Configuration Reference

Paul Bryan, Mark Craig, Jamie Nelson, Joanne Henry

Table of Contents

Reference documentation for OpenIG. OpenIG provides a high-performance reverse proxy server with specialized session management and credential replay functionality.

Preface

This reference covers OpenIG configuration.

Who Should Use this Reference

This reference is for OpenIG designers, developers, and administrators.

For API specifications, see the appropriate Javadoc.

Reserved Routes

OpenIG reserves all paths starting with `/openig` for administrative use.

Resources exposed under `/openig` are only accessible to local client applications.

Reserved Field Names

OpenIG 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

OpenIG configuration uses JSON notation.

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

array

JSON array.

boolean

Either `true` or `false`.

configuration expression

Expression for which no context is available.

A configuration expression, described in Expressions(5) is independent of the request, response, and contexts, so do not use expressions that reference their properties. You can, however, use `${env['variable']}`, `${system['property']}`, and all the built-in functions listed in Functions(5).

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.

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`: microseconds
- `nanoseconds`, `nanosecond`, `nanosec`, `nanos`, `nano`, `ns`: nanoseconds

expression

See Expressions(5).

lvalue-expression

Expression yielding an object whose value is to be set.

number

JSON number.

object

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

pattern

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

pattern-template

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

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

reference

Either references an object configured in the heap by the object's name or uses a local, inline configuration object where the name is optional.

string

JSON string.

About ForgeRock Common REST

For many REST APIs that are not defined by external standards, ForgeRock products provide common ways to access web resources and collections of resources. This section covers what is

common across products. Adapt the examples to your types of resources and to your deployment.

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`
- `_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.

Create

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

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

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

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

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

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

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

You can use the following parameters:

`_prettyPrint=true`

Format the body of the response.

`_fields=field[,field…]`

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

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

Read

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

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

Parameters

You can use the following parameters:

`_prettyPrint=true`

Format the body of the response.

`_fields=field[,field…]`

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

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

`_mimeType=mime-type`

Some resources have fields whose values are multi-media resources such as a profile photo for example.

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

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

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

Update

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

JSON response:

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

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

You can use the following parameters:

`_prettyPrint=true`

Format the body of the response.

`_fields=field[,field⋯]`

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

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

Delete

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

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

Parameters

You can use the following parameters:

`_prettyPrint=true`

Format the body of the response.

`_fields=field[,field⋯]`

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

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

Patch

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

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

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

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

PATCH operations apply to three types of targets:

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

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

Patch Operation: Add

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

If the target field is single-valued, then the value you include in the PATCH replaces the value of the target. Examples of a single-valued field include: object, string, boolean, or number. An `add` operation has different results on two standard types of arrays:

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

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.

The following `copy` operation takes the value from the source named `/hot/potato`, and then runs a `replace` operation on the target value, `/hot/tamale`.

```
[
  {
    "operation" : "copy",
    "field" : "/hot/potato",
    "value" : "/hot/tamale"
  }
]
```

If the source and value are configured as arrays, the result depends on whether the array has list semantics or set semantics, as described in Patch Operation: Add.

Patch Operation: Increment

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

```
[
  {
    "operation" : "increment",
    "field" : "/user/payment",
    "value" : "1000"
  }
]
```

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

Patch Operation: Move

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

The following `move` operation is equivalent to a `remove` operation on the source named `/hot/potato`, followed by a `replace` operation on the target value, `/hot/tamale`.

```
[
  {
    "operation" : "move",
    "field" : "/hot/potato",
    "value" : "/hot/tamale"
  }
]
```

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

Patch Operation: Remove

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

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

If the object has more than one `phoneNumber`, those values are stored as an array. A `remove` operation has different results on two standard types of arrays:

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

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

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

Patch Operation: Replace

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

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

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

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

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

Apply the following operations on that resource:

```
[
  {
    "operation" : "remove",
    "field" : "/fruits/0",
    "value" : ""
```

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

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

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

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

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

Patch Operation: Transform

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

```
[
  {
    "operation" : "transform",
    "field" : "/objects",
    "value" : {
      "script" : {
        "type" : "text/javascript",
        "file" : "something.js"
      }
    }
  },
]
```

Patch Operation Limitations

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

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

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

`_prettyPrint=true`

Format the body of the response.

`_fields=field[,field…]`

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

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

Action

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

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

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

`_prettyPrint=true`

Format the body of the response.

`_fields=field[,field…]`

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

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

Query

To query a resource collection (or resource container if you prefer to think of it that way), perform an HTTP GET and accept a JSON response, including at least a `_queryExpression`, `_queryFilter`, or `_queryId` parameter. These parameters cannot be used together:

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

The server returns the result as a JSON object including a "results" array and other fields related to the query string parameters that you specify. .Parameters

You can use the following parameters:

`_queryFilter=filter-expression`

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

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

```
Expr           = OrExpr
OrExpr         = AndExpr ( 'or' AndExpr ) *
AndExpr        = NotExpr ( 'and' NotExpr ) *
NotExpr        = '!' PrimaryExpr | PrimaryExpr
PrimaryExpr    = '(' Expr ')' | ComparisonExpr | PresenceExpr | LiteralExpr
ComparisonExpr = Pointer OpName JsonValue
PresenceExpr   = Pointer 'pr'
LiteralExpr    = 'true' | 'false'
Pointer        = JSON pointer
OpName         = 'eq' |  # equal to
                 'co' |  # contains
                 'sw' |  # starts with
                 'lt' |  # less than
                 'le' |  # less than or equal to
                 'gt' |  # greater than
                 'ge' |  # greater than or equal to
                 STRING  # extended operator
JsonValue      = NUMBER | BOOLEAN | '"' UTF8STRING '"'
STRING         = ASCII string not containing white-space
UTF8STRING     = UTF-8 string possibly containing white-space
```

Note that white space, double quotes ("), parentheses, and exclamation characters need URL encoding in HTTP query strings.

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.

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

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

`_queryId=identifier`

Specify a query by its identifier.

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

`_pagedResultsCookie=string`

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

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

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

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

`_pagedResultsOffset=integer`

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

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

`_pageSize=integer`

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

`_totalPagedResultsPolicy=string`

When a `_pageSize` is specified, and non-zero, the server calculates the "totalPagedResults", in accordance with the `totalPagedResultsPolicy`, and provides the value as part of the response. The "totalPagedResults" is either an estimate of the total number of paged results (`_totalPagedResultsPolicy=ESTIMATE`), or the exact total result count (`_totalPagedResultsPolicy=EXACT`). If no count policy is specified in the query, or if `_totalPagedResultsPolicy=NONE`, result counting is disabled, and the server returns value of -1 for "totalPagedResults".

`_sortKeys=[-]field[,[-]field…]`

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

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

`_prettyPrint=true`

Format the body of the response.

`_fields=field[,field…]`

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

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

HTTP Status Codes

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

200 OK

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

201 Created

The request succeeded and the resource was created.

204 No Content

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

304 Not Modified

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

400 Bad Request

The request was malformed.

401 Unauthorized

The request requires user authentication.

403 Forbidden

Access was forbidden during an operation on a resource.

404 Not Found

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

405 Method Not Allowed

The HTTP method is not allowed for the requested resource.

406 Not Acceptable

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

409 Conflict

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

410 Gone

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

412 Precondition Failed

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

415 Unsupported Media Type

The request is in a format not supported by the requested resource for the requested method.

428 Precondition Required

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

500 Internal Server Error

The server encountered an unexpected condition that prevented it from fulfilling the request.

501 Not Implemented

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

503 Service Unavailable

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

Formatting Conventions

Most examples in the documentation are created in GNU/Linux or Mac OS X operating environments. If distinctions are necessary between operating environments, examples are labeled with the operating environment name in parentheses. To avoid repetition file system directory names are often given only in UNIX format as in `/path/to/server` , even if the text applies to `C:\path\to\server` as well. Absolute path names usually begin with the placeholder `/path/to/` . This path might translate to `/opt/` , `C:\Program Files\` , or somewhere else on your system. Command-line, terminal sessions are formatted as follows:

```
$ echo $JAVA_HOME
/path/to/jdk
```

Command output is sometimes formatted for narrower, more readable output even though formatting parameters are not shown in the command. Program listings are formatted as follows:

```
class Test {
    public static void main(String [] args)  {
        System.out.println("This is a program listing.");
    }
}
```

Accessing Documentation Online

Open Identity Platform Community publishes comprehensive documentation online:

- The Open Identity Platform Community Documentation offers a large and increasing number of up-to-date, practical articles that help you deploy and manage Open Identity Platform software.
- Open Identity Platform product documentation, such as this document, aims to be technically accurate and complete with respect to the software documented. It is visible to everyone and covers all product features and examples of how to use them.

Joining the Open Identity Platform Community

Visit the community resource center where you can find information about each project, download nightly builds, browse the resource catalog, ask and answer questions on the forums, find community events near you, and of course get the source code as well.

Getting Support and the Contacting Open Identity Platform Community

Open Identity Platform Community Approved Vendors provide support services, professional services, trainings, and partner services to assist you in setting up and maintaining your deployments.

Required Configuration

You must specify at least the entry point for incoming requests, the OpenIG Servlet, and the heap objects that configure and initialize objects, with dependency injection.

GatewayHttpApplication — configure OpenIG

Description

The GatewayHttpApplication is the entry point for all incoming requests. It is responsible for initializing a heap of objects, described in Heap Objects(5), and providing the main Handler that receives all the incoming requests. The configuration is loaded from a JSON-encoded configuration file, expected by default at `$HOME/.openig/config/config.json`. The GatewayHttpApplication creates the following objects by default:

- An AuditDecorator that you can use to trigger notification for audit events. The default AuditDecorator is named audit. For details, see AuditDecorator(5).
- A BaseUriDecorator that you can use to override the scheme, host, and port of the existing request URI. The default BaseUriDecorator is named baseURI. For details, see BaseUriDecorator(5).
- A CaptureDecorator that you can use to capture requests and response messages. The default CaptureDecorator is named capture. For details, see CaptureDecorator(5).
- A TimerDecorator that you can use to record time spent within Filters and Handlers. The default TimerDecorator is named timer. For details, see TimerDecorator(5).

The GatewayHttpApplication declares default configurations in the heap for the following objects:

- A ClientHandler named `ClientHandler` for communicating with protected applications. For details, see ClientHandler(5).
- A ClientHandler named `ForgeRockClientHandler` for sending a ForgeRock Common Audit transaction ID when communicating with protected applications. The default object wraps the `ClientHandler`.

The GatewayHttpApplication also looks for an object named Session in the heap. If it finds such an object, it uses that object as the default session producer. For example, to store session information in an HTTP cookie on the user-agent, you can define a JwtSession named Session in `config.json`. If you do that, however, stored session information must fit the constraints for storage in a JWT and in a cookie, as described in JwtSession(5). If no such object is found, `session` is based on the Servlet HttpSession that is handled by the container where OpenIG runs.

Usage

```
{
  "handler": Handler reference or inline Handler declaration,
  "heap": [ configuration object, ... ],
  "logSink":  LogSink reference,
```

```
  "temporaryStorage": TemporaryStorage reference
}
```

Properties

"handler": *Handler reference, required*

Dispatch all requests to this handler.

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

See also Handlers.

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

The heap object configuration, described in Heap Objects(5).

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

"logSink": *LogSink reference, optional*

Send log messages to this LogSink.

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

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

"temporaryStorage": *TemporaryStorage reference, optional*

Cache content during processing based on this TemporaryStorage configuration.

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

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

See also TemporaryStorage(5).

Javadoc

org.forgerock.openig.http.GatewayHttpApplication

Heap Objects — configure and initialize objects, with dependency injection

Description

A heap is a collection of associated objects, initialized from declarative configuration artifacts. All configurable objects in OpenIG are heap objects. Heap objects are created and initialized by associated heaplets, which retrieve any objects an object depends on from the heap. The heap configuration is included as an object in the GatewayHttpApplication configuration, as described in GatewayHttpApplication(5).

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.

Automatically Created Objects

OpenIG automatically creates some configuration objects that it needs for its own use. An automatically created object can be overridden by creating a heap object with the same name. Automatically created objects include the following:

`"ApiProtectionFilter"`

The default filter used to protect administrative APIs on reserved routes. Reserved routes are described in Reserved Routes.

Default: a filter that allows access only from the loopback address.

To override this filter, declare a different filter with the same name in the top-level heap found in `config.json`.

`"LogSink"`

The default object to use for writing all audit and performance logging.

Default: A ConsoleLogSink object named "LogSink" with the default configuration is added to the top-level heap.

Routes can use this object without explicitly defining it. To override this object, create a LogSink heap object with the same name.

See also ConsoleLogSink(5).

`"TemporaryStorage"`

The default object to use for managing temporary buffers.

Default: a TemporaryStorage object named "TemporaryStorage" with the default configuration is added to the top-level heap.

Routes can use this object without explicitly defining it. To override this object, create a TemporaryStorage heap object with the same name.

See also TemporaryStorage(5).

Implicit Properties

Every heap object has a set of implicit properties, which can be overridden on an object-by-object basis:

`"logSink"`: ***string***

Specifies the heap object that should be used for audit and performance logging.

Default: `LogSink`.

`"temporaryStorage"`: ***string***

Specifies the heap object that should be used for temporary buffer storage.

Default: `TemporaryStorage`.

Configuration Settings — configure objects

Description

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

Expressions can retrieve the values for configuration settings from system properties or environment variables. When OpenIG 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 OpenIG 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 OpenIG starts up or when the route is reloaded. If the configuration setting is optional, there is no error.

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

```
{
  "handler": {
    "type": "Chain",
    "config": {
      "filters": [
        {
          "type": "ThrottlingFilter",
          "config": {
            "requestGroupingPolicy": "${request.headers['UserId'][0]}",
            "rate": {
              "numberOfRequests": "${integer(system["requestsPerSecond"])}",
              "duration": "10 seconds"
            }
          }
        }
      ],
      "handler": "ClientHandler"
    }
  },
  "monitor" : "${boolean(env["ENABLE_MONITORING"])}",
  "condition": "${matches(request.uri.path, '^/throttle-simple')}"
}
```

If `"requestsPerSecond"=150` and `"ENABLE_MONITORING"=false`, after the expressions are evaluated OpenIG views the example route as follows:

```
{
  "handler": {
```

```
    "type": "Chain",
    "config": {
      "filters": [
        {
          "type": "ThrottlingFilter",
          "config": {
            "requestGroupingPolicy": "${request.headers['UserId'][0]}",
            "rate": {
              "numberOfRequests": 150,
              "duration": "10 seconds"
            }
          }
        }
      ],
      "handler": "ClientHandler"
    }
  },
  "monitor" : false,
  "condition": "${matches(request.uri.path, '^/throttle-simple')}"
}
```

For information about expressions, see Expressions(5).

Handlers

Handler objects process an HTTP request by producing an associated response.

Chain — dispatch the request to ordered list of filters and finally a handler

Description

A chain is responsible for dispatching a request to an ordered list of filters, and finally a handler.

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 once the request has traversed all of the specified filters.

See also Handlers.

Example

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

```
        "handler": "ClientHandler"
    }
}
```

Javadoc

org.forgerock.openig.filter.Chain

ClientHandler — submit requests to remote servers

Description

Submits requests to remote servers.

Usage

```
{
    "name": string,
    "type": "ClientHandler",
    "config": {
        "connections": number,
        "disableReuseConnection": boolean,
        "disableRetries": boolean,
        "hostnameVerifier": string,
        "soTimeout": duration string,
        "connectionTimeout": duration string,
        "numberOfWorkers": number,
        "sslCipherSuites": array,
        "sslContextAlgorithm": string,
        "sslEnabledProtocols": array,
        "keyManager": KeyManager reference(s),
        "trustManager": TrustManager reference(s),
    }
}
```

Properties

`"connections"`: ***number, optional***

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

Default: 64

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

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

A duration is a lapse of time expressed in English, such as `23 hours 59 minutes and 59 seconds`.

Durations are not case sensitive.

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`: microseconds
- `nanoseconds`, `nanosecond`, `nanosec`, `nanos`, `nano`, `ns`: nanoseconds

Default: 10 seconds

`"disableRetries"`: ***boolean, optional***

Whether to disable automatic retries for failed requests.

Default: `false`

`"disableReuseConnection"`: ***boolean, optional***

Whether to disable connection reuse.

Default: `false`

`"hostnameVerifier"`: ***string, optional***

How to handle hostname verification for outgoing SSL connections.

Set this to one of the following values:

- `ALLOW_ALL`: turn off verification.
- `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: `ALLOW_ALL`

`"numberOfWorkers"`: ***number, optional***

The number of worker threads dedicated to processing outgoing requests.

