the old session fails.

`"skewAllowance"`: ***configuration expression<duration>, optional***

The time to add to the validity period of a JWT to allow for clock skew between different servers. To support a zero-trust policy, the skew allowance is by default zero.

A `skewAllowance` of 2 minutes affects the validity period as follows:

- A JWT with an `iat` of 12:00 is valid from 11:58 on the Microgateway clock.
- A JWT with an `exp` 13:00 is expired after 13:02 on the Microgateway clock.

Default: `zero`

8.4.3. More Information

org.forgerock.openig.jwt.JwtSessionManager

Chapter 8.5
KeyManager

8.5.1. Description

This represents the configuration for a Java Secure Socket Extension KeyManager, which manages the keys used to authenticate an SSLSocket to a peer. The configuration references the keystore that actually holds the keys.

8.5.2. Usage

```
{
  "name": string,
  "type": "KeyManager",
  "config": {
    "keystore": KeyStore reference,
    "passwordSecretId": configuration expression<secret-id>,
    "alg": configuration expression<string>,
    "secretsProvider": SecretsProvider reference
  }
}
```

8.5.3. Properties

`"keystore"`: *KeyStore reference, required*

The KeyStore that references the store for key certificates. When `keystore` is used in a KeyManager, it queries for private keys; when `keystore` is used in a TrustManager, it queries for certificates.

Provide either the name of the KeyStore object defined in the heap, or an inline KeyStore configuration object.

When ClientHandler or ReverseProxyHandler use `keystore`, the keystore can be different to that used by the web container.

See also "*KeyStore*".

`"passwordSecretId"`: *configuration expression<secret-id>, required*

The secret ID of the password required to read private keys from the KeyStore.

For information about supported formats for `secret-id`, see secret-id.

`"alg"` *configuration expression<string>, optional*

The certificate algorithm to use.

Default: the default for the platform, such as `SunX509`.

See also "*Expressions*".

`"secretsProvider"`: *SecretsProvider reference, optional*

The SecretsProvider to query for the keystore password. For more information, see "*SecretsProvider*".

Default: The route's default secret service. For more information, see "*secrets*".

8.5.4. Example

The following example configures a key manager that depends on a KeyStore configuration. The keystore takes a password supplied as a Java system property when starting the container where Microgateway runs, as in `-Dkeypass=password`. This configuration uses the default certificate algorithm:

```
{
  "name": "MyKeyManager",
  "type": "KeyManager",
  "config": {
    "keystore": {
      "type": "KeyStore",
      "config": {
        "url": "file://${env['HOME']}/keystore.jks",
        "passwordSecretId": "keymanager.keystore.secret.id",
        "secretsProvider": "SystemAndEnvSecretStore"
      }
    },
    "passwordSecretId": "keymanager.secret.id",
    "secretsProvider": "SystemAndEnvSecretStore"
  }
}
```

8.5.5. More Information

org.forgerock.openig.security.KeyManagerHeaplet

JSSE Reference Guide , "*KeyStore*", "*TrustManager*"

Chapter 8.6
KeyStore

8.6.1. Description

This represents the configuration for a Java KeyStore, which stores cryptographic private keys and public key certificates.

8.6.2. Usage

```
{
  "name": name,
  "type": "KeyStore",
  "config": {
    "url": configuration expression<uri string>,
    "passwordSecretId": configuration expression<secret-id>,
    "type": configuration expression<string>,
    "secretsProvider": SecretsProvider reference
  }
}
```

8.6.3. Properties

`"url"`: *configuration expression<uri string>, required*

URL to the keystore file.

See also "*Expressions*".

`"passwordSecretId"`: *configuration expression<secret-id>, optional*

The secret ID of the password required to read private keys from the KeyStore.

For information about supported formats for `secret-id`, see secret-id.

If the KeyStore is used as a truststore to store only public key certificates of peers and no password is required to do so, then you do not have to specify this field.

Default: No password is set.

See also "*Expressions*".

`"type"`: *configuration expression<string>, optional*

The KeyStore type. For a list of types, see KeyStore Types.

Default: When this property is not configured, the type is given by the keystore extension, as follows:

Extension	Type
`.jks`	`JKS`
`.jceks`	`JCEKS`
`.p12`, `.pfx`, `.pkcs12`, and all other extensions	`PKCS12`

`"secretsProvider"`: *SecretsProvider reference, optional*

The SecretsProvider to query for the keystore password. For more information, see "*SecretsProvider*".

Default: The route's default secret service. For more information, see "*secrets*".

8.6.4. Example

The following example configures a KeyStore that references a Java KeyStore file, `$HOME/keystore.jks`. The KeyStore takes a password supplied as a Java system property when starting the container where Microgateway runs, as in `-Dkeypass=password`. As the KeyStore file uses the default format, no type is specified:

```
{
  "name": "MyKeyStore",
  "type": "KeyStore",
  "config": {
    "url": "file://${env['HOME']}/keystore.jks",
    "passwordSecretId": "${system['keypass']}",
    "secretsProvider": "SystemAndEnvSecretStore"
  }
}
```

8.6.5. More Information

org.forgerock.openig.security.KeyStoreHeaplet

JSSE Reference Guide , "*KeyManager*", "*TrustManager*"

Chapter 8.7

Issuer

8.7.1. Description

An Issuer describes an OAuth 2.0 Authorization Server or an OpenID Provider that Microgateway can use as a OAuth 2.0 client or OpenID Connect relying party.

An Issuer is generally referenced from a ClientRegistration, described in "*ClientRegistration*".

8.7.2. Usage

```
{
  "name": string,
  "type": "Issuer",
  "config": {
    "wellKnownEndpoint": URL string,
    "authorizeEndpoint": URI expression,
    "registrationEndpoint": URI expression,
    "tokenEndpoint": URI expression,
    "userInfoEndpoint": URI expression,
    "issuerHandler": Handler reference,
    "issuerRepository": Issuer repository reference,
    "supportedDomains": [ domain pattern, ... ]
  }
}
```

8.7.3. Properties

If the provider has a well-known configuration URL as defined for OpenID Connect 1.0 Discovery that returns JSON with at least authorization and token endpoint URLs, then you can specify that URL in the provider configuration. Otherwise, you must specify at least the provider authorization and token endpoint URLs, and optionally the registration endpoint and user info endpoint URLs.

The provider configuration object properties are as follows:

`"name"`: *string, required*

A name for the provider configuration.

`"wellKnownEndpoint"`: *URL string, required unless authorizeEndpoint and tokenEndpoint are specified*

The URL to the well-known configuration resource as described in OpenID Connect 1.0 Discovery.

`"authorizeEndpoint"`: *expression, required unless obtained through wellKnownEndpoint*

The URL to the provider's OAuth 2.0 authorization endpoint.

See also "*Expressions*".

`"registrationEndpoint"`: *expression, optional*

The URL to the provider's OpenID Connect dynamic registration endpoint.

See also "*Expressions*".

`"tokenEndpoint"`: *expression, required unless obtained through wellKnownEndpoint*

The URL to the provider's OAuth 2.0 token endpoint.

See also "*Expressions*".

`"userInfoEndpoint"`: *expression, optional*

The URL to the provider's OpenID Connect UserInfo endpoint.

Default: no UserInfo is obtained from the provider.

See also "*Expressions*".

`"issuerHandler"`: *Handler reference, optional*

Invoke this HTTP client handler to communicate with the authorization server.

Provide either the name of a Handler object defined in the heap, or an inline Handler configuration object.

Usually set this to the name of a ClientHandler configured in the heap, or a chain that ends in a ClientHandler.

Default: Microgateway uses the default ClientHandler.

See also "Handlers", "*ClientHandler*".

`"issuerRepository"`: *Issuer repository reference, optional*

A repository of OAuth 2.0 issuers, built from discovered issuers and the Microgateway configuration.

Provide the name of an IssuerRepository object defined in the heap.

Default: Look up an issuer repository named `IssuerRepository` in the heap. If none is explicitly defined, then a default one named `IssuerRepository` is created in the current route.

See also "*IssuerRepository*".

`"supportedDomains"`: *array of patterns, optional*

List of patterns matching domain names handled by this issuer, used as a shortcut for OpenID Connect discovery before performing OpenID Connect dynamic registration.

In summary when the OpenID Provider is not known in advance, it might be possible to discover the OpenID Provider Issuer based on information provided by the user, such as an email address. The OpenID Connect discovery specification explains how to use WebFinger to discover the issuer. Microgateway can discover the issuer in this way. As a shortcut Microgateway can also use supported domains lists to find issuers already described in the Microgateway configuration.

To use this shortcut, Microgateway extracts the domain from the user input, and looks for an issuer whose supported domains list contains a match.

Supported domains patterns match host names with optional port numbers. Do not specify a URI scheme such as HTTP. Microgateway adds the scheme. For instance, `*.example.com` matches any host in the `example.com` domain. You can specify the port number as well as in `host.example.com:8443`. Patterns must be valid regular expression patterns according to the rules for the Java Pattern class.

8.7.4. Examples

The following example shows an AM issuer configuration for AM. AM exposes a well-known endpoint for the provider configuration, but this example demonstrates use of the other fields:

```
{
    "name": "openam",
    "type": "Issuer",
    "config": {
        "authorizeEndpoint":
          "https://openam.example.com:8443/openam/oauth2/authorize",
        "registration_endpoint":
          "https://openam.example.com:8443/openam/oauth2/connect/register",
        "tokenEndpoint":
          "https://openam.example.com:8443/openam/oauth2/access_token",
        "userInfoEndpoint":
          "https://openam.example.com:8443/openam/oauth2/userinfo",
        "supportedDomains": [ "mail.example.*", "docs.example.com:8443" ]
    }
}
```

The following example shows an issuer configuration for Google:

```
{
    "name": "google",
    "type": "Issuer",
    "config": {
        "wellKnownEndpoint":
          "https://accounts.google.com/.well-known/openid-configuration",
        "supportedDomains": [ "gmail.*", "googlemail.com:8052" ]
    }
}
```

8.7.5. More Information

org.forgerock.openig.filter.oauth2.client.Issuer

Chapter 8.8
JdbcDataSource

8.8.1. Description

Manages connections to a JDBC data source.

8.8.2. Usage

```
{
  "name": string,
  "type": "JdbcDataSource",
  "config": {
    "dataSourceClassName": configuration expression<string>,
    "driverClassName": configuration expression<string>,
    "executor": object,
    "jdbcUrl": configuration expression<url>,
    "passwordSecretId": configuration expression<secret-id>,
    "poolName": configuration expression<string>,
    "properties": object,
    "secretsProvider": SecretsProvider reference,
    "username": configuration expression<string>
  }
}
```

8.8.3. Properties

`"dataSourceClassName"`: *configuration expression<string>, optional*

The data source class name to use to connect to the database.

Depending on the underlying data source, use either `jdbcUrl`, or `dataSourceClassName` with `url`. See the "Properties".

`"driverClassName"`: *configuration expression<string>, optional*

Class name of the JDBC connection driver. The following examples can be used:

- MySQL Connector/J: `com.mysql.jdbc.Driver`
- H2: `org.h2.Driver`

This property is optional, but required for older JDBC drivers.

`"executor"`: *ScheduledExecutorService reference, optional*

The service to schedule the execution of maintenance tasks.

Default: "*ScheduledExecutorService*".

`"jdbcUrl"`: *configuration expression<string>, optional*

The JDBC URL to use to connect to the database.

Depending on the underlying data source, use either `jdbcUrl`, or `dataSourceClassName` with `url`. See the "Properties".

`"passwordSecretId"`: *configuration expression<secret-id>, required if the database is password-protected*

The secret ID of the password to access the database.

`"poolName"`: *configuration expression<string>, optional*

The connection pool name. Use to identify a pool easily for maintenance and monitoring.

`"properties"`: *object, optional*

Server properties specific to the type of data source being used. For information about available options, see the data source documentation.

`"secretsProvider"`: *SecretsProvider reference, optional*

The "*SecretsProvider*" to use to resolve queried secrets, such as passwords and cryptographic keys. Provide either the name of a SecretsProvider object defined in the heap, or specify a SecretsProvider object inline.

`"username"`: *configuration expression<string>, optional*

The username to access the database.

8.8.4. Example

The following example configures a JdbcDataSource with a `dataSourceClassName` and `url`:

```
"config": {
  "username": "testUser",
  "dataSourceClassName": "org.h2.jdbcx.JdbcDataSource",
  "properties": {
    "url": "jdbc:h2://localhost:3306/auth"
  },
  "passwordSecretId: "database.password",
  "secretsProvider": "MySecretsProvider",
}
```

The following example configures a JdbcDataSource with `jdbcUrl` alone:

```
"config": {
  "username": "testUser",
  "jdbcUrl": "jdbc:h2://localhost:3306/auth",
  "passwordSecretId: "database.password",
  "secretsProvider": "MySecretsProvider",
}
```

The following example configures a JdbcDataSource with `jdbcUrl` and `driverName`. Use this format for older drivers, where `jdbcUrl` does not provide enough information:

```
"config": {
  "username": "testUser",
  "jdbcUrl": "jdbc:h2://localhost:3306/auth",
  "driverName": "org.h2.Driver",
  "passwordSecretId: "database.password",
  "secretsProvider": "MySecretsProvider",
}
```

8.8.5. More Information

org.forgerock.openig.sql.JdbcDataSourceHeaplet

Chapter 8.9

IssuerRepository

8.9.1. Description

A repository to store OAuth2 issuers that are discovered or built from the configuration.

It is not normally necessary to change this object. Change it only for the following tasks:

- To isolate different repositories in the same route.
- To view the interactions of the well-known endpoint, for example, if the `issuerHandler` is delegating to another handler.

8.9.2. Usage

```
{
   "name": string,
   "type": "IssuerRepository",
   "config": {
     "issuerHandler": Handler reference
   }
}
```

8.9.3. Properties

The object properties are as follows:

`"issuerHandler"`: ***handler reference, optional***

The default handler to fetch OAuth2 issuer configurations from the well-known endpoint.

Provide the name of a Handler object defined in the heap, or an inline Handler configuration object.

Default: ForgeRockClientHandler

8.9.4. More Information

org.forgerock.openig.filter.oauth2.client.IssuerRepository

Chapter 8.10

ScheduledExecutorService

8.10.1. Description

An executor service to schedule tasks for execution after a delay or for repeated execution with a fixed interval of time in between each execution. You can configure the number of threads in the executor service and how the executor service is stopped.

The `ScheduledExecutorService` is shared by all downstream components that use an executor service.

8.10.2. Usage

```
{
  "name": string,
  "type": "ScheduledExecutorService",
  "config": {
    "corePoolSize":  configuration expression<integer>,
    "gracefulStop":  configuration expression<boolean>,
    "gracePeriod":  configuration expression<duration>
  }
}
```

8.10.3. Properties

`"corePoolSize"`: *configuration expression<integer>, optional*

The minimum number of threads to keep in the pool. If this property is an expression, the expression is evaluated as soon as the configuration is read.

The value must be an integer greater than zero.

Default: 1

`"gracefulStop"`: *configuration expression<boolean>, optional*

Defines how the executor service stops.

If true, the executor service does the following:

- Blocks the submission of new jobs.

- If a grace period is defined, waits for up to that maximum time for submitted and running jobs to finish.
- Removes submitted jobs without running them.
- Attempts to end running jobs.

If false, the executor service does the following:

- Blocks the submission of new jobs.
- If a grace period is defined, ignores it.
- Removes submitted jobs without running them.
- Attempts to end running jobs.

Default: true

`"gracePeriod"`: *configuration expression<duration>, optional*

The maximum time that the executor service waits for running jobs to finish before it stops. If this property is an expression, the expression is evaluated as soon as the configuration is read.

