Reference

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.

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

For API specifications, see the appropriate Javadoc.

The examples in this guide use some of the following third-party tools:

- **`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

Reserved routes

By default, IG 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).

Reserved field names

IG 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`.

Field value conventions

IG 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`.

certificate
: `java.security.cert.Certificate` instance.

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

enumeration
: A collections of constants.

expression
: See Expressions.

configuration expression
: Expression evaluated at configuration time, when routes are loaded. See Configuration Expressions.

runtime expression

Expression evaluated at runtime, for each request and response. See Runtime Expressions.

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

JsonValue
: An object (JsonObject), an array (JsonArray), a number (JsonNumber), a string (JsonString), true (JsonValue.TRUE), false (JsonValue.FALSE), or null (JsonValue.NULL).

lvalue-expression
: Expression yielding an object whose value is to be set.

  Properties whose format is `lvalue-expression` cannot consume streamed content. They must be written with `$` instead of `#`.

map
: An object that maps keys to values. Keys must be unique, and can map to at most one value.

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 Commons Secrets Service, as described in Secrets.

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

string
: JSON string.

url
: String representation for a resource available via the Internet. For more information, see Uniform Resource Locators (URL).

About ForgeRock Common REST

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

NOTE

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

Common REST resources

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

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

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

Common REST verbs

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

Create
: Add a new resource.

  This verb maps to HTTP PUT or HTTP POST.

  For details, see Create.

Read
: Retrieve a single resource.

  This verb maps to HTTP GET.

  For details, see Read.

Update
: Replace an existing resource.

  This verb maps to HTTP PUT.

  For details, see Update.

Delete
: Remove an existing resource.

  This verb maps to HTTP DELETE.

  For details, see Delete.

Patch
: Modify part of an existing resource.

  This verb maps to HTTP PATCH.

  For details, see Patch.

Action
: Perform a predefined action.

  This verb maps to HTTP POST.

  For details, see Action.

Query
: Search a collection of resources.

  This verb maps to HTTP GET.

For details, see Query.

Common REST parameters

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

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

NOTE

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

Continue reading for details about how to use each parameter.

Common REST extension points

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

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

Common REST API documentation

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

Use the following query string parameters to retrieve API descriptors:

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

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

_crestapi
: Serves a native Common REST API descriptor.

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

NOTE

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

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

To publish OpenAPI documentation

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

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

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

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

2. Retrieve the OpenAPI-compliant descriptor.

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

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

3. If necessary, edit the descriptor.

> For example, add security definitions to describe the API protection.
>
> 4. Publish the descriptor using a tool such as Swagger UI.

Create

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

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

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

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

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

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

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

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

Parameters

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

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

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

_prettyPrint=true
Format the body of the response.

Read

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

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

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

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

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

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

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

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

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

_prettyPrint=true
Format the body of the response.

Update

To update a resource, perform an HTTP PUT including the case-sensitive identifier (`_id`) as the final element of the path to the resource, and the JSON resource as the payload. Use the `If-Match: _rev` header to check that you are actually updating the version you modified. Use `If-Match: *` if the version does not matter. Accept a JSON response:

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

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

NOTE

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

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

Parameters

_fields=field[,field…]

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

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

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

_prettyPrint=true

Format the body of the response.

Delete

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

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

Parameters

_fields=field[,field…]

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

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

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

_prettyPrint=true

Format the body of the response.

Patch

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

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

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

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

PATCH operations apply to three types of targets:

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

ForgeRock PATCH supports multiple `operations`:

Patch operation: add

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

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

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

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

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

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

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

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

The following is the resulting resource:

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

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

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

The resulting resource would have the following invalid JSON structure:

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

Patch operation: copy

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

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

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

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

Patch operation: increment

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

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

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

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

Patch operation: move

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

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

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

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

Patch operation: remove

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

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

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

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

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

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

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

Patch operation: replace

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

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

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

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

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

Apply the following operations on that resource:

```
[
  {
```

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

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

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

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

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

Patch operation: transform

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

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

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

Patch operation limitations

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

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

Parameters

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

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

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

_prettyPrint=true
: Format the body of the response.

Action

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

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

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

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

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

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

_prettyPrint=true
: Format the body of the response.

Query

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

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

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

Parameters

_countOnly=true

Return a count of query results without returning the resources.

This parameter requires protocol version 2.2 or later.

_fields=field[,field…]

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

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

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

_queryFilter=filter-expression

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

`_queryId=identifier`

Specify a query by its identifier.

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

`_pagedResultsCookie=string`

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

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

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

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

`_pagedResultsOffset=integer`

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

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

`_pageSize=integer`

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

_prettyPrint=true

Format the body of the response.

_totalPagedResultsPolicy=string

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

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

_sortKeys=(|-)__field__[,(|-)field…]

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

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

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

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

HTTP status codes

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

200 OK

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

201 Created

The request succeeded and the resource was created.

204 No Content

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

304 Not Modified

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

400 Bad Request
: The request was malformed.

401 Unauthorized
: The request requires user authentication.

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

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

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

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

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

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

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

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

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

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

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

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

Required configuration

AdminHttpApplication (`admin.json`)

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 file, expected at `$HOME/.openig/config/admin.json`. Objects configured in `admin.json` cannot be used by `config.json` or any IG routes.

Default objects

IG provides default objects in `admin.json`. To override a default object, configure an object with the same name in `admin.json`.

Configure default objects in `admin.json` and `config.json` seperately. An object configured in `admin.json` with the same name as an object configured in `config.json` is not the same object.

`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.

`ProxyOptions`
: A proxy to which a ClientHandler or ReverseProxyHandler can submit requests, and an AmService can submit Websocket notifications. For more information, see ProxyOptions.

`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 Commons Secrets Service. The default SecretsService is a SystemAndEnvSecretStore with the default configuration. For more information, see Secrets.

TemporaryStorage
Manages temporary buffers. 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.

Provided objects

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

*"**ApiProtectionFilter**"*
A filter to protect administrative APIs on reserved routes. By default, only the loopback address can access reserved routes.

For an example that uses an ApiProtectionFilter, see Set up the UMA example. For information about reserved routes, see Reserved routes.

*"**MetricsProtectionFilter**"*
A filter 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.

For an example that uses a MetricsProtectionFilter, see Protect monitoring endpoints.

*"**StudioProtectionFilter**"*
A filter to protect the Studio endpoint when IG is running in development mode.

When IG is running in development mode, by default the Studio endpoint is open and accessible.

For an example that uses a StudioProtectionFilter, see Restricting access to Studio.

Usage

```
{
  "heap": [ object, ... ],
  "connectors": [ object, ... ],
  "vertx": object,
  "gatewayUnits": configuration expression<number>,
  "mode": configuration expression<enumeration>,
  "prefix": configuration expression<string>,
  "properties": object,
  "secrets": {
    "stores": [ SecretStore reference, ... ]
  },
  "temporaryDirectory": configuration expression<string>,
  "temporaryStorage": TemporaryStorage reference,
  "preserveOriginalQueryString": configuration
expression<boolean>,
  "session": object,
  "streamingEnabled": configuration expression<boolean>
}
```

Properties

"heap": array of 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.

"connectors": array of objects, required

Supported only for IG in standalone mode.

Server port configuration, when IG is server-side.

When an application sends requests to IG, or requests services from IG, IG is acting as a server of the application (server-side), and the application is acting as a client.

```
{
  "connectors" : [
    {
      "port": [ configuration expression<number>, ... ],
      "tls": ServerTlsOptions reference,
      "vertx": object
    },
```

```
    {
      ...
    }
  ]
}
```

`port`: array of configuration expression<numbers>, required

One or more ports on which IG is connected. When more than one port is defined, IG is connected to each port.

`tls`: ServerTlsOptions reference, optional

Configure options for connections to TLS-protected endpoints, based on ServerTlsOptions. Define the object inline or in the heap.

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

`vertx`: object, optional

Vert.x-specific configuration for this connector, where IG does not provide its own first-class configuration. Vert.x options are described in HttpServerOptions.

For properties where IG provides its own first-class configuration, Vert.x configuration options are disallowed, and the IG configuration option takes precedence over Vert.x options configured in `vertx`. Vert.x values are evaluated as configuration expressions.

The following Vert.x configuration options are disallowed server-side:

- `port`
- `useAlpn`
- `ssl`
- `enabledCipherSuites`
- `enabledSecureTransportProtocols`
- `jdkSslEngineOptions`
- `keyStoreOptions`
- `openSslEngineOptions`
- `pemKeyCertOptions`
- `pemTrustOptions`
- `pfxKeyCertOptions`
- `pfxTrustOptions`
- `trustStoreOptions`
- `clientAuth`

The following example configures Vert.x-specific options for the connection on 8080, and TLS options for the connection on 8443:

```
{
  "connectors": [{
    "port": 8080,
    "vertx": {
      "maxWebSocketFrameSize": 128000,
      "maxWebSocketMessageSize": 256000,
      "compressionLevel": 4,
      "maxHeaderSize": 16384
    }
  },
  {
    "port": 8443,
    "tls": "ServerTlsOptions-1"
  }]
}
```

`vertx`: object, optional

Supported only for IG in standalone mode. Vert.x-specific configuration used to more finely-tune Vert.x instances.

Vert.x values are evaluated as configuration expressions.

Use the Vert.x options described in VertxOptions, with the following exceptions:

- `metricsOptions` : Not used
- `metricsEnabled` : Enable Vertx metrics. Default: `true` .

For an example, see Monitoring Services.

IG proxies all WebSocket subprotocols by default. To proxy specific WebSocket subprotocols only, list them as follows:

```
"vertx": {
  "webSocketSubProtocols": ["v1.notifications.forgerock.org",
... ]
}
```

"`gatewayUnits`": configuration expression<number>, optional

Supported only for IG in standalone mode.

The number of parallel instances of IG to bind to an event loop. All instances listen on the same ports.

Default: The number of cores available to the JVM.

`mode`: configuration expression<enumeration>, optional

Set the IG 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.

  You can create, edit, and deploy routes through IG Studio, and manage routes through Common REST, without authentication or authorization.

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

  For information about how to switch to development mode, see Switching from production mode to development mode.

  For information about restricting access to Studio in development mode, see Restricting access to Studio.

- **Production mode (immutable mode)**

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

  Studio is effectively disabled, and you cannot manage, list, or even read routes through Common REST.

  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 IG configuration.

  For information about how to switch to production mode, see Switching from development mode to production mode.

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

  Default: `production`

"`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`": object, optional

Configuration parameters declared as property variables for use in the configuration. See also Route properties.

Default: Null

"`secrets`": SecretStore reference, optional

> IMPORTANT
>
> This property is deprecated; use SecretsProvider instead. For more information, refer to Deprecation.

One or more secret stores, as defined in Secrets object and secret stores.

"`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: `$HOME/.openig/tmp` (on Windows, `%appdata%\OpenIG\OpenIG\tmp`)

"`temporaryStorage`": TemporaryStorage reference, optional

The TemporaryStorage object to buffer content during processing.

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

Incoming requests use the temporary storage buffer as follows:

- For standalone mode only when `streamingEnabled` is `false`:
  - The request is loaded into the IG storage defined in `temporaryStorage`, before it enters the chain.
  - If the content length of a request is more than the buffer limit, IG returns an `HTTP 413 Payload Too Large`.
- For web container mode:
  - The request is loaded into the web container buffer, which is managed externally.
  - IG does not access the web container buffer until a filter, handler, or other object tries to access the request body.
  - At that point, IG transfers the content of the web container buffer to its own temporary storage.
  - If the web container buffer is bigger than the IG temporary storage, a buffer exception occurs.

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.

"`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. IG 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`

"`session`": object, optional

IMPORTANT

Supported only for IG in standalone mode.

Configures stateful sessions for IG in standalone mode. For information about IG sessions, see Sessions.

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

"`cookie`": object, optional

The configuration of the cookie used to store the stateful session.

Default: The session cookie is treated as a host-based cookie.

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

The session cookie name.

Default: `IG_SESSIONID`

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

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

Default: `true`

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

The path protected by the session.

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

Default: `/`.

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

Options to manage the circumstance in which the session cookie is sent to the server. The following values are listed in order of strictness, with most strict first:

- `STRICT`: Send the session cookie only if the request was initiated from the session cookie domain. Not case-sensitive. Use this value to reduce the risk of cross-site request forgery (CSRF) attacks.
- `LAX`: Send the session cookie only with GET requests in a first-party context, where the URL in the address bar matches the session cookie

domain. Not case-sensitive. Use this value to reduce the risk of cross-site request forgery (CSRF) attacks.

- `NONE` : Send the session cookie whenever a request is made to the session cookie domain. With this setting, consider setting `secure` to `true` to prevent browsers from rejecting the session cookie. For more information, see SameSite cookies

Default: `LAX`

NOTE

For CDSSO, set `"sameSite":"none"` and `"secure":"true"` . For security reasons, many browsers require the connection used by the browser to be secure (HTTPS) for `"sameSite":"none"` . Therefore, if the connection used by the browser is not secure (HTTP), the browser might not supply cookies with `"sameSite":"none"` .

"secure": configuration expression<boolean>, optional

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

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

Default: `false`

NOTE

For CDSSO, set `"sameSite":"none"` and `"secure":"true"` . For security reasons, many browsers require the connection used by the browser to be secure (HTTPS) for `"sameSite":"none"` . Therefore, if the connection used by the browser is not secure (HTTP), the browser might not supply cookies with `"sameSite":"none"` .

"timeout": configuration expression<duration>, optional

The duration after which idle sessions are automatically timed out.

The value must be above zero, and no greater than 3650 days (approximately 10 years). IG truncates the duration of longer values to 3650 days.

Default: 30 minutes

"streamingEnabled": configuration expression<boolean>, optional

IMPORTANT

Supported only for IG in standalone mode.

When `true`, the content of HTTP requests and responses is streamed; it is available for processing bit-by-bit, as soon as it is received.

When `false`, the content is buffered into the storage defined in `temporaryStorage`, and is available for processing after it has all been received.

When this property is `true`, consider the following requirements to prevent IG from blocking an executing thread to wait for streamed content:

- Runtime expressions that consume streamed content **must** be written with `#` instead of `$`.

  For more information, see runtime expression.

- In scripts and Java extensions, never use a `Promise` blocking method, such as `get()`, `getOrThrow()`, or `getOrThrowUninterruptibly()`, to obtain the response.

  For more information, see Scripts.

Default: `false`

Example configuration files

Default configuration

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

Standalone mode | Web container mode

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

Overriding the default ApiProtectionFilter

The following example shows an `admin.json` file configured to override the default `ApiProtectionFilter` that protects the reserved administrative route. This example is used in Set up the UMA example.

Standalone mode | Web container mode

```
{
  "prefix": "openig",
  "connectors": [
    { "port" : 8080 }
  ],
  "heap": [
    {
      "name": "ClientHandler",
      "type": "ClientHandler"
    },
    {
      "name": "ApiProtectionFilter",
      "type": "CorsFilter",
      "config": {
        "policies": [
          {
            "acceptedOrigins": [
"http://app.example.com:8081" ],
            "acceptedMethods": [ "GET", "POST", "DELETE" ],
            "acceptedHeaders": [ "Content-Type" ]
          }
        ]
      }
    }
  ]
}
```

More information

org.forgerock.openig.http.AdminHttpApplication

GatewayHttpApplication (`config.json`)

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 file, expected by default at `$HOME/.openig/config/config.json`. Objects configured in `config.json` can be used by `config.json` and any IG route. They cannot be used by `admin.json`.

If you provide a `config.json`, the IG 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.

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.

Studio deploys and undeploys routes through a main router named `_router`, which is the name of the main router in the default configuration. If you use a custom `config.json`, make sure that it contains a main router named `_router`.

Default objects

IG creates objects by default in `config.json`. To override a default object, configure an object with the same name in `config.json`.

Configure default objects in `config.json` and `admin.json` seperately. An object configured in `config.json` with the same name as an object configured in `admin.json` is not the same object.

`BaseUriDecorator`
: A decorator to override 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.

ProxyOptions
A proxy to which a ClientHandler or ReverseProxyHandler can submit requests, and an AmService can submit Websocket notifications. For more information, see ProxyOptions.

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 Commons Secrets Service. The default SecretsService is a SystemAndEnvSecretStore with the default configuration. For more information, see Secrets.

TemporaryStorage
Manages temporary buffers. 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.

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, session information is stored in the IG cookie, called by default `IG_SESSIONID`.

For more information, see Sessions and JwtSession.

Usage

```
{
  "handler": Handler reference,
  "heap": [ object, ... ],
  "properties": object,
  "secrets": {
```

```
    "stores": [ SecretStore reference, ... ]
  },
  "temporaryStorage": TemporaryStorage reference
}
```

Properties

"handler": Handler reference, required

The Handler to which IG dispaches requests.

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

"heap": array of 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": object, optional

Configuration parameters declared as property variables for use in the configuration. See also Route properties.

Default: Null

"secrets": SecretStore reference, optional

> IMPORTANT
>
> This property is deprecated; use SecretsProvider instead. For more information, refer to Deprecation.

One or more secret stores, as defined in Secrets object and secret stores.

"temporaryStorage": TemporaryStorage reference, optional

The TemporaryStorage object to buffer content during processing.

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

Incoming requests use the temporary storage buffer as follows:

- For standalone mode only when `streamingEnabled` is `false`:
  - The request is loaded into the IG storage defined in `temporaryStorage`, before it enters the chain.
  - If the content length of a request is more than the buffer limit, IG returns an `HTTP 413 Payload Too Large`.
- For web container mode:

- The request is loaded into the web container buffer, which is managed externally.
- IG does not access the web container buffer until a filter, handler, or other object tries to access the request body.
- At that point, IG transfers the content of the web container buffer to its own temporary storage.
- If the web container buffer is bigger than the IG temporary storage, a buffer exception occurs.

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.

Example configuration files

Default configuration

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

```
{
    "heap": [
        {
            "name": "_router",
            "type": "Router",
            "config": {
                "scanInterval": "&{ig.router.scan.interval|10
seconds}",
                "defaultHandler": {
                    "type": "DispatchHandler",
                    "config": {
                        "bindings": [
                            {
                                "condition": "${request.method ==
'GET' and request.uri.path == '/'}",
                                "handler": {
                                    "type": "WelcomeHandler"
                                }
                            },
                            {
                                "condition": "${request.uri.path
== '/'}",
                                "handler": {
                                    "type":
```

```
"StaticResponseHandler",
                                    "config": {
                                        "status": 405,
                                        "reason": "Method Not
Allowed"
                                    }
                                }
                            },
                            {
                                "handler": {
                                    "type":
"StaticResponseHandler",
                                    "config": {
                                        "status": 404,
                                        "reason": "Not Found"
                                    }
                                }
                            }
                        ]
                    }
                }
            }
        ],
        "handler": "_router"
}
```

Notice the following features of the default configuration:

- The handler contains a main router named `_router`. When IG receives an incoming request, `_router` routes the request to the first route in the configuration whose condition is satisfied.
- If the request doesn't satisfy the condition of any route, it is routed to the defaultHandler. If the request is to access the IG welcome page, IG dispatches the request. Otherwise, IG returns an HTTP status 404 (Resource not found), because the requested resource does not exist.

Example config.json used in the doc

The following example of `config.json` is used in many of the examples in the documentation:

```
{
  "handler": {
    "type": "Router",
```

```
    "name": "_router",
    "baseURI": "http://app.example.com:8081",
    "capture": "all"
  },
  "heap": [
    {
      "name": "JwtSession",
      "type": "JwtSession"
    },
    {
      "name": "capture",
      "type": "CaptureDecorator",
      "config": {
        "captureEntity": true,
        "_captureContext": true
      }
    }
  ]
}
```

Notice the following features of the file:

- The handler contains a main router named `_router`. When IG receives an incoming request, `_router` routes the request to the first route in the configuration whose condition is satisfied.
- The `baseURI` changes the request URI to point the request to the sample application.
- The `capture` captures the body of the HTTP request and response.
- The JwtSession object in the heap can be used in routes to store the session information as JSON Web Tokens (JWT) in a cookie. For more information, see JwtSession.

More information

org.forgerock.openig.http.GatewayHttpApplication

Heap objects

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 IG are heap objects.

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

Usage

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

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.

More information

org.forgerock.openig.heap.Heap

Configuration settings

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 IG 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 IG 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 IG starts up or when the route is reloaded. If the configuration setting is optional, there is no error.

In the following example, `"timer"` is defined by an expression that recovers the environment variable `"ENABLE_TIMER"` and transforms it into a boolean. Similarly, `"numberOfRequests"` is defined by an expression that recovers the system property `"requestsPerSecond"` and transforms it into an integer:

```
{
  "name": "throttle-simple-expressions1",
  "timer": "${bool(env['ENABLE_TIMER'])}",
  "baseURI": "http://app.example.com:8081",
  "condition": "${find(request.uri.path, '^/home/throttle-simple-
expressions1')}",
  "handler": {
    "type": "Chain",
    "config": {
      "filters": [
        {
          "type": "ThrottlingFilter",
          "name": "ThrottlingFilter-1",
          "config": {
            "requestGroupingPolicy": "",
            "rate": {
              "numberOfRequests":
"${integer(system['requestsPerSecond'])}",
              "duration": "10 s"
            }
          }
        }
      ],
      "handler": "ReverseProxyHandler"
    }
  }
}
```

If `"requestsPerSecond"=6` and `"ENABLE_TIMER"=true`, after the expressions are evaluated IG views the example route as follows:

```
{
  "name": "throttle-simple-expressions2",
  "timer": true,
  "baseURI": "http://app.example.com:8081",
  "condition": "${find(request.uri.path, '^/home/throttle-simple-
expressions2')}",
  "handler": {
    "type": "Chain",
    "config": {
      "filters": [
        {
          "type": "ThrottlingFilter",
          "name": "ThrottlingFilter-1",
          "config": {
            "requestGroupingPolicy": "",
            "rate": {
              "numberOfRequests": 6,
              "duration": "10 s"
            }
          }
        }
      ],
      "handler": "ReverseProxyHandler"
    }
  }
}
```

For information about expressions, see Expressions.

Supported system properties

The following properties are supported in IG. Their names have a special meaning in IG, and they should be used only for their stated purpose:

ig.instance.dir*, *IG_INSTANCE_DIR

The full path to the directory containing configuration and data for the IG instance.

Default: Linux, `$HOME/.openig`; Windows, %appdata%\OpenIG

org.forgerock.json.jose.jwe.compression.max.decompressed.size.bytes

The maximum size in bytes to which a compressed JWT can be decompressed.

Default: 32 KBytes

org.forgerock.secrets.preferDeterministicEcdsa

When this property is `true`, and the following conditions are met, JWTs are signed with a deterministic Elliptic Curve Digital Signature Algorithm (ECDSA):

- ECDSA is used for signing
- Bouncy Castle is installed

Default: `true`

org.forgerock.http.TrustTransactionHeader

When this property is `true`, all incoming `X-ForgeRock-TransactionId` headers are trusted. Monitoring or reporting systems that consume the logs can allow requests to be correlated as they traverse multiple servers.

Default: `false`

org.forgerock.http.util.ignoreFormParamDecodingError

When this property is `true`, form encoding errors caused by invalid characters are ignored, and encoded values are used instead.

Default: `false`

org.forgerock.json.jose.jwe.compression.max.decompressed.size.bytes

The maximum size in bytes to which a compressed JWT can be decompressed.

Default: 32 KBytes

OPENIG_BASE, openig.base (deprecated)

IMPORTANT

These properties are deprecated; use `ig.instance.dir` or `IG_INSTANCE_DIR` instead. For more information, refer to Deprecation.

The full path to the directory containing configuration and data for the IG instance.

Default: Linux, `$HOME/.openig`; Windows, %appdata%\OpenIG

Handlers

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

Chain

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.

Usage

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

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.

Example

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

More information

org.forgerock.openig.filter.ChainHandlerHeaplet

ClientHandler

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

When IG relays the request to the third-party service, IG is acting as a client of the service. IG is *client-side*. A third-party service is one that IG calls for data, such as an HTTP API or AM, or one to which IG submits data.

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

Use ClientHandler to submit requests to third-party services such as AM or HTTP APIs. Do not use it to proxy requests to a protected application. To proxy requests to a protected application, use a ReverseProxyHandler instead.

When uploading or downloading large files, prevent timeout issues by increasing the value of `soTimeout`, and using a streaming mode, as follows:

- In standalone mode, configure the `streamingEnabled` property of AdminHttpApplication.
- In web container mode, configure the `asyncBehaviour` property in this handler.

Usage

```
{
  "name": string,
  "type": "ClientHandler",
  "config": {
    "vertx": object,
    "connections": configuration expression<number>,
```

```
    "disableReuseConnection": configuration expression<boolean>,
    "stateTrackingEnabled": configuration expression<boolean>,
    "soTimeout": configuration expression<duration>,
    "connectionTimeout": configuration expression<duration>,
    "connectionTimeToLive": configuration expression<duration>,
    "numberOfWorkers": configuration expression<number>,
    "protocolVersion": configuration expression<enumeration>,
    "http2PriorKnowledge": configuration expression<boolean>,
    "proxyOptions": ProxyOptions reference,
    "temporaryStorage": TemporaryStorage reference,
    "tls": ClientTlsOptions reference,
    "asyncBehavior": configuration expression<enumeration>,
    "retries": object,
    "circuitBreaker": object,
    "hostnameVerifier": configuration expression<enumeration>,
//deprecated
    "proxy": Server reference, //deprecated
    "systemProxy": boolean, //deprecated
    "keyManager": KeyManager reference(s), //deprecated
    "sslCipherSuites": array, //deprecated
    "sslContextAlgorithm": string, //deprecated
    "sslEnabledProtocols": array, //deprecated
    "trustManager": TrustManager reference(s) //deprecated
  }
}
```

Properties

"vertx": object, optional

Vert.x-specific configuration for the handler, where IG does not provide its own first-class configuration. Vert.x options are described in HttpClientOptions.

The `vertx` object is read as a map, and values are evaluated as configuration expressions.

For properties where IG provides its own first-class configuration, Vert.x configuration options are disallowed, and the IG configuration option takes precedence over Vert.x options configured in `vertx`. The following Vert.x configuration options are disallowed client-side:

- `port`
- `connectTimeout`
- `idleTimeout`
- `idleTimeoutUnit`

- `protocolVersion`
- `http2ClearTextUpgrade`
- `verifyHost`
- `ssl`
- `enabledSecureTransportProtocols`
- `enabledCipherSuites`
- `keyStoreOptions`
- `keyCertOptions`
- `pemKeyCertOptions`
- `pfxKeyCertOptions`
- `trustOptions`
- `trustStoreOptions`
- `pemTrustOptions`
- `pfxTrustOptions`
- `useAlpn`
- `alpnVersions`

"`connections`": configuration expression<number>, optional

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

Default: 64

"`connectionTimeout`": configuration expression<duration>, optional

Time to wait to establish a connection, expressed as a duration

Default: 10 seconds

"`connectionTimeToLive`": configuration expression<duration>, optional

NOTE

Not supported for IG in standalone mode.

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

"`disableReuseConnection`": configuration expression<boolean>, optional

Not supported in standalone mode.

A flag to disable connection reuse:

- `true` : Connection reuse is disabled
- `false` : Connection reuse is enabled

Default: `false`

"`stateTrackingEnabled`": configuration expression<boolean>, optional

Not supported in standalone mode.

By default, the Apache HTTP Client does not allow connection reuse when a client certificate is used for authentication. However, because the client certificate is defined at the client level, it is acceptable for requests to the same target to share a client certificate.

Use in combination with `disableReuseConnection` :

disableReuseConnection	stateTrackingEnabled	Description
`false` (default)	`true` (default)	Do not allow connection reuse when a client certificate is used for authentication.
`false` (default)	`false`	Allow connection reuse when a client certificate is used for authentication.
`true`	`true` (default) or `false`	Do not allow connection reuse.

Default: `true`

"`numberOfWorkers`": configuration expression<number>, optional

Not supported in standalone mode.

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.

"`protocolVersion`": configuration expression<enumeration>, optional

Not supported in web container mode.

The version of HTTP protocol to use when processing requests:

- `HTTP/2` :
  - For HTTP, process requests using HTTP/1.1.
  - For HTTPS, process requests using HTTP/2.
- `HTTP/1.1` :
  - For HTTP and HTTPS, process requests using HTTP/1.1.
- Not set:
  - For HTTP, process requests using HTTP/1.1.
  - For HTTPS with `alpn` enabled in ClientTlsOptions, process requests using HTTP/1.1, with an HTTP/2 upgrade request. If the targeted server can use HTTP/2, the client uses HTTP/2.

    For HTTPS with `alpn` disabled in ClientTlsOptions, process requests using HTTP/1.1, without an HTTP/2 upgrade request.

    Note that `alpn` is enabled by default in ClientTlsOptions.

Default: Not set

NOTE

In HTTP/1.1 request messages, a `Host` header is required to specify the host and port number of the requested resource. In HTTP/2 request messages, the `Host` header is not available.

In scripts or custom extensions that use HTTP/2, use `UriRouterContext.originalUri.host` or `UriRouterContext.originalUri.port` in requests.

"`http2PriorKnowledge`": configuration expression<boolean>, optional

Not supported in web container mode.

A flag for whether the client should have prior knowledge that the server supports HTTP/2. This property is for cleartext (non-TLS requests) only, and is used only when `protocolVersion` is HTTP/2.

- `false` : The client checks whether the server supports HTTP/2 by sending an HTTP/1.1 request to upgrade the connection to HTTP/2:
  - If the server supports HTTP/2, the server upgrades the connection to HTTP/2, and subsequent requests are processed over HTTP/2.
  - If the server does not support HTTP/2, the connection is not upgraded, and subsequent requests are processed over HTTP/1.
- `true` : The client does not check that the server supports HTTP/2. The client sends HTTP/2 requests to the server, assuming that the server supports HTTP/2.

Default: `false`

"`proxyOptions`": ProxyOptions 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.

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

Default: A heap object named `ProxyOptions`.

"`soTimeout`": configuration expression<duration>, 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

"`temporaryStorage`": TemporaryStorage reference, optional

The TemporaryStorage object to buffer the request and response, when the `streamingEnabled` property of admin.json is `false`.

Default: A heap object named `TemporaryStorage`.

`tls`: ClientTlsOptions reference, optional

IMPORTANT

Use of a TlsOptions reference is deprecated; use ClientTlsOptions instead. For more information, refer to Deprecation.

Configure options for connections to TLS-protected endpoints, based on ClientTlsOptions. Define the object inline or in the heap.

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

`"asyncBehavior": configuration expression<enumeration>, optional

IMPORTANT

Not supported for IG in standalone mode. For information about how to configure streaming for IG in standalone mode, see the `streamingEnabled` property of AdminHttpApplication.

Specifies how the HTTP client behaves for asynchronous responses:

- `streaming` : Responses are processed as soon as all headers are received. The entity content is downloaded in a background thread. The value is not case-sensitive.

  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. The value is not case-sensitive.

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

  TIP

  To prevent timeout issues with large files, consider increasing the value of `soTimeout` , and setting `asyncBehavior` to `streaming` .

Default: `non_streaming`

"retries": object, optional

Enable and configure retry for requests.

During the execution of a request to a remote server, if a runtime exception occurs, or a condition is met, IG waits for a delay, and then schedules a new execution of the request. IG tries until the allowed number of retries is reached or the execution succeeds.

A warning-level entry is logged if all retry attempts fail; a debug-level entry is logged if a retry succeeds.

```
"retries": {
  "enabled": configuration expression<boolean>,
  "condition": runtime expression<boolean>,
  "executor": ScheduledExecutorService reference,
  "count": configuration expression<number>,
  "delay": configuration expression<duration>,
  }
}
```

"enabled": configuration expression<boolean>, optional

Enable retries.

Default: `true`

"condition": runtine expression<boolean>, optional

An inline IG expression to define a condition based on the response, such as an error code.

The condition is evaluated as follows:

- If `true`, IG retries the request until the value in `count` is reached.
- If `false`, IG retries the request only if a runtime exception occurs, until the value in `count` is reached.

Default: `${false}`

"`executor`": ScheduledExecutorService reference, optional

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

Default: `ScheduledExecutorService`

See also ScheduledExecutorService.

"`count`": configuration expression<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.

Retries caused by any runtime exception or triggered condition are included in the count.

Default: `5`

"`delay`": _configuration expression<duration>, optional

The time 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`

The following example configures a retry when a downstream component returns a `502 Bad Gateway` response code:

```
"retries": {
    "enabled": true,
    "condition": "${response.status.code == 502}"
}
```

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
      }
    }
  }
}
```

"circuitBreaker": object, optional

Enable and configure a circuit breaker to trip when the number of failures exceeds a configured threshold. Calls to downstream services are stopped, and a runtime exception is returned. The circuit breaker is reset after the configured delay.

```
{
  "circuitBreaker": {
    "enabled": configuration expression<boolean>,
    "maxFailures":  configuration expression<integer>,
    "openDuration": configuration expression<duration>,
    "openHandler": Handler reference,
    "slidingCounter":  object,
```

```
    "executor":  ScheduledExecutorService reference
  }
}
```

"`enabled`": configuration expression<boolean>, optional
A flag to enable the circuit breaker.

Default: `true`

"`maxFailures`": configuration expression<number>, required
The maximum number of failed requests allowed in the window given by `size`, before the circuit breaker trips. The value must be greater than zero.

IMPORTANT

When `retries` is set, the circuit breaker does not count retried requests as failures. Bear this in mind when you set `maxFailures`.

In the following example, a request can fail and then be retried three times. If it fails the third retry, the request has failed four times, but the circuit breaker counts only one failure.

```
{
  "retries": {
    "count": 3,
    "delay": "1 second"
  }
}
```

"`openDuration`": configuration expression<duration>, required
The duration for which the circuit stays open after the circuit breaker trips. The `executor` schedules the circuit to be closed after this duration.

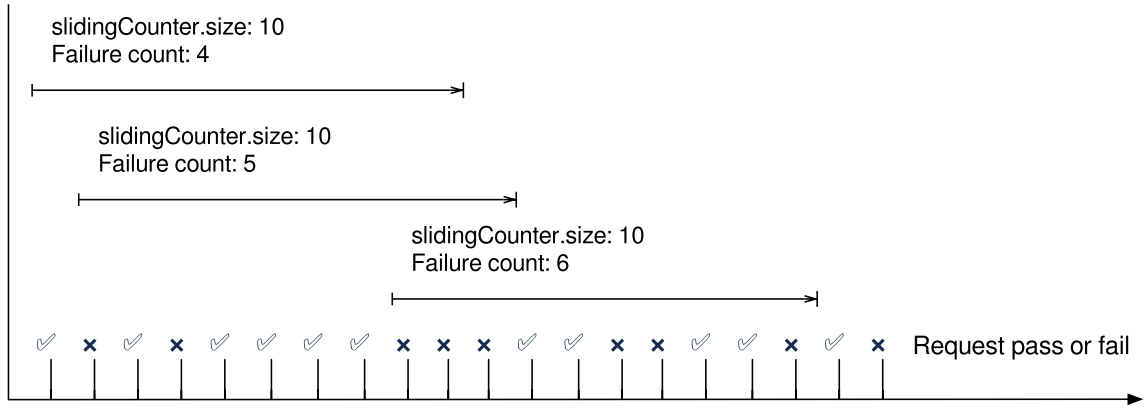
"`openHandler`": Handler reference, optional
The Handler to call when the circuit is open.

Default: A handler that throws a RuntimeException with a "circuit-breaker open" message.

"`slidingCounter`": object, optional

A sliding window error counter. The circuit breaker trips when the number of failed requests in the number of requests given by `size` reaches `maxFailures`.

The following image illustrates how the sliding window counts failed requests:




```
{
  "slidingCounter":  {
    "size": configuration expression<number>
  }
}
```

"`size`": configuration expression<number>, required

The size of the sliding window in which to count errors.

The value of `size` must be greater than zero, and greater than the value of `maxFailures`, otherwise an exception is thrown.

"`executor`": ScheduledExecutorService reference, optional

A ScheduledExecutorService to schedule closure of the circuit after the duration given by `openDuration`.

Default: The default ScheduledExecutorService in the heap

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

IMPORTANT

This property is deprecated; use the `tls` property instead to configure ClientTlsOptions. For more information, refer to Deprecation.

The 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.

  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`

"proxy": Server reference, optional

IMPORTANT

This property is deprecated; use `proxyOptions` instead. For more information, refer to Deprecation.

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,
  "password": configuration expression<string> //deprecated
}
```

"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.

"secretsProvider": SecretsProvider reference, optional

The SecretsProvider to query for the proxy's password.

"password": configuration expression<string>, required if the proxy requires authentication

IMPORTANT

IMPORTANT

This property is deprecated and is not considered secure. Use `passwordSecretId` instead. For more information, refer to Deprecation.

Password to access the proxy server.

"systemProxy": boolean, optional

IMPORTANT

This property is deprecated; use `proxyOptions` instead. For more information, refer to Deprecation.

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.

"keyManager": KeyManager reference(s), optional

IMPORTANT

This property is deprecated; use the `tls` property instead to configure ClientTlsOptions. For more information, refer to Deprecation.

The key manager(s) that handle(s) this client's keys and certificates.

The value of this field can be a single reference, or an array of references.

Provide either the name(s) of KeyManager object(s) defined in the heap, or specify the configuration object(s) inline.

You can specify either a single KeyManager, as in `"keyManager": "MyKeyManager"`, or an array of KeyManagers, as in `"keyManager": [ "FirstKeyManager", "SecondKeyManager" ]`.

If you do not configure a key manager, then the client cannot present a certificate, and so cannot play the client role in mutual authentication.

"`sslCipherSuites`": array of strings, optional

IMPORTANT

This property is deprecated; use the `tls` property instead to configure ClientTlsOptions. For more information, refer to Deprecation.

Array of cipher suite names, used to restrict the cipher suites allowed when negotiating transport layer security for an HTTPS connection.

For details about the available cipher suite names, see the documentation for the Java virtual machine (JVM) used by the container where you run IG. For Oracle Java, see the list of JSSE Cipher Suite Names.

Default: Allow any cipher suite supported by the JVM.

"`sslContextAlgorithm`": string, optional

IMPORTANT

This property is deprecated; use the `tls` property instead to configure ClientTlsOptions. For more information, refer to Deprecation.

The SSLContext algorithm name, as listed in the table of SSLContext Algorithms for the Java Virtual Machine used by the container where IG runs.

Default: TLS

"`sslEnabledProtocols`": array of strings, optional

IMPORTANT

This property is deprecated; use the `tls` property instead to configure ClientTlsOptions. For more information, refer to Deprecation.

Array of protocol names, used to restrict the protocols allowed when negotiating transport layer security for an HTTPS connection.

For details about the available protocol names, see the documentation for the Java virtual machine (JVM) used by the container where you run IG. For Oracle Java, see the list of Additional JSSE Standard Names.

Default: Allow any protocol supported by the JVM.

"`trustManager`": TrustManager reference(s), optional

IMPORTANT

This property is deprecated; use the `tls` property instead to configure ClientTlsOptions. For more information, refer to Deprecation.

The trust managers that handle(s) peers' public key certificates.

The value of this field can be a single reference, or an array of references.

Provide either the name(s) of TrustManager object(s) defined in the heap, or specify the configuration object(s) inline.

You can specify either a single TrustManager, as in `"trustManager": "MyTrustManager"`, or an array of KeyManagers, as in `"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`.

Example

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

```
{
  "name": "Client",
  "type": "ClientHandler",
  "config": {
    "tls": {
      "type": "ClientTlsOptions",
      "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"
              }
            }
          }
        }
      }
    }
  }
```

More information

org.forgerock.openig.handler.ClientHandlerHeaplet

DesKeyGenHandler (deprecated)

IMPORTANT

This object is deprecated and not considered secure. For more information, refer to Deprecation.

Consider using AM's password replay post-authentication plugin class `com.sun.identity.authentication.spi.JwtReplayPassword`. The plugin encrypts the password captured by AM during authentication, and stores it in a session property. IG looks up the property, decrypts it, and replays the password. For an example, see Get login credentials from AM.

Generates a DES key for use with AM.

Usage

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

More information

org.forgerock.openig.handler.DesKeyGenHandler

DispatchHandler

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.

Usage

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

Properties

"bindings": array of objects, required

One or more condition and handler bindings.

"condition": runtime expression<boolean>, optional

A flag to indicate that a condition is met. The condition can be based on the request, context, or IG runtime environment, such as system properties or environment variables.

Conditions are defined using IG expressions, as described in Expressions, and are evaluated as follows:

- `true`: The request is dispatched to the associated handler.
- `false`: The next condition in the list is evaluated.

For example conditions and requests that match them, see Example conditions and requests.

Default: `${true}`

"handler": Handler reference, required

A Handler to dispatch the request to if the associated condition yields `true`.

"baseURI": runtime expression<url>,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

Example

The following sample is from a SAML 2.0 federation configuration:

- If the incoming URI matches `/saml`, then IG dispatches to a SamlFederationHandler without further processing.
- If the previous condition is not met, and the user name is not set in the session context, then IG dispatches the request to a SPInitiatedSSORedirectHandler.

  In this case, the user has not authenticated with the SAML 2.0 Identity Provider, so the SPInitiatedSSORedirectHandler initiates SAML 2.0 SSO from the Service Provider, which is IG.
- If neither of the previous conditions are met, the request goes through a LoginChain handler.

```
{
    "name": "DispatchHandler",
    "type": "DispatchHandler",
    "config": {
        "bindings": [
            {
                "condition": "${find(request.uri.path,
'^/saml')}",
                "handler": "SamlFederationHandler"
            },
            {
                "condition": "${empty session.username}",
                "handler": "SPInitiatedSSORedirectHandler",
                "baseURI": "http://www.example.com:8081"
            },
            {
                "handler": "LoginChain",
                "baseURI": "http://www.example.com:8081"
            }
        ]
    }
}
```

More information

org.forgerock.openig.handler.DispatchHandler

Expressions

ForgeRockClientHandler

The ForgeRockClientHandler is a Handler available by default on the heap that chains a default ClientHandler with a TransactionIdOutboundFilter.

This Handler supports ForgeRock audit by supporting the initiation or propagation of audit information from IG to the audit framework. For more information, see AuditService.

The following default ForgeRockClientHandler is available as a default object on the heap, and can be referenced by the name ForgeRockClientHandler.

```
{
      "name": "ForgeRockClientHandler",
      "type": "Chain",
      "config": {
          "filters": [ "TransactionIdOutboundFilter" ],
          "handler": "ClientHandler"
      }
}
```

Example

For an example that uses ForgeRockClientHandler to log interactions between IG and AM, see Decorating IG's interactions with AM.

More information

org.forgerock.openig.heap.Keys

JwkSetHandler

Expose cryptographic keys cryptographic keys as JWK set. Use this handler so that a downstream application can reuse the exposed keys for their assigned purpose.

Consider the following limitations:

- When the public key is not available, the corresponding private key cannot be exposed. Note, however, that it is not recommended to expose private keys as JWK.

- Keys in secure storage, such as a Hardware Security Module (HSM) or remote server, cannot be exposed.

For a description of how secrets are managed, see Secrets.

For information about JWKs and JWK Sets, see JSON Web Key (JWK).

Usage

```
{
  "name": string,
  "type": "JwkSetHandler",
  "config": {
    "secretsProvider": SecretsProvider reference,
    "purposes": [ object, ... ]
  }
}
```

"secretsProvider": SecretsProvider reference, required
The SecretsProvider to use to resolve the secret. Provide either the name of a SecretsProvider object defined in the heap, or specify a SecretsProvider object inline.

"purposes": array of objects, required
One or more purposes for the secret.

```
{
  "purposes": [
    {
      "secretId": configuration expression<secret-id>,
      "keyUsage": configuration expression<enumeration>
    },
    ...
  ]
}
```

"secretId": configuration expression<secret-id>, required
The secret ID of the key used by the JwtBuilderFilter to sign or encrypt the JWT.

"keyUsage": configuration expression<enumeration>, required
The allowed use of the key:

- `AGREE_KEY` : Export the private key used in the key agreement protocol, for example, Diffie-Hellman.
- `ENCRYPT` : Export the public key used to encrypt data.
- `DECRYPT` : Export the private key used to decrypt data.

- `SIGN` : Export the private key used to sign data.
- `VERIFY` : Export the public key used to verify signature data.
- `WRAP_KEY` : Export the public key used to encrypt (wrap) other keys.
- `UNWRAP_KEY` : Export the private key used to decrypt (unwrap) other keys.

Examples

This example uses a JwkSetHandler to expose a signing key used by the JwtBuilderFilter:

1. Set an environment variable for the base64-encoded secret to sign the JWT:

```
$ export SIGNING_KEY_SECRET_ID='cGFzc3dvcmQ='
```

2. Add the following route to IG:

Linux | Windows

```
$HOME/.openig/config/routes/jwksethandler.json
```

```
{
  "name": "jwksethandler",
  "condition": "${find(request.uri.path,
'/jwksethandler')}",
  "heap": [
    {
      "name": "SecretKeyPropertyFormat-1",
      "type": "SecretKeyPropertyFormat",
      "config": {
        "format": "BASE64",
        "algorithm": "AES"
      }
    },
    {
      "name": "SystemAndEnvSecretStore-1",
      "type": "SystemAndEnvSecretStore",
      "config": {
        "mappings": [{
          "secretId": "signing.key.secret.id",
          "format": "SecretKeyPropertyFormat-1"
        }]
      }
```

```
    }
  ],
  "handler": {
    "type": "Chain",
    "config": {
      "filters": [
        {
          "name": "JwtBuilderFilter-1",
          "type": "JwtBuilderFilter",
          "config": {
            "template": {
              "name":
"${contexts.userProfile.commonName}",
              "email":
"${contexts.userProfile.rawInfo.mail[0]}"
            },
            "secretsProvider": "SystemAndEnvSecretStore-
1",
            "signature": {
              "secretId": "signing.key.secret.id",
              "algorithm": "HS256"
            }
          }
        }
      ],
      "handler": {
        "type": "JwkSetHandler",
        "config": {
          "secretsProvider": "SystemAndEnvSecretStore-1",
          "purposes": [{
            "secretId": "signing.key.secret.id",
            "keyUsage": "SIGN"
          }]
        }
      }
    }
  }
}
```

Notice the following features of the route:

- The route matches requests to `/jwksethandler`.
- The JWT signing key is managed by the SysEnvStoreSecretStore in the heap, which refers to the SecretKeyPropertyFormat for the secret's format.

- The JwtBuilderFilter `signature` property refers to the JWT signing key in the SysEnvStoreSecretStore.
- The JwkSetHandler refers to the JWT signing key.

3. Go to http://ig.example.com:8080/jwksethandler.

   The signing key is displayed as an array, as follows:

```
{
  "keys": [
    {
      "k": "cGFzc3dvcmQ",
      "kid": "signing.key.secret.id",
      "kty": "oct",
      "use": "sig"
    }
  ]
}
```

   The JWK set secret is ULR base64-encoded. Although the secret is set with the value `cGFzc3dvcmQ=`, the value `cGFzc3dvcmQ` is exposed.

More information

org.forgerock.openig.handler.JwkSetHandler

ReverseProxyHandler

Proxy requests to protected applications.

When IG relays the request to the protected application, IG is acting as a client of the application. IG is *client-side*.

If IG 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.

Use ReverseProxyHandler in a route to proxy requests to a protected application. To submit requests to third-party services, such as AM or HTTP APIs, use a ClientHandler instead.