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

Default: One thread per CPU available to the JVM.

`"keyManager"`: ***KeyManager reference(s), optional***

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.

See also KeyManager(5).

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

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

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`: microseconds
- `nanoseconds`, `nanosecond`, `nanosec`, `nanos`, `nano`, `ns`: nanoseconds

Default: 10 seconds

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

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 OpenIG. For Oracle Java, see the list of

JSSE Cipher Suite Names.

Default: Allow any cipher suite supported by the JVM.

`"sslContextAlgorithm"`: ***string, optional***

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

Default: `TLS`

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

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 OpenIG. For Oracle Java, see the list of Additional JSSE Standard Names.

Default: Allow any protocol supported by the JVM.

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

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`.

See also TrustManager(5).

Example

The following object configures a `ClientHandler` named `Client`, with non-default security settings:

```
{
  "name": "Client",
  "type": "ClientHandler",
  "config": {
        "hostnameVerifier": "STRICT",
        "sslContextAlgorithm": "TLSv1.2",
        "keyManager": {
            "type": "KeyManager",
            "config": {
                "keystore": {
```

```
                    "type": "KeyStore",
                    "config": {
                        "url": "file://${env['HOME']}/keystore.jks",
                        "password": "${system['keypass']}"
                    }
                },
                "password": "${system['keypass']}"
            }
        },
        "trustManager": {
            "type": "TrustManager",
            "config": {
                "keystore": {
                    "type": "KeyStore",
                    "config": {
                        "url": "file://${env['HOME']}/truststore.jks",
                        "password": "${system['trustpass']}"
                    }
                }
            }
        }
    }
}
```

Javadoc

org.forgerock.openig.handler.ClientHandler

DesKeyGenHandler — generate a DES key

Description

Generates a DES key for use with OpenAM as described in Configuring Password Capture in the *Gateway Guide*.

Usage

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

Javadoc

org.forgerock.openig.handler.DesKeyGenHandler

DispatchHandler — dispatch to one of a list of handlers

Description

Dispatches to one of a list of handlers. When a request is handled, each handler's `condition` is evaluated. If a condition expression yields `true`, then the request is dispatched to the associated handler with no further processing.

Usage

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

Properties

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

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

`"condition"`: ***expression, optional***

Condition to evaluate to determine if associated handler should be dispatched to. If omitted, then dispatch is unconditional.

See also Expressions(5).

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

Dispatch to this handler if the associated condition yields `true`.

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

See also Handlers.

`"baseURI"`: ***string, optional***

Overrides the existing request URI, making requests relative to a new base URI. Only scheme,

host and port are used in the supplied URI.

Default: leave URI untouched.

Example

The following sample is from a SAML 2.0 federation configuration. If the incoming URI starts with `/saml`, then OpenIG dispatches to a SamlFederationHandler. If the user name is not set in the session context, then the user has not authenticated with the SAML 2.0 Identity Provider, so OpenIG dispatches to a SPInitiatedSSORedirectHandler to initiate SAML 2.0 SSO from the Service Provider, which is OpenIG. All other requests go through a LoginChain handler:

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

Javadoc

org.forgerock.openig.handler.DispatchHandler

MonitorEndpointHandler — return basic audit statistics in JSON format

Description

This handler collates basic audit statistics, returning them in JSON format.

Interface Stability: Deprecated (For details, see ForgeRock Product Interface Stability.)

You decorate the objects to audit by adding your own audit tags. The handler updates the count of messages in progress, completed, and internal errors for each audit event, initializing the counts at OpenIG startup time. When accessed, it returns the sums organized by object under audit using the tags that you defined.

Usage

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

Example

The following sample route adds a monitor endpoint at `/monitor`:

```
{
    "handler": {
        "type": "MonitorEndpointHandler"
    },
    "condition": "${request.method == 'GET'
                    and request.uri.path == '/monitor'}"
    "audit": "Monitor route"
}
```

After adding audit tags to a number of other routes, the JSON returned from the monitor endpoint shows statistics since OpenIG started. The following example is formatted for legibility:

```
{
    "ForgeRock.com route": {
        "in progress": 0,
        "completed": 6,
        "internal errors": 0
    },
    "ForgeRock.org route": {
        "in progress": 0,
        "completed": 15,
        "internal errors": 0
    },
    "Monitor route": {
        "in progress": 1,
        "completed": 1,
        "internal errors": 0
    },
    "Static login route": {
        "in progress": 0,
        "completed": 12,
```

```
        "internal errors": 0
    },
    "HTTP Basic route": {
        "in progress": 0,
        "completed": 21,
        "internal errors": 3
    }
}
```

Javadoc

org.forgerock.openig.audit.monitor.MonitorEndpointHandler

Route — Configuration for handling a specified request

Description

In OpenIG, a route is represented by a separate JSON configuration file and that handles a request, described in Request(5), and context, described in Contexts(5), when a specified condition is met.

A top-level Router, as described in Router(5), is responsible for reloading the route configuration. Use a Router to call route handlers, rather than calling a route directly as the handler of the top-level configuration. By default the Router rereads the configurations periodically, so that configuration changes to routes apply without restarting OpenIG.

Each separate route has its own Heap of configuration objects. The route's Heap inherits from its parent Heap, which is the global heap for top-level routes, so the route configuration can reference configuration objects specified in the top-level Router configuration file.

For examples of route configurations see Configuring Routes in the *Gateway Guide*.

Usage

```
{
    "handler": Handler reference or inline Handler declaration,
    "heap": [ configuration object, ... ],
    "condition": expression,
    "monitor": boolean expression OR object,
    "name": string,
    "session": Session reference
}
```

Properties

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

For this route, dispatch the request to this handler.

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

See also Handlers.

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

Heap object configuration for objects local to this route.

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

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

See also Heap Objects(5).

`"condition"`: *expression, optional*

Whether the route accepts to handle the request.

Default: If the condition is not set, or is `null`, then this route accepts any request.

All paths starting with `/openig` are reserved for administrative use by OpenIG. Expressions such as the following never match externally configured routes: `${matches(request.uri.path, '^/openig/my/path')}`. In effect, such routes are ignored.

See also Expressions(5).

`"monitor"`: *boolean expression OR object, optional*

This property lets you specify whether to maintain statistics about the route, an optionally to specify the percentiles in the distribution for which to record response times.

Use a boolean or boolean expression to activate monitoring with the default percentiles configuration. When the boolean expression resolves to `true`, statistics for the route are exposed over REST as described in "The REST API for Monitoring".

Default: `false` (with percentiles `0.999`, `0.9999`, and `0.99999`)

Use an object instead of a boolean to specify percentiles:

```
{
    "monitor": {
        "enabled": boolean expression OR boolean,
        "percentiles": array of numbers
    }
}
```

The configuration object fields include the following:

`"enabled"`: ***boolean expression, required***

Whether to maintain statistics about the route, as described above.

`"percentiles"`: ***array of decimal numbers, optional***

The percentiles in the distribution for which to maintain response time statistics. If you specify percentiles, only those percentiles are used. The default percentile settings no longer apply.

Each value in the array is a decimal representation of a percentage. For example, `0.999` represents 99.9%.

The statistic maintained for a percentile is the response time in milliseconds after which *percentile* of responses were sent. For example, the statistic for `0.999` corresponds to the response time in milliseconds after which 99.9% of responses were sent. The statistic for `0.5` corresponds to the response time in milliseconds after which half of all responses were sent.

Default: `[ 0.999, 0.9999, 0.99999 ]`

`"name"`: ***string, optional***

Name for the route, used by the Router to order the routes.

Default: Route configuration file name

`"session"`: ***Session reference, optional***

Session storage implementation used by this route, such as a JwtSession as described in JwtSession(5).

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

Default: do not change the session storage implementation for `session`.

The REST API for Monitoring

When the route has `"monitor": "${true}"`, monitoring statistics are exposed at a registered endpoint. OpenIG logs the paths to registered endpoints when the log level is `INFO` or finer. Look for messages such as the following in the log:

```
Monitoring endpoint available at
 '/openig/api/system/objects/router-handler/routes/00-monitor/monitoring'
```

To access the endpoint over HTTP or HTTPS, prefix the path with the OpenIG scheme, host, and port to obtain a full URL, such as `http://localhost:8080/openig/api/system/objects/router-handler/routes/00-monitor/monitoring`.

The monitoring REST API supports only read (HTTP GET). For a detailed introduction to common REST APIs, see About ForgeRock Common REST.

In the present implementation, OpenIG does not have mechanisms for resetting or for persisting monitoring statistics. When you set `"monitor": true` on the route, or when you start the OpenIG container, monitoring statistics are collected. When the OpenIG container stops, monitoring statistics are discarded.

A JSON monitoring resource with the default percentiles has the following form. Field values are described in comments:

```
{
    "requests": {
        "total": number,              // Total requests
        "active": number              // Requests being processed
    },
    "responses": {
        "total": number,              // Total responses
        "info": number,               // Informational responses (1xx)
        "success": number,            // Successful responses    (2xx)
        "redirect": number,           // Redirection responses   (3xx)
        "clientError": number,        // Client error responses  (4xx)
        "serverError": number,        // Server error responses  (5xx)
        "other": number,              // Responses with status code >= 600
        "errors": number,             // An exception was thrown.
        "null": number                // Responses not handled by OpenIG
    },
    "throughput": {                   // Responses per second
        "mean": number,               // Mean (average) since monitoring started
        "lastMinute": number,         // One-minute moving average rate
        "last5Minutes": number,       // Five-minute moving average rate
        "last15Minutes": number       // 15-minute moving average rate
    },
    "responseTime": {                 // Response times in milliseconds
        "mean": number,               // Mean (average) response time
        "median": number,             // Median response time
        "standardDeviation": number,  // Std. dev. for response time
        "total": number,              // Cumulative resp. processing time
        "percentiles": {              // Response times in ms after which:
            "0.999": number,          // 99.9% of responses were sent
            "0.9999": number,         // 99.99% of responses were sent
            "0.99999": number         // 99.999% of responses were sent
        }
    }
}
```

TIP

When reading percentiles, use map notation. The keys start with a digit, and so are not suitable for use with dot notation, as shown in the following example:

```
threeNines = responseTime.percentiles['0.999']  // Correct
threeNines = responseTime.percentiles.0.999     // Wrong: syntax error
```

The JSON resource is written from a live object. As a result, field values can appear as inconsistent. For example, the sum of responses and in-flight requests might be different from the count of all requests. Counters can change as the JSON representation of the object is written.

Router — Route processing to distinct configurations

Description

A Router is a handler that routes request processing to separate configuration files. Each separate configuration file then defines a Route, as described in Route(5).

The Router reloads configuration files for Routes from the specified directory at the specified scan interval.

Usage

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

An alternative value for type is RouterHandler.

Properties

"defaultHandler": *Handler reference, optional*

Default handler for this Router.

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

The router routes the request to the first route whose condition expression is satisfied. If no route condition matches, then the request is routed to the default handler if one is configured.

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

See also Handlers.

"directory": *expression, optional*

Base directory from which to load configuration files for routes.

Default: default base directory for route configuration files. For details, see Installing OpenIG in the *Gateway Guide*.

IMPORTANT	If you define a new Router in the default base directory, then you must set the directory property to a different directory from the default base directory in order to avoid a circular reference to the new Router.

See also Expressions(5).

`"scanInterval"`: ***integer, optional***

Interval in seconds after which OpenIG scans the specified directory for changes to configuration files.

Default: 10 (seconds)

To prevent OpenIG from reloading Route configurations after you except at startup, set the scan interval to -1.

Javadoc

org.forgerock.openig.handler.router.RouterHandler

SamlFederationHandler — play the role of SAML 2.0 Service Provider

Description

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

NOTE	This handler does not support filtering. Specifically, do not use this as the handler for a Chain, which can include filters. More generally, do not use this handler when its use depends on something in the response. The response can be handled independently of OpenIG, and can be `null` when control returns to OpenIG. For example, do not use this handler in a `SequenceHandler` where the `postcondition` depends on the response.

Usage

```
{
    "name": string,
    "type": "SamlFederationHandler",
    "config": {
        "assertionMapping": object,
        "redirectURI": string,
        "assertionConsumerEndpoint": string,
```

```
        "authnContext": string,
        "authnContextDelimiter": string,
        "logoutURI": string,
        "sessionIndexMapping": string,
        "singleLogoutEndpoint": string,
        "singleLogoutEndpointSoap": string,
        "SPinitiatedSLOEndpoint": string,
        "SPinitiatedSSOEndpoint": string,
        "subjectMapping": string
    }
}
```

Properties

"assertionMapping": *object, required*

The assertionMapping defines how to transform attributes from the incoming assertion to attribute value pairs in OpenIG.

Each entry in the `assertionMapping` object has the form `localName: incomingName`, where *incomingName* is used to fetch the value from the incoming assertion, and *localName* is the name of the attribute set in the session. Avoid using dot characters (`.`) in the *localName*, as the `.` character also serves as a query separator in expressions.

The following shows an example of an assertionMapping object:

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

If the incoming assertion contains the statement:

```
mail = george@example.com
```

```
mailPassword = costanza
```

Then the following values are set in the session:

```
username = george@example.com
```

```
password = costanza
```

For this to work, you must 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.

When protecting multiple service providers, use unique *localName* settings. Otherwise different handlers can overwrite each others' data.

`"redirectURI"`: ***string, required***

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

This is how OpenIG and the Federation component work together to provide SSO. When OpenIG 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.

`"assertionConsumerEndpoint"`: ***string, optional***

Default: `fedletapplication` (same as the Fedlet)

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

`"authnContext"`: ***string, optional***

Name of the session field to hold the value of the authentication context. Avoid using dot characters (`.`) in the field name, as the `.` character also serves as a query separator in expressions.

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 OpenIG 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"`: ***string, optional***

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

Default: `|`

`"logoutURI"`: ***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"`: ***string, optional***

Name of the session field to hold the value of the session index. Avoid using dot characters (`.`) in the field name, as the `.` character also serves as a query separator in expressions.

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 OpenIG 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"`: ***string, optional***

Default: `fedletSLORedirect` (same as the Fedlet)

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

`"singleLogoutEndpointSoap"`: ***string, optional***

Default: `fedletSloSoap` (same as the Fedlet)

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

`"SPinitiatedSLOEndpoint"`: ***string, optional***

Default: `SPInitiatedSLO`

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

`"SPinitiatedSSOEndpoint"`: ***string, optional***

Default: `SPInitiatedSSO`

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

`"subjectMapping"`: ***string, optional***

Name of the session field to hold the value of the subject name. Avoid using dot characters (`.`) in the field name, as the `.` character also serves as a query separator in expressions.

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 OpenIG 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

The following sample configuration is corresponds to a scenario where OpenIG 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"
    }
}
```

Javadoc

org.forgerock.openig.handler.saml.SamlFederationHandler

ScriptableHandler — handle a request by using a script

Description

Handles a request by using a script.

The script must return either a Promise<Response or a Response.

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.

Classes

The following classes are imported automatically for Groovy scripts:

- `org.forgerock.http.Client`

- `org.forgerock.http.Filter`
- `org.forgerock.http.Handler`
- `org.forgerock.http.filter.throttling.ThrottlingRate`
- `org.forgerock.http.util.Uris`
- `org.forgerock.util.AsyncFunction`
- `org.forgerock.util.Function`
- `org.forgerock.util.promise.NeverThrowsException`
- `org.forgerock.util.promise.Promise`
- `org.forgerock.services.context.Context`
- `org.forgerock.http.protocol.*`

Objects

The script has access to the following global objects:

Any parameters passed as args

You can use the configuration to pass parameters to the script by specifying an args object.

Take care when naming keys in the args object. Attempts to reuse the name of another global object cause the script to fail and OpenIG to return a response with HTTP status code 500 Internal Server Error.

`attributes`

The attributes object provides access to a context map of arbitrary attributes, which is a mechanism for transferring transient state between components when processing a single request.

Use `session` for maintaining state between successive requests from the same logical client.

`context`

The processing context.

This context is the leaf of a chain of contexts. It provides access to other Context types, such as SessionContext, AttributesContext, and ClientContext, through the `context.asContext(ContextClass.class)` method.

`request`

The HTTP request.

`globals`

This object is a Map that holds variables that persist across successive invocations.

`http`

An embedded client for making outbound HTTP requests, which is an org.forgerock.http.Client.

If a `"clientHandler"` is set in the configuration, then that Handler is used. Otherwise, the default

ClientHandler configuration is used.

For details, see Handlers.

`ldap`

The ldap object provides an embedded LDAP client.

Use this client to perform outbound LDAP requests, such as LDAP authentication.

`logger`

The logger object provides access to the server log sink.

`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.

Usage

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

Properties

`"type"`: ***string, required***

The Internet media type (formerly MIME type) of the script, `"application/x-groovy"` for Groovy

`"file"`: ***expression***

Path to the file containing the script; mutually exclusive with `"source"`

Relative paths in the file field are relative to the base location for scripts. The base location depends on the configuration. For details, see Installing OpenIG in the *Gateway Guide*.

The base location for Groovy scripts is on the classpath when the scripts are executed. If therefore some Groovy scripts are not in the default package, but instead have their own package names, they belong in the directory corresponding to their package name. For example, a script in package `com.example.groovy` belongs under `openig-`

base/scripts/groovy/com/example/groovy/.

"source": *string*

The script as a string; mutually exclusive with "file"

"args": *map, optional*

Parameters passed from the configuration to the script.

The configuration object is a map whose values can be scalars, arrays, objects and so forth, as in the following example.