If all jobs finish before the grace period, the executor service stops without waiting any longer. If jobs are still running after the grace period, the executor service removes the scheduled tasks, and notifies the running tasks for interruption.

When `gracefulStop` is `false`, the grace period is ignored.

Default: 10 seconds

For information about supported formats for `duration`, see duration.

8.10.4. Example

The following example creates a thread pool to execute tasks. When the executor service is instructed to stop, it blocks the submission of new jobs, and waits for up to 10 seconds for submitted and running jobs to complete before it stops. If any jobs are still submitted or running after 10 seconds, the executor service stops anyway and prints a message.

```
{
    "name": "ExecutorService",
    "comment": "Default service for executing tasks in the background.",
    "type": "ScheduledExecutorService",
    "config": {
        "corePoolSize": 5,
        "gracefulStop": true,
        "gracePeriod": "10 seconds"
    }
}
```

8.10.5. More Information

org.forgerock.openig.thread.ScheduledExecutorServiceHeaplet

Chapter 8.11

SecretsProvider

8.11.1. Description

Use the specified secret stores to resolve queried secrets, such as passwords and cryptographic keys. Attempt to resolve the secret with the secret stores in the order that they are declared in the array.

8.11.2. Usage

```
{
  "name": string,
  "type": "SecretsProvider",
  "config": {
    "stores": [ secret store declaration, ... ]
  }
}
```

This object can alternatively be configured in a compact format, without the SecretsProvider declaration, as follows:

- With an inline secret store:

```
"secretsProvider": {
  "type": "secret store type1",
  "config": {...}
}
```

- With multiple inline secret stores:

```
"secretsProvider": [
  {
    "type": "secret store type1",
    "config": {...}
  }
  {
    "type": "secret store type2",
    "config": {...}
  }
]
```

- With a referenced secret store:

```
"secretsProvider": "mySecretStore1"
```

- With multiple referenced secret stores:

```
"secretsProvider": [
  "mySecretStore1", "mySecretStore2"
]
```

8.11.3. Properties

`"stores"`: *array of secret store declarations, required*

One or more secret stores to provide access to stored secrets. Configure secret stores described in "Secrets".

8.11.4. More Information

"*StatelessAccessTokenResolver*"

"Secrets"

org.forgerock.secrets.SecretsProvider

Chapter 8.12

TemporaryStorage

8.12.1. Description

Allocates temporary buffers for caching streamed content during request processing. Initially uses memory; when the memory limit is exceeded, switches to a temporary file.

8.12.2. Usage

```
{
     "name": string,
     "type": "TemporaryStorage",
     "config": {
         "initialLength": number,
         "memoryLimit": number,
         "fileLimit": number,
         "directory": string
     }
}
```

8.12.3. Properties

`"initialLength"`: *number, optional*

Initial size of the memory buffer.

Default: 8 192 bytes (8 KB). Maximum: The value of `"memoryLimit"`.

`"memoryLimit"`: *number, optional*

Maximum size of the memory buffer. When the memory buffer is full, the content is transferred to a temporary file.

Default: 65 536 bytes (64 KB). Maximum: 2 147 483 647 bytes (2 GB).

`"fileLimit"`: *number, optional*

Maximum size of the temporary file. If the downloaded file is bigger than this value, an OverflowException is thrown.

Default: 1 073 741 824 bytes (1 GB). Maximum: 2 147 483 647 bytes (2 GB).

`"directory"`: *string, optional*

The directory where temporary files are created.

Default: The value of the system property `java.io.tmpdir`, typically `/tmp` on Unix systems, or `/var/tmp` on Linux.

8.12.4. More Information

org.forgerock.openig.io.TemporaryStorageHeaplet

Chapter 8.13
TlsOptions

8.13.1. Description

Configure connections to TLS-protected endpoints. Use in "*ClientHandler*", "*ReverseProxyHandler*", and "*AmService*".

8.13.2. Usage

```
{
  "type": TlsOptions,
  "config": {
    "keyManager": KeyManager reference or [ KeyManager reference, ...],
    "sslCipherSuites": [ configuration expression<string>, ...],
    "sslContextAlgorithm": configuration expression<string>,
    "sslEnabledProtocols": [ configuration expression<string>, ...],
    "trustManager": TrustManager reference or [ TrustManager reference, ...]
  }
}
```

8.13.3. Properties

`"keyManager"`: *KeyManager reference or array of KeyManager references, optional*

One or more key managers to handle this client's keys and certificates.

Provide either the name(s) of KeyManager object(s) defined in the heap, or specify the configuration object(s) inline.

Specify a single KeyManager or an array of KeyManagers:

- `"keyManager": "MyKeyManager"`
- `"keyManager": [ "FirstKeyManager", "SecondKeyManager" ]`

If keyManager is not configured, the client can't authenticate itself in a mutual authentication scenario.

See also "*KeyManager*".

`"sslCipherSuites"`: *array of configuration expression<string>, optional*

Array of cipher suite names, used to restrict the cipher suites allowed when negotiating transport layer security for an HTTPS connection.

For information about the available cipher suite names, see the documentation for the Java virtual machine (JVM) used by the container where you run Microgateway. For Oracle Java, see the list of *JSSE Cipher Suite Names* .

Default: Allow any cipher suite supported by the JVM.

`"sslContextAlgorithm"`: *configuration expression<string>, optional*

The `SSLContext` algorithm name, as listed in the table of *`SSLContext` Algorithms* for the Java Virtual Machine used by the container where Microgateway runs.

Default: `TLS`

`"sslEnabledProtocols"`: *array of configuration expression<string>, optional*

Array of protocol names, used to restrict the protocols allowed when negotiating transport layer security for an HTTPS connection.

For information about the available protocol names, see the documentation for the Java virtual machine (JVM) used by the container where you run Microgateway. For Oracle Java, see the list of *Additional JSSE Standard Names* .

Default: Allow any protocol supported by the JVM.

`"trustManager"`: *TrustManager reference or array of TrustManager references, optional*

One or more trust managers that handle peers' public key certificates.

Provide either the name(s) of TrustManager object(s) defined in the heap, or specify the configuration object(s) inline.

Specify a single TrustManager or an array of TrustManager:

- `"trustManager": "TrustManager"`
- `"trustManager": [ "FirstTrustManager", "SecondTrustManager" ]`

If you do not configure a trust manager, then the client uses only the default Java truststore. The default Java truststore depends on the Java environment. For example, `$JAVA_HOME/lib/security/cacerts`.

See also "*TrustManager*".

8.13.4. Example

```
{
```

```
  "tls": {
    "type": "TlsOptions",
    "config": {
      "sslContextAlgorithm": "TLSv1.2",
      "keyManager": {
        "type": "KeyManager",
        "config": {
          "keystore": {
            "type": "KeyStore",
            "config": {
              "url": "file://${env['HOME']}/keystore.jks",
              "passwordSecretId": "keymanager.keystore.secret.id",
              "secretsProvider": "SystemAndEnvSecretStore"
            }
          },
          "passwordSecretId": "keymanager.secret.id",
          "secretsProvider": "SystemAndEnvSecretStore"
        }
      },
      "trustManager": {
        "type": "TrustManager",
        "config": {
          "keystore": {
            "type": "KeyStore",
            "config": {
              "url": "file://${env['HOME']}/truststore.jks",
              "passwordSecretId": "trustmanager.keystore.secret.id",
              "secretsProvider": "SystemAndEnvSecretStore"
            }
          }
        }
      }
    }
  }
}
```

Chapter 8.14

TrustManager

8.14.1. Description

This represents the configuration for a Java Secure Socket Extension TrustManager, which manages the trust material (typically X.509 public key certificates) used to decide whether to accept the credentials presented by a peer. The configuration references the keystore that actually holds the trust material.

8.14.2. Usage

```
{
    "name": string,
    "type": "TrustManager",
    "config": {
      "keystore": KeyStore reference,
      "alg": string
    }
}
```

8.14.3. Properties

`"keystore"`: *KeyStore reference, required*

The KeyStore that references the store for key certificates. When `keystore` is used in a KeyManager, it queries for private keys; when `keystore` is used in a TrustManager, it queries for certificates.

Provide either the name of the KeyStore object defined in the heap, or an inline KeyStore configuration object.

When ClientHandler or ReverseProxyHandler use `keystore`, the keystore can be different to that used by the web container.

See also "*KeyStore*".

`"alg"` *string, optional*

The certificate algorithm to use.

Default: the default for the platform, such as `SunX509`.

8.14.4. Example

The following example configures a trust manager that depends on a KeyStore configuration. This configuration uses the default certificate algorithm:

```
{
  "name": "MyTrustManager",
  "type": "TrustManager",
  "config": {
    "keystore": {
      "type": "KeyStore",
      "config": {
        "url": "file://${env['HOME']}/keystore.jks",
        "passwordSecretId": "${system['keypass']}",
        "secretsProvider": "SystemAndEnvSecretStore"
      }
    }
  }
}
```

8.14.5. More Information

org.forgerock.openig.security.TrustManagerHeaplet

JSSE Reference Guide , "*KeyManager*", "*KeyStore*"

Chapter 8.15

TrustAllManager

8.15.1. Description

The TrustAllManager blindly trusts all server certificates presented the servers for protected applications. It can be used instead of a "*TrustManager*" in test environments to trust server certificates that were not signed by a well-known CA, such as self-signed certificates.

The TrustAllManager is not safe for production use. Use a properly configured "*TrustManager*" instead.

8.15.2. Usage

```
{
    "name": string,
    "type": "TrustAllManager"
}
```

8.15.3. Example

The following example configures a client handler that blindly trusts server certificates when Microgateway connects to servers over HTTPS:

```
{
    "name": "BlindTrustClientHandler",
    "type": "ReverseProxyHandler",
    "config": {
        "trustManager": {
            "type": "TrustAllManager"
        }
    }
}
```

8.15.4. More Information

org.forgerock.openig.security.TrustAllManager

Chapter 8.16

UmaService

8.16.1. Description

The UmaService includes a list of resource patterns and associated actions that define the scopes for permissions to matching resources. When creating a share using the REST API described below, you specify a path matching a pattern in a resource of the UmaService.

UMA 2.0 is supported with AM 5.5 and later versions. UMA 1.0 is supported with AM 5.1 and later versions.

8.16.2. Usage

```
{
    "type": "UmaService",
    "config": {
        "protectionApiHandler": Handler reference,
        "amService": AmService reference,   // Use either "amService"
        "wellKnownEndpoint": URI string,    // or "wellKnownEndpoint", but not both.
        "resources": [ resource, ... ]
    }
}
```

8.16.3. Properties

`"protectionApiHandler"`: *Handler reference, required*

The handler to use when interacting with the UMA authorization server to manage resource sets, such as a ClientHandler capable of making an HTTPS connection to the server.

For more information, see "Handlers".

`"amService"`: *AmService reference, required if `"wellKnownEndpoint"` is not configured*

The AmService heap object to use for the URI to the well-known endpoint for this UMA authorization server. The endpoint is extrapolated from the `url` property of the AmService, and takes the realm into account.

If the UMA authorization server is AM, use this property to define the endpoint.

If `amService` is configured, it takes precedence over `wellKnownEndpoint`.

For more information about the endpoint for UMA configuration, see Discovering UMA Configuration in the AM *User-Managed Access (UMA) 2.0 Guide*.

See also, "*AmService*".

`"wellKnownEndpoint"`: *URI string, required if `"amService"` is not configured*

The URI to the well-known endpoint for this UMA authorization server.

If the UMA authorization server is not AM, use this property to define the endpoint.

If `amService` is configured, it takes precedence over `wellKnownEndpoint`.

In this example, the UMA configuration is in the default realm of AM:

```
https://openam.example.com:8088/openam/uma/.well-known/uma2-configuration
```

In this example, the UMA configuration is in a European customer realm:

```
https://openam.example.com:8088/openam/uma/realms/root/realms/customer/realms/europe/.well-known/uma2-configuration
```

For more information about the endpoint for UMA configuration, see Discovering UMA Configuration in the AM *User-Managed Access (UMA) 2.0 Guide*.

`"resources"`: *array of resources, required*

Resource objects matching the resources the resource owner wants to share.

Each resource object has the following form:

```
{
    "pattern": resource pattern,
    "actions": [
        {
            "scopes": [ scope string, ... ],
            "condition": runtime expression<boolean>
        },
        {
            ...
        }
    ]
}
```

Each resource pattern can be seen to represent an application, or a consistent set of endpoints that share scope definitions. The actions map each request to the associated scopes. This configuration serves to set the list of scopes in the following ways:

1. When registering a resource set, Microgateway uses the list of actions to provide the aggregated, exhaustive list of all scopes that can be used.
2. When responding to an initial request for a resource, Microgateway derives the scopes for the ticket based on the scopes that apply according to the request.

3. When verifying the RPT, Microgateway checks that all required scopes are encoded in the RPT.

A description of each field follows:

`"pattern"`: *resource pattern, required*

A pattern matching resources to be shared by the resource owner, such as `.*` to match any resource path, and `/photos/.*` to match paths starting with `/photos/`.

See also "*Patterns*".

`"actions"`: *array of action objects, optional*

A set of actions on matching resources that the resource owner can authorize.

When granting permission, the resource owner specifies the action scope. Conditions specify what the scopes mean in concrete terms. A given scope matches a requesting party operation when the corresponding condition evaluates to `true`.

`"scopes"`: *array of scope strings, optional*

Scope strings to identify permissions.

For example, `#read` (read access on a resource).

`"condition"`: *runtime expression<boolean>, required*

A boolean expression representing the meaning of a scope.

For example, `${request.method == 'GET'}` (true when reading a resource).

See also "*Expressions*".

8.16.4. The REST API for Shares

The REST API for UMA shares is exposed at a registered endpoint. Microgateway logs the paths to registered endpoints when the log level is `INFO` or finer. Look for messages such as the following in the log:

```
UMA Share endpoint available at
 '/openig/api/system/objects/_router/routes/00-uma/objects/umaservice/share'
```

To access the endpoint over HTTP or HTTPS, prefix the path with the Microgateway scheme, host, and port to obtain a full URL, such as `http://localhost:8080/openig/api/system/objects/_router/routes/00-uma/objects/umaservice/share`.

The UMA REST API supports create (POST only), read, delete, and query (`_queryFilter=true` only). For an introduction to common REST APIs, see "About ForgeRock Common REST".

In the present implementation, Microgateway does not have a mechanism for persisting shares. When the Microgateway container stops, the shares are discarded.

A share object has the following form:

```
{
    "path": pattern,
    "pat": UMA protection API token (PAT) string,
    "id": unique identifier string,
    "resource_id": unique identifier string,
    "user_access_policy_uri": URI string
}
```

The fields are as follows:

`"path"`: *pattern, required*

A pattern matching the path to protected resources, such as `/photos/.*`.

This pattern must match a pattern defined in the UmaService for this API.

See also "*Patterns*".

`"pat"`: *PAT string, required*

A PAT granted by the UMA authorization server given consent by the resource owner.

In the present implementation, Microgateway has access only to the PAT, not to any refresh tokens.

`"id"`: *unique identifier string, read-only*

This uniquely identifies the share. This value is set by the service when the share is created, and can be used when reading or deleting a share.

`"resource_id"`: *unique identifier string, read-only*

This uniquely identifies the UMA resource set registered with the authorization server. This value is obtained by the service when the resource set is registered, and can be used when setting access policy permissions.

`"user_access_policy_uri"`: *URI string, read-only*

This URI indicates the location on the UMA authorization server where the resource owner can set or modify access policies. This value is obtained by the service when the resource set is registered.

8.16.5. More Information

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

org.forgerock.openig.uma.UmaSharingService

Part 9

Property Value Substitution

In an environment with multiple Microgateway instances, you can require similar but not identical configurations across the different instances.