When uploading or downloading large files, prevent timeout issues by increasing the value of `soTimeout`, and using a streaming mode, as follows:

- In standalone mode, configure the `streamingEnabled` property of AdminHttpApplication.

- In web container mode, configure the `asyncBehaviour` property in this handler.

Usage

```
{
  "name": string,
  "type": "ReverseProxyHandler",
  "config": {
    "vertx": object,
    "connections": configuration expression<number>,
    "disableReuseConnection": configuration expression<boolean>,
    "stateTrackingEnabled": configuration expression<boolean>,
    "soTimeout": configuration expression<duration>,
    "connectionTimeout": configuration expression<duration>,
    "connectionTimeToLive": configuration expression<duration>,
    "numberOfWorkers": configuration expression<number>,
    "protocolVersion": configuration expression<enumeration>,
    "http2PriorKnowledge": configuration expression<boolean>,
    "proxyOptions": ProxyOptions reference,
    "temporaryStorage": TemporaryStorage reference,
    "tls": ClientTlsOptions reference,
    "asyncBehavior": configuration expression<enumeration>,
    "retries": object,
    "circuitBreaker": object,
    "websocket": object,
    "hostnameVerifier": configuration expression<enumeration>,
//deprecated
    "proxy": Server reference, //deprecated
    "systemProxy": boolean, //deprecated
    "keyManager": KeyManager reference(s), //deprecated
    "sslCipherSuites": array, //deprecated
    "sslContextAlgorithm": string, //deprecated
    "sslEnabledProtocols": array, //deprecated
    "trustManager": TrustManager reference(s) //deprecated
  }
}
```

Properties

"vertx": object, optional

Vert.x-specific configuration for the handler, where IG does not provide its own first-class configuration. Vert.x options are described in HttpClientOptions.

The `vertx` object is read as a map, and values are evaluated as configuration expressions.

For properties where IG provides its own first-class configuration, Vert.x configuration options are disallowed, and the IG configuration option takes precedence over Vert.x options configured in `vertx`. The following Vert.x configuration options are disallowed client-side:

- `port`
- `connectTimeout`
- `idleTimeout`
- `idleTimeoutUnit`
- `protocolVersion`
- `http2ClearTextUpgrade`
- `verifyHost`
- `ssl`
- `enabledSecureTransportProtocols`
- `enabledCipherSuites`
- `keyStoreOptions`
- `keyCertOptions`
- `pemKeyCertOptions`
- `pfxKeyCertOptions`
- `trustOptions`
- `trustStoreOptions`
- `pemTrustOptions`
- `pfxTrustOptions`
- `useAlpn`
- `alpnVersions`

"`connections`": configuration expression<number>, optional

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

Default: 64

"`connectionTimeout`": configuration expression<duration>, optional

Time to wait to establish a connection, expressed as a duration

Default: 10 seconds

"`connectionTimeToLive`": configuration expression<duration>, optional

NOTE

NOTE

Not supported for IG in standalone mode.

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

"`disableReuseConnection`": configuration expression<boolean>, optional

Not supported in standalone mode.

A flag to disable connection reuse:

- `true` : Connection reuse is disabled
- `false` : Connection reuse is enabled

Default: `false`

"`stateTrackingEnabled`": configuration expression<boolean>, optional

Not supported in standalone mode.

By default, the Apache HTTP Client does not allow connection reuse when a client certificate is used for authentication. However, because the client certificate is defined at the client level, it is acceptable for requests to the same target to share a client certificate.

Use in combination with `disableReuseConnection` :

disableReuseConnection	stateTrackingEnabled	Description
`false` (default)	`true` (default)	Do not allow connection reuse when a client certificate is used for authentication.
`false` (default)	`false`	Allow connection reuse when a client certificate is used for authentication.
`true`	`true` (default) or `false`	Do not allow connection reuse.

Default: `true`

"`numberOfWorkers`": configuration expression<number>, optional

Not supported in standalone mode.

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.

"`protocolVersion`": configuration expression<enumeration>, optional

Not supported in web container mode.

The version of HTTP protocol to use when processing requests:

- `HTTP/2` :
  - For HTTP, process requests using HTTP/1.1.
  - For HTTPS, process requests using HTTP/2.
- `HTTP/1.1` :
  - For HTTP and HTTPS, process requests using HTTP/1.1.
- Not set:
  - For HTTP, process requests using HTTP/1.1.
  - For HTTPS with `alpn` enabled in ClientTlsOptions, process requests using HTTP/1.1, with an HTTP/2 upgrade request. If the targeted server can use HTTP/2, the client uses HTTP/2.

    For HTTPS with `alpn` disabled in ClientTlsOptions, process requests using HTTP/1.1, without an HTTP/2 upgrade request.

    Note that `alpn` is enabled by default in ClientTlsOptions.

Default: Not set

NOTE

In HTTP/1.1 request messages, a `Host` header is required to specify the host and port number of the requested resource. In HTTP/2 request messages, the `Host` header is not available.

In scripts or custom extensions that use HTTP/2, use `UriRouterContext.originalUri.host` or `UriRouterContext.originalUri.port` in requests.

"`http2PriorKnowledge`": configuration expression<boolean>, optional

Not supported in web container mode.

A flag for whether the client should have prior knowledge that the server supports HTTP/2. This property is for cleartext (non-TLS requests) only, and is used only when `protocolVersion` is HTTP/2.

- `false` : The client checks whether the server supports HTTP/2 by sending an HTTP/1.1 request to upgrade the connection to HTTP/2:
  - If the server supports HTTP/2, the server upgrades the connection to HTTP/2, and subsequent requests are processed over HTTP/2.
  - If the server does not support HTTP/2, the connection is not upgraded, and subsequent requests are processed over HTTP/1.
- `true` : The client does not check that the server supports HTTP/2. The client sends HTTP/2 requests to the server, assuming that the server supports HTTP/2.

Default: `false`

"`proxyOptions`": ProxyOptions 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.

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

Default: A heap object named `ProxyOptions` .

"`soTimeout`": configuration expression<duration>, 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

"`temporaryStorage`": TemporaryStorage reference, optional

The TemporaryStorage object to buffer the request and response, when the `streamingEnabled` property of admin.json is `false` .

Default: A heap object named `TemporaryStorage` .

`tls`: ClientTlsOptions reference, optional

IMPORTANT

Use of a TlsOptions reference is deprecated; use ClientTlsOptions instead. For more information, refer to Deprecation.

Configure options for connections to TLS-protected endpoints, based on ClientTlsOptions. Define the object inline or in the heap.

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

`"asyncBehavior": configuration expression<enumeration>, optional

IMPORTANT

Not supported for IG in standalone mode. For information about how to configure streaming for IG in standalone mode, see the `streamingEnabled` property of AdminHttpApplication.

Specifies how the HTTP client behaves for asynchronous responses:

- `streaming` : Responses are processed as soon as all headers are received. The entity content is downloaded in a background thread. The value is not case-sensitive.

  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. The value is not case-sensitive.

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

  TIP

  To prevent timeout issues with large files, consider increasing the value of `soTimeout` , and setting `asyncBehavior` to `streaming` .

Default: `non_streaming`

"`retries`": object, optional

Enable and configure retry for requests.

During the execution of a request to a remote server, if a runtime exception occurs, or a condition is met, IG waits for a delay, and then schedules a new execution of the request. IG tries until the allowed number of retries is reached or the execution succeeds.

A warning-level entry is logged if all retry attempts fail; a debug-level entry is logged if a retry succeeds.

```
"retries": {
  "enabled": configuration expression<boolean>,
```

```
  "condition": runtime expression<boolean>,
  "executor": ScheduledExecutorService reference,
  "count": configuration expression<number>,
  "delay": configuration expression<duration>,
  }
}
```

"`enabled`": configuration expression<boolean>, optional

Enable retries.

Default: `true`

"`condition`": runtine expression<boolean>, optional

An inline IG expression to define a condition based on the response, such as an error code.

The condition is evaluated as follows:

- If `true` , IG retries the request until the value in `count` is reached.
- If `false` , IG retries the request only if a runtime exception occurs, until the value in `count` is reached.

  Default: `${false}`

"`executor`": ScheduledExecutorService reference, optional

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

Default: `ScheduledExecutorService`

See also ScheduledExecutorService.

"`count`": configuration expression<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.

Retries caused by any runtime exception or triggered condition are included in the count.

Default: `5`

"`delay`": _configuration expression<duration>, optional

The time 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`

The following example configures a retry when a downstream component returns a `502 Bad Gateway` response code:

```
"retries": {
    "enabled": true,
    "condition": "${response.status.code == 502}"
}
```

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
      }
    }
  }
}
```

***"circuitBreaker"**: object, optional*

Enable and configure a circuit breaker to trip when the number of failures exceeds a configured threshold. Calls to downstream services are stopped, and a runtime exception is returned. The circuit breaker is reset after the configured delay.

```
{
  "circuitBreaker": {
    "enabled": configuration expression<boolean>,
    "maxFailures":  configuration expression<integer>,
    "openDuration": configuration expression<duration>,
    "openHandler": Handler reference,
    "slidingCounter":  object,
    "executor":  ScheduledExecutorService reference
  }
}
```

"`enabled`": configuration expression<boolean>, optional

A flag to enable the circuit breaker.

Default: `true`

"`maxFailures`": configuration expression<number>, required

The maximum number of failed requests allowed in the window given by `size`, before the circuit breaker trips. The value must be greater than zero.

IMPORTANT

When `retries` is set, the circuit breaker does not count retried requests as failures. Bear this in mind when you set `maxFailures`.

In the following example, a request can fail and then be retried three times. If it fails the third retry, the request has failed four times, but the circuit breaker counts only one failure.

```
{
  "retries": {
    "count": 3,
    "delay": "1 second"
  }
}
```

"`openDuration`": configuration expression<duration>, required

The duration for which the circuit stays open after the circuit breaker trips. The `executor` schedules the circuit to be closed after this duration.

"`openHandler`": Handler reference, optional

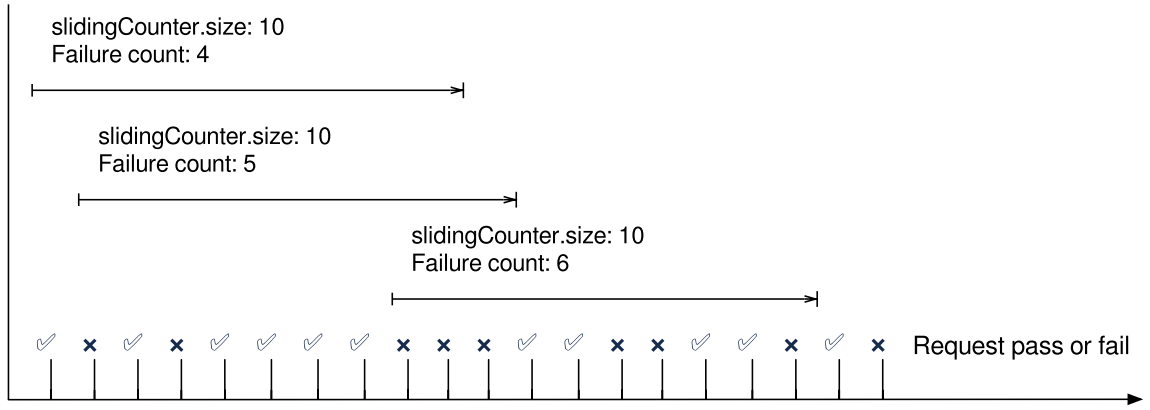
The Handler to call when the circuit is open.

Default: A handler that throws a RuntimeException with a "circuit-breaker open" message.

"`slidingCounter`": object, optional

A sliding window error counter. The circuit breaker trips when the number of failed requests in the number of requests given by `size` reaches `maxFailures`.

The following image illustrates how the sliding window counts failed requests:




```
{
  "slidingCounter":  {
    "size": configuration expression<number>
  }
}
```

"`size`": configuration expression<number>, required

The size of the sliding window in which to count errors.

The value of `size` must be greater than zero, and greater than the value of `maxFailures`, otherwise an exception is thrown.

"`executor`": ScheduledExecutorService reference, optional

A ScheduledExecutorService to schedule closure of the circuit after the duration given by `openDuration`.

Default: The default ScheduledExecutorService in the heap

"`websocket`": object, optional

Object to configure upgrade from HTTP or HTTPS protocol to WebSocket protocol.

Every key/value of the `websocket` object is evaluated as a configuration expression.

In standalone mode, list the subprotocols that are proxied by IG in the `vertx` property of AdminHttpApplication (`admin.json`).

For more information and an example of proxying WebSocket traffic, see Proxy WebSocket traffic

IMPORTANT

When IG is running in Jetty, it cannot proxy WebSocket traffic.

```
{
  "websocket": {
    "enabled": configuration expression<boolean>,
    "connectionTimeout": configuration expression<duration>,
    "soTimeout": configuration expression<duration>,
    "numberOfSelectors": configuration expression<number>,
    "tls": ClientTlsOptions reference,
    "vertx": object,
    "keyManager": KeyManager reference(s), //deprecated
    "sslCipherSuites": array, //deprecated
    "sslContextAlgorithm": string, //deprecated
    "sslEnabledProtocols": array, //deprecated
    "trustManager": TrustManager reference(s) //deprecated
  }
}
```

For more information, see The WebSocket Protocol.

"`enabled`": configuration expression<boolean>,optional

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

Default: `false`

"`connectionTimeout`": configuration expression<duration>, optional

The maximum time allowed to establish a WebSocket connection.

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

"`soTimeout`": configuration expression<duration>, 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`.

"`numberOfSelectors`": configuration expression<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`": ClientTlsOptions reference, optional

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

Default: Use ClientTlsOptions defined for the handler

"`vertx`": object, optional

Vert.x-specific configuration for the WebSocket connection, where IG does not provide its own first-class configuration. Vert.x options are described in HttpClientOptions.

For properties where IG provides its own first-class configuration, Vert.x configuration options are disallowed, and the IG configuration option takes precedence over Vert.x options configured in `vertx`. The following Vert.x configuration options are disallowed client-side:

- `port`
- `connectTimeout`
- `idleTimeout`
- `idleTimeoutUnit`
- `protocolVersion`
- `http2ClearTextUpgrade`
- `verifyHost`
- `ssl`
- `enabledSecureTransportProtocols`
- `enabledCipherSuites`
- `keyStoreOptions`
- `keyCertOptions`
- `pemKeyCertOptions`
- `pfxKeyCertOptions`
- `trustOptions`
- `trustStoreOptions`
- `pemTrustOptions`
- `pfxTrustOptions`
- `useAlpn`

- `alpnVersions`

The following default `vertx` configuration provided by this handler overrides the Vert.x defaults:

- `tryUsePerFrameWebSocketCompression = true`
- `tryUsePerMessageWebSocketCompression = true`

"hostnameVerifier": configuration expression<enumeration>, optional

IMPORTANT

This property is deprecated; use the ClientTlsOptions property `hostnameVerifier` instead.

If a ReverseProxyHandler includes the deprecated `"hostnameVerifier": "ALLOW_ALL"` configuration, it takes precedence, and deprecation warnings are written to the logs.

For more information, refer to Deprecation.

The 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.

  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`

"proxy": Server reference, optional

IMPORTANT

This property is deprecated; use `proxyOptions` instead. For more information, refer to Deprecation.

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,
  "password": configuration expression<string> //deprecated
}
```

"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.

"secretsProvider": SecretsProvider reference, optional

The SecretsProvider to query for the proxy's password.

"password": configuration expression<string>, required if the proxy requires authentication

IMPORTANT

This property is deprecated and is not considered secure. Use `passwordSecretId` instead. For more information, refer to Deprecation.

Password to access the proxy server.

"systemProxy": boolean, optional

IMPORTANT

This property is deprecated; use `proxyOptions` instead. For more information, refer to Deprecation.

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.

"keyManager": KeyManager reference(s), optional

IMPORTANT

This property is deprecated; use the `tls` property instead to configure ClientTlsOptions. For more information, refer to Deprecation.

The key manager(s) that handle(s) this client's keys and certificates.

The value of this field can be a single reference, or an array of references.

Provide either the name(s) of KeyManager object(s) defined in the heap, or specify the configuration object(s) inline.

You can specify either a single KeyManager, as in `"keyManager": "MyKeyManager"`, or an array of KeyManagers, as in `"keyManager": [ "FirstKeyManager", "SecondKeyManager" ]`.

If you do not configure a key manager, then the client cannot present a certificate, and so cannot play the client role in mutual authentication.

"sslCipherSuites": array of strings, optional

IMPORTANT

This property is deprecated; use the `tls` property instead to configure ClientTlsOptions. For more information, refer to Deprecation.

Array of cipher suite names, used to restrict the cipher suites allowed when negotiating transport layer security for an HTTPS connection.

For details about the available cipher suite names, see the documentation for the Java virtual machine (JVM) used by the container where you run IG. For Oracle Java, see the list of JSSE Cipher Suite Names.

Default: Allow any cipher suite supported by the JVM.

"sslContextAlgorithm": string, optional

IMPORTANT

IMPORTANT

This property is deprecated; use the `tls` property instead to configure ClientTlsOptions. For more information, refer to Deprecation.

The SSLContext algorithm name, as listed in the table of SSLContext Algorithms for the Java Virtual Machine used by the container where IG runs.

Default: TLS

"`sslEnabledProtocols`": array of strings, optional

IMPORTANT

This property is deprecated; use the `tls` property instead to configure ClientTlsOptions. For more information, refer to Deprecation.

Array of protocol names, used to restrict the protocols allowed when negotiating transport layer security for an HTTPS connection.

For details about the available protocol names, see the documentation for the Java virtual machine (JVM) used by the container where you run IG. For Oracle Java, see the list of Additional JSSE Standard Names.

Default: Allow any protocol supported by the JVM.

"`trustManager`": TrustManager reference(s), optional

IMPORTANT

This property is deprecated; use the `tls` property instead to configure ClientTlsOptions. For more information, refer to Deprecation.

The trust managers that handle(s) peers' public key certificates.

The value of this field can be a single reference, or an array of references.

Provide either the name(s) of TrustManager object(s) defined in the heap, or specify the configuration object(s) inline.

You can specify either a single TrustManager, as in `"trustManager": "MyTrustManager"`, or an array of KeyManagers, as in `"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`.

More information

org.forgerock.openig.handler.ReverseProxyHandlerHeaplet

ResourceHandler

Serves static content from a directory.

Usage

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

Properties

"directories": array of configuration expression<strings>, 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 expression<strings>, 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

"temporaryStorage": TemporaryStorage reference, optional

A TemporaryStorage object for the static content.

Default: `TemporaryStorage` heap object

Example

The following example serves requests to `http://ig.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://ig.example.com:8080/website`:

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

More information

org.forgerock.openig.handler.resources.ResourceHandler

org.forgerock.http.protocol.Entity

Route

Routes are JSON-encoded configuration files that you add to IG to manage requests. You can add routes in the following ways:

- Manually into the filesystem.
- Through Common REST commands. For information, see Create and Edit Routes Through Common REST.

- Through Studio. For information, see the Studio guide.

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 IG configuration.

For examples of route configurations see Configure routers and routes.

Usage

```
{
  "handler": Handler reference,
  "heap": [ object, ... ],
  "condition": runtime expression<boolean>,
  "name": string,
  "secrets": {
    "stores": [ SecretStore reference, ... ]
  }, //deprecated
  "session": AsyncSessionManager reference,
  "auditService": AuditService reference,
  "globalDecorators": map,
  "decorator name": Decorator object
}
```

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 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

A condition based on the request, context, or IG runtime environment, such as system properties or environment variables.

- `true` : The request is dispatched to the route.
- `false` : The condition for the next route in the configuration is evaluated.
- No condition: the request is dispatched unconditionally to the route.

TIP

For debugging, log the routes for which IG evaluates a condition, and the route that matches a condition. Add the following line to a custom `$HOME/.openig/config/logback.xml` , and restart IG:

```
<logger name="org.forgerock.openig.handler.router.RouterHandler"
level="trace" />
```

For information, see Managing logs.

For example conditions and requests that match them, see Example Conditions and Requests.

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 is ignored: `${find(request.uri.path, '^/openig/my/path')}` .

Default: `${true}`

"`name`": string, optional

Route name.

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.
- When you add a route through Studio, you can edit the default route ID.

CAUTION: The filename of a route cannot be `default.json`. The route `name` property or route ID cannot be `default`.

Default: route ID

"`secrets`": SecretStore reference, optional

IMPORTANT

This property is deprecated; use SecretsProvider instead. For more information, refer to Deprecation.

One or more secret stores, as defined in Secrets object and secret stores.

"`session`": AsyncSessionManager reference. reference, optional

Stateless session implementation for this route. Define an AuthenticatedEncryptedJwtSessionManager object inline or in the heap.

When a request enters the route, IG 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.json`) 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.

For more information, see AsyncSessionManager, and Sessions.

"`auditService`": AuditService reference, optional

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`": map, optional

A map of one or more decorator names to decorator configuration, as described in Decorators. All compatible objects in the route are decorated with the mapped 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: Empty

"decorator name": Decorator object, 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.

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	Monitoring type	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 IG.

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

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

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	Monitoring type	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.

Name	Monitoring type	Description
`response.null`	Counter	Number of responses that were not handled by IG.
`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 and was not executed.
`response.time`	Timer	Time-series summary statistics.

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

Router

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 IG configuration when routes are added, removed, or changed. The router updates the IG configuration without needing to restart IG 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 IG is running.

IMPORTANT

Studio deploys and undeploys routes through a main router named `_router`, which is the name of the main router in the default configuration. If you use a custom `config.json`, make sure that it contains a main router named `_router`.

Usage

```
{
    "name": string,
    "type": "Router",
    "config": {
        "defaultHandler": Handler reference,
        "directory": configuration expression<string>,
        "scanInterval": configuration expression<duration>
    }
}
```

An alternative value for type is RouterHandler.

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, IG aborts the request with an internal error.

See also Handlers.

"`directory`": configuration expression<string>, optional

Directory from which to load route configuration files.

Default: The default directory for route configuration files, at `$HOME/.openig` (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`": configuration expression<duration>, optional

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

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	Monitoring type	Description
`ig_router_deployed_routes`	`Gauge`	Number of routes deployed in the configuration.

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

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	Monitoring type	Description
`deployed-routes`	`Gauge`	Number of routes deployed in the configuration.

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

More information

org.forgerock.openig.handler.router.RouterHandler

SamlFederationHandler

A handler to play the role of SAML 2.0 Service Provider (SP).

Consider the following requirements for SamlFederationHandler:

- This handler does not support filtering; do not use it as the handler for a chain, which can include filters.
- Do not use this handler when its use depends on something in the response. The response can be handled independently of IG, and can be `null` when control returns to IG. For example, do not use this handler in a `SequenceHandler` where the `postcondition` depends on the response.
- Requests to the SamlFederationHandler must not be rebased, because the request URI must match the endpoint in the SAML metadata.

SAML in Deployments With Multiple Instances of IG

IG uses a Java fedlet to implement SAML. When IG acts as a SAML service provider, the session information is stored in the fedlet, not the session cookie. In deployments that use multiple instances of IG as a SAML service provider, it is therefore necessary to set up sticky sessions so that requests always hit the instance where the SAML interaction was started.

For information, see Session state considerations in AM's *SAML v2.0 guide*.

For more information, see About session stickiness and session replication for Tomcat and About session stickiness and session replication for Jetty

Usage

```
{
    "name": string,
    "type": "SamlFederationHandler",
    "config": {
        "assertionMapping": map,
        "redirectURI": configuration expression<url>,
        "secretsProvider": SecretsProvider reference,
        "assertionConsumerEndpoint": configuration
expression<url>,
        "authnContext": configuration expression<string>,
        "authnContextDelimiter": configuration expression<string>,
        "logoutURI": configuration expression<url>,
        "sessionIndexMapping": configuration expression<string>,
        "singleLogoutEndpoint": configuration expression<url>,
        "singleLogoutEndpointSoap": configuration expression<url>,
        "SPinitiatedSLOEndpoint": configuration expression<url>,
        "SPinitiatedSSOEndpoint": configuration expression<url>,
        "useOriginalUri": configuration expression<boolean>,
        "subjectMapping": configuration expression<string>
```

```
    }
}
```

Properties

"assertionMapping": map, required

A name-value map of `localName:incomingName`, where:

- `localName` is a string for the attribute name set in the session
- `incomingName` is a configuration expression<string> for the attribute name set in the incoming assertion

The following example is an `assertionMapping` object:

```
{
    "username": "mail",
    "password": "mailPassword"
}
```

If the incoming assertion contains the statement:

```
mail = demo@example.com
mailPassword = demopassword
```

Then the following values are set in the session:

```
username = demo@example.com
password = demopassword
```

For this to work, edit the `<Attribute name="attributeMap">` element in the SP extended metadata file, `$HOME/.openig/SAML/sp-extended.xml`, so that it matches the assertion mapping configured in the SAML 2.0 Identity Provider (IDP) metadata.

Because the dot character (`.`) serves as a query separator in expressions, do not use dot characters in the localName.

To prevent different handlers from overwriting each others' data, use unique localName settings when protecting multiple service providers.

"redirectURI": configuration expression<url>, required

The page that the filter used to HTTP POST a login form recognizes as the login page for the protected application.

This is how IG and the Federation component work together to provide SSO. When IG detects the login page of the protected application, it redirects to the Federation component. Once the Federation handler validates the SAML exchanges with the IDP, and sets the required session attributes, it redirects back to the login page of the protected application. This allows the filter used to HTTP POST a login form to finish the job by creating a login form to post to the application based on the credentials retrieved from the session attributes.

"`secretsProvider`": SecretsProvider reference, optional

The SecretsProvider object to query for keys when AM provides signed or encrypted SAML assertions.

When this property is not set, the keys are provided by direct keystore look-ups based on entries in the SP extended metadata file, `sp-extended.xml`.

For more information, see SecretsProvider.

Default: Empty.

"`assertionConsumerEndpoint`": configuration expression<string>, optional

Default: `fedletapplication` (same as the Fedlet)

If you modify this attribute, change the metadata to match.

"`authnContext`": configuration expression<string>, optional

Name of the session field to hold the value of the authentication context. Because the dot character (`.`) serves as a query separator in expressions, do not use dot characters in the field name.

Use this setting when protecting multiple service providers, as the different configurations must not map their data into the same fields of `session`. Otherwise different handlers can overwrite each others' data.

As an example, if you set `"authnContext": "myAuthnContext"`, then IG sets `session.myAuthnContext` to the authentication context specified in the assertion. When the authentication context is password over protected transport, then this results in the session containing `"myAuthnContext": "urn:oasis:names:tc:SAML:2.0:ac:classes:PasswordProtectedTransport"`.

Default: map to `session.authnContext`

"`authnContextDelimiter`": configuration expression<string>, optional

The authentication context delimiter used when there are multiple authentication contexts in the assertion.

Default: `|`

"`logoutURI`": configuration expression<string>, optional

Set this to the URI to visit after the user is logged out of the protected application.

You only need to set this if the application uses the single logout feature of the Identity Provider.

"`sessionIndexMapping`": configuration expression<string>, optional

Name of the session field to hold the value of the session index. Because the dot character (`.`) serves as a query separator in expressions, do not use dot characters in the field name.

Use this setting when protecting multiple service providers, as the different configurations must not map their data into the same fields of `session`. Otherwise different handlers can overwrite each others' data.

As an example, if you set `"sessionIndexMapping": "mySessionIndex"`, then IG sets `session.mySessionIndex` to the session index specified in the assertion. This results in the session containing something like `"mySessionIndex": "s24ccbbffe2bfd761c32d42e1b7a9f60ea618f9801"`.

Default: map to `session.sessionIndex`

"`singleLogoutEndpoint`": configuration expression<string>, optional

Default: `fedletSLORedirect` (same as the Fedlet)

If you modify this attribute, change the metadata to match.

"`singleLogoutEndpointSoap`": configuration expression<string>, optional

Default: `fedletSloSoap` (same as the Fedlet)

If you modify this attribute, change the metadata to match.

"`SPinitiatedSLOEndpoint`": configuration expression<string>, optional

Default: `SPInitiatedSLO`

If you modify this attribute, change the metadata to match.

"`SPinitiatedSSOEndpoint`": configuration expression<string>, optional

Default: `SPInitiatedSSO`

If you modify this attribute, change the metadata to match.

"`useOriginalUri`": configuration expression<boolean>, optional

When `true`, use the original URI instead of a rebased URI when validating RelayState and Assertion Consumer Location URLs. Use this property if a `baseUri` decorator is used in the route or in `config.json`.

Default: `false`

"`subjectMapping`": configuration expression<string>, optional

Name of the session field to hold the value of the subject name. Because the dot character (`.`) serves as a query separator in expressions, do not use dot characters in the field name.

Use this setting when protecting multiple service providers, as the different configurations must not map their data into the same fields of `session`. Otherwise different handlers can overwrite each others' data.

As an example, if you set `"subjectMapping": "mySubjectName"`, then IG sets `session.mySubjectName` to the subject name specified in the assertion. If the subject name is an opaque identifier, then this results in the session containing something like `"mySubjectName": "vtOk+APj1s9Rr4yCka6V9pGUuzuL"`.

Default: map to `session.subjectName`

Example

For an example of how to set up IG as a SAML service provider, see SAML 2.0 single sign-on and federation.

In the following example, IG receives a SAML 2.0 assertion from the IDP, and then logs the user in to the protected application using the username and password from the assertion:

```
{
    "name": "SamlFederationHandler",
    "type": "SamlFederationHandler",
    "config": {
        "assertionMapping": {
            "username": "mail",
            "password": "mailPassword"
        },
        "redirectURI": "/login",
        "logoutURI": "/logout"
    }
}
```

More information

org.forgerock.openig.handler.saml.SamlFederationHandler

ScriptableHandler

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.

Usage

```
{
    "name": string,
    "type": "ScriptableHandler",
    "config": {
        "type": configuration expression<string>,
        "file": configuration expression<string>, // Use either
"file"
        "source": [ string, ... ],                // or "source",
but not both
        "args": map,
        "clientHandler": Handler reference
    }
}
```

Properties

For information about properties for ScriptableHandler, see Scripts.

More information

org.forgerock.openig.handler.ScriptableHandler

SequenceHandler

Processes a request through a sequence of handlers and post conditions, as follows:

- A request is treated by `handler1`, and then `postcondition1` is evaluated.
- If `postcondition1` is true, the request is then treated by `handler2`, and so on.

```
{
  "handler": handler1,
  "postcondition": expression1
},
{
  "handler": handler2,
  "postcondition": expression2
},
...
```

Use this handler for multi-request processing, such as retrieving a form, extracting form content (for example, a nonce), and then submitting it in a subsequent request.

Usage

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

Properties

"bindings": array of objects, required
A list of handler and postcondition bindings.

"handler": Handler reference, required
The handler to dispatch the request to when it is the first handler in the bindings, or for subsequent handlers when their previous `postcondition` yields `true`.

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

"postcondition": runtime expression<boolean>, optional
A flag to indicate that a post condition is met:

- `true`: The request is dispatched to the next handler in `bindings`.
- `false`: The sequence stops.

Postconditions are defined using IG expressions, as described in Expressions.

Default: `${true}`

More information

org.forgerock.openig.handler.SequenceHandler

StaticResponseHandler

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

Usage

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

Properties

"status": Status object
The response status. For more information, see Status.

"reason": configuration expression<string>, optional
Used only for custom HTTP status codes. For more information, see Response Status Codes and Status Code Registry.

"headers": map, optional
One or more headers to set for a response, with the format `name: [ value, … ]`, where:

- *name* is a configuration expression<string> for a header name. If multiple expressions resolve to the same final string, *name* has multiple values.
- *value* one or more a runtime expression<strings> for header values.

When the property `entity` is used, set a `Content-Type` header with the correct content type value. The following example sets the content type of a message entity in the response:

```
"headers": {
  "Content-Type": [ "text/html; charset=UTF-8" ]
```

```
}
```

The following example is used in `05-federate.json` to redirect the original URI from the request:

```
"headers": {
  "Location": [
     "http://sp.example.com:8080/saml/SPInitiatedSSO"
  ]
}
```

CAUTION

For IG in web container mode, sending or receiving headers or trailers whose names or values contain non-ASCII characters can cause unspecified behavior at the HTTP server level, and character corruption at the ClientHandler/ReverseProxyHandler level.

Default: Empty

"trailers": map, optional

One or more trailers to set for a response, with the format `name: [ value, … ],` where:

- *name* is a configuration expression<string> for a trailer name. If multiple expressions resolve to the same string, *name* has multiple values.

  The following trailer names are not allowed:

  - Message framing headers (for example, `Transfer-Encoding` and `Content-Length`)
  - Routing headers (for example, `Host`)
  - Request modifiers (for example, controls and conditionals such as `Cache-Control`, `Max-Forwards`, and `TE`)
  - Authentication headers (for example, `Authorization` and `Set-Cookie`)
  - `Content-Encoding`
  - `Content-Type`
  - `Content-Range`
  - `Trailer`

- *value* is one or more runtime expression<strings> for trailer values.

CAUTION

CAUTION

For IG in web container mode, sending or receiving headers or trailers whose names or values contain non-ASCII characters can cause unspecified behavior at the HTTP server level, and character corruption at the ClientHandler/ReverseProxyHandler level.

Default: Empty

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

The message entity body to include in a response.

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

Methods are provided for accessing the entity as byte, string, or JSON content. For information, see Entity.

IMPORTANT

Attackers during reconnaissance can use response messages to identify information about a deployment. For security, limit the amount of information in messages, and avoid using words that help identify IG.

Default: Empty

Example

```
{
  "name": "ErrorHandler",
  "type":"StaticResponseHandler",
  "config": {
    "status": 500,
    "headers": {
      "Content-Type": [ "text/html; charset=UTF-8" ]
    },
    "entity": "<html><h2>Epic #FAIL</h2></html>"
  }
}
```

More information

org.forgerock.openig.handler.StaticResponseHandler

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

AllowOnlyFilter

Authorizes a request to continue processing if it satisfies at least one of the configured rules. Otherwise, passes the request to the FailureHandler or returns an HTTP 401 Unauthorized, with an empty response body.

This filter manages requests from the *last request sender*, otherwise called the *request from the last hop*, or the *request from a direct client*.

For debugging, configure the AllowOnlyFilter `name`, and add the following logger to `logback.xml`, replacing filter_name with the name:

```
org.forgerock.openig.filter.allow.AllowOnlyFilter.filter_name
```

For more information, see Managing logs.

Usage

```
{
  "name": string,
  "type": "AllowOnlyFilter",
  "config": {
    "rules": [ object, ... ],
    "failureHandler": Handler reference
  }
}
```

Properties

"rules": array of objects, required

An array of one or more `rules` configuration objects to specify criteria for the request.

When more than one `rules` configuration object is included in the array, the request must match at least one of the configuration objects.

When more than one property is specified in the `rules` configuration (for example, `from` and `destination`) the request must match criteria for each property.

```
{
  "rules": [
    {
      "name": configuration expression<string>,
      "from": [ object, ... ],
      "destination": [ object, ... ],
      "when": configuration expression<boolean>
    },
    ...
  ]
}
```

"name": configuration expression<string>, optional

A name for the `rules` configuration. When logging is configured for the AllowOnlyFilter, the rule name appears in the logs.

"from": array of objects, required

An array of one or more `from` configuration objects to specify criteria about the last request sender (the direct client).

When more than one `from` configuration object is included in the array, the last request sender must match at least one of the configuration objects.

When both `ip` and `certificate` properties are included in the configuration, the last request sender must match criteria for both properties.

```
"from": [
  {
    "ip": {
      "list": [configuration expression<string>, ...],
      "resolver": runtime expression<string>
    },
    "certificate" : {
      "subjectDNs" : Pattern[]
    }
  },
  ...
]
```

"ip": object, optional

Criteria about the IP address of the last request sender.

"`list`": array of configuration expression<strings>, required:

An array of IP addresses or IP address ranges, using IPv4 or IPv6, and CIDR notation. The following example includes different formats:

```
"list": ["127.0.0.1", "::1", "192.168.0.0/16",
"1234::/16"]
```

The IP address of the last request sender must match at least one of the specified IP addresses or IP address ranges.

"`resolver`": runtime expression<string>, optional:

An expression that returns an IP address as a string. The following example returns an IP address from the first item in `X-Forwarded-For`:

```
"resolver": "${request.headers['X-Forwarded-For']
[0]}"
```

Default: Resolve the IP address from the following items, in the following order:

I. If there is a `Forwarded` header, use the IP address of the last hop.

II. Otherwise, if there is an `X-Forwarded-For` header, use the IP address of the last hop.

III. Otherwise, use the IP address of the connection.

"`certificate`": array of objects, optional

An array of `certificate` configuration objects that specify criteria about the certificate of the last request sender.

"`subjectDNs`": array of patterns, required:

An array of patterns to represent the expected distinguished name of the certificate subject, the `subjectDN`. The `subjectDN` of the last request sender must match at least one of the patterns.

"`destination`": array of objects, optional

An array of `destination` configuration objects to specify criteria about the request destination.

When more than one `destination` configuration object is included in the array, the request destination must match at least one of the configuration objects.

When more than one property is specified in the `destination` configuration, for example `hosts` and `ports`, the request destination must match criteria for each property.

```
"destination": [
  {
    "hosts": [pattern, ... ],
    "ports": [configuration expression<string>, ... ],
    "methods": [configuration expression<string>, ... ],
    "paths": [pattern, ... ]
  },
  ...
]
```

"`hosts`": array of patterns, optional

An array of *case-insensitive* patterns to match the `request.host` attribute. Patterns are matched with the Java Pattern class.

When this property is configured, the request destination must match at least one host pattern in the array.

Default: Any host is allowed.

"`ports`": array of configuration expression<strings>, optional

An array of strings to match the `request.port` attribute. Specify values in the array as follows:

- Array of single ports, for example `["80", "90"]`.
- Array of port ranges, for example `["100:200"]`.
- Array of single ports and port ranges, for example `["80", "90", "100:200"]`.

When this property is configured, the destination port must match at least one entry in the array.

Default: Any port is allowed.

"`methods`": array of configuration expression<strings>, optional

An array of HTTP methods to match the `request.method` attribute.

When this property is configured, the request method must match at least one method in the array.

Default: Any method is allowed.

"`paths`": array of patterns, optional

An array of *case-sensitive* patterns to match the `request.url_path` attribute. Patterns are matched with the Java Pattern class.

When this property is configured, the destination path must match at least one path in the array.

Default: Any path is allowed.

"when": runtime expression<boolean>, optional

A flag to indicate that the request meets a condition. When `true`, the request is allowed.

The following condition is met when the first value of `h1` is `1`:

```
"when": "${request.headers['h1'][0] == '1'}"
```

Default: `${true}`

"failureHandler": Handler reference, optional

Handler to treat the request if none of the declared rules are satisfied.

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

Default: HTTP 401 Unauthorized, with an empty response body.

See also Handlers.

Examples

In the following example, a request is authorized if the last request sender satisfies *either* of the following conditions:

- Certificate subjectDN matches `.*CN=test$` *or* `CN=me`, *and* the IP address is in the range 1.2.3.0/24.
- IP address is 123.43.56.8.

```
"from": [
   {
     "certificate": {
          "subjectDNs": [".*CN=test$", "CN=me"]
     },
     "ip": {
       "list": ["1.2.3.0/24"]
     }
   },
   {
     "ip": {
       "list": ["123.43.56.8"]
     }
```

```
    },
]
```

In the following example, a request is authorized if the request destination satisfies *all* of the following conditions:

- The host is `myhost1.com` *or* `www.myhost1.com`
- The port is `80`.
- The method is `POST` *or* `GET`
- The path matches `/user/*`.

```
"destination": [
  {
    "hosts": ["myhost1.com", "www.myhost1.com"],
    "ports": ["80"],
    "methods": ["POST", "GET"],
    "paths": ["/user/*"]
  }
]
```

The following example authorizes a request to continue processing if the requests meets the conditions set by *either* `rule1` or `rule2`:

```
{
  "type": "AllowOnlyFilter",
  "config": {
    "rules": [
      {
        "name": "rule1",
        "from": [
          {
            "certificate": {
              "subjectDNs": [".*CN=test$", "CN=me"]
            },
            "ip": {
              "list": ["1.2.3.0/24"]
            }
          }
        ],
        "destination": [
          {
```

```
            "hosts": ["myhost1.com", "www.myhost1.com"],
            "ports": ["80"],
            "methods": ["POST", "GET"],
            "paths": ["/user/*"]
          }
        ],
        "when": "${request.headers['h1'][0] == '1'}"
      },
      {
        "name":"rule2",
        "when": "${request.headers['h1'][0] == '2'}"
      }
    ]
  }
}
```

More information

org.forgerock.openig.filter.allow.AllowOnlyFilter

AmSessionIdleTimeoutFilter

Forces the revocation of AM sessions that have been idle for a specified duration. The AmSessionIdleTimeoutFilter issues an authenticated and encrypted JWT to track activity on the AM session, and conveys it within a persistent cookie. The tracking token contains the following parts:

- The time when the user was last seen active
- A hash of the AM session cookie, to bind the tracking token to the AM session cookie
- The idle timeout

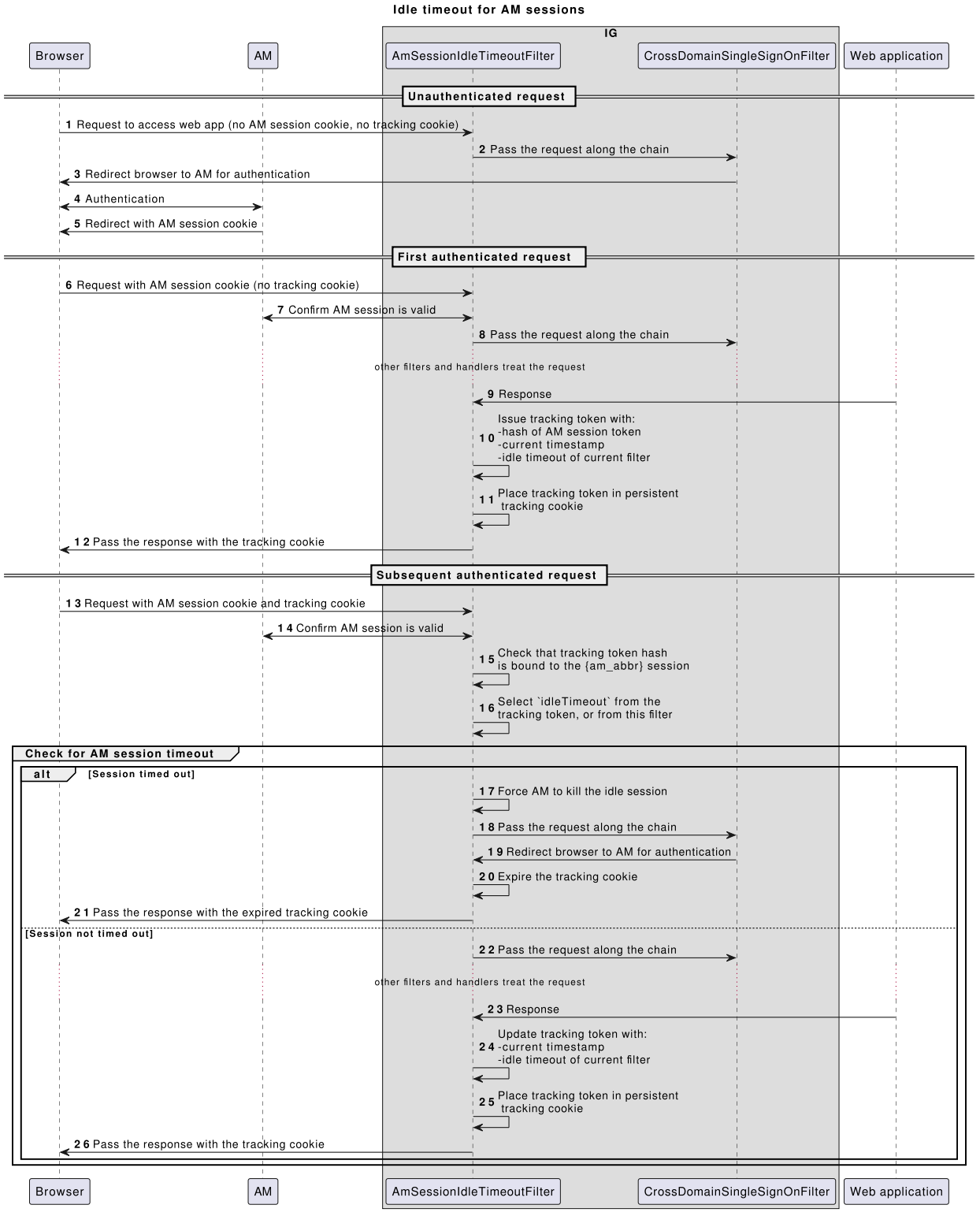
Multiple filter instances can share the same tracking token, for example, in a clustered IG configuration, or when a federation of applications protected by authentication filters need to have a flexible idle timeout strategy.

AmSessionIdleTimeoutFilter requires the following configuration:

- In AM, client-side sessions must be enabled for the realm in which the tracking token operates. See Configure client-side sessions in AM's *Sessions Guide*.
- In AM, client-side session denylisting must be enabled. See Configure client-side session denylisting in AM's *Sessions Guide*.

- The AmSessionIdleTimeoutFilter must be placed in a route before a filter that uses the AM session token, such as a SingleSignOnFilter or PolicyEnforcementFilter.
- In production environments, and when multiple AmSessionIdleTimeoutFilters use the same tracking token, the encryption must not rely on the default configuration. It must be configured identically on each filter that uses the tracking token.

The following image shows the flow of information when an AmSessionIdleTimeoutFilter sits in front of a CrossDomainSingleSignOnFilter, to manage AM session timeout.



[1-5] When the AmSessionIdleTimeoutFilter receives an unauthenticated request, it passes the request along the chain, and the CrossDomainSingleSignOnFilter manages authentication.

[6-8] When the AmSessionIdleTimeoutFilter receives an authenticated request, it checks that the AM session token is valid, and then passes the request along the chain.

[9-10] If the AM session was valid, the AmSessionIdleTimeoutFilter issues a tracking token on the response flow, containing the following information:

- Hash of the AM session token
- Current timestamp
- Idle timeout of the current filter

If the AM session was not valid, the AmSessionIdleTimeoutFilter does nothing on the response flow.

[11-12] The AmSessionIdleTimeoutFilter places the tracking token in persistent tracking cookie, and sends it with the response, to be used in the next request.

[13-15] When the same or another AmSessionIdleTimeoutFilter receives an authenticated request with a tracking token, it checks that the AM session token is valid, and checks that tracking token hash is bound to the AM session.

[16] Depending on the strategy set by `idleTimeoutUpdate`, the AmSessionIdleTimeoutFilter selects the value for `idleTimeout` from the tracking token (set by the AmSessionIdleTimeoutFilter in a previous request) or from its own value of `idleTimeout` (if this is a different instance of AmSessionIdleTimeoutFilter).

The AmSessionIdleTimeoutFilter checks for AM session timeout. If the last activity time plus the idle timeout is before the current time, the session has timed out. For example, a session with the following values has timed out:

- last activity time: 15h30 today
- idle timeout: 5 mins
- current time: 15h40

[17-21] The AM session has timed out, so the AmSessionIdleTimeoutFilter does the following:

- Forces AM to revoke the session.
- Passes the request along the chain.
- Expires the tracking cookie on the response flow, and sends it with the response.