```
{
    "args": {
        "title": "Coffee time",
        "status": 418,
        "reason": [
            "Not Acceptable",
            "I'm a teapot",
            "Acceptable"
        ],
        "names": {
            "1": "koffie",
            "2": "kafe",
            "3": "cafe",
            "4": "kafo"
        }
    }
}
```

The script can then 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 details regarding this status code see RFC 7168, Section 2.3.3 418 I'm a Teapot.

Args parameters can reference objects defined in the heap using expressions. For example, the following excerpt shows the heap that defines SampleFilter:

```
{
    "heap": [
        {
            "name": "SampleFilter",
            "type": "SampleFilter",
            "config": {
                "name": "X-Greeting",
                "value": "Hello world"
            }
```

```
        }
    ]
}
```

To pass `SampleFilter` to the script, the following example uses an expression in the args parameters:

```
{
    "args": {
        "filter": "${heap['SampleFilter']}"
    }
}
```

The script can then reference `SampleFilter` as `filter`.

For details about the heap, see Heap Objects(5).

`"clientHandler"`, ***ClientHandler reference, optional***

A Handler for making outbound HTTP requests.

Default: Use the default ClientHandler.

For details, see Handlers.

Javadoc

org.forgerock.openig.handler.ScriptableHandler

SequenceHandler — process request through sequence of handlers

Description

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

Usage

```
{
    "name": string,
    "type": "SequenceHandler",
    "config": {
        "bindings": [
```

```
            {
                "handler": Handler reference,
                "postcondition": expression
            }
        ]
    }
}
```

Properties

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

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

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

Dispatch to this handler.

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

See also Handlers.

"postcondition": ***expression, optional***

Evaluated to determine if the sequence continues.

Default: unconditional.

See also Expressions(5).

Javadoc

org.forgerock.openig.handler.SequenceHandler

StaticResponseHandler — create static response to a request

Description

Creates a static response to a request.

Usage

```
{
     "name": string,
     "type": "StaticResponseHandler",
     "config": {
         "status": number,
```

```
        "reason": string,
        "version": string,
        "headers": {
            name: [ expression, ... ], ...
        },
        "entity": expression
    }
}
```

Properties

`"status"`: *number, required*

The response status code (for example, 200).

`"reason"`: *string, optional*

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

`"version"`: *string, optional*

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

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

Header fields to set in the response. The `name` specifies the header name, with an associated array of expressions to evaluate as values.

`"entity"`: *expression, optional*

The message entity expression to be evaluated and included in the response.

Conforms to the `Content-Type` header and sets `Content-Length`.

See also Expressions(5).

Example

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

Javadoc

org.forgerock.openig.handler.StaticResponseHandler

Filters

Filter objects intercept requests and responses during processing.

AssignmentFilter — conditionally assign values to expressions

Description

Conditionally assigns values to expressions before the request and after the response is handled.

Usage

```
{
    "name": string,
    "type": "AssignmentFilter",
    "config": {
        "onRequest": [
            {
                "condition": expression,
                "target": lvalue-expression,
                "value": expression
            }, ...
        ],
        "onResponse": [
            {
                "condition": expression,
                "target": lvalue-expression,
                "value": 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"`: ***expression, optional***

Expression to evaluate to determine if an assignment should occur. Omitting the condition makes the assignment unconditional.

See also Expressions(5).

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

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

See also Expressions(5).

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

Expression that yields the value to be set in the target.

See also Expressions(5).

Example

This is an example of how you would capture credentials and store them in the OpenIG session during a login request. Notice the credentials are captured on the request, but not marked as valid until the response returns a positive 302. The credentials would then be used to login a user to a different application:

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

Javadoc

org.forgerock.openig.filter.AssignmentFilter

ConditionEnforcementFilter — verify a condition to continue the chain of execution

Description

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

Usage

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

Properties

`"condition"`: *boolean expression, required*

Expression that evaluates to `true` or `false`, to determine whether a request should continue to be executed.

See also Expressions.

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

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

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

See also Handlers.

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

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"
    }
}
```

Javadoc

org.forgerock.openig.filter.ConditionEnforcementFilter

CookieFilter — manage, suppress, relay cookies

Description

Manages, suppresses and relays cookies. Managed cookies are intercepted by the cookie filter itself and stored in the gateway session; managed cookies are not transmitted to the user agent. Suppressed cookies are removed from both request and response. Relayed cookies are transmitted freely between user agent and 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": [ string, ... ],
         "suppressed": [ string, ... ],
         "relayed": [ string, ... ],
         "defaultAction": string
     }
}
```

Properties

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

A list of the names of cookies to be managed.

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

A list of the names of cookies to be suppressed.

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

A list of the names of cookies to be relayed.

`"defaultAction"`: ***string, optional***

Action to perform for cookies that do not match an action set. Must be one of: `"MANAGE"`, `"RELAY"`, `"SUPPRESS"`. Default: `"MANAGE"`.

Javadoc

org.forgerock.openig.filter.CookieFilter

CryptoHeaderFilter — encrypt, decrypt headers

Description

Encrypts or decrypts headers in a request or response.

Usage

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

Properties

`"messageType"`: ***string, required***

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

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

`"operation"`: ***string, required***

Indicates whether to encrypt or decrypt.

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

`"key"`: ***expression, required***

Base64 encoded key value.

See also Expressions(5).

`"algorithm"`: ***string, optional***

Algorithm used for encryption and decryption.

Default: `AES/ECB/PKCS5Padding`

`"keyType"`: ***string, optional***

Algorithm name for the secret key.

Default: `AES`

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

The names of header fields to encrypt or decrypt.

Default: Do not encrypt or decrypt any headers

Example

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

Javadoc

org.forgerock.openig.filter.CryptoHeaderFilter

EntityExtractFilter — extract pattern from message entity

Description

Extracts regular expression patterns from a message entity. The extraction results are stored in a "target" object. For a given matched pattern, as described in Patterns(5), the value stored in the object is either the result of applying its associated pattern template (if specified) or the match result itself otherwise.

Usage

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

Properties

"messageType": *string, required*

The message type to extract patterns from.

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

"charset": *string, optional*

Overrides the character set encoding specified in message.

Default: the message encoding is used.

"target": *lvalue-expression, required*

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

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

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

See also Expressions(5).

`"key"`: ***string, required***

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

`"pattern"`: ***pattern, required***

The regular expression pattern to find in the entity.

See also Patterns(5).

`"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(5).

Examples

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

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

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

The following example reads the response looking for the OpenAM 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"
            }
        ]
    }
}
```

Javadoc

org.forgerock.openig.filter.EntityExtractFilter

FileAttributesFilter — retrieve record from a file

Description

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

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

Usage

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

For an example see Log in With Credentials From a File in the *Gateway Guide*.

Properties

`"file"`: ***expression, required***

The file containing the record to be read.

See also Expressions(5).

`"charset"`: ***string, optional***

The character set in which the file is encoded.

Default: `"UTF-8"`.

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

The separator character, which is one of the following:

`COLON`

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

`COMMA`

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

`TAB`

Tab separated values, with support for quoted literal strings.

+ Default: `COMMA`

`"header"`: ***boolean, optional***

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

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

Default: `true`.

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

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

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

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

Expression that yields the target object to contain the record.

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

See also Expressions(5).

`"key"`: ***string, required***

The key used for the lookup operation.

`"value"`: ***expression, required***

Expression that yields the value to be looked-up within the file.

See also Expressions(5).

Javadoc

org.forgerock.openig.filter.FileAttributesFilter

HeaderFilter — remove and add headers

Description

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

Usage

```
{
     "name": string,
     "type": "HeaderFilter",
     "config": {
         "messageType": string,
         "remove": [ string, ... ],
         "add": {
            name: [ string, ... ], ...
         }
     }
}
```

Properties

`"messageType"`: ***string, required***

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

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

The names of header fields to remove from the message.

`"add"`: ***object, optional***

Header fields to add to the message. The `name` specifies the header name, with an associated array of string values.

Examples

Replace the host header on the incoming request with `myhost.com`:

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

Add a Set-Cookie header in the response:

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

Add headers `custom1` and `custom2` to the request:

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

Javadoc

org.forgerock.openig.filter.HeaderFilter

HttpBasicAuthFilter — perform HTTP Basic authentication

Description

Performs authentication through the HTTP Basic authentication scheme. For more information, see RFC 2617.

If challenged for authentication via a `401 Unauthorized` status code by the server, this filter retries the request with credentials attached. Once 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 of no credentials yielded from expressions), then processing is diverted to the specified authentication failure handler.

Usage

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

Properties

`"username"`: *expression, required*

Expression that yields the username to supply during authentication.

See also Expressions(5).

`"password"`: *expression, required*

Expression that yields the password to supply during authentication.

See also Expressions(5).

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

Dispatch to this Handler if authentication fails.

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

See also Handlers.

`"cacheHeader"`: ***boolean, optional***

Whether 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
    }
}
```

Javadoc

org.forgerock.openig.filter.HttpBasicAuthFilter

LocationHeaderFilter — rewrites Location headers

Description

Rewrites Location headers on responses that generate a redirect that would take the user directly to the application being proxied rather than taking the user through OpenIG.

For example, if OpenIG listens on `https://proxy.example.com:443/` and the application it protects listens on `http://www.example.com:8080/`, then you can configure this filter to rewrite redirects that would take the user to locations under `http://www.example.com:8080/` to go instead to locations under `https://proxy.example.com:443/`.

Usage

```
{
    "name": string,
    "type": "LocationHeaderFilter",
    "config": {
```

```
        "baseURI": expression
    }
}
```

An alternative value for type is RedirectFilter.

Properties

"baseURI": *expression, optional*

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

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

See also Expressions(5).

Example

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

Javadoc

org.forgerock.openig.filter.LocationHeaderFilter

OAuth2ClientFilter — Authenticate an end user with OAuth 2.0 delegated authorization

Description

An OAuth2ClientFilter is a filter that authenticates an end user using OAuth 2.0 delegated authorization. The filter can act as an OpenID Connect relying party as well as an OAuth 2.0 client.

The client filter does not include information about identity providers, or information about static registration with identity providers. For information about an identity provider, see Issuer(5). For information about registration with an identity provider, see ClientRegistration(5).

In the case where all users share the same identity provider, you can configure the filter as a client of a single provider by referencing a single client registration name for the filter. You can also configure the filter to work with multiple providers, taking the user to a login handler page—often

full of provider logos, and known as a *Nascar page*. The name comes from Nascar race cars, some of which are covered with sponsors' logos—to choose a provider.

What an OAuth2ClientFilter does depends on the incoming request URI. In the following list *clientEndpoint* represents the value of the clientEndpoint in the filter configuration:

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

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

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

`clientEndpoint/login?registration=registrationName&goto=url`

Redirect the end user for authorization with the specified *registration*, which is the name of a ClientRegistration configuration as described in ClientRegistration(5).

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.

Ultimately if the entire process is successful, the filter saves the authorization state in the context and redirects the user-agent to the specified URL.

`clientEndpoint/logout?goto=url`

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

`clientEndpoint/callback`

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

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

Other request URIs

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

Usage

```
{
  "name": string,
  "type": "OAuth2ClientFilter",
  "config": {
    "clientEndpoint": expression,
    "failureHandler": Handler reference,
    "discoveryHandler": Handler reference,
    "loginHandler": Handler reference,
    "registrations": [ ClientRegistration reference(s) ],
    "metadata": dynamic registration client metadata object,
```

```
    "cacheExpiration": duration string,
    "executor": executor,
    "target": expression,
    "defaultLoginGoto": expression,
    "defaultLogoutGoto": expression,
    "requireHttps": boolean,
    "requireLogin": boolean
  }
}
```

Properties

"clientEndpoint": *expression, required*

Base URI for the filter.

For example, if you set "clientEndpoint": "/openid", then the service URIs for this filter on your OpenIG server are /openid/login, /openid/logout, and /openid/callback.

See also Expressions(5).

"failureHandler": *Handler reference, required*

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

If this handler is invoked, then the target in the context can be populated with information such as the exception, client registration, and error.

The failure object in the target is a simple map, similar to the following example:

```
{
    "client_registration": "ClientRegistration name string",
    "error": {
        "realm": "optional string",
        "scope": [ "optional required scope string", ... ],
        "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.

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

Invoke this HTTP client handler to communicate with the OpenID Provider for OpenID Connect Discovery.

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: OpenIG uses the default ClientHandler.

See also Handlers, ClientHandler(5).

`"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 identity provider. 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.

For an example of a login handler where no client registrations are defined, see Preparing OpenIG for Discovery and Dynamic Registration in the *Gateway Guide*. The following example shows a login handler that allows the user to choose from two client registrations: `openam` and `google`:

```
{
    "name": "NascarPage",
    "type": "StaticResponseHandler",
    "config": {
        "status": 200,
        "entity": "<html><p><a

href='/openid/login?registration=openam&goto=${urlEncodeQueryParameterNameOrValue(c
ontexts.router.originalUri)}'
                  >OpenAM Login</a></p>
                  <p><a
```

```
href='/openid/login?registration=google&goto=${contexts.router.originalUri}'
                    >Google Login</a></p>
                    </html>"
    }
}
```

See also Handlers.

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

List of client registrations that authenticate OpenIG to the identity providers. 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 identity provider as described in ClientRegistration(5).

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

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

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

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

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

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

`"client_name"`: ***string, optional***

Name of the client to present to the end user.

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

Array of scope strings to request of the OpenID Provider.

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

Duration for which to cache user-info resources.

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

A duration is a lapse of time expressed in English, such as `23 hours 59 minutes and 59 seconds`.

Durations are not case sensitive.

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`: microseconds
- `nanoseconds`, `nanosecond`, `nanosec`, `nanos`, `nano`, `ns`: nanoseconds

Default: 20 seconds

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

`"executor"`: *executor, optional*

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

Default: `ScheduledExecutorService`

See also ScheduledExecutorService(5).

`"target"`: *expression, optional*

Expression that yields the target object whose value is to be set, such as `${attributes.openid}`.

Default: `${attributes.openid}`

See also Expressions(5).

`"defaultLoginGoto"`: *expression, optional*

The URI to redirect to after successful authentication and authorization.

Default: return an empty page.

See also Expressions(5).

`"defaultLogoutGoto"`: *expression, optional*

The URI to redirect to after successful logout.

Default: return an empty page.

See also Expressions(5).

`"requireHttps"`: ***boolean, optional***

Whether to require that requests use the HTTPS scheme.

Default: true.

`"requireLogin"`: ***boolean, optional***

Whether to require authentication for all incoming requests.
Default: true.

Example

The following example configures an OAuth 2.0 client filter. The base client endpoint is `/openid`. The filter uses well-known configuration endpoints to obtain configuration information for OpenAM and for Google as providers. The client credentials are not shown.

When a incoming request is made to `/openid/login`, this filter takes the user to a NascarPage to choose an identity provider. It then handles negotiation for authorization with the provider.

If the authorization process completes successfully, then the filter injects the authorization state data into `attributes.openid`.

At the end of the interaction, the aim of this configuration is simply to dump the data obtained back in the response:

```
{
    "name": "OpenIDConnectClient",
    "type": "OAuth2ClientFilter",
    "config": {
        "target"               : "${attributes.openid}",
        "clientEndpoint"       : "/openid",
        "loginHandler"         : "NascarPage",
        "registrations"        : [ "openam", "google" ],
        "failureHandler"       : "Dump",
        "defaultLoginGoto"     : "/dump",
        "defaultLogoutGoto"    : "/unprotected",
        "requireHttps"         : false,
        "requireLogin"         : true
    }
}
```

For details regarding configuration of providers, see Issuer(5) and ClientRegistration(5).

Notice that this configuration is for development and testing purposes only, and is not secure ("requireHttps": false). Make sure you do require HTTPS in production environments.

Javadoc

org.forgerock.openig.filter.oauth2.client.OAuth2ClientFilter

See Also

Issuer(5), ClientRegistration(5)

The OAuth 2.0 Authorization Framework

OAuth 2.0 Bearer Token Usage

OpenID Connect site, in particular the list of standard OpenID Connect 1.0 scope values

---

OAuth2ResourceServerFilter — validate a request containing an OAuth 2.0 access token

Description

An OAuth2ResourceServerFilter is a filter that 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 `1fc0e143-f248-4e50-9c13-1d710360cec9`:

```
Authorization: Bearer 1fc0e143-f248-4e50-9c13-1d710360cec9
```

The filter extracts the access token, and then validates it against the configured tokenInfoEndpoint URL.

On successful validation, the filter creates a new context for the authorization server response, at `${contexts.oauth2}`.

The context is named `oauth2` and can be reached at `contexts.oauth2` or `contexts['oauth2']`.

The context contains data such as the access token, which can be reached at `contexts.oauth2.accessToken` or `contexts['oauth2'].accessToken`.

Regarding errors, if the filter configuration and access token together result in an invalid request to the authorization server, the filter returns an HTTP 400 Bad Request response to the user-agent.

If the access token is missing from the request, the filter returns an HTTP 401 Unauthorized response to the user-agent:

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Bearer realm="OpenIG"
```

If the access token is not valid, for example, because it has expired, the filter also returns an HTTP 401 Unauthorized response to the user-agent.