Property value substitution enables you to do the following:

- Define a configuration that is specific to a single instance, for example, setting the location of the keystore on a particular host.
- Define a configuration whose parameters vary between different environments, for example, the URLs and passwords for test, development, and production environments.
- Disable certain capabilities on specific nodes.

Property value substitution uses *configuration tokens* to introduce variables into the server configuration. For information, see "*Configuration Tokens*".

The substitution follows a process of token resolution, JSON evaluation, and data transformation, as described in the following sections:

- "*JSON Evaluation*"
- "*Token Resolution*"
- "*Transformations*"

Chapter 9.1
Configuration Tokens

9.1.1. Description

A configuration token is a simple reference to a value. When configuration tokens are resolved, the result is always a string. Transformation described in "*Transformations*" can be used to coerce the output type.

9.1.1.1. Configuration Tokens for File System

Microgateway provides `ig.instance.dir` and `ig.instance.url` to define the file system directory and URL for configuration files.

Their values are computed at startup, and evaluate to a directory such as `/path/to/microservices/identity-gateway` (`%appdata%\OpenIG`). You can use these tokens in your configuration without explicitly setting their values.

9.1.2. Syntax

Configuration tokens follow the syntax `&{token[|default]}`, as follows:

- Are preceded by an ampersand, `&`
- Are enclosed in braces, `{}`
- Define default values with a vertical bar (`|`) after the configuration token
- Are in lowercase
- Use the period as a separator, `.`

When a configuration token is supplied in a configuration parameter, it is always inside a string enclosed in quotation marks, as shown in the following example:

```
"&{listen.port|8080}"
```

To escape a string with the syntax of a configuration token, use a backslash (`\`). The following string is treated as normal text:

```
"\&{listen.port|8080}"
```

A configuration property can include a mix of static values and expressions, as shown in the following example:

```
"&{hostname}.example.com"
```

Configuration tokens can be nested inside other configuration tokens as shown in the following example:

```
"&{&{protocol.scheme}.port}"
```

Default values or values in the property resolver chain can be nested, as shown in the following example:

```
"&{&{protocol.scheme|http}.port|8080}"
```

Chapter 9.2
JSON Evaluation

9.2.1. Description

JSON evaluation is the process of substituting configuration tokens and transforming JSON nodes for an entire JSON configuration. After JSON evaluation, all configuration tokens and transformations in the configuration are replaced by values.

At startup, Microgateway evaluates the configuration tokens in `config.json` and `admin.json`. When routes are deployed, Microgateway evaluates the configuration tokens in the route.

Configuration tokens are matched with tokens available in the chain of resolvers, and the configuration token is substituted with the value available in the resolver. For information about each of the resolvers mentioned in the following section, see "*Token Resolution*".

Microgateway searches for matching tokens in the chain of resolvers, using the following order of precedence:

1. Local resolver:
   - The route resolver for the route being deployed
2. Intermediate resolver:
   - All intermediate route resolvers (for example, for parent routes to the route being deployed) up to the bootstrap resolver
3. Bootstrap resolver:
   a. Environment variables resolver
   b. System properties resolver
   c. Token source file resolvers
   d. Hardcoded default values

The first resolver that matches the token returns the value of the token.

If the token can't be resolved, Microgateway uses the default value defined with the configuration token. If there is no default value, the token can't be resolved and an error occurs:

- If the configuration token is in `config.json` or `admin.json`, Microgateway fails to start up.

- If the configuration token is in a route, the route fails to load.

When configuration tokens are nested inside other configuration tokens, the tokens are evaluated bottom-up, or leaf-first. For example, if the following configuration token takes only the default values, it is resolved as follows:

1. `"&{&{protocol.scheme|http}.port|8080}"`
2. `"&{http.port|8080}"`

   When `&{protocol.scheme|http}` takes the default value `http`.
3. `"8080"`

   When `&{http.port|8080}` takes the default value `8080`.

If the configuration includes a transformation, Microgateway applies the transformation after the token is substituted. When transformations are nested inside other transformations, the transformations are applied bottom-up, or leaf-first. For more information, see "*Transformations*".

Chapter 9.3
Token Resolution

9.3.1. Description

At startup, the bootstrap resolver builds a chain of resolvers to resolve configuration tokens included in `config.json` and `admin.json`. When a route is deployed, *route resolvers* build on the chain to add resolvers for the route.

9.3.2. Route Token Resolvers

When a route is deployed in Microgateway a route resolver is created to resolve the configuration tokens for the route. The resolvers uses token values defined in the `properties` section of the route.

If the token can't be resolved locally, the route resolver accesses token values recursively in a parent route.

For more information, about route properties, see "*Properties*".

9.3.3. Environment Variables Resolver

When the bootstrap resolver resolves a configuration token to an environment variable, it replaces the lowercase and periods (`.`) in the token to match the convention for environment variables.

Environment variable keys are transformed as follows:

- Periods (.) are converted to underscores
- All characters are transformed to uppercase

The following example sets the value of an environment variable for the port number:

```
$ export LISTEN_PORT=8080
```

In the following Microgateway configuration, the value of `port` is `8080`:

```
{
  "port": "&{listen.port}"
}
```

9.3.4. System Properties Resolver

The system property name must match a configuration token exactly. The following example sets a system property for a port number:

```
$ java -Dlisten.port=8080 -jar start.jar
```

In the following Microgateway configuration, the value of `port` is `8080`:

```
{
  "port": "&{listen.port}"
}
```

9.3.5. Token Source File Resolvers

Token source files have the `.json` or `.properties` extension. The bootstrap resolver uses the files to add file resolvers to the chain of resolvers:

- **JSON file resolvers**

  Token source files with the `.json` extension take a JSON format. The token name is mapped either to the JSON attribute name or to the JSON path.

  Each of the following `.json` files set the value for the configuration token `product.listen.port` :

  ```
  {
    "product.listen.port": 8080
  }
  ```

  ```
  {
    "product.listen {
       "port": 8080
    }
  }
  ```

  ```
  {
    "product" {
      "listen" {
        "port": 8080
      }
    }
  }
  ```

- **Properties file resolvers**

  Token source files with the `.properties` extension are Java properties files. They contain a flat list of key/value pairs, and keys must match tokens exactly.

  The following `.properties` file also sets the value for the tokens `listen.port` and `listen.address`:

  ```
  listen.port=8080
  listen.address=192.168.0.10
  ```

Token source files are stored in one or more directories defined by the environment variable `IG_ENVCONFIG_DIRS` or the system property `ig.envconfig.dirs`.

If token source files are in multiple directories, each directory must be specified in a comma-separated list. Microgateway doesn't scan subdirectories. The following example sets an environment variable to define two directories that hold token source files:

```
$ export IG_ENVCONFIG_DIRS="/myconfig/directory1, /myconfig/directory2"
```

At startup, the bootstrap resolver scans the directories in the specified order, and adds a resolver to the chain of resolvers for each token source file in the directories.

Although the bootstrap resolver scans the directories in the specified order, within a directory it scans the files in a nondeterministic order.

Note the following constraints for using the same configuration token more than once:

- Do not define the same configuration token more than once in a single file. There is no error, but you won't know which token is used.
- Do not define the same configuration token in more than one file in a single directory. An error occurs.

  Important

  This constraint implies that you can't have backup `.properties` and `.json` files in a single directory if they define the same tokens.

- You can define the same configuration token once in several files that are located in different directories, but the first value that Microgateway reads during JSON evaluation is used.

Note

When logging is enabled at the DEBUG level for token resolvers, the origin of the token value is logged.

If you are using the default logback implementation, add the following line to your `logback.xml` to enable logging:

```
<logger name="org.forgerock.config.resolvers" level="DEBUG"/>
```

Chapter 9.4

Transformations

A set of built-in transformations are available to coerce strings to other data types. The transformations can be applied to any string, including strings resulting from the resolution of configurations tokens.

After transformation, the JSON node representing the transformation is replaced by the result value.

The following sections describe how to use transformations, and describe the transformations available:

- "Usage"
- "array"
- "bool"
- "decodeBase64"
- "encodeBase64"
- "int"
- "list"
- "number"
- "object"

9.4.1. Usage

```
{
   "$transformation": string or transformation
}
```

A transformation is a JSON object with a required main attribute, starting with a `$`. The following example transforms a string to an integer:

```
{"$int": string}
```

The value of a transformation value can be a JSON string or another transformation that results in a string. The following example shows a nested transformation:

```
{
  "$array": {
    "$base64:decode": string
  }
}
```

The input string must match the format expected by the transformation. In the previous example, because the final transformation is to an array, the input string must be a string that represents an array, such as `"[ \"one\", \"two\" ]"`.

In the first transformation, the encoded string is transformed to a base64-decoded string. In the second, the string is transformed into a JSON array, for example, `[ "one", "two" ]`.

9.4.2. array

```
{"$array": string}
```

Returns a JSON array of the argument.

Argument

string

String representing a JSON array.

Returns

array

JSON array.

The following example transformation results in the JSON array `[ "one", "two" ]`:

```
{"$array": "[ \"one\", \"two\" ]"}
```

9.4.3. bool

```
{"$bool": string}
```

Returns a Boolean with a value represented by the argument.

Returns `true` if the input value equals `"true"` (ignoring case). Otherwise, returns `false`.

Argument

string

String containing the boolean representation.

Returns

boolean

Boolean value represented by the argument.

If the configuration token `&{capture.entity}"` resolves to `"true"`, the following example transformation results in the value `true`:

```
{"$bool": "&{capture.entity}"}
```

9.4.4. decodeBase64

```
{
  "$base64:decode": string,
  "$charset": "charset"
}
```

Transforms a base64-encoded string into a decoded string.

If `$charset` is specified, the decoded value is interpreted with the character set.

Argument

string

Base64-encoded string.

Parameters

`$charset`

The name of a Java character set, as described in Class Charset.

Returns

string

Base64-decoded string in the given character set.

The following example transformation returns the `Hello` string:

```
{
 "$base64:decode": "SGVsbG8=",
 "$charset": "UTF-8"
}
```

9.4.5. encodeBase64

```
{
  "$base64:encode": string,
  "$charset": "charset"
}
```

Transforms a string into a base64-encoded string. Transforms to `null` if the string is `null`.

If `$charset` is specified, the string is encoded with the character set.

Argument

string

String to encode with the given character set.

Parameters

`$charset`

The name of a Java character set, as described in Class Charset.

Returns

string

Base64-encoded string.

9.4.6. int

```
{"$int": string}
```

Transforms a string into an integer.

If the parameter is not a valid number in radix 10, returns `null`.

Argument

string

String containing the integer representation.

Returns

int

Integer value represented by the argument.

The following example transformation results in the integer `1234`:

```
{"$int": "1234"}
```

9.4.7. list

```
{$list: string}
```

Transforms a comma-separated list of strings into a JSON array of strings

Argument

string

A string representing a comma-separated list of strings.

Returns

array

The JSON array of the provided argument. Values are not trimmed of leading spaces.

The following example transformation results in the array of strings `["Apple","Banana","Orange","Strawberry"]`:

```
{"$list": "Apple,Banana,Orange,Strawberry"}
```

The following example transformation results in the array of strings `["Apple"," Banana"," Orange"," Strawberry"]`, including the untrimmed spaces:

```
{"$list": "Apple, Banana, Orange, Strawberry"}
```

The following example transformation results in the array of strings `["1","2","3","4"]`, and not an array of JSON numbers `[1,2,3,4]`:

```
{"$list": "1,2,3,4"}
```

9.4.8. number

```
{$number: string}
```

Transform a string into a Java number, as defined in Class Number.

Argument

strings

A string containing the number representation.

Returns

number

The number value represented by the argument.

The following example transformation results in the number `0.999`:

```
{"$number": ".999"}
```

9.4.9. object

```
{"$object": string}
```

Transforms a string representation of a JSON object into a JSON object.

Argument

string

String representation of a JSON object.

Returns

object

JSON object of the argument.

The following example transformation

```
{"$object": "{\"ParamOne\":{\"InnerParamOne\":\"InnerParamOneValue\",\"InnerParamTwo\": false}}"}
```

results in the following JSON object:

```
{
  "ParamOne": {
    "InnerParamOne": "myValue",
    "InnerParamTwo": false
  }
}
```

Part 10

Expressions

Many configuration parameters support dynamic expressions.

Chapter 10.1
Expressions

10.1.1. Description

Expressions are specified as configuration parameter values for a number of built-in objects. Such expressions conform to the Universal Expression Language as specified in JSR-245.

10.1.2. General Syntax

All expressions follow standard Universal Expression Language syntax: `${expression}`. The expression can be a simple reference to a value, a function call, or arbitrarily complex arithmetic, logical, relational and conditional operations. When supplied within a configuration parameter, an expression is always a string enclosed in quotation marks, for example: `"${request.method}"`.

10.1.3. Configuration and Runtime Expressions

Expressions are evaluated at configuration time (when routes are loaded), or at runtime (when Microgateway is running).

When expressions are evaluated, they access the current environment through the implicit object `openig`. The object has the following properties:

- `baseDirectory`, the path to the base location for Microgateway files. The default location is `/path/to/microservices/identity-gateway` (`%appdata%\OpenIG`).
- `configDirectory`, the path to the Microgateway configuration files. The default location is `/path/to/microservices/identity-gateway/config` (`%appdata%\OpenIG\config`).
- `temporaryDirectory`, the path to the Microgateway temporary files. The default location is `/path/to/microservices/identity-gateway/tmp` (`%appdata%\OpenIG\tmp`).

configuration expression

Expression evaluated at configuration time, when routes are loaded.

Configuration expressions can refer to the system heap properties, the built-in functions listed in "*Functions*", the `${env['variable']}`, and `${system['property']}`. Because configuration expressions

are evaluated before any requests are made, they cannot refer to the runtime properties, `request`, `response`, or `context`. For more information, see "*Expressions*".

runtime expression

Expression evaluated at runtime, for each request and response.

Runtime expressions can refer to the same information as configuration expressions, plus the following objects:

- `attributes`: org.forgerock.services.context.AttributesContext Map<String, Object>, obtained from `AttributesContext.getAttributes()`. For information, see "*AttributesContext*".
- `context`: org.forgerock.services.context.Context object.
- `contexts`: map<string, context> object. For information, see "*Contexts*".
- `request`: org.forgerock.http.protocol.Request object. For information, see "*Request*".
- `response`: org.forgerock.http.protocol.Response object, available only when the expression is intended to be evaluated on the response flow. For information, see "*Response*".
- `session`: org.forgerock.http.session.Session object, available only when the expression is intended to be evaluated for both request and response flow. For information, see "*SessionContext*".

10.1.4. Value Expressions

A value expression references a value relative to the scope supplied to the expression. For example, `"${request.method}"` references the method of an incoming HTTP request.

An *lvalue-expression* is a specific type of value expression that references a value to be written. For example, `"${session.gotoURL}"` specifies a session attribute named `gotoURL` to write a value to. Attempts to write values to read-only values are ignored.

10.1.5. Indexed Properties

Properties of values are accessed using the `.` and `[]` operators, and can be nested arbitrarily.

The value expressions `"${request.method}"` and `"${request['method']}"` are equivalent.

To prevent errors, in property names containing characters that are also expression operators, use the `[]` operator instead of the `.` operator. For example, use `contexts.amSession.properties['a-b-c']` instead of `contexts.amSession.properties.a-b-c`.

In the case of arrays, the index of an element in the array is expressed as a number in brackets. For example, `"${request.headers['Content-Type'][0]}"` references the first `Content-Type` header value in a request. If a property does not exist, then the index reference yields a `null` (empty) value.