[22-26] The session has not timed out, so the AmSessionIdleTimeoutFilter does the following:

- Passes the request along the chain.
- Updates the tracking token on the response flow, with the current timestamp and the value for `idleTimeOut`, using the same value for that was selected in step 16.
- Places the tracking token in a persistent tracking cookie, and sends it with the response, to be used in the next request.

Usage

```
{
  "name": string,
   "type": "AmSessionIdleTimeoutFilter",
   "config": {
     "amService": AmService reference,
     "idleTimeout": configuration expression<duration>,
     "sessionToken": runtime expression<string>,
     "idleTimeoutUpdate": configuration expression<enumeration>,
     "secretsProvider": SecretsProvider reference,
     "encryptionSecretId": configuration expression<secret-id>,
     "encryptionMethod": configuration expression<string>,
     "cookie": object
   }
}
```

Properties

"amService": AmService reference, required

The AmService that refers to the AM instance that issue tracked session token.

"idleTimeout": configuration expression<duration>, required

The time a session can be inactive before it is considered as idle.

When an AmSessionIdleTimeoutFilter creates the tracking token, the token's value for `idleTimeout` is set by this property. When a different AmSessionIdleTimeoutFilter accesses the same tracking token, depending on the strategy set by `idleTimeoutUpdate`, the token's value for `idleTimeout` can be updated by the second AmSessionIdleTimeoutFilter.

"sessionToken": runtime expression<string>, optional

The location of the AM session token in the request. The following example accesses the first value of the request cookie `iPlanetDirectoryPro`:

```
"sessionToken": "${request.cookies['iPlanetDirectoryPro']
[0].value}"
```

For information about the AM session cookie, see Find the name of your AM session cookie.

Default: `${request.cookies['<cookie name defined in the referenced AmService>'][0].value}`

`idleTimeoutUpdate`: configuration expression<enumeration>, required

When multiple AmSessionIdleTimeoutFilters use the same tracking token, this property selects whether to use the `idleTimeout` from this filter or from the tracking token.

Use one of the following values:

- `NEVER` : Use the idle timeout from the tracking token, and ignore the idle timeout from this filter.
- `ALWAYS` : Use the idle timeout from this filter, and ignore the idle timeout from the tracking token.
- `INCREASE_ONLY` : Compare the idle timeout from this filter and the tracking token, and use the longest value.
- `DECREASE_ONLY` : Compare the idle timeout from this filter and the tracking token, and use the shortest value.

Default: `ALWAYS`

"`secretsProvider`": SecretsProvider reference, optional

The SecretsProvider to query for secrets to encrypt the tracking token.

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

The secret ID of the encryption key used to encrypt the tracking cookie.

In production environments, and when multiple AmSessionIdleTimeoutFilters use the same tracking cookie, the encryption must not rely on the default configuration. It must be configured identically on each filter that uses the cookie.

Authenticated encryption is achieved with a symmetric encryption key. Therefore, the secret must refer to a symmetric key.

For more information, see RFC 5116.

Default: When no `secretsProvider` is provides, IG generates a random symmetric key for authenticated encryption.

"`encryptionMethod`": configuration expression<string>, optional

The algorithm to use for authenticated encryption. For information about allowed encryption algorithms, see RFC 7518: "enc" (Encryption Algorithm) Header Parameter Values for JWE.

Default: A256GCM

"`cookie`": object, optional

Configuration of the activity tracking 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

The cookie name.

Default: `x-ig-activity-tracker`

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

Domain to which the cookie applies.

Default: The fully qualified hostname of the IG host.

"`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 to apply to the cookie.

Default: `/`

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

Options to manage the circumstances in which a cookie is sent to the server. Use one of the following values to reduce the risk of CSRF attacks:

- `STRICT` : Send the cookie only if the request was initiated from the cookie domain. Not case-sensitive. Use this value to reduce the risk of cross-site request forgery (CSRF) attacks.

- `LAX` : Send the cookie only with GET requests in a first-party context, where the URL in the address bar matches the cookie domain. Not case-sensitive. Use this value to reduce the risk of cross-site request forgery (CSRF) attacks.
- `NONE` : Send the cookie whenever a request is made to the cookie domain. With this setting, consider setting `secure` to `true` to prevent browsers from rejecting the cookie. For more information, see SameSite cookies.

Default: Null

"secure": configuration expression<boolean>, optional

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

Default: `false`

Example

```
{
  "type": "AmSessionIdleTimeoutFilter",
  "config": {
    "sessionToken": "${request.cookies['iPlanetDirectoryPro']
[0].value}",
    "amService": "AmService",
    "idleTimeout": "1 minute",
    "idleTimeoutUpdate": "ALWAYS",
    "cookie": {
      "name": "x-ig-activity-tracker",
      "domain": null,
      "path": "/",
      "secure": false,
      "httpOnly": true,
      "sameSite": null
    },
   "secretsProvider": "secrets-provider-ref",
   "encryptionMethod": "A256GCM",
   "encryptionSecretId": "crypto.key.secret.id"
  }
}
```

More information

org.forgerock.openig.openam.session.AmSessionIdleTimeoutFilter

AssignmentFilter

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.

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
      }, ...
    ]
  }
}
```

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.

Default: `${true}`

"target": <lvalue-expression>, required

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

"value": runtime expression<object> , 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.

Examples

Add info to a session

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

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

Capture and store login credentials

The following example captures credentials and stores them in the IG 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.queryParams['username'][0]}"
      },
      {
        "target": "${session.authPassword}",
        "value": "${request.queryParams['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}"
      }
    ]
  }
}
```

More information

org.forgerock.openig.filter.AssignmentFilter

AuthorizationCodeOAuth2ClientFilter

The AuthorizationCodeOAuth2ClientFilter 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.

AuthorizationCodeOAuth2ClientFilter 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.

Service URIs

Service URIs are constructed from the `clientEndpoint` , 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, and then the goto URI.

The goto URL must use the same scheme, host, and port as the original URI, or be a relative URI (just the path). Otherwise, the request fails with an error.

To redirect a request to a site that does not meet the goto URL criteria, change the original URI by using a ForwardedRequestFilter.

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 successful, the filter saves the authorization state in the session and redirects the user-agent to the goto URL.

The goto URL must use the same scheme, host, and port as the original URI, or be a relative URI (just the path). Otherwise, the request fails with an error.

To redirect a request to a site that does not meet the goto URL criteria, change the original URI by using a ForwardedRequestFilter.

clientEndpoint/logout?goto=url

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

The goto URL must use the same scheme, host, and port as the original URI, or be a relative URI (just the path). Otherwise, the request fails with an error.

To redirect a request to a site that does not meet the goto URL criteria, change the original URI by using a ForwardedRequestFilter.

If no goto URL is specified in the request, use `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 provided to the login endpoint during login.

Other request URIs

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

Usage

```
{
  "name": string,
  "type": "AuthorizationCodeOAuth2ClientFilter",
  "config": {
    "clientEndpoint": runtime expression<uri string>,
    "failureHandler": Handler reference,
    "loginHandler": Handler reference,
    "registrations": [ ClientRegistration reference, ... ],
    "metadata": object,
    "cacheExpiration": configuration expression<duration>,
    "executor": ScheduledExecutorService reference,
    "target": lvalue-expression,
    "defaultLoginGoto": runtime expression<url>,
    "defaultLogoutGoto": runtime expression<url>,
    "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>,
    "oAuth2SessionKey": configuration expression<string>,
    "secretsProvider": SecretsProvider reference
  }
}
```

Properties

"clientEndpoint": runtime expression<url>, 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": "${find(request.uri.path, '^/discovery')}"
```

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

- As a sub path of the route condition:

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

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

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

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 AuthorizationCodeOAuth2ClientFilter tries to refresh the access token:

- If the token is refreshed, the AuthorizationCodeOAuth2ClientFilter 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 AuthorizationCodeOAuth2ClientFilter 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": [ "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 optional

List of client registrations that authenticate IG 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": <object>, required for dynamic client registration and ignored otherwise

The values of the object are evaluated as configuration expression<strings>.

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 configuration expression<url>, required

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

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

"client_name": configuration expression<string>, optional

Name of the client to present to the end user.

"scope": _configuration expression<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, or with authorization servers that support RFC 7591, *OAuth 2.0 Dynamic Client Registration Protocol*

"cacheExpiration": configuration expression<duration>, optional

Duration for which to cache user-info resources.

IG lazily fetches user info from the OpenID provider. In other words, IG only fetches the information when a downstream Filter or Handler uses the user info. Caching allows IG 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.

"`executor`": ScheduledExecutorService reference, optional

A ScheduledExecutorService to schedule the execution of tasks, such as the eviction of entries in the OpenID Connect user information cache.

Default: `ScheduledExecutorService`

"`target`": 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`: Value of the OAuth 2.0 access token
- `scope`: Scopes associated with the OAuth 2.0 access token
- `token_type`: Authentication token type; for example, `Bearer`
- `expires_in`: Number of milliseconds until the OAuth 2.0 access token expires
- `id_token`: Value of the OpenID Connect token
- `id_token_claims`: Claims used in the OpenID Connect token
- `client_endpoint`: URL to the client endpoint
- `client_registration`: Client ID of the OAuth 2.0 client that enables IG to communicate as an OAuth 2.0 client with an authorization server
- `user_info`: Profile attributes of an authenticated user; for example, `sub`, `name`, `family_name`

Data is provided to the target as follows:

- If the authorization process completes successfully, the AuthorizationCodeOAuth2ClientFilter 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}"
        ]
      }
    }
}
```

For information about setting up this example, see Authenticate Automatically to the Sample Application.

- 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<url>,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 goto URL must use the same scheme, host, and port as the original URI, or be a relative URI (just the path). Otherwise, the request fails with an error.

To redirect a request to a site that does not meet the goto URL criteria, change the original URI by using a ForwardedRequestFilter.

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.

"`defaultLogoutGoto`": runtime expression<url>,optional

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

The goto URL must use the same scheme, host, and port as the original URI, or be a relative URI (just the path). Otherwise, the request fails with an error.

To redirect a request to a site that does not meet the goto URL criteria, change the original URI by using a ForwardedRequestFilter.

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.

"`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 IG 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`.

"`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 HTTP basic access authentication, 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. IG 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: Error Response.
- 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, it is `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, it is not used.

"`oAuth2SessionKey`": configuration expression<string>, optional

A key to identify an OAuth 2.0 session. The key can be any character string.

To share the same OAuth 2.O session when a user accesses different applications protected by IG, use the same key in each filter.

Default: The complete client endpoint URI. AuthorizationCodeOAuth2ClientFilters do not share OAuth 2.O sessions.

"`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.

Examples

For examples, see the following sections:

- Use AM As a single OpenID connect provider
- Use multiple OpenID Connect providers
- Discover and dynamically register with OpenID connect providers

More information

org.forgerock.openig.filter.oauth2.client.OAuth2ClientFilter

Issuer

ClientRegistration

The OAuth 2.0 Authorization Framework

The OAuth 2.0 Authorization Framework: Bearer Token Usage

OpenID Connect site, in particular the list of standard OpenID Connect 1.0 scope values.

CapturedUserPasswordFilter

Makes an AM password available to IG 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}`.

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>,
      "key": configuration expression<string> //deprecated
    }
}
```

Properties

"amService": AmService reference, required

The AmService heap object to use for the password. See also, AmService.

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

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

"keyType": configuration expression<enumeration>, required

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

- `AES` AES for JWT-based AES_128_CBC_HMAC_SHA_256 encryption. For more information, see AES_128_CBC_HMAC_SHA_256 in the IETF *JSON Web Algorithms*.
- `DES` for DES/ECB/NoPadding

IMPORTANT

This value is deprecated, and considered unsecure. For more information, refer to Deprecation.

"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 Default secrets object.

"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.

"key": configuration expression<string>, required

IMPORTANT

IMPORTANT

This property is deprecated and is not considered secure. Use `keySecretId` instead. For more information, refer to Deprecation.

Base64 encoded key value to decrypt the AM password.

Examples

The following example route is used to get login credentials from AM in Get login credentials from AM.

```
{
  "name": "04-replay",
  "condition": "${find(request.uri.path, '^/replay')}",
  "heap": [
    {
      "name": "SystemAndEnvSecretStore-1",
      "type": "SystemAndEnvSecretStore"
    },
    {
      "name": "AmService-1",
      "type": "AmService",
      "config": {
        "agent": {
          "username": "ig_agent",
          "passwordSecretId": "agent.secret.id"
        },
        "secretsProvider": "SystemAndEnvSecretStore-1",
        "url": "http://am.example.com:8088/openam/"
      }
    },
    {
      "name": "CapturedUserPasswordFilter",
      "type": "CapturedUserPasswordFilter",
      "config": {
        "ssoToken": "${contexts.ssoToken.value}",
        "keySecretId": "aes.key",
        "keyType": "AES",
        "secretsProvider": "SystemAndEnvSecretStore-1",
        "amService": "AmService-1"
      }
    }
  ],
  "handler": {
    "type": "Chain",
```

```
    "config": {
      "filters": [
        {
          "type": "SingleSignOnFilter",
          "config": {
            "amService": "AmService-1"
          }
        },
        {
          "type": "PasswordReplayFilter",
          "config": {
            "loginPage": "${true}",
            "credentials": "CapturedUserPasswordFilter",
            "request": {
              "method": "POST",
              "uri": "http://app.example.com:8081/login",
              "form": {
                "username": [
                  "${contexts.ssoToken.info.uid}"
                ],
                "password": [
                  "${contexts.capturedPassword.value}"
                ]
              }
            }
          }
        }
      ],
      "handler": "ReverseProxyHandler"
    }
  }
}
```

More information

org.forgerock.openig.openam.CapturedUserPasswordFilter

org.forgerock.openig.openam.CapturedUserPasswordContext

CapturedUserPasswordContext

SessionInfoFilter

CertificateThumbprintFilter

Extracts a Java certificate from a trusted header or from a TLS connection, computes the SHA-256 thumbprint of that certificate, and makes the thumbprint available for the ConfirmationKeyVerifierAccessTokenResolver. Use this filter to enable verification of certificate-bound access tokens.

CertificateThumbprintFilter computes and makes available the SHA-256 thumbprint of a client certificate as follows:

- Evaluates a runtime expression and yields a `java.security.cert.Certificate`
- Hashes the certificate using SHA-256
- Base64url-encodes the result
- Stores the result in the contexts chain

The runtime expression can access or build a client certificate from any information present at runtime, such as a PEM in a header, or a pre-built certificate.

Use CertificateThumbprintFilter with ConfirmationKeyVerifierAccessTokenResolver when the IG instance is behind the TLS termination point, for example, when IG is running behind a load balancer or other ingress point.

Usage

```
{
  "name": string,
  "type": "CertificateThumbprintFilter",
  "config": {
    "certificate": runtime expression<certificate>,
    "failureHandler": Handler reference,
  }
}
```

Properties

"certificate": runtime expression<certificate>, required

An EL expression which, when evaluated, yields an instance of a `java.security.cert.Certificate`.

Use the following Functions in the expression to define hash, decoding, and certificate format:

- `digestSha256`, to calculate the SHA-256 hash of the certificate.
- `decodeBase64url`, to decode an incoming base64url-encoded string.
- `pemCertificate`, to convert a PEM representation string into a certificate.

See Examples.

"failureHandler": Handler reference, optional

Handler to treat the request on failure.

Provide an inline handler configuration object, or the name of a handler object declared in the heap. See also Handlers.

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

Examples

The following example use the certificate associated with the incoming HTTP connection:

```
{
  "name": "CertificateThumbprintFilter-1",
  "type": "CertificateThumbprintFilter",
  "config": {
    "certificate": "${contexts.client.certificates[0]}"
  }
}
```

The following example is adapted for a deployment with NGINX as the TLS termination, where NGINX fronts IG. NGINX provides the client certificate associated with its own incoming connection in the `x-ssl-client-cert` header. The certificate is encoded as PEM, and then url-encoded:

```
{
  "name": "CertificateThumbprintFilter-2",
  "type": "CertificateThumbprintFilter",
  "config": {
    "certificate": "${pemCertificate(urlDecode(request.headers['x-ssl-client-cert'][0]))}"
  }
}
```

More information

org.forgerock.openig.filter.oauth2.cnf.CertificateThumbprintFilter

CircuitBreakerFilter

Monitors failures. When the number of failures reaches a configured failure threshold, the circuit breaker trips, and the circuit is considered *open*. Calls to downstream filters

are stopped, and a runtime exception is returned.

After a configured delay, the circuit breaker is reset, and is the circuit considered *closed*. Calls to downstream filters are restored.

Usage

```
{
  "name": string,
  "type": "CircuitBreakerFilter",
  "config": {
    "maxFailures":  configuration expression<integer>,
    "openDuration": configuration expression<duration>,
    "openHandler": Handler reference,
    "slidingCounter": object,
    "executor":  ScheduledExecutorService reference
  }
}
```

Properties

"maxFailures": configuration expression<number>, required

The maximum number of failed requests allowed in the window given by `size`, before the circuit breaker trips. The value must be greater than zero.

"openDuration": configuration expression<duration>, required

The duration for which the circuit stays open after the circuit breaker trips. The `executor` schedules the circuit to be closed after this duration.

"openHandler": Handler reference, optional

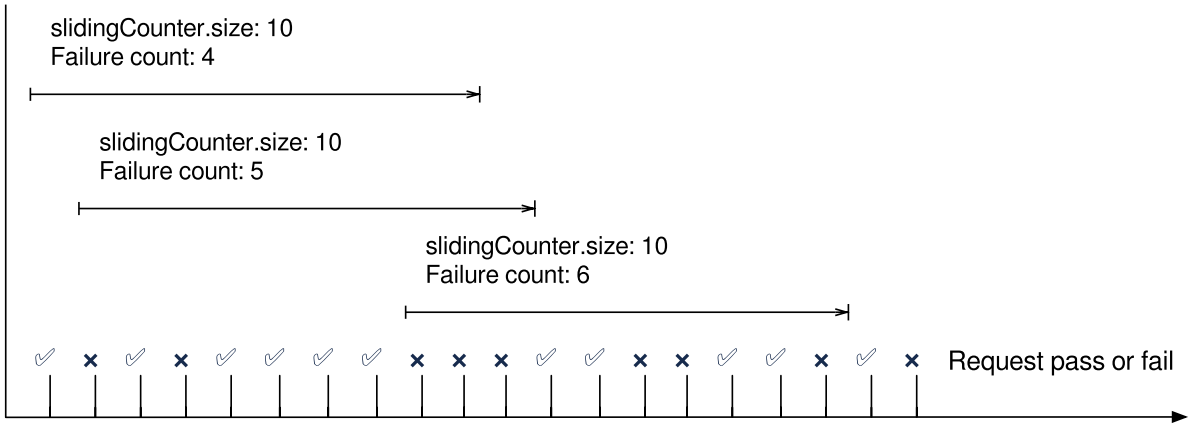
The Handler to call when the circuit is open.

Default: A handler that throws a RuntimeException with a "circuit-breaker open" message.

"slidingCounter": object, optional

A sliding window error counter. The circuit breaker trips when the number of failed requests in the number of requests given by `size` reaches `maxFailures`.

The following image illustrates how the sliding window counts failed requests:




```
{
  "slidingCounter":  {
    "size": configuration expression<number>
  }
}
```

"size": configuration expression<number>, required

The size of the sliding window in which to count errors.

The value of `size` must be greater than zero, and greater than the value of `maxFailures`, otherwise an exception is thrown.

"executor": ScheduledExecutorService reference, optional

A ScheduledExecutorService to schedule closure of the circuit after the duration given by `openDuration`.

Default: The default ScheduledExecutorService in the heap

Example

In the following example, the circuit breaker opens after 11 failures in the previous 100 requests, throwing a runtime exception with a "circuit-breaker open" message. The default ScheduledExecutorService in the heap closes the circuit-breaker after 10 seconds.

```
{
  "type": "CircuitBreakerFilter",
  "config": {
    "maxFailures": 10,
    "openDuration": "10 seconds",
    "openHandler": {
      "type": "StaticResponseHandler",
      "config": {
        "status": 500,
```

```
        "headers": {
          "Content-Type": [ "text/plain" ]
        },
        "entity": "Too many failures; circuit opened to protect
downstream services."
      }
    },
    "slidingCounter": {
      "size": 100
    }
  }
}
```

More information

org.forgerock.openig.filter.circuitbreaker.CircuitBreakerFilter

ClientCredentialsOAuth2ClientFilter

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 access token is valid for the configured scopes.

The ClientCredentialsOAuth2ClientFilter obtains the client's access token by using the `client_credentials` grant type. Client authentication is provided by the `endpointHandler` property, which uses a client authentication filter, such as ClientSecretBasicAuthenticationFilter. The filter refreshes the access token as required.

Use the ClientCredentialsOAuth2ClientFilter in a service-to-service context, where services need to access resources protected by OAuth 2.0.

Usage

```
{
  "name": string,
  "type": "ClientCredentialsOAuth2ClientFilter",
  "config": {
    "secretsProvider": SecretsProvider reference,
    "tokenEndpoint": configuration expression<url>,
    "scopes": [ configuration expression<string>, ... ],
    "endpointHandler": Handler reference,
    "clientId": configuration expression<sting>, //deprecated
    "clientSecretId": configuration expression<secret-id>,
```

```
//deprecated
    "handler": Handler reference //deprecated
  }
}
```

Properties

"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<strings>, optional

Array of scope strings to request from the authorization server.

Default: Empty, request no scopes.

"endpointHandler": Handler reference, optional

The Handler to exchange tokens on the authorization endpoint.

Configure this property as a Chain, using one of the following client authentication filters:

- ClientSecretBasicAuthenticationFilter
- ClientSecretPostAuthenticationFilter
- PrivateKeyJwtClientAuthenticationFilter

```
{
  "name": "myHandler",
  "type": "Chain",
  "config": {
    "handler": "ForgeRockClientHandler",
    "filters": [
      {
        "type": "ClientSecretBasicAuthenticationFilter",
        "config": {
          "clientId": "myConfidentialClient",
          "clientSecretId": "my.client.secret.id",
          "secretsProvider" : "mySystemAndEnvSecretStore",
        }
      }
    ]
```

```
    }
}
```

Default: ForgeRockClientHandler

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

IMPORTANT

This property is deprecated. Use `endpointHandler` instead. For more information, refer to Deprecation.

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

If you use the deprecated properties, provide `clientId`, `clientSecretId` to obtain the client secret, which authenticates using the `client_secret_basic` method.

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

IMPORTANT

This property is deprecated. Use `endpointHandler` instead. For more information, refer to Deprecation.

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

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

IMPORTANT

This property is deprecated. Use `endpointHandler` instead. For more information, refer to Deprecation.

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

Examples

For an example, refer to Using OAuth 2.0 client credentials.

More information

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

org.forgerock.openig.filter.oauth2.OAuth2ResourceServerFilterHeaplet

OAuth2ResourceServerFilter

The OAuth 2.0 Authorization Framework

The OAuth 2.0 Authorization Framework: Bearer Token Usage

ClientSecretBasicAuthenticationFilter

Supports client authentication with the method `client_secret_basic`. Clients that have received a `client_secret` value from the authorization server authenticate through the HTTP basic access 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=...
```

Use this filter with an endpoint `Handler` that requires `client_secret_basic` authentication. For example, `endpointHandler` in the OAuth2TokenExchangeFilter or ClientCredentialsOAuth2ClientFilter.

Usage

```
{
  "name": string,
  "type": "ClientSecretBasicAuthenticationFilter",
  "config": {
    "clientId": configuration expression<string>,
    "clientSecretId": configuration expression<secret-id>,
    "secretsProvider": SecretsProvider reference
  }
}
```

Configuration

"clientId": configuration expression<string>, required
The OAuth 2.0 client ID to use for authentication.

"clientSecretId": configuration expression<secret-id>, required
The OAuth 2.0 client secret to use for authentication.

"`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.

Example

```
{
  "name": "ExchangeHandler",
  "type": "Chain",
  "config": {
    "handler": "ForgeRockClientHandler",
    "filters": [
      {
        "type": "ClientSecretBasicAuthenticationFilter",
        "config": {
          "clientId": "serviceConfidentialClient",
          "clientSecretId": "client.secret.id",
          "secretsProvider" : "SystemAndEnvSecretStore-1"
        }
      }
    ]
  }
}
```

ClientSecretPostAuthenticationFilter

Supports client authentication with the method `client_secret_post`. Clients that have received a `client_secret` value from the authorization server authenticate 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=...
```

Use this filter with an endpoint `Handler` that requires `client_secret_post` authentication. For example, `endpointHandler` in the OAuth2TokenExchangeFilter or ClientCredentialsOAuth2ClientFilter.

Usage

```
{
  "name": string,
  "type": "ClientSecretPostAuthenticationFilter",
  "config": {
    "clientId": configuration expression<string>,
    "clientSecretId": configuration expression<secret-id>,
    "secretsProvider": SecretsProvider reference
  }
}
```

Configuration

"clientId": configuration expression<string>, required
The OAuth 2.0 client ID to use for authentication.

"clientSecretId": configuration expression<secret-id>, required
The OAuth 2.0 client secret to use for authentication.

"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.

ConditionalFilter

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.

Usage

```
{
    "name": string,
    "type": "ConditionalFilter",
    "config": {
        "condition": runtime expression<boolean>,
        "delegate": Filter reference
```

```
    }
}
```

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.

"delegate": Filter reference, required

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

See also Filters.

Example

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

```
{
  "type": "Chain",
  "config": {
    "filters": [{
      "type": "ConditionalFilter",
      "config": {
        "condition": "${not (find(request.uri.path, '.js$') or
find(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.

More information

org.forgerock.openig.filter.ConditionalFilter

ConditionEnforcementFilter

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 IG returns 403 Forbidden and the request is stopped.

Usage

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

Properties

"condition": runtime expression<boolean>, required

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

"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 declared in the heap. See also Handlers.

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

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"
```

```
    }
}
```

More information

org.forgerock.openig.filter.ConditionEnforcementFilter

ChainOfFilters

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.

Usage

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

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.

Example

```
{
    "name": "MyChainOfFilters",
    "type": "ChainOfFilters",
```

```
    "config": {
        "filters": [ "Filter1", "Filter2" ]
    }
}
```

More information

org.forgerock.openig.filter.ChainFilterHeaplet

CookieFilter

Manages, suppresses, and relays cookies as follows:

- **Manage**, to store cookies from the protected application in the IG 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 the AM session cookie, that are used by IG 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.

Usage

```
{
    "name": string,
    "type": "CookieFilter",
    "config": {
        "managed": [ configuration expression<string>, ... ],
        "suppressed": [ configuration expression<string>, ... ],
```

```
        "relayed": [ configuration expression<string>, ... ],
        "defaultAction": configuration expression<enumeration>
    }
}
```

Properties

"managed": array of configuration expression<strings>, optional
A list of the names of cookies to be managed.

"suppressed": array of configuration expression<strings>, optional
A list of the names of cookies to be suppressed.

"relayed": array of configuration expression<strings>, optional
A list of the names of cookies to be relayed.

"defaultAction": configuration expression<enumeration>, optional
Action to perform for cookies that do not match an action set. Set to `"MANAGE"`, `"RELAY"`, or `"SUPPRESS"`. Default: `"MANAGE"`.

More information

org.forgerock.openig.filter.CookieFilter

CorsFilter

Configures policies for cross-origin resource sharing (CORS), to allow cross-domain requests from user agents.

Usage

```
{
   "name": string,
   "type": "CorsFilter",
   "config": {
     "policies": [ object, ... ],
     "failureHandler": Handler reference
   }
}
```

Properties

"policies": array of objects, required

One or more policies to apply to the request. A policy is selected when the origin of the request matches the accepted `origins` of the policy.

When multiple policies are declared, they are tried in the order that they are declared, and the first matching policy is selected.

```
{
  "acceptedOrigins": [ configuration expression<url>, ... ] or
"*",
  "acceptedMethods": [ configuration expression<string>, ... ]
or "*",
  "acceptedHeaders": [ configuration expression<string>, ... ]
or "*",
  "exposedHeaders": [ configuration expression<string>, ... ],
  "maxAge": configuration expression<duration>,
  "allowCredentials": configuration expression<boolean>,
  "origins": [ configuration expression<url>, ... ] or "*"
//deprecated
}
```

"acceptedOrigins": array of configuration expression<urls> or "*", required

A comma-separated list of *origins*, to match the origin of the CORS request. Alternatively, use `*` to allow requests from any URL.

If the request origin is not in the list of accepted origins, the failure handler is invoked or an HTTP 403 Forbidden is returned, and the request stops being executed.

Origins are URLs with a scheme, hostname, and optionally a port number, for example, `http://www.example.com`. If a port number is not defined, origins with no port number or with the default port number (80 for HTTP, 443 for HTTPS) are accepted.

Examples:

```
{
  "acceptedOrigins": [
    "http://www.example.com",
    "https://example.org:8433"
  ]
}
```

```
{
  "acceptedOrigins": "*"
}
```

"`acceptedMethods`": array of configuration expression<strings> or `"*"`, optional

A comma-separated list of case-sensitive HTTP method names that are allowed when making CORS requests. Alternatively, use `*` to allow requests with any method.

In preflight requests, browsers use the `Access-Control-Request-Method` header to let the server know which HTTP method might be used in the actual request.

- If all requested methods are allowed, the requested methods are returned in the preflight response, in the `Access-Control-Allow-Methods` header.
- If any requested method is not allowed, the `Access-Control-Allow-Methods` header is omitted. The failure handler is not invoked, but the user-agent interprets the preflight response as a CORS failure.

Examples:

```
{
  "acceptedMethods": [
    "GET",
    "POST",
    "PUT",
    "MyCustomMethod"
  ]
}
```

```
{
  "acceptedMethods": "*"
}
```

Default: All methods are rejected.

"`acceptedHeaders`": array of configuration expression<strings> or `"*"`, optional

A comma-separated list of case-insensitive request header names that are allowed when making CORS requests. Alternatively, use `*` to allow requests with any header.

In preflight requests, browsers use the `Access-Control-Request-Headers` header to let the server know which HTTP headers might be used in the actual request.

- If all requested headers are allowed, the requested headers are returned in the preflight response, in the `Access-Control-Allow-Headers` header.
- If any requested header is not allowed, the `Access-Control-Allow-Headers` header is omitted. The failure handler is not invoked, but the user-agent interprets the preflight response as a CORS failure.

Examples:

```
{
  "acceptedHeaders": [
    "iPlanetDirectoryPro",
    "X-OpenAM-Username",
    "X-OpenAM-Password",
    "Accept-API-Version",
    "Content-Type",
    "If-Match",
    "If-None-Match"
  ]
}
```

```
{
  "acceptedHeaders": "*"
}
```

Default: All requested headers are rejected.

*"**exposedHeaders**": list of configuration expression<string>, optional*

A comma-separated list of case-insensitive response header names that are returned in the `Access-Control-Expose-Headers` header.

Only headers in this list, safe headers, and the following simple response headers are exposed to frontend JavaScript code:

- `Cache-Control`
- `Content-Language`
- `Expires`
- `Last-Modified`
- `Pragma`
- `Content-Type`

Example:

```
{
  "exposedHeaders": [
    "Access-Control-Allow-Origin",
    "Access-Control-Allow-Credentials",
    "Set-Cookie"
  ]
}
```

Default: No headers are exposed.

"`maxAge`": configuration expression<duration>, optional

The maximum duration for which a browser is allowed to cache a preflight response. The value is included in the `Access-Control-Max-Age` header of preflight responses.

When this `maxAge` is greater than the browser's maximum internal value, the browser value takes precedence.

Default: 5 seconds

"`allowCredentials`": configuration expression<boolean>, optional

A flag to allow requests that use credentials, such as cookies, authorization headers, or TLS client certificates.

Set to `true` to set the `Access-Control-Allow-Credentials` header to `true`, and allow browsers to expose the response to frontend JavaScript code.

Default: False

"`origins`": list of configuration expression<url> or "", required*

> IMPORTANT
>
> This property is deprecated; use `acceptedOrigins` instead. For more information, refer to Deprecation.

A comma-separated list of origins, to match the origin of the CORS request. Alternatively, use `*` to allow requests from any URL.

Origins are URLs with a scheme, hostname, and optionally a port number, for example, `http://www.example.com`. If a port number is not defined, origins with no port number or with the default port number (80 for HTTP, 443 for HTTPS) are accepted.

"`failureHandler`": Handler reference, optional

Handler invoked during the preflight request, when the request origin does not match any of the `acceptedOrigins` defined in `policies`.

The failure handler is not invoked when requested headers or requested methods are not allowed.

Provide an inline handler configuration object, or the name of a handler object declared in the heap. See also Handlers.

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

More information

org.forgerock.http.filter.cors.CorsFilter

org.forgerock.openig.filter.CorsFilter

https://fetch.spec.whatwg.org/#http-cors-protocol

CrossDomainSingleSignOnFilter

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

For an example of how to configure CDSSO in AM and IG, and information about the flow of data between AM, IG, and a protected application, see Authenticate with CDSSO.

WebSocket notifications for sessions

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

For information about setting up WebSocket notifications, using them to clear the session cache, and including them in the server logs, see WebSocket notifications.

Usage

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

```
    }
}
```

Properties

"amService": AmService reference, required

The AmService heap object to use. See AmService.

"redirectEndpoint": configuration expression<url>, required

The URI to which AM redirects the browser with the authentication token or an authentication error. The filter checks that the authentication was initiated by IG.

Configure this URI to be the same as that in AM.

To make sure that the redirect is routed back to the CrossDomainSingleSignOnFilter, 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": "${find(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": "${find(request.uri.path, '^/home/cdsso')}"
  ```

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

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

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

"`authenticationService`": configuration expression<string>,optional

The name of an AM authentication tree or authentication chain to use for authentication.

Default: AM's default authentication service.

For an example that uses `authenticationService`, see Authenticate with SSO through an AM authentication tree.

For more information about authentication trees and chains, see Authentication nodes and trees and Authentication modules and chains in AM's *Authentication and SSO* guide.

"`authCookie`": object, optional

The configuration of the cookie used to store the authentication.

```
{
  "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

Options to manage the circumstances in which a cookie is sent to the server. Use one of the following values to reduce the risk of CSRF attacks:

- `STRICT`: Send the cookie only if the request was initiated from the cookie domain. Not case-sensitive.

  Use this value to reduce the risk of cross-site request forgery (CSRF) attacks.
- `LAX`: Send the cookie only with GET requests in a first-party context, where the URL in the address bar matches the cookie domain. Not case-sensitive.

  Use this value to reduce the risk of cross-site request forgery (CSRF) attacks.
- `NONE`: Send the cookie whenever a request is made to the cookie domain. Not case-sensitive.

  With this setting, consider setting `secure` to `true` to prevent browsers from rejecting the cookie. For more information, see SameSite cookies.

Default: `LAX`

NOTE

For CDSSO, set `"sameSite":"none"` and `"secure":"true"`. For security reasons, many browsers require the connection used by the browser to be secure (HTTPS) for `"sameSite":"none"`. Therefore, if the connection used by the browser is not secure (HTTP), the browser might not supply cookies with `"sameSite":"none"`.

"`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`

"`defaultLogoutLandingPage`": configuration expression<url>, optional

The URL to which a request is redirected if `logoutExpression` is evaluated as true.

If this property is not an absolute URL, the request is redirected to the IG domain name.

This parameter is effective only when `logoutExpression` is specified.

Default: None, processing continues.

"`logoutExpression`": runtime expression<boolean>, optional

A flag to indicate that a logout condition is met. The condition is based on the request:

- `true` : The AM session token for the end user is revoked. If a `defaultLogoutLandingPage` is specified, the request is redirected to that page.
- `false` : 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` :

```
${find(request.uri, '/logout')}
```

- The request path starts with `/logout` :

```
${find(request.uri.path, '^/logout')}
```

- The request query includes the `logOff=true` query parameter:

```
${find(request.uri.query, 'logOff=true')}
```

Default: `${false}`

"`failureHandler`": Handler reference, optional

Handler to treat the request if an error occurs during authentication.

If an error occurs 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.

If no failure handler is configured, 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 configured, the token must be signed. If not configured, IG does not verify the signature.

For information about how signatures are validated, see Validating the signature of signed tokens. For information about how each type of secret store resolves named secrets, see Secrets.

"`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.

Example

In the following example from Single sign-on and cross-domain single sign-on, IG uses authentication from AM on a different domain to process a request:

```
{
  "name": "cdsso",
  "baseURI": "http://app.example.com:8081",
  "condition": "${find(request.uri.path, '^/home/cdsso')}",
  "heap": [
    {
      "name": "SystemAndEnvSecretStore-1",
      "type": "SystemAndEnvSecretStore"
    },
    {
      "name": "AmService-1",
      "type": "AmService",
```

```
      "config": {
        "url": "http://am.example.com:8088/openam",
        "realm": "/",
        "version": "7.2",
        "agent": {
          "username": "ig_agent_cdsso",
          "passwordSecretId": "agent.secret.id"
        },
        "secretsProvider": "SystemAndEnvSecretStore-1",
        "sessionCache": {
          "enabled": false
        }
      }
    }
  ],
  "handler": {
    "type": "Chain",
    "config": {
      "filters": [
        {
          "name": "CrossDomainSingleSignOnFilter-1",
          "type": "CrossDomainSingleSignOnFilter",
          "config": {
            "redirectEndpoint": "/home/cdsso/redirect",
            "authCookie": {
              "path": "/home",
              "name": "ig-token-cookie"
            },
            "amService": "AmService-1",
            "verificationSecretId": "verify",
            "secretsProvider": {
              "type": "JwkSetSecretStore",
              "config": {
                "jwkUrl":
"http://am.example.com:8088/openam/oauth2/connect/jwk_uri"
              }
            }
          }
        }
      ],
      "handler": "ReverseProxyHandler"
    }
  }
}
```

More information

org.forgerock.openig.openam.SingleSignOnFilter

CdSsoContext

CdSsoFailureContext

SsoTokenContext

CryptoHeaderFilter (deprecated)

IMPORTANT

This object is deprecated and is not considered secure. For more information, refer to Deprecation.

The CryptoHeaderFilter conveys encrypted data between hosts by using insecure ECB mode ciphers. Consider using a JwtBuilderFilter with a HeaderFilter for a more secure way to pass identity or other runtime information to the protected application.

Encrypts or decrypts headers in a request or response, using a symmetric or asymmetric key. CryptoHeaderFilter supports key rotation.

Usage

```
{
  "name": string,
  "type": "CryptoHeaderFilter",
  "config": {
    "messageType": configuration expression<enumeration>,
    "operation": configuration expression<enumeration>,
    "keySecretId": configuration expression<secret-id>,
    "secretsProvider": SecretsProvider reference,
    "algorithm": configuration expression<string>,
    "charset": configuration expression<string>,
    "headers": [ configuration expression<string>, ... ]
  }
}
```

Properties

"messageType": configuration expression<enumeration>, required

The type of message whose headers to encrypt or decrypt.

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

"`operation`": configuration expression<enumeration>, required
Indication of whether to encrypt or decrypt.

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

"`keySecretId`": configuration expression<secret-id>, required
The secret ID of the key to encrypt or decrypt the headers.

"`secretsProvider`": SecretsProvider reference, required
The SecretsProvider object to query for the key to encrypt or decrypt the headers. For more information, see SecretsProvider.

"`algorithm`": configuration expression<string>, optional
The algorithm name, mode, and padding used for encryption and decryption.

CryptoHeaderFilter does not support EC-based encryption. Use other cipher algorithm values given in Java Security Standard Algorithm Names.

Default: `AES/ECB/PKCS5Padding`

"`charset`": configuration expression<string>, optional
The name of the charset used to encrypt or decrypt values, as described in Class Charset.

Default: `UTF-8`

"`headers`": array of configuration expression<strings>, optional
The names of header fields to encrypt or decrypt.

Default: Empty

Example

```
{
  "name": "DecryptReplayPasswordFilter",
  "type": "CryptoHeaderFilter",
  "config": {
    "messageType": "REQUEST",
    "operation": "DECRYPT",
    "keySecretId": "decryption.secret.id",
    "secretsProvider": "KeyStoreSecretStore-1",
    "algorithm": "AES/ECB/PKCS5Padding",
    "headers": [ "replaypassword" ]
  }
}
```

More information

org.forgerock.openig.filter.CryptoHeaderFilter

CsrfFilter

Prevent Cross Site Request Forgery (CSRF) attacks when using cookie-based authentication, as follows:

- When a session is created or updated for a client, generate a CSRF token as a hash of the session cookie.
- Send the token in a response header to the client, and require the client to provide that header in subsequent requests.
- In subsequent requests, compare the provided token to the generated token.
- If the token is not provided or can't be validated, reject the request and return a valid CSRF token transparently in the response header.

  Rogue websites that attempt CSRF attacks operate in a different website domain to the targeted website. Because of same-origin policy, rogue websites can't access a response from the targeted website, and cannot, therefore, access the CSRF token.

Usage

```
{
  "name": string,
  "type": "CsrfFilter",
  "config": {
    "cookieName": configuration expression<string>,
    "headerName": configuration expression<string>,
    "excludeSafeMethods": configuration expression<boolean>,
    "failureHandler": Handler reference
  }
}
```

Properties

"cookieName": configuration expression<string>, required

The name of the HTTP session cookie used to store the session ID. For example, use the following cookie names for the following processes:

- SSO with the SingleSignOnFilter: Use the name of the AM HTTP session cookie, (default, `iPlanetDirectoryPro`). For information about the AM session cookie, see Find the name of your AM session cookie.

- CDSSO with the CrossDomainSingleSignOnFilter: Use the name configured in `authCookie.name`.
- OpenID Connect with the AuthorizationCodeOAuth2ClientFilter: Use the name of the IG HTTP session cookie (default, `IG_SESSIONID`). For information about the IG session cookie, see admin.json.
- SAML: Use the name of the IG HTTP session cookie (default, `IG_SESSIONID`). For information about the IG session cookie, see admin.json.

"`headerName`": configuration expression<string>, optional

The name of the header that carries the CSRF token. The same header is used to create and verify the token.

Default: `X-CSRF-Token`

"`excludeSafeMethods`": configuration expression<boolean>, optional

Whether to exclude GET, HEAD, and OPTION methods from CSRF testing. In most cases, these methods are assumed as safe from CSRF.

Default: `true`

"`failureHandler`": Handler reference, optional

Handler to treat the request if the CSRF the token is not provided or can't be validated. Provide an inline handler declaration, or the name of a handler object defined in the heap.

Although IG returns the CSRF token transparently in the response header, this handler cannot access the CSRF token.

Default: Handler that generates `HTTP 403 Forbidden`.

Example

For an example of how to harden protection against CSRF attacks, see Protect against CSRF attacks.

```
{
  "name": "CsrfFilter-1",
  "type": "CsrfFilter",
  "config": {
    "cookieName": "openig-jwt-session",
    "headerName": "X-CSRF-Token",
    "excludeSafeMethods": true
  }
}
```

More information

org.forgerock.openig.filter.CsrfFilterHeaplet

DateHeaderFilter

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

Usage

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

Properties

There are no configuration properties for this filter.

Example

The following example includes a DateHeaderFilter in a chain:

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

More information

For information about Date format, see RFC 7231 - Date.

This filter is also available to support Financial-Grade API, for information, see Financial-grade API Security Profile 1.0 - Part 1: Baseline

org.forgerock.openig.filter.DateHeaderFilter

EncryptedPrivateKeyJwtClientAuthenticationFilter

Supports client authentication with the `private_key_jwt` client-assertion, using a signed and encrypted JWT.

Clients send a signed and encrypted JWT to the authorization server. IG builds, signs and encrypts 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
```

Use this filter with an endpoint `Handler` that requires authentication with the `private_key_jwt` client-assertion, using an encrypted JWT. For example, the `endpointHandler` handler in the OAuth2TokenExchangeFilter.

Usage

```
{
  "name": string,
  "type": "EncryptedPrivateKeyJwtClientAuthenticationFilter",
  "config": {
    "encryptionAlgorithm": configuration expression<enumeration>,
    "encryptionMethod": configuration expression<string>,
    "encryptionSecretId": configuration expression<secret-id>,
    "clientId": configuration expression<string>,
    "tokenEndpoint": configuration expression<url>,
    "secretsProvider": SecretsProvider reference,
```

```
    "signingSecretId": configuration expression<string>,
    "signingAlgorithm": configuration expression<string>,
    "jwtExpirationTimeout": configuration expression<duration>,
    "claims": map
  }
}
```

Configuration

"encryptionAlgorithm": configuration expression<string>, required
The algorithm name used for encryption and decryption. Use algorithm names from Java Security Standard Algorithm Names.

"encryptionMethod": configuration expression<string>, optional
The algorithm method to use for encryption. Use algorithms from RFC 7518, section-5.1.

"encryptionSecretId": configuration expression<secret-id>, required
The secret-id of the keys used to encrypt the JWT.

"clientId": configuration expression<string>, required
The `client_id` obtained when registering with the authorization server.

"tokenEndpoint": configuration expression<url>, required
The URL to the authorization server's OAuth 2.0 token endpoint.

"secretsProvider": SecretsProvider reference, required
The SecretsProvider to resolve queried secrets, such as passwords and cryptographic keys. For allowed formats, see SecretsProvider.

"signingSecretId": configuration expression<string>, required
Reference to the keys used to sign the JWT.

"signingAlgorithm": configuration expression<string>, optional
The JSON Web Algorithm (JWA) used to sign the JWT, such as:

- `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`

"jwtExpirationTimeout": configuration expression<duration>, optional
The duration for which the JWT is valid.

Default: 1 minute

"`claims`": map, optional

The claims used in the authentication.

The map can be structured as follows:

- As a configuration expression of a map, where the evaluation is an object of type `Map<String, Object>`.
- As a map whose key is a string, and whose values are configuration expressions.

Default: Empty

EntityExtractFilter

Extracts regular expression patterns from a message entity, and stores their values in a `target` object. Use this object in password replay, to find a login path or extract a nonce.

If the message `type` is `REQUEST`, the pattern is extracted before the request is handled. If the message `type` is `RESPONSE`, the pattern is extracted out of the response body.

Each pattern can have an associated template, which is applied to its match result.

For information, see Patterns.

Usage

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

Properties

"`messageType`": configuration expression<enumeration>, required
The message type to extract patterns from.

Must be `REQUEST` or `RESPONSE`.

"`charset`": configuration expression<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`": configuration expression<string>, required
Name of the element in the 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.