If the scopes for the access token do not match the specified required scopes, the filter returns an

HTTP 403 Forbidden response to the user-agent.

Usage

```
{
  "name": string,
  "type": "OAuth2ResourceServerFilter",
  "config": {
    "providerHandler": Handler reference,
    "scopes": [ expression, ... ],
    "tokenInfoEndpoint": URL string,
    "cacheExpiration": duration string,
    "executor": executor,
    "requireHttps": boolean,
    "realm": string
  }
}
```

An alternative value for type is OAuth2RSFilter.

Properties

`"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: OpenIG uses the default ClientHandler.

See also Handlers, ClientHandler(5).

`"scopes"`: *array of expressions, required*

The list of required OAuth 2.0 scopes for this protected resource.

See also Expressions(5).

`"tokenInfoEndpoint"`: *URL string, required*

The URL to the token info endpoint of the OAuth 2.0 authorization server.

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

Duration for which to cache OAuth 2.0 access tokens.

Caching allows OpenIG to avoid repeated requests for token info when reusing the information over a short period.

A duration is a lapse of time expressed in English, such as `23 hours 59 minutes and 59 seconds`.

Durations are not case sensitive.

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`: microseconds
- `nanoseconds`, `nanosecond`, `nanosec`, `nanos`, `nano`, `ns`: nanoseconds

Default: 1 minute

Set this to disabled or zero to disable caching. When caching is disabled, each request triggers a new request to the authorization server to verify the access token.

`"executor"`: *executor, optional*

An executor service to schedule the execution of tasks, such as the eviction of entries in the access token cache.

Default: `ScheduledExecutorService`

See also ScheduledExecutorService(5).

`"requireHttps"`: *boolean, optional*

Whether to require that requests use the HTTPS scheme.

Default: true

`"realm"`: *string, optional*

HTTP authentication realm to include in the WWW-Authenticate response header field when returning an HTTP 401 Unauthorized status to a user-agent that need to authenticate.

Default: OpenIG

Example

The following example configures an OAuth 2.0 protected resource filter that expects scopes email and profile (and returns an HTTP 403 Forbidden status if the scopes are not present), and validates access tokens against the OpenAM token info endpoint. It caches access tokens for up to 2 minutes:

```
{
    "name": "ProtectedResourceFilter",
    "type": "OAuth2ResourceServerFilter",
```

```
    "config": {
        "providerHandler": "ClientHandler",
        "scopes": [
            "email",
            "profile"
        ],
        "tokenInfoEndpoint": "https://openam.example.com:8443/openam/oauth2/tokeninfo
",
        "cacheExpiration": "2 minutes"
    }
}
```

Javadoc

org.forgerock.openig.filter.oauth2.OAuth2ResourceServerFilterHeaplet

See Also

The OAuth 2.0 Authorization Framework

OAuth 2.0 Bearer Token Usage

PasswordReplayFilter — replay credentials with a single filter

Description

Replays credentials in a single composite filter for the following cases:

- When the request is for a login page
- When the response contains a login page

When the response contains a login page, a PasswordReplayFilter can extract values from the response entity and reuse the values when replaying credentials.

A PasswordReplayFilter does not retry failed authentication attempts.

Usage

```
{
    "name": string,
    "type": "PasswordReplayFilter",
    "config": {
        "request": request configuration object,
        "loginPage": expression,
        "loginPageContentMarker": pattern,
```

```
        "credentials": Filter reference,
        "headerDecryption": crypto configuration object,
        "loginPageExtractions": [ extract configuration object, ... ]
    }
}
```

Properties

"request": *request configuration object, required*

The request that replays the credentials.

The request configuration object has the following fields:

"method": *string, required*

The HTTP method to be performed on the resource such as GET or POST.

"uri": *string, required*

The fully qualified URI of the resource to access such as http://www.example.com/login.

"entity": *expression, optional*

The entity body to include in the request.

This setting is mutually exclusive with the form setting when the method is set to POST.

See also Expressions(5).

"form": *object, optional*

A form to include in the request.

The param specifies the form parameter name. Its value is an array of expressions to evaluate as form field values.

This setting is mutually exclusive with the entity setting when the method is set to POST.

"headers": *object, optional*

Header fields to set in the request.

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

"version": *string, optional*

The HTTP protocol version.

Default: "HTTP/1.1".

The implementation uses a StaticRequestFilter. The fields are the same as those described in StaticRequestFilter(5).

`"loginPage"`: ***expression, required unless loginPageContentMarker is defined***

An expression that is true when a login page is requested, false otherwise.

For example, the following expression specifies that an HTTP GET to the path `/login` is a request for a login page:

```
${matches(request.uri.path, '/login') and (request.method == 'GET')}
```

OpenIG only evaluates the expression for the request, not for the response.

See also Expressions(5).

`"loginPageContentMarker"`: ***pattern, required unless loginPage is defined***

A pattern that matches when a response entity is that of a login page.

See also Patterns(5).

`"credentials"`: ***Filter reference, optional***

Filter that injects credentials, making them available for replay. Consider using a `FileAttributesFilter` or a `SqlAttributesFilter`.

When this is not specified, credentials must be made available to the request by other means.

See also Filters.

`"headerDecryption"`: ***crypto configuration object, optional***

Object to decrypt request headers that contain credentials to replay.

The crypto configuration object has the following fields:

`"key"`: ***expression, required***

Base64 encoded key value.

See also Expressions(5).

`"algorithm"`: ***string, optional***

Algorithm used for decryption.

Default: `AES/ECB/PKCS5Padding`

`"keyType"`: ***string, optional***

Algorithm name for the secret key.

Default: `AES`

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

The names of header fields to decrypt.

Default: Do not decrypt any headers.

`"loginPageExtractions"`: ***extract configuration array, optional***

Object to extract values from the login page entity.

The extract configuration array is a series of configuration objects. To extract multiple values, use multiple extract configuration objects. Each object has the following fields:

`"name"`: ***string, required***

Name of the field where the extracted value is put.

The names are mapped into `attributes.extracted`.

For example, if the name is `nonce`, the value can be obtained with the expression `${attributes.extracted.nonce}`.

The name `isLoginPage` is reserved to hold a boolean that indicates whether the response entity is a login page.

`"pattern"`: ***pattern, required***

The regular expression pattern to find in the entity.

The pattern must contain one capturing group. (If it contains more than one, only the value matching the first group is placed into `attributes.extracted`.)

For example, suppose the login page entity contains a nonce required to authenticate, and the nonce in the page looks like `nonce='n-0S6_WzA2Mj'`. To extract `n-0S6_WzA2Mj`, set `"pattern": "nonce='(.*)'"`.

See also Patterns(5).

Examples

The following example route authenticates requests using static credentials whenever the request is for `/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 is for `/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": "ClientHandler"
    }
}
```

For additional examples, see Configuration Templates in the *Gateway Guide*, and the Javadoc for the PasswordReplayFilter class.

Javadoc

org.forgerock.openig.filter.PasswordReplayFilterHeaplet

PolicyEnforcementFilter — enforce policy decisions from OpenAM

Description

This filter requests policy decisions from OpenAM, which allows or denies the request based on the request context, the request URI, and the OpenAM policies.

- If the request is allowed, processing continues.
- If the request is denied, OpenIG returns 403 Forbidden.
- If an error occurs during the process, OpenIG returns 500 Internal Server Error.

This filter allows you to specify the subject by SSO token, JWT, or JWT claims.

This filter can add contextual attributes (accessible through `${attributes}`), and some elements returned by the policy decision, such as attributes and advices.

NOTE

In the OpenAM policy, remember to configure the `Resources` parameter with the URI of the protected application.

The request URI from OpenIG must match the `Resources` parameter defined in the OpenAM 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 OpenAM policy accordingly.

Usage

```
{
    "name": string,
    "type": "PolicyEnforcementFilter",
    "config": {
        "openamUrl": URI expression,
        "pepUsername": expression,
        "pepPassword": expression,
        "pepRealm": string,
        "ssoTokenSubject": expression,
        "jwtSubject": expression,
        "claimsSubject": map or expression,
        "amHandler": Handler reference,
        "realm": string,
        "ssoTokenHeader": string,
        "application": string,
        "cacheMaxExpiration": duration string,
        "target": lvalue-expression,
        "environment": map or expression,
        "executor": executor
    }
}
```

Properties

"openamUrl": ***URI expression, required***

The URL to an OpenAM service, such as `https://openam.example.com:8443/openam/`.

See also Expressions(5).

"pepUsername": ***expression, required***

The OpenAM username of the user with permission to request policy decisions.

See also Expressions(5).

"pepPassword": ***expression, required***

The OpenAM password of the user with permission to request policy decisions.

See also Expressions(5).

"pepRealm": ***string, optional***

The realm of the user with permission to request policy decisions.

Default: The value used by `realm`.

"ssoTokenSubject": ***expression, required if neither of the following properties are present: "jwtSubject", "claimsSubject"***

An expression evaluating to the OpenAM SSO token ID string for the subject making the request

to the protected resource.

See also Expressions(5).

`"jwtSubject"`: ***expression, required if neither of the following properties are present: "ssoTokenSubject", "claimsSubject"***

An expression evaluating to the JWT string for the subject making the request to the protected resource.

To use the raw id_token (base64, not decoded) returned by the OpenID Connect Provider during authentication, place an `OAuth2ClientFilter` filter before the PEP filter, and then use `${attributes.openid.id_token}` as the expression value.

See also OAuth2ClientFilter(5) and Expressions(5).

`"claimsSubject"`: ***map or expression, required if neither of the following properties are present: "jwtSubject", "ssoTokenSubject"***

A representation of JWT claims for the subject. The subject must be specified, but the JWT claims can contain other information such as the token issuer, expiration, and so on.

If this property is a map, the structure must have the format `Map<String, Object>`. The value is evaluated as an expression.

```
"claimsSubject": {
          "sub": "${attributes.subject_identifier}",
          "iss": "openam.example.com"
      }
```

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

```
"claimsSubject": "${attributes.openid.id_token_claims}"
```

See also Expressions(5).

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

The handler to use when requesting policy decisions from OpenAM.

In production, use a ClientHandler that is capable of making an HTTPS connection to OpenAM.

Default: OpenIG uses the `ForgeRockClientHandler`.

See also Handlers.

`"realm"`: ***string, optional***

The OpenAM realm to use when requesting policy decisions.

Default: `/` (Top Level Realm)

`"ssoTokenHeader"`: ***string, optional***

The name of the HTTP header to use when supplying the SSO token ID for the user making a policy decision request.

Default: `iPlanetDirectoryPro`

`"application"`: ***string, optional***

The OpenAM application to use when requesting policy decisions.

Default: OpenIG does not specify an application when making a policy decision request. As a result, the application is `iPlanetAMWebAgentService`, which is the default for OpenAM.

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

Maximum duration for which to cache policy decision responses. If the time-to-live value in the policy decision response is shorter, then OpenIG expires the decision according to the shorter lifetime.

This setting prevents OpenIG from having to issue a new request for every policy decision, including even repeated requests by the same subject for the same resource.

NOTE	Cached policy decisions remain in the OpenIG cache even after a user logs out of OpenAM and the OpenAM session becomes invalid.

A duration is a lapse of time expressed in English, such as `23 hours 59 minutes and 59 seconds`.

Durations are not case sensitive.

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`: microseconds
- `nanoseconds`, `nanosecond`, `nanosec`, `nanos`, `nano`, `ns`: nanoseconds

Default: 1 minute

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

A map in the attributes context where the "attributes" and "advices" map fields from the policy decision are saved.

Example: `${attributes.policy.attributes}` and `${attributes.policy.advices}`

Default: `${attributes.policy}`

`"environment"`: ***map or expression, optional***

Environment conditions can be defined in an OpenAM policy to set the circumstances under which the policy applies. For example, environment conditions can specify that the policy applies only during working hours or only when accessing from a specific IP address.

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

```
"environment": {
          "IP": [ "${contexts.client.remoteAddress}" ]
      }
```

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

```
"environment": "${attributes.my_environment}"
```

`"executor"`: ***executor, optional***

An executor service to schedule the execution of tasks, such as the eviction of entries in the policy decision cache.

Default: `ScheduledExecutorService`

See also ScheduledExecutorService(5).

Example

The following example requests a policy decision from OpenAM before allowing a request to continue. The `policyAdmin` user is an OpenAM subject with permission to request policy decisions. The user making the request to the protected resource is identified by an SSO token ID string. The realm defaults to OpenAM's top-level realm:

```
{
    "handler": {
        "type": "Chain",
        "config": {
            "filters": [
                {
                    "type": "PolicyEnforcementFilter",
                    "config": {
                        "openamUrl": "https://openam.example.com:8443/openam/",
                        "pepUsername": "policyAdmin",
                        "pepPassword": "${env['POLICY_ADMIN_PWD']}",
                        "ssoTokenSubject": "${attributes.SSOCurrentUser}",
                        "claimsSubject": "${attributes.openid.id_token_claims}",
```

```
                        "target": "${attributes.currentPolicy}",
                        "environment": {
                            "IP": [ "${contexts.client.remoteAddress}" ]
                         }
                    }
                }
            ],
            "handler": "ClientHandler"
        }
    }
}
```

Javadoc

org.forgerock.openig.openam.PolicyEnforcementFilter

See Also

Requesting Policy Decisions

ScriptableFilter — process requests and responses by using a script

Description

Processes requests and responses by using a script.

The script must return either a Promise<Response or a Response.

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.

Classes

The following classes are imported automatically for Groovy scripts:

- `org.forgerock.http.Client`
- `org.forgerock.http.Filter`
- `org.forgerock.http.Handler`
- `org.forgerock.http.filter.throttling.ThrottlingRate`

- `org.forgerock.http.util.Uris`
- `org.forgerock.util.AsyncFunction`
- `org.forgerock.util.Function`
- `org.forgerock.util.promise.NeverThrowsException`
- `org.forgerock.util.promise.Promise`
- `org.forgerock.services.context.Context`
- `org.forgerock.http.protocol.*`

Objects

The script has access to the following global objects:

Any parameters passed as args

You can use the configuration to pass parameters to the script by specifying an args object.

Take care when naming keys in the args object. If you reuse the name of another global object, cause the script to fail and OpenIG to return a response with HTTP status code 500 Internal Server Error.

`attributes`

The attributes object provides access to a context map of arbitrary attributes, which is a mechanism for transferring transient state between components when processing a single request.

Use `session` for maintaining state between successive requests from the same logical client.

`context`

The processing context.

This context is the leaf of a chain of contexts. It provides access to other Context types, such as SessionContext, AttributesContext, and ClientContext, through the `context.asContext(ContextClass.class)` method.

`request`

The HTTP request.

`globals`

This object is a Map that holds variables that persist across successive invocations.

`http`

An embedded client for making outbound HTTP requests, which is an org.forgerock.http.Client.

If a `"clientHandler"` is set in the configuration, then that Handler is used. Otherwise, the default ClientHandler configuration is used.

For details, see Handlers.

`ldap`

The ldap object provides an embedded LDAP client.

Use this client to perform outbound LDAP requests, such as LDAP authentication.

`logger`

The logger object provides access to the server log sink.

`next`

The next object refers to the next handler in the filter chain.

`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.

When you have finished processing the request, 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.

Usage

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

Properties

`"type"`: *string, required*

The Internet media type (formerly MIME type) of the script, `"application/x-groovy"` for Groovy

`"file"`: *expression*

Path to the file containing the script; mutually exclusive with `"source"`

Relative paths in the file field are relative to the base location for scripts. The base location depends on the configuration. For details, see Installing OpenIG in the *Gateway Guide*.

The base location for Groovy scripts is on the classpath when the scripts are executed. If

therefore some Groovy scripts are not in the default package, but instead have their own package names, they belong in the directory corresponding to their package name. For example, a script in package `com.example.groovy` belongs under `openig-base/scripts/groovy/com/example/groovy/`.

`"source"`: *string*

The script as a string; mutually exclusive with `"file"`

`"args"`: *object, optional*

Parameters passed from the configuration to the script.

The configuration object is a map whose values can be scalars, arrays, objects and so forth, as in the following example:

```
{
    "args": {
        "title": "Coffee time",
        "status": 418,
        "reason": [
            "Not Acceptable",
            "I'm a teapot",
            "Acceptable"
        ],
        "names": {
            "1": "koffie",
            "2": "kafe",
            "3": "cafe",
            "4": "kafo"
        }
    }
}
```

The script can then 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 details regarding this status code see RFC 7168, Section 2.3.3 418 I'm a Teapot.

Args parameters can reference objects defined in the heap using expressions. For example, the following excerpt shows the heap that defines `SampleFilter`:

```
{
    "heap": [
        {
            "name": "SampleFilter",
            "type": "SampleFilter",
```

```
            "config": {
                "name": "X-Greeting",
                "value": "Hello world"
            }
        }
    ]
}
```

NOTE `SampleFilter` is a customized filter implemented as an extension of OpenIG. For information about sample filter, see Implementing a Customized Sample Filter in the *Gateway Guide*.

To pass `SampleFilter` to the script, the following example uses an expression in the args parameters:

```
{
    "args": {
        "filter": "${heap['SampleFilter']}"
    }
}
```

The script can then reference `SampleFilter` as `filter`.

For details about the heap, see Heap Objects(5).

`"clientHandler"`, ***ClientHandler reference, optional***

A Handler for making outbound HTTP requests.

Default: Use the default ClientHandler.

For details, see Handlers.

Javadoc

org.forgerock.openig.filter.ScriptableFilter

SqlAttributesFilter — execute SQL query

Description

Executes a SQL query through a prepared statement and exposes its first result. Parameters in the prepared statement are derived from expressions. The query result is exposed in an object whose location is specified by the `target` expression. If the query yields no result, then the resulting object is empty.

The execution of the query is performed lazily; it does not occur until the first attempt to access a

value in the target. This defers the overhead of connection pool, network and database query processing until a value is first required. This also means that the parameters expressions is not evaluated until the object is first accessed.

Usage

```
{
     "name": string,
     "type": "SqlAttributesFilter",
     "config": {
         "dataSource": string,
         "preparedStatement": string,
         "parameters": [ expression, ... ],
         "target": lvalue-expression
     }
}
```

Properties

`"dataSource"`: ***string, required***

The JNDI name of the factory for connections to the physical data source.

`"preparedStatement"`: ***string, required***

The parameterized SQL query to execute, with `?` parameter placeholders.

`"parameters"`: ***array of expressions, optional***

The parameters to evaluate and include in the execution of the prepared statement.

See also Expressions(5).

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

Expression that yields the target object that will contain the query results.

See also Expressions(5).

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.

Javadoc

org.forgerock.openig.filter.SqlAttributesFilter

StaticRequestFilter — create new request

Description

Creates a new request, replacing any existing request. The request can include an entity specified in the `entity` parameter. Alternatively, the request can include a form, specified in the `form` parameter, which is included in an entity encoded in `application/x-www-form-urlencoded` format if request method is `POST`, or otherwise as (additional) query parameters in the URI. The `form` and `entity` parameters cannot be used together when the `method` is set to `POST`.

Usage

```
{
     "name": string,
     "type": "StaticRequestFilter",
     "config": {
         "method": string,
         "uri": string,
         "version": string,
         "headers": {
             name: [ expression, ... ], ...
         },
         "form": {
```

```
            param: [ expression, ... ], ...
        },
        "entity": expression
    }
}
```

Properties

`"method"`: *string, required*

The HTTP method to be performed on the resource (for example, `"GET"`).

`"uri"`: *string, required*

The fully-qualified URI of the resource to access (for example, `"http://www.example.com/resource.txt"`).

`"version"`: *string, optional*

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

`"headers"`: *object, optional*

Header fields to set in the request.

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

`"form"`: *object, optional*

A form to include in the request.

The `param` specifies the form parameter name. Its value is an array of expressions to evaluate as form field values.

This setting is mutually exclusive with the `entity` setting when the `method` is set to `POST`.

`"entity"`: *expression, optional*

The entity body to include in the request.

This setting is mutually exclusive with the `form` setting when the `method` is set to `POST`.

See also Expressions(5).

Example

```
{
    "name": "LoginRequestFilter",
    "type": "StaticRequestFilter",
    "config": {
        "method": "POST",
        "uri": "http://10.10.0.2:8080/wp-login.php",
        "form": {
            "log": [ "george" ],
```

```
            "pwd": [ "bosco" ],
            "rememberme": [ "forever" ],
            "redirect_to": [ "http://portal.example.com:8080/wp-admin/" ],
            "testcookie": [ "1" ]
        }
    }
}
```

Javadoc

org.forgerock.openig.filter.StaticRequestFilter

SwitchFilter — divert requests to another handler

Description

Conditionally diverts requests to another handler. If a `condition` evaluates to `true`, then the request is dispatched to the associated `handler` with no further processing by the switch filter.

Usage

```
{
    "name": string,
    "type": "SwitchFilter",
    "config": {
        "onRequest": [
            {
                "condition": expression,
                "handler": Handler reference,
            },
            ...
        ],
        "onResponse": [
            {
                "condition": expression,
                "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"`: ***expression, optional***

Condition to evaluate to determine if the request or response should be dispatched to the handler.

Default: unconditional dispatch to the handler.

See also Expressions(5).

`"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"
            }
        ]
    }
}
```

Javadoc

org.forgerock.openig.filter.SwitchFilter

TokenTransformationFilter — transform a token issued by OpenAM to another type

Description

This filter transforms a token issued by OpenAM to another token type.

The current implementation uses the REST Security Token Service (STS) APIs. It supports transforming an OpenID Connect ID Token (`id_token`) into a SAML 2.0 assertion where the subject confirmation method is Bearer, as described in Profiles for the OASIS Security Assertion Markup Language (SAML) V2.0.

The configuration for this filter references a REST STS instance that must be set up in OpenAM before this filter can be used. The REST STS instance exposes a pre-configured transformation under a specific REST endpoint. See the OpenAM documentation for details about setting up a REST STS instance.

Any 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 OpenIG administrator.

Usage

```
{
    "name": "string",
    "type": "TokenTransformationFilter",
    "config": {
        "openamUri": URL string,
        "realm": OpenAM realm name string,
        "username": "${attributes.username}",
        "password": "${attributes.password}",
        "idToken": "${attributes.id_token}",
        "target": "${attributes.saml_assertions}",
        "instance": "oidc-to-saml",
        "amHandler": Handler reference,
        "ssoTokenHeader": string
    }
}
```

Properties

`"openamUri"`: *URL string, required*

The base URL to an OpenAM service, such as `https://openam.example.com:8443/openam/`.

Authentication and REST STS requests are made to this service.

`"realm"`: *string, optional*

The OpenAM realm containing both the OpenAM user who can make the REST STS request and

whose credentials are the username and password, and the STS instance described by the instance field.

Default: / (Top Level Realm)

`"username"`: ***expression, required***

The username for authenticating OpenIG as an OpenAM REST STS client.

See also Expressions(5).

`"password"`: ***expression, required***

The password for authenticating OpenIG as an OpenAM REST STS client.

See also Expressions(5).

`"idToken"`: ***expression, required***

An expression evaluating to OpenID Connect ID token.

The expected value is a string that is the JWT encoded `id_token`.

See also Expressions(5).

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

An expression evaluating to the location where the SAML 2.0 assertion is injected following successful transformation.

The value of the SAML 2.0 assertion is a string.

See also Expressions(5).

`"instance"`: ***expression, required***

An expression evaluating to name of the REST STS instance.

This expression is evaluated when the route is initialized, so the expression cannot refer to `request` or `contexts`.

See also Expressions(5).

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

The handler to use for authentication and STS requests to OpenAM.

In production, use a ClientHandler that is capable of making an HTTPS connection to OpenAM.

Default: OpenIG uses the `ForgeRockClientHandler`.

See also Handlers.

`"ssoTokenHeader"`: ***string, optional***

The name of the HTTP header to use when supplying the SSO token ID for the REST STS client subject.

Default: `iPlanetDirectoryPro`

Example

For an example of how to set up and test the token transformation filter, see Transforming OpenID Connect ID Tokens Into SAML Assertions in the *Gateway Guide*.

The following example uses the REST STS instance `oidc-to-saml` to request transformation of an OpenID Connect ID token into a SAML 2.0 assertion. Both the subject authenticating to access the REST endpoint, and the REST STS instance are in the realm `/sts`. The subject credentials for authentication to OpenAM are provided in the attributes context at `sts.username` and `sts.password`. The ID token to transform is provided in the attributes context at `sts.id_token`. The resulting SAML 2.0 assertion is injected as a string in the attribute context at `sts.saml_assertions`:

```
{
    "type": "TokenTransformationFilter",
    "config": {
        "openamUri": "https://openam.example.com/openam/",
        "realm": "/sts",
        "username": "${attributes.sts.username}",
        "password": "${attributes.sts.password}",
        "idToken": "${attributes.sts.id_token}",
        "target": "${attributes.sts.saml_assertions}",
        "instance": "oidc-to-saml",
        "amHandler": "ClientHandler"
    }
}
```

Javadoc

org.forgerock.openig.openam.TokenTransformationFilter

UmaFilter — protect access as an UMA resource server

Description

This filter acts as a policy enforcement point, protecting access as a User-Managed Access (UMA) resource server. Specifically, this filter ensures that a request for protected resources includes a valid requesting party token with appropriate scopes before allowing the response to flow back to the requesting party.

Usage

```
{
    "type": "UmaFilter",
    "config": {
```

```
        "protectionApiHandler": Handler reference,
        "umaService": UmaService reference,
        "realm": 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(5).

"realm": *string, optional*

The UMA realm set in the response to a request for a protected resource that does not include a requesting party token enabling access to the resource.

Default: `uma`

See Also

User-Managed Access (UMA) Profile of OAuth 2.0

org.forgerock.openig.uma.UmaResourceServerFilter

Decorators

Decorators are objects that decorate other heap objects, adding the new behavior that the decorator provides. For example, you can configure a decorator object for capturing requests and responses to a file and then decorate other objects in the heap to trigger the capture.

To decorate other objects individually, use a local decoration by adding the decorator's name value as a top-level field of the object. For example, suppose a capture decorator named capture is defined in the global configuration, `config.json`. The decorator is configured to capture the entity but not the context:

```
{
    "name": "capture",
    "type": "CaptureDecorator",
    "config": {
        "captureEntity": true,
        "_captureContext": true
    }
}
```

The following ClientHandler configuration would then capture requests including the entity before they are forwarded to the server:

```
{
    "name": "ClientHandler",
    "type": "ClientHandler",
    "capture": "request"
}
```

To decorate the handler for a route, add the decorator as a top-level field of the route. The following route includes an audit decoration on the handler. This configuration decorates the ClientHandler only for the current route. It does not decorate other uses of ClientHandler in other routes:

```
{
    "handler": "ClientHandler",
    "audit": "Default route"
}
```

The decoration as a top-level field also does not decorate heap objects. To decorate all applicable objects defined within a Route's heap, configure globalDecorators as a top-level field of the Route. The globalDecorators field takes a map of the decorations to apply. For example, the following route has audit and capture decorations that apply to the Chain, HeaderFilter, and StaticResponseHandler. In other words, the decorations apply to all objects in this route's heap:

```
{
```

```
    "globalDecorators": {
        "audit": "My static route",
        "capture": "all"
    },
    "handler": {
        "type": "Chain",
        "config": {
            "filters": [
                {
                    "type": "HeaderFilter",
                    "config": {
                        "messageType": "RESPONSE",
                        "add": [
                            {
                                "X-Powered-By": [
                                    "OpenIG"
                                ]
                            }
                        ]
                    }
                }
            ],
            "handler": {
                "type": "StaticResponseHandler",
                "config": {
                    "status": 200,
                    "entity": "Hello World"
                }
            }
        }
    },
    "condition": "${matches(request.uri.path, '^/static')}"
}
```

Decorations are inherited as follows:

- Local decorations that are part of an object's declaration are inherited wherever the object is used.
- The globalDecorations on a route are inherited on child routes.

To prevent loops, decorators themselves cannot be decorated. Instead, decorators apply only to specific types of objects such as Filters and Handlers.

OpenIG defines some decorators, such as audit, baseURI, capture, and timer. You can use these without configuring them explicitly. For details, see GatewayHttpApplication(5).

Take care when defining decorator names not to use names that unintentionally clash with field names for the decorated objects. For all heap objects, avoid decorators named `config`, `name`, and `type`. For Routes, avoid decorators named `auditService`, `baseURI`, `condition`, `globalDecorators`, `heap`, `handler`, `name`, and `session`. In `config.json`, also avoid `logSink` and `temporaryStorage`. In addition,

avoid decorators named `comment` or `comments`. The best way to avoid a clash with other field names is to avoid OpenIG reserved field names, which include all purely alphanumeric field names. Instead use dots in your decorator names, such as `my.decorator`.

Decorations can apply more than once. For example, if you set a decoration both on a Route and also on an object defined within the route, then OpenIG can apply the decoration twice. The following Route results in the request being captured twice:

```
{
  "handler": {
    "type": "ClientHandler",
    "capture": "request"
  },
  "capture": "all"
}
```

OpenIG applies decorations in this order.

1. Local decorations
2. globalDecorations (first those of the parent, then those declared in the current route)
3. Route decorations (those decorating a route's handler)

Interface Stability: Evolving (For details, see ForgeRock Product Interface Stability.)

AuditDecorator — trigger notification of audit events for Filters and Handlers

Description

Triggers notification of audit events for applicable Filters and Handlers.

Interface Stability: Deprecated (For details, see ForgeRock Product Interface Stability.)

OpenIG first notifies an audit system sink. The audit system sink takes responsibility for forwarding notifications to registered audit event listeners. The listeners take responsibility for dealing with the audit events. What a listener does is implementation specific, but it could for example publish the event to an endpoint or to a central system, log the event in a file, or raise an alert.

To help listeners determine what to do with audit events, each audit event holds the following information about what it represents:

`event.data`

A reference to the data involved in the event, providing access to the `request`, `response`, and `contexts` objects.

`event.source`

The source of the audit event, meaning the name of the object under audit.

For details, see org.forgerock.openig.audit.AuditSource.

`event.tags`

Strings that qualify the event. Entities receiving notifications can use the tags to select audit events of interest.

Define your own audit tags in order to identify particular events or routes.

OpenIG provides the following built-in tags in org.forgerock.openig.audit.Tag:

- `request`: This event happens before OpenIG calls the decorated object.
- `response`: This event happens after the call to the decorated object returns or throws an exception.

  When decorating a Filter, realize that the filter returns after handling the response, even if it only filters the request and so does nothing to the response but pass it along.

- `completed`: This event happens when the processing unit under audit has successfully handled the response. This tag always complements a `response` tag.

  Note that `completed` says nothing about the client application's perception of whether the result of the response was successful. For example, a Handler could successfully pass back an HTTP 404 Not Found response.

- `exception`: This event happens when the processing unit under audit handled the request and response processing with errors. This tag always complements a `response` tag.

  Note that the source object might not have thrown an exception itself, so it is not necessarily the source of the error.

  Also note that `exception` says nothing about the client application's perception of whether the result of the response was a failure. For example, another processing unit could still pass back a success response to the client application or proxy that engaged the request.

`event.timestamp`

Timestamp indicating when the event happened, with millisecond precision.

Decorated Object Usage

```
{
    "name": string,
    "type": string,
    "config": object,
    "audit": string or array of strings
}
```

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

The unique name of the object, just like an object that is not decorated.

`"type"`: string, required

The class name of the decorated object, which must be either a Filter or a Handler.