10.1.6. Operators

Universal Expression Language provides the following operators:

- Index property value: `[]`, `.`
- Change precedence: `()`
- Arithmetic: `+` (binary), `-` (binary), `*`, `/`, `div`, `%`, `mod`, `-` (unary)
- Logical: `and`, `&&`, `or`, `||`, `not`, `!`
- Relational: `==`, `eq`, `!=`, `ne`, `<`, `lt`, `>`, `gt`, `<=`, `le`, `>=`, `ge`
- Empty: `empty`

  Prefix operation that can be used to determine whether a value is null or empty.
- Conditional: `?`, `:`

Operators have the following precedence, from highest to lowest, and from left to right:

- `[] .`
- `()`
- `-` (unary) `not ! empty`
- `* / div % mod`
- `+` (binary) `-` (binary)
- `< > <= >= lt gt le ge`
- `== != eq ne`
- `&& and`
- `|| or`
- `? :`

10.1.7. System Properties and Environment Variables

You can use expressions to retrieve Java system properties, and to retrieve environment variables.

For system properties, `${system['property']}` yields the value of *property*, or `null` if there is no value for *property*. For example, `${system['user.home']}` yields the home directory of the user running the application server for Microgateway.

For environment variables, `${env['variable']}` yields the value of *variable*, or `null` if there is no value for *variable*. For example, `${env['HOME']}` yields the home directory of the user running the application server for Microgateway.

10.1.8. Token Resolution

Runtime expressions have access to evaluated configuration tokens described in "*JSON Evaluation*". For example, the following boolean expression returns `true` if the configuration token `my.status.code` resolves to `200`:

```
${integer(_token.resolve('my.status.code', '404')) == 200}
```

10.1.9. Functions

A number of built-in functions described in "*Functions*" can be called within an expression.

The syntax is `${function(parameter, ...)}`, where zero or more parameters are supplied to the function. For example:

- `"${bool(env['ENABLE_TIMER'])}"` recovers the environment variable `"ENABLE_TIMER"` and transforms it into a boolean
- `"${toLowerCase(request.method)}"` yields the method of the request, converted to lower case.

Functions can be operands for operations, and can yield parameters for other function calls.

10.1.10. Escaping Literal Expressions

The character `\` is treated as an escape character when it is followed by `${` or `#{`. For example, the expression `${true}` normally evaluates to `true`. To include the string `${true}` in an expression, write `${true}`

When the character `\` is followed by any other character sequence, it is not treated as an escape character. For example, `${` evaluates to `${`, but `\$` evaluates to `\$`.

10.1.11. Embedding Expressions

Although an expression cannot be embedded as `${expression}` inside another expression, embedding system property, environment variable, and function expressions within each other is fine. Do not enclose the embedded elements in `${}`.

10.1.12. Examples

```
"${request.uri.path == '/wordpress/wp-login.php'
   and request.form['action'][0] != 'logout'}"

"${request.uri.host == 'wiki.example.com'}"

"${request.cookies[keyMatch(request.cookies,'^SESS.*')][0].value}"

"${toString(request.uri)}"

"${request.method == 'POST' and request.uri.path == '/wordpress/wp-login.php'}"

"${request.method != 'GET'}"

"${request.headers['cookie'][0]}"

"${request.uri.scheme == 'http'}"

"${not (response.status.code == 302 and not empty session.gotoURL)}"

"${response.headers['Set-Cookie'][0]}"

"${request.headers['host'][0]}"

"${not empty system['my-variable'] ? system['my-variable'] : '/path/to'}/logs/gateway.log"
```

10.1.13. More Information

For more information, see "*Contexts*", "*Functions*", "*Request*", and "*Response*".

Chapter 10.2
Functions

10.2.1. Description

A set of built-in functions that can be called from within expressions, which are described in "*Expressions*".

10.2.2. array

```
array(strings...)
```

Returns an array of the strings given as argument.

Parameters

strings

Strings to put in the array.

Returns

array

Resulting array of containing the given strings.

10.2.3. boolean

```
bool(string)
```

Returns a Boolean with a value represented by the specified string.

The returned Boolean represents a true value if the string argument is not `null` and is equal to the string `"true"`, ignoring case.

Parameters

string

String containing the boolean representation.

Returns

Boolean

Boolean value represented by the string.

10.2.4. contains

```
contains(object, value)
```

Returns `true` if the object contains the specified value. If the object is a string, a substring is searched for the value. If the object is a collection or array, its elements are searched for the value.

Parameters

object

Object to search for.

value

Value to search for.

Returns

true

If the object contains the specified value.

10.2.5. decodeBase64

```
decodeBase64(string)
```

Returns the base64-decoded string, or `null` if the string is not valid base64.

Parameters

string

Base64-encoded string to decode.

Returns

string

Base64-decoded string.

10.2.6. decodeBase64url

```
decodeBase64url(string)
```

Returns the decoded value of the provided base64url-encoded string, or `null` if the string was not valid base64url.

Parameters

string

Base64url-encoded string to decode.

Returns

string

Base64url-decoded string.

10.2.7. encodeBase64

```
encodeBase64(string)
```

Returns the base64-encoded string, or `null` if the string is `null`.

Parameters

string

String to encode into base64.

Returns

string

Base64-encoded string.

10.2.8. encodeBase64url

```
encodeBase64url(string)
```

Returns the base64url-encoded string, or `null` if the string is `null`.

Parameters

string

String to encode into base64url.

Returns

string

Base64url-encoded string.

10.2.9. fileToUrl

```
fileToUrl(file)
```

Converts a `java.io.File` into a string representation for the URL of the file or directory.

Parameters

file

File or directory for which to build the URL.

For example, `${fileToUrl(openig.configDirectory)}/myProperties.json`.

Returns

file

String representation for the URL of the file or directory, or `null` if the file or directory is `null`.

For example, `file:///home/gcostanza/.openig/config/myProperties.json`.

10.2.10. formDecodeParameterNameOrValue

```
formDecodeParameterNameOrValue(string)
```

Returns the string that results from decoding the provided form encoded parameter name or value as per `application/x-www-form-urlencoded`, which can be `null` if the input is `null`.

Parameters

string

Parameter name or value.

Returns

string

String resulting from decoding the provided form encoded parameter name or value as per `application/x-www-form-urlencoded`.

10.2.11. formEncodeParameterNameOrValue

```
formEncodeParameterNameOrValue(string)
```

Returns the string that results from form encoding the provided parameter name or value as per `application/x-www-form-urlencoded`, which can be `null` if the input is `null`.

Parameters

string

Parameter name or value.

Returns

string

String resulting from form encoding the provided parameter name or value as per `application/x-www-form-urlencoded`.

10.2.12. indexOf

```
indexOf(string, substring)
```

Returns the index within a string of the first occurrence of a specified substring.

Parameters

string

String in which to search for the specified substring.

substring

Value to search for within the string.

Returns

number

Index of the first instance of substring, or -1 if not found.

The index count starts from 1, not 0.

10.2.13. integer

```
integer(string)
```

Transforms the string parameter into an integer. If the parameter is not a valid number in radix 10, returns null.

Parameters

string

String containing the integer representation.

Returns

integer

Integer value represented by the string.

10.2.14. integerWithRadix

```
integer(string, radix)
```

Uses the radix as the base for the string, and transforms the string into a base-10 integer. For example:

- `("20", 8)`: Transforms `20` in base `8`, and returns `16`.
- `("11", 16)` Transforms `11` in base `16`, and returns `17`.

If either parameter is not a valid number, returns null.

Parameters

string

String containing the integer representation, and an integer containing the radix representation.

Returns

integer

Integer value in base-10.

10.2.15. join

```
join(strings, separator)
```

Joins an array of strings into a single string value, with a specified separator.

Parameters

separator

Separator to place between joined elements.

strings

Array of strings to be joined.

Returns

string

String containing the joined strings.

10.2.16. keyMatch

```
keyMatch(map, pattern)
```

Returns the first key found in a map that matches the specified regular expression pattern, or `null` if no such match is found.

Parameters

map

Map whose keys are to be searched.

pattern

String containing the regular expression pattern to match.

Returns

string

First matching key, or `null` if no match found.

10.2.17. length

```
length(object)
```

Returns the number of items in a collection, or the number of characters in a string.

Parameters

object

Object whose length is to be determined.

Returns

number

Length of the object, or 0 if length could not be determined.

10.2.18. matchingGroups

```
matchingGroups(string, pattern)
```

Returns an array of matching groups for the specified regular expression pattern applied to the specified string, or `null` if no such match is found. The first element of the array is the entire match, and each subsequent element correlates to any capture group specified within the regular expression.

Parameters

string

String to be searched.

pattern

String containing the regular expression pattern to match.

Returns

array

Array of matching groups, or `null` if no such match is found.

10.2.19. matches

```
matches(string, pattern)
```

Returns `true` if the string contains a match for the specified regular expression pattern.

Parameters

string

String to be searched.

pattern

String containing the regular expression pattern to find.

Returns

true

String contains the specified regular expression pattern.

10.2.20. pathToUrl

```
pathToUrl(path)
```

Converts the given path into the string representation of its URL.

Parameters

path

Path of a file or directory as a string.

For example, `${pathToUrl(system['java.io.tmpdir'])}`.

Returns

string

String representation for the URL of the path, or `null` if the path is `null`.

For example, `file:///var/tmp`.

10.2.21. read

```
read(string)
```

Takes a file name as a string, interprets the content of the file with the UTF-8 character set, and returns the content of the file as a plain string.

Provides the absolute path to the file, or a path relative to the location of the Java system property `user.dir`.

Parameters

string

Name of the file to read.

Returns

string

Content of the file, or `null` on error.

10.2.22. readProperties

```
readProperties(string)
```

Takes a Java Properties file name as a `string`, and returns the content of the file as a key/value map of properties, or `null` on error (due to the file not being found, for example).

Either provide the absolute path to the file, or a path relative to the location of the Java system property `user.dir`.

For example, to get the value of the `key` property in the properties file `/path/to/my.properties`, use `${readProperties('/path/to/my.properties')['key']}`.

Parameters

string

Name of the Java Properties file to read.

Returns

object

Key/value map of properties or `null` on error.

10.2.23. readWithCharset

```
readWithCharset(string, charset)
```

Takes a file name as a string, interprets the content of the file with the specified Java character set, and returns the content of the file as a plain string.

Parameters

string

Name of the file to read.

Provides the absolute path to the file, or a path relative to the location of the Java system property `user.dir`.

charset

Name of a Java character set with which to interpret the file, as described in Class Charset.

Returns

string

Content of the file, or `null` on error.

10.2.24. split

```
split(string, pattern)
```

Splits the specified string into an array of substrings around matches for the specified regular expression pattern.

Parameters

string

String to be split.

pattern

Regular expression to split substrings around.

Returns

array

Resulting array of split substrings.

10.2.25. toJson

```
toJson(JSON string)
```

Converts a JSON string to a JSON structure.

Parameters

JSON string

JSON string representing a JavaScript object.

For example, the string value contained in `contexts.amSession.properties.userDetails` contains the JSON object `{"email":"test@example.com"}`.

Returns

JSON structure

JSON structure, or `null` on error.

In the expression `"${toJson(contexts.amSession.properties.userDetails).email}"`, the string value is treated as JSON, and the expression evaluates to `test@example.com`.

10.2.26. toLowerCase

```
toLowerCase(string)
```

Converts all of the characters in a string to lower case.

Parameters

string

String whose characters are to be converted.

Returns

string

String with characters converted to lower case.

10.2.27. toString

```
toString(object)
```

Returns the string value of an arbitrary object.

Parameters

object

Object whose string value is to be returned.

Returns

string

String value of the object.

10.2.28. toUpperCase

```
toUpperCase(string)
```

Converts all of the characters in a string to upper case.

Parameters

string

String whose characters are to be converted.

Returns

string

String with characters converted to upper case.

10.2.29. trim

```
trim(string)
```

Returns a copy of a string with leading and trailing whitespace omitted.

Parameters

string

String whose white space is to be omitted.

Returns

string

String with leading and trailing white space omitted.

10.2.30. urlDecode

```
urlDecode(string)
```

Returns the URL decoding of the provided string.

This is equivalent to "formDecodeParameterNameOrValue".

Parameters

string

String to be URL decoded, which may be `null`.

Returns

string

URL decoding of the provided string, or `null` if string was `null`.

10.2.31. urlEncode

```
urlEncode(string)
```

Returns the URL encoding of the provided string.

This is equivalent to "formEncodeParameterNameOrValue".

Parameters

string

String to be URL encoded, which may be `null`.

Returns

string

URL encoding of the provided string, or `null` if string was `null`.

10.2.32. urlDecodeFragment

```
urlDecodeFragment(string)
```

Returns the string that results from decoding the provided URL encoded fragment as per RFC 3986, which can be `null` if the input is `null`.

Parameters

string

URL encoded fragment.

Returns

string

String resulting from decoding the provided URL encoded fragment as per RFC 3986.

10.2.33. urlDecodePathElement

```
urlDecodePathElement(string)
```

Returns the string that results from decoding the provided URL encoded path element as per RFC 3986, which can be `null` if the input is `null`.

Parameters

string

The path element.

Returns

string

String resulting from decoding the provided URL encoded path element as per RFC 3986.

10.2.34. urlDecodeQueryParameterNameOrValue

```
urlDecodeQueryParameterNameOrValue(string)
```

Returns the string that results from decoding the provided URL encoded query parameter name or value as per RFC 3986, which can be `null` if the input is `null`.

Parameters

string

Parameter name or value.

Returns

string

String resulting from decoding the provided URL encoded query parameter name or value as per RFC 3986.

10.2.35. urlDecodeUserInfo

```
urlDecodeUserInfo(string)
```

Returns the string that results from decoding the provided URL encoded userInfo as per RFC 3986, which can be `null` if the input is `null`.

Parameters

string

URL encoded userInfo.

Returns

string

String resulting from decoding the provided URL encoded userInfo as per RFC 3986.

10.2.36. urlEncodeFragment

```
urlEncodeFragment(string)
```

Returns the string that results from URL encoding the provided fragment as per RFC 3986, which can be `null` if the input is `null`.

Parameters

string

Fragment.

Returns

string

The string resulting from URL encoding the provided fragment as per RFC 3986.

10.2.37. urlEncodePathElement

```
urlEncodePathElement(string)
```

Returns the string that results from URL encoding the provided path element as per RFC 3986, which can be `null` if the input is `null`.

Parameters

string

Path element.

Returns

string

String resulting from URL encoding the provided path element as per RFC 3986.

10.2.38. urlEncodeQueryParameterNameOrValue

```
urlEncodeQueryParameterNameOrValue(string)
```

Returns the string that results from URL encoding the provided query parameter name or value as per RFC 3986, which can be `null` if the input is `null`.

Parameters

string

Parameter name or value.

Returns

string

String resulting from URL encoding the provided query parameter name or value as per RFC 3986.

10.2.39. urlEncodeUserInfo

```
urlEncodeUserInfo(string)
```

Returns the string that results from URL encoding the provided userInfo as per RFC 3986, which can be `null` if the input is `null`.

Parameters

string

userInfo.

Returns

string

String resulting from URL encoding the provided userInfo as per RFC 3986.

10.2.40. More Information

Some functions are provided by org.forgerock.openig.el.Functions.

Other functions are provided by org.forgerock.http.util.Uris.

Chapter 10.3
Patterns

10.3.1. Description

Patterns in configuration parameters and expressions use the standard Java regular expression Pattern class. For more information on regular expressions, see Oracle's tutorial on Regular Expressions.

10.3.2. Pattern Templates

A regular expression pattern template expresses a transformation to be applied for a matching regular expression pattern. It may contain references to capturing groups within the match result. Each occurrence of `$g` (where *g* is an integer value) is substituted by the indexed capturing group in a match result. Capturing group zero `"$0"` denotes the entire pattern match. A dollar sign or numeral literal immediately following a capture group reference can be included as a literal in the template by preceding it with a backslash (`\`). Backslash itself must be also escaped in this manner.