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 `wpLogintokenvalue="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"
            }
        ]
    }
}
```

More information

org.forgerock.openig.filter.EntityExtractFilter

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.

Usage

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

Properties

There are no configuration properties for this filter.

Example

The following example, based on Validate Certificate-Bound Access Tokens, adds a FapiInteractionIdFilter to the end of the chain:

```
{
  "name": "mtls",
  "condition": "${find(request.uri.path, '/mtls')}",
  "handler": {
    "type": "Chain",
    "config": {
      "filters": [ {
          "name": "OAuth2ResourceServerFilter-1",
           ...
        },
        {
          "type": "FapiInteractionIdFilter"
        }
      ],
      "handler": {
          "name": "StaticResponseHandler-1",
          ...
      }
    }
  }
}
```

More information

org.forgerock.openig.filter.finance.FapiInteractionIdFilter

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

FragmentFilter

Tracks the fragment part of a URI when a request triggers a login redirect.

- Before authentication, the filter captures the URI fragment information and stores it in a cookie.
- After authentication, when the request is issued again to the original URI, the filter redirects the browser to the original URI, including any URI fragment.

Use this filter with SingleSignOnFilter, CrossDomainSingleSignOnFilter, AuthorizationCodeOAuth2ClientFilter, and PolicyEnforcementFilter. This filter is not required for SAML because the final redirect is done with a DispatchHandler and a StaticResponseFilter.

Usage

```
{
  "name": string,
  "type": "FragmentFilter",
  "config": {
    "fragmentCaptureEndpoint": configuration expression<string>,
    "noJavaScriptMessage": configuration expression<string>,
    "cookie": object
  }
}
```

"fragmentCaptureEndpoint": configuration expression<string>, required
The IG endpoint used to capture the fragment form data.

Configure the endpoint to match the condition of the route in which the filter is used.

"noJavaScriptMessage": configuration expression<string>, optional
A message to display on the fragment form when JavaScript is not enabled.

Default: No message

"cookie": object, optional
The configuration of the cookie used to store the fragment information.

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

"`name`": configuration expression<string>, optional
Cookie name.

Default: `ig-fragment-cookie`

"`domain`": configuration expression<string>, optional
Domain to which the cookie applies.

Default: The fully qualified hostname of the IG host.

"`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 to apply to the cookie.

Default: `/`

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

Options to manage the circumstances in which a cookie is sent to the server. Use one of the following values to reduce the risk of CSRF attacks:

- `STRICT` : Send the cookie only if the request was initiated from the cookie domain. Not case-sensitive.

  Use this value to reduce the risk of cross-site request forgery (CSRF) attacks.
- `LAX` : Send the cookie only with GET requests in a first-party context, where the URL in the address bar matches the cookie domain. Not case-sensitive.

  Use this value to reduce the risk of cross-site request forgery (CSRF) attacks.
- `NONE` : Send the cookie whenever a request is made to the cookie domain. Not case-sensitive.

  With this setting, consider setting `secure` to `true` to prevent browsers from rejecting the cookie. For more information, see SameSite cookies.

Default: `LAX`

"`secure`": configuration expression<boolean>, optional
Flag to limit the scope of the cookie to secure channels.

Default: `false`

"`maxAge`": configuration expression<duration>, optional
The maximum duration for which the FragmentFilter cookie can be valid.

When this `maxAge` is greater than the browser's maximum internal value, the browser value takes precedence.

Default: 1 hour

Example

For an example of how the FragmentFilter is used in an SSO flow, see Persist URI fragments in login redirects.

More information

AuthRedirectContext

org.forgerock.openig.filter.FragmentFilter

URI Fragment

RFC 3986: Fragment

FileAttributesFilter

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.

Usage

```
{
     "name": string,
```

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

For an example see Log in with credentials from a file.

Properties

"file": configuration expression<string>, required
The file containing the record to be read.

"charset": configuration expression<string>, optional
The character set in which the file is encoded.

Default: `"UTF-8"`.

"separator": configuration expression<enumeration>, 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": configuration expression<boolean>,optional
A flag to 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 configuration expression<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": configuration expression<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.

More information

org.forgerock.openig.filter.FileAttributesFilter

ForwardedRequestFilter

Rebase the request URI to a computed scheme, host name, and port.

Use this filter to configure redirects when a request is forwarded by an upstream application such as a TLS offloader.

Usage

```
{
  "name": string,
  "type": "ForwardedRequestFilter",
  "config": {
    "scheme": runtime expression<string>,
    "host": runtime expression<string>,
    "port": runtime expression<number>
  }
}
```

Properties

At least one of `scheme`, `host`, or `port` must be configured.

"scheme": runtime expression<string>, optional
The scheme to which the request is rebased, for example, `https`.

Default: Not rebased to a different scheme

"host": runtime expression<string>, optional
The host to which the request is rebased.

Default: Not rebased to a different host

"port": runtime expression<number>, optional
The port to which the request is rebased.

Default: Not rebased to a different port

Example

In the following configuration, IG runs behind an AWS load balancer, to perform a login page redirect to an authentication party, using the original URI requested by the client.

IG can access the URI used by the load balancer to reach IG, but can't access the original request URI.

The load balancer breaks the original request URI into the following headers, and adds them to the incoming request:

- `X-Forwarded-Proto`: Scheme
- `X-Forwarded-Port`: Port
- `Host`: Original host name, and possibly the port.

```
{
  "type": "ForwardedRequestFilter",
  "config": {
    "scheme": "${request.headers['X-Forwarded-Proto'][0]}",
    "host": "${split(request.headers['Host'][0], ':')[0]}",
    "port": "${integer(request.headers['X-Forwarded-Port'][0])}"
  }
}
```

More information

org.forgerock.openig.filter.ForwardedRequestFilter

HeaderFilter

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.

Usage

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

Properties

`"messageType": configuration expression<enumeration>, required

The type of message for which to filter headers. Must be either `"REQUEST"` or `"RESPONSE"`.

"remove": array of configuration expression<strings>, optional

The names of header fields to remove from the message.

"add": object, optional

One or more headers to add to a request, with the format *`name`*`: [ `*`value`*`, … ]`, where:

- *name* is a string for a header name.
- *value* is a runtime expression that resolves to one or more header values.

CAUTION

For IG in web container mode, sending or receiving headers or trailers whose names or values contain non-ASCII characters can cause unspecified behavior at the HTTP server level, and character corruption at the ClientHandler/ReverseProxyHandler level.

Examples

Replace 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" ]
    }
  }
}
```

Add 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" ]
    }
  }
}
```

Add 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" ]
    }
  }
}
```

Add 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}"]
    }
  }
}
```

Add headers and logging results

The following example adds a message to the request and response as it passes through the Chain, and the `capture` on the ReverseProxyHandler logs the result. With IG and the sample application set up as described in the Getting started, access this route on http://ig.example.com:8080/home/chain.

```
{
  "condition": "${find(request.uri.path, '^/home/chain')}",
  "handler": {
    "type": "Chain",
    "comment": "Base configuration defines the capture decorator",
    "config": {
      "filters": [
        {
          "type": "HeaderFilter",
          "comment": "Add a header to all requests",
          "config": {
            "messageType": "REQUEST",
            "add": {
              "MyHeaderFilter_request": [
                "Added by HeaderFilter to request"
```

```
              ]
            }
          }
        },
        {
          "type": "HeaderFilter",
          "comment": "Add a header to all responses",
          "config": {
            "messageType": "RESPONSE",
            "add": {
              "MyHeaderFilter_response": [
                "Added by HeaderFilter to response"
              ]
            }
          }
        }
      ],
      "handler": {
        "type": "ReverseProxyHandler",
        "comment": "Log request, pass it to the sample app, log
response",
        "capture": "all",
        "baseURI": "http://app.example.com:8081"
      }
    }
  }
}
```

The chain receives the request and context and processes it as follows:

- The first `HeaderFilter` adds a header to the incoming request.
- The second `HeaderFilter` manages responses not requests, so it simply passes the request and context to the handler.
- The `ReverseProxyHandler` captures (logs) the request.
- The `ReverseProxyHandler` forwards the transformed request to the protected application.
- The protected application passes a response to the `ReverseProxyHandler` .
- The `ReverseProxyHandler` captures (logs) the response.
- The second `HeaderFilter` adds a header added to the response.
- The first `HeaderFilter` is configured to manage requests, not responses, so it simply passes the response back to IG.

The following example lists some of the HTTP requests and responses captured as they flow through the chain. You can search the log files for `MyHeaderFilter_request` and `MyHeaderFilter_response`.

```
# Original request from user-agent
GET http://ig.example.com:8080/home/chain HTTP/1.1
Accept: /
Host: ig.example.com:8080

# Add a header to the request (inside IG) and direct it to the
protected application
GET http://app.example.com:8081/home/chain HTTP/1.1
Accept: /
Host: ig.example.com:8080
MyHeaderFilter_request: Added by HeaderFilter to request

# Return the response to the user-agent
HTTP/1.1 200 OK
Content-Length: 1809
Content-Type: text/html; charset=ISO-8859-1

# Add a header to the response (inside IG)
HTTP/1.1 200 OK
Content-Length: 1809
MyHeaderFilter_response: Added by HeaderFilter to response
```

More information

org.forgerock.openig.filter.HeaderFilter

HttpBasicAuthenticationClientFilter

Authenticates clients according to the HTTP basic access authentication scheme.

HTTP basic access authentication is a simple challenge and response mechanism, where a server requests credentials from a client, and the client passes them to the server in an `Authorization` header. The credentials are base-64 encoded. To protect them, use SSL encryption for the connections between the server and client. For more information, see RFC 2617.

TIP

Tip

Compare the purpose of this filter with that of the following filters:

- ClientCredentialsOAuth2ClientFilter, which authenticates clients by their OAuth 2.0 credentials to obtain an access token from an authorization server.
- ClientSecretBasicAuthenticationFilter, which fulfils the same role of transforming OAuth 2.0 credentials to an `Authorization` header, but is more strict for OAuth 2.0 requirements.

Use HttpBasicAuthenticationClientFilter in a service-to-service context, where services need to access resources protected by HTTP basic access authentication.

Usage

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

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.

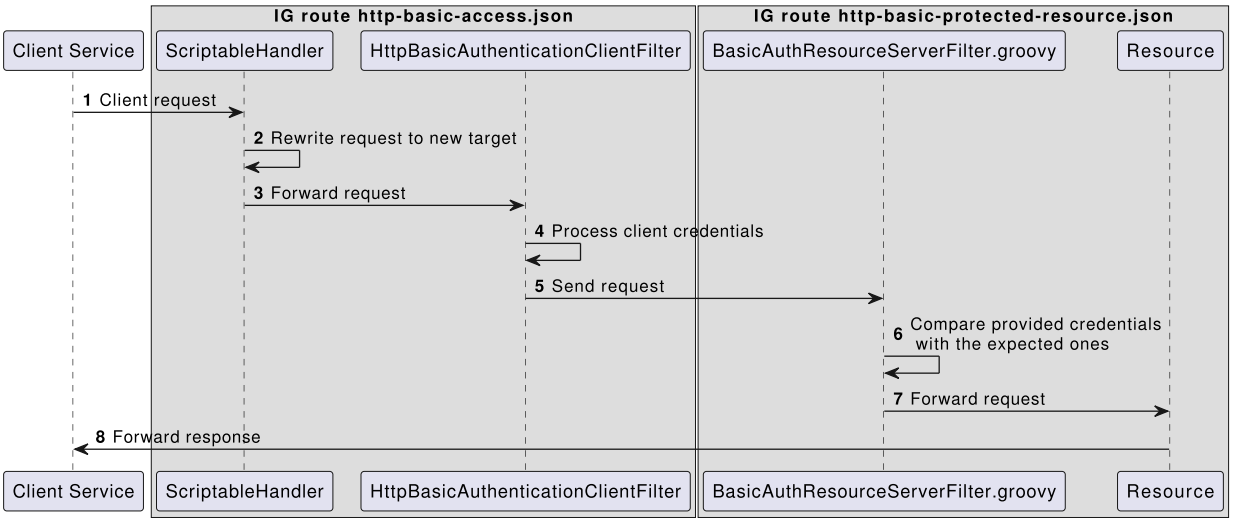
"secretsProvider": SecretsProvider reference, required
The SecretsProvider to use to obtain the `passwordSecretId`. Provide either the name of a SecretsProvider object defined in the heap, or specify a SecretsProvider object inline.

"urlEncodeCredentials": configuration expression<boolean>, optional
Set to `true` to URL-encoded credentials before base64-encoding them.

Default: `false`

Example

The following example shows the flow of information when a client service accesses a resource protected by HTTP basic access authentication:



Set Up the Example

1. Add the following script to the IG configuration as `$HOME/.openig/scripts/groovy/BasicAuthResourceServerFilter.groovy` (on Windows, `%appdata%\OpenIG\scripts\groovy\BasicAuthResourceServerFilter.groovy`):

```
/*
 * This script is a simple implementation of HTTP basic access authentication on
 * server side.
 * It expects the following arguments:
 *  - realm: the realm to display when the user-agent prompts for
 *    username and password if none were provided.
 *  - username: the expected username
 *  - password: the expected password
 */

import static org.forgerock.util.promise.Promises.newResultPromise;

import java.nio.charset.Charset;
import org.forgerock.util.encode.Base64;

String authorizationHeader = request.getHeaders().getFirst("Authorization");
if (authorizationHeader == null) {
```

```
    // No credentials provided, reply that they are
needed.
    Response response = new
Response(Status.UNAUTHORIZED);
    response.getHeaders().put("WWW-Authenticate",
"Basic realm=\"" + realm + "\"");
    return newResultPromise(response);
}

String expectedAuthorization = "Basic " +
Base64.encode((username + ":" +
password).getBytes(Charset.defaultCharset()))
if (!expectedAuthorization.equals(authorizationHeader))
{
    return newResultPromise(new
Response(Status.FORBIDDEN));
}
// Credentials are as expected, let's continue
return next.handle(context, request);
```

The script is a simple implementation of the HTTP basic access authentication scheme.

For information about scripting filters and handlers, see Extend IG.

2. Add the following route to IG:

Linux | Windows

```
$HOME/.openig/config/routes/http-basic-access.json
```

```
{
  "name": "http-basic-access",
  "baseURI": "http://ig.example.com:8080",
  "condition" : "${find(request.uri.path, '^/http-basic-
access')}",
  "heap": [
    {
      "name": "httpBasicAuthEnabledClientHandler",
      "type": "Chain",
      "capture": "all",
      "config": {
        "filters": [
```

```
            {
              "type": "HttpBasicAuthenticationClientFilter",
              "config": {
                "username": "myclient",
                "passwordSecretId": "password.secret.id",
                "secretsProvider": {
                  "type": "Base64EncodedSecretStore",
                  "config": {
                    "secrets": {
                      "password.secret.id": "cGFzc3dvcmQ="
                    }
                  }
                }
              }
            }
          ],
          "handler": "ForgeRockClientHandler"
        }
      }
    ],
    "handler": {
      "type": "ScriptableHandler",
      "config": {
        "type": "application/x-groovy",
        "clientHandler":
"httpBasicAuthEnabledClientHandler",
        "source": [
          "request.uri.path = '/http-basic-protected-
resource'",
          "return http.send(context, request);"
        ]
      }
    }
}
```

Note the following features of the route:

- The route matches requests to `/http-basic-access`.
- The ScriptableHandler rewrites the request to target it to `/http-basic-protected-resource`, and then calls the HTTP client, that has been redefined to use the httpBasicAuthEnabledClientHandler.
- The httpBasicAuthEnabledClientHandler calls the HttpBasicAuthenticationClientFilter to authenticate the client, using the client's credentials.

3. Add the following route to IG:

Linux | Windows

$HOME/.openig/config/routes/http-basic-protected-resource.json

```
{
  "name": "http-basic-protected-resource",
  "condition": "${find(request.uri.path, '^/http-basic-protected-resource')}",
  "handler": {
    "type": "Chain",
    "config": {
      "filters": [
        {
          "name": "HttpBasicAuthResourceServerFilter",
          "type": "ScriptableFilter",
          "config": {
            "type": "application/x-groovy",
            "file": "BasicAuthResourceServerFilter.groovy",
            "args": {
              "realm": "IG Protected Area",
              "username": "myclient",
              "password": "password"
            }
          }
        }
      ],
      "handler": {
        "type": "StaticResponseHandler",
        "config": {
          "status": 200,
          "headers": {
            "Content-Type": [ "text/html; charset=UTF-8" ]
          },
          "entity": "<html><body><h2>Access Granted</h2></body></html>"
        }
      }
    }
  }
}
```

Notice the following features of the route:

- The route matches requests to `/http-basic-protected-resource`.
- The ScriptableFilter provides a script to implement a simple HTTP basic access authentication scheme, that compares the provided credentials with the expected credentials.
- When the client is authenticated, the StaticResponseHandler returns a message that access is granted.

4. Access the route on http://ig.example.com:8080/http-basic-access.

   Because the expected credentials were provided in the request, a message shows that access is granted.

More information

org.forgerock.openig.filter.HttpBasicAuthenticationClientFilter

HttpBasicAuthFilter

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

This filter performs HTTP basic access authentication, 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.

Usage

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

```
    }
}
```

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": configuration expression<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`

Example

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

More information

org.forgerock.openig.filter.HttpBasicAuthFilter

IdTokenValidationFilter

Validates an ID token by checking the standard claims, `aud`, `exp`, and `iat`. If specified in the configuration, this filter also checks the ID token issuer and signature.

This filter passes data into the context as follows:

- If the JWT is validated, the request continues down the chain. The data is provided in the JwtValidationContext.
- If the JWT is not validated, data is provided in the JwtValidationErrorContext.

  If a failure handler is configured, the request passes to the failure handler. Otherwise, an HTTP 403 Forbidden is returned.

The `iat` claim is required, and the `iat` minus the `skewAllowance` must be before the current time on the IG clock. For information, see OpenID Connect Core 1.0 incorporating errata set 1.

Usage

```
{
   "name": string,
   "type": "IdTokenValidationFilter",
   "config": {
     "idToken": runtime expression<string>,
     "audience": configuration expression<string>,
     "issuer": configuration expression<string>,
     "skewAllowance": configuration expression<duration>,
     "verificationSecretId": configuration expression<secret-id>,
     "secretsProvider": SecretsProvider reference,
     "customizer": JwtValidatorCustomizer reference,
     "failureHandler": Handler reference
   }
}
```

Properties

"idToken": runtime expression<string>, required

The ID token as an expression representing the JWT or signed JWT in the request. Cannot be null.

"`audience`": configuration expression<string>, required

The `aud` claim to check on the JWT. Cannot be null.

"`issuer`": configuration expression<string>, optional

The `iss` claim to check on the JWT. Can be null.

"`skewAllowance`": configuration expression<duration>, optional

The duration 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 IG clock.
- A JWT with an `exp` 13:00 is expired after 13:02 on the IG clock.

Default: `zero`

"`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 configured, the token must be signed. If not configured, IG does not verify the signature.

For information about how signatures are validated, see Validating the signature of signed tokens. For information about how each type of secret store resolves named secrets, see Secrets.

"`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.

"`customizer`": JwtValidatorCustomizer reference, optional

A set of validation constraints for JWT claims and sub-claims. If a claim is not validated against the constraint, the JWT is not validated.

The customizer does not override existing constraints, such as `aud`, `iss`, `exp`, and `iat`, which are predefined in the IdTokenValidationFilter, Defining a new constraint on an already constrained claim has an impact only if the new constraint is more restrictive.

JwtValidatorCustomizer provides a ScriptableJwtValidatorCustomizer to enrich a `builder` object by using its methods. Get more information about the following items:

- The `builder` object, at Available Objects.

- Transformer methods, to enrich the builder object, at org.forgerock.openig.util.JsonValues.
- Constraints, at org.forgerock.openig.tools.jwt.validation.Constraints.
- Other properties for ScriptableJwtValidatorCustomizer, at Scripts.

The following examples provide checks:

Check that the value of the claim `greaterThan5` is greater than 5

```
"customizer": {
  "type": "ScriptableJwtValidatorCustomizer",
  "config": {
    "type": "application/x-groovy",
    "source": [
      "builder.claim('/greaterThan5', JsonValue::asInteger,
isGreaterThan(5))"
    ]
  }
}
```

Check that the value of the claim `sub` is `george`

```
"customizer": {
  "type": "ScriptableJwtValidatorCustomizer",
  "config": {
    "type": "application/x-groovy",
    "source": [
      "builder.claim('subname', JsonValue::asString,
isEqualTo('george'))"
    ]
  }
}
```

Check that the value of the custom sub-claim is `ForgeRock`

```
"customizer": {
  "type": "ScriptableJwtValidatorCustomizer",
  "config": {
    "type": "application/x-groovy",
    "source": [
      "builder.claim('customclaim/subclaim',
JsonValue::asString, isEqualTo('ForgeRock'));"
    ]
```

```
  }
}
```

Check the value of multiple claims

```
"customizer": {
  "type": "ScriptableJwtValidatorCustomizer",
  "config": {
    "type": "application/x-groovy",
    "source": [
      "builder.claim('aud', listOf(JsonValue::asString),
contains('My App'))",
      "       .claim('iat', instant(), isInThePast())",
      "       .claim('exp', instant(), isInTheFuture());",
      "builder.claim('iss', JsonValue::asString,
isEqualTo('ForgeRock AM'));"
    ]
  }
}
```

Check that the value of `val1` is greater than `val2`

```
"customizer": {
  "type": "ScriptableJwtValidatorCustomizer",
  "config": {
    "type": "application/x-groovy",
    "source": [ "builder.claim('/val1',
JsonValue::asInteger,
isGreaterThan(claim('/val2').asInteger()))" ]
  }
}
```

Check that the value of `val1` is greater than `val2`, when both are YYYY-MM-DD dates

```
"customizer": {
  "type": "ScriptableJwtValidatorCustomizer",
  "config": {
    "type": "application/x-groovy",
    "source": [
      "Function<JsonValue, java.time.LocalDate, Exception>
asDate() {",
      "  return (jsonValue) ->
java.time.LocalDate.parse(jsonValue.asString());",
      "}",
```

```
      "builder.claim('claim1', asDate(),
isGreaterThan(claim('claim2').as(asDate())));"
    ]
  }
}
```

Check that the claim issuer matches the regex pattern

```
"customizer": {
  "type": "ScriptableJwtValidatorCustomizer",
  "config": {
    "type": "application/x-groovy",
    "source": [ "builder.claim('iss', JsonValue::asString,
find(~/.*am\.example\.(com|org)/))" ]
  }
}
```

Default: Claims are not validated

"`failureHandler`": Handler reference, optional

Handler to treat the request on failure.

Provide an inline handler configuration object, or the name of a handler object declared in the heap. See also Handlers.

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

Example

Validate an id_token

1. Set up AM:
   a. Set up AM as described in Validate access tokens through the introspection endpoint.
   b. Select **Applications** > **OAuth 2.0** > **Clients**, and add the additional scope `openid` to `client-application`.
2. Set up IG:
   a. Add the following route to IG:

Linux | Windows

```
$HOME/.openig/config/routes/idtokenvalidation.json
```

```
{
  "name": "idtokenvalidation",
  "condition": "${find(request.uri.path,
'^/idtokenvalidation')}",
  "capture": "all",
  "handler": {
    "type": "Chain",
    "config": {
      "filters": [{
        "type": "IdTokenValidationFilter",
        "config": {
          "idToken": "<id_token_value>",
          "audience": "client-application",
          "issuer":
"http://am.example.com:8088/openam/oauth2",
          "failureHandler": {
            "type": "ScriptableHandler",
            "config": {
              "type": "application/x-groovy",
              "source": [
                "def response = new
Response(Status.FORBIDDEN)",
                "response.headers['Content-Type'] =
'text/html; charset=utf-8'",
                "def errors =
contexts.jwtValidationError.violations.collect{it.descr
iption}",
                "def display = \"<html>Can't validate
id_token:<br> ${contexts.jwtValidationError.jwt} \"",
                "display <<=\"<br><br>For the following
errors:<br> ${errors.join(\"<br>\")}</html>\"",
                "response.entity=display as String",
                "return response"
              ]
            }
          },
          "verificationSecretId": "verify",
          "secretsProvider": {
            "type": "JwkSetSecretStore",
            "config": {
              "jwkUrl":
"http://am.example.com:8088/openam/oauth2/connect/jwk_u
ri"
            }
          }
```

```
        }
      }],
      "handler": {
        "type": "StaticResponseHandler",
        "config": {
          "status": 200,
          "headers": {
            "Content-Type": [ "text/html; charset=UTF-
8" ]
          },
          "entity": "<html><body>Validated id_token:
<br> ${contexts.jwtValidation.value}</body></html>"
        }
      }
    }
  }
}
```

Notice the following features of the route:

- The route matches requests to `/idtokenvalidation`.
- A SecretsProvider declares a JwkSetSecretStore to validate secrets for signed JWTs, which specifies the URL to a JWK set on AM that contains the signing keys.
- The property `verificationSecretId` is configured with a value. If this property is not configured, the filter does not verify the signature of tokens.
- The JwkSetSecretStore specifies the URL to a JWK set on AM, that contains signing keys identified by a `kid`. The signature of the token is verified as follows:
  - If the value of a `kid` in the JWK set matches a `kid` in the the signed JWT, the JwkSetSecretStore verifies the signature.
  - If the JWT doesn't have a `kid`, or if the JWK set doesn't contain a key with the same value, the JwkSetSecretStore looks for valid secrets with the same purpose as the value of `verificationSecretId`.
- If the filter validates the token, the StaticResponseHandler displays the token value from the context `${contexts.jwtValidation.value}`. Otherwise, the ScriptableHandler displays the token value and a list of violations from the context `${contexts.jwtValidationError.violations}`

3. Test the setup:

a. In a terminal window, use a `curl` command similar to the following to retrieve an id_token:

```
$ curl -s \
--user "client-application:password" \
--data
"grant_type=password&username=demo&password=Ch4ng31t&sc
ope=openid" \
http://am.example.com:8088/openam/oauth2/access_token

{
 "access_token":"...",
 "scope":"openid",
 "id_token":"...",
 "token_type":"Bearer",
 "expires_in":3599
}
```

- In the route, replace `<id_token_value>` with the value of the `id_token` returned in the previous step.

b. Access the route on http://ig.example.com:8080/idtokenvalidation.

The validated token is displayed.

- In the route, invalidate the token by changing the value of the audience or issuer, and then access the route again.

  The value of the token, and the reasons that the token is invalid, are displayed.

More information

org.forgerock.openig.filter.oauth2.client.IdTokenValidationFilterHeaplet

org.forgerock.openig.filter.jwt.JwtValidationContext

org.forgerock.openig.filter.jwt.JwtValidationErrorContext

OpenID Connect Core 1.0 incorporating errata set 1

JwtBuilderFilter

Collects data at runtime, packs it in a JSON Web Token (JWT), and places the resulting JWT into the JwtBuilderContext.

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

- 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.

For a flexible way to pass identity or other runtime information to the protected application, use this filter with a HeaderFilter.

To enable downstream filters and handlers to verify signed and/or encrypted JWTs built by this filter, use this filter with a JwkSetHandler.

Usage

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

Properties

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

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

If this property is a map, the structure must have the format `Map<String, Object>`, and map values are evaluated as runtime expressions. 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, `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 Default secrets object.

"`signature`": object, optional

IMPORTANT

The use of unsigned or unencrypted JWTs is deprecated and not considered secure. For more information, refer to Deprecation.

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

`JwtBuilderFilter.encryption` takes precedence over this property.

```
{
  "signature": {
    "secretId": configuration expression<secret-id>,
    "algorithm": configuration expression<string>,
    "encryption": object,
    "keystore":  KeyStore reference, //deprecated
    "alias": configuration expression<string>, //deprecated
    "password": configuration expression<string> //deprecated
  }
}
```

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

The secret ID of the key to sign the JWT.

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

The algorithm to sign the JWT.

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

- Algorithms described in RFC 7518: 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 IG 6.1, `Ed25519` described in CFRG Elliptic Curve Diffie-Hellman (ECDH) and Signatures in JSON Object Signing and Encryption (JOSE).

  Default: RS256

"encryption": object, optional

Configuration to encrypt the JWT signature.

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

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

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

"algorithm": configuration expression<string>, required

The algorithm used to encrypt the JWT signature.

For information about available algorithms, see RFC 7518: "alg" (Algorithm) Header Parameter Values for JWE.

"method": configuration expression<string>, required

The method used to encrypt the JWT signature.

For information about available methods, see RFC 7518: "enc" (Encryption Algorithm) Header Parameter Values for JWE.

"keystore": KeyStore reference, optional

IMPORTANT

This property is deprecated; use the `signature` subproperty `secretId` instead. For more information, refer to Deprecation.

The Java KeyStore containing the key used to sign the JWT.

The name of a KeyStore object defined in the heap, or an inline KeyStore configuration object.

"alias": configuration expression<string>, required if `signature` is used

IMPORTANT

IMPORTANT

This property is deprecated; use the `signature` subproperty `secretId` instead. For more information, refer to Deprecation.

Alias for the key.

Note Because KeyStore converts all characters in its key aliases to lower case, use only lowercase in alias definitions of a KeyStore.

"password": expression<string>, required if `signature` is used

IMPORTANT

This property is deprecated; use the `signature` subproperty `secretId` instead. For more information, refer to Deprecation.

Password for the key.

"encryption": object, optional

IMPORTANT

The use of unsigned or unencrypted JWTs is deprecated and not considered secure. For more information, refer to Deprecation.

Configuration to encrypt the JWT. This property take precedence over `JwtBuilderFilter.signature`.

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

"secretId": secret-id, optional

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

"algorithm": configuration expression<string>, required

The algorithm used to encrypt the JWT.

For information about available algorithms, see RFC 7518: "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 RFC 7518: "enc" (Encryption Algorithm) Header Parameter Values for JWE.

Examples

For examples, see Pass runtime data downstream in a JWT

More information

org.forgerock.openig.filter.JwtBuilderFilter

org.forgerock.openig.filter.JwtBuilderContext

JwtValidationFilter

Validates an unsigned, signed, encrypted, or signed and encrypted JWT. The order of signing and encryption is not important; a JWT can be signed and then encrypted, or encrypted and then signed.

If the JWT is validated, the request continues down the chain. The data is provided in the JwtValidationContext.

If the JWT is not validated, data is provided in the JwtValidationErrorContext. If a failure handler is configured, the request passes to the failure handler. Otherwise, an HTTP 403 Forbidden is returned.

Usage

```
{
  "name": string,
  "type": "JwtValidationFilter",
  "config": {
    "jwt": runtime expression<string>,
    "verificationSecretId": configuration expression<secret-id>,
    "decryptionSecretId": configuration expression<secret-id>,
    "secretsProvider": SecretsProvider reference,
    "skewAllowance": configuration expression<duration>,
    "customizer": JwtValidatorCustomizer reference,
    "failureHandler": Handler reference
  }
}
```

Properties

"`jwt`": runtime expression<string>, required

The value of the JWT in the request. Cannot be null.

"`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 configured, the token must be signed. If not configured, IG does not verify the signature.

For information about how signatures are validated, see Validating the signature of signed tokens. For information about how each type of secret store resolves named secrets, see Secrets.

"`decryptionSecretId`": configuration expression<secret-id>, required if AM secures access tokens with encryption

The secret ID for the secret to verify the encryption of tokens.

If configured, the token must be encrypted. If not configured, IG does not verify the encryption.

For information about how each type of secret store resolves named secrets, see Secrets.

"`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.

"`customizer`": JwtValidatorCustomizer reference, optional

A set of validation constraints for JWT claims and sub-claims. If a claim is not validated against the constraint, the JWT is not validated.

The customizer does not override existing constraints, such as `aud`, `iss`, `exp`, and `iat`, which are predefined in the IdTokenValidationFilter, Defining a new constraint on an already constrained claim has an impact only if the new constraint is more restrictive.

JwtValidatorCustomizer provides a ScriptableJwtValidatorCustomizer to enrich a `builder` object by using its methods. Get more information about the following items:

- The `builder` object, at Available Objects.
- Transformer methods, to enrich the builder object, at org.forgerock.openig.util.JsonValues.
- Constraints, at org.forgerock.openig.tools.jwt.validation.Constraints.
- Other properties for ScriptableJwtValidatorCustomizer, at Scripts.

The following examples provide checks:

Check that the value of the claim `greaterThan5` is greater than 5

```
"customizer": {
  "type": "ScriptableJwtValidatorCustomizer",
  "config": {
    "type": "application/x-groovy",
    "source": [
      "builder.claim('/greaterThan5', JsonValue::asInteger,
isGreaterThan(5))"
    ]
  }
}
```

Check that the value of the claim `sub` is `george`

```
"customizer": {
  "type": "ScriptableJwtValidatorCustomizer",
  "config": {
    "type": "application/x-groovy",
    "source": [
      "builder.claim('subname', JsonValue::asString,
isEqualTo('george'))"
    ]
  }
}
```

Check that the value of the custom sub-claim is `ForgeRock`

```
"customizer": {
  "type": "ScriptableJwtValidatorCustomizer",
  "config": {
    "type": "application/x-groovy",
    "source": [
      "builder.claim('customclaim/subclaim',
JsonValue::asString, isEqualTo('ForgeRock'));"
    ]
  }
}
```

Check the value of multiple claims

```
"customizer": {
  "type": "ScriptableJwtValidatorCustomizer",
  "config": {
    "type": "application/x-groovy",
    "source": [
      "builder.claim('aud', listOf(JsonValue::asString),
contains('My App'))",
      "       .claim('iat', instant(), isInThePast())",
      "       .claim('exp', instant(), isInTheFuture());",
      "builder.claim('iss', JsonValue::asString,
isEqualTo('ForgeRock AM'));"
    ]
  }
}
```

Check that the value of `val1` is greater than `val2`

```
"customizer": {
  "type": "ScriptableJwtValidatorCustomizer",
  "config": {
    "type": "application/x-groovy",
    "source": [ "builder.claim('/val1',
JsonValue::asInteger,
isGreaterThan(claim('/val2').asInteger()))" ]
  }
}
```

Check that the value of `val1` is greater than `val2`, when both are YYYY-MM-DD dates

```
"customizer": {
  "type": "ScriptableJwtValidatorCustomizer",
  "config": {
    "type": "application/x-groovy",
    "source": [
      "Function<JsonValue, java.time.LocalDate, Exception>
asDate() {",
      "  return (jsonValue) ->
java.time.LocalDate.parse(jsonValue.asString());",
      "}",
      "builder.claim('claim1', asDate(),
isGreaterThan(claim('claim2').as(asDate())));"
    ]
```

```
    }
}
```

Check that the claim issuer matches the regex pattern

```
"customizer": {
  "type": "ScriptableJwtValidatorCustomizer",
  "config": {
    "type": "application/x-groovy",
    "source": [ "builder.claim('iss', JsonValue::asString,
find(~/.*am\.example\.(com|org)/))" ]
  }
}
```

Default: Claims are not validated

"`skewAllowance`": configuration expression<duration>, optional

The duration 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 IG clock.
- A JWT with an `exp` 13:00 is expired after 13:02 on the IG clock.

Default: `zero`

"`failureHandler`": Handler reference, optional

Handler to treat the request on failure.

Provide an inline handler configuration object, or the name of a handler object declared in the heap. See also Handlers.

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

Example

For an example of using JwtValidationFilter, see Validate JWTs.

More information

org.forgerock.openig.filter.jwt.JwtValidationFilter

org.forgerock.openig.filter.jwt.JwtValidationContext

org.forgerock.openig.filter.jwt.JwtValidationErrorContext

OpenID Connect Core 1.0 incorporating errata set 1

LocationHeaderFilter

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 IG.

Usage

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

An alternative value for type is RedirectFilter.

Properties

"baseURI": runtime expression<url>,optional

The base URI of the IG 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.

Example

In the following example, IG listens on `https://ig.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://ig.example.com:443`.

```
{
     "name": "LocationRewriter",
     "type": "LocationHeaderFilter",
```

```
      "config": {
          "baseURI": "https://ig.example.com:443/"
       }
}
```

More information

org.forgerock.openig.filter.LocationHeaderFilter

OAuth2ClientFilter

In IG 7.2, this filter was renamed to AuthorizationCodeOAuth2ClientFilter.

For backward compatibility, the name OAuth2ClientFilter can still be used in routes in this release. However, to prevent problems in future releases, update your configuration as soon as possible.

OAuth2ResourceServerFilter

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:

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`

Error	Response from the filter to the user-agent
There is no access token in the request header.	`HTTP 401 Unauthorized WWW-Authenticate: Bearer realm="IG"`
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`

Usage

```
{
  "name": string,
  "type": "OAuth2ResourceServerFilter",
  "config": {
    "accessTokenResolver": AccessTokenResolver reference,
    "cache": object,
    "executor": Executor service reference,
    "requireHttps": configuration expression<boolean>,
    "realm": configuration expression<string>,
    "scopes": [ runtime expression<string>, ... ] or
ScriptableResourceAccess reference
  }
}
```

An alternative value for type is OAuth2RSFilter.

Properties

"accessTokenResolver": AccessTokenResolver reference, required

Resolves an access token against an authorization server. Configure one of the following access token resolvers:

- TokenIntrospectionAccessTokenResolver
- StatelessAccessTokenResolver
- ConfirmationKeyVerifierAccessTokenResolver
- ScriptableAccessTokenResolver

To decorate an AccessTokenResolver, add the decoration at the `accessTokenResolver` level. The following example uses the default timer decorator to record the time that a TokenIntrospectionAccessTokenResolver takes to process a request:

```
{
  "accessTokenResolver": {
    "type": "TokenIntrospectionAccessTokenResolver",
    "config": {
      ...
    },
  "timer": true
  }
}
```

"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, IG can reuse the token information without repeatedly asking the authorization server to verify the access token. When caching is disabled, IG must ask the authorization server to verify the access token for each request.

(From AM 6.5.3.) When an access_token is revoked on AM, the CacheAccessTokenResolver can delete the token from the cache when both of the following conditions are true:

- The `notification` property of AmService is enabled.
- The delegate AccessTokenResolver provides the token metadata required to update the cache.

When a refresh_token is revoked on AM, all associated access tokens are automatically and immediately revoked.

```
"cache": {
  "enabled": configuration expression<boolean>,
  "defaultTimeout": configuration expression<duration>,
  "maxTimeout": configuration expression<duration>,
  "amService": AmService reference,
  "onNotificationDisconnection": configuration
expression<enumeration>
}
```

enabled: configuration expression<boolean>, optional

Enable or disable caching.

Default: `false`

`defaultTimeout`: configuration expression<duration>, 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`, IG uses the token expiry value.

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"
}
```

Default: `1 minute`

`maxTimeout`: configuration expression<duration>, 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`, IG uses the `maxTimeout`.

The duration cannot be `zero` or `unlimited`.

`"amService"`: AmService reference, optional

(From AM 6.5.3.) The AmService to use for the WebSocket notification service. To evict revoked access tokens from the cache, enable the `notifications` property of AmService.

`onNotificationDisconnection`: configuration expression<enumeration>, optional

An `amService` must be configured for this property to have effect.

The strategy to manage the cache when the WebSocket notification service is disconnected, and IG receives no notifications for AM events. If the cache is not cleared it can become outdated, and IG can allow requests on revoked sessions or tokens.

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`

"executor": Executor service reference, 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`": configuration expression<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: OpenIG

"`scopes`": array of runtime expression<strings> or ScriptableResourceAccess <reference>, required

A list of one of more scopes required by the OAuth 2.0 access token.

array of runtime expression<strings>, 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.

ScriptableResourceAccess <reference>, required if "array of runtime expression<strings>" is not used

A script that produces a list of one or more scopes.

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.

```
{
  "name": string,
  "type": "ScriptableResourceAccess",
  "config": {
    "type": configuration expression<string>,
    "file": configuration expression<string>, // Use either
"file"
    "source": [ string, ... ],               // or
"source", but not both.
    "args": object,
    "clientHandler": Handler reference
```

```
    }
}
```

Default: Empty

Examples

For examples using OAuth2ResourceServerFilter, see Act as an OAuth 2.0 resource server.

Defining required scopes by using a script

The following example uses a ScriptableResourceAccess object to define the scopes required in a request:

- If the request path is `/rs-tokeninfo`, only the scopes `mail` is required.
- If the request path is `/rs-tokeninfo/employee`, the scopes `mail` and `employeenumber` are required.

1. Set up and test the example in Validate access tokens through the introspection endpoint.
2. Add the following route to IG:

Linux | Windows

```
$HOME/.openig/config/routes/rs-dynamicscope.json
```

```
{
  "name": "rs-dynamicscope",
  "baseURI": "http://app.example.com:8081",
  "condition": "${find(request.uri.path, '^/rs-
dynamicscope')}",
  "heap": [
    {
      "name": "SystemAndEnvSecretStore-1",
      "type": "SystemAndEnvSecretStore"
    },
    {
      "name": "AmService-1",
      "type": "AmService",
      "config": {
        "agent": {
```

```
          "username": "ig_agent",
          "passwordSecretId": "agent.secret.id"
        },
        "secretsProvider": "SystemAndEnvSecretStore-1",
        "url": "http://am.example.com:8088/openam/",
        "version": "7.2"
      }
    }
  ],
  "handler": {
    "type": "Chain",
    "config": {
      "filters": [
        {
          "name": "OAuth2ResourceServerFilter-1",
          "type": "OAuth2ResourceServerFilter",
          "config": {
            "scopes": {
              "name": "myscript",
              "type": "ScriptableResourceAccess",
              "config": {
                "type": "application/x-groovy",
                "source": [
                  "// Minimal set of required scopes",
                  "def scopes = [ 'mail' ] as Set",
                  "if (request.uri.path =~ /employee$/)
{",
                  "  // Require another scope to access
this resource",
                  "  scopes += 'employeenumber'",
                  "}",
                  "return scopes"
                ]
              }
            },
            "requireHttps": false,
            "realm": "OpenIG",
            "accessTokenResolver": {
              "name": "token-resolver-1",
              "type":
"TokenIntrospectionAccessTokenResolver",
              "config": {
                "amService": "AmService-1",
                "providerHandler": {
                  "type": "Chain",
```

```
                    "config": {
                      "filters": [
                        {
                          "type":
"HttpBasicAuthenticationClientFilter",
                          "config": {
                            "username": "ig_agent",
                            "passwordSecretId":
"agent.secret.id",
                            "secretsProvider":
"SystemAndEnvSecretStore-1"
                          }
                        }
                      ],
                      "handler": "ForgeRockClientHandler"
                    }
                  }
                }
              }
            }
          }
        ],
        "handler": {
          "type": "StaticResponseHandler",
          "config": {
            "status": 200,
            "headers": {
              "Content-Type": [ "text/html; charset=UTF-
8" ]
            },
            "entity": "<html><body><h2>Decoded
access_token: ${contexts.oauth2.accessToken.info}</h2>
</body></html>"
          }
        }
      }
    }
  }
}
```

3. Test the setup with the `mail` scope only:

    a. In a terminal, use a **`curl`** command to retrieve an access token:

```
$ mytoken=$(curl -s \
--user "client-application:password" \
--data
```

```
"grant_type=password&username=demo&password=Ch4ng31
t&scope=mail" \
http://am.example.com:8088/openam/oauth2/access_tok
en | jq -r ".access_token")
```

b. Confirm that the access token is returned for the `/rs-dynamicscope` path:

```
$ curl -v http://ig.example.com:8080/rs-
dynamicscope --header "Authorization: Bearer
${mytoken}"

{
  active = true,
  scope = mail,
  client_id = client-application,
  user_id = demo,
  token_type = Bearer,
  exp = 158...907,
  sub = demo,
  iss = http://am.example.com:8088/openam/oauth2,
...
  ...
}
```

- Confirm that the access token **is not** returned for the `/rs-dynamicscope/employee` path:

```
$ curl -v http://ig.example.com:8080/rs-
dynamicscope/employee --header "Authorization:
Bearer ${mytoken}"
```

4. Test the setup with the scopes `mail` and `employeenumber`:

a. In a terminal window, use a **`curl`** command similar to the following to retrieve an access token:

```
$ mytoken=$(curl -s \
--user "client-application:password" \
--data
"grant_type=password&username=demo&password=Ch4ng31
t&scope=mail%20employeenumber" \
http://am.example.com:8088/openam/oauth2/access_tok
en | jq -r ".access_token")
```

b. Confirm that the access token **is** returned for the `/rs-dynamicscope/employee` path:

```
$ curl -v http://ig.example.com:8080/rs-
dynamicscope/employee --header "Authorization:
Bearer ${mytoken}"
```

More information

org.forgerock.openig.filter.oauth2.OAuth2ResourceServerFilterHeaplet

org.forgerock.http.oauth2.OAuth2Context

org.forgerock.http.oauth2.AccessTokenInfo

OAuth2Context

ConfirmationKeyVerifierAccessTokenResolver

TokenIntrospectionAccessTokenResolver

StatelessAccessTokenResolver

ScriptableAccessTokenResolver

The OAuth 2.0 Authorization Framework

The OAuth 2.0 Authorization Framework: Bearer Token Usage

OAuth2TokenExchangeFilter

Identifies a client's access token or ID token (a *subject token*), and communicates with an authorization service, such as AM, to exchange it for a new token (an *issued token*):

- When the OAuth2TokenExchangeFilter succesfully exchanges a token, it injects the issued token and its scopes into the OAuth2TokenExchangeContext.
- When the OAuth2TokenExchangeFilter fails to exchange a token, it injects information about the failure into the OAuth2FailureContext, which is provided to the `failureHandler` .

The scopes for issued token can be *restricted* or *expanded* by the authorization services:

- Restricted when the token scopes are a subset of those available to the subject token.
- Expanded when they have scopes that are not included in the subject token.

Use this filter in the *impersonation* use case. For more information, see OAuth 2.0 Token Exchange in AM's *OAuth 2.0 guide*.

Usage

```
{
  "name": string,
  "type": "OAuth2TokenExchangeFilter",
    "config": {
      "subjectToken": runtime expression<string>,
      "amService": AmService reference,
      "endpoint": configuration expression<url>,
      "subjectTokenType": configuration expression<string>,
      "requestedTokenType": configuration expression<string>,
      "scopes": [ runtime expression<string>, ... ] or
ScriptableResourceAccess reference,
      "resource": configuration expression<url>,
      "audience": configuration expression<string>,
      "endpointHandler": Handler reference,
      "failureHandler": Handler reference
  }
}
```

Configuration

"subjectToken": runtime expression<string>, required

The location of the subject token in the inbound request.

"amService": AmService reference, required if endpoint is not configured

The AmService to use as the authorization service.

Configure either 'amService' or 'endpoint'. If both are configured, 'amService' takes precedence.

"endpoint": configuration expression<url>, required if amService is not configured

The URI for the authorization service.

Configure either 'amService' or 'endpoint'. If both are configured, 'amService' takes precedence.

"subjectTokenType": configuration expression<string>, optional

The subject token type.

Default: `URN_ACCESS_TOKEN`

"requestedTokenType": configuration expression<string>, optional

The type of token being requested.

Default: `URN_ACCESS_TOKEN`

"scopes": array of runtime expression<strings> or ScriptableResourceAccess <reference>, required

A list of one of more scopes required by the OAuth 2.0 access token.

array of runtime expression<strings>, 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.

ScriptableResourceAccess <reference>, required if "array of runtime expression<strings>" is not used

A script that produces a list of one or more scopes.

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.

```
{
  "name": string,
  "type": "ScriptableResourceAccess",
  "config": {
    "type": configuration expression<string>,
    "file": configuration expression<string>, // Use either
"file"
    "source": [ string, ... ],                // or
"source", but not both.
    "args": object,
    "clientHandler": Handler reference
  }
}
```

Default: Empty

"resource": configuration expression<url>, optional

The target service URI where the issued token is intended to be used.

"audience": configuration expression<url>, optional

The target service name where the token is intended to be used.

"endpointHandler": Handler reference, optional

The handler to exchange tokens on the authorization endpoint.