See also Filters and Handlers.

`"config"`: object, required unless empty

The configuration of the object, just like an object that is not decorated.

`"audit"`: string or array of strings, required

Set the value to the tag(s) used to select audit events of interest.

To activate the audit decoration without setting any user-defined tags, set audit to any other value, such as `"audit": true`.

Examples

The following example triggers an audit event on a default route:

```
{
    "handler": "ClientHandler",
    "audit": "Default route"
}
```

The following example triggers an audit event only on a particular object:

```
{
    "name": "My Serious Error Handler",
    "type": "StaticResponseHandler",
    "config": {
        "status": 500,
        "reason": "Error",
        "entity": "<html><p>Epic #FAIL</h2></html>"
    },
    "audit": "Epic failure"
}
```

To observe audit events, use a registered audit agent such as a MonitorEndpointHandler, which is described in MonitorEndpointHandler(5).

Javadoc

org.forgerock.openig.audit.decoration.AuditDecorator

BaseUriDecorator — override scheme, host, and port of request URI

Description

Overrides the scheme, host, and port of the existing request URI, rebasing the URI and so making requests relative to a new base URI. Rebasing changes only the scheme, host, and port of the request URI. Rebasing does not affect the path, query string, or fragment.

Decorator Usage

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

A BaseUriDecorator does not have configurable properties.

OpenIG creates a default BaseUriDecorator named baseURI at startup time in the top-level heap, so you can use baseURI as the decorator name without adding the decorator declaration explicitly.

Decorated Object Usage

```
{
    "name": string,
    "type": string,
    "config": object,
    decorator name: string
}
```

"name": string, required except for inline objects

The unique name of the object, just like an object that is not decorated

"type": string, required

The class name of the decorated object, which must be either a Filter or a Handler.

See also Filters and Handlers.

"config": object, required unless empty

The configuration of the object, just like an object that is not decorated

***decorator name*: string, required**

A string representing the scheme, host, and port of the new base URI. The port is optional when using the defaults (80 for HTTP, 443 for HTTPS).

OpenIG ignores this setting if the value is not a string.

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/"
}
```

Javadoc

org.forgerock.openig.decoration.baseuri.BaseUriDecorator

CaptureDecorator — capture request and response messages

Description

Captures request and response messages for further analysis.

Decorator Usage

```
{
    "name": string,
    "type": "CaptureDecorator",
    "config": {
        "logSink": LogSink reference,
        "captureEntity": boolean,
        "captureContext": boolean
    }
}
```

The decorator configuration has these properties:

`"logSink"`: ***LogSink reference, optional***

Capture requests and responses to this LogSink.

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

Default: use the LogSink configured for the decorated object. This makes it possible to keep all logs in a central location.

`"captureEntity"`: ***boolean, optional***

Whether the message entity should be captured.

The filter omits binary entities, instead writing a `[binary entity]` marker to the file.

Default: false

`"captureContext"`: ***boolean, optional***

Whether the context should be captured as JSON.

Default: false

Decorated Object Usage

```
{
    "name": string,
    "type": string,
    "config": object,
    decorator name: capture point(s)
}
```

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

The unique name of the object, just like an object that is not decorated

`"type"`: **string, required**

The class name of the decorated object, which must be either a Filter or a Handler.

See also Filters and Handlers.

`"config"`: **object, required unless empty**

The configuration of the object, just like an object that is not decorated

decorator name: **capture point(s), optional**

The *decorator name* must match the name of the CaptureDecorator. For example, if the CaptureDecorator has `"name": "capture"`, then *decorator name* is capture.

The capture point(s) are either a single string, or an array of strings. The strings are documented here in lowercase, but are not case-sensitive:

`"all"`
: Capture at all available capture points

`"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

Decorator configured to log the entity:

```
{
    "name": "capture",
    "type": "CaptureDecorator",
    "config": {
        "captureEntity": true
    }
}
```

Decorator configured not to log the entity:

```
{
    "name": "capture",
    "type": "CaptureDecorator"
}
```

Decorator configured to log the context in JSON format, excluding the request and the response:

```
{
    "name": "capture",
    "type": "CaptureDecorator",
    "config": {
        "captureContext": true
    }
```

```
}
```

To capture requests and responses with the entity before sending the request and before returning the response, do so as in the following example:

```
{
    "heap": [
        {
            "name": "capture",
            "type": "CaptureDecorator",
            "config": {
                "captureEntity": true
            }
        },
        {
            "name": "ClientHandler",
            "type": "ClientHandler",
            "capture": [
                "request",
                "response"
            ]
        }
    ],
    "handler": "ClientHandler"
}
```

To capture all transformed requests and responses as they leave filters, decorate the Route as in the following example. This Route uses the default CaptureDecorator:

```
{
    "handler": {
        "type": "Chain",
        "config": {
            "filters": [
                {
                    "type": "HeaderFilter",
                    "config": {
                        "messageType": "REQUEST",
                        "add": {
                            "X-RequestHeader": [
                                "Capture at filtered_request point",
                                "And at filtered_response point"
                            ]
                        }
                    }
                },
                {
                    "type": "HeaderFilter",
```

```
                    "config": {
                        "messageType": "RESPONSE",
                        "add": {
                            "X-ResponseHeader": [
                                "Capture at filtered_response point"
                            ]
                        }
                    }
                }
            ],
            "handler": {
                "type": "StaticResponseHandler",
                "config": {
                    "status": 200,
                    "reason": "OK",
                    "entity": "<html><p>Hello, World!</p></html>"
                }
            }
        }
    },
    "capture": [
        "filtered_request",
        "filtered_response"
    ]
}
```

To capture the context as JSON, excluding the request and response, before sending the request and before returning the response, do so as in the following example:

```
{
    "heap": [
        {
            "name": "capture",
            "type": "CaptureDecorator",
            "config": {
                "captureContext": true
            }
        },
        {
            "name": "ClientHandler",
            "type": "ClientHandler",
            "capture": [
                "request",
                "response"
            ]
        }
    ],
    "handler": "ClientHandler"
}
```

Javadoc

org.forgerock.openig.decoration.capture.CaptureDecorator

TimerDecorator — record times to process Filters and Handlers

Description

Records time in milliseconds to process applicable Filters and Handlers. OpenIG writes the records to the LogSink configured for the decorated heap object. If no LogSink is defined for the decorated heap object, then OpenIG writes to the LogSink configured for the heap. Records include the time elapsed while processing the request and response, and for Filters the elapsed time spent processing the request and response within the Filter itself.

OpenIG records times at log level `STAT`.

The TimerDecorator is not applicable to the GatewayHttpApplication, as the GatewayHttpApplication is not declared in the heap. For details, see GatewayHttpApplication(5).

Decorator Usage

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

A TimerDecorator does not have configurable properties.

OpenIG configures a default TimerDecorator named timer. You can use timer as the decorator name without explicitly declaring a decorator named timer.

Decorated Object Usage

```
{
    "name": string,
    "type": string,
    "config": object,
    decorator name: boolean
}
```

"name": string, required except for inline objects

The unique name of the object, just like an object that is not decorated

`"type"`: **string, required**

The class name of the decorated object, which must be either a Filter or a Handler.

See also Filters and Handlers.

`"config"`: **object, required unless empty**

The configuration of the object, just like an object that is not decorated

`decorator name`: **boolean, required**

OpenIG looks for the presence of the *decorator name* field for the TimerDecorator.

To activate the timer, set the value of the *decorator name* field to `true`.

To deactivate the TimerDecorator temporarily, set the value to `false`.

Examples

To record times spent within the client handler, and elapsed time for operations traversing the client handler, use a configuration such as the following:

```
{
    "handler": {
        "type": "ClientHandler"
    },
    "timer": true
}
```

This configuration could result in the following log messages:

```
TUE DEC 02 17:20:08 CET 2014 (STAT) @Timer[top-level-handler]
Started
------------------------------
TUE DEC 02 17:20:08 CET 2014 (STAT) @Timer[top-level-handler]
Elapsed time: 40 ms
```

When you decorate a Filter with a TimerDecorator, OpenIG can record two timer messages in the LogSink: the elapsed time for operations traversing the Filter, and the elapsed time spent within the Filter.

To record times spent within all Filters and the handler, decorate the Route as in the following example:

```
{
  "handler": {
    "type": "Chain",
    "config": {
      "filters": [
```

```
        {
          "type": "OAuth2ResourceServerFilter",
          "config": {
            "providerHandler": "ClientHandler",
            "scopes": [
              "mail",
              "employeenumber"
            ],
            "tokenInfoEndpoint":
"http://openam.example.com:8088/openam/oauth2/tokeninfo",
            "requireHttps": false
          },
          "capture": "filtered_request",
          "timer": true
        },
        {
          "type": "AssignmentFilter",
          "config": {
            "onRequest": [
              {
                "target": "${session.username}",
                "value": "${contexts.oauth2.accessToken.info.mail}"
              },
              {
                "target": "${session.password}",
                "value": "${contexts.oauth2.accessToken.info.employeenumber}"
              }
            ]
          },
          "timer": true
        },
        {
          "type": "StaticRequestFilter",
          "config": {
            "method": "POST",
            "uri": "http://app.example.com:8081",
            "form": {
              "username": [
                "${session.username}"
              ],
              "password": [
                "${session.password}"
              ]
            }
          },
          "timer": true
        }
      ],
      "handler": "ClientHandler"
    }
  },
```

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

This configuration could result in the following log messages:

```
THU DEC 11 16:06:23 CET 2014 (STAT)
@Timer[{OAuth2ResourceServerFilter}/handler/config/filters/0]
Started
------------------------------
THU DEC 11 16:06:23 CET 2014 (STAT)
@Timer[{AssignmentFilter}/handler/config/filters/1]
Started
------------------------------
THU DEC 11 16:06:23 CET 2014 (STAT)
@Timer[{StaticRequestFilter}/handler/config/filters/2]
Started
------------------------------
THU DEC 11 16:06:23 CET 2014 (STAT)
@Timer[{StaticRequestFilter}/handler/config/filters/2]
Elapsed time: 119 ms
------------------------------
THU DEC 11 16:06:23 CET 2014 (STAT)
@Timer[{StaticRequestFilter}/handler/config/filters/2]
Elapsed time (within the object): 1 ms
------------------------------
THU DEC 11 16:06:23 CET 2014 (STAT)
@Timer[{AssignmentFilter}/handler/config/filters/1]
Elapsed time: 128 ms
------------------------------
THU DEC 11 16:06:23 CET 2014 (STAT)
@Timer[{AssignmentFilter}/handler/config/filters/1]
Elapsed time (within the object): 7 ms
------------------------------
THU DEC 11 16:06:23 CET 2014 (STAT)
@Timer[{OAuth2ResourceServerFilter}/handler/config/filters/0]
Elapsed time: 211 ms
------------------------------
THU DEC 11 16:06:23 CET 2014 (STAT)
@Timer[{OAuth2ResourceServerFilter}/handler/config/filters/0]
Elapsed time (within the object): 81 ms
```

You can then deactivate the timer by setting the values to `false`:

```
{
    "timer": false
}
```

Javadoc

org.forgerock.openig.decoration.timer.TimerDecorator

Logging Framework

OpenIG uses the objects in this section to log events to the console or to files.

ConsoleLogSink — log to standard error

Description

A log sink that writes log entries to the standard error stream.

Usage

```
{
     "name": string,
     "type": "ConsoleLogSink",
     "config": {
         "level": string,
         "stream": string
     }
}
```

Properties

`"level"`: *string, optional*

The level of log entries.

Must be one of the following settings. These are ordered from most verbose to least verbose:

- `ALL` (log all messages)
- `TRACE` (log low-level tracing information)
- `DEBUG` (log debugging information)
- `STAT` (log performance measurement statistics)
- `CONFIG` (log configuration information)
- `INFO` (log general information)
- `WARNING` (log potential problems)
- `ERROR` (log serious failures)
- `OFF` (log no messages)

Default: `INFO`.

`"stream"`: *string, optional*

The standard output to use to display logs in the console.

Must be one of the following settings:

- `ERR` (use standard error: System.err)
- `OUT` (use standard output: System.out)
- `AUTO` (select standard error or output depending on the message log level: TRACE, DEBUG, STAT, CONFIG, INFO print to `System.out`; WARNING and ERROR print to `System.err`)

Default: `ERR`.

Example

```
{
     "name": "LogSink",
     "comment": "Default sink for logging information.",
     "type": "ConsoleLogSink",
     "config": {
         "level": "DEBUG",
         "stream": "AUTO"
     }
}
```

Javadoc

org.forgerock.openig.log.ConsoleLogSink

FileLogSink — log to a file

Description

A log sink that writes log entries to a file using the UTF-8 character set.

Usage

```
{
     "name": string,
     "type": "FileLogSink",
     "config": {
         "file": configuration expression,
         "level": string
     }
}
```

Properties

"`file`": ***configuration expression, required***

The path to the log file.

A configuration expression, described in Expressions(5) is independent of the request, response, and contexts, so do not use expressions that reference their properties. You can, however, use `${env['variable']}`, `${system['property']}`, and all the built-in functions listed in Functions(5).

`"level"`: *string, optional*

The level of log entries.

Must be one of the following settings. These are ordered from most verbose to least verbose:

- `ALL` (log all messages)
- `TRACE` (log low-level tracing information)
- `DEBUG` (log debugging information)
- `STAT` (log performance measurement statistics)
- `CONFIG` (log configuration information)
- `INFO` (log general information)
- `WARNING` (log potential problems)
- `ERROR` (log serious failures)
- `OFF` (log no messages)

Default: `INFO`.

Example

```
{
     "name": "LogSink",
     "type": "FileLogSink",
     "config": {
         "file": "${system['log'] ? system['log'] : '/tmp/proxy.log'}",
         "level": "DEBUG"
     }
}
```

Javadoc

org.forgerock.openig.log.FileLogSink

Slf4jLogSink — delegate log writing to SLF4J

Description

A log sink that delegates the writing of logs to SLF4J. OpenIG uses the Logback implementation of the SLF4J API. Use this log sink to define different logging behavior for routes and third-party dependencies.

A default configuration for logging is defined in OpenIG. To change the configuration, create a file `$HOME/.openig/config/logback.xml`. For a description of the available parameters, see the Logback website.

Usage

```
{
    "name": string,
    "type": "Slf4jLogSink",
    "config": {
        "base": string
    }
}
```

Properties

`"base"`: *string, optional*

The name for a logger that can be defined in `logback.xml`. The logger identifies a route or third-party dependency for which to define different logging behavior.

Logger names follow a hierarchical naming rule. When an object logs a message to `Slf4jLogSink`, a descendant logger is created the whose name is a concatenation of the base and the object name, separated with a `.`. For example, when an object `MyObject` logs a message to to an `Slf4jLogSink` with base `com.example.app`, a logger named `com.example.app.myobject` is created.

The hierarchical naming allows you to configure `logback.xml` with different logging characteristics for different components in a route.

Default: Empty string.

Example

In the following example, requests from the client filter, `MyObject`, create a logger called `com.example.app.myobject`.

```
{
   "name": "MyLogSink",
       "type": "Slf4jLogSink",
       "config": {
           "base": "com.example.app"
       }
},
{
   "name": "MyObject",
       "type": "OAuth2ClientFilter",
       "config": {
           "logSink": "MyLogSink"
  }
```

```
}
```

The following `logback.xml` sets the logging level to `DEBUG` for requests from the client filter, and to `INFO` for other requests with the base `com.example.app`.