10.3.3. More Information

Java Pattern class

Regular Expressions tutorial

Part 11

Scripts

Use scripts with the following scriptable object types:

- "*ScriptableFilter*", to customize flow of requests and responses
- "*ScriptableHandler*", to customize creation of responses
- "*ScriptableThrottlingPolicy*", to customize throttling rates
- "*ScriptableAccessTokenResolver*" to customize resolution and validation of OAuth 2.0 access tokens
- `ScriptableResourceAccess` in "*OAuth2ResourceServerFilter*", to customize the list of OAuth 2.0 scopes required in an OAuth 2.0 access_token

After updating a script that is used in a route, leave at least one second before processing a request. The Groovy interpreter needs time to detect and take the update into account.

Important

When you are writing scripts or Java extensions, never use a `Promise` blocking method, such as `get()`, `getOrThrow()`, or `getOrThrowUninterruptibly()`, to obtain the response.

A promise represents the result of an asynchronous operation. Therefore, using a blocking method to wait for the result can cause deadlocks and/or race issues.

Chapter 11.1
Scripts

11.1.1. Usage

```
  "name": string,
  "type": scriptable object type,
  "config": {
    "type": string,
    "file": expression,                     // Use either "file"
    "source": string or array of strings,   // or "source", but not both.
    "args": object,
    "clientHandler": Handler reference
  }
}
```

11.1.2. Properties

`"type"`: *string, required*

The Internet media type (formerly MIME type) of the script, `"application/x-groovy"` for Groovy

`"file"`: *expression*

Path to the file containing the script; mutually exclusive with `"source"`.

Relative paths are with respect to to the base location for scripts. The base location depends on the configuration.

The base location for Groovy scripts is on the classpath when the scripts are executed. If some Groovy scripts are not in the default package, but instead have their own package names, they belong in the directory corresponding to their package name. For example, a script in package `com.example.groovy` belongs under *`openig-base`*`/scripts/groovy/com/example/groovy/`.

`"source"`: *string or array of strings, required if `"file"` is not used*

The script as a string or array of strings; mutually exclusive with `"file"`.

The following example shows the source of a script as an array of strings:

```
"source": [
  "Response response = new Response(Status.OK)",
  "response.entity = 'foo'",
  "return response"
]
```

"args": *map, optional*

Parameters passed from the configuration to the script.

- The following example configures arguments as a map whose values can are scalars, arrays, and objects:

```
{
  "args": {
    "title": "Coffee time",
    "status": 418,
    "reason": [
      "Not Acceptable",
      "I'm a teapot",
      "Acceptable"
    ],
    "names": {
      "1": "koffie",
      "2": "kafe",
      "3": "cafe",
      "4": "kafo"
    }
  }
}
```

  A script can access the args parameters in the same way as other global objects. The following example sets the response status to `I'm a teapot`:

```
response.status = Status.valueOf(418, reason[1])
```

  For information about the 418 status code see RFC 7168, Section 2.3.3 *418 I'm a Teapot* .

- The following example configures arguments as strings and numbers for a ScriptableThrottlingPolicy:

```
"args": {
  "status": "gold",
  "rate": 6,
  "duration": "10 seconds"
}
```

  The following lines set the throttling rate to 6 requests each 10 seconds when the response status is `gold`:

```
if (attributes.rate.status == status) {
  return new ThrottlingRate(rate, duration)
}
```

- The following example configures arguments that reference a SampleFilter defined in the heap:

```
{
  "heap": [
    {
      "name": "SampleFilter",
      "type": "SampleFilter",
      "config": {
        "name": "X-Greeting",
        "value": "Hello world"
      }
    }
  ]
}
```

The following line uses an expression in the args parameter to pass SampleFilter to the script:

```
{
  "args": {
    "filter": "${heap['SampleFilter']}"
  }
}
```

The script can then reference SampleFilter as `filter`.

`"clientHandler"`, *ClientHandler reference, optional*

A Handler for making outbound HTTP requests to third-party services. In a script, `clientHandler` is wrapped within the global object `http`.

Default: The default ClientHandler.

For details, see "Handlers".

For more information, see org.forgerock.http.Client.

11.1.3. Available Objects

The following global objects are available to scripts:

Any parameters passed as args

You can use the configuration to pass parameters to the script by specifying an args object.

The args object is a map whose values can be scalars, arrays, and objects. The args object can reference objects defined in the heap by using expressions, for example, `"${heap['ObjectName']}"`.

The values for script arguments can be defined as configuration expressions, and evaluated at configuration time.

Script arguments cannot refer to `context` and `request`, but `context` and `request` variables can be accessed directly within scripts.

Take care when naming keys in the args object. If you reuse the name of another global object, cause the script to fail and Microgateway to return a response with HTTP status code 500 Internal Server Error.

All heap objects

The heap object configuration, described in "*Heap Objects*".

`openig`

An implicit object that provides access to the environment when expressions are evaluated.

`attributes`

The attributes object provides access to a context map of arbitrary attributes, which is a mechanism for transferring transient state between components when processing a single request.

Use `session` for maintaining state between successive requests from the same logical client.

`context`

The processing context.

This context is the leaf of a chain of contexts. It provides access to other Context types, such as SessionContext, AttributesContext, and ClientContext, through the `context.asContext(ContextClass.class)` method.

`contexts`

a map<string, context> object. For information, see "*Contexts*".

`request`

The HTTP request.

`globals`

This object is a Map that holds variables that persist across successive invocations.

`http`

An embedded client for making outbound HTTP requests, which is an org.forgerock.http.Client.

If a `"clientHandler"` is set in the configuration, then that Handler is used. Otherwise, the default ClientHandler configuration is used.

For details, see "Handlers".

`ldap`

The ldap object provides an embedded LDAP client.

Use this client to perform outbound LDAP requests, such as LDAP authentication.

`logger`

The logger object provides access to a unique SLF4J logger instance for scripts, where the logger instance is named with the script name.

`next`

The object named `next` refers to the next element in the chain, which can be the following filter or the terminal handler. If the next object in the chain is a filter, Microgateway wraps it in a handler.

`session`

The session object provides access to the session context, which is a mechanism for maintaining state when processing a successive requests from the same logical client or end user.

Use `attributes` for transferring transient state between components when processing a single request.

11.1.4. Imported Classes

The following classes are imported automatically for Groovy scripts:

- org.forgerock.http.Client
- org.forgerock.http.Filter
- org.forgerock.http.Handler
-] org.forgerock.http.filter.throttling.ThrottlingRate
- org.forgerock.http.util.Uris
- org.forgerock.util.AsyncFunction
- org.forgerock.util.Function
- org.forgerock.util.promise.NeverThrowsException
- org.forgerock.util.promise.Promise
- org.forgerock.services.context.Context
- org.forgerock.http.protocol.*
- org.forgerock.http.oauth2.AccessTokenInfo
- org.forgerock.json.JsonValue, and all its static methods, including `json(Object)`, `array(Object...)`, `object(fields...)`, and `field(String, Object)`

11.1.5. More Information

- "*ScriptableFilter*", org.forgerock.openig.filter.ScriptableFilter, and org.forgerock.http.Filter
- "*ScriptableHandler*", org.forgerock.openig.handler.ScriptableHandler, and org.forgerock.http.Handler
- "*ScriptableThrottlingPolicy*", org.forgerock.openig.filter.throttling.ScriptableThrottlingPolicy.Heaplet, and org.forgerock.http.filter.throttling.ThrottlingPolicy
- `ScriptableResourceAccess` in "*OAuth2ResourceServerFilter*", org.forgerock.openig.filter.oauth2.ScriptableResourceAccess, and org.forgerock.http.oauth2.ResourceAccess
- `ScriptableAccessTokenResolver` in "*OAuth2ResourceServerFilter*", org.forgerock.openig.filter.oauth2.ScriptableAccessTokenResolver, and org.forgerock.http.oauth2.AccessTokenResolver

Part 12
Properties

Configuration parameters can be declared as properties in the Microgateway configuration or in an external JSON file.

Chapter 12.1
Properties

12.1.1. Description

Configuration parameters, such as host names, port numbers, and directories, can be declared as property variables in the Microgateway configuration or in an external JSON file. The variables can then be used in expressions in routes and in `config.json` to set the value of configuration parameters.

Properties can be inherited across the router, so a property defined in `config.json` can be used in any of the routes in the configuration.

Storing the configuration centrally and using variables for parameters that can be different for each installation makes it easier to deploy Microgateway in different environments without changing a single line in your route configuration.

12.1.2. Usage

12.1.2.1. Simple Property Configured Inline

```
{
  "properties": {
    "<variable name>": "valid JSON value"
  }
}
```

12.1.2.2. Group Property Configured Inline

```
{
  "properties": {
    "<group name>": {
      ["<variable name>": "valid JSON value", ... ]
    }
  }
}
```

12.1.2.3. Properties Configured in One or More External Files

```
{
  "properties": {
    "$location": expression
  }
}
```

In this example, `description1` and `description2` prefix the variable names contained in the external file.

```
{
  "properties": {
    "description1" {
      "$location": expression
    {
    "description2" {
      "$location": expression
    {
  }
}
```

12.1.3. Properties

`"<variable name>"`: *string*

The name of a variable to use in the Microgateway configuration. The variable can be used in expressions in routes or in `config.json` to assign the value of a configuration parameter.

The value assigned to the variable can be any valid JSON value: string, number, boolean, array, object, or null.

`"<group name>"`: *string, required*

The name of a group of variables to use in the Microgateway configuration. The group name and variable name are combined using dot notation in an expression.

In the following example from `config.json`, the property group `directories` contains two variables that define the location of files:

```
{
  "properties": {
    "directories": {
      "config": "${openig.configDirectory.path}",
      "auditlog": "/tmp/logs"
    }
  }
}
```

The group name and variable name are combined using dot notation in the following example to define the directory where the audit log is stored:

```
{
  "type": "AuditService",
  "config": {
    "eventHandlers": [
      {
        "class": "org.forgerock.audit.handlers.csv.CsvAuditEventHandler",
        "config": {
          "name": "csv",
          "logDirectory": "${directories.auditlog}",
          . . .
```

`"$location"`: *expression, required*

The location and name of one or more JSON files where property variables are configured.

Files must be `.json` files, and contain property variables with a key/value format, where the key cannot contain the period (`.`) separator.

For example, this file is correct:

```
{
  "openamLocation": "http://openam.example.com:8088/openam/",
  "portNumber": 8081
}
```

This file would cause an error:

```
{
  "openam.location": "http://openam.example.com:8088/openam/",
  "port.number": 8081
}
```

Property Variables Configured In One File

In the following example, the location of the file that contains the property variables is defined as an expression:

```
{
  "properties": {
    "$location": "${fileToUrl(openig.configDirectory)}/myProperties.json"
  }
}
```

In the following example, the location of the file that contains the property variables is defined as a string:

```
{
  "properties": {
    "$location": "file:///Users/user-id/.openig/config/myProperties.json"
  }
}
```

The file location can be defined as any real URL.

The file `myProperties.json` contains the base URL of an AM service and the port number of an application.

```
{
  "openamLocation": "http://openam.example.com:8088/openam/",
  "appPortNumber": 8081
}
```

Properties Variables Configured In Multiple Files

In the following example, the property variables are contained in two files, defined as a set of strings:

```
{
  "properties": {
    "urls": {
      "$location": "file://path-to-file/myUrlProperties.json"
    },
    "ports": {
       "$location": "file://path-to-file/myPortProperties.json"
    }
  }
}
```

The file `myUrlProperties.json` contains the base URL of the sample application:

```
{
  "appUrl": "http://app.example.com"
}
```

The file `myPortProperties.json` contains the port number of an application:

```
{
  "appPort": 8081
}
```

The base config file, `config.json`, can use the properties as follows:

```
{
  "properties": {
    "urls": {
      "$location": "file:///Users/user-id/.openig/config/myUrlProperties.json"
    },
    "ports": {
      "$location": "file:///Users/user-id/.openig/config/myPortProperties.json"
    }
  },
  "handler": {
    "type": "Router",
    "name": "_router",
    "baseURI": "${urls.appUrl}:${ports.appPort}",
    . . .
```

Part 13

Requests, Responses, and Contexts

This part of the reference describes the Microgateway object model. The top-level objects are request, response, and contexts.

Contexts provide contextual information about the handled request, such as information about the client making the request, the session, the authentication or authorization identity of the principal, and any other state information associated with the request. Contexts provide a means to access state information throughout the duration of the HTTP session between the client and protected application, including when this involves interaction with additional services.

Each filter can add to the contextual information by enriching the existing context (for example, by storing objects in sessions or attributes), or by providing a new context tailored for a specific purpose.

Unlike session information, which spans multiple request/response exchanges, contexts last only for the duration of the request/response exchange, and are then lost.

Chapter 13.1

AttributesContext

13.1.1. Description

Provides a map for request attributes. When Microgateway processes a single request, it injects transient state information about the request into this context. Attributes stored when processing one request are not accessible when processing a subsequent request.

Microgateway automatically provides access to the `attributes` field through the `attributes` bindings in expressions. For example, to access a username with an expression, use `${attributes.credentials.username}` instead of `${contexts.attributes.attributes.credentials.username}`

Use "*SessionContext*" to maintain state between successive requests from the same logical client.

13.1.2. Properties

The context is named `attributes`, and is accessible at `${attributes}`. The context has the following property:

`"attributes"`: map

Map of information conveyed between filters and handlers. Cannot be null.

13.1.3. More Information

org.forgerock.services.context.AttributesContext

Chapter 13.2

CapturedUserPasswordContext

13.2.1. Description

Provides the decrypted AM password of the current user. When the "*CapturedUserPasswordFilter*" processes a request, it injects the decrypted password from AM into this context.

13.2.2. Properties

The context is named `capturedPassword`, and is accessible at `${contexts.capturedPassword}`. The context has the following properties:

`"raw"`: byte[]

The decrypted password as bytes.

`"value"`: string

The decrypted password as a UTF-8 string.

13.2.3. More Information

org.forgerock.openig.openam.CapturedUserPasswordContext

Chapter 13.3
ClientContext

13.3.1. Description

Information about the client sending a request. When Microgateway receives a request, it injects information about the client sending the request into this context.

13.3.2. Properties

The context is named `client`, and is accessible at `${contexts.client}`. The context has the following properties:

`"certificates"`: array

List of X.509 certificates presented by the client

If the client does not present any certificates, Microgateway returns an empty list.

Never `null`.

`"isExternal"`: boolean

True if the client connection is external.

`"isSecure"`: boolean

True if the client connection is secure.

`"localAddress"`: string

The IP address of the interface that received the request

`"localPort"`: number

The port of the interface that received the request

`"remoteAddress"`: string

The IP address of the client (or the last proxy) that sent the request

`"remotePort"`: number

The source port of the client (or the last proxy) that sent the request

`"remoteUser"`: string

The login of the user making the request, or `null` if unknown

This is likely to be `null` unless you have deployed Microgateway with a non-default deployment descriptor that secures the Microgateway web application.

`"userAgent"`: string

The value of the User-Agent HTTP header in the request if any, otherwise `null`

13.3.3. More Information

org.forgerock.services.context.ClientContext

Chapter 13.4

Contexts

13.4.1. Description

The root object for request context information.

Contexts is a map of available contexts, which implement the Context interface. The contexts map's keys are strings and the values are context objects. A context holds type-safe information useful for processing requests and responses. The `contexts` map is populated dynamically when creating bindings for evaluation of expressions and scripts.

All context objects use their version of the following properties:

"*context-Name*": string

Name of the context.

"*context-ID*": string

Read-only string uniquely identifying the context object.

"*context-rootContext*": boolean

True if the context object is a RootContext (has no parent).

"*context-Parent*": Context object

Parent of this context object.