Configure this property as a Chain, using one of the following filters for client authentication:

- ClientSecretBasicAuthenticationFilter
- ClientSecretPostAuthenticationFilter
- EncryptedPrivateKeyJwtClientAuthenticationFilter
- PrivateKeyJwtClientAuthenticationFilter

```
{
  "name": "EndpointHandler",
  "type": "Chain",
  "config": {
    "handler": "ForgeRockClientHandler",
    "filters": [
      {
        "type": "ClientSecretBasicAuthenticationFilter",
        "config": {
          "clientId": "serviceConfidentialClient",
          "clientSecretId": "client.secret.id",
          "secretsProvider" : "SystemAndEnvSecretStore-1",
        }
      }
    ]
  }
}
```

Default: ForgeRockClientHandler

"`failureHandler`": Handler <reference>, optional

Handler to manage a failed request.

Provide an inline handler configuration object, or the name of a handler object declared in the heap. The handler can access information in the OAuth2FailureContext.

Default: `500 Internal Server Error` , the request stops being executed.

Example

For an example of how this filter is used, see OAuth 2.0 token exchange.

More information

org.forgerock.http.oauth2.OAuth2TokenExchangeFilter

OAuth2TokenExchangeContext

OAuth2FailureContext

The OAuth 2.0 Authorization Framework

PasswordReplayFilter

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.

Usage

```
{
  "name": string,
  "type": "PasswordReplayFilter",
  "config": {
    "request": object,
    "loginPage": runtime expression<boolean>,
    "loginPageContentMarker": pattern,
    "credentials": Filter reference,
    "headerDecryption": object,
    "loginPageExtractions": [ object, ... ]
  }
}
```

Properties

"request": <object>, required

The HTTP request message that replays the credentials.

```
{
  "request": object,
    "method": config expression<string>,
    "uri": runtime expression<string>,
    "version": configuration expression<string>,
    "entity": runtime expression<string>,
    "headers": map,
    "form": map
}
```

For information about the properties of `request` see Request.

The JSON object of `request` is the `config` content of a StaticRequestFilter.

"`loginPage`": runtime expression<boolean>, required unless **`loginPageContentMarker`** *is defined*

`true` : Direct the request to a login page, extract credentials, and replay them.

`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` :

```
${find(request.uri.path, '/login') and (request.method ==
'GET')}
```

"`loginPageContentMarker`": pattern, required unless **`loginPage`** *is defined*

A pattern that matches when a response entity is a login page.

For an example route that uses this property, see Login form with password replay and cookie filters.

See also Patterns.

"`credentials`": Filter reference, optional

Filter that injects credentials, making them available for replay. Consider using a `FileAttributesFilter` or an `SqlAttributesFilter` .

When this is not specified, credentials must be made available to the request by other means.

See also Filters.

"`headerDecryption`": object, optional

Object to decrypt request headers that contain credentials to replay.

```
{
  "headerDecryption": object,
    "algorithm": configuration expression<string>,
    "headers": [ configuration expression<string>, ... ],
    "keySecretId": configuration expression<secret-id>,
    "secretsProvider": SecretsProvider reference,
    "charset": configuration expression<string>,
    "key": string, //deprecated
    "keyType": string //deprecated
}
```

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

Algorithm used for decryption. Use the same algorithm that is used to send the encrypted credentials.

Default: `AES/ECB/PKCS5Padding`

"`headers`": array of configuration expression<strings>, optional

The names of header fields to decrypt.

Default: Do not decrypt any headers.

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

The secret ID of the key to encrypt or decrypt the headers. This property takes precedence over the deprecated property `key` .

"`secretsProvider`": SecretsProvider reference, required

The SecretsProvider to resolve queried secrets, such as passwords and cryptographic keys. For allowed formats, see SecretsProvider.

"`charset`": configuration expression<string>, optional

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

Default: `UTF-8`

"`key`": string, optional

> IMPORTANT
>
> The use of this property is deprecated; use `keySecretId` and `secretsProvider` instead. For more information, refer to Deprecation.

Base64 encoded key value.

"`keyType`": string, required

> IMPORTANT
>
> The use of this property is deprecated; use `keySecretId` and `secretsProvider` instead. For more information, refer to Deprecation.

Algorithm name for the secret key.

Default: AES

"`loginPageExtractions`": array of <objects>, optional

Objects to extract values from the login page entity.

```
{
  "loginPageExtractions": [
```

```
    {
      "name": string,
      "pattern": pattern
    },
    ...
  ]
}
```

For an example route that uses this property, see Login which requires a hidden value from the login page.

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='(.*)'"`.

Example

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"
    }
  }
}
```

For additional examples, see Configuration templates, and the Javadoc for the PasswordReplayFilter class.

More information

org.forgerock.openig.filter.PasswordReplayFilterHeaplet

PolicyEnforcementFilter

Requests policy decisions from AM, which allows or denies the request based on the request context, the request URI, and the AM policies.

Processing requests after a policy decision

After a policy decision, IG continues to process requests as follows:

- If the request is allowed, processing continues.
- If the request is denied with advices, IG checks whether it can respond to the advices. If IG 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 IG cannot respond to the advice, IG forwards the request to a `failureHandler` declared in the `PolicyEnforcementFilter`. If there is no `failureHandler`, IG returns a 403 Forbidden.
- If an error occurs during the process, IG returns 500 Internal Server Error.

Using advices from policy decisions

Attributes and advices returned by the policy decision are stored in the `policyDecision` context. For information, see PolicyDecisionContext.

IG 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.

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 IG 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.

WebSocket notifications for policy changes

When WebSocket notifications are set up for changes to policies, IG receives a notification from AM when a policy decision is created, deleted, or updated.

For information about setting up WebSocket notifications, using them to clear the policy cache, and including them in the server logs, see WebSocket Notifications.

Usage

```
{
  "name": string,
  "type": "PolicyEnforcementFilter",
  "config": {
    "amService": AmService reference,
    "pepRealm": configuration expression<string>,
    "ssoTokenSubject": runtime expression<string>,
    "jwtSubject": runtime expression<string>,
    "claimsSubject": map,
    "cache": object,
    "application": configuration expression<string>,
    "environment": map,
    "failureHandler": Handler reference,
    "resourceUriProvider": ResourceUriProvider reference,
    "executor": Executor service reference //deprecated
  }
}
```

Properties

"amService": AmService reference, required

The AmService object to use for the following properties:

- `agent`, the credentials of the IG 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 IG 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.

  The AM version is derived as follows, in order of precedence:

  - Discovered value: AmService discovers the AM version. If `version` is configured with a different value, AmService ignores the value of `version` and issues a warning.
  - Value in `version`: AmService cannot discover the AM version, and `version` is configured.

- Default value of AM 6: AmService cannot discover the AM version, and `version` is not configured.

See also AmService.

"`pepRealm`": configuration expression<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 there is no SSO (API protection), `ssoTokenSubject` usually points to a header value such as `${request.headers.iPlanetDirectoryPro[0]}`, where `iPlanetDirectoryPro` is the name of the default AM session cookie. To find the name of your AM session cookie, see Find the name of your AM session cookie.

"`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 `AuthorizationCodeOAuth2ClientFilter` filter before the PEP filter, and then use `${attributes.openid.id_token}` as the expression value.

See also AuthorizationCodeOAuth2ClientFilter and Expressions.

"`claimsSubject`": 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.

The map can be structured as follows:

- As a runtime expression of a map, where the evaluation is an object of type `Map<String, Object>`.

- As a map whose key is a string, and whose value is a runtime expression.

Examples

```
"claimsSubject": "${attributes.openid.id_token_claims}"
```

```
"claimsSubject": {
  "sub": "${attributes.subject_identifier}",
  "iss": "am.example.com"
}
```

For an example of using `claimsSubject` as a map, see Example Policy Enforcement Using claimsSubject on this reference page.

"application": configuration 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"
}
```

`enabled`: configuration expression<boolean>, optional

Enable or disable caching of policy decisions.

When a policy decision is cached, IG can reuse the policy decision without repeatedly asking AM for a new policy decision. When caching is disabled, IG 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, IG uses that value or the `maxTimeout`.

Default: `1 minute`

"`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`, IG uses the `maximumTimeToCache`.

The duration cannot be `zero` or `unlimited`.

`onNotificationDisconnection`: configuration expression<enumeration>, optional

The strategy to manage the cache when the WebSocket notification service is disconnected, and IG receives no notifications for AM events. If the cache is not

cleared it can become outdated, and IG can allow requests on revoked sessions or tokens.

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, optional

Strings to forward to AM with the a policy decision request, to represent the environment (or context) of the request. Forward any HTTP header or any value that the AM policy definition can use.

The map can be structured as follows:

- As a runtime expression of a map, where the evaluation is an object of type `Map<String, Object>`.
- As a map whose key is a string, and whose value is a runtime expression.

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.

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}"
  ]
}
```

"`failureHandler`": Handler reference, optional

Handler to treat the request if it is 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,
        "headers": {
          "Content-Type": [ "text/plain; charset=UTF-8" ]
        },
        "entity": "Restricted area. You do not have sufficient
privileges."
      }
    }
  }
}
```

Provide an inline handler configuration object, or the name of a handler object declared 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 a 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": configuration expression<boolean>,
```

```
    "includeQueryParams": configuration expression<boolean>
  }
}
```

`useOriginalUri`: configuration expression<boolean>, optional

A flag to use the original URI of the request as the resource URL in policy decision requests to AM.

- `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`: configuration expression<boolean>, optional

A flag to 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://ig.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://ig.example.com:8080/login` .

  For AM policies that specify resource URLs without query parameters, use this option to reduce the amount of cached information.

Default: `true`

`ScriptableResourceUriProvider`

Use a script to return a resource URL to include in policy decision requests to AM. The result of the script must be a string that represents the resource URL.

```
"resourceUriProvider": {
  "type": "ScriptableResourceUriProvider",
  "config": {
    "type": configuration expression<string>,
    "file": configuration expression<string>, // Use either
"file"
    "source": [ string, ... ],                // 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()"
    ]
  }
}
```

"executor": Executor service reference, optional

IMPORTANT

This property is deprecated; use the `cache` subproperty `executor` instead. For more information, refer to Deprecation.

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

Default: `ForkJoinPool.commonPool()`

Example policy enforcement using claimsSubject

This route is a variant of the route in Enforce AM policy decisions in the same domain. It enforces a policy decision from AM, using `claimsSubject` instead of `ssoTokenSubject` to identify the subject.

To use this route:

1. Set up AM as described in Enforce AM policy decisions in the same domain.
2. In AM, select the policy you created in the previous step, and add a new resource:
   - **Resource Type**: `URL`
   - **Resource pattern**: `*://*:*/*`
   - **Resource value**: `http://app.example.com:8081/home/pep-claims`
3. In the same policy, add the following subject condition:

- Any of
- **Type** : OpenID Connect/JwtClaim
- **claimName** : iss
- **claimValue** : am.example.com

4. Set an environment variable for the IG agent password, and then restart IG:

```
$ export AGENT_SECRET_ID='cGFzc3dvcmQ='
```

The password is retrieved by a SystemAndEnvSecretStore, and must be base64-encoded.

5. Add the following route to serve static resources, such as .css, for the sample application:

Linux | Windows

```
$HOME/.openig/config/routes/static-resources.json
```

```
{
  "name" : "sampleapp-resources",
  "baseURI" : "http://app.example.com:8081",
  "condition": "${find(request.uri.path,'^/css')}",
  "handler": "ReverseProxyHandler"
}
```

6. Add the following route to IG:

Linux | Windows

```
$HOME/.openig/config/routes/04-pep-claims.json
```

```
{
  "name": "pep-claims",
  "baseURI": "http://app.example.com:8081",
  "condition": "${find(request.uri.path, '^/home/pep-
claims')}",
  "heap": [
    {
      "name": "SystemAndEnvSecretStore-1",
      "type": "SystemAndEnvSecretStore"
```

```
    },
    {
      "name": "AmService-1",
      "type": "AmService",
      "config": {
        "url": "http://am.example.com:8088/openam",
        "agent": {
          "username": "ig_agent",
          "passwordSecretId": "agent.secret.id"
        },
        "secretsProvider": "SystemAndEnvSecretStore-1",
        "version": "7.2"
      }
    }
  ],
  "handler": {
    "type": "Chain",
    "config": {
      "filters": [
        {
          "name": "SingleSignOnFilter-1",
          "type": "SingleSignOnFilter",
          "config": {
            "amService": "AmService-1"
          }
        },
        {
          "name": "PolicyEnforcementFilter-1",
          "type": "PolicyEnforcementFilter",
          "config": {
            "application": "PEP-SSO",
            "claimsSubject": {
              "sub": "${contexts.ssoToken.info.uid}",
              "iss": "am.example.com"
            },
            "amService": "AmService-1"
          }
        }
      ],
      "handler": "ReverseProxyHandler"
    }
  }
}
```

7. Go to http://ig.example.com:8080/home/pep-claims.

8. Log in to AM as user `demo`, password `Ch4ng31t`.

   AM returns a policy decision that grants access to the sample application.

More information

org.forgerock.openig.openam.PolicyEnforcementFilter

org.forgerock.openig.openam.PolicyDecisionContext

PolicyDecisionContext

AM's Authorization guide

PrivateKeyJwtClientAuthenticationFilter

Supports client authentication with the `private_key_jwt` client-assertion, using an unencrypted JWT.

Clients send a signed JWT to the authorization server. IG 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
```

Use this filter with an endpoint handler that requires authentication with the with the `private_key_jwt` client-assertion, using an unencrypted JWT. For example, the `endpointHandler` handler in the OAuth2TokenExchangeFilter.

Usage

```
{
  "name": string,
  "type": "PrivateKeyJwtClientAuthenticationFilter",
  "config": {
    "clientId": configuration expression<string>,
    "tokenEndpoint": configuration expression<url>,
```

```
    "secretsProvider": SecretsProvider reference,
    "signingSecretId": configuration expression<secret-id>,
    "signingAlgorithm": configuration expression<string>,
    "jwtExpirationTimeout": configuration expression<duration>,
    "claims": map
  }
}
```

Configuration

"clientId": configuration expression<string>, required
The `client_id` obtained when registering with the authorization server.

"tokenEndpoint": configuration expression<url>, required
The URL to the authorization server's OAuth 2.0 token endpoint.

"secretsProvider": SecretsProvider reference, required
The SecretsProvider to resolve queried secrets, such as passwords and cryptographic keys. For allowed formats, see SecretsProvider.

"signingSecretId": configuration expression<string>, required
Reference to the keys used to sign the JWT.

"signingAlgorithm": configuration expression<string>, optional
The JSON Web Algorithm (JWA) used to sign the JWT, such as:

- `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`

"jwtExpirationTimeout": configuration expression<duration>, optional
The duration for which the JWT is valid.

Default: 1 minute

"claims": map, optional
The claims used in the authentication.

The map can be structured as follows:

- As a configuration expression of a map, where the evaluation is an object of type `Map<String, Object>`.
- As a map whose key is a string, and whose values are configuration expressions.

Default: Empty

ResourceOwnerOAuth2ClientFilter

IMPORTANT

This filter uses the *Resource Owner Password Credentials* grant type. According to information in the The OAuth 2.0 Authorization Framework, minimize use of this grant type and utilize other grant types whenever possible. Use this filter in a service-to-service context, where services need to access resources protected by OAuth 2.0.

Authenticates OAuth 2.0 clients by using the resource owner'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.

Client authentication is provided by the `endpointHandler` property, which uses a client authentication filter.

The ResourceOwnerOAuth2ClientFilter refreshes the access token as required.

For more information, see RFC 6749 - Resource Owner Password Grant.

Usage

```
{
  "name": string,
  "type": "ResourceOwnerOAuth2ClientFilter",
  "config": {
    "username": configuration expression<string>,
    "passwordSecretId": configuration expression<secret-id>,
    "secretsProvider": SecretsProvider reference,
    "tokenEndpoint": configuration expression<url>,
    "scopes": [ configuration expression<string>, ... ],
    "endpointHandler": Handler reference
  }
}
```

Properties

"username": configuration expression<string>, required

The resource owner username to supply during authentication.

"passwordSecretId": configuration expression<secret-id>, required

The secret ID to obtain the resource owner password.

"`secretsProvider`": SecretsProvider reference, required

The SecretsProvider to resolve queried secrets, such as passwords and cryptographic keys. For allowed formats, see SecretsProvider.

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

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

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

Array of scope strings to request from the authorization server.

Default: Empty, request no scopes.

"`endpointHandler`": Handler reference, optional

The Handler to exchange tokens on the authorization endpoint.

Configure this property as a Chain, using one of the following client authentication filters:

- ClientSecretBasicAuthenticationFilter
- ClientSecretPostAuthenticationFilter
- PrivateKeyJwtClientAuthenticationFilter

```
{
  "name": "myHandler",
  "type": "Chain",
  "config": {
    "handler": "ForgeRockClientHandler",
    "filters": [
      {
        "type": "ClientSecretBasicAuthenticationFilter",
        "config": {
          "clientId": "myConfidentialClient",
          "clientSecretId": "my.client.secret.id",
          "secretsProvider" : "mySystemAndEnvSecretStore",
        }
      }
    ]
  }
}
```

Default: ForgeRockClientHandler

Examples

For an example, see Using OAuth 2.0 resource owner password credentials.

More information

org.forgerock.openig.filter.oauth2.client.ResourceOwnerOAuth2ClientFilterHeaplet

org.forgerock.openig.filter.oauth2.OAuth2ResourceServerFilterHeaplet

OAuth2ResourceServerFilter

The OAuth 2.0 Authorization Framework

The OAuth 2.0 Authorization Framework: Bearer Token Usage

ScriptableFilter

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.
- For information about creating scriptable objects in Studio, see Scripts in Studio and Configure scriptable throttling.

Usage

```
{
    "name": string,
    "type": "ScriptableFilter",
    "config": {
        "type": configuration expression<string>,
        "file": configuration expression<string>, // Use either
"file"
        "source": [ string, ... ],                // or "source",
but not both.
        "args": map,
        "clientHandler": Handler reference
```

```
    }
}
```

Properties

For information about properties for ScriptableFilter, see Scripts.

Examples

For an example scriptable filter that recovers policy advices from AM, see the `failureHandler` property of PolicyEnforcementFilter.

More information

Scripts

org.forgerock.openig.filter.ScriptableFilter

SessionInfoFilter

Calls the AM endpoint for session information, and makes the data available as a new context to downstream IG filters and handlers. For information, see SessionInfoContext.

WebSocket notifications for sessions

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

For information about setting up WebSocket notifications, using them to clear the session cache, and including them in the server logs, see WebSocket notifications.

Usage

```
{
  "name": string,
  "type": "SessionInfoFilter",
  "config": {
    "amService": AmService reference,
    "ssoToken": runtime expression<string>
  }
}
```

Properties

"amService": AmService reference, required

The AmService object to use for communication with AM.

The following `sessionProperties`, are retrieved from AM:

- 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 IG 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.

"ssoToken": runtime expression<string>, optional

Location of the AM SSO or CDSSO token.

This property can take the following values:

- `${contexts.ssoToken.value}`, when the SingleSignOnFilter is used for authentication
- `${contexts.ssoToken.value}` or `${contexts.cdsso.token}`, when the CrossDomainSingleSignOnFilter is used for authentication
- `${request.headers['mySsoToken'][0]}`, where the SSO or CDSSO 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 or CDSSO tokens.

Examples

For an example that uses the SessionInfoFilter, see Retrieve a username from the sessionInfo context.

More information

org.forgerock.openig.openam.SessionInfoFilter

org.forgerock.openig.openam.SessionInfoContext

SessionInfoContext

AM's Authorization guide

SetCookieUpdateFilter

Updates the attribute values of Set-Cookie headers in a cookie. This filter facilitates the transition to the SameSite and secure cookie settings required by newer browsers. Use SetCookieUpdateFilter at the beginning of a chain to guarantee security along the chain.

Set-Cookie headers must conform to grammar in RFC 6265: Set-Cookie.

Usage

```
{
  "name": string,
  "type": "SetCookieUpdateFilter",
  "config": {
   "cookies": {
     "cookie-name": {
       "attribute-name": "attribute-value",
       ...
     }
     ...
   }
 }
}
```

Properties

"cookies": map, required

Configuration that matches case-sensitive cookie names to response cookies, and specifies how matching cookies attribute values should be updated. Each cookie begins with a name-value pair, where the value is one or more attribute-value pairs.

cookie-name: pattern, required

The name of a cookie contained in the `Set-Cookie` header, as a pattern.

To change the attribute value on all existing cookies, specify `.*`.

If a cookie is named more than once, either explicitly or by the wildcard (`*`), the rules are applied to the cookie in the order they appear in the map.

In the following example, the SameSite attribute of the CSRF cookie first takes the value `none`, and then that value is overwritten by the value `LAX`.

```
"cookies": {
  "CSRF": {
    "value": "myValue",
```

```
      "secure": ${true},
      "SameSite": "none"
    }
    ".*": {
      "SameSite": "LAX"
    }
  }
```

attribute-name: *enumeration*, *required*
A case-insensitive enumeration of a Set-Cookie attribute name.

Attribute names include `SameSite`, `secure`, `http-only`, `value`, `expires`, `Max-Age`, `path`, and `domain`. For more information, see RFC 6265: Set-Cookie.

attribute-value: *runtime expression<string, boolean, or integer>*, *required*
The replacement value for the named attribute. The value must conform to the expected type for the attribute name:

- `secure`: runtime expression<boolean>. Required if `SameSite` is `none`
- `http-only`: runtime expression<boolean>.
- `Max-Age`: runtime expression<number>.
- `SameSite`, and all other attribute names: runtime expression<string>.

For all values except `expires`, specify `${previous}` to reuse the existing value for the attribute. The following example adds five seconds to the `Max-Age` attribute:

```
"Max-Age": "${integer(previous+5)}",
```

If the named the Set-Cookie header doesn't contain the named attribute, `${previous}` returns null.

Examples

The following example updates attributes of all existing Set-Cookie headers:

```
{
  "name": "SetCookieUpdateFilter",
  "condition": "${find(request.uri.path, '/home')}",
  "baseURI": "http://app.example.com:8081",
  "heap": [],
  "handler": {
    "type": "Chain",
```

```
    "config": {
      "filters": [{
        "type": "SetCookieUpdateFilter",
        "config": {
          "cookies": {
            ".*": {
              "SameSite": "LAX",
              "domain": "ig.example.com",
              "Max-Age": "${session.maxAge}",
              "Secure": "${true}"
            }
          }
        }
      }],
      "handler": "ReverseProxyHandler"
    }
  }
}
```

More information

org.forgerock.openig.filter.SetCookieUpdateFilter

SingleSignOnFilter

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, IG checks the `goto` query parameter for the presence of the redirection marker:
  - If the redirection marker parameter is present (for example, `_ig=true`), IG fails the request because the cookie domain is incorrectly configured.
  - If the redirection marker is not present, IG redirects the user-agent to the AM login page or another provided login page. For information about enabling, disabling, and naming the redirection marker, see the `redirectionMarker` property on this page.

For more information about SSO, see Single sign-on and cross-domain single sign-on.

TIP

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 IG 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

When AM is using CTS-based sessions, it does not monitor idle time for client-side sessions, and so refresh requests are ignored.

When the SingleSignOnFilter is used for authentication with AM, after a time AM can view the session as idle even though the user continues to interact with IG. The user session can eventually time out.

(From AM 6.5.3.) When AM is using CTS-based sessions, use the `sessionIdleRefresh` property of AmService to refresh idle sessions, and prevent unwanted timeouts.

WebSocket notifications for sessions

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

For information about setting up WebSocket notifications, using them to clear the session cache, and including them in the server logs, see WebSocket notifications.

Usage

```
{
  "name": string,
  "type": "SingleSignOnFilter",
    "config": {
      "amService": AmService reference,
      "authenticationService": configuration expression<string>,
      "redirectionMarker": object,
      "realm": configuration expression<string>,
      "defaultLogoutLandingPage": configuration expression<url>,
      "loginEndpoint": runtime expression<url>,
      "logoutExpression": runtime expression<boolean>,
```

```
      "logoutEndpoint": pattern //deprecated
   }
}
```

Properties

"amService": AmService reference, required

An AmService object to use for the following properties:

- `agent` , the credentials of the IG 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 IG 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.

  The AM version is derived as follows, in order of precedence:

  - Discovered value: AmService discovers the AM version. If `version` is configured with a different value, AmService ignores the value of `version` and issues a warning.
  - Value in `version` : AmService cannot discover the AM version, and `version` is configured.
  - Default value of AM 6: AmService cannot discover the AM version, and `version` is not configured.

redirectionMarker: object, optional

```
"redirectionMarker": {
   "enabled": configuration expression<boolean>,
   "name": configuration expression<string>
}
```

enabled: configuration expression<boolean>, optional

When `true` , use a redirection marker to limit the number of authentication redirects.

When there is no SSO session due to, for example, SSO cookie name misconfiguration, an authentication request fails and is redirected back to IG. The scenario can result in infinite authentication redirects.

Consider enabling the redirection marker during development, then disabling it to prevent it being captured in bookmarks.

+ Default: `true`

`name`: configuration expression<string>, optional

The name of the redirection marker query parameter to use when `enabled` is `true`.

Default: `_ig`

"`authenticationService`": configuration expression<string>,optional

The name of an AM authentication tree or authentication chain to use for authentication.

Default: AM's default authentication service.

For an example that uses `authenticationService`, see Authenticate with SSO through an AM authentication tree.

For more information about authentication trees and chains, see Authentication nodes and trees and Authentication modules and chains in AM's *Authentication and SSO* guide.

"`realm`": configuration expression<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 if `logoutExpression` is evaluated as true.

If this property is not an absolute URL, the request is redirected to the IG domain name.

This parameter is effective only when `logoutExpression` 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 AM's Authorization guide. For information about supported advice types in IG, 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 Authenticating (browser) in AM's *Authentication and SSO* 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://am.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.rout
  er.originalUri)}"
  ```

  When using this option, review the cookie domains to make sure that cookies set by the authentication server are properly conveyed to the IG instance.

"`logoutExpression`": runtime expression<boolean>, optional

A flag to indicate that a logout condition is met. The condition is based on the request:

- `true`: The AM session token for the end user is revoked. If a `defaultLogoutLandingPage` is specified, the request is redirected to that

page.

- `false` : 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` :

```
${find(request.uri, '/logout')}
```

- The request path starts with `/logout` :

```
${find(request.uri.path, '^/logout')}
```

- The request query includes the `logOff=true` query parameter:

```
${find(request.uri.query, 'logOff=true')}
```

Default: `${false}`

"logoutEndpoint": pattern, optional

IMPORTANT

The use of this property is deprecated; use `logoutExpression` instead. For more information, refer to Deprecation.

A string denoting a regular expression pattern for a URL. When a request matches the pattern, IG performs the logout process and 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.

Default: Logout is not managed by this filter.

More information

org.forgerock.openig.openam.SingleSignOnFilter

org.forgerock.openig.openam.SsoTokenContext

SsoTokenContext

SqlAttributesFilter

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.

Usage

```
{
  "name": string,
  "type": "SqlAttributesFilter",
   "config": {
     "dataSource": JdbcDataSource reference,
     "preparedStatement": configuration expression<string>,
     "parameters": [ runtime expression<string>, ... ],
     "target": lvalue-expression
   }
}
```

Properties

"dataSource": JdbcDataSource reference, required

IMPORTANT

`dataSource` as a JNDI lookup name is deprecated; use `dataSource` as a JdbcDataSource configuration object instead. For more information, refer to Deprecation.

The JdbcDataSource to use for connections. Configure JdbcDataSource as described in JdbcDataSource.

"preparedStatement": configuration expression<string>, required

The parameterized SQL query to execute, with `?` parameter placeholders.

"parameters": array of runtime expressions<strings>, 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.

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.

More information

org.forgerock.openig.filter.SqlAttributesFilter

StaticRequestFilter

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`.

Usage

```
{
  "name": string,
  "type": "StaticRequestFilter",
  "config": {
    "method": configuration expression<string>,
    "uri": runtime expression<url>,
    "version": configuration expression<string>,
    "headers": {
      configuration expression<string>: [ runtime
expression<string>, ... ], ...
    },
    "form": {
      configuration expression<string>: [ runtime
expression<string>, ... ], ...
    },
    "entity": runtime expression<string>
  }
}
```

Properties

"method": configuration expression<string>, required

The HTTP method to be performed on the resource; for example, `GET`.

"uri": runtime expression<url>, required

The fully-qualified URI of the resource being accessed; 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 mutable URI.

"version": configuration expression<string>, optional

Protocol version.

Default: `"HTTP/1.1"`

"headers": map, optional

One or more headers to set for a request, with the format *`name`*`: [ `*`value`*`, … ]`, where:

- *name* is a configuration expression<string> that resolve to a header name. If multiple expressions resolve to the same final string, *name* has multiple values.
- *value* is one or more runtime expression<strings> that resolve to header values.

In the following example, the header name is the value of the system variable defined in `cookieHeaderName`. The header value is stored in `contexts.ssoToken.value`:

```
"headers": {
  "${application['header1Name']}": [
    "${application['header1Value'}"
  ]
}
```

CAUTION

For IG in web container mode, sending or receiving headers or trailers whose names or values contain non-ASCII characters can cause unspecified behavior at the HTTP server level, and character corruption at the ClientHandler/ReverseProxyHandler level.

Default: Empty

"form": map, optional

A form to include in the request and/or `application/x-www-form-urlencoded` entity, as name-value pairs, where:

- *name* is a configuration expression<string> that resolves to a form parameter name.
- *value* is one or more runtime expression<strings> that resolve to form parameter values.

When a Request `method` is `POST`, `form` is mutually exclusive with `entity`.

Examples:

- In the following example, the field parameter names and values are hardcoded in the form:

```
"form": {
  "username": [
    "demo"
  ],
  "password": [
    "password"
  ]
}
```

- In the following example, 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 parameter takes 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]}"
  ]
}
```

Default: Empty

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

The message entity body to include in a request.

When a Request `method` is `POST`, `entity` is mutually exclusive with `form`.

Methods are provided for accessing the entity as byte, string, or JSON content. For information, see Entity.

IMPORTANT

Attackers during reconnaissance can use messages to identify information about a deployment. For security, limit the amount of information in messages, and avoid using words that help identify IG.

Default: Empty

Example

In the following example, IG replaces the browser's original HTTP GET request with an HTTP POST login request containing credentials to authenticate to the sample

application. For information about how to set up and test this example, see the Getting started.

```
{
  "handler": {
    "type": "Chain",
    "config": {
      "filters": [
        {
          "type": "StaticRequestFilter",
          "config": {
            "method": "POST",
            "uri": "http://app.example.com:8081/login",
            "form": {
              "username": [
                "demo"
              ],
              "password": [
                "Ch4ng31t"
              ]
            }
          }
        }
      ],
      "handler": "ReverseProxyHandler"
    }
  },
  "condition": "${find(request.uri.path, '^/static')}"
}
```

More information

org.forgerock.openig.filter.StaticRequestFilter

SwitchFilter

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.

Usage

```
{
    "name": string,
    "type": "SwitchFilter",
    "config": {
        "onRequest": [
            {
                "condition": runtime expression<boolean>,
                "handler": Handler reference
            },
            ...
        ],
        "onResponse": [
            {
                "condition": runtime expression<boolean>,
                "handler": Handler reference
            },
            ...
        ]
    }
}
```

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

A flag to indicate that a condition is met:

- `true`: The request is dispatched to the handler.
- `false`: The request is not dispatched to the handler, and the next condition in the list is tried.

  When the last condition in the list returns `false`, the request is passed to the next filter or handler in the chain.

Default: `${true}`

"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.

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"
            }
        ]
    }
}
```

More information

org.forgerock.openig.filter.SwitchFilter

ThrottlingFilter

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, IG 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://ig.example.com:8080/home/throttle-scriptable HTTP/1.1
. . .
HTTP/1.1 429 Too Many Requests
Retry-After: 10
```

Usage

```
{
    "name": string,
    "type": "ThrottlingFilter",
    "config": {
        "requestGroupingPolicy": runtime expression<string>,
        "throttlingRatePolicy": ThrottlingPolicy reference,  //Use
either "throttlingRatePolicy"
        "rate": {                                            //or
"rate", but not both.
            "numberOfRequests": configuration expression<number>,
            "duration": configuration expression<duration>
        },
        "cleaningInterval": configuration expression<duration>,
        "executor": ScheduledExecutorService reference
    }
}
```

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": ThrottlingPolicy reference, 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": 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`": configuration expression<integer>, required
The number of requests allowed through the filter in the time specified by `"duration"`.

"`duration`": configuration expression<duration>, required
A time interval during which the number of requests passing through the filter is counted.

"`cleaningInterval`": configuration expression<duration>, optional
The time to wait before cleaning outdated partitions. The value must be more than zero but not more than one day.

"`executor`": ScheduledExecutorService reference, 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.

Examples

- Example of a Mapped Throttling Policy
- Example of a Scriptable Throttling Policy

The following route defines a throttling rate of 6 requests/10 seconds to requests. For information about how to set up and test this example, see Configure Simple Throttling.

```
{
  "name": "00-throttle-simple",
  "baseURI": "http://app.example.com:8081",
  "condition": "${find(request.uri.path, '^/home/throttle-
simple')}",
  "handler": {
    "type": "Chain",
    "config": {
      "filters": [
        {
          "type": "ThrottlingFilter",
          "name": "ThrottlingFilter-1",
          "config": {
            "requestGroupingPolicy": "",
            "rate": {
```

```
                "numberOfRequests": 6,
                "duration": "10 s"
              }
            }
          }
        ],
        "handler": "ReverseProxyHandler"
      }
    }
}
```

More information

org.forgerock.openig.filter.throttling.ThrottlingFilterHeaplet

TokenTransformationFilter

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 IG administrator.

Usage

```
{
    "name": "string",
    "type": "TokenTransformationFilter",
```

```
    "config": {
        "amService": AmService reference,
        "idToken": runtime expression<string>,
        "instance": configuration expression<string>,
        "username": configuration expression<string>, //deprecated
        "password": configuration expression<string>  //deprecated
    }
}
```

Properties

"amService": AmService reference, required

The AmService heap object to use for the following properties:

- `agent`, the credentials of the IG agent in AM, to authenticate IG as an AM REST STS client, and to communicate WebSocket notifications from AM to IG. 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.

"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`.

"instance": configuration expression<string>, 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`.

"username": string, required

IMPORTANT

The use of this property is deprecated; use the AmService property `agent` instead. For more information, refer to Deprecation.

The username to authenticate IG as an AM REST STS client.

"password": expression, required

IMPORTANT

The use of this property is deprecated; use the AmService property `agent` instead. For more information, refer to Deprecation.

The password to authenticate IG as an AM REST STS client.

Example

The following example shows a configuration for a TokenTransformationFilter:

```
{
  "type": "TokenTransformationFilter",
  "config": {
    "amService": "MyAmService",
    "idToken": "${attributes.openid.id_token}",
    "instance": "openig"
  }
}
```

For an example of how to set up and test the TokenTransformationFilter, see Transform OpenID Connect ID tokens into SAML assertions.

More information

org.forgerock.openig.openam.TokenTransformationFilter

org.forgerock.openig.openam.StsContext

StsContext

TransactionIdOutboundFilter

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

The default TransactionIdOutboundFilter is created by IG, and used in ForgeRockClientHandler, as follows:

```
{
     "name": "ForgeRockClientHandler",
     "type": "ForgeRockClientHandler",
     "config": {
         "filters": [ "TransactionIdOutboundFilter" ],
         "handler": "ClientHandler"
```

```
    }
}
```

More information

org.forgerock.http.filter.TransactionIdOutboundFilter

UmaFilter

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.

Usage

```
{
    "name": string,
    "type": "UmaFilter",
    "config": {
        "protectionApiHandler": Handler reference,
        "umaService": UmaService reference,
        "realm": configuration expression<string>
    }
}
```

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": configuration expression<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`

More information

User-Managed Access (UMA) Profile of OAuth 2.0

org.forgerock.openig.uma.UmaResourceServerFilter

UriPathRewriteFilter

Rewrite a URL path, using a bidirectional mapping:

- In the request flow, `fromPath` is mapped to `toPath`.
- In the response flow, `toPath` is mapped to `fromPath`.

IG overwrites a response header only when all of the following conditions are true:

- The response includes a header such as `Location` or `Content-Location`
- The URI of the response header matches the mapping
- The value of response header is a relative path or its `scheme://host:port` value matches the base URI.

Usage

```
{
  "name": string,
  "type": "UriPathRewriteFilter",
  "config": {
    "mappings": object,
    "failureHandler": Handler reference
  }
}
```

Properties

"mappings": object, required

One or more bidirectional mappings between URL paths. For example mappings, request scenarios, and an example route, see Examples.

```
{
  "mappings": {
    "/fromPath1": "/toPath1",
    "/fromPath2": "/toPath2",
```

```
    ...
  }
}
```

Paths are given by a `configuration expression<string>`. Consider the following points when you define paths:

- The incoming URL must start with the mapping path.
- When more than one mapping applies to a URL, the most specific mapping is used.
- Duplicate `fromPath` values are removed without warning.
- Trailing slashes `/` are removed from path values.
- If the response includes a `Location` or `Content-Location` header with a `toPath` in its URL, the response is rewritten with `fromPath`.

"failureHandler": handler reference, optional

Failure handler to be invoked if an invalid URL is produced when the request path is mapped, or when the response `Location` or `Content-Location` header URI path is reverse-mapped.

Provide an inline handler declaration, or the name of a handler object defined in the heap. See also Handlers.

Default: HTTP 500

Examples

Valid and invalid mapping examples

The following mapping examples are valid:

- Single `fromPath` and `toPath`

```
"mappings": {
  "/fromPath1": "/toPath1",
  "/fromPath2": "/toPath2"
}
```

- Expressions in the `fromPath` and `toPath`

```
"mappings": {
  "/${join(array(`fromPath`, 'path1'), `/`)}":
"/${join(array(`toPath`, 'path2'), `/`)}"
}
```

- Expressions in the `fromPath` and `toPath` that use predefined heap properties

```
"mappings": {
  "${fromPath}": "${toPath}"
}
```

- No mappings—the configuration is valid, but has no effect

```
"mappings": { }
```

- Duplicate `toPath`

```
"mappings": {
  "/fromPath1": "/toPath",
  "/fromPath2": "/toPath"
}
```

- Duplicate `fromPath` —the configuration is overwritten without warning

```
"mappings": {
  "/fromPath": "/toPath1",
  "/fromPath": "/toPath2"
}
```

The following mapping examples are not valid

- No `toPath`

```
"mappings": {
  "/fromPath": ""
}
```

```
"mappings": {
  "/fromPath": "${unknown}"
}
```

- Invalid `toPath`

```
"mappings": {
  "/fromPath": "${invalidExpression}"
}
```

- No `fromPath`

```
"mappings": {
  "": "/toPath"
}
```

```
"mappings": {
    "${unknown}": "/toPath"
}
```

- Invalid `fromPath`

```
"mappings": {
    "${invalidExpression}": "/toPath"
}
```

Example request scenarios

Description	Mapping	Inbound URI	Rewritten URI
Basic path	`"mappings": { "/fromPath": "/toPath" }`	http://example.com/fromPath/remainder	http://example.com/toPath/remainder
Root context, where the inbound request URI has a `/` path segment	`"mappings": { "/": "/rootcontext" }`	http://example.com/	http://example.com/rootcontext/
Root context, where the inbound URI has a `/` path segment	`"mappings": { "/rootcontext": "/" }`	http://example.com/rootcontext/	http://example.com/
Root context, where the inbound request URI has an empty path	`"mappings": { "/": "/rootcontext" }`	http://example.com	http://example.com/rootcontext

Description	Mapping	Inbound URI	Rewritten URI
Root context, where the rewritten URI has an empty path	`"mappings": { "/rootcontext": "/" }`	http://example.com/rootcontext	http://example.com
Root context, with path remainder	`"mappings": { "/": "/rootcontext" }`	http://example.com/remainder	http://example.com/rootcontext/remainder
Root context, with path remainder	`"mappings": { "/rootcontext": "/" }`	http://example.com/rootcontext/remainder	http://example.com/remainder
Root context, where the trailing `/` on `toPath` is ignored	`"mappings": { "/": "/rootcontext/" }`	http://example.com/remainder	http://example.com/rootcontext/remainder
Path with dot-segments:	`"mappings": { "/fromPath": "/toPath1/../toPath2" }`	http://example.com/fromPath	http://example.com/toPath1/../toPath2
Path with syntax:	`"mappings": { "/fromPath;v=1.1": "/toPath,1.1" }`	http://example.com/fromPath;v=1.1	http://example.com/toPath,1.1
Path with syntax:	`"mappings": { "/$fromPath": "/$toPath" }`	http://example.com/$fromPath	http://example.com/$toPath

Description	Mapping	Inbound URI	Rewritten URI
Path with query parameters	`"mappings": { "/fromPath": "/toPath" }`	http://example.com/fromPath?param1&param2=2	http://example.com/toPath?param1&param2=2
Path with fragment	`"mappings": { "/fromPath": "/toPath" }`	http://example.com/fromPath#fragment	http://example.com/toPath#fragment

Example route

The example route changes a request URL as follows:

- The `baseURI` overrides the scheme, host, and port of a request URL.
- The UriPathRewriteFilter remaps the path of a request URL.

  Requests to `http://ig.example.com:8080/mylogin` are mapped to `http://app.example.com:8081/login`.

  Requests to `http://ig.example.com:8080/welcome` are mapped to `http://app.example.com:8081/home`.

  Requests to `http://ig.example.com:8080/other` are mapped to `http://app.example.com:8081/not-found`, and result in an HTTP 404.

  Requests to `http://ig.example.com:8080/badurl` are mapped to the invalid URL `http://app.example.com:8081[`, and invoke the failure handler.

```
{
  "name": "UriPathRewriteFilter",
  "baseURI": "http://app.example.com:8081",
  "handler": {
    "type": "Chain",
    "config": {
      "filters": [
        {
          "type": "UriPathRewriteFilter",
          "config": {
            "mappings": {
              "/mylogin": "/login",
              "/welcome": "/home",
```

```
            "/other": "/not-found",
            "/badurl": "["
          },
          "failureHandler": {
            "type": "StaticResponseHandler",
            "config": {
              "status": 500,
              "headers": {
                "Content-Type": [
                  "text/plain"
                ]
              },
              "entity": "Invalid URL produced"
            }
          }
        }
      }
    ],
    "handler": "ClientHandler"
  }
 }
}
```

More information

org.forgerock.openig.filter.UriPathRewriteFilter

RFC 3986: Path

UserProfileFilter

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 IG filters and handlers, in the context UserProfileContext.

Usage

```
{
  "name": string,
  "type": "UserProfileFilter",
```

```
  "config": {
    "username": runtime expression<string>,
    "userProfileService": UserProfileService reference,
    "amService": AmService reference, //deprecated
    "ssoToken": runtime expression<string>,  //deprecated
    "profileAttributes": array of runtime expressions<string>
//deprecated
  }
}
```

Properties

"username": runtime 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": [ configuration expression<string>,
... ],
    "realm": configuration expression<string>
  }
}
```

"amService": AmService reference, required

The AmService heap object to use for the following properties:

- `agent`, the credentials of the IG 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 IG agent in AM.
- `version`: The version of the AM server.

  The AM version is derived as follows, in order of precedence:

- Discovered value: AmService discovers the AM version. If `version` is configured with a different value, AmService ignores the value of `version` and issues a warning.
- Value in `version`: AmService cannot discover the AM version, and `version` is configured.
- Default value of AM 6: AmService cannot discover the AM version, and `version` is not configured.

"cache": object, optional

Caching of AM user profiles, based on *Caffeine*. For more information, see the GitHub entry, Caffeine.

When caching is enabled, IG can reuse cached profile attributes without repeatedly querying AM. When caching is disabled, IG must query AM for each request, to retrieve the required user profile attributes.

Default: No cache.

```
"cache": {
    "enabled": configuration expression<boolean>,
    "executor": Executor reference,
    "maximumSize": configuration expression<number>,
    "maximumTimeToCache": configuration
expression<duration>,
}
```

`enabled`: configuration expression<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 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>, required

The maximum duration for which to cache user profiles.

The duration cannot be `zero`.

`profileAttributes`: array of configuration expression<strings>, 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`.

"`ssoToken`": runtime expressions<string>, optional

IMPORTANT

The use of this property is deprecated; use `username` instead. For more information, refer to Deprecation.

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.

If an SSO token is not available in the request, an error is returned.

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.

"`amService`": AmService reference, optional

IMPORTANT

IMPORTANT

The use of this property is deprecated; use the `userProfileService` subproperty `amService` instead. For more information, refer to Deprecation.

The AmService heap object to use.

"`profileAttributes`": array of runtime expressions<string>, optional

IMPORTANT

The use of this property is deprecated; use the `userProfileService` subproperty `profileAttributes` instead. For more information, refer to Deprecation.

List of field names in AM to retrieve.

When this property is not configured, all available fields are returned. When this property is configured, the specified fields and the following fields are returned: `username`, `_id`, and `_rev`.

Field names are defined by the underlying repository in AM. For example, when AM is installed with the default configuration, the repository is ForgeRock Directory Server.

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()}`.

Default: All available fields are returned.

Example

For examples that use the UserProfileFilter, see Get user profile information from AM.

More information

org.forgerock.openig.openam.UserProfileFilter

org.forgerock.openig.tools.userprofile.UserProfileService

org.forgerock.openig.openam.UserProfileContext

UserProfileContext

AM's Authorization guide

Decorators

IG provides the following decorators to extend what objects can do.

For an overview of how decorators are implemented in IG, see Decorators.

BaseUriDecorator

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.

Decorator Usage

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

A BaseUriDecorator does not have configurable properties.

IG 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

Decorated Object Usage

```
{
    "name": string,
    "type": string,
    "config": object,
    decorator name: runtime expression<url>
}
```

"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 Filters or a Handlers.

"config": object required unless empty

The configuration of the object, just like an object that is not decorated

decorator name: runtime expression<url>, 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.

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/"
}
```

More information

org.forgerock.openig.decoration.baseuri.BaseUriDecorator

CaptureDecorator

Captures request and response messages in SLF4J logs, 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.

IMPORTANT

IMPORTANT

During debugging, consider using a CaptureDecorator to capture the entity and context of requests and responses. However, increased logging consumes resources, such as disk space, and can cause performance issues. In production, reduce logging by disabling the CaptureDecorator properties `captureEntity` and `captureContext`, or setting `maxEntityLength`.

Decorator Usage

```
{
  "name": string,
  "type": "CaptureDecorator",
  "config": {
    "captureEntity": configuration expression<boolean>,
    "captureContext": configuration expression<boolean>,
    "maxEntityLength": configuration expression<number>,
    "masks": object
  }
}
```

"captureEntity": configuration expression<boolean>, optional

When `true`, capture the message entity and write it to the logs. The message entity is the body of the HTTP message, which can be a JSON document, XML, HTML, image, or other information.

When `false`, do not capture the message entity, and write nothing to the logs.

If the `Content-Type` header is set for a request or response, the decorator uses it to decode the request or response messages, and then writes them to the logs. If the `Content-Type` header is not set, the decorator does not write the request or response messages to the logs.

The decorator excludes binary entities from the capture, instead of writing a `[binary entity]` marker.

Default: `false`

"captureContext": configuration expression<boolean>, optional

When `true`, capture the the context as JSON. The context chain is used when processing the request inside IG in the filters and handlers.

Default: `false`

"maxEntityLength": configuration expression<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)

"masks": object, optional

The configuration to mask the values of headers and attributes in the logs.