```
<?xml version="1.0" encoding="UTF-8"?><configuration>

<appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">
 <encoder>
  <pattern>
   %d{HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n
  </pattern>
 </encoder>
</appender>

<logger name="com.example.app" level="INFO"/>
<logger name="com.example.app.myobject" level="DEBUG"/>

<root level="DEBUG">
 <appender-ref ref="STDOUT"/>
</root>

</configuration>
```

For an example configuration, see Separating Logs for Different Routes in the *Gateway Guide*.

Javadoc

org.forgerock.openig.log.Slf4jLogSink

Audit Framework

OpenIG uses the ForgeRock common audit framework to log system boundary events using an implementation that is common across the ForgeRock platform.

AuditService — enable common audit service for a route

Description

This object serves to configure the audit service for a route. The audit service uses the ForgeRock common audit event framework.

The route is decorated with an `auditService` field whose value references the configuration, either inline or from the heap.

Usage

```
{
     "name": string,
     "type": "AuditService",
     "config": {
         "config": object,
         "event-handlers": array
     }
}
```

Properties

`"config"`: *object, required*

This object configures the audit service itself, rather than event handlers. If the configuration uses only default settings, you can omit the field instead of including an empty object as the field value.

The configuration object has the following fields:

`"handlerForQueries"`: *string, optional*

This references the name of the event handler to use when querying audit event messages over REST.

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

This lists fully qualified event handler class names for event handlers available to the audit service.

`"filterPolicies"`: *object, optional*

These policies indicate what fields and values to include and to exclude from audit event

messages.

The filter policies object has these fields:

`"field"`: ***object, optional***

Audit event fields use JSON pointer notation, and are taken from the JSON schema for the audit event content.

Default: Include all fields.

The field object specifies which fields to include and to exclude:

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

This holds a list of audit event fields to exclude.

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

This holds a list of audit event fields to include.

`"value"`: ***object, optional***

Default: Include all messages.

The value object specifies field values based on which messages are included and excluded:

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

This holds a list of audit event field values.

When a value matches, the message is excluded.

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

This holds a list of audit event field values.

When a value matches, the message is included.

`"event-handlers"`: ***array of configuration objects, required***

This array of audit event handler configuration objects defines the event handlers that deal with audit events.

Each event handler configuration depends on type of the event handler.

OpenIG supports the following audit event handlers:

- CsvAuditEventHandler(5)
- JdbcAuditEventHandler(5)
- SyslogAuditEventHandler(5)
- ElasticsearchAuditEventHandler(5)

Example

The following example configures an audit service to log access event messages in a comma-separated variable file, named `/path/to/audit/logs/access.csv`:

```
{
    "name": "AuditService",
    "type": "AuditService",
    "config": {
        "config": {},
        "event-handlers": [
            {
                "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": "ClientHandler",
    "auditService": "AuditService"
}
```

Javadoc

org.forgerock.audit.AuditService

CsvAuditEventHandler — log audit events to CSV format files

Description

An audit event handler that responds to events by logging messages to files in comma-separated variable (CSV) format.

The configuration is declared in an audit service configuration. For details, see AuditService(5).

Usage

```
{
    "class": "org.forgerock.audit.handlers.csv.CsvAuditEventHandler",
    "config": {
        "name": string,
        "logDirectory": string,
        "topics": array,
        "enabled": boolean,
        "formatting": {
            "quoteChar": single-character string,
            "delimiterChar": single-character string,
            "endOfLineSymbols": string
        },
        "buffering": {
            "enabled": boolean,
            "autoFlush": boolean
        },
        "security": {
            "enabled": boolean,
            "filename": string,
            "password": string,
            "signatureInterval": duration
        },
        "fileRetention": {
            "maxDiskSpaceToUse": number,
            "maxNumberOfHistoryFiles": number,
            "minFreeSpaceRequired": number
        },
        "fileRotation": {
            "rotationEnabled": boolean,
            "maxFileSize": number,
            "rotationFilePrefix": string,
            "rotationFileSuffix": string,
            "rotationInterval": duration,
            "rotationTimes": array
        },
        "rotationRetentionCheckInterval": duration
    }
}
```

The values in this configuration object can use expressions as long as they resolve to the correct types for each field. For details about expressions, see Expressions(5).

Configuration

The `"config"` object has the following properties:

`"name"`: ***string, required***

The name of the event handler.

`"logDirectory"`: ***string, required***

The file system directory where log files are written.

`"topics"`: ***array of strings, required***

The topics that this event handler intercepts.

OpenIG handles access events that occur at the system boundary, such as arrival of the initial request and departure of the final response.

Set this to `"topics": [ "access" ]`.

`"enabled"`: ***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"`: ***single-character string, optional***

The character used to quote CSV entries.

Default: `"`.

`"delimiterChar"`: ***single-character string, optional***

The character used to delimit CSV entries.

Default: `,`.

`"endOfLineSymbols"`: ***string, optional***

The character or characters that separate a line.

Default: system-dependent line separator defined for the JVM.

`"buffering"`: ***object, optional***

Buffering settings for writing CSV log files. The default is for messages to be written to the log file for each event.

The buffering object has the following fields:

`"enabled"`: ***boolean, optional***

Whether log buffering is enabled.

Default: false.

`"autoFlush"`: ***boolean, optional***

Whether events are automatically flushed after being written.

Default: true.

`"security"`: ***object, optional***

Security settings for CSV log files. These settings govern tamper-evident logging, whereby messages are signed. By default tamper-evident logging is not enabled.

The security object has the following fields:

`"enabled"`: ***boolean, optional***

Whether tamper-evident logging is enabled.

Default: false.

Tamper-evident logging depends on a specially prepared keystore. For details, see "Preparing a Keystore for Tamper-Evident Logs".

`"filename"`: ***string, required***

File system path to the keystore containing the private key for tamper-evident logging.

The keystore must be a keystore of type `JCEKS`. For details, see "Preparing a Keystore for Tamper-Evident Logs".

`"password"`: ***string, required***

The password for the keystore for tamper-evident logging.

This password is used for the keystore and for private keys. For details, see "Preparing a Keystore for Tamper-Evident Logs".

`"signatureInterval"`: ***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.

A duration is a lapse of time expressed in English, such as `23 hours 59 minutes and 59 seconds`.

Durations are not case sensitive.

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`: microseconds
- `nanoseconds`, `nanosecond`, `nanosec`, `nanos`, `nano`, `ns`: nanoseconds

`"fileRetention"`: ***object, optional***

File retention settings for CSV log files.

The file retention object has the following fields:

`"maxDiskSpaceToUse"`: ***number, optional***

The maximum disk space in bytes the audit logs can occupy. A setting of 0 or less indicates that the policy is disabled.

Default: 0.

`"maxNumberOfHistoryFiles"`: ***number, optional***

The maximum number of historical log files to retain. A setting of -1 disables pruning of old history files.

Default: 0.

`"minFreeSpaceRequired"`: ***number, optional***

The minimum free space in bytes that the system must contain for logs to be written. A setting of 0 or less indicates that the policy is disabled.

Default: 0.

`"fileRotation"`: ***object, optional***

File rotation settings for CSV log files.

The file rotation object has the following fields:

`"rotationEnabled"`: ***boolean, optional***

Whether file rotation is enabled for CSV log files.

Default: false.

`"maxFileSize"`: ***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"`: ***string, optional***

The prefix to add to a log file on rotation.

This has an effect when time-based file rotation is enabled.

`"rotationFileSuffix"`: ***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"`: ***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.

A duration is a lapse of time expressed in English, such as `23 hours 59 minutes and 59 seconds`.

Durations are not case sensitive.

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`: microseconds
- `nanoseconds`, `nanosecond`, `nanosec`, `nanos`, `nano`, `ns`: nanoseconds

`"rotationTimes"`: ***array of 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.

+ A duration is a lapse of time expressed in English, such as `23 hours 59 minutes and 59 seconds`.

Durations are not case sensitive.

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`: microseconds
- `nanoseconds`, `nanosecond`, `nanosec`, `nanos`, `nano`, `ns`: nanoseconds

`"rotationRetentionCheckInterval"`: ***duration, optional***

The time interval after which to check file rotation and retention policies for updates.

Default: 5 seconds

A duration is a lapse of time expressed in English, such as `23 hours 59 minutes and 59 seconds`.

Durations are not case sensitive.

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`: microseconds
- `nanoseconds`, `nanosecond`, `nanosec`, `nanos`, `nano`, `ns`: nanoseconds

Preparing a Keystore for Tamper-Evident Logs

Tamper-evident logging depends on a public key/private key pair and on a secret key that are stored together in a JCEKS keystore. Follow these steps to prepare the keystore:

1. Generate a key pair in the keystore.

   The CSV event handler expects a JCEKS-type keystore with a key alias of `Signature` for the signing key, where the key is generated with the `RSA` key algorithm and the `SHA256withRSA` signature algorithm:

   ```
   $ keytool \
    -genkeypair \
    -keyalg RSA \
    -sigalg SHA256withRSA \
    -alias "Signature" \
    -dname "CN=openig.example.com,O=Example Corp,C=FR" \
    -keystore /path/to/audit-keystore \
    -storetype JCEKS \
    -storepass password \
    -keypass password
   ```

2. Generate a secret key in the keystore.

   The CSV event handler expects a JCEKS-type keystore with a key alias of `Password` for the symmetric key, where the key is generated with the `HmacSHA256` key algorithm and 256-bit key size:

   ```
   $ keytool \
    -genseckey \
    -keyalg HmacSHA256 \
    -keysize 256 \
    -alias "Password" \
    -keystore /path/to/audit-keystore \
    -storetype JCEKS \
    -storepass password \
    -keypass password
   ```

3. Verify the content of the keystore:

   ```
   $ keytool \
    -list \
    -keystore /path/to/audit-keystore \
    -storetype JCEKS \
    -storepass password

   Keystore type: JCEKS
   Keystore provider: SunJCE

   Your keystore contains 2 entries

   signature, Nov 27, 2015, PrivateKeyEntry,
   ```

```
Certificate fingerprint (SHA1): 4D:CF:CC:29:...:8B:6E:68:D1
password, Nov 27, 2015, SecretKeyEntry,
```

Example

For instructions on recording audit events in a CSV file, see To Record Audit Events In a CSV File in the *Gateway Guide*.

The following example configures a CSV audit event handler to write a log file, `/path/to/audit/logs/access.csv`, that is signed every 10 seconds to make it tamper-evident:

```
{
    "name": "csv",
    "topics": [
        "access"
    ],
    "logDirectory": "/path/to/audit/logs/",
    "security": {
        "enabled": "true",
        "filename": "/path/to/audit-keystore",
        "password": "password",
        "signatureInterval": "10 seconds"
    }
}
```

Javadoc

org.forgerock.audit.handlers.csv.CsvAuditEventHandler

JdbcAuditEventHandler — log audit events to relational database

Description

An audit event handler that responds to events by logging messages to an appropriately configured relational database table.

The configuration is declared in an audit service configuration. For details, see AuditService(5).

Usage

```
{
    "class": "org.forgerock.audit.handlers.jdbc.JdbcAuditEventHandler",
    "config": {
```

```
        "name": string,
        "topics": array,
        "databaseType": string,
        "enabled": boolean,
        "buffering": {
            "enabled": boolean,
            "writeInterval": duration,
            "autoFlush": boolean,
            "maxBatchedEvents": number,
            "maxSize": number,
            "writerThreads": number
        },
        "connectionPool": {
            "dataSourceClassName": string,
            "jdbcUrl": string,
            "username": string,
            "password": string,
            "autoCommit": boolean,
            "connectionTimeout": number,
            "idleTimeout": number,
            "maxLifetime": number,
            "minIdle": number,
            "maxPoolSize": number,
            "poolName": string
        },
        "tableMappings": [
            {
                "event": string,
                "table": string,
                "fieldToColumn": {
                    "event-field": "database-column"
                }
            }
        ]
    }
}
```

The values in this configuration object can use expressions as long as they resolve to the correct types for each field. For details about expressions, see Expressions(5).

Configuration

The `"config"` object has the following properties:

`"name"`: *string, required*

The name of the event handler.

`"topics"`: *array of strings, required*

The topics that this event handler intercepts.

OpenIG handles access events that occur at the system boundary, such as arrival of the initial request and departure of the final response.

Set this to `"topics": [ "access" ]`.

`"databaseType"`: ***string, required***

The database type name.

Built-in support is provided for `oracle`, `mysql`, and `h2`. Unrecognized database types rely on a GenericDatabaseStatementProvider.

`"enabled"`: ***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"`: ***boolean, optional***

Whether log buffering is enabled.

Default: false.

`"writeInterval"`: ***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.

A duration is a lapse of time expressed in English, such as `23 hours 59 minutes and 59 seconds`.

Durations are not case sensitive.

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`: microseconds
- `nanoseconds`, `nanosecond`, `nanosec`, `nanos`, `nano`, `ns`: nanoseconds

`"autoFlush"`: ***boolean, optional***

Whether the events are automatically flushed after being written.

Default: true.

`"maxBatchedEvents"`: ***number, optional***

The maximum number of event messages batched into a PreparedStatement.

Default: 100.

`"maxSize"`: ***number, optional***

The maximum size of the queue of buffered event messages.

Default: 5000.

`"writerThreads"`: ***number, optional***

The number of threads to write buffered event messages to the database.
Default: 1.

`"connectionPool"`: ***object, required***

Connection pool settings for sending messages to the database.

The connection pool object has the following fields:

`"dataSourceClassName"`: ***string, optional***

The class name of the data source for the database.

`"jdbcUrl"`: ***string, required***

The JDBC URL to connect to the database.

`"username"`: ***string, required***

The username identifier for the database user with access to write the messages.

`"password"`: ***number, optional***

The password for the database user with access to write the messages.

`"autoCommit"`: ***boolean, optional***

Whether to commit transactions automatically when writing messages.

Default: true.

`"connectionTimeout"`: ***number, optional***

The number of milliseconds to wait for a connection from the pool before timing out.

Default: 30000.

"idleTimeout": *number, optional*

The number of milliseconds to allow a database connection to remain idle before timing out.

Default: 600000.

"maxLifetime": *number, optional*

The number of milliseconds to allow a database connection to remain in the pool.

Default: 1800000.

"minIdle": *number, optional*

The minimum number of idle connections in the pool.

Default: 10.

"maxPoolSize": *number, optional*

The maximum number of connections in the pool.

Default: 10.

"poolName": *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": *string, required*

The audit event that the table mapping is for.

Set this to `access`.

"table": *string, required*

The name of the database table that corresponds to the mapping.

"fieldToColumn": *object, required*

This object maps the 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

The following example configures a JDBC audit event handler using a local MySQL database, writing to a table named `auditaccess`:

```
{
```

```
    "class": "org.forgerock.audit.handlers.jdbc.JdbcAuditEventHandler",
    "config": {
        "databaseType": "mysql",
        "name": "jdbc",
        "topics": [
            "access"
        ],
        "connectionPool": {
            "jdbcUrl":
"jdbc:mysql://localhost:3306/audit?allowMultiQueries=true&characterEncoding=utf8",
            "username": "audit",
            "password": "audit"
        },
        "tableMappings": [
            {
                "event": "access",
                "table": "auditaccess",
                "fieldToColumn": {
                    "_id": "id",
                    "timestamp": "timestamp_",
                    "eventName": "eventname",
                    "transactionId": "transactionid",
                    "userId": "userid",
                    "trackingIds": "trackingids",
                    "server/ip": "server_ip",
                    "server/port": "server_port",
                    "client/host": "client_host",
                    "client/ip": "client_ip",
                    "client/port": "client_port",
                    "request/protocol": "request_protocol",
                    "request/operation": "request_operation",
                    "request/detail": "request_detail",
                    "http/request/secure": "http_request_secure",
                    "http/request/method": "http_request_method",
                    "http/request/path": "http_request_path",
                    "http/request/queryParameters": "http_request_queryparameters",
                    "http/request/headers": "http_request_headers",
                    "http/request/cookies": "http_request_cookies",
                    "http/response/headers": "http_response_headers",
                    "response/status": "response_status",
                    "response/statusCode": "response_statuscode",
                    "response/elapsedTime": "response_elapsedtime",
                    "response/elapsedTimeUnits": "response_elapsedtimeunits"
                }
            }
        ]
    }
}
```

Examples including statements to create tables are provided in the JDBC handler library, `forgerock-`

`audit-handler-jdbc-version.jar`, that is built into the OpenIG .war file. Unpack the library, then find the examples under the `db/` folder.

Javadoc

org.forgerock.audit.handlers.jdbc.JdbcAuditEventHandler

SyslogAuditEventHandler — log audit events to the system log

Description

An audit event handler that responds to events by logging messages to the UNIX system log as governed by RFC 5424, The Syslog Protocol.

The configuration is declared in an audit service configuration. For details, see AuditService(5).

Usage

```
{
    "class": "org.forgerock.audit.handlers.syslog.SyslogAuditEventHandler",
    "config": {
        "name": string,
        "topics": array,
        "protocol": string,
        "host": string,
        "port": number,
        "connectTimeout": number,
        "facility": "string",
        "buffering": {
            "enabled": boolean,
            "maxSize": number
        },
        "severityFieldMappings": [
            {
                "topic": string,
                "field": string,
                "valueMappings": {
                    "field-value": "syslog-severity"
                }
            }
        ]
    }
}
```

The values in this configuration object can use expressions as long as they resolve to the correct

types for each field. For details about expressions, see Expressions(5).

Configuration

The `"config"` object has the following properties:

`"name"`: *string, required*

The name of the event handler.

`"topics"`: *array of strings, required*

The topics that this event handler intercepts.

OpenIG handles access events that occur at the system boundary, such as arrival of the initial request and departure of the final response.

Set this to `"topics": [ "access" ]`.

`"protocol"`: *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"`: *string, required*

The hostname of the Syslog daemon to which to send event messages. The hostname must resolve to an IP address.

`"port"`: *number, required*

The port of the Syslog daemon to which to send event messages.

The value must be between 0 and 65535.

`"connectTimeout"`: *number, required when using TCP*

The number of milliseconds to wait for a connection before timing out.

`"facility"`: *string, required*

The Syslog facility to use for event messages.

Set this to one of the following values:

`kern`

Kernel messages

`user`

User-level messages

`mail`

Mail system

`daemon`
: System daemons

`auth`
: Security/authorization messages

`syslog`
: Messages generated internally by `syslogd`

`lpr`
: Line printer subsystem

`news`
: Network news subsystem

`uucp`
: UUCP subsystem

`cron`
: Clock daemon

`authpriv`
: Security/authorization messages

`ftp`
: FTP daemon

`ntp`
: NTP subsystem

`logaudit`
: Log audit

`logalert`
: Log alert

`clockd`
: Clock daemon

`local0`
: Local use 0

`local1`
: Local use 1

`local2`
: Local use 2

`local3`
: Local use 3

`local4`
: Local use 4

`local5`
: Local use 5

`local6`
: Local use 6

`local7`
: Local use 7

`"buffering"`: ***object, optional***
: Buffering settings for writing to the system log facility. The default is for messages to be written to the log for each event.

  The buffering object has the following fields:

  `"enabled"`: ***boolean, optional***
  : Whether log buffering is enabled.

    Default: false.

  `"maxSize"`: ***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"`: ***string, required***
  : The audit event topic to which the mapping applies.

    Set this to `access`.

  `"field"`: ***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": {
        "protocol": "TCP",
        "host": "https://syslogd.example.com",
        "port": 6514,
        "connectTimeout": 30000,
        "facility": "local0",
        "severityFieldMappings": [
            {
                "topic": "access",
                "field": "response/status",
                "valueMappings": {
```

```
                    "FAILED": "INFORMATIONAL",
                    "SUCCESSFUL": "INFORMATIONAL"
                }
            }
        ]
    }
}
```

Javadoc

org.forgerock.audit.handlers.syslog.SyslogAuditEventHandler

ElasticsearchAuditEventHandler — log audit events in the Elasticsearch search and analytics engine

Description

An audit event handler that responds to events by logging messages in the Elasticsearch search and analytics engine.