13.4.2. Properties

The contexts object can provide access to the following contexts for each request:

- "*AttributesContext*"
- "*ClientContext*"
- "*SessionContext*"
- "*UriRouterContext*"
- "*TransactionIdContext*"

The contexts object can provide access to the following contexts when related filters are used:

- "*CapturedUserPasswordContext*"
- "*CdSsoContext*"
- "*CdSsoFailureContext*"
- "*JwtBuilderContext*"
- "*OAuth2Context*"
- "*PolicyDecisionContext*"
- "*SessionInfoContext*"
- "*SsoTokenContext*"
- "*StsContext*"
- "*UserProfileContext*"

13.4.3. More Information

org.forgerock.services.context.Context

Chapter 13.5
CdSsoContext

13.5.1. Description

Provides the cross-domain SSO properties for the CDSSO token, the user ID of the session, and the full claims set. When the "*CrossDomainSingleSignOnFilter*" processes a request, it injects the information in this context.

13.5.2. Properties

The context is named `cdsso`, and is accessible at `${contexts.cdsso}`. The context has the following properties:

`"claimsSet"`: *JwtClaimsSet object, required*

Full claims set for the identity of the authenticated user. Cannot be null.

`"cookieInfo"`: *object*

Configuration data for the CDSSO authentication cookie, with the following attributes:

- `name`: Cookie name (string)
- `domain`: Cookie domain (optional string)
- `path`: Cookie path (string)

None of the attributes can be null.

`redirectEndpoint"`: *string*

Redirect endpoint URI configured for communication with AM. Cannot be null.

`"sessionUid"`: *string*

Universal session ID. Cannot be null.

`"token"`: *string*

Value of the CDSSO token. Cannot be null.

13.5.3. More Information

org.forgerock.openig.openam.CdSsoContext

Chapter 13.6

CdSsoFailureContext

13.6.1. Description

Contains the error details for any error that occurred during cross-domain SSO authentication. When the "*CrossDomainSingleSignOnFilter*" processes a request, should an error occur that prevents authentication, the error details are captured in this context.

13.6.2. Properties

The context is named `cdssoFailure`, and is accessible at `${contexts.cdssoFailure}`. The context has the following properties:

`"error"`: *The error (string)*

The error that occurred during authentication. Cannot be null.

`"description"`: *Error description (string)*

A description of the error that occurred during authentication. Cannot be null.

`"throwable"`: *Throwable*

Any `Throwable` associated with the error that occured during authentication. Can be null.

13.6.3. More Information

org.forgerock.openig.openam.CdSsoFailureContext

Chapter 13.7
JwtBuilderContext

13.7.1. Description

When the "*JwtBuilderFilter*" processes a request, it stores provided data in this context. This context returns the JWT as string for downstream use.

13.7.2. Properties

The context is named `jwtBuilder`, and is accessible at `${contexts.jwtBuilder}`. The context has the following properties:

`"value"`: *string*

The base64url encoded UTF-8 parts of the JWT, containing name-value pairs of data. Cannot be null.

13.7.3. More Information

org.forgerock.openig.filter.JwtBuilderFilter

org.forgerock.openig.filter.JwtBuilderContext

Chapter 13.8
OAuth2Context

13.8.1. Description

Provides OAuth 2.0 access tokens. When the "*OAuth2ResourceServerFilter*" processes a request, it injects the access token into this context.

13.8.2. Properties

The context name is `oauth2`, and the context has the following property:

`"accessToken"`: AccessTokenInfo

The AccessTokenInfo is built from the following properties:

`"info"`: (map<string>, object)

Raw JSON as a map.

`"token"`: string

Access token identifier issued from the authorization server.

`"scopes"`: space separated string

Scopes associated to this token.

`"expiresAt"`: long

Timestamp (in milliseconds since epoch) when the token expires.

13.8.3. More Information

org.forgerock.http.oauth2.OAuth2Context

org.forgerock.http.oauth2.AccessTokenInfo

Chapter 13.9
PolicyDecisionContext

13.9.1. Description

Provides attributes and advices returned by AM policy decisions. When the "*PolicyEnforcementFilter*" processes a request, it injects the attributes and advices into this context.

13.9.2. Properties

The context is named `policyDecision`, and is accessible at `${contexts.policyDecision}`. The context has the following properties:

`"attributes"`: unmodifiable map

The map of attributes provided in the policy decision. Can be empty, but not null.

`"jsonAttributes"`: JsonValue

The map of attributes provided in the policy decision. Cannot be null.

`"advices"`: map

The map of advices provided in the policy decision. Can be empty, but not null.

`"jsonAdvices"`: JsonValue

The map of advices provided in the policy decision. Cannot be null.

13.9.3. More Information

org.forgerock.openig.openam.PolicyDecisionContext.html

Chapter 13.10

Request

13.10.1. Description

An HTTP request message.

13.10.2. Properties

`"method"`: *string*

The method to be performed on the resource. Example: `"GET"`.

`"uri"`: *object*

The fully-qualified URI of the resource being accessed. Example: `"http://www.example.com/resource.txt"`.

See also "*URI*".

`"version"`: *string*

Protocol version. Example: `"HTTP/1.1"`.

`"headers"`: *object*

Exposes message header fields as name-value pairs, where name is header name and value is an array of header values.

`"cookies"`: *object*

Exposes incoming request cookies as name-value pairs, where name is cookie name and value is an array of string cookie values.

`"form"`: *object*

Exposes query parameters and/or `application/x-www-form-urlencoded` entity as name-value pairs, where name is the field name and value is an array of string values.

`"entity"`: *object*

The message entity body.

Methods are provided for accessing the entity as byte, string, or JSON content. For information, see Entity.

13.10.3. More Information

org.forgerock.http.protocol.Request

org.forgerock.http.protocol.Entity

Chapter 13.11

Response

13.11.1. Description

An HTTP response message.

13.11.2. Properties

`"cause"`: *Exception object*

The cause of an error if the status code is in the range 4xx-5xx. Possibly null.

`"status"`: *Status object*

The response status.

For details, see "*Status*".

`"version"`: *string*

Protocol version. Example: `"HTTP/1.1"`.

`"headers"`: *object*

Exposes message header fields as name-value pairs, where name is header name and value is an array of header values.

`"entity"`: *object*

The message entity body.

Methods are provided for accessing the entity as byte, string, or JSON content. For information, see Entity.

13.11.3. More Information

org.forgerock.http.protocol.Response

org.forgerock.http.protocol.Entity

Chapter 13.12

SessionContext

13.12.1. Description

Provides access to information about stateful and stateless sessions.

To process a single request, consider using "*AttributesContext*" to transfer transient state between components and prevent Microgateway from creating additional sessions.

Microgateway automatically provides access to the `session` field through the `session` bindings in expressions. For example, to access a username with an expression, use `${session.username}` instead of `${contexts.session.session.username}`

13.12.2. Properties

The context is named `session`, and is accessible at `${contexts.session}`. The context has the following properties:

`"session"`: map

A map of attributes that are name-value pairs of the format `Map<String, Object>`.

Any object type can be stored in the session.

13.12.3. More Information

org.forgerock.http.session.SessionContext

Chapter 13.13

SessionInfoContext

13.13.1. Description

Provides AM session information and properties. When the "*SessionInfoFilter*" processes a request, it injects info and properties from the AM session into this context.

13.13.2. Properties

The context is named `amSession`, and is accessible at `${contexts.amSession}`. The context has the following properties:

`"asJsonValue()"`: JsonValue

Raw JSON.

`"latestAccessTime"`: instant

The timestamp of when the session was last used. Can be null if the DN is not resident on the SSO token, or if the time cannot be obtained from the session.

`"maxIdleExpirationTime"`: instant

The timestamp of when the session would time out for inactivity. Can be null if the DN is not resident on the SSO token, or if the time cannot be obtained from the session.

`"maxSessionExpirationTime"`: instant

The timestamp of when the session would time out regardless of activity. Can be null if the DN is not resident on the SSO token, or if the time cannot be obtained from the session.

`"properties"`: map

The read-only map of properties bound to the session. Can be empty, but not null.

The following properties are retrieved:

- When `sessionProperties` in AmService is configured, listed session properties with a value.
- When `sessionProperties` in AmService is not configured, all session properties with a value.

- Properties with a value that are required by Microgateway but not specified by `sessionProperties` in AmService. For example, when the session cache is enabled, session properties related to the cache are automatically retrieved.

Properties with a value are returned, properties with a null value are not returned.

`"realm"`: string

The realm as specified by AM, in a user-friendly slash (/) separated format. Can be null if the DN is not resident on the SSO token.

`"sessionHandle"`: string

The handle to use for logging out of the session. Can be null if the handle is not available for the session.

`"universalId"`: string

The DN that AM uses to uniquely identify the user. Can be null if it cannot be obtained from the SSO token.

`"username"`: string

A user-friendly version of the username. Can be null if the DN is not resident on the SSO token, or empty if it cannot be obtained from the DN.

13.13.3. More Information

org.forgerock.openig.openam.SessionInfoContext

Chapter 13.14
SsoTokenContext

13.14.1. Description

Provides SSO tokens and their validation information. When the "*SingleSignOnFilter*" processes a request, it injects the value of the SSO token and additional information in this context.

13.14.2. Properties

The context is named `ssoToken`, and is accessible at `${contexts.ssoToken}`. The context has the following properties:

`"info"`: map

Information associated with the SSO token, such as `realm` or `uid`. Cannot be null.

`"loginEndpoint"`: url

URL for the login endpoint, evaluated from the configuration of SingleSignOnFilter.

`"value"`: string

The value of the SSO token. Cannot be null.

13.14.3. More Information

org.forgerock.openig.openam.SsoTokenContext

Chapter 13.15

Status

13.15.1. Description

Represents an HTTP response status. For details, see *RFC 7231: HTTP/1.1 Semantics and Content*, Section 6.1. Overview of Status Codes .

13.15.2. Properties

`"code"`: *integer*

Three-digit integer reflecting the HTTP status code.

`"family"`: *enum*

Family Enum value representing the class of response that corresponds to the code:

`Family.INFORMATIONAL`

Status code reflects a provisional, informational response: 1xx.

`Family.SUCCESSFUL`

The server received, understood, accepted and processed the request successfully. Status code: 2xx.

`Family.REDIRECTION`

Status code indicates that the client must take additional action to complete the request: 3xx.

`Family.CLIENT_ERROR`

Status code reflects a client error: 4xx.

`Family.SERVER_ERROR`

Status code indicates a server-side error: 5xx.

`Family.UNKNOWN`

Status code does not belong to one of the known families: 600+.

`"reasonPhrase"`: ***string***

The human-readable reason-phrase corresponding to the status code.

For details, see *RFC 7231: HTTP/1.1 Semantics and Content*, Section 6.1. Overview of Status Codes .

`"isClientError"`: ***boolean***

True if Family.CLIENT_ERROR.

`"isInformational"`: ***boolean***

True if Family.INFORMATIONAL.

`"isRedirection"`: ***boolean***

True if Family.REDIRECTION.

`"isServerError"`: ***boolean***

True if Family.SERVER_ERROR.

`"isSuccessful"`: ***boolean***

True if Family.SUCCESSFUL.

13.15.3. More Information

org.forgerock.http.protocol.Status

Chapter 13.16

StsContext

13.16.1. Description

Provides the result of a token transformation. When the "*TokenTransformationFilter*" processes a request, it injects the result into this context.

13.16.2. Properties

The context is named `sts`, and is accessible at `${contexts.sts}`. The context has the following properties:

`"issuedToken"`: string

The result of the token transformation.

13.16.3. More Information

org.forgerock.openig.openam.StsContext

Chapter 13.17

TransactionIdContext

13.17.1. Description

The transaction ID of a request. When Microgateway receives a request, it injects the transaction ID into this context.

13.17.2. Properties

The context is named `transactionId`, and is accessible at `${contexts.transactionId}`. The context has the following properties:

`"transactionId"`: TransactionId

The ID of the transaction.

13.17.3. More Information

org.forgerock.services.TransactionIdContext

org.forgerock.services.context.TransactionIdContext

Chapter 13.18
URI

13.18.1. Description

Represents a Uniform Resource Identifier (URI) reference.

13.18.2. Properties

`"scheme"`: *string*

The scheme component of the URI, or `null` if the scheme is undefined.

`"authority"`: *string*

The decoded authority component of the URI, or `null` if the authority is undefined.

Use "rawAuthority" to access the raw (encoded) component.

`"userInfo"`: *string*

The decoded user-information component of the URI, or `null` if the user information is undefined.

Use "rawUserInfo" to access the raw (encoded) component.

`"host"`: *string*

The host component of the URI, or `null` if the host is undefined.

`"port"`: *number*

The port component of the URI, or `null` if the port is undefined.

`"path"`: *string*

The decoded path component of the URI, or `null` if the path is undefined.

Use "rawPath" to access the raw (encoded) component.

`"query"`: *string*

The decoded query component of the URI, or `null` if the query is undefined.

Note

The query key and value is decoded. However, because a query value can be encoded more than once in a redirect chain, even though it is decoded it can contain unsafe ASCII characters.

Use "rawQuery" to access the raw (encoded) component.

`"fragment"`: *string*

The decoded fragment component of the URI, or `null` if the fragment is undefined.

Use "rawFragment" to access the raw (encoded) component.

13.18.3. More Information

org.forgerock.http.MutableUri

Chapter 13.19
UriRouterContext

13.19.1. Description

Provides routing information associated with a request. When Microgateway routes a request, it injects information about the routing into this context.

13.19.2. Properties

The context is named `router`, and is accessible at `${contexts.router}`. The context has the following properties:

`"baseUri"`: string

The portion of the request URI which has been routed so far.

`"matchedUri"`: string

The portion of the request URI that matched the URI template.

`"originalUri"`: URI

The original target URI for the request, as received by the web container.

The value of this field is read-only.

`"remainingUri"`: string

The portion of the request URI that is remaining to be matched.

`"uriTemplateVariables"`: map

An unmodifiable map, where the keys and values are strings. The map contains the parsed URI template variables keyed on the URI template variable name.

13.19.3. More Information

org.forgerock.http.routing.UriRouterContext

Chapter 13.20
UserProfileContext

13.20.1. Description

When the "*UserProfileFilter*" processes a request, it injects the user profile information into this context. This context provides raw JSON representation, and convenience accessors that map commonly used LDAP field names to a context names.

13.20.2. Properties

The context is named `userProfile`, and is accessible at `${contexts.userProfile}`. The context has the following properties:

`"username"`: string

User-friendly version of the username. This field is always fetched. If the underlying data store doesn't include `username`, this field is null.

Example of use: `${contexts.userProfile.username}`

`"realm"`: string

Realm as specified by AM, in a user-friendly slash (/) separated format. Can be null.

Example of use: `${contexts.userProfile.realm}`

`"distinguishedName"`: string

Distinguished name of the user. Can be null.

Example of use: `${contexts.userProfile.distinguishedName}`

`"commonName"`: string

Common name of the user. Can be null.

Example of use: `${contexts.userProfile.commonName}`

`"rawInfo"`: (map<string>, object)

Unmodifiable map of the user profile information.

This context contains the object structure of the AM user profile. Any individual field can be retrieved from the map. Depending on the requested fields, the context can be empty or values can be null.

Examples of use: `${contexts.userProfile.rawInfo}`, `${contexts.userProfile.rawInfo.username}`, `${contexts.userProfile.rawInfo.employeeNumber[0]}`.

`"asJsonValue()"`: JsonValue

Raw JSON of the user profile information.

Example of use: `${contexts.userProfile.asJsonValue()}`

13.20.3. More Information

org.forgerock.openig.openam.UserProfileContext

"*UserProfileFilter*"

Part 14

Access Token Resolvers

This section describes the objects available for OAuth 2.0 access_token resolution.