For an example, see Masking Values of Headers and Attributes.

```
{
  "masks": {
    "headers": [ pattern, ... ],
    "trailers": [ pattern, ... ]
    "attributes": [ pattern, ... ]
    "mask": [ configuration expression<string>, ... ]
  }
}
```

"headers": array of patterns, optional

The case-insensitive name of one or more headers whose value to mask in the logs.

The following value masks headers called `X-OpenAM-Username`, `X-OpenAM-Password` and `x-openam-token`:

```
"headers": ["X-OpenAM-.*"]
```

Default: None

"trailers": array of patterns, optional

The case-insensitive name of one or more trailers whose value to mask in the logs.

The following value masks trailers called `Expires`:

```
"trailers": ["Expires"]
```

Default: None

"attributes": array of patterns, optional

The case-insensitive name of one or more attributes whose value to mask in the logs.

Default: None

"mask": configuration expression<string>, optional
Text to replace the masked header value or attribute value in the logs.

Default: `*****`

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 decorated object.

"type": string, required except for inline objects, 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 decorated object, as documented in the object reference page.

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.

Examples

Log the entity

The following example decorator is configured to log the entity:

```
{
    "name": "capture",
    "type": "CaptureDecorator",
    "config": {
        "captureEntity": true
    }
}
```

Do not log the entity

The following example decorator is configured not to log the entity:

```
{
  "name": "capture",
  "type": "CaptureDecorator"
}
```

Log the context

The following example decorator is configured to log the context in JSON format, excluding the request and the response:

```
{
  "name": "capture",
  "type": "CaptureDecorator",
  "config": {
    "captureContext": true
  }
}
```

Log requests and responses with the entity

The following example decorator is configured to log requests and responses with the entity, before sending the request and before returning the response:

```
{
  "heap": [
    {
      "name": "capture",
      "type": "CaptureDecorator",
      "config": {
        "captureEntity": true
      }
    },
    {
      "name": "ReverseProxyHandler",
      "type": "ReverseProxyHandler",
      "capture": [
        "request",
        "response"
      ]
    }
  ],
  "handler": "ReverseProxyHandler"
}
```

Capture transformed requests and responses

The following example uses the default CaptureDecorator to capture transformed requests and responses, as they leave filters:

```
{
  "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,
          "headers": {
            "Content-Type": [ "text/html; charset=UTF-8" ]
          },
          "entity": "<html><body><p>Hello world!</p></body>
</html>"
        }
      }
    }
  },
  "capture": [
    "filtered_request",
    "filtered_response"
  ]
}
```

Capture the context as JSON

The following example captures the context as JSON, excluding the request and response, before sending the request and before returning the response:

```
{
  "heap": [
    {
      "name": "capture",
```

```
      "type": "CaptureDecorator",
      "config": {
        "captureContext": true
      }
    },
    {
      "name": "ReverseProxyHandler",
      "type": "ReverseProxyHandler",
      "capture": [
        "request",
        "response"
      ]
    }
  ],
  "handler": "ReverseProxyHandler"
}
```

Mask values of headers and attributes

This example captures the context, and then masks the value of the cookies and credentials in the logs. To try it, set up the example in Log in with credentials from a file, replace that route with the following route, and search the route log file for the text `MASKED`:

```
{
  "heap": [{
    "name": "maskedCapture",
    "type": "CaptureDecorator",
    "config": {
      "captureContext": true,
      "masks": {
        "headers": [ "cookie*", "set-cookie*" ],
        "attributes": [ "credentials" ],
        "mask": "MASKED"
      }
    }
  }],
  "name": "02-file-masked",
  "condition": "${find(request.uri.path, '^/profile')}",
  "maskedCapture": "all",
  "handler": {
    "type": "Chain",
    "baseURI": "http://app.example.com:8081",
    "config": {
```

```
      "filters": [
        {
          "type": "PasswordReplayFilter",
          "config": {
            "loginPage": "${find(request.uri.path,
'^/profile/george') and (request.method == 'GET')}",
            "credentials": {
              "type": "FileAttributesFilter",
              "config": {
                "file": "/tmp/userfile.txt",
                "key": "email",
                "value": "george@example.com",
                "target": "${attributes.credentials}"
              }
            },
            "request": {
              "method": "POST",
              "uri": "http://app.example.com:8081/login",
              "form": {
                "username": [
                  "${attributes.credentials.username}"
                ],
                "password": [
                  "${attributes.credentials.password}"
                ]
              }
            }
          }
        },
        "handler": "ReverseProxyHandler"
    }
  }
}
```

More information

org.forgerock.openig.decoration.capture.CaptureDecorator

TimerDecorator

Records time to process filters, handlers, and access token resolvers.

Decorator usage

```
{
    "name": string,
    "type": "TimerDecorator",
    "config": {
        "timeUnit": configuration expression<string>
    }
}
```

IG configures a default TimerDecorator named `timer`. Use `timer` as the decorator name without explicitly declaring a decorator named timer.

"timeUnit": configuration expression<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`

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 to decorate.

"type": string, required

The class name of the object to decorate, which must be a Filter, Handler, or the `accessTokenResolver` property of OAuth2ResourceServerFilter.

"config": object, optional

The configuration of the object, just like an object that is not decorated.

Default: Empty

decorator name: configuration expression<boolean>, required

IG looks for the presence of the decorator name field for the TimerDecorator:

- `true` : Activate the timer
- `false` : Deactivate the TimerDecorator

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 IG is set up as described in the documentation, the endpoint is http://ig.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	Monitoring type	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.
`ig_timerdecorator_filter_downstream_seconds`	`Summary`	Time to process the request and response in filters and handlers that are downstream of the decorated filter.

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 IG is set up as described in the documentation, the endpoint is http://ig.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	Monitoring type	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.

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
```

Examples

The following example uses the default timer decorator to record the time that TokenIntrospectionAccessTokenResolver takes to process a request:

```
{
  "accessTokenResolver": {
    "name": "TokenIntrospectionAccessTokenResolver-1",
    "type": "TokenIntrospectionAccessTokenResolver",
    "config": {
      "amService": "AmService-1",
      ...
    },
    "timer": true
  }
}
```

The following example defines a customized timer decorator in the heap, and uses it to record the time that the SingleSignOnFilter takes to process a request:

```
{
  "heap": [
    {
      "name": "mytimerdecorator",
      "type": "TimerDecorator",
      "config": {
        "timeUnit": "nano"
      }
    },
    ...
  ],
  "handler": {
    "type": "Chain",
    "config": {
      "filters": [
        {
          "type": "SingleSignOnFilter",
          "config": {
            ...
```

```
            },
            "mytimerdecorator": true
          }
        ],
        "handler": "ReverseProxyHandler"
      }
    }
}
```

More information

org.forgerock.openig.decoration.timer.TimerDecorator

Audit framework

IG 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.

The following objects are available for auditing:

AuditService

Configure an 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.

For information, see Record custom audit events.

To configure auditing for the whole configuration in a deployment, define an AuditService object named `AuditService` in the top-level heap.

To configure different auditing in different parts of the deployment, define multiple AuditServices with different names, in routes or lower-level heaps.

When you define multiple AuditServices that use JsonAuditEventHandler or CsvAuditEventHandler, configure each of the event handlers with a different

`logDirectory` . This prevents the AuditServices from logging to the same audit logging file.

Default audit service

By default, there is no auditing of a configuration. The NoOpAuditService object provides an empty audit service to the top-level heap and its child routes.

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.

Usage

```
{
  "name": string,
  "type": "AuditService",
  "config": {
    "config": object,
    "eventHandlers": [ object, ...],
    "topicsSchemasDirectory": configuration expression<string>,
    "event-handlers": [ object, ...] // deprecated
  }
}
```

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.

```
{
  "config": {
    "handlerForQueries": configuration_expression<string>,
    "availableAuditEventHandlers":
[configuration_expression<string>, ...],
    "caseInsensitiveFields": [configuration_expression<string>,
...],
    "filterPolicies": {
```

```
      "field": {
        "includeIf": [configuration_expression<string>, ...],
        "excludeIf": [configuration_expression<string>, ...]
      }
    }
  }
}
```

"`handlerForQueries`": configuration expression<string>, optional

The name of the event handler to use when querying audit event messages over REST.

"`availableAuditEventHandlers`": array of configuration expression<strings>, optional

A list of fully qualified event handler class names for event handlers available to the audit service.

"`caseInsensitiveFields`": array of configuration expression<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 safelist to specify which event fields appear in the logs. By default, only event fields that are safelisted are included in the audit event logs. For more information about safelisting, see Safelisting Audit Event Fields for the Logs.

"`field`": object, optional

This property specifies non-safelisted event fields to include in the logs, and safelisted 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 safelisted event fields in the logs.

"`includeIf`": array of configuration expression<strings>, optional:

A list of non-safelisted 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

IMPORTANT

Before you include non-safelisted 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 configuration expression<strings>, optional:

A list of safelisted 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"
    ]
  }
}
```

For an example route that excludes fields, see Exclude Safelisted Audit Event Fields From Logs.

"`eventHandlers`": array of Event Handler 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. IG supports the event handlers listed in AuditFramework.

"`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: `$HOME/.openig/audit-schemas` (Windows, `%appdata%\OpenIG\OpenIG\audit-schemas`)

For an example of how to configure custom audit events, see Record custom audit events.

The following example schema includes the mandatory fields, `_id`, `timestamp`, `transactionId`, and `eventName`, and an optional `customField`:

```
{
  "schema": {
    "$schema": "http://json-schema.org/draft-04/schema#",
    "id": "/",
    "type": "object",
    "properties": {
      "_id": {
        "type": "string"
      },
      "timestamp": {
        "type": "string"
      },
      "transactionId": {
        "type": "string"
      },
      "eventName": {
        "type": "string"
      },
      "customField": {
        "type": "string"
      }
    }
  }
}
```

"event-handlers": array of configuration objects, required

IMPORTANT

This property is deprecated. Use `eventHandlers` instead. For more information, refer to Deprecation.

The event handlers that deal with audit events.

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"
}
```

More information

NoOpAuditService

org.forgerock.audit.AuditService

CsvAuditEventHandler

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 or other applications that treat data as code.

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<string>,
      "delimiterChar": configuration expression<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>
    },
    "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": {
      "maxDiskSpaceToUse": configuration expression<number>,
      "maxNumberOfHistoryFiles": configuration expression<number>,
      "minFreeSpaceRequired": configuration expression<number>
    },
    "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.

Configuration

"name": configuration expression<string>, required

The name of the event handler.

"logDirectory": configuration expression<string>, required

The file system directory where this event handler writes log files.

When multiple AuditServices are defined in the deployment, prevent them from logging to the same audit logging file by setting different values for `logDirectory`.

"topics": array of configuration expression<strings>, required

One or more topics that this event handler intercepts. IG 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 IG configuration.

  To record custom audit events, configure AuditService in the heap, and refer to it from the route or subroutes. For an example of how to set up custom audit events, see Record custom audit events.

"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<string>, optional

A single character to quote CSV entries.

Default: "

"delimiterChar": configuration expression<string>, optional

A single character to delimit CSV entries.

Default: ,

"endOfLineSymbols": configuration expression<string>, optional

A character or characters to separate a line.

Default: System-dependent line separator defined for the JVM

"`buffering`": object, optional

Do not enable `buffering` when `security` is configured for tamper-evident logging.

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`": object, optional

When `security` is configured for tamper-evident logging, do not enable `buffering`.

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 an example, see Recording Access Audit Events in CSV.

"`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 an example, see Recording Access Audit Events in CSV.

"`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 an example, see Recording Access Audit Events in CSV.

*"**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.

*"**fileRotation**": object, optional*
File rotation settings for log files.

*"**rotationEnabled**": configuration expression<boolean>, optional*
A flag to enable rotation of 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.

*"**rotationTimes**": array of configuration expression<durations>, 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.

*"**`fileRetention`**": object, optional*

File retention settings for log files.

*"**`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.

Default: 0

*"**`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.

Default: 0

*"**`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.

Default: 5 seconds

Example

For information about how to record audit events in a CSV file, see Recording Access Audit Events in CSV.

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/secrets/audit-keystore",
    "password": "password",
    "signatureInterval": "10 seconds"
  }
}
```

More information

org.forgerock.audit.handlers.csv.CsvAuditEventHandler

ElasticsearchAuditEventHandler (deprecated)

IMPORTANT

This object is deprecated; use one of the following objects instead:

- SyslogAuditEventHandler
- JsonAuditEventHandler, with `elasticsearchCompatible` set to `true`

For more information, refer to Deprecation.

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.

Usage

Configure the ElasticsearchAuditEventHandler within an AuditService:

```
{
  "type": "AuditService",
  "config": {
    "config": {},
    "eventHandlers": [{
      "class":
"org.forgerock.audit.handlers.elasticsearch.ElasticsearchAuditEven
tHandler",
      "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.

IG searches first for a handler named `ElasticsearchClientHandler`. If not found, IG searches for a client handler named `AuditClientHandler`. If not found, IG 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": {}
}
```

Properties

"`name`": configuration expression<string>, required

The name of the event handler.

"`topics`": array of configuration expression<strings>, required

One or more topics that this event handler intercepts. IG 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 IG configuration.

  To record custom audit events, configure AuditService in the heap, and refer to it from the route or subroutes. For an example of how to set up custom audit events, see Record custom audit events.

"`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

"`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

Example

In the following example, an Elasticsearch audit event handler logs audit events for access. For an example of setting up and testing this configuration, see [maintenance-guide:].

```
{
  "name": "30-elasticsearch",
  "baseURI": "http://app.example.com:8081",
  "condition": "${find(request.uri.path, '^/home/elasticsearch-
audit')}",
  "heap": [
    {
      "name": "AuditService",
      "type": "AuditService",
      "config": {
        "eventHandlers": [
```

```
          {
            "class":
"org.forgerock.audit.handlers.elasticsearch.ElasticsearchAuditEven
tHandler",
            "config": {
              "name": "elasticsearch",
              "indexMapping": {
                "indexName": "audit"
              },
              "connection": {
                "host": "localhost",
                "port": 9200,
                "useSSL": false
              },
              "topics": [
                "access"
              ]
            }
          }
        ]
      }
    }
  ],
  "auditService": "AuditService",
  "handler": "ReverseProxyHandler"
}
```

More information

org.forgerock.audit.handlers.elasticsearch.ElasticsearchAuditEventHandler

JdbcAuditEventHandler

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.

To configure IG to use the database, add the database .jar file containing the Driver as follows:

- For IG in standalone mode, create the directory `$HOME/.openig/extra`, where `$HOME/.openig` is the instance directory, and add .jar files to the directory.
- For IG in web container mode, add .jar files to the web container classpath. For example, in Jetty use `/path/to/jetty/webapps/ROOT/WEB-INF/lib`.

- For IG in standalone mode, the JDBC handler library is in the `lib` directory.
- For IG in web container mode, the JDBC handler library is built into the IG .war file.

Unpack the library, then find the examples under the `db/` folder.

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>
      }
    ]
```

```
    }
}
```

Configuration

"name": configuration expression<string>, required
The name of the event handler.

"topics": array of configuration expression<strings>, required
One or more topics that this event handler intercepts. IG 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 IG configuration.

  To record custom audit events, configure AuditService in the heap, and refer to it from the route or subroutes. For an example of how to set up custom audit events, see Record custom audit events.

"databaseType": configuration expression<string>, required
The database type name.

Built-in support is provided for `oracle` , `mysql` , and `h2` .

"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.

"`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
When a JdbcDataSource object named `AuditService` is defined in the route heap. This configuration is not required.

Connection pool settings for sending messages to the database.

"`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`": <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.

Example

Examples including statements to create tables are provided in the JDBC handler library, `forgerock-audit-handler-jdbc-version.jar`.

For an example of using JdbcAuditEventHandler, see Recording Access Audit Events in a Database.

In the following example, IG events are logged to an h2 database:

```
{
  "name": "audit-jdbc",
  "baseURI": "http://app.example.com:8081",
  "condition": "${find(request.uri.path, '^/home/audit-jdbc')}",
  "heap": [
    {
      "name": "SystemAndEnvSecretStore-1",
      "type": "SystemAndEnvSecretStore"
    },
    {
      "name": "AuditDataSource",
      "type": "JdbcDataSource",
      "config": {
        "dataSourceClassName" : "org.h2.jdbcx.JdbcDataSource",
        "username"            : "sa",
        "passwordSecretId"    : "database.password",
        "secretsProvider"     : "SystemAndEnvSecretStore-1",
        "properties" : {
          "url"               : "jdbc:h2:tcp://localhost/~/test"
        }
      }
    },
    {
      "name": "AuditService",
      "type": "AuditService",
      "config": {
        "eventHandlers": [
          {
            "class":
"org.forgerock.audit.handlers.jdbc.JdbcAuditEventHandler",
            "config": {
              "databaseType": "h2",
              "name": "jdbc",
              "topics": [
                "access"
              ],
              "tableMappings": [
                {
```

```
                    "event": "access",
                    "table": "audit.auditaccess",
                    "fieldToColumn": {
                      "_id": "id",
                      "timestamp": "timestamp_",
                      "eventName": "eventname",
                      "transactionId": "transactionid",
                      "userId": "userid",
                      "trackingIds": "trackingids",
                      "server/ip": "server_ip",
                      "server/port": "server_port",
                      "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"
                    }
                  }
                ]
              }
            }
          ]
        }
      }
  ],
  "auditService": "AuditService",
  "handler": "ReverseProxyHandler"
}
```

More information

org.forgerock.audit.handlers.jdbc.JdbcAuditEventHandler

JmsAuditEventHandler

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 .jar files, to provide clients with connectivity, message storage, and message delivery functionality.

  Add the .jar files to the configuration as follows:

- For IG in standalone mode, create the directory `$HOME/.openig/extra`, where `$HOME/.openig` is the instance directory, and add .jar files to the directory.
- For IG in web container mode, add .jar files to the web container classpath. For example, in Jetty use `/path/to/jetty/webapps/ROOT/WEB-INF/lib`.
- 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.

Declare the configuration in an audit service, as described in AuditService.

Usage

```
{
  "name": string,
  "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.

Configuration

For a list of properties in the `"config"` object, see JMS Audit Event Handler in IDM's *Integrator's guide*.

"name": configuration expression<string>, required
The name of the event handler.

"topics": array of configuration expression<strings>, required
One or more topics that this event handler intercepts. IG 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 IG configuration.

  To record custom audit events, configure AuditService in the heap, and refer to it from the route or subroutes. For an example of how to set up custom audit events, see Record custom audit events.

"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 map values are evaluated as configuration expression<strings>.

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.

Example

In the following example, a JMS audit event handler delivers audit events in batches. The handler is configured to use the ActiveMQ JNDI message broker, on port 61616. For an example of setting up and testing this configuration, see Recording Access Audit Events in JMS.

```
{
  "name": "30-jms",
  "MyCapture" : "all",
  "baseURI": "http://app.example.com:8081",
  "condition" : "${request.uri.path ==
'/activemq_event_handler'}",
  "heap": [
    {
      "name": "AuditService",
      "type": "AuditService",
      "config": {
        "eventHandlers" : [
          {
            "class" :
"org.forgerock.audit.handlers.jms.JmsAuditEventHandler",
            "config" : {
              "name" : "jms",
              "topics": [ "access" ],
              "deliveryMode" : "NON_PERSISTENT",
              "sessionMode" : "AUTO",
              "jndi" : {
                "contextProperties" : {
                  "java.naming.factory.initial" :
"org.apache.activemq.jndi.ActiveMQInitialContextFactory",
                  "java.naming.provider.url" :
"tcp://am.example.com:61616",
                  "topic.audit" : "audit"
                },
```

```
                "topicName" : "audit",
                "connectionFactoryName" : "ConnectionFactory"
              }
            }
          }
        ],
        "config" : { }
      }
    }
  ],
  "auditService": "AuditService",
  "handler" : {
    "type" : "StaticResponseHandler",
    "config" : {
      "status" : 200,
      "headers" : {
        "Content-Type" : [ "text/plain; charset=UTF-8" ]
      },
      "entity" : "Message from audited route"
    }
  }
}
```

More information

org.forgerock.audit.handlers.jms.JmsAuditEventHandler

JsonAuditEventHandler

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.

Usage

```
{
  "name": string,
  "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>
        }
      }
    }]
  }
}
```

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<strings>, required

One or more topics that this event handler intercepts. IG 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 IG configuration.

  To record custom audit events, configure AuditService in the heap, and refer to it from the route or subroutes. For an example of how to set up custom audit events, see Record custom audit events.

"`logDirectory`": configuration expression<string>, required

The file system directory where this event handler writes log files.

When multiple AuditServices are defined in the deployment, prevent them from logging to the same audit logging file by setting different values for `logDirectory` .

`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.

"`rotationEnabled`": configuration expression<boolean>, optional

A flag to enable rotation of 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.

"`rotationTimes`": array of configuration expression<durations>, 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.

"`fileRetention`": object, optional
File retention settings for log files.

"`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.

Default: 0

"`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.

Default: 0

"`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.

Default: 5 seconds

"`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

"`maxSize`": configuration expression<number>, optional

The maximum number of event messages in the queue of buffered event messages.

Default: 10000

Example

For an example of setting up and testing this configuration, see Recording Access Audit Events in JSON.

More information

org.forgerock.audit.handlers.json.JsonAuditEventHandler

JsonStdoutAuditEventHandler

Logs events to JSON standard output (stdout).

Declare the configuration in an audit service, as described in AuditService.

Usage

```
{
  "name": string,
  "type": "AuditService",
  "config": {
    "config": {},
    "eventHandlers": [
    {
      "class":
"org.forgerock.audit.handlers.json.stdout.JsonStdoutAuditEventHand
ler",
      "config": {
        "name": configuration expression<string>,
```

```
        "topics": [ configuration expression<string>, ... ],
        "elasticsearchCompatible": configuration
expression<boolean>
      }
    }
  }
}
```

Configuration

"name": configuration expression<string>, required

The name of the event handler.

"topics": array of configuration expression<strings>, required

One or more topics that this event handler intercepts. IG 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 IG configuration.

  To record custom audit events, configure AuditService in the heap, and refer to it from the route or subroutes. For an example of how to set up custom audit events, see Record custom audit events.

elasticsearchCompatible: configuration expression<boolean>, optional

Set to `true` to enable compatibility with ElasticSearch JSON format. For more information, see the ElasticSearch documentation.

Default: `false`

Example

In the following example, a JsonStdoutAuditEventHandler logs audit events. For an example of setting up and testing this configuration, see Recording access audit events to standard output.

```
{
  "name": "30-jsonstdout",
  "baseURI": "http://app.example.com:8081",
  "condition": "${find(request.uri.path, '^/home/jsonstdout-
```

```
audit')}",
  "heap": [
    {
      "name": "AuditService",
      "type": "AuditService",
      "config": {
        "eventHandlers": [
          {
            "class":
"org.forgerock.audit.handlers.json.stdout.JsonStdoutAuditEventHand
ler",
            "config": {
              "name": "jsonstdout",
              "elasticsearchCompatible": false,
              "topics": [
                "access"
              ]
            }
          }
        ],
        "config": {}
      }
    }
  ],
  "auditService": "AuditService",
  "handler": "ReverseProxyHandler"
}
```

More information

org.forgerock.audit.handlers.json.stdout.JsonStdoutAuditEventHandler

NoOpAuditService

Provides an empty audit service to the top-level heap and its child routes. Use NoOpAuditService to prevent routes from using the parent audit service, when an AuditService is not explicitly defined.

For information about how to override the default audit service, see Default Audit Service.

Usage

```
{
  "name": "AuditService",
  "type": "NoOpAuditService"
 }
```

```
"auditService": "NoOpAuditService"
```

More information

AuditService

org.forgerock.audit.NoOpAuditService

SyslogAuditEventHandler

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.

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.

Configuration

"name": configuration expression<string>, required
The name of the event handler.

"topics": array of configuration expression<strings>, required
One or more topics that this event handler intercepts. IG 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 IG configuration.

  To record custom audit events, configure AuditService in the heap, and refer to it from the route or subroutes. For an example of how to set up custom audit events, see Record custom audit events.

"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<enumeration>, required

The Syslog facility to use for event messages. Set 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.

"`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.

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"
        }
      }
    ]
  }
}
```

More information

org.forgerock.audit.handlers.syslog.SyslogAuditEventHandler

SplunkAuditEventHandler (deprecated)

IMPORTANT

This object is deprecated; use SyslogAuditEventHandler or JsonAuditEventHandler instead. For more information, refer to Deprecation.

The Splunk audit event handler logs IG events to a Splunk system.

For an example of setting up and testing Splunk, see Recording Access Audit Events in Splunk.

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 Splunk 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.

IG searches first for a handler named `SplunkAuditEventHandler`. If not found, IG searches for a client handler named `AuditClientHandler`. If not found, IG 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": {}
}
```

Configuration

"name": configuration expression<string>, required

The name of the event handler.

"topics": array of configuration expression<strings>, required

One or more topics that this event handler intercepts. IG 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 IG configuration.

  To record custom audit events, configure AuditService in the heap, and refer to it from the route or subroutes. For an example of how to set up custom audit events, see Record custom audit events.

"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 IG 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 IG 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)

"authzToken": configuration expression<string>, required

The authorization token associated with the configured HTTP event collector.

Example

In the following example, IG events are logged to a Splunk system.

```
{
  "name": "30-splunk",
  "baseURI": "http://app.example.com:8081",
  "condition": "${find(request.uri.path, '^/home/splunk-audit')}",
  "heap": [
    {
      "name": "AuditService",
      "type": "AuditService",
      "config": {
        "eventHandlers": [
          {
            "class":
"org.forgerock.audit.handlers.splunk.SplunkAuditEventHandler",
            "config": {
              "name": "splunk",
              "enabled": true,
              "authzToken": "<splunk-authorization-token>",
              "connection": {
```

```
                "host": "localhost",
                "port": 8088,
                "useSSL": false
              },
              "topics": [
                "access"
              ],
              "buffering": {
                "maxSize": 10000,
                "maxBatchedEvents": 500,
                "writeInterval": "100 ms"
              }
            }
          }
        ]
      }
    }
  ],
  "auditService": "AuditService",
  "handler": "ReverseProxyHandler"
}
```

For an example of setting up and testing this configuration, see Recording Access Audit Events in Splunk.

More information

org.forgerock.audit.handlers.splunk.SplunkAuditEventHandler

Monitoring

The following sections describe monitoring endpoints exposed by IG, and the metrics available at the endpoints.

For information about how to set up and maintain monitoring, see Monitoring services.

Vert.x Metrics

Not supported in web container mode.

Vert.x metrics for HTTP clients, TCP clients, and servers are available by default at the Prometheus Scrape Endpoint and Common REST Monitoring Endpoint endpoints. Vert.x metrics provide low-level information about requests and responses, such as the

number of bytes, duration, the number of concurrent requests. The available metrics are based on those described in Vert.x core tools metrics.

For more information about Vert.x and IG, see the `vertx` object in AdminHttpApplication (`admin.json`), and Monitoring Vert.x Metrics.

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
}
```

Monitoring Endpoints

This section describes the monitoring endpoints exposed in IG.

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 website.

When IG is set up as described in the documentation, the endpoint is http://ig.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

For an example that queries the Prometheus Scrape Endpoint, see Monitor the Prometheus Scrape Endpoint.

Common REST Monitoring Endpoint

All ForgeRock products expose a monitoring endpoint where metrics are exposed as a JSON format monitoring resource.

When IG is set up as described in the documentation, the endpoint is `http://ig.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

For an example that queries the Common REST Monitoring Endpoint, see Monitor the Common REST Monitoring Endpoint.

Throttling policies

To protect applications from being overused by clients, use a ThrottlingFilter with one of the following policies to limit how many requests clients can make in a defined time:

MappedThrottlingPolicy

Maps different throttling rates to different groups of requests, according to the evaluation of `throttlingRateMapper`.

Usage

```
{
    "name": string,
    "type": "ThrottlingFilter",
    "config": {
        "requestGroupingPolicy": runtime expression<string>,
        "throttlingRatePolicy": {
            "type": "MappedThrottlingPolicy",
            "config": {
                "throttlingRateMapper": runtime
expression<string>,
                "throttlingRatesMapping": {
                    "mapping1": {
                        "numberOfRequests": configuration
expression<number>,
                        "duration": configuration
expression<duration>
```

```
                    },
                    "mapping2": {
                        "numberOfRequests": configuration
expression<number>,
                        "duration": configuration
expression<duration>
                    }
                },
                "defaultRate": {
                    "numberOfRequests": configuration
expression<number>,
                    "duration": configuration expression<duration>
                }
            }
        }
    }
}
```

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": configuration expression<integer>, required

The number of requests allowed through the filter in the time specified by `"duration"`.

"duration": configuration expression<duration>, required

A time interval during which the number of requests passing through the filter is counted.

"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`": configuration expression<integer>, required

The number of requests allowed through the filter in the time specified by `"duration"`.

"`duration`": configuration expression<duration>, required

A time interval during which the number of requests passing through the filter is counted.

Example of a Mapped Throttling Policy

In the following example, requests from users with different statuses are mapped to different throttling rates. For information about how to set up and test this example, see Configure Mapped Throttling.

```
{
  "name": "00-throttle-mapped",
  "baseURI": "http://app.example.com:8081",
  "condition": "${find(request.uri.path, '^/home/throttle-
mapped')}",
  "heap": [
    {
      "name": "SystemAndEnvSecretStore-1",
      "type": "SystemAndEnvSecretStore"
    },
    {
      "name": "AmService-1",
      "type": "AmService",
      "config": {
        "agent": {
          "username": "ig_agent",
          "passwordSecretId": "agent.secret.id"
        },
        "secretsProvider": "SystemAndEnvSecretStore-1",
        "url": "http://am.example.com:8088/openam/",
        "version": "7.2"
      }
    }
  ],
  "handler": {
    "type": "Chain",
    "config": {
      "filters": [
        {
          "name": "OAuth2ResourceServerFilter-1",
          "type": "OAuth2ResourceServerFilter",
```

```
          "config": {
            "scopes": [
              "mail",
              "employeenumber"
            ],
            "requireHttps": false,
            "realm": "OpenIG",
            "accessTokenResolver": {
              "name": "token-resolver-1",
              "type": "TokenIntrospectionAccessTokenResolver",
              "config": {
                "amService": "AmService-1",
                "providerHandler": {
                  "type": "Chain",
                  "config": {
                    "filters": [
                      {
                        "type":
"HttpBasicAuthenticationClientFilter",
                        "config": {
                          "username": "ig_agent",
                          "passwordSecretId": "agent.secret.id",
                          "secretsProvider":
"SystemAndEnvSecretStore-1"
                        }
                      }
                    ],
                    "handler": "ForgeRockClientHandler"
                  }
                }
              }
            }
          },
          {
            "name": "ThrottlingFilter-1",
            "type": "ThrottlingFilter",
            "config": {
              "requestGroupingPolicy":
"${contexts.oauth2.accessToken.info.mail}",
              "throttlingRatePolicy": {
                "name": "MappedPolicy",
                "type": "MappedThrottlingPolicy",
                "config": {
                  "throttlingRateMapper":
```

```
"${contexts.oauth2.accessToken.info.status}",
                "throttlingRatesMapping": {
                  "gold": {
                    "numberOfRequests": 6,
                    "duration": "10 s"
                  },
                  "silver": {
                    "numberOfRequests": 3,
                    "duration": "10 s"
                  },
                  "bronze": {
                    "numberOfRequests": 1,
                    "duration": "10 s"
                  }
                },
                "defaultRate": {
                  "numberOfRequests": 1,
                  "duration": "10 s"
                }
              }
            }
          }
        ],
        "handler": "ReverseProxyHandler"
      }
    }
}
```

More information

org.forgerock.openig.filter.throttling.MappedThrottlingPolicyHeaplet

ScriptableThrottlingPolicy

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.

For information about script properties, available global objects, and automatically imported classes, see Scripts.

- For an example of how to create a ScriptableThrottlingPolicy in Studio, see Configure scriptable throttling.

Usage

```
{
  "type": "ThrottlingFilter",
  "config": {
    "requestGroupingPolicy": runtime expression<string>,
    "throttlingRatePolicy": {
      "name": string,
      "type": "ScriptableThrottlingPolicy",
      "config": {
        "type": configuration expression<string>,
        "file": configuration expression<string>, // Use either
"file"
        "source": [ string, ... ],                // or "source",
but not both
        "args": map,
        "clientHandler": Handler reference
        }
    }
  }
}
```

Properties

For information about properties for ScriptableThrottlingPolicy, see Scripts.

Example of a scriptable throttling policy

In the following example, the `DefaultRateThrottlingPolicy` delegates the management of throttling to the scriptable throttling policy. For information about how to set up and test this example, see Configure scriptable throttling.

```
{
  "name": "00-throttle-scriptable",
  "baseURI": "http://app.example.com:8081",
  "condition": "${find(request.uri.path, '^/home/throttle-
scriptable')}",
  "heap": [
```

```
    {
      "name": "SystemAndEnvSecretStore-1",
      "type": "SystemAndEnvSecretStore"
    },
    {
      "name": "AmService-1",
      "type": "AmService",
      "config": {
        "agent": {
          "username": "ig_agent",
          "passwordSecretId": "agent.secret.id"
        },
        "secretsProvider": "SystemAndEnvSecretStore-1",
        "url": "http://am.example.com:8088/openam/",
        "version": "7.2"
      }
    }
  ],
  "handler": {
    "type": "Chain",
    "config": {
      "filters": [
        {
          "name": "OAuth2ResourceServerFilter-1",
          "type": "OAuth2ResourceServerFilter",
          "config": {
            "scopes": [
              "mail",
              "employeenumber"
            ],
            "requireHttps": false,
            "realm": "OpenIG",
            "accessTokenResolver": {
              "name": "token-resolver-1",
              "type": "TokenIntrospectionAccessTokenResolver",
              "config": {
                "amService": "AmService-1",
                "providerHandler": {
                  "type": "Chain",
                  "config": {
                    "filters": [
                      {
                        "type":
"HttpBasicAuthenticationClientFilter",
                        "config": {
```

```
                        "username": "ig_agent",
                        "passwordSecretId": "agent.secret.id",
                        "secretsProvider":
"SystemAndEnvSecretStore-1"
                      }
                    }
                  ],
                  "handler": "ForgeRockClientHandler"
                }
              }
            }
          }
        },
        {
          "name": "ThrottlingFilter-1",
          "type": "ThrottlingFilter",
          "config": {
            "requestGroupingPolicy":
"${contexts.oauth2.accessToken.info.mail}",
            "throttlingRatePolicy": {
              "type": "DefaultRateThrottlingPolicy",
              "config": {
                "delegateThrottlingRatePolicy": {
                  "name": "ScriptedPolicy",
                  "type": "ScriptableThrottlingPolicy",
                  "config": {
                    "type": "application/x-groovy",
                    "source": [
                      "if (contexts.oauth2.accessToken.info.status
== status) {",
                      "   return new ThrottlingRate(rate,
duration)",
                      "} else {",
                      "   return null",
                      "}"
                    ],
                    "args": {
                      "status": "gold",
                      "rate": 6,
                      "duration": "10 seconds"
                    }
                  }
                },
                "defaultRate": {
```

```
                    "numberOfRequests": 1,
                    "duration": "10 s"
                  }
                }
              }
            }
          }
        ],
        "handler": "ReverseProxyHandler"
      }
    }
}
```

More information

org.forgerock.openig.filter.throttling.ScriptableThrottlingPolicy.Heaplet

DefaultRateThrottlingPolicy

Provides a default throttling rate if the delegating throttling policy returns `null`.

Usage

```
{
   "name": string,
   "type": "ThrottlingFilter",
   "config": {
       "requestGroupingPolicy": expression,
       "throttlingRatePolicy": {
           "type": "DefaultRateThrottlingPolicy",
           "config": {
               "delegateThrottlingRatePolicy":
ThrottlingRatePolicy reference,
               "defaultRate": {
                   "numberOfRequests": configuration
expression<number>,
                   "duration": configuration expression<duration>
               }
           }
       }
   }
}
```

Properties

"delegateThrottlingRatePolicy": ThrottlingRatePolicy 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": configuration expression<integer>, required

The number of requests allowed through the filter in the time specified by `"duration"`.

"duration": configuration expression<duration>, required

A time interval during which the number of requests passing through the filter is counted.

Example

For an example of how this policy is used, see Example of a Scriptable Throttling Policy.

More information

org.forgerock.openig.filter.throttling.DefaultRateThrottlingPolicyHeaplet

Miscellaneous configuration objects

The following objects can be defined in the configuration:

AmService

Holds information about the configuration of an instance of AM. The AmService is available to IG filters that communicate with that instance.

When IG uses an AmService, IG is positioned as the client of the service. By default, IG is subscribed to Websocket notifications from AM, and the WebSocket connection can be secured by ClientTlsOptions.

Usage

```
{
  "name": string,
   "type": "AmService",
   "config": {
     "agent": object,
     "secretsProvider": SecretsProvider reference,
     "notifications": object,
     "realm": configuration expression<string>,
     "amHandler": Handler reference,
     "sessionCache": object,
     "sessionIdleRefresh": object,
     "sessionProperties": [ configuration expression<string>, ...
],
     "ssoTokenHeader": configuration expression<string>,
     "url": configuration expression<url>,
     "version": configuration expression<string>
   }
}
```

Properties

"agent": object, required

The credentials of the IG 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.

"password": configuration expression<string>, required

IMPORTANT

This property is deprecated and is not considered secure. Use `passwordSecretId` instead. For more information, refer to Deprecation.

The agent password.

"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 Default secrets object.

"`realm`": configuration expression<string>, optional

The AM realm in which the IG agent is created.

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.

AmService does not use `amHandler` to subscribe to WebSocket notifications from AM. To subscribe to WebSocket notifications from AM, configure a ClientTlsOptions object in the heap, and refer to it from the `amHandler` object and the `notifications` subproperty `tls`.

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 and ClientHandler.

"`notifications`": object, optional

Configure a WebSocket notification service, to subscribe to Websocket notifications from AM.

To subscribe to WebSocket notifications from AM, configure a ClientTlsOptions object in the heap, and refer to it from the `amHandler` object and the `notifications` subproperty `tls`. Alternatively, use `proxyOptions` to share a proxy configuration between the `amHandler` and the notification service.

For information, see WebSocket Notifications.

```
{
  "notifications": {
    "enabled": configuration expression<boolean>,
    "initialConnectionAttempts": configuration
expression<number>,
    "reconnectDelay": configuration expression<duration>,
    "tls": ClientTlsOptions reference,
    "proxyOptions": ProxyOptions reference,
  }
}
```

`enabled`: configuration expression<boolean>, optional

A flag to enable WebSocket notifications. Set to `false` to disable WebSocket notifications.

Default: `true`

`"initialConnectionAttempts"`: configuration expression<number>, optional

The maximum number of times IG attempts to open a WebSocket connection before failing to deploy a route. For no limit, set this property to `-1`.

If the WebSocket connection fails **after** it has been opened and the route is deployed, IG attempts to reconnect to it an unlimited number of times.

Default: `5`

`reconnectDelay`: configuration expression<duration>, optional

The time between attempts to re-establish a lost WebSocket connection.

When a WebSocket connection is lost, IG waits for this delay and then attempts to re-establish the connection. If subsequent attempts fail, IG waits and tries again an unlimited number of times.

Default: `5 seconds`

`heartbeatInterval`: configuration expression<duration>, optional

The interval at which the AmService issues a heartbeat on WebSocket connections. When activity on the connection is low, the heartbeat prevents middleware or policies situated between IG and AM from closing the connection for timeout.

Set to zero or unlimited to disable heartbeats.

Default: `1 minute`

`tls`: ClientTlsOptions reference, optional

IMPORTANT

IMPORTANT

Use of a TlsOptions reference is deprecated; use ClientTlsOptions instead. For more information, refer to Deprecation.

Configure options for WebSocket connections to TLS-protected endpoints. Define a ClientTlsOptions object inline or in the heap.

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

"`proxyOptions`": ProxyOptions 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.

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

Default: A heap object named `ProxyOptions`.

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

The URL of the AM service. When AM is running locally, this value could be `https://am.example.com/openam`. When AM is running in the ForgeRock Identity Cloud, this value could be `https://myTenant.forgeblocks.com/am`.

"`sessionCache`": object, optional

In AM, if the realm includes a customized session property safelist, include `AMCtxId` in the list of properties. The customized session property safelist overrides the global session property safelist.

Enable and configure caching of session information from AM, based on *Caffeine*. For more information, see the GitHub entry, Caffeine.

When `sessionCache` is enabled, IG 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 IG instances.

When `sessionCache` is disabled, IG must ask AM to verify the token for each request.

IG 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 IG evicts session info entries from the cache, the next time the token is presented, IG 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, IG evicts a cached token almost as soon as it is revoked on AM, and in this way stays synchronized with AM. Subsequent requests to IG 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 IG 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*>, 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` from SessionInfo, `maxSessionExpirationTime` is used.

Default:

- When `sessionIdleRefresh` is set, idle timeout of AM minus 30 seconds.
- When `sessionIdleRefresh` is not set, `maxSessionExpirationTime`, from SessionInfo.

`onNotificationDisconnection`: configuration expression<enumeration>, optional

The strategy to manage the cache when the WebSocket notification service is disconnected, and IG receives no notifications for AM events. If the cache is not cleared it can become outdated, and IG can allow requests on revoked sessions or tokens.

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`

"sessionIdleRefresh": object, optional

(From AM 6.5.3.) Enable and configure periodic refresh of idle sessions. Use this property when AM is using CTS-based sessions. AM does not monitor idle time for client-side sessions, and so refresh requests are ignored.

When the SingleSignOnFilter is used for authentication with AM, after a time, AM can view the session as idle even though the user continues to interact with IG. The user session eventually times out and the user must re-authenticate.

When the SingleSignOnFilter filter is used with the PolicyEnforcementFilter, the session is refreshed each time IG requests a policy decision from AM. The session is less likely to become idle, and this property less required.

```
{
  "sessionIdleRefresh": {
    "enabled": configuration expression<boolean>,
    "interval": configuration expression<duration>
  }
}
```

enabled: configuration expression<boolean>, optional

Enable refresh of idle sessions.

Default: `false`

interval: configuration expression<duration>, optional

Duration to wait after a session becomes idle before requesting a session refresh.

Consider setting the refresh interval in line with the latest access time update frequency of AM. For example, if IG requests a refresh every 60 seconds, but the update frequency of AM is 5 minutes, AM ignores most of the IG requests.

IMPORTANT

Each session refresh must be reflected in the AM core token service. Setting the interval to a duration lower than one minute can adversely impact AM performance.

Default: `5 minutes`

"sessionProperties": array of configuration expression<strings>, optional

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 header name or cookie name where this AM server expects to find SSO tokens.

If a value for `ssoTokenHeader` is provided, IG uses that value. Otherwise, IG queries the AM `/serverinfo/*` endpoint for the header or cookie name.

Default: Empty. IG queries AM for the cookie name.

"`version`": configuration expression<string>, optional

The version number of the AM server. IG uses the AM version to establish endpoints for its interaction with AM.

The AM version is derived as follows, in order of precedence:

- Discovered value: AmService discovers the AM version. If `version` is configured with a different value, AmService ignores the value of `version` and issues a warning.
- Value in `version`: AmService cannot discover the AM version, and `version` is configured.
- Default value of AM 6: AmService cannot discover the AM version, and `version` is not configured.

If you use a feature that is supported only in a higher AM version than discovered or specifed, a message can be logged or an error thrown. For example, if `sessionIdleRefresh` is enabled but the AM version is below the required AM 6.5.3, an error like the following is logged:

```
sessionIdleRefresh is only supported with AM version 6.5.3 and
above,
the configured AM version is 6
```

Default: AM 6.

More information

org.forgerock.openig.tools.am.AmService

ClientRegistration

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.

Usage

```
{
  "name": string,
  "type": "ClientRegistration",
  "config": {
    "clientId": configuration expression<string>,
    "issuer": Issuer reference,
    "scopes": [ configuration expression<string>, ...],
    "registrationHandler": Handler reference,
    "authenticatedRegistrationHandler": Handler reference,
    "clientSecret": configuration expression<string>, //deprecated
    "clientSecretId": configuration expression<secret-id>,
//deprecated
    "jwtExpirationTimeout": duration, //deprecated
    "keystore": reference, //deprecated
    "privateKeyJwtAlias": string, //deprecated
    "privateKeyJwtPassword": string, //deprecated
    "privateKeyJwtSecretId": configuration expression<secret-id>,
//deprecated
    "secretsProvider": SecretsProvider reference, //deprecated
    "tokenEndpointAuthMethod": enumeration, //deprecated
    "tokenEndpointAuthSigningAlg": string //deprecated
  }
}
```

Properties

"name": string, required

IMPORTANT

The use of the `name` property to identify a client that is registering with the authorization server is deprecated; use `clientId` instead. For more information, refer to the Deprecated section of the *Release Notes*.

"clientId": configuration expression<string>, required

The `client_id` obtained when registering with the authorization server. See also Expressions.

"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.

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

Array of scope strings to present to the user for approval, and include in tokens so that protected resources can make decisions about access.

Default: Empty

"`registrationHandler`": Handler reference, optional

HTTP client handler to invoke during client registration, to access endpoints that do not require client authentication. 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: ClientHandler.

"`authenticatedRegistrationHandler`": Handler reference, optional

HTTP client handler to invoke during client registration, to access endpoints that require client authentication. Configure this property as a Chain, using one of the following filters for client authentication:

- ClientSecretBasicAuthenticationFilter
- ClientSecretPostAuthenticationFilter
- EncryptedPrivateKeyJwtClientAuthenticationFilter
- PrivateKeyJwtClientAuthenticationFilter

```
{
  "name": "AuthenticatedRegistrationHandler",
  "type": "Chain",
  "config": {
    "handler": "ForgeRockClientHandler",
    "filters": [
      {
        "type": "ClientSecretBasicAuthenticationFilter",
        "config": {
          "clientId": "service-client",
          "clientSecretId": "client.secret.id",
          "secretsProvider" : "SystemAndEnvSecretStore-1",
        }
      }
    ]
```

```
    }
}
```

Default: `registrationHandler` with no authentication filter.

"`clientSecret`": _configuration expression<string>, required if `tokenEndpointAuthMethod` is `client_secret_basic` or `client_secret_post`

IMPORTANT

This property is deprecated; use `authenticatedRegistrationHandler` instead. For more information, refer to Deprecation.

The secret required to authenticate the client to the authorization server.

"`clientSecretId`": _configuration expression<secret-id>, required if `tokenEndpointAuthMethod` is `client_secret_basic` or `client_secret_post`

IMPORTANT

This property is deprecated; use `authenticatedRegistrationHandler` instead. For more information, refer to Deprecation.

The secret ID of the client secret required to authenticate the client to the authorization server.

"`jwtExpirationTimeout`": duration, optional

IMPORTANT

This property is deprecated; use `authenticatedRegistrationHandler` instead. For more information, refer to Deprecation.

When `private_key_jwt` is used for authentication, this property specifies the duration for which the JWT is valid.

Default: 1 minute

"`keystore`": reference, required if `private_key_jwt` is used

IMPORTANT

The use of this property is deprecated; use `authenticatedRegistrationHandler` instead. For more information, refer to Deprecation.

The Java KeyStore containing the private key that is used to sign the JWT.

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

"`privateKeyJwtAlias`": string, required if private_key_jwt is used

IMPORTANT

The use of this property is deprecated; use `authenticatedRegistrationHandler` instead. For more information, refer to Deprecation.

Name of the private key contained in the KeyStore.

"`privateKeyJwtPassword`": string, required if private_key_jwt is used

IMPORTANT

The use of this property is deprecated; use `authenticatedRegistrationHandler` instead. For more information, refer to Deprecation.

Password to access the private key contained in the KeyStore.

"`privateKeyJwtSecretId`": configuration expression<secret-id>, required when `private_key_jwt` *is used for client authentication*

IMPORTANT

This property is deprecated; use `authenticatedRegistrationHandler` instead. For more information, refer to Deprecation.

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

"`secretsProvider`": SecretsProvider reference, optional

IMPORTANT

This property is deprecated; use `authenticatedRegistrationHandler` instead. For more information, refer to Deprecation.

The SecretsProvider object to query for the client secret. For more information, see SecretsProvider.

Default: The route's default secret service. For more information, refer to Default secrets object.

"`tokenEndpointAuthMethod`": enumeration, optional

IMPORTANT

This property is deprecated; use `authenticatedRegistrationHandler` instead. For more information, refer to Deprecation.

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. IG 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

IMPORTANT

This property is deprecated; use `authenticatedRegistrationHandler` instead. For more information, refer to Deprecation.

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`

Example

See Use AM as a single OpenID Connect provider.

More information

org.forgerock.openig.filter.oauth2.client.ClientRegistration

Issuer, AuthorizationCodeOAuth2ClientFilter

The OAuth 2.0 Authorization Framework

The OAuth 2.0 Authorization Framework: Bearer Token Usage

OpenID Connect

ClientTlsOptions

Configures connections to the TLS-protected endpoint of servers, when IG is client-side.

When IG is *client-side*, IG sends requests to a proxied application, or requests services from a third-party application. IG is acting as a client of the application, and the application is acting as a server.

Use ClientTlsOptions in ClientHandler, ReverseProxyHandler, and AmService.

Usage

```
{
  "name": string,
  "type": "ClientTlsOptions",
  "config": {
    "keyManager": [ KeyManager reference, ...],
    "trustManager": [ TrustManager reference, ...],
    "sslCipherSuites": [ configuration expression<string>, ...],
    "sslContextAlgorithm": configuration expression<string>,
    "sslEnabledProtocols": [ configuration expression<string>,
...],
    "alpn": object,
    "hostnameVerifier": configuration expression<enumeration>
  }
}
```

Properties

"keyManager": array of KeyManager references references, optional

One or more KeyManager objects to serve the same secret key and certificate pair for TLS connections to all server names in the deployment. Key managers are used to prove the identity of the local peer during TLS handshake, as follows:

- When ServerTlsOptions is used in an HTTPS connector configuration (server-side), the KeyManagers to which ServerTlsOptions refers are used to prove this

IG's identity to the remote peer (client-side). This is the usual TLS configuration setting (without mTLS).

- When ClientTlsOptions is used in a ClientHandler or ReverseProxyHandler configuration (client-side), the KeyManagers to which ClientTlsOptions refers are used to prove this IG's identity to the remote peer (server-side). This configuration is used in mTLS scenarios.

  Provide the name of one or more of the following objects defined in the heap, or configure one or more of the following objects inline:

  - KeyManager
  - SecretsKeyManager

Default: None

"`sslCipherSuites`": array of configuration expression<strings>, 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). 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 (JVM).

Default: `TLS`

"`sslEnabledProtocols`": array of configuration expression<strings>, 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). For Oracle Java, see the list of Additional JSSE Standard Names.

Follow these protocol recommendations:

- Use TLS 1.3 when it is supported by available libraries, otherwise use TLS 1.2.
- If TLS 1.1 or TLS 1.0 is required for backwards compatibility, use it only with express approval from enterprise security.
- Do use deprecated versions SSL 3 or SSL 2.

Default: TLS 1.3, TLS 1.2

"`trustManager`": array of TrustManager references, optional

One or more of the following TrustManager objects to manage IG's public key certificates:

- TrustManager
- SecretsTrustManager
- TrustAllManager

Trust managers verify the identity of a peer by using certificates, as follows:

- When ServerTlsOptions is used in an HTTPS connector configuration (server-side), the TrustManagers to which ServerTlsOptions refers are used to verify the remote peer's identity (client-side). This configuration is used in mTLS scenarios.
- When ClientTlsOptions is used in a ClientHandler or a ReverseProxyHandler configuration (client-side), the TrustManager to which ClientTlsOptions refers are used to verify the remote peer's identity (server-side). This is the usual TLS configuration setting (without mTLS).

If `trustManager` is not configured, IG uses the default Java truststore to verify the remote peer's identity. The default Java truststore depends on the Java environment. For example, `$JAVA_HOME/lib/security/cacerts`.

Default: None

"`alpn`": object, optional

Not supported in web container mode.

A flag to enable the Application-Layer Protocol Negotiation (ALPN) extension for TLS connections.

```
{
  "alpn": {
    "enabled": configuration expression<boolean>
  }
}
```

`enabled`: configuration expression<boolean>, optional

- `true`: Enable ALPN. Required for HTTP/2 connections over TLS
- `false`: Disable ALPN.

Default: `true`

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

The method to handle hostname verification for outgoing SSL connections.

For backward compatibility, when a ClientHandler or ReverseProxyHandler includes the deprecated `"hostnameVerifier": "ALLOW_ALL"` configuration, it takes

precedence over this property. A deprecation warning is written to the logs.

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.

  If the SSL endpoint uses a raw IP address rather than a fully-qualified hostname, you must configure this property as `ALLOW_ALL` .

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

  CAUTION

  The `ALLOW_ALL` setting allows a certificate issued for one company to be accepted as a valid certificate for another company.

- `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`

Example

```
{
  "tls": {
    "type": "ClientTlsOptions",
    "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"
            }
          }
        }
      }
    }
  }
}
```

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.

Usage

```
{
  "filter or handler": {
  "type": "Delegate",
  [decorator reference, ...],
    "config": {
      "delegate": object
    }
  }
}
```

Example

For an example of how to delegate tasks to `ForgeRockClientHandler`, and capture IG's interaction with AM, see Decorating IG's Interactions With AM.

More information

org.forgerock.openig.decoration.DelegateHeaplet

JwtSession

Configures settings for stateless sessions.

Session information is serialized as a secure JWT, that is encrypted and signed, and optionally compressed. The resulting JWT string is placed in a cookie, which contains 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, IG 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. 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. To reduce the JWT cookie size:
  - Use the default `true` value for the JwtSession property `useCompression`, to compress the session JWT before it is placed 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 IG 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.

For information about IG sessions, see Sessions.

Usage

```
{
  "name": string,
  "type": "JwtSession",
  "config": {
    "authenticatedEncryptionSecretId": configuration
expression<secret-id>,
    "encryptionMethod": configuration expression<string>,
    "cookie": object,
    "sessionTimeout": configuration expression<duration>,
    "persistentCookie": configuration expression<boolean>,
    "secretsProvider": SecretsProvider reference,
    "skewAllowance": configuration expression<duration>,
    "useCompression": configuration expression<boolean>,
    "encryptionSecretId": configuration expression<secret-id>,
//deprecated
    "signatureSecretId": configuration expression<secret-id>,
//deprecated
    "keystore": KeyStore reference, //deprecated
    "alias": string, //deprecated
    "password": configuration expression, //deprecated
    "sharedSecret": string, //deprecated
    "cookieName": string,  //deprecated
    "cookieDomain": string  //deprecated
  }
}
```

Properties

"authenticatedEncryptionSecretId": configuration expression<secret-id>, optional

The secret ID of the encryption key used to perform authenticated encryption on a JWT. Authenticated encryption encrypts data and then signs it with HMAC, in a single step.

Authenticated encryption is achieved with a symmetric encryption key. Therefore, the secret must refer to a symmetric key.

For more information, see RFC 5116.

Default: IG generates a default symmetric key for authenticated encryption. Consequently, IG instances cannot share the JWT session.

"`encryptionMethod`": configuration expression<string>, optional
The algorithm to use for authenticated encryption. For information about allowed encryption algorithms, see RFC 7518: "enc" (Encryption Algorithm) Header Parameter Values for JWE.

Default: A256GCM

"`cookie`": object, optional
The configuration of the cookie used to store the encrypted JWT.

Default: The cookie is treated as a host-based 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

Options to manage the circumstances in which a cookie is sent to the server. Use one of the following values to reduce the risk of CSRF attacks:

- `STRICT`: Send the cookie only if the request was initiated from the cookie domain. Not case-sensitive.

  Use this value to reduce the risk of cross-site request forgery (CSRF) attacks.
- `LAX`: Send the cookie only with GET requests in a first-party context, where the URL in the address bar matches the cookie domain. Not case-sensitive.

  Use this value to reduce the risk of cross-site request forgery (CSRF) attacks.
- `NONE`: Send the cookie whenever a request is made to the cookie domain. Not case-sensitive.

  With this setting, consider setting `secure` to `true` to prevent browsers from rejecting the cookie. For more information, see SameSite cookies.

Default: `LAX`

"`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`": configuration expression<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, IG truncates the duration to 3650 days.

Default: 30 minutes

"`persistentCookie`": configuration expression<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. IG 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 Default secrets object.

"`skewAllowance`": configuration expression<duration>, optional

The duration 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 IG clock.
- A JWT with an `exp` 13:00 is expired after 13:02 on the IG clock.

Default: `zero`

"`useCompression`": configuration expression boolean, optional

A flag to compress the session JWT before it is placed in a cookie. When a session stores large items, such as tokens, use the default value `true` to reduce the size of the cookie.

Default: `true`

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

IMPORTANT

The use of this property is deprecated; use `authenticatedEncryptionSecretId` and `encryptionMethod` instead. For more information, refer to Deprecation.

The secret ID of the encryption key used to encrypt the JWT.

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

IMPORTANT

IMPORTANT

The use of this property is deprecated; use `authenticatedEncryptionSecretId` and `encryptionMethod` instead. For more information, refer to Deprecation.

The secret ID of the JWT signature required to sign and verify the JWTs. HMAC-SHA-256 is used to sign JWTs.

The value of this attribute must be:

- Base64-encoded.
- At least 32 bytes (32 characters)/256 bits long after base64 decoding. If the provided key is too short, an error message is created.

Default: Random data is generated as the key, and the IG instance can verify only the sessions it has created.

"`keystore`": KeyStore reference, optional

IMPORTANT

The use of this property is deprecated; use `authenticatedEncryptionSecretId` and `encryptionMethod` instead. For more information, refer to Deprecation.

The keystore holding the key pair with the private key used to encrypt the JWT.

Provide either the name of the KeyStore object defined in the heap, or the inline KeyStore configuration object inline.

Default: When no keystore is specified, IG generates a unique key pair, and stores the key pair in memory. With JWTs encrypted using a unique key pair generated at runtime, IG cannot decrypt the JWTs after a restart, nor can it decrypt such JWTs encrypted by another IG server.

"`alias`": string, required when keystore is used

IMPORTANT

The use of this property is deprecated; use `authenticatedEncryptionSecretId` and `encryptionMethod` instead. For more information, refer to Deprecation.

Alias for the private key.

"`password`": configuration expression, required when keystore is used

IMPORTANT

IMPORTANT

The use of this property is deprecated; use `authenticatedEncryptionSecretId` and `encryptionMethod` instead. For more information, refer to Deprecation.

The password to read the private key from the keystore.

"`sharedSecret`": string, optional

IMPORTANT

The use of this property is deprecated; use `authenticatedEncryptionSecretId` and `encryptionMethod` instead. For more information, refer to Deprecation.

Specifies the key used to sign and verify the JWTs.

This attribute is expected to be base-64 encoded. The minimum key size after base-64 decoding is 32 bytes/256 bits (HMAC-SHA-256 is used to sign JWTs). If the provided key is too short, an error message is created.

If this attribute is not specified, random data is generated as the key, and the IG instance can verify only the sessions it has created.

"`cookieName`": string, optional

IMPORTANT

The use of this property is deprecated; use `cookie` instead. For more information, refer to Deprecation.

The name of the JWT cookie stored on the user-agent.

Default: openig-jwt-session

"`cookieDomain`" string, optional

IMPORTANT

The use of this property is deprecated; use `cookie` instead. For more information, refer to Deprecation.

The name of the 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.

Default: The domain is not specified. The JWT cookie can be accessed only from the host where the cookie was created.

Example

For information about configuring a JwtSession with authenticated encryption, see Encrypt JWT sessions.

For information about managing multiple instances of IG in the same deployment, see the Installation guide.

More information

For information about IG sessions, see Sessions.

org.forgerock.openig.session.jwt.JwtSessionManager

KeyManager

The configuration of a Java Secure Socket Extension KeyManager to manage private keys for IG. The configuration references the keystore that holds the keys.

When IG acts as a server, it uses a KeyManager to prove its identity to the client. When IG acts as a client, it uses a KeyManager to prove its identity to the server.

Usage

```
{
  "name": string,
  "type": "KeyManager",
  "config": {
    "keystore": KeyStore reference,
    "passwordSecretId": configuration expression<secret-id>,
    "alg": configuration expression<string>,
    "secretsProvider": SecretsProvider reference,
    "password": string //deprecated
  }
}
```

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` in web container mode, 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.

"`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 Default secrets object.

"`password`": string, required

IMPORTANT

The use of this property is deprecated; use `passwordSecretId` instead. For more information, refer to Deprecation.

The password to read private keys from the keystore.

Example

The following example configures a KeyManager that depends on a KeyStore configuration. The KeyManager and KeyStore passwords are provided by Java system properties or environment variables, and retrieved by the SystemAndEnvSecretStore. By default, the password values must be base64-encoded.

```
{
  "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"
  }
}
```

More information

org.forgerock.openig.security.KeyManagerHeaplet

JSSE Reference guide, KeyStore, TrustManager

KeyStore

The configuration for a Java KeyStore, which stores cryptographic private keys and public key certificates.

Usage

```
{
  "name": name,
  "type": "KeyStore",
  "config": {
    "url": configuration expression<url>,
    "passwordSecretId": configuration expression<secret-id>,
    "type": configuration expression<string>,
    "secretsProvider": SecretsProvider reference,
    "password": string //deprecated
  }
}
```

Properties

"url": configuration expression<url>, 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.

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 secret store type.

"`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 Default secrets object.

"`password`": string, optional

IMPORTANT

The use of this property is deprecated; use `passwordSecretId` instead. For more information, refer to Deprecation.

The password to read private keys from the keystore.

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.

Example

The following example configures a KeyStore that references the Java KeyStore file `$HOME/keystore.jks`. The KeyStore password is provided by a Java system property or environment variable, and retrieved by the SystemAndEnvSecretStore. By default, the password value must be base64-encoded.

```
{
  "name": "MyKeyStore",
  "type": "KeyStore",
  "config": {
    "url": "file://${env['HOME']}/keystore.jks",
    "passwordSecretId": "keystore.secret.id",
    "secretsProvider": "SystemAndEnvSecretStore"
  }
}
```

More information

org.forgerock.openig.security.KeyStoreHeaplet

JSSE Reference guide, KeyManager, TrustManager

Issuer

Describes an OAuth 2.0 Authorization Server or an OpenID Provider that IG can use as a OAuth 2.0 client or OpenID Connect relying party.

An Issuer is generally referenced from a ClientRegistration, described in ClientRegistration.

Usage

```
{
  "name": string,
  "type": "Issuer",
  "config": {
    "wellKnownEndpoint": configuration expression<url>,
    "authorizeEndpoint": configuration expression<url>,
    "registrationEndpoint": configuration expression<url>,
    "tokenEndpoint": configuration expression<url>,
    "userInfoEndpoint": configuration expression<url>,
    "issuerHandler": Handler reference,
    "issuerRepository": Issuer repository reference,
    "supportedDomains": [ pattern, ... ]
  }
}
```

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`": configuration expression<url>, required unless authorizeEndpoint and tokenEndpoint are specified

The URL to the well-known configuration resource as described in OpenID Connect 1.0 Discovery.

"`authorizeEndpoint`": configuration expression<url>, required unless obtained through wellKnownEndpoint

The URL to the provider's OAuth 2.0 authorization endpoint.

"`registrationEndpoint`": configuration expression<url>, optional

The URL to the provider's OpenID Connect dynamic registration endpoint.

"`tokenEndpoint`": configuration expression<url>, required unless obtained through wellKnownEndpoint

The URL to the provider's OAuth 2.0 token endpoint.

"`userInfoEndpoint`": configuration expression<url>, optional

The URL to the provider's OpenID Connect UserInfo endpoint.

Default: no UserInfo is obtained from the provider.

"`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: IG 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 IG 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

One or more domain patterns to match domain names that are 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. IG can discover the issuer in this way. As a shortcut IG can also use supported domains lists to find issuers already described in the IG configuration.

To use this shortcut, IG 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. IG 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.

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://am.example.com:8443/openam/oauth2/authorize",
        "registration_endpoint":

"https://am.example.com:8443/openam/oauth2/connect/register",
        "tokenEndpoint":

"https://am.example.com:8443/openam/oauth2/access_token",
        "userInfoEndpoint":
          "https://am.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" ]
    }
}
```

More information

org.forgerock.openig.filter.oauth2.client.Issuer

IssuerRepository

Stores OAuth 2 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.

Usage

```
{
   "name": string,
   "type": "IssuerRepository",
   "config": {
     "issuerHandler": Handler reference
   }
}
```

Properties

"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

More information

org.forgerock.openig.filter.oauth2.client.IssuerRepository

JdbcDataSource

Manages connections to a JDBC data source.

To configure the connection pool, add a JdbcDataSource object named `AuditService` in the route heap.

Usage

```
{
  "name": string,
  "type": "JdbcDataSource",
  "config": {
    "dataSourceClassName": configuration expression<string>,
    "driverClassName": configuration expression<string>,
    "executor": ScheduledExectutorService reference,
    "jdbcUrl": configuration expression<url>,
    "passwordSecretId": configuration expression<secret-id>,
    "poolName": configuration expression<string>,
    "properties": object,
    "secretsProvider": SecretsProvider reference,
    "username": configuration expression<string>
  }
}
```

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
A ScheduledExecutorService for maintenance tasks.

Default: ScheduledExecutorService.

"`jdbcUrl`": configuration expression<url>, 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. The values of the object are evaluated as configuration expression<strings>.

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.

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

"`username`": configuration expression<string>, optional
The username to access the database.

Example

For an example that uses JdbcDataSource, see Log In With Credentials From a Database and Recording Access Audit Events in a Database.

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"
}
```

More information

org.forgerock.openig.sql.JdbcDataSourceHeaplet

ProxyOptions

A proxy to which a ClientHandler or ReverseProxyHandler can submit requests, and an AmService can submit Websocket notifications.

Use this object to configure a proxy for AM notifications, and use it in a ClientHandler or ReverseProxyHandler, and again in an AmService notifications block.

Usage

Use one of the following ProxyOption types with the `proxyOptions` option of ClientHandler, ReverseProxyHandler, and AmService:

- No proxy.

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

- System defined proxy options.

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

- Custom proxy

```
{
  "name": string,
  "type": "CustomProxyOptions",
  "config": {
    "uri": configuration expression<url>,
    "username": configuration expression<string>,
    "passwordSecretId": configuration expression<secret-id>,
    "secretsProvider": SecretsProvider reference
  }
}
```

Default: NoProxyOptions

Properties

"uri": configuration expression<url>, 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.

"secretsProvider": _ SecretsProvider <reference>, optional_

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

Example

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

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

In the following example, the AmService notification service passes Websocket notifications to the proxy server, which requires authentication:

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

ScheduledExecutorService

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.

Usage

```
{
  "name": string,
  "type": "ScheduledExecutorService",
  "config": {
    "corePoolSize": configuration expression<number>,
    "gracefulStop": configuration expression<boolean>,
    "gracePeriod": configuration expression<duration>
  }
}
```

Properties

"corePoolSize": configuration expression<number>, 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

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"
    }
}
```

More information

org.forgerock.openig.thread.ScheduledExecutorServiceHeaplet

ServerTlsOptions

When IG is *server-side*, applications send requests to IG or request services from IG. IG is acting as a server of the application, and the application is acting as a client.

ServerTlsOptions configures the TLS-protected endpoint when IG is server-side.

In standalone mode, use ServerTlsOptions in `admin.json`.

Usage

```
{
  "type": "ServerTlsOptions",
  "config": {
    "keyManager": [ KeyManager reference, ...], // Use
"keyManager" or
    "sni": object,                              // "sni", but not
both
    "trustManager": [ TrustManager reference, ...],
    "sslCipherSuites": [ configuration expression<string>, ...],
    "sslContextAlgorithm": configuration expression<string>,
    "sslEnabledProtocols": [ configuration expression<string>,
...],
    "alpn": object,
    "clientAuth": configuration expression<enumeration>,
  }
}
```

Properties

Either `sni` or `keyManager` must be configured. When both are configured, `sni` takes precedence and a warning is logged. When neither is configured, an exception is thrown and a warning is logged.

"sni": object, required if keyManager is not configured

Not supported in web container mode.

Server Name Indication (SNI) is an extension of the TLS handshake, to serve different secret key and certificate pairs to the TLS connections on different server names. Use this property to host multiple domains on the same machine. For more information, see Server Name Indication.

During a TLS handshake, vert.x accesses secret key and certificate pairs sychronously; they are loaded in memory at IG startup, and **must** be present. You must restart IG to update a secret key and certificate pair.

For an example that uses this property, see Serve different certificates for TLS connections to different server names.

```
{
  "sni": {
    "serverNames": map,
    "defaultSecretId": configuration expression<secret-id>,
    "secretsProvider": SecretsProvider reference,
  }
}
```

***serverNames**: map of configuration expression<string>:configuration expression<secret-id>, required*

A map of server name to secret ID:

- The server name is the name of server provided during TLS handshake
- The secret ID represents the secret key and certificate pair to use for that server

```
"serverNames": {
  "app1.example.com": "my.app1.secretId",
  "app2.example.com": "my.app2.secretId",
  "*.test.com": "my.wildcard.test.secretId"
}
```

```
"serverNames": {
  "${server.name.available.at.config.time}" :
"${secret.id.available.at.config.time}"
}
```

Note the following points:

- **One server cannot be mapped to multiple certificates.**

  IG cannot provide multiple certificates for the same server name, as is allowed by Java's KeyManagers.

- **Multiple servers can be mapped to one certificate.**

  Map server names individually. In the following configuration, both server names use the same certificate:

```
"serverNames": {
  "cat.com" : "my.secret.id",
  "dog.org" : "my.secret.id"
}
```

Use the `*` wildcard in the server name to map groups of server names. In the following configuration, `app1.example.com` and `app2.example.com` use the same certificate:

```
"serverNames": {
  "*.example.com": "my.wildcard.secret.id"
}
```

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

The secret ID representing the certificate to use when an unmapped server name is provided during TLS handshake.

For information about how IG manages secrets, see Secrets.

"`secretsProvider`": SecretsProvider reference, required

The SecretsProvider to query for each secret ID. For more information, see SecretsProvider.

"`keyManager`": array of KeyManager references, required if `sni` is not configured

One or more KeyManager objects to serve the same secret key and certificate pair for TLS connections to all server names in the deployment. Key managers are used to prove the identity of the local peer during TLS handshake, as follows:

- When ServerTlsOptions is used in an HTTPS connector configuration (server-side), the KeyManagers to which ServerTlsOptions refers are used to prove this IG's identity to the remote peer (client-side). This is the usual TLS configuration setting (without mTLS).
- When ClientTlsOptions is used in a ClientHandler or ReverseProxyHandler configuration (client-side), the KeyManagers to which ClientTlsOptions refers are used to prove this IG's identity to the remote peer (server-side). This configuration is used in mTLS scenarios.

  Provide the name of one or more of the following objects defined in the heap, or configure one or more of the following objects inline:

  - KeyManager
  - SecretsKeyManager

Default: None

"`sslCipherSuites`": array of configuration expression<strings>, 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). 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 (JVM).

Default: `TLS`

"`sslEnabledProtocols`": array of configuration expression<strings>, 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). For Oracle Java, see the list of Additional JSSE Standard Names.

Follow these protocol recommendations:

- Use TLS 1.3 when it is supported by available libraries, otherwise use TLS 1.2.
- If TLS 1.1 or TLS 1.0 is required for backwards compatibility, use it only with express approval from enterprise security.
- Do use deprecated versions SSL 3 or SSL 2.

Default: TLS 1.3, TLS 1.2

"`trustManager`": array of TrustManager references, optional

One or more of the following TrustManager objects to manage IG's public key certificates:

- TrustManager
- SecretsTrustManager
- TrustAllManager

Trust managers verify the identity of a peer by using certificates, as follows:

- When ServerTlsOptions is used in an HTTPS connector configuration (server-side), the TrustManagers to which ServerTlsOptions refers are used to verify the remote peer's identity (client-side). This configuration is used in mTLS scenarios.
- When ClientTlsOptions is used in a ClientHandler or a ReverseProxyHandler configuration (client-side), the TrustManager to which ClientTlsOptions refers are used to verify the remote peer's identity (server-side). This is the usual TLS configuration setting (without mTLS).

If `trustManager` is not configured, IG uses the default Java truststore to verify the remote peer's identity. The default Java truststore depends on the Java environment. For example, `$JAVA_HOME/lib/security/cacerts`.

Default: None

"`alpn`": object, optional

Not supported in web container mode.

A flag to enable the Application-Layer Protocol Negotiation (ALPN) extension for TLS connections.

```
{
  "alpn": {
    "enabled": configuration expression<boolean>
  }
}
```

`enabled`: configuration expression<boolean>, optional

- `true` : Enable ALPN. Required for HTTP/2 connections over TLS
- `false` : Disable ALPN.

Default: `true`

"`clientAuth`": configuration expression<enumeration>, optional

The authentication expected from the client. Use one of the following values:

- `REQUIRED` : Require the client to present authentication. If it is not presented, then decline the connection.
- `REQUEST` : Request the client to present authentication. If it is not presented, then accept the connection anyway.
- `NONE` : Accept the connection without requesting or requiring the client to present authentication.

Default: `NONE`

Example

See the following examples that use ServerTlsOptions:

- Set up IG for HTTPS (server-side) in standalone mode
- Serve different certificates for TLS connections to different server names

TemporaryStorage

Allocates temporary buffers for caching streamed content during request processing. Initially uses memory; when the memory limit is exceeded, switches to a temporary file.

Usage

```
{
  "name": string,
  "type": "TemporaryStorage",
  "config": {
    "initialLength": configuration expression<number>,
    "memoryLimit": configuration expression<number>,
    "fileLimit": configuration expression<number>,
    "directory": configuration expression<string>
  }
}
```

Properties

"initialLength": configuration expression<number>, optional

Initial size of the memory buffer.

Default: 8 192 bytes (8 KB). Maximum: The value of `"memoryLimit"`.

"memoryLimit": configuration expression<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": configuration expression<number>, optional

Maximum size of the temporary file. If the file is bigger than this value, IG responds with an OverflowException.

Default: 1 073 741 824 bytes (1 GB). Maximum: 2 147 483 647 bytes (2 GB).

"directory": configuration expression<string>, optional

The directory where temporary files are created.

Default: `$HOME/.openig/tmp` (on Windows, `%appdata%\OpenIG\OpenIG\tmp`)

More information

org.forgerock.openig.io.TemporaryStorageHeaplet

TlsOptions (deprecated)

IMPORTANT

This object is deprecated; use ClientTlsOptions instead. For more information, refer to Deprecation.

Configures connections to the TLS-protected endpoint of servers, when IG is client-side.

Usage

```
{
  "name": string,
  "type": "TlsOptions",
  "config": {
    "keyManager": [ KeyManager reference, ...],
    "trustManager": [ TrustManager reference, ...],
    "sslCipherSuites": [ configuration expression<string>, ...],
    "sslContextAlgorithm": configuration expression<string>,
    "sslEnabledProtocols": [ configuration expression<string>,
...],
  }
}
```

Properties

"keyManager": array of KeyManager references references, optional

One or more KeyManager objects to serve the same secret key and certificate pair for TLS connections to all server names in the deployment. Key managers are used to prove the identity of the local peer during TLS handshake, as follows:

- When ServerTlsOptions is used in an HTTPS connector configuration (server-side), the KeyManagers to which ServerTlsOptions refers are used to prove this IG's identity to the remote peer (client-side). This is the usual TLS configuration setting (without mTLS).
- When ClientTlsOptions is used in a ClientHandler or ReverseProxyHandler configuration (client-side), the KeyManagers to which ClientTlsOptions refers are used to prove this IG's identity to the remote peer (server-side). This configuration is used in mTLS scenarios.

  Provide the name of one or more of the following objects defined in the heap, or configure one or more of the following objects inline:

  - KeyManager
  - SecretsKeyManager

Default: None

"sslCipherSuites": array of configuration expression<strings>, 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). 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 (JVM).

Default: `TLS`

"`sslEnabledProtocols`": array of configuration expression<strings>, 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). For Oracle Java, see the list of Additional JSSE Standard Names.

Follow these protocol recommendations:

- Use TLS 1.3 when it is supported by available libraries, otherwise use TLS 1.2.
- If TLS 1.1 or TLS 1.0 is required for backwards compatibility, use it only with express approval from enterprise security.
- Do use deprecated versions SSL 3 or SSL 2.

Default: TLS 1.3, TLS 1.2

"`trustManager`": array of TrustManager references, optional

One or more of the following TrustManager objects to manage IG's public key certificates:

- TrustManager
- SecretsTrustManager
- TrustAllManager

Trust managers verify the identity of a peer by using certificates, as follows:

- When ServerTlsOptions is used in an HTTPS connector configuration (server-side), the TrustManagers to which ServerTlsOptions refers are used to verify the remote peer's identity (client-side). This configuration is used in mTLS scenarios.
- When ClientTlsOptions is used in a ClientHandler or a ReverseProxyHandler configuration (client-side), the TrustManager to which ClientTlsOptions refers are used to verify the remote peer's identity (server-side). This is the usual TLS configuration setting (without mTLS).

If `trustManager` is not configured, IG uses the default Java truststore to verify the remote peer's identity. The default Java truststore depends on the Java environment. For example, `$JAVA_HOME/lib/security/cacerts`.

Default: None

TrustManager

The configuration of a Java Secure Socket Extension TrustManager to manage trust material (typically X.509 public key certificates) for IG. The configuration references the keystore that holds the trust material.

When IG acts as a client, it uses a TrustManager to verify that the server is trusted. When IG acts as a server, it uses a TrustManager to verify that the client is trusted.

Usage

```
{
    "name": string,
    "type": "TrustManager",
    "config": {
      "keystore": KeyStore reference,
      "alg": configuration expression<string>
    }
}
```

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` in web container mode, the keystore can be different to that used by the web container.

See also KeyStore.

"alg": configuration expression<string>, optional

The certificate algorithm to use.

Default: the default for the platform, such as `SunX509`.

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"
      }
    }
  }
}
```

More information

org.forgerock.openig.security.TrustManagerHeaplet

JSSE reference guide, KeyManager, KeyStore

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.

Usage

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

Example

The following example configures a client handler that blindly trusts server certificates when IG connects to servers over HTTPS:

```
{
    "name": "BlindTrustClientHandler",
    "type": "ReverseProxyHandler",
    "config": {
        "trustManager": {
            "type": "TrustAllManager"
        }
    }
}
```

More information

org.forgerock.openig.security.TrustAllManager

UmaService

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.

Usage

```
{
    "name": string,
    "type": "UmaService",
    "config": {
        "protectionApiHandler": Handler reference,
        "amService": AmService reference,                    //
Use either "amService"
        "wellKnownEndpoint": configuration expression<url>,  // or
"wellKnownEndpoint", but not both.
        "resources": [ object, ... ]
    }
}
```

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, see Using the well-known UMA endpoint in AM's *User-Managed Access (UMA) 2.0 guide*.

See also AmService.

"`wellKnownEndpoint`": configuration expression<url>, 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`.

Examples:

- In this example, the UMA configuration is in the default realm of AM:

```
https://am.example.com:8088/openam/uma/.well-known/uma2-configuration
```

- In this example, the UMA configuration is in a European customer realm:

```
https://am.example.com:8088/openam/uma/realms/root/realms/customer/realms/europe/.well-known/uma2-configuration
```

For more information, see Using the well-known UMA endpoint in AM's *User-Managed Access (UMA) 2.0 guide*.

"`resources`": array of objects, 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:

a. When registering a resource set, IG uses the list of actions to provide the aggregated, exhaustive list of all scopes that can be used.

b. When responding to an initial request for a resource, IG derives the scopes for the ticket based on the scopes that apply according to the request.

c. When verifying the RPT, IG checks that all required scopes are encoded in the RPT.

A description of each field follows:

"pattern": 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 objects, optional

A set of scopes to authorize when the corresponding condition evaluates to `true`.

```
"actions": [
  {
    "scopes": [ "#read" ],
    "condition": "${request.method == 'GET'}"
  },
  {
    "scopes": [ "#create" ],
    "condition": "${request.method == 'POST'}"
```

```
    }
]
```

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

One or more scopes that are authorized when the corresponding condition evaluates to `true`.

For example, the scope `#read` grants read-access to a resource.

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

When the condition evaluates to `true`, the corresponding scope is authorized.

For example, the condition `${request.method == 'GET'}` is true when reading a resource.

REST API for shares

The REST API for UMA shares is exposed at a registered endpoint. IG 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 IG 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, IG does not have a mechanism for persisting shares. When IG stops, the shares are discarded.

For information about API descriptors for the UMA share endpoint, see Understanding IG APIs with API descriptors. For information about Common REST, see About ForgeRock Common REST.

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
}
```

"`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, IG 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.

More information

User-Managed Access (UMA) 2.0 Grant for OAuth 2.0 Authorization

org.forgerock.openig.uma.UmaSharingService

Property value substitution

In an environment with multiple IG 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

Configuration Tokens

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.

Configuration Tokens for File System

IG 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 `$HOME/.openig` (`%appdata%\OpenIG`). You can use these tokens in your configuration without explicitly setting their values.

For information about how to change the default values, see Configure default location.

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}"
```

JSON Evaluation

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, IG evaluates the configuration tokens in `config.json` and `admin.json`. When routes are deployed, IG 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.

IG 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, IG 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`, IG 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, IG 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.

Token Resolution

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.

Route Token Resolvers

When a route is deployed in IG 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 Route properties.

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 IG configuration, the value of `port` is `8080`:

```
{
  "port": "&{listen.port}"
}
```

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 IG configuration, the value of `port` is `8080`:

```
{
  "port": "&{listen.port}"
}
```

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. IG 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 IG 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" />
  ```

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

```
{
   "$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" ]`.

array

```
{"$array": string}
```

Returns a JSON array of the argument.

Argument	Returns
string String representing a JSON array.	***array*** JSON array of the argument.

The following example transformation results in the JSON array `[ "one", "two" ]`:

```
{"$array": "[ \"one\", \"two\" ]"}
```

bool

```
{"$bool": string}
```

Returns `true` if the input value equals `"true"` (ignoring case). Otherwise, returns `false`.

Argument	Returns
string String containing the boolean representation.	***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}"}
```

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	Parameters	Returns
string Base64-encoded string.	***$charset*** The name of a Java character set, as described in Class Charset.	***string*** Base64-decoded string in the given character set.

The following example transformation returns the `Hello` string:

```
{
 "$base64:decode": "SGVsbG8=",
 "$charset": "UTF-8"
}
```

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	Parameters	Returns
string String to encode with the given character set.	*$charset* The name of a Java character set, as described in Class Charset.	*string* Base64-encoded string.

int

```
{"$int": string}
```

Transforms a string into an integer.

If the parameter is not a valid number in radix 10, returns `null`.

Argument	Returns
string String containing the integer representation.	*int* Integer value represented by the argument.

The following example transformation results in the integer `1234`:

```
{"$int": "1234"}
```

list

```
{"$list": string}
```

Transforms a comma-separated list of strings into a JSON array of strings

Argument	Returns
string A string representing a comma-separated list of strings.	*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"}
```

number

```
{"$number": string}
```

Transform a string into a Java number, as defined in Class Number.

Argument	Returns
strings A string containing the number representation.	*number* The number value represented by the argument.

The following example transformation results in the number `0.999`:

```
{"$number": ".999"}
```

object

```
{"$object": string}
```

Transforms a string representation of a JSON object into a JSON object.

Argument	Returns
string String representation of a JSON object.	***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
  }
}
```

string

```
{"$string": placeholder string}
```

Transforms a string representation of a JSON object into a placeholder string. Placeholder strings are not encrypted.

Use this transformation for placeholder strings that that must not be encrypted.

Argument	Returns
string String representation of a JSON object.	*placeholder string* Placeholder string.

This example transformation:

```
{
  "someAttributeExpectingString": { "$string": "&
{ig.instance.dir}" }
}
```

results in this JSON object:

```
{
  "someAttributeExpectingString": "/path/to/ig"
}
```

Expressions

Expressions that conform to the Universal Expression Language as specified in JSR-245 can be used to specify configuration parameter values.

General Syntax

All expressions follow standard Universal Expression Language syntax: `${expression}` or `#{expression}`:

An 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}"`.

Configuration and Runtime Expressions

Expressions are evaluated at configuration time (when routes are loaded), or at runtime (when IG 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 IG files. The default location is `$HOME/.openig` (`%appdata%\OpenIG`).
- `configDirectory`, the path to the IG configuration files. The default location is `$HOME/.openig/config` (`%appdata%\OpenIG\config`).
- `temporaryDirectory`, the path to the IG temporary files. The default location is `$HOME/.openig/tmp` (`%appdata%\OpenIG\OpenIG\tmp`).

For information about how to change the default values, see Configure default location.

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`.

Runtime expressions

Expressions evaluated at runtime, for each request and response. IG evaluates runtime expressions as follows:

- When expressions are written with `$`, IG evaluates them immediately. If the expression consumes streamed content, for example, the content of a request or response, IG blocks the executing thread until all of the content is available. *Immediate evaluation* of expressions can block the executing thread.
- When expressions are written with `#`, IG defers evaluation until all of the streamed content is available. *Deferred evaluation* of expressions does not block the executing thread.

For expressions that consume streamed content, make sure that IG does a deferred evaluation, by writing the expression with `#` instead of `$`.

In standalone mode, when the `streaming` property in `admin.json` is `true`, expressions that consume streamed content **must** be written with `#` instead of `$`.

Examples
The following expressions do not consume streamed content, so it is safe for IG to do an immediate evaluation. The expressions can therefore be written with `$`:

```
$ {find(request.uri.path, '^/foo')}
```

```
$ {request.headers['Content-Type'][0] == 'application/json'}
```

The following example expression is a Route condition. The Route is accessed if the request contains json with the attribute `answer`, whose value is `42`. IG must defer evaluation of the expression until it receives the entire body of the reqest, transfoms it to JSON view, and introspects it for the attribute `answer`:

```
{
  "condition": "#{request.entity.json['answer'] == 42}",
  "handler": ...
}
```

The following example expression is a DispatchHandler condition. The request is dispatched if the request contains a form field with the attribute `answer`, whose value is `42`. IG must defer evaluation of the expression until it receives the entire body of the reqest, transforms it to a form view, and introspects it for the attribute `answer`:

```
"bindings": [
  {
    "condition": "#{request.entity.form['answer'] == 42}",
    "handler": ...
  }
]
```

The following example expression is for an AssignmentFilter. A user ID is captured from a response and stored in the `FooBar` header. The first expression defines the target, and IG evaluates it immediately. The second expression uses the response entity, so IG must defer evaluation until it receives the entire body of the response:

```
"onResponse": [
  {
    "target": "${response.headers['X-IG-FooBar']}",
    "value": "#{toString(response.entity.json['userId'])}"
  }
]
```

The following example expression is for a JwtBuilderFilter. The content of the requests is mapped as a string, so IG must defer evaluation of the expression until it receives the entire body of the request:

```
{
  "template": {
    "content": "#{request.entity.string}"
```

```
    }
}
```

Available Objects
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.

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.

Properties whose format is `lvalue-expression` cannot consume streamed content. They must be written with `$` instead of `#` .

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.

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`
- `? :`

System Properties and Environment Variables

You can use expressions to retrieve Java system properties, and to retrieve nvironment 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 IG.

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 IG.

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}
```

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 lowercase.

Functions can be operands for operations, and can yield parameters for other function calls.

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.

Embedding Expressions

Consider the following points when embedding expressions:

- System properties, environment variables, or function expressions can be embedded within expressions.

  The following example embeds an environment variable in the argument for a `read()` function. The value of `entity` is set to the contents of the file `$HOME/.openig/html/defaultResponse.html`, where `$HOME/.openig` is the instance directory:

  ```
  "entity": "${read('&
  {ig.instance.dir}/html/defaultResponse.html')}"
  ```

- Expressions cannot be embedded inside other expressions, as `${expression}`.
- Embedded elements cannot be enclosed in `${}`.

Extensions

IG offers a plugin interface for extending expressions. See Key extension points.

If your deployment uses expression plugins, read the plugin documentation about the additional expressions you can use.

Examples

```
"${request.uri.path == '/wordpress/wp-login.php'
   and request.queryParams['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"
```

More information

For more information, see Contexts, Functions, Request, and Response.

Functions

A set of built-in functions that can be called from within expressions, which are described in Expressions.

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.

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.

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.

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.

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.

digestSha256

```
digestSha256(byte array or string)
```

Calculates the SHA-256 hash of an incoming object.

Parameters

byte array or string

The bytes to be hashed. If a string is provided, this function uses the UTF-8 charset to get the bytes from the string.

Returns

byte array

SHA-256 hash as a byte array, or null if the hash could not be calculated.

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.

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.

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`.

find

```
find(string, pattern)
```

Attempts to find the next subsequence of the input string that matches the pattern.

Parameters
string
The input string.

pattern
A regular expression pattern.

Returns
boolean

- `true` if a subsequence of the input string matches the regular expression pattern.

- `false` if the input string is null, or a subsequence of it does not match the regular expression pattern.

findGroups

```
findGroups(string, pattern)
```

Attempts to find a string that matches the regular expression or groups specified in the regular expression.

Parameters
string
The input on which the regular expression is applied.

pattern
A regular expression pattern.

Returns
array
An array containing the result of a find on the regular expression against the input string, or null if no result is found.

The first element of the array is the entire match, and each subsequent element correlates to a capture group specified in the regular expression.

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`.

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`.

indexOf

```
indexOf(string, substring)
```

Returns the index of the first instance of a specified substring inside a string.

Characters in the provided string are UTF-16, based on 16-bit code units. Each character is encoded as at least two bytes, and some extended characters are encoded as four bytes.

When this function processes a 2-byte character, it counts it as one 16-bit character. When it processes a 4-byte character, it counts it as two 16-bit characters.

Examples:

- The unicode character `a` (U+0061) has the UTF-16 value `0x0061`. The function `{{indexOf('afooBar', 'Bar')}}` evaluates to `4`.
- The unicode character 𐐗 (U+10057) has the UTF-16 value `0xD800 0xDC57`. The function `{{indexOf('𐐗fooBar', 'Bar')}}` evaluates to `5`.

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 the substring, or -1 if not found.

The index count starts from 1, not 0.

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.

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.

ipMatch

```
ipMatch(string, string)
```

Returns `true` if the provided IP address matches the range provided by the Classless Inter-Domain Routing (CIDR), or `false` otherwise.

Parameters
string
IP address of a request sender, in IPv4 or IPv6

string
CIDR defining an IP address range

Returns
Boolean
`true or false`

join

```
join(values, separator)
```

Returns a string joined with the given separator, using either of the following values:

- Array of strings (`String[]`)
- Iterable value (`Iterable<String>`)

The function uses the `toString` result from each value.

Parameters
separator
Separator to place between joined values.

strings
Array of values to be joined.

Returns
string
String containing the joined values.

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.

length

```
length(object)
```

Returns the number of items in a collection, or the number of characters in a string.

Characters in the provided string are UTF-16, based on 16-bit code units. Each character is encoded as at least two bytes, and some extended characters are encoded as four bytes.

When this function processes a 2-byte character, it counts it as one 16-bit character. When it processes a 4-byte character, it counts it as two 16-bit characters.

Examples:

- The unicode character `a` (U+0061) has the UTF-16 value `0x0061`. The function `{{length('a')}}` evaluates to `1`.
- The unicode character 𐐗 (U+10057) has the UTF-16 value `0xD800 0xDC57`. The function `{{length('𐐗')}}` evaluates to `2`.

Parameters
object
A collection or string, whose length is to be determined.

Returns
number
Length of the collection or string, or 0 if length could not be determined.

matches (deprecated)

IMPORTANT

This function is deprecated. Use the matchesWithRegex or find function instead. For more information, refer to Deprecation.

```
matches(string, pattern)
```

Returns `true` if the string contains a match for the specified regular expression pattern.

Parameters

string

The input string.

pattern

A regular expression pattern.

Returns

true

String contains the specified regular expression pattern.

matchesWithRegex

```
matchesWithRegex(string, pattern)
```

Attempts to match the entire input string against the regular expression pattern.

Parameters

string

The input string.

pattern

A regular expression pattern.

Returns

boolean

- `true` if the entire input string matches the regular expression pattern.
- `false` if the input string is null, or the entire input string does not match the regular expression pattern.

matchingGroups (deprecated)

IMPORTANT

This function is deprecated. Use the findGroups function instead. For more information, refer to Deprecation.

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.

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`.

pemCertificate

```
(string)
```

Convert the incoming character sequence into a certificate.

Parameters

string

Character sequence representing a PEM-formatted certificate

Returns

string

A `Certificate` instance, or null if the function failed to load a certificate from the incoming object.

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.

readProperties

```
readProperties(string)
```

Takes a Java Properties filename 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).

Java properties files are expected to be encoded with ISO-8859-1. Characters that cannot be directly represented in ISO-8859-1 can be written using Unicode escapes, as defined in Unicode Escapes, in *The Java™ Language Specification*.

Parameters

string

The absolute path to the Java properties file, or a path relative to the location of the Java system property `user.dir`.

For example, to return the value of the `key` property in the Java properties file `/path/to/my.properties`, provide `${readProperties('/path/to/my.properties')['key']}`.

Returns

object

Key/value map of properties or `null` on error.

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.

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.

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`.

toLowerCase

```
toLowerCase(string)
```

Converts all of the characters in a string to lowercase.

Parameters

string

String whose characters are to be converted.

Returns

string

String with characters converted to lowercase.

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.

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.

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.

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`.

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`.

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.

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.

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.

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.

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.

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.

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.

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.

More information

Some functions are provided by org.forgerock.openig.el.Functions.

Other functions are provided by org.forgerock.http.util.Uris.

Patterns

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.

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.

More information

Java Pattern class

Regular Expressions tutorial

Scripts

Use scripts with the following 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

When a script is accessed, IG compiles and then caches the script. Unless the script is changed, IG continues to use the cached version.

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.

Usage

```
{
  "name": string,
  "type": scriptable object type,
  "config": {
    "type": string,
    "file": configuration expression<string>,  // Use either
"file"
    "source": [ string, ... ],                // or "source", but
not both.
    "args": map,
    "clientHandler": Handler reference
  }
}
```

Properties

"type": string, required

The Internet media type (formerly MIME type) of the script, "application/x-groovy" for Groovy

"file": configuration expression<string>

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`": array of strings>, required if `file` is not used

The script as one or more 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

A name-value map of `configuration parameter:script parameters`, where:

- `configuration parameter` is a string
- `script parameters` is one or more configuration expressions that resolve to strings

Examples:

- 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: 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.

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 IG 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.

builder

For ScriptableJwtValidatorCustomizer only.

Used by the ScriptableJwtValidatorCustomizer and JwtValidationFilter to create constraints to test JWT claims and sub-claims. The purpose of the ScriptableJwtValidatorCustomizer is to enrich the `builder` object.

For information about methods to enrich the `builder` instance, see JwtValidator.Builder.

constraints

The constraints object, all its static methods, `constant(String)`, and `claim(String)`.

Use this object for JWT validation with the `customizer` property of JwtValidationFilter.

`claim(String)` must be followed by one of the following methods: `asString()`, `asInteger()`, `asLong()`, `asDouble()`, `asBoolean()`, `as(yourCustomJsonValueTransformer)`

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.

IMPORTANT

The `request.form` method, used in scripts to read or set query and form parameters, is deprecated. Use the following replacement settings:

- `Request.getQueryParams()` to read query parameters
- `Entity.getForm()` to read form parameters
- `Entity.setForm()` to set form parameters

For more information, refer to Deprecation.

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 (deprecated)

IMPORTANT

The `LdapClient` class and the `ldap` script binding are deprecated. For more information, refer to Deprecation.

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.

For information about logging for scripts, see Logging In Scripts.

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, IG 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.

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.Header
- 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)`
- org.forgerock.openig.util.JsonValues and all its static methods.
- org.forgerock.openig.tools.jwt.validation.Constraints and all its static methods.

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.openig.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
- `ScriptableJwtValidatorCustomizer` in JwtValidationFilter and org.forgerock.openig.filter.jwt.ScriptableJwtValidatorCustomizer

Route properties

Configuration parameters, such as host names, port numbers, and directories, can be declared as property variables in the IG 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 IG in different environments without changing a single line in your route configuration.

Usage

Simple property configured inline

```
{
  "properties": {
    "<variable name>": "valid JSON value"
  }
}
```

Group property configured inline

```
{
  "properties": {
    "<group name>": {
      "<variable name>": "valid JSON value", ...
    }
  }
}
```

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
    }
  }
}
```

Properties

"<variable name>": configuration expression<string>

The name of a variable to use in the IG 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.