Chapter 14.1

TokenIntrospectionAccessTokenResolver

14.1.1. Description

In OAuth2ResourceServerFilter, use the token introspection endpoint, `/oauth2/introspect`, to resolve access tokens and retrieve metadata about the token. The endpoint typically returns the time until the token expires, the OAuth 2.0 *scopes* associated with the token, and potentially other information.

The introspection endpoint is defined as a standard method for resolving access tokens, in RFC-7662, *OAuth 2.0 Token Introspection* .

14.1.2. Usage

Use this resolver with the `accessTokenResolver` property of OAuth2ResourceServerFilter.

```
"accessTokenResolver": {
  "type": "TokenIntrospectionAccessTokenResolver",
  "config": {
    "amService": AmService reference,   // Use either "amService"
    "endpoint": URI string,             // or "endpoint", but not both.
    "providerHandler": Handler reference
  }
}
```

14.1.3. Properties

`"amService"`: *AmService reference, required if `"endpoint"` is not configured*

The AmService heap object to use for the token introspection endpoint. The endpoint is extrapolated from the `url` property of the AmService.

When the authorization server is AM, use this property to define the token introspection endpoint.

If `amService` is configured, it takes precedence over `endpoint`.

See also, "*AmService*".

`"endpoint"`: *URI string, required if `"amService"` is not configured*

The URI for the token introspection endpoint. Use `/oauth2/introspect`.

When the authorization server is not AM, use this property to define the token introspection endpoint.

If `amService` is configured, it takes precedence over `endpoint`.

`"providerHandler"`: *Handler reference, optional*

Invoke this HTTP client handler to send token info requests.

Provide either the name of a Handler object defined in the heap, or an inline Handler configuration object.

Default: ForgeRockClientHandler

If you use the AM token introspection endpoint, this handler can be a `Chain` containing a `HeaderFilter` to add the authorization to the request header, as in the following example:

```
"providerHandler": {
  "type": "Chain",
  "config": {
    "filters": [
      {
        "type": "HeaderFilter",
        "config": {
          "messageType": "request",
          "add": {
            "Authorization": [ "Basic ${encodeBase64('<client_id>:<client_secret>')}" ]
          }
        }
      }
    ],
    "handler": "ForgeRockClientHandler"
  }
}
```

14.1.4. More Information

org.forgerock.http.oauth2.resolver.TokenIntrospectionAccessTokenResolver

"*OAuth2ResourceServerFilter*"

Chapter 14.2

StatelessAccessTokenResolver

14.2.1. Description

Locally resolve and validate stateless access_tokens issued by AM, without referring to AM.

AM can be configured to secure access_tokens by signing or encrypting. The StatelessAccessTokenResolver must be configured for signature or encryption according to the AM configuration.

Supported with OpenAM 13.5, and AM 5 and later versions.

14.2.2. Usage

Use this resolver with the `accessTokenResolver` property of OAuth2ResourceServerFilter.

```
"accessTokenResolver": {
  "type": "StatelessAccessTokenResolver",
  "config": {
    "issuer": URI string,
    "secretsProvider": SecretsProvider reference,
    "verificationSecretId": configuration expression<secret-id>, // Use "verificationSecretId" or
    "decryptionSecretId": configuration expression<secret-id>,   // "decryptionSecretId", but not both
    "skewAllowance": configuration expression<duration>
  }
}
```

14.2.3. Properties

`"issuer"`: *Issuer reference, required*

URI of the AM instance responsible for issuing access_tokens.

`"secretsProvider"`: *SecretsProvider reference, optional*

The "*SecretsProvider*" to use to resolve queried secrets, such as passwords and cryptographic keys. Provide either the name of a SecretsProvider object defined in the heap, or specify a SecretsProvider object inline.

Default: Use the global secrets service.

`"verificationSecretId"`: *configuration expression<secret-id>, required if AM secures access_tokens with a signature*

The secret ID for the secret used to verify the signature of signed access_tokens.

Depending on the type of secret store that is used to verify signatures, use the following values:

- For JwkSetSecretStore, use any non-empty string that conforms to the field convention for secret-id. The value of the string is not used.
- For other types of secret stores:
  - `null`: No signature verification is required.
  - A `kid` as a string: Signature verification is required with the provided `kid`. The StatelessAccessTokenResolver searches for the matching `kid` in the SecretsProvider or global secrets service.

Use either `verificationSecretId` or `decryptionSecretId`, according to the configuration of the token provider in AM. If AM is configured to sign **and** encrypt tokens, encryption takes precedence over signing.

`"decryptionSecretId"`: *configuration expression<secret-id>, required if AM secures access_tokens with encryption*

The secret ID for the secret used to decrypt the JWT, for confidentiality.

Use either `verificationSecretId` or `decryptionSecretId`, according to the configuration of the token provider in AM. If AM is configured to sign **and** encrypt the token, encryption takes precedence over signing.

For information about supported formats for `secret-id`, see secret-id.

`"skewAllowance"`: *configuration expression<duration>, optional*

The time to add to the validity period of a JWT to allow for clock skew between different servers. To support a zero-trust policy, the skew allowance is by default zero.

A `skewAllowance` of 2 minutes affects the validity period as follows:

- A JWT with an `iat` of 12:00 is valid from 11:58 on the Microgateway clock.
- A JWT with an `exp` 13:00 is expired after 13:02 on the Microgateway clock.

Default: `zero`

14.2.4. More Information

org.forgerock.openig.filter.oauth2.StatelessAccessTokenResolver

"*OAuth2ResourceServerFilter*"

Chapter 14.3

OpenAmAccessTokenResolver

14.3.1. Description

In OAuth2ResourceServerFilter, use the AM token info endpoint, `/oauth2/tokeninfo`, to resolve access tokens and retrieve information. The endpoint typically returns the time until the token expires, the OAuth 2.0 *scopes* associated with the token, and potentially other information.

14.3.2. Usage

Use this resolver with the `accessTokenResolver` property of OAuth2ResourceServerFilter.

```
"accessTokenResolver": {
  "type": "OpenAmAccessTokenResolver",
  "config": {
    "amService": AmService reference,
    "providerHandler": Handler reference
  }
}
```

14.3.3. Properties

`"amService"`: *AmService reference, required*

The AmService heap object to use for the token info endpoint. The endpoint is extrapolated from the `url` property of the AmService.

See also, "*AmService*".

`"providerHandler"`: *Handler reference, optional*

Invoke this HTTP client handler to send token info requests.

Provide either the name of a Handler object defined in the heap, or an inline Handler configuration object.

Tip

To facilitate auditing, configure this handler with a `ForgeRockClientHandler`, which sends a ForgeRock Common Audit transaction ID when it communicates with protected applications.

Alternatively, configure this handler as a chain containing a `TransactionIdOutboundFilter`, as in the following configuration:

```
providerHandler : {
  "type": "Chain",
  "config": {
    "handler": "MySecureClientHandler",
    "filters": [ "TransactionIdOutboundFilter" ]
  }
}
```

Default: ForgeRockClientHandler

14.3.4. More Information

org.forgerock.http.oauth2.resolver.OpenAmAccessTokenResolver

"*OAuth2ResourceServerFilter*"

Chapter 14.4

ConfirmationKeyVerifierAccessTokenResolver

Supported with AM 6.5.1 and later versions.

In OAuth2ResourceServerFilter, use the ConfirmationKeyVerifierAccessTokenResolver to verify that certificate-bound OAuth 2.0 bearer tokens presented by clients use the same mTLS-authenticated HTTP connection.

When a client obtains an access_token from AM by using mTLS, AM can optionally use a confirmation key to bind the access_token to a certificate. When the client connects to Microgateway using that certificate, the ConfirmationKeyVerifierAccessTokenResolver verifies that the confirmation key corresponds to the certificate.

This proof-of-possession interaction ensures that only the client in possession of the key corresponding to the certificate can use the access_token to access protected resources.

To use the ConfirmationKeyVerifierAccessTokenResolver, the following configuration is required in AM:

- OAuth 2.0 clients must be registered using an X.509 certificate, that is self-signed or signed in public key infrastructure (PKI)
- The AM client authentication method must be `self_signed_client_auth` or `tls_client_auth`.
- AM must be configured to bind a confirmation key to each client certificate.

The ConfirmationKeyVerifierAccessTokenResolver delegates the token resolution to a specified AccessTokenResolver, which retrieves the token information. The ConfirmationKeyVerifierAccessTokenResolver verifies the confirmation keys bound to the access_token, and then acts as follows:

- If there is no confirmation key, pass the request down the chain.
- If the confirmation key matches the client certificate, pass the request down the chain.
- If the confirmation key doesn't match the client certificate, throw an error.
- If the confirmation key method is not supported by Microgateway, throw an error.

For information about issuing certificate-bound OAuth 2.0 access_tokens, see AM's Certificate-Bound Proof-of-Possession. For information about authenticating an OAuth 2.0 client using mTLS certificates, see AM's Authenticating Clients Using Mutual TLS.

14.4.1. Usage

Use this resolver with the `accessTokenResolver` property of "*OAuth2ResourceServerFilter*".

```
"accessTokenResolver": {
  "type": "ConfirmationKeyVerifierAccessTokenResolver",
  "config": {
    "delegate": accessTokenResolver reference
  }
}
```

14.4.2. Properties

`"delegate"`: *accessTokenResolver reference, required*

The access token resolver to use for resolving access_tokens. Use any access token resolver described in "Access Token Resolvers".

14.4.3. More Information

org.forgerock.openig.http.oauth2.resolver.ConfirmationKeyVerifierAccessTokenResolver

OAuth 2.0 Mutual TLS Client Authentication and Certificate Bound Access Tokens

"*OAuth2ResourceServerFilter*"

Chapter 14.5

ScriptableAccessTokenResolver

14.5.1. Description

In OAuth2ResourceServerFilter, use a Groovy script to resolve access tokens against an authorization server.

Receive a string representing an access token and use a Groovy script to create an instance or promise of `org.forgerock.http.oauth2.AccessTokenInfo`.

14.5.2. Usage

Use this resolver with the `accessTokenResolver` property of OAuth2ResourceServerFilter.

```
"accessTokenResolver": {
  "type": "ScriptableAccessTokenResolver",
  "config": {
    "type": string,
    "file": expression,                     // Use either "file"
    "source": string or array of strings,   // or "source", but not both.
    "args": object,
    "clientHandler": Handler reference
  }
}
```

14.5.3. Properties

For information about properties for ScriptableAccessTokenResolver, see "*Scripts*".

14.5.4. More Information

org.forgerock.openig.filter.oauth2.ScriptableAccessTokenResolver

"*OAuth2ResourceServerFilter*"

Chapter 14.6

CacheAccessTokenResolver

Enable and configure caching of OAuth 2.0 access_tokens, based on *Caffeine*. For more information, see the GitHub entry, *Caffeine*.

This resolver configures caching of access_tokens, and delegates their resolution to another AccessTokenResolver.

For an alternative way to cache OAuth 2.0 access_tokens, configure the `cache` property of OAuth2ResourceServerFilter.

14.6.1. Usage

```
{
  "name": string,
  "type": "CacheAccessTokenResolver",
  "config": {
    "delegate": AccessTokenResolver reference,
    "enabled": configuration expression<boolean>,
    "defaultTimeout": configuration expression<duration>,
    "executor": ScheduledExecutorService reference,
    "maximumSize": configuration expression<number>,
    "maximumTimeToCache": configuration expression<duration>
  }
}
```

14.6.2. Properties

`"delegate"`: *AccessTokenResolver reference, required*

Delegate access_token resolution to one of the access_token resolvers in "Access Token Resolvers".

`enabled`: *configuration expression<boolean>, optional*

Enable caching.

When an access_token is cached, Microgateway can reuse the token information without repeatedly asking the authorization server to verify the access_token. When caching is disabled, Microgateway must ask the authorization server to validate the access_token for each request.

Default: `true`

`defaultTimeout`: ***configuration expression<duration>, optional***

The duration for which to cache an OAuth 2.0 access_token when it doesn't provide a valid expiry value or `maximumTimeToCache`.

If the `defaultTimeout` is longer than the `maximumTimeToCache`, then the `maximumTimeToCache` takes precedence.

Default: `1 minute`

`"executor"`: ***ScheduledExecutorService reference, optional***

An executor service to schedule the execution of tasks, such as the eviction of entries from the cache.

Default: `ForkJoinPool.commonPool()`

`"maximumSize"`: ***configuration expression<number>, optional***

The maximum number of entries the cache can contain.

Default: Unlimited/unbound

`"maximumTimeToCache"`: ***configuration expression<duration>, optional***

The maximum duration for which to cache access_tokens.

Cached access_tokens are expired according to their expiry time and `maximumTimeToCache`, as follows:

- If the expiry time is *before* the current time plus the `maximumTimeToCache`, the cached token is expired when the expiry time is reached.
- If the expiry time is *after* the current time plus the `maximumTimeToCache`, the cached token is expired when the `maximumTimeToCache` is reached

The duration cannot be `zero` or `unlimited`.

Default: The token expiry time or `defaultTimeout`

Part 15
Secrets

Microgateway uses the ForgeRock Commons Secrets Service to manage secrets, such as passwords and cryptographic keys.

Chapter 15.1

secrets

Microgateway automatically creates a secrets object in each route in the configuration, and in `config.json` and `admin.json`.

When the secrets object is not used to declare a secrets store in the configuration, Microgateway creates a default "*SystemAndEnvSecretStore*" in the local secrets service. When the secrets object is used to declare a secrets store, the default is not installed in the local secrets service.

15.1.1. Usage

```
{
  "secrets": {
    "stores": [ inline secret store declaration, ... ]
  }
}
```

15.1.2. Properties

`"stores"`: *array of secret store declarations, required*

One or more inline declarations of the following secret stores:

- "*Base64EncodedSecretStore*"
- "*FileSystemSecretStore*"
- "*HsmSecretStore*"
- "*JwkSetSecretStore*"
- "*KeyStoreSecretStore*"
- "*SystemAndEnvSecretStore*"

15.1.3. Example

The following example configures two secret stores:

```
{
  "secrets": {
    "stores": [
       {
        "type": "FileSystemSecretStore",
        "config": {
          "directory": "/path/to/secrets",
          "format": "BASE64"
        }
      },
      {
        "type": "SystemAndEnvSecretStore",
        "config": {
          "format": "PLAIN"
        }
      }
    ]
  }
}
```

Chapter 15.2

Base64EncodedSecretStore

15.2.1. Description

Manage a repository of generic secrets, such as passwords or simple shared secrets, whose values are base64-encoded, and hard-coded in the route.

Secrets from Base64EncodedSecretStore never expire.

Important

Use Base64EncodedSecretStore for testing or evaluation only, to store passwords locally. In production, use an alternative secret store.

For a description of how secrets are managed, see About Secrets.

15.2.2. Usage

```
{
  "name": string,
  "type": "Base64EncodedSecretStore",
  "config": {
    "secrets": map
  }
}
```

15.2.3. Properties

`"secrets"`: *map, required*

A list of one or more secret ID/string pairs:

```
{
  "secrets": {
    "secret-id": "string",
    ...
  }
}
```

Each pair has the form "*secret-id*": "*string*", where:

- *secret-id* is the ID of a secret used in a route
- *string* is the base64-encoded value of the secret

In the following example, Base64EncodedSecretStore configures two base64-encoded secrets:

```
{
  "type": "Base64EncodedSecretStore",
  "config": {
    "secrets": {
      "agent.password": "d2VsY29tZQ==",
      "crypto.header.key": "Y2hhbmdlaXQ="
    }
  }
}
```

In the following example, the values of the secrets are provided by a configuration token and a configuration expression, whose values are substituted when the route is loaded:

```
{
  "type": "Base64EncodedSecretStore",
  "config": {
    "secrets": {
      "agent.password": "&{secret.value|aGVsbG8=}",
      "crypto.header.key": "${readProperties('file.property')['b64.key.value']}"
    }
  }
}
```

For information about supported formats for `secret-id`, see secret-id.