- In the following example from `config.json`, the URL of an application is declared as a property variable named `appLocation`. The variable is then used by the `baseURI` parameter of the handler, and can be used again in other routes in the configuration.

```
{
  "properties": {
    "appLocation": "http://app.example.com:8081"
  },
  "handler": {
    "type": "Router",
    "baseURI": "${appLocation}",
    "capture": "all"
  }
}
```

- In the following example, the property variable `ports` is added to define an array of port numbers used by the configuration. The `ports` variable is referenced in the `appLocation` variable, and is resolved at runtime with the value in the ports array:

```
{
  "properties": {
    "ports": [8080, 8081, 8088],
    "appLocation": "http://app.example.com:${ports[1]}"
  },
  "handler": {
    "type": "Router",
```

```
    "baseURI": "${appLocation}",
    "capture": "all"
  }
}
```

- In the following example route, the request path is declared as the property variable `uriPath`, with the value `hello`, and the variable is used by the route condition:

```
{
  "properties": {
    "uriPath": "hello"
  },
  "handler": {
    "type": "StaticResponseHandler",
    "config": {
      "status": 200,
      "headers": {
        "Content-Type": [ "text/plain; charset=UTF-8" ]
      },
      "entity": "Hello world!"
    }
  },
  "condition": "${matchesWithRegex(request.uri.path,
'^/welcome') or matchesWithRegex(request.uri.path, '&
{uriPath}')}"
}
```

When IG is set up as described in the Getting started, requests to `ig.example.com:8080/hello` or `ig.example.com:8080/welcome` can access the route.

"<group name>": <object>, required

The name of a group of variables to use in the IG 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`": configuration expression<string>, 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://am.example.com:8088/openam/",
  "portNumber": 8081
}
```

This file would cause an error:

```
{
  "openam.location": "http://am.example.com:8088/openam/",
  "port.number": 8081
}
```

Examples

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://am.example.com:8088/openam/",
  "appPortNumber": 8081
}
```

Property 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}",
    . . .
```

Requests, responses, 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.

AttributesContext

Provides a map for request attributes. When IG 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.

IG 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.

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.

More information

org.forgerock.services.context.AttributesContext

AuthRedirectContext

Used by the FragmentFilter to indicate that a login redirect is pending.

Properties

"isImpendingIgRedirectNotified": boolean
Returns `true` if a downstream filter notifies IG that it will redirect a login attempt. Otherwise, `false`.

"notifyImpendingIgRedirect"
IG sets this context to `true` when a filter triggers a redirect which will be followed by a subsequent request to the original URI.

For example, a request to `example.com/profile` triggers a login redirect to `example.com/login`. After authentication, the request is expected to be redirected to the original URI, `example.com/profile`.

More information

FragmentFilter

org.forgerock.openig.filter.AuthRedirectContext

URI Fragment

Fragment RFC

CapturedUserPasswordContext

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.

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.

More information

org.forgerock.openig.openam.CapturedUserPasswordContext

ClientContext

Information about the client sending a request. When IG receives a request, it injects information about the client sending the request into this context.

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, IG 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 IG with a non-default deployment descriptor that secures the IG web application.

*"**`userAgent`**": string*
The value of the User-Agent HTTP header in the request if any, otherwise `null`.

More information

org.forgerock.services.context.ClientContext

Contexts

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.

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
- JwtValidationContext and JwtValidationErrorContext
- JwtBuilderContext
- OAuth2Context
- OAuth2TokenExchangeContext
- OAuth2FailureContext
- PolicyDecisionContext
- SessionInfoContext
- SsoTokenContext
- StsContext
- UserProfileContext

More information

org.forgerock.services.context.Context

CdSsoContext

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.

Properties

The context is named `cdsso`, and is accessible at `${contexts.cdsso}`. The context has the following properties:

"claimsSet": org.forgerock.json.jose.jwt.JwtClaimsSet
Full JwtClaimsSet for the identity of the authenticated user. Cannot be null.

Access claims as follows:

- Claims with a getter by using the property name. For example, access `getSubject` with `contexts.cdsso.claimsSet.subject`.
- All other claims by using the `getClaim` method. For example, access `subname` with `contexts.cdsso.claimsSet.getClaim('subname')`.

"cookieInfo": org.forgerock.openig.http.CookieBuilder
Configuration data for the CDSSO authentication cookie, with the following attributes:

- `name` : Cookie name (string)
- `domain` : (Optional) Cookie domain (string)
- `path` : Cookie path (string)

No attribute can be null.

"redirectEndpoint": java.lang.String
Redirect endpoint URI configured for communication with AM. Cannot be null.

"sessionUid": java.lang.String
Universal session ID. Cannot be null.

"token": java.lang.String
Value of the CDSSO token. Cannot be null.

More information

org.forgerock.openig.openam.CdSsoContext

CdSsoFailureContext

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.

Properties

The context is named `cdssoFailure`, and is accessible at `${contexts.cdssoFailure}`. The context has the following properties:

*"**`error`**": string*
: The error that occurred during authentication. Cannot be null.

*"**`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.

More information

org.forgerock.openig.openam.CdSsoFailureContext

JwtBuilderContext

When the JwtBuilderFilter processes a request, it stores provided data in this context. This context returns the JWT as string, JsonValue, or map for downstream use.

Properties

The context is named `jwtBuilder`, and is accessible at `${contexts.jwtBuilder}`, with the following properties:

*"**`value`**": string*
: The base64url encoded UTF-8 parts of the JWT, containing name-value pairs of data. Cannot be null.

*"**`claims`**": map*
: JWT claims as a map, with the format `Map<String, Object>`.

*"**`claimsAsJsonValue`**": jsonvalue*
: JWT claims as a JsonValue.

More information

org.forgerock.openig.filter.JwtBuilderFilter

org.forgerock.openig.filter.JwtBuilderContext

JwtValidationContext

Provides the properties of a JWT after validation. When the JwtValidationFilter validates a JWT, or the IdTokenValidationFilter validates an id_token, it injects a copy of the JWT and its claims into this context.

Properties

The context is named `jwtValidation`, and is accessible at `${contexts.jwtValidation}`. The context has the following properties:

*"**value**": string*
: The value of the JWT. Cannot be null.

*"**claims**": JWT claims set*
: A copy of the claims as a JwtClaimsSet.

*"**info**": map*
: A copy of the claims as a map.

*"**jwt**": JWT*
: A copy of the JWT.

More information

org.forgerock.openig.filter.jwt.JwtValidationFilter

org.forgerock.openig.filter.oauth2.client.IdTokenValidationFilterHeaplet

org.forgerock.openig.filter.jwt.JwtValidationContext

org.forgerock.openig.filter.jwt.JwtValidationErrorContext

JwtValidationErrorContext

Provides the properties of a JWT after validation fails. When the JwtValidationFilter fails to validate a JWT, or the IdTokenValidationFilter fails to validate an id_token, it injects the JWT and a list of violations into this context.

Properties

The context is named `jwtValidationError`, and is accessible at `${contexts.jwtValidationError}`. The context has the following properties:

*"**`jwt`**": string*
The value of the JWT. Cannot be null.

*"**`violations`**": list of violations*
An unmodifiable list of violations. For more information, see the Violation class.

More information

org.forgerock.openig.filter.jwt.JwtValidationFilter

org.forgerock.openig.filter.oauth2.client.IdTokenValidationFilterHeaplet

org.forgerock.openig.filter.jwt.JwtValidationContext

org.forgerock.openig.filter.jwt.JwtValidationErrorContext

OAuth2Context

Provides OAuth 2.0 access tokens. When the OAuth2ResourceServerFilter processes a request, it injects the access token into this context.

Properties

The context name is `oauth2`, and is accessible at `${contexts.oauth2}`. The context has the following properties:

*"**`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.

More information

org.forgerock.http.oauth2.OAuth2Context

org.forgerock.http.oauth2.AccessTokenInfo

OAuth2TokenExchangeContext

When the OAuth2TokenExchangeFilter succesfully issues a token, it injects the issued token and its scopes into this context.

Properties

The context name is `OAuth2TokenExchangeContext`, and is accessible at `${contexts.oauth2TokenExchange}`.

The context has the following properties:

"issuedToken": string
: The token issued by the authorization server.

"issuedTokenType": string
: The token type URN.

"scopes": set of strings
: One or more scopes associated with the issued token, for example, `"scope1"`, `"scope2"`, `"scope3"`.

"rawInfo": raw JSON as a JsonValue
: The raw token info as issued by the authorization server.

More information

org.forgerock.http.oauth2.OAuth2TokenExchangeContext

OAuth2FailureContext

OAuth2TokenExchangeFilter

RFC 6749: Error Response

OAuth2FailureContext

When an OAuth 2.0 authorization operation fails, the error and error description provided by the authorization service are injected into this context for use downstream.

For example, when the OAuth2TokenExchangeFilter fails to exchange a token, it injects the error and description into this context. The context is passed into calls to the `failureHandler` in the OAuth2TokenExchangeFilter.

This context supports OAuth 2.0 error messages in the format given by RFC 6749.

Properties

The context is named `OAuth2Failure`, and is accessible at `${contexts.oauth2failure}`. The context has the following properties:

*"**`error`**": string*
: The error field name.

*"**`description`**": string*
: Error description field name.

*"**`exception`**": Exception*
: The OAuth 2.0 exception associated with the token exchange error.

More information

org.forgerock.http.oauth2.OAuth2FailureContext

OAuth2TokenExchangeContext

OAuth2TokenExchangeFilter

RFC 6749: Error Response

PolicyDecisionContext

Provides attributes and advices returned by AM policy decisions. When the PolicyEnforcementFilter processes a request, it injects the attributes and advices into this context.

Properties

The context is named `policyDecision`, and is accessible at `${contexts.policyDecision}`. The context has the following properties:

*"**`attributes`**": unmodifiable map*
: A map of attribute names to their values, provided in the policy decision. Can be empty, but not null.

*"**`jsonAttributes`**": JsonValue*
: A map of attribute names to their values, provided in the policy decision. Cannot be null.

*"**`advices`**": map*
: A map of advice names to their values, provided in the policy decision. Can be empty, but not null.

*"**`jsonAdvices`**": JsonValue*
A map of advice names to their values, provided in the policy decision. Cannot be null.

*"**`actions`**": map*
A map of action name keys to Boolean values that indicate whether the action is allowed (true) or denied (false) for the specified resource. Can be empty, but not null.

*"**`jsonActions`**": JsonValue*
A map of action name keys to Boolean values that indicate whether the action is allowed (true) or denied (false) for the specified resource. Cannot be null.

*"**`resource`**": string*
The resource value that was used when making the policy request. Can be empty, but not null.

More information

org.forgerock.openig.openam.PolicyDecisionContext.html

Request

An HTTP request message. Access the content of the request by using expressions.

Properties

*"**`method`**": java.lang.String*
The HTTP method; for example, `GET`.

*"**`uri`**": java.net.URI*
The fully-qualified URI of the resource being accessed; for example, `http://www.example.com/resource.txt`.

*"**`version`**": java.lang.String*
The protocol version used for the request; for example, `HTTP/2`.

*"**`headers`**": org.forgerock.http.protocol.Headers*

One or more headers in the request, with the format *`header_name`*: [*`header_value`*, …]. The following example accesses the first value of the request header `UserId`:

```
pass:[${request.headers['UserId'][0]}
```

*"**`cookies`**": org.forgerock.http.protocol.RequestCookies*

Incoming request cookies, with the format `cookie_name: [ cookie_value, … ]`. The following example accesses the first value of the request cookie `my-jwt`:

```
pass:[${request.cookies['my-jwt'][0].value}
```

*"**`entity`**": Entity*

The message body. The following example accesses the subject token from the request entity:

```
pass:[#{request.entity.form['subject_token'][0]}]
```

*"**`queryParams`**": Form*

Returns a copy of the query parameters decoded as a form. Modifications to the returned form are not reflected in the request.

More information

org.forgerock.http.protocol.Request

Response

An HTTP response message. Access the content of the response by using expressions.

Properties

*"**`cause`**": java.lang.Exception*

The cause of an error if the status code is in the range 4xx-5xx. Possibly null.

*"**`status`**": Status*

The response status.

*"**`version`**": java.lang.String*

The protocol version used the response; for example, `HTTP/2`.

*"**`headers`**": org.forgerock.http.protocol.Headers*

One or more headers in the response. The following example accesses the first value of the response header `Content-Type`:

```
pass:[${response.headers['Content-Type'][0]}]
```

*"**`trailers`**": org.forgerock.http.protocol.Headers*

One or more trailers in the response. The following example accesses the first value of the response trailer `Content-Length`:

```
pass:[${response.trailers['Content-Length'][0]}]
```

*"**`entity`**": Entity*

The message entity body. The following example accesses the user ID from the response:

```
pass:[#{toString(response.entity.json['userId'])}]
```

More information

org.forgerock.http.protocol.Response

SessionContext

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 IG from creating additional sessions.

IG 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}`

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.

More information

org.forgerock.http.session.SessionContext

SessionInfoContext

Provides AM session information and properties. When the SessionInfoFilter processes a request, it injects info and properties from the AM session into this context.

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 IG 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.

More information

org.forgerock.openig.openam.SessionInfoContext

SsoTokenContext

Provides SSO tokens and their validation information. When the SingleSignOnFilter or CrossDomainSingleSignOnFilter processes a request, it injects the value of the SSO token and additional information in this context.

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.

More information

org.forgerock.openig.openam.SsoTokenContext

Status

An HTTP response status.

Properties

*"**`code`**": integer*
Three-digit integer reflecting the HTTP status code.

*"**`family`**": enumeration*

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.

*"**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.

More information

Response Status Codes.

org.forgerock.http.protocol.Status

StsContext

Provides the result of a token transformation. When the TokenTransformationFilter processes a request, it injects the result into this context.

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.

More information

org.forgerock.openig.openam.StsContext

TransactionIdContext

The transaction ID of a request. When IG receives a request, it injects the transaction ID into this context.

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.

More information

org.forgerock.services.TransactionIdContext

org.forgerock.services.context.TransactionIdContext

URI

Represents a Uniform Resource Identifier (URI) reference.

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.

More information

org.forgerock.http.MutableUri

UriRouterContext

Provides routing information associated with a request. When IG routes a request, it injects information about the routing into this context.

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**": url*
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.

More information

org.forgerock.http.routing.UriRouterContext

UserProfileContext

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.

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()}`

More information

org.forgerock.openig.openam.UserProfileContext

UserProfileFilter

Access token resolvers

The following objects are available to resolve OAuth 2.0 access tokens:

TokenIntrospectionAccessTokenResolver

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.

Usage

Use this resolver with the `accessTokenResolver` property of OAuth2ResourceServerFilter.

```
"accessTokenResolver": {
  "type": "TokenIntrospectionAccessTokenResolver",
  "config": {
    "amService": AmService reference,          // Use either
"amService"
    "endpoint": configuration expression<url>, // or "endpoint",
but not both.
    "providerHandler": Handler reference
  }
}
```

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": configuration expression<url>, 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"
  }
}
```

Example

For an example route that uses the token introspection endpoint, see Validate access tokens through the introspection endpoint.

More information

org.forgerock.openig.filter.oauth2.TokenIntrospectionAccessTokenResolverHeaplet

OAuth2ResourceServerFilter

StatelessAccessTokenResolver

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.

Usage

Use this resolver with the `accessTokenResolver` property of OAuth2ResourceServerFilter.

```
"accessTokenResolver": {
  "type": "StatelessAccessTokenResolver",
  "config": {
```

```
    "issuer": configuration expression<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>,
    "signatureSecretId" : configuration expression<secret-id>,
//deprecated
    "encryptionSecretId" : configuration expression<secret-id>
//deprecated
  }
}
```

Properties

"issuer": configuration expression<string>, 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: The route's default secret service. For more information, see Default secrets object.

"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.

For information about how signatures are validated, see Validating the signature of signed tokens. For information about how each type of secret store resolves named secrets, see Secrets.

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.

"`skewAllowance`": configuration expression<duration>, optional

The duration 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 IG clock.
- A JWT with an `exp` 13:00 is expired after 13:02 on the IG clock.

Default: `zero`

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

IMPORTANT

This property is deprecated. Use `verificationSecretId` instead. For more information, refer to Deprecation.

The secret ID for the secret used to verify the signature.

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

IMPORTANT

This property is deprecated. Use `decryptionSecretId` instead. For more information, refer to Deprecation.

The secret ID for the secret used to decrypt data.

Example

For examples of how to set up and use StatelessAccessTokenResolver to resolve signed and encrypted access tokens, see Validate stateless access tokens with the StatelessAccessTokenResolver.

More information

org.forgerock.openig.filter.oauth2.StatelessAccessTokenResolver

OAuth2ResourceServerFilter

OpenAmAccessTokenResolver (deprecated)

IMPORTANT

This object is deprecated because the corresponding feature was deprecated in AM 6.5. Consider using the TokenIntrospectionAccessTokenResolver to resolve access tokens and retrieve metadata about the token.

For more information, refer to Deprecation.

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.

Usage

Use this resolver with the `accessTokenResolver` property of OAuth2ResourceServerFilter.

```
"accessTokenResolver": {
  "type": "OpenAmAccessTokenResolver",
  "config": {
    "amService": AmService reference,
    "providerHandler": Handler reference,
    "endpoint": configuration expression<url>
  }
}
```

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

"endpoint": configuration expression<url>, required if `amService` is not configured

The URI for the authorization service.

More information

org.forgerock.http.oauth2.resolver.OpenAmAccessTokenResolver

OAuth2ResourceServerFilter

ConfirmationKeyVerifierAccessTokenResolver

Supported from AM 6.5.1.

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 IG 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 IG, throw an error.

For an example that uses the ConfirmationKeyVerifierAccessTokenResolver, see Validate certificate-bound access tokens.

For information about issuing certificate-bound OAuth 2.0 access tokens, see Certificate-bound proof-of-possession in AM's *OAuth 2.0 guide*. For information about authenticating an OAuth 2.0 client using mTLS certificates, see Authenticating clients using mutual TLS in AM's *OAuth 2.0 guide* .

Usage

Use this resolver with the `accessTokenResolver` property of OAuth2ResourceServerFilter.

```
"accessTokenResolver": {
  "type": "ConfirmationKeyVerifierAccessTokenResolver",
  "config": {
    "delegate": AccessTokenResolver reference
  }
}
```

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.

Examples

For an example that uses the ConfirmationKeyVerifierAccessTokenResolver with the following route, see Validate certificate-bound access tokens.

More information

org.forgerock.openig.filter.oauth2.cnf.ConfirmationKeyVerifierAccessTokenResolver

OAuth 2.0 Mutual TLS Client Authentication and Certificate Bound Access Tokens

OAuth2ResourceServerFilter

ScriptableAccessTokenResolver

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`.

Usage

Use this resolver with the `accessTokenResolver` property of OAuth2ResourceServerFilter.

```
"accessTokenResolver": {
  "type": "ScriptableAccessTokenResolver",
  "config": {
    "type": configuration expression<string>,
    "file": configuration expression<string>,  // Use either
"file"
    "source": [ string, ... ],                 // or "source", but
not both.
    "args": map,
    "clientHandler": Handler reference
  }
}
```

Properties

For information about properties for ScriptableAccessTokenResolver, see Scripts.

More information

org.forgerock.openig.filter.oauth2.ScriptableAccessTokenResolver

OAuth2ResourceServerFilter

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 OAuth 2.0 access tokens, and delegates their resolution to another AccessTokenResolver. Use this resolver with AM or any OAuth 2.0 access token provider.

For an alternative way to cache OAuth 2.0 access tokens, configure the `cache` property of OAuth2ResourceServerFilter.

Usage

```
{
  "name": string,
  "type": "CacheAccessTokenResolver",
  "config": {
    "delegate": AccessTokenResolver reference,
    "enabled": configuration expression<boolean>,
    "defaultTimeout": configuration expression<duration>,
    "executor": Executor reference,
    "maximumSize": configuration expression<number>,
    "maximumTimeToCache": configuration expression<duration>,
    "amService": AmService reference,
    "onNotificationDisconnection": configuration
expression<enumeration>
  }
}
```

Properties

"`delegate`": AccessTokenResolver reference, required

Delegate access token resolution to one of the access token resolvers in Access token resolvers.

To use AM WebSocket notification to evict revoked access tokens from the cache, the delegate must be able to provide the token metadata required to update the cache.

- The `notification` property of AmService is enabled.

- The delegate AccessTokenResolver provides the token metadata required to update the cache.

`enabled`: configuration expression<boolean>, optional

Enable caching.

When an access token is cached, IG can reuse the token information without repeatedly asking the authorization server to verify the access token. When caching is disabled, IG 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`": Executor 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`

"`amService`": AmService reference, optional

(From AM 6.5.3.) An AmService to use for the WebSocket notification service.

When an access token is revoked on AM, the CacheAccessTokenResolver can delete the token from the cache when both of the following conditions are true:

- The `notification` property of AmService is enabled.
- The delegate AccessTokenResolver provides the token metadata required to update the cache.

  When a refresh_token is revoked on AM, all associated access tokens are automatically and immediately revoked.

See also AmService.

`onNotificationDisconnection`: configuration expression<enumeration>, optional

An `amService` must be configured for this property to have effect.

The strategy to manage the cache when the WebSocket notification service is disconnected, and IG receives no notifications for AM events. If the cache is not cleared it can become outdated, and IG can allow requests on revoked sessions or tokens.

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`

Example

For an example that uses the CacheAccessTokenResolver, see Cache access tokens.

Secrets object and secret stores

IG uses the Commons Secrets Service to manage secrets, such as passwords and cryptographic keys.

For more information about how IG manages secrets, see Secrets.

Default secrets object (deprecated)

IMPORTANT

This object is deprecated; use SecretsProvider instead. For more information, refer to Deprecation.

IG 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, IG 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.

Usage

```
{
  "secrets": {
```

```
    "stores": [ SecretStore reference, ... ]
  }
}
```

Properties

"stores": array of SecretStore references, required

One or more of the following secret stores:

- Base64EncodedSecretStore
- FileSystemSecretStore
- HsmSecretStore
- JwkSetSecretStore
- KeyStoreSecretStore
- SystemAndEnvSecretStore

Example

The following example configures two secret stores:

```
{
  "secrets": {
    "stores": [
       {
        "type": "FileSystemSecretStore",
        "config": {
          "directory": "/path/to/secrets",
          "format": "BASE64"
        }
      },
      {
        "type": "SystemAndEnvSecretStore",
        "config": {
          "format": "PLAIN"
        }
      }
    ]
  }
}
```

Base64EncodedSecretStore

Manage a repository of generic secrets, such as passwords or simple shared secrets, whose values are base64-encoded, and hard-coded in the route.

The secrets provider queries the Base64EncodedSecretStore for a named secret, identified by the `secret-id` in the `"secret-id": "string"` pair. The Base64EncodedSecretStore returns the matching secret.

The secrets provider builds the secret, checking that the secret's constraints are met, and returns a unique secret. If the secret's constraints are not met, the secrets provider cannot build the secret and the secret query fails.

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 Secrets.

Usage

```
{
  "name": string,
  "type": "Base64EncodedSecretStore",
  "config": {
    "secrets": map
  }
}
```

Properties

"secrets": map, required

Map of one or more secret ID/string pairs:

```
{
  "secrets": {
    "secret-id": "configuration expression<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']}"
    }
  }
}
```

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">
```

Example

For an example that uses Base64EncodedSecretStore, see `client-credentials.json` in Using OAuth 2.0 client credentials.

More information

Secrets

org.forgerock.openig.secrets.Base64EncodedSecretStore

FileSystemSecretStore

Manage a store of secrets held in files, specified as follows:

- Each file must contain only one secret.
- The file must be in the directory specified by the property `directory`.
- The filename must match the `mappings` property `secretId`.
- The file content must match the `mappings` property `format`. For example, if the mapping specifies `BASE64`, the file content must be base64-encoded.

Secrets are read lazily from the filesystem.

The secrets provider queries the FileSystemSecretStore for a named secret, identified by the name of a file in the specified directory, without the prefix/suffix defined in the store configuration. The FileSystemSecretStore returns the secret that exactly matches the name.

The secrets provider builds the secret, checking that the secret's constraints are met, and returns a unique secret. If the secret's constraints are not met, the secrets provider cannot build the secret and the secret query fails.

For a description of how secrets are managed, see Secrets.

Usage

```
{
  "name": string,
  "type": "FileSystemSecretStore",
  "config": {
    "directory": configuration expression<string>,
    "format": SecretPropertyFormat reference,
    "suffix": configuration expression<string>,
    "mappings": [ object, ... ],
    "leaseExpiry": configuration expression<duration>
```

```
    }
}
```

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: SecretPropertyFormat reference, optional

Format in which the secret is stored. Use one of the following values, or define a format:

- `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

"mappings": array of objects, optional

One or more mappings to define a secret:

secretId: configuration expression<secret-id>, required

The ID of the secret used in your configuration.

format: SecretPropertyFormat reference, required

The format and algorithm of the secret. Use SecretKeyPropertyFormat or PemPropertyFormat.

"leaseExpiry": configuration expression<duration>, optional

The amount of time that secrets produced by this store can be cached before they must be refreshed.

If the duration is `zero` or `unlimited` , IG issues a warning, and uses the default value.

Default: 5 minutes

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">
```

Example

For an example that uses FileSystemSecretStore, see Pass Runtime Data in a JWT Signed With a PEM.

More information

Secrets

org.forgerock.openig.secrets.FileSystemSecretStoreHeaplet

HsmSecretStore

Manage a store of secrets with a hardware security module (HSM) device or a software emulation of an HSM device, such as SoftHSM.

The secrets provider queries the HsmSecretStore for a named secret, identified by a secret ID and a stable ID, corresponding to the `secret-id` / `aliases` mapping. The HsmSecretStore returns a list of matching secrets.

The secrets provider builds the secret, checking that the secret's constraints are met, and returns a unique secret. If the secret's constraints are not met, the secrets provider cannot build the secret and the secret query fails.

For a description of how secrets are managed, see Secrets.

Usage

```
{
  "name": string,
  "type": "HsmSecretStore",
  "config": {
    "providerName": configuration expression<string>,
    "storePassword": configuration expression<secret-id>,
    "secretsProvider": SecretsProvider reference,
    "mappings": [ object, ... ],
    "leaseExpiry": configuration expression<duration>
```

```
    }
}
```

Properties

"providerName": configuration expression<string>, 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 how IG manages secrets, see Secrets.

"secretsProvider": SecretsProvider reference, optional

The SecretsProvider object to query for the `storePassword`.

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

"mappings": array of objects, required

One or more mappings of one secret ID to one or more aliases.

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.

aliases: array of configuration expression<strings>, required

One or more aliases for the secret ID.

"leaseExpiry": configuration expression<duration>, optional

The amount of time that secrets produced by this store can be cached before they must be refreshed.

If the duration is `zero` or `unlimited`, IG issues a warning, and uses the default value.

Default: 5 minutes

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">
```

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 IG.

   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 IG 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 IG JVM with the following system property that points to the provider config file:

```
-Djava.security.properties=file://path-to-security-extension-
file
```

The following example route is based on the examples in JwtBuilderFilter, replacing the KeyStoreSecretStore with an HsmSecretStore:

```
{
  "name": "hsm-jwt-signature",
  "condition": "${find(request.uri.path, '/hsm-jwt-signature$')}",
  "baseURI": "http://app.example.com:8081",
  "heap": [
    {
      "name": "SystemAndEnvSecretStore-1",
      "type": "SystemAndEnvSecretStore"
    },
    {
      "name": "AmService-1",
      "type": "AmService",
      "config": {
        "agent": {
          "username": "ig_agent",
          "passwordSecretId": "agent.secret.id"
        },
        "secretsProvider": "SystemAndEnvSecretStore-1",
        "url": "http://am.example.com:8088/openam",
        "version": "7.2"
      }
    },
    {
      "name": "HsmSecretStore-1",
      "type": "HsmSecretStore",
      "config": {
        "providerName": "SunPKCS11-SoftHSM",
        "storePassword": "hsm.pin",
        "secretsProvider": "SystemAndEnvSecretStore-1",
        "mappings": [{
          "secretId": "id.key.for.signing.jwt",
          "aliases": [ "signature-key" ]
        }]
      }
    }
  ],
  "handler": {
    "type": "Chain",
```

```
"config": {
  "filters": [{
    "name": "SingleSignOnFilter-1",
    "type": "SingleSignOnFilter",
    "config": {
      "amService": "AmService-1"
    }
  }, {
    "name": "UserProfileFilter-1",
    "type": "UserProfileFilter",
    "config": {
      "username": "${contexts.ssoToken.info.uid}",
      "userProfileService": {
        "type": "UserProfileService",
        "config": {
          "amService": "AmService-1"
        }
      }
    }
  }, {
    "name": "JwtBuilderFilter-1",
    "type": "JwtBuilderFilter",
    "config": {
      "template": {
        "name": "${contexts.userProfile.commonName}",
        "email": "${contexts.userProfile.rawInfo.mail[0]}"
      },
      "secretsProvider": "HsmSecretStore-1",
      "signature": {
        "secretId": "id.key.for.signing.jwt"
      }
    }
  }, {
    "name": "HeaderFilter-1",
    "type": "HeaderFilter",
    "config": {
      "messageType": "REQUEST",
      "add": {
        "x-openig-user": ["${contexts.jwtBuilder.value}"]
      }
    }
  }],
  "handler": "ReverseProxyHandler"
}
```

```
    }
}
```

More information

Secrets

org.forgerock.openig.secrets.HsmSecretStoreHeaplet

JwkSetSecretStore

Manages a secret store for JSON Web Keys (JWK) from a local or remote JWK Set.

The secrets provider queries the JwkSetSecretStore, as follows:

- If the JWT contains a `kid`, the secrets provider queries the JwkSetSecretStore for a named secret, identified by value of the `kid` of a JWK stored in the JwkSetSecretStore.
- If the JWT doesn't contain a `kid`, the secrets provider queries the JwkSetSecretStore for list of valid secrets, whose purpose matches the secret ID and any purpose contraints. The JwkSetSecretStore returns the secrets in the order that they are listed in the JWK set.

The secrets provider builds the secret, checking that the secret's constraints are met, and returns a unique secret. If the secret's constraints are not met, the secrets provider cannot build the secret and the secret query fails.

For a description of how secrets are managed, see Secrets.

For information about JWKs and JWK Sets, see SON Web Key (JWK).

Usage

```
{
  "name": string,
  "type": "JwkSetSecretStore",
  "config": {
    "jwkUrl": configuration expression<url>,
    "handler": Handler reference,
    "cacheTimeout": configuration expression<duration>,
    "cacheMissCacheTime": configuration expression<duration>,
    "leaseExpiry": configuration expression<duration>
  }
}
```

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`.

The cache cannot be deactivated. If a value lower than 10 seconds is configured, a warning is logged and the default value is used instead.

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

"leaseExpiry": configuration expression<duration>, optional

The amount of time that secrets produced by this store can be cached before they must be refreshed.

If the duration is `zero` or `unlimited`, IG issues a warning, and uses the default value.

Default: 5 minutes

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">
```

Example

For an example of how to set up and use JwkSetSecretStore to validate signed access tokens, see Validate signed access_tokens with the StatelessAccessTokenResolver and JwkSetSecretStore.

More information

org.forgerock.openig.secrets.JwkSetSecretStoreHeaplet

JSON Web Key (JWK)

KeyStoreSecretStore

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.

The secrets provider queries the KeyStoreSecretStore for a named secret, identified by a secret ID and a stable ID, corresponding to the `secret-id` / `aliases` mapping. The KeyStoreSecretStore returns a secret that exactly matches the name, and whose purpose matches the secret ID and any purpose contraints.

The secrets provider builds the secret, checking that the secret's constraints are met, and returns a unique secret. If the secret's constraints are not met, the secrets provider cannot build the secret and the secret query fails.

For a description of how secrets are managed, see Secrets.

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": [ object, ... ],
    "leaseExpiry": configuration expression<duration>
   }
}
```

Properties

"file": configuration expression<string>, required

The path to the KeyStore file.

"storeType": configuration expression<string>, optional

The secret store type.

"storePassword": configuration expression<secret-id>, required

The secret ID of the password to access the KeyStore.

IG searches for the value of the password until it finds it, first locally, then in parent routes, then in `config.json`.

To create a store password, add a file containing the password. The filename must corresponds to the secret ID, and the file content must contain only the password, with no trailing spaces or carriage returns.

"keyEntryPassword": configuration expression<secret-id>, optional

The secret ID of the password to access entries in the KeyStore.

To create an entry password, add a file containing the password. The filename must corresponds to the secret ID, and the file content must contain only the password, with no trailing spaces or carriage returns.

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.

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 Default secrets object.

"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.

`aliases`: array of configuration expression<strings>, required

One or more aliases for the secret ID.

`"leaseExpiry"`: configuration expression<duration>, optional

The amount of time that secrets produced by this store can be cached before they must be refreshed.

If the duration is `zero` or `unlimited`, IG issues a warning, and uses the default value.

Default: 5 minutes

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">
```

Example

For examples of routes that use KeyStoreSecretStore, see the examples in JwtBuilderFilter.

More information

org.forgerock.secrets.keystore.KeyStoreSecretStore

org.forgerock.openig.secrets.KeyStoreSecretStoreHeaplet

PemPropertyFormat

The format of a secret used with a mappings configuration in FileSystemSecretStore and SystemAndEnvSecretStore. Privacy-Enhanced Mail (PEM) is a file format for storing and sending cryptographic keys, certificates, and other data, based on standards in https://www.rfc-editor.org/rfc/rfc7468[Textual Encodings of PKIX, PKCS, and CMS Structures. By default, OpenSSL generates keys using the PEM format.

Encryption methods and ciphers used for PEM encryption must be supported by the Java Cryptography Extension.

PEM keys have the following format, where the PEM label is associated to the type of stored cryptographic material:

```
-----BEGIN {PEM label}-----
Base64-encoded cryptographic material
-----END {PEM label}-----
```

PEM Label	Stored Cryptographic Material
`CERTIFICATE`	X.509 Certificate
`PUBLIC KEY`	X.509 SubjectPublicKeyInfo
`PRIVATE KEY`	PKCS#8 Private Key
`ENCRYPTED PRIVATE KEY`	Encrypted PKCS#8 Private Key
`EC PRIVATE KEY`	EC Private Key
`RSA PRIVATE KEY`	PKCS#1 RSA Private Key
`RSA PUBLIC KEY`	PKCS#1 RSA Public Keys
`DSA PRIVATE KEY`	PKCS#1-style DSA Private Key
`HMAC SECRET KEY`	HMAC Secret Keys
`AES SECRET KEY`	AES Secret Keys
`GENERIC SECRET`	Generic Secrets (passwords, API keys, etc)

Note the following points about the key formats:

- PKCS#1 is the standard that defines RSA. For more information, see RFC 8017: RSA Public Key Syntax.

- PKCS#1-style DSA and EC keys are not defined in any standard, but are adapted from the RSA format.
- `HMAC SECRET KEY`, `AES SECRET KEY`, and `GENERIC SECRET` are a ForgeRock extension, and not currently supported by any other tools.

  The following example is non-standard PEM encoding of an HMAC symmetric secret key. The payload is base64-encoded random bytes that are the key material, with no extra encoding.

  ```
  -----BEGIN HMAC SECRET KEY-----
  Pj/Vel...thB0U=
  -----END HMAC SECRET KEY-----
  ```

  Run the following example command to create the key:

  ```
  cat <<EOF
  -----BEGIN HMAC SECRET KEY-----
  $(head -c32 /dev/urandom | base64)
  -----END HMAC SECRET KEY-----
  EOF
  ```

Usage

```
{
  "name": string,
  "type": "PemPropertyFormat",
  "config": {
   "decryptionSecretId": configuration expression<secret-id>,
   "secretsProvider": SecretsProvider reference
  }
}
```

Properties

"decryptionSecretId": configuration expression<secret-id>, optional

The secret ID for the secret to decrypt a PKCS#8 private key.

"secretsProvider": SecretsProvider reference, required when `decryptionSecretId` is used

The SecretsProvider object to query for the decryption secret. For more information, see SecretsProvider.

Example

For examples of use, see Pass Runtime Data in a JWT Signed With a PEM and Pass Runtime Data in a JWT Signed and Encrypted With a PEM.

More information

org.forgerock.openig.secrets.PemPropertyFormatHeaplet

SecretsKeyManager

Uses the Commons Secrets Service to manage keys that authenticate a TLS connection to a peer. The configuration references the keystore that holds the keys.

Usage

```
{
   "name": string,
   "type": "SecretsKeyManager",
   "config": {
     "signingSecretId": configuration expression<secret-id>,
     "secretsProvider": SecretsProvider reference
 }
}
```

Properties

"signingSecretId": configuration expression<secret-id>, required

The secret ID used to retrieve private signing keys.

"secretsProvider": SecretsProvider reference, required

The SecretsProvider to query for secrets to resolve the private signing key. For more information, see SecretsProvider.

Example

The following example uses a private key found from a keystore for TLS handshake.

```
{
  "type": "SecretsKeyManager",
  "config": {
    "signingSecretId": "key.manager.secret.id",
    "secretsProvider": {
```

```
      "type": "KeyStoreSecretStore",
      "config": {
        "file": "path/to/certs/ig.example.com.p12",
        "storePassword": "keystore.pass",
        "secretsProvider": "SecretsPasswords",
        "mappings": [{
          "secretId": "key.manager.secret.id",
          "aliases": [ "ig.example.com" ]
        }]
      }
    }
  }
}
```

More information

Secrets

org.forgerock.openig.secrets.SecretsKeyManagerHeaplet

SecretKeyPropertyFormat

The format of a secret used with a secret store.

Usage

```
{
  "name": string,
  "type": "SecretKeyPropertyFormat",
  "config": {
    "format": SecretPropertyFormat reference,
    "algorithm": configuration expression<string>
  }
}
```

Properties

`format`: SecretPropertyFormat *reference*, optional

Format in which the secret is stored. Use one of the following values, or define a format:

- `BASE64` : Base64-encoded
- `PLAIN` : Plain text

Default: `BASE64`

"algorithm": configuration expression<string>, required

The algorithm name used for encryption and decryption. Use algorithm names given in Java Security Standard Algorithm Names.

Example

```
{
  "type": "SecretKeyPropertyFormat",
  "config": {
    "format": "PLAIN",
    "algorithm": "AES"
  }
}
```

More information

Secrets

org.forgerock.openig.secrets.SecretKeyPropertyFormatHeaplet

SecretsProvider

Uses the specified secret stores to resolve queried secrets, such as passwords and cryptographic keys. Attempts to resolve the secret with the secret stores in the order that they are declared in the array.

Usage

```
{
  "name": string,
  "type": "SecretsProvider",
  "config": {
    "stores": [ SecretStore reference, ... ]
  }
}
```

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"
]
```

See Example for more example configurations.

Properties

"stores": array of SecretStore references, required

One or more secret stores to provide access to stored secrets. Configure secret stores described in Secrets.

Example

The following SecretsProvider is used in Discover and Dynamically Register With OpenID Connect Providers.

```
"secretsProvider": {
  "type": "SecretsProvider",
```

```
  "config": {
    "stores": [
      {
        "type": "KeyStoreSecretStore",
        "config": {
          "file": "/path/to/keystore.jks",
          "mappings": [
            {
              "aliases": [ "myprivatekeyalias" ],
              "secretId": "private.key.jwt.signing.key"
            }
          ],
          "storePassword": "keystore.secret.id",
          "storeType": "JKS",
          "secretsProvider": "SystemAndEnvSecretStore-1"
        }
      }
    ]
  }
}
```

The following example shows the equivalent SecretsProvider configuration with an inline compact format:

```
"secretsProvider": {
  "name": "KeyStoreSecretStore-1",
  "type": "KeyStoreSecretStore",
  "config": {
    "file": "/path/to/keystore.jks",
    "mappings": [
      {
        "aliases": [ "myprivatekeyalias" ],
        "secretId": "private.key.jwt.signing.key"
      }
    ],
    "storePassword": "keystore.secret.id",
    "storeType": "JKS"
  }
}
```

The following example shows the equivalent SecretsProvider configuration with a compact format, referencing a KeyStoreSecretStore object in the heap:

```
"secretsProvider": "KeyStoreSecretStore-1"
```

More information

StatelessAccessTokenResolver

Secrets

org.forgerock.secrets.SecretsProvider

SecretsTrustManager

Uses the Commons Secrets Service to manage trust material that verifies the credentials presented by a peer. Trust material is usually public key certificates. The configuration references the secrets store that holds the trust material.

Usage

```
{
  "name": string,
  "type": "SecretsTrustManager",
  "config": {
    "verificationSecretId": configuration expression<secret-id>,
    "secretsProvider": SecretsProvider reference,
    "checkRevocation": configuration expression<boolean>
  }
}
```

Properties

"verificationSecretId": configuration expression<secret-id>, required

The secret ID to retrieve trusted certificates.

"secretsProvider": SecretsProvider reference, required

The SecretsProvider to query for secrets to resolve trusted certificates. For more information, see SecretsProvider.

"checkRevocation": configuration expression<boolean>, optional

Specifies whether to check for certificate revocation.

Default: `true`

Example

The following example trusts a list of certificates found in a given keystore:

```
{
  "type": "SecretsTrustManager",
  "config": {
    "verificationSecretId": "trust.manager.secret.id",
    "secretsProvider": {
      "type": "KeyStoreSecretStore",
      "config": {
        "file": "path/to/certs/truststore.p12",
        "storePassword": "keystore.pass",
        "secretsProvider": "SecretsPasswords",
        "mappings": [{
          "secretId": "trust.manager.secret.id",
          "aliases": [ "alias-of-trusted-cert-1", "alias-of-
trusted-cert-2" ]
        }]
      }
    }
  }
}
```

More information

Secrets

org.forgerock.openig.secrets.SecretsTrustManagerHeaplet

SystemAndEnvSecretStore

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`.

The secrets provider queries the SystemAndEnvSecretStore for a named secret, identified by the name of a system property or environment variable. The SystemAndEnvSecretStore returns a secret that exactly matches the name.

The secrets provider builds the secret, checking that the secret's constraints are met, and returns a unique secret. If the secret's constraints are not met, the secrets provider cannot build the secret and the secret query fails.

For a description of how secrets are managed, see Secrets.

Usage

```
{
  "name": string,
  "type": "SystemAndEnvSecretStore",
  "config": {
    "format": SecretPropertyFormat reference,
    "mappings": [ object, ... ],
    "leaseExpiry": configuration expression<duration>
    }
}
```

Properties

`format`: SecretPropertyFormat *reference*, optional

Format in which the secret is stored. Use one of the following values, or define a format:

- `BASE64` : Base64-encoded
- `PLAIN` : Plain text

Default: `BASE64`

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

One or more mappings to define a secret:

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

The ID of the secret used in your configuration.

`format`: SecretPropertyFormat *reference*, required

The format and algorithm of the secret. Use SecretKeyPropertyFormat or PemPropertyFormat.

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

The amount of time that secrets produced by this store can be cached before they must be refreshed.

If the duration is `zero` or `unlimited` , IG issues a warning, and uses the default value.

Default: 5 minutes

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">
```

Example

For an example of how to uses a SystemAndEnvSecretStore to manage a password, see the example in Authenticate with SSO through the default authentication service

More information

Secrets

org.forgerock.openig.secrets.SystemAndEnvSecretStoreHeaplet

Supported standards

IG 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 and Authorization Grants
- 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)

RFC 8705: OAuth 2.0 Mutual-TLS Client Authentication and Certificate-Bound Access Tokens

OpenID Connect 1.0

IG 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, IG supports *Normal Claims*. The optional *Aggregated Claims* and *Distributed Claims* representations are not supported by IG.

- 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. IG's APIs are RESTful APIs.

Security Assertion Markup Language (SAML)

Standard, XML-based framework for implementing a SAML service provider. IG supports multiple versions of SAML including 2.0, 1.1, and 1.0.

Specifications are available from the OASIS standards page.

Other Standards

RFC 4627: The application/json Media Type for JavaScript Object Notation (JSON). JSON text is encoded with Unicode; IG reads and stores JSON as Unicode.

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.

Internationalization

IG supports internationalization (i18n) to facilitate localization for target audiences that vary in culture, region, or language.

Information type	Character set/encoding
HTTP header names and values	US-ASCII
HTTP trailer names and values	US-ASCII
Response entities for StaticResponseHandler	The `Content-Type` header must be set. For text content, the character set must also be specified; for example: • `Content-Type: text/html; charset=utf-8` • `Content-Type: text/plain; charset=utf-8` The entity must conform to the content type.
Text in request and response entities for CaptureDecorator	If the `Content-Type` header is set for the request or response, the decorator uses it to decode the text in request or response messages, and then writes them to the logs. If the `Content-Type` header is not set, the decorator does not write the request or response messages to the logs.

Information type	Character set/encoding
Logs	The system default character set where IG is running. To use a different character set, configure `logback.xml` as described in Change the character set and format of log messages.
IG configuration files	UTF-8
Hostnames	US-ASCII Non US-ASCII characters must be escaped with Punycode encoding.
URIs	US-ASCII Non US-ASCII and reserved characters must be escaped with percent-encoding.

Was this helpful?