15.2.4. Log Level

To facilitate debugging secrets for the Base64EncodedSecretStore, in `logback.xml` add a logger defined by the fully qualified package name of the Base64EncodedSecretStore. The following line in `logback.xml` sets the log level to `ALL`:

```
<logger name="org.forgerock.openig.secrets.Base64EncodedSecretStore" level="ALL">
```

15.2.5. More Information

"Secrets"

org.forgerock.openig.secrets.Base64EncodedSecretStore

Chapter 15.3

FileSystemSecretStore

15.3.1. Description

Manage a store of secrets held in files in a specified directory. Each file must contain only one secret, in the `format` declared in the configuration. Secrets are read lazily from the filesystem, and are cached indefinitely.

Secrets from FileSystemSecretStore never expire.

For a description of how secrets are managed, see About Secrets.

15.3.2. Usage

```
{
  "name": string,
  "type": "FileSystemSecretStore",
  "config": {
    "directory": configuration expression<string>,
    "format": configuration expression<enumeration>,
    "suffix": configuration expression<string>
  }
}
```

15.3.3. Properties

`"directory"`: *configuration expression<string>, required*

File path to a directory containing secret files. This object checks the specified directory, but not its subdirectories.

`"format"`: *configuration expression<enumeration>, optional*

Format in which the secret is stored. Use one of the following values:

- `BASE64`: Base64-encoded
- `PLAIN`: Plain text

Default: `BASE64`

`"suffix"`: *configuration expression<string>, optional*

File suffix.

When set, the FileSystemSecretStore will append that suffix to the secret ID and try to find a file with the mapped name.

Default: None

15.3.4. Log Level

To facilitate debugging secrets for the FileSystemSecretStore, in `logback.xml` add a logger defined by the fully qualified package name of the property resolver. The following line in `logback.xml` sets the log level to `ALL`:

```
<logger name="org.forgerock.secrets.propertyresolver" level="ALL">
```

15.3.5. More Information

"Secrets"

org.forgerock.openig.secrets.FileSystemSecretStoreHeaplet

Chapter 15.4

HsmSecretStore

15.4.1. Description

Manage a store of secrets with a hardware security module (HSM) device or a software emulation of an HSM device, such as SoftHSM.

Secrets from HsmSecretStore have a non-configurable lease duration of five minutes. The secret can be used for five minutes before it is refreshed or discarded.

For a description of how secrets are managed, see About Secrets.

15.4.2. Usage

```
{
  "name": string,
  "type": "HsmSecretStore",
  "config": {
    "providerName": configuration expression<string>,
    "storePassword": configuration expression<secret-id>,
    "secretsProvider": SecretsProvider reference,
    "mappings": [ configuration object, ... ]
  }
}
```

15.4.3. Properties

`"providerName"`: *configuration expression<enumeration>, required*

The name of the pre-installed Java Security Provider supporting an HSM. Use a physical HSM device, or a software emulation of an HSM device, such as SoftHSM.

For the SunPKCS11 provider, concatenate `"providerName"` with the prefix `SunPKCS11-`. For example, declare the following for the name `FooAccelerator`:

```
"providerName": "SunPKCS11-FooAccelerator"
```

`"storePassword"`: *configuration expression<secret-id>, required*

The secret ID of the password to access the HsmSecretStore.

For information about supported formats for `secret-id`, see secret-id.

`"secretsProvider"`: *SecretsProvider reference, optional*

The SecretsProvider object to query for the `storePassword`. For more information, see "*SecretsProvider*".

Default: The route's default secret service. For more information, see "*secrets*".

`"mappings"`: *array of objects, required*

One or more mappings of one secret ID to one or more aliases. The secret store uses the mappings as follows:

- When the secret is used to create signatures or encrypt values, the secret store uses the *active secret*, the first alias in the list.
- When the secret is used to verify signatures or decrypt data, the secret store tries all of the mapped aliases in the list, starting with the first, and stopping when it finds a secret that can successfully verify signature or decrypt the data.

The following example maps a secret ID to two aliases:

```
"mappings": [
  {
    "secretId": "global.pcookie.crypt",
    "aliases": [ "rsapair72-1", "rsapair72-2" ]
  }
]
```

`secretId`: *configuration expression<secret-id>, required*

The ID of the secret used in your configuration.

For information about supported formats for `secret-id`, see secret-id.

`aliases`: *array of configuration expression<string>, required*

One or more aliases for the secret ID.

15.4.4. Log Level

To facilitate debugging secrets for the HsmSecretStore, in `logback.xml` add a logger defined by the fully qualified package name of the HsmSecretStore. The following line in `logback.xml` sets the log level to `ALL`:

```
<logger name="org.forgerock.secrets.keystore" level="ALL">
```

15.4.5. Example

To set up this example:

1. Set up and test the example in "*JwtBuilderFilter*", and then replace the KeyStoreSecretStore in that example with an HsmSecretStore.
2. Set an environment variable for the HsmSecretStore password, `storePassword`, and then restart Microgateway.

   For example, if the HsmSecretStore password is `password`, set the following environment variable:

   ```
   export HSM.PIN='cGFzc3dvcmQ='
   ```

   The password is retrieved by the SystemAndEnvSecretStore, and must be base64-encoded.
3. Create a provider config file, as specified in the PKCS#11 Reference Guide.
4. Depending on your version of Java, create a `java.security.ext` file for the Microgateway instance, with the following content:

   ```
   security.provider.<number>=<provider-name> <path-to-provider-cfg-file>
   ```

   or

   ```
   security.provider.<number>=<class-name> <path-to-provider-cfg-file>
   ```
5. Start the Microgateway JVM with the following system property that points to the provider config file:

   ```
   -Djava.security.properties=file://path-to-security-extension-file
   ```

15.4.6. More Information

"Secrets"

org.forgerock.openig.secrets.HsmSecretStoreHeaplet

Chapter 15.5

JwkSetSecretStore

Manages a secret store for JSON Web Keys (JWK) from a local or remote JWK Set.

Secrets from JwkSetSecretStore have a non-configurable lease duration, equal to the value of `cacheTimeout`. The secret can be used for that duration before it is refreshed or discarded.

For information about JWKs and JWK Sets, see RFC-7517, JSON Web Key (JWK).

15.5.1. Usage

```
{
  "name": string,
  "type": "JwkSetSecretStore",
  "config": {
    "jwkUrl": configuration expression<url>,
    "handler": Handler reference or inline handler declaration,
    "cacheTimeout": configuration expression<duration>,
    "cacheMissCacheTime": configuration expression<duration>
  }
}
```

15.5.2. Properties

`"jwkUrl"`: *configuration expression<url>, required*

A URL that contains the client's public keys in JWK format.

`"handler"`: *Handler reference, optional*

An HTTP client handler to communicate with the `jwkUrl`.

Usually set this property to the name of a ClientHandler configured in the heap, or a chain that ends in a ClientHandler.

Default: ClientHandler

`"cacheTimeout"`: *configuration expression<duration>, optional*

Delay before the cache is reloaded. The cache contains the `jwkUrl`.

Default: 2 minutes

`"cacheMissCacheTime"`: *configuration expression<duration>, optional*

If the `jwkUrl` is looked up in the cache and is not found, this is the delay before the cache is reloaded.

Default: 2 minutes

15.5.3. Log Level

To facilitate debugging secrets for the JwkSetSecretStore, in `logback.xml` add a logger defined by the fully qualified package name of the JwkSetSecretStore. The following line in `logback.xml` sets the log level to `ALL`:

```
<logger name="org.forgerock.secrets.jwkset" level="ALL">
```

15.5.4. More Information

org.forgerock.openig.secrets.JwkSetSecretStoreHeaplet

RFC-7517, JSON Web Key (JWK)

Chapter 15.6
KeyStoreSecretStore

15.6.1. Description

Manages a secret store for cryptographic keys and certificates, based on a standard Java KeyStore.

The KeyStore is typically file-based PKCS12 KeyStore. Legacy proprietary formats such as JKS and JCEKS are supported, but implement weak encryption and integrity protection mechanisms. Consider not using them for new functionality.

Secrets from KeyStoreSecretStore have a non-configurable lease duration of five minutes. The secret can be used for five minutes before it is refreshed or discarded.

For a description of how secrets are managed, see About Secrets.

15.6.2. Usage

```
{
  "name": string,
  "type": "KeyStoreSecretStore",
  "config": {
    "file": configuration expression<string>,
    "storeType": configuration expression<string>,
    "storePassword": configuration expression<string>,
    "keyEntryPassword": configuration expression<string>,
    "secretsProvider": SecretsProvider reference,
    "mappings": [ configuration object, ... ]
  }
}
```

15.6.3. Properties

`"file"`: *configuration expression<string>, required*

The path to the KeyStore file.

`"storeType"`: *storeType reference, optional*

The KeyStore type. For a list of types, see KeyStore Types.

Default: When this property is not configured, the type is given by the keystore extension, as follows:

Extension	Type
`.jks`	`JKS`
`.jceks`	`JCEKS`
`.p12`, `.pfx`, `.pkcs12`, and all other extensions	`PKCS12`

`"storePassword"`: *configuration expression<secret-id>, required*

The secret ID of the password to access the KeyStore.

Microgateway searches for the value of the password until it finds it, first locally, then in parent routes, then in `config.json`.

For information about supported formats for `secret-id`, see secret-id.

`"keyEntryPassword"`: *configuration expression<secret-id>, optional*

The secret ID of the password to access entries in the KeyStore.

When this property is used, the password must be the same for all entries in the KeyStore. If JKS uses different password for entries, `keyEntryPassword` doesn't work.

For information about supported formats for `secret-id`, see secret-id.

Default: The value of `storePassword`

`"secretsProvider"`: *SecretsProvider reference, optional*

The SecretsProvider object to query for the keystore password and key entry password. For more information, see "*SecretsProvider*".

Default: The route's default secret service. For more information, see "*secrets*".

`"mappings"`: *array of objects, required*

One or more mappings of one secret ID to one or more aliases. The secret store uses the mappings as follows:

- When the secret is used to create signatures or encrypt values, the secret store uses the *active secret*, the first alias in the list.
- When the secret is used to verify signatures or decrypt data, the secret store tries all of the mapped aliases in the list, starting with the first, and stopping when it finds a secret that can successfully verify signature or decrypt the data.

```
"mappings": [
  {
    "secretId": "id.key.for.signing.jwt",
    "aliases": [ "SigningKeyAlias", "AnotherSigningKeyAlias" ]
  },
  {
    "secretId": "id.key.for.encrypting.jwt",
    "aliases": ["EncryptionKeyAlias"]
  }
]
```

`secretId`: *configuration expression<secret-id>, required*

The ID of the secret used in your configuration.

For information about supported formats for `secret-id`, see secret-id.

`aliases`: *array of configuration expression<string>, required*

One or more aliases for the secret ID.

15.6.4. Log Level

To facilitate debugging secrets for the KeyStoreSecretStore, in `logback.xml` add a logger defined by the fully qualified package name of the KeyStoreSecretStore. The following line in `logback.xml` sets the log level to `ALL`:

```
<logger name="org.forgerock.secrets.keystore" level="ALL">
```

15.6.5. Example

For examples of routes that use KeyStoreSecretStore, see the examples in "*JwtBuilderFilter*".

15.6.6. More Information

org.forgerock.secrets.keystore.KeyStoreSecretStore

org.forgerock.openig.secrets.KeyStoreSecretStoreHeaplet

Chapter 15.7

SystemAndEnvSecretStore

15.7.1. Description

Manage a store of secrets from system properties and environment variables.

A secret ID must conform to the convention described in secret-id. The reference is then transformed to match the environment variable name, as follows:

- Periods (.) are converted to underscores.
- Characters are transformed to uppercase.

For example, `my.secret.id` is transformed to `MY_SECRET_ID`.

Secrets from SystemAndEnvSecretStore never expire.

For a description of how secrets are managed, see About Secrets.

15.7.2. Usage

```
{
  "name": string,
  "type": "SystemAndEnvSecretStore",
  "config": {
    "format": configuration expression<enumeration>
  }
}
```

15.7.3. Properties

`"format"`: *configuration expression<enumeration>, optional*

Format in which the secret is stored. Use one of the following values:

- `BASE64`: Base64-encoded
- `PLAIN`: Plain text

Default: `BASE64`

15.7.4. Log Level

To facilitate debugging secrets for the SystemAndEnvSecretStore, in `logback.xml` add a logger defined by the fully qualified package name of the property resolver. The following line in `logback.xml` sets the log level to `ALL`:

```
<logger name="org.forgerock.secrets.propertyresolver" level="ALL">
```

15.7.5. More Information

"Secrets"

org.forgerock.openig.secrets.SystemAndEnvSecretStoreHeaplet

Chapter 16

Supported Standards

Microgateway implements the following RFCs, Internet-Drafts, and standards:

OAuth 2.0

RFC 6749: The OAuth 2.0 Authorization Framework

RFC 6750: The OAuth 2.0 Authorization Framework: Bearer Token Usage

RFC 7515: JSON Web Signature (JWS)

RFC 7516: JSON Web Encryption (JWE)

RFC 7517: JSON Web Key (JWK)

RFC 7518: JSON Web Algorithms (JWA)

RFC 7519: JSON Web Token (JWT)

RFC 7523: JSON Web Token (JWT) Profile for OAuth 2.0 Client Authentication

RFC 7591: OAuth 2.0 Dynamic Client Registration Protocol

RFC 7662: OAuth 2.0 Token Introspection

RFC 7800: Proof-of-Possession Key Semantics for JSON Web Tokens (JWTs) with Internet-Draft: OAuth 2.0 Mutual TLS Client Authentication and Certificate Bound Access Tokens

OpenID Connect 1.0

Microgateway can be configured to play the role of OpenID Connect relying party. The OpenID Connect specifications depend on OAuth 2.0, JSON Web Token, Simple Web Discovery and related specifications. The following specifications make up OpenID Connect 1.0.

- OpenID Connect Core 1.0 defines core OpenID Connect 1.0 features.

  Note

  In section 5.6 of the specification, Microgateway supports *Normal Claims*. The optional *Aggregated Claims* and *Distributed Claims* representations are not supported by Microgateway.

- OpenID Connect Discovery 1.0 defines how clients can dynamically discover information about OpenID Connect providers.

- OpenID Connect Dynamic Client Registration 1.0 defines how clients can dynamically register with OpenID Connect providers.
- OAuth 2.0 Multiple Response Type Encoding Practices defines additional OAuth 2.0 response types used in OpenID Connect.

User-Managed Access (UMA) 2.0

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

Federated Authorization for User-Managed Access (UMA) 2.0

Representational State Transfer (REST)

Style of software architecture for web-based, distributed systems. Microgateway's APIs are RESTful APIs.

Security Assertion Markup Language (SAML)

Standard, XML-based framework for implementing a SAML service provider. Microgateway supports multiple versions of SAML including 2.0, 1.1, and 1.0.

Specifications are available from the OASIS standards page.

Other Standards

RFC 2616: *Hypertext Transfer Protocol -- HTTP/1.1*.

RFC 2617: *HTTP Authentication: Basic and Digest Access Authentication*, supported as an authentication module.

RFC 4510: *Lightweight Directory Access Protocol (LDAP)*, for authentication modules and when accessing data stores.

RFC 5280: *Internet X.509 Public Key Infrastructure Certificate*, supported for certificate-based authentication.

RFC 5785: *Defining Well-Known Uniform Resource Identifiers (URIs)*.

RFC 6265: *HTTP State Management Mechanism* regarding HTTP Cookies and `Set-Cookie` header fields.