The configuration is declared in an audit service configuration. For information, see AuditService(5).

For Elasticsearch downloads and installation instructions, see the Elasticsearch Getting Started document.

A special client handler called `ElasticsearchClientHandler` can be defined to send audit events to Elasticsearch. You can use this client handler to capture the exchange between the audit service and Elasticsearch, or to wrap the search with a filter, for example, the `OAuth2ClientFilter`.

To define an `ElasticsearchClientHandler`, create the following object in the heap for the Elasticsearch audit event handler

```
{
  "name": "ElasticsearchClientHandler",
  "type": "ClientHandler",
  "config": {},
}
```

Usage

```
{
  "class": "org.forgerock.audit.handlers.elasticsearch.ElasticsearchAuditEventHandler
",
  "config": {
```

```
    "connection" : {
      "host" : string,
      "port" : number,
      "useSSL" : boolean,
      "username" : string,
      "password" : string
    },
    "indexMapping" : {
      "indexName" : string
    },
    "buffering" : {
      "enabled" : boolean,
      "writeInterval" : duration,
      "maxSize" : number,
      "maxBatchedEvents" : number
    },
    "topics" : [ string, ... ]
  }
}
```

The values in this configuration object can use expressions if they resolve to the correct types for each field. For information about expressions, see Expressions(5).

Properties

The `"config"` object has the following properties:

`"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"`: ***string, optional***

Hostname or IP address of Elasticsearch. The hostname must resolve to an IP address.

Default: `localhost`

`"port"`: ***number, optional***

The port used by Elasticsearch. The value must be between 0 and 65535.

Default: `9200`

`"useSSL"`: ***boolean, optional***

Setting to use or not use SSL/TLS to connect to Elasticsearch.

Default: `false`

`"username"`: ***string, optional***

Username when Basic Authentication is enabled through Elasticsearch Shield.

`"password"`: ***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"`: ***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"`: ***boolean, optional***

Setting to use or not use log buffering.

Default: false.

`"writeInterval"`: ***duration, required if buffering is enabled***

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

A duration is a lapse of time expressed in English, such as `23 hours 59 minutes and 59 seconds`.

Durations are not case sensitive.

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`: microseconds
- `nanoseconds`, `nanosecond`, `nanosec`, `nanos`, `nano`, `ns`: nanoseconds

"maxBatchedEvents": ***number, optional***

The maximum number of event messages in a batch write to Elasticsearch for each `writeInterval`.

Default: 500

"maxSize": ***number, optional***

The maximum number of event messages in the queue of buffered event messages.
Default: 10000

"topics": ***array of strings, required***

The topics that this event handler intercepts.

OpenIG handles access events that occur at the system boundary, such as arrival of the initial request and departure of the final response.

Set this to `"topics": [ "access" ]`.

Example

For instructions on recording audit events in Elasticsearch, see To Record Audit Events In Elasticsearch in the *Gateway Guide*.

The following example configures an Elasticsearch audit event handler:

```
{
  "class" : "
org.forgerock.audit.handlers.elasticsearch.ElasticsearchAuditEventHandler",
  "config" : {
    "connection" : {
      "useSSL" : false,
      "host" : "localhost",
      "port" : "9200"
    },
    "indexMapping" : {
      "indexName" : "audit"
    },
    "buffering" : {
      "enabled" : false,
      "maxSize" : 20000,
      "writeInterval" : "1 second",
      "maxBatchedEvents" : "500"
    },
    "topics" : [
      "access"
    ]
  }
}
```

Throttling Filters and Policies

To protect applications from being overused by clients, use a throttling filter to limit how many requests clients can make in a defined time.

ThrottlingFilter — limit the rate of requests

Description

Limits the rate that requests pass through a filter. The maximum number of requests that a client is allowed to make in a defined time is called the *throttling rate*.

The throttling filter uses a strategy based on the token bucket algorithm, which allows some bursts. Because of traffic bursts, the throttling rate can occasionally be higher than the defined limit - for example, with a throttling rate of 10 requests/10 seconds there can be more than 10 requests in the 10 second duration. However, the number of concurrent requests cannot exceed that defined for the throttling rate - for example, with a throttling rate of 10 requests/10 seconds there cannot be more than 10 concurrent requests.

When the throttling rate is reached, OpenIG 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://openig.example.com:8080/throttle-scriptable HTTP/1.1

    . . .

    HTTP/1.1 429 Too Many Requests
    Retry-After: 10
```

Usage

```
{
    "name": string,
    "type": "ThrottlingFilter",
    "config": {
        "requestGroupingPolicy": expression,
        "throttlingRatePolicy": reference or inline declaration,  //Use either
"policy"
        "rate": {                                                 //or "rate", but not
both.
            "numberOfRequests": integer,
            "duration": duration string
        },
        "cleaningInterval": duration string,
        "executor": executor
    }
```

```
}
```

Properties

`"requestGroupingPolicy"`: ***expression, required***

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.

Default: Empty string. The value for this expression must not be null.

See also Expressions(5).

`"throttlingRatePolicy"`: ***reference or inline declaration, required if "rate" is not used***

A reference to or inline declaration of a policy to apply for throttling rate. The following policies can be used:

- MappedThrottlingPolicy(5)
- ScriptableThrottlingPolicy(5)
- DefaultRateThrottlingPolicy(5)

This value for this parameter must not be null.

`"rate"`: ***rate object, required if "throttlingRatePolicy" is not used***

The throttling rate to apply to requests. The rate is calculated as the number of requests divided by the duration.

`"numberOfRequests"`: ***integer, required***

The number of requests allowed through the filter in the time specified by `"duration"`.

`"duration"`: ***duration string, required***

A time interval during which the number of requests passing through the filter is counted.

A duration is a lapse of time expressed in English, such as `23 hours 59 minutes and 59 seconds`.

Durations are not case sensitive.

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`: microseconds
- `nanoseconds`, `nanosecond`, `nanosec`, `nanos`, `nano`, `ns`: nanoseconds

`"cleaningInterval"`: ***duration, optional***

The time to wait before cleaning outdated partitions. The value must be more than zero but not more than one day.

`"executor"`: ***executor, optional***

An executor service to schedule the execution of tasks, such as the clean up of partitions that are no longer used.

Default: `ScheduledExecutorService`

See also ScheduledExecutorService(5).

Examples

The following links provide examples of how the throttling policies are implemented:

- "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 Configuring a Simple Throttling Filter in the *Gateway Guide*.

```
{
  "handler": {
    "type": "Chain",
    "config": {
      "filters": [
        {
          "type": "ThrottlingFilter",
          "config": {
            "requestGroupingPolicy": "${request.headers['UserId'][0]}",
            "rate": {
              "numberOfRequests": 6,
              "duration": "10 seconds"
            }
          }
        }
      ],
      "handler": "ClientHandler"
    }
  },
  "condition": "${matches(request.uri.path, '^/throttle-simple')}"
```

```
}
```

Javadoc

org.forgerock.openig.filter.throttling.ThrottlingFilterHeaplet

---

MappedThrottlingPolicy — map throttling rates to groups of requests

Description

Maps different throttling rates to different groups of requests, according to the evaluation of `throttlingRateMapper`.

Usage

```
{
    "type": "ThrottlingFilter",
    "config": {
        "requestGroupingPolicy": expression,
        "throttlingRatePolicy": {
            "type": "MappedThrottlingPolicy",
            "config": {
                "throttlingRateMapper": expression<string>,
                "throttlingRatesMapping": {
                    "mapping1": {
                        "numberOfRequests": integer,
                        "duration": duration string
                    },
                    "mapping2": {
                        "numberOfRequests": integer,
                        "duration": duration string
                    }
                },
                "defaultRate": {
                    "numberOfRequests": integer,
                    "duration": duration string
                }
            }
        }
    }
}
```

Properties

`"throttlingRateMapper"`: ***expression, 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"`.

`"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.

The number of mappings is not limited to two.

`"numberOfRequests"`: ***integer, required***

The number of requests allowed through the filter in the time specified by `"duration"`.

`"duration"`: ***duration string, required***

A time interval during which the number of requests passing through the filter is counted.

A duration is a lapse of time expressed in English, such as `23 hours 59 minutes and 59 seconds`.

Durations are not case sensitive.

Negative durations are not supported.

The following units can be used in durations:

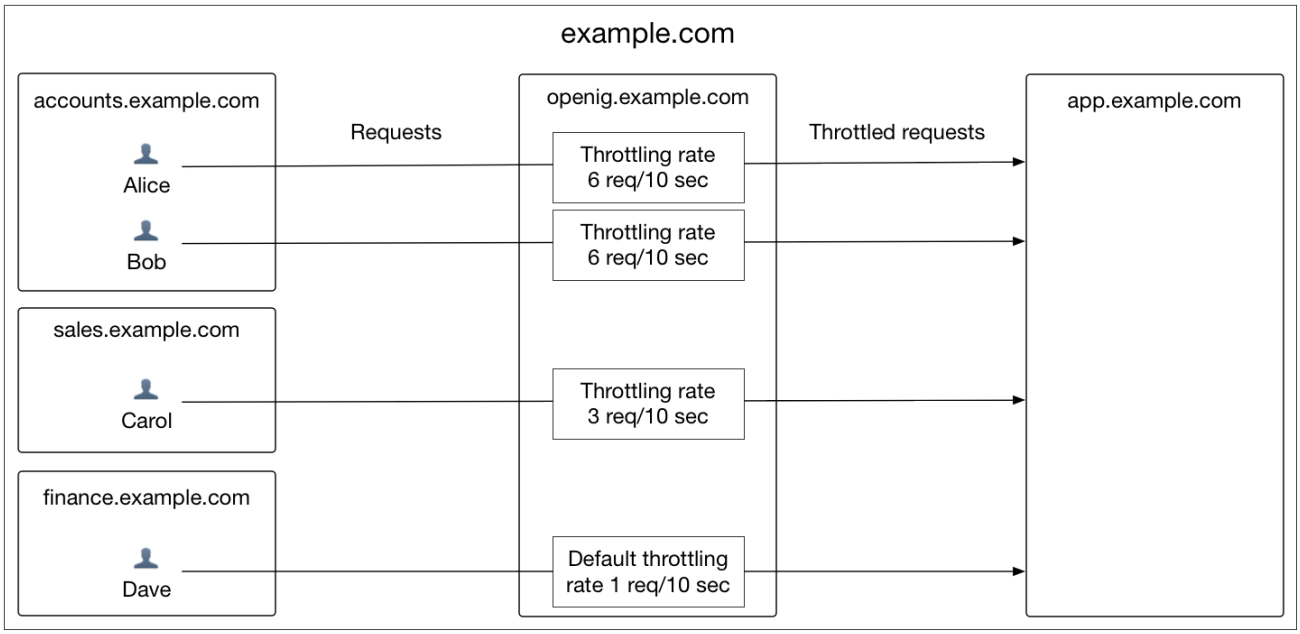
- `indefinite`, `infinity`, `undefined`, `unlimited`: unlimited duration
- `zero`, `disabled`: zero-length duration
- `days`, `day`, `d`: days
- `hours`, `hour`, `h`: hours
- `minutes`, `minute`, `min`, `m`: minutes
- `seconds`, `second`, `sec`, `s`: seconds
- `milliseconds`, `millisecond`, `millisec`,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

Example of a Mapped Throttling Policy

In the following example, requests from users in the accounts and sales departments of `example.com` are mapped to different throttling rates. Requests from other departments use the default throttling rate. For information about how to set up and test this example, see Configuring a Mapped Throttling Filter in the *Gateway Guide*.

Alice and Bob both send requests from accounts, and so they each have a throttling rate of 6 requests/10 seconds. The throttling rate is applied independently to Alice and Bob, so no matter how many requests Alice sends in 10 seconds, Bob can still send up to 6 requests in the same 10 seconds. Carol sends requests from sales, with a throttling rate of 3 requests/10 seconds. Dave sends requests from finance, with the default rate of 1 request/10 seconds.

The throttling rate is assigned according to the evaluation of `throttlingRateMapper`. In the example, this parameter evaluates to the value of the request header `X-Forwarded-For`, representing the hostname of the department.




```
{
  "handler": {
    "type": "Chain",
    "config": {
      "filters": [
        {
          "type": "ThrottlingFilter",
          "config": {
            "requestGroupingPolicy": "${request.headers['UserId'][0]}",
            "throttlingRatePolicy": {
              "type": "MappedThrottlingPolicy",
              "config": {
                "throttlingRateMapper": "${request.headers['X-Forwarded-For'][0]}",
                "throttlingRatesMapping": {
                  "accounts.example.com": {
```

```
                  "numberOfRequests": 6,
                  "duration": "10 seconds"
                },
                "sales.example.com": {
                  "numberOfRequests": 3,
                  "duration": "10 seconds"
                }
              },
              "defaultRate": {
                "numberOfRequests": 1,
                "duration": "10 seconds"
              }
            }
          }
        }
      }
    ],
    "handler": "ClientHandler"
  }
},
"condition": "${matches(request.uri.path, '^/throttle-mapped')}"
}
```

Javadoc

org.forgerock.openig.filter.throttling.MappedThrottlingPolicyHeaplet

---

ScriptableThrottlingPolicy — script to map throttling rates

Description

Uses a script to look up throttling rates to apply to groups of requests.

Usage

```
{
   "type": "ThrottlingFilter",
   "config": {
       "requestGroupingPolicy": expression,
       "throttlingRatePolicy": {
           "type": "ScriptableThrottlingPolicy",
           "config": {
               "type": string,
               "file": string,     // Use either "file"
               "source": string    // or "source", but not both
```

```
            }
        }
    }
}
```

Properties

`"type"`: *string, required*

The Internet media type (formerly MIME type) of the script. For Groovy, the value is `"application/x-groovy"`.

`"file"`: *string, required if "source" is not used*

The path to the file containing the script.

Relative paths in this field are relative to the base location for scripts, which depends on the configuration. For information, see Installing OpenIG in the *Gateway Guide*.

The base location for Groovy scripts is on the classpath when the scripts are executed. If a Groovy script is not in the default package, but instead has its own package name, it belongs in the directory corresponding to the package name. For example, a script in package `com.example.groovy` belongs under `openig-base/scripts/groovy/com/example/groovy/`.

`"source"`: *string, required if "file" is not used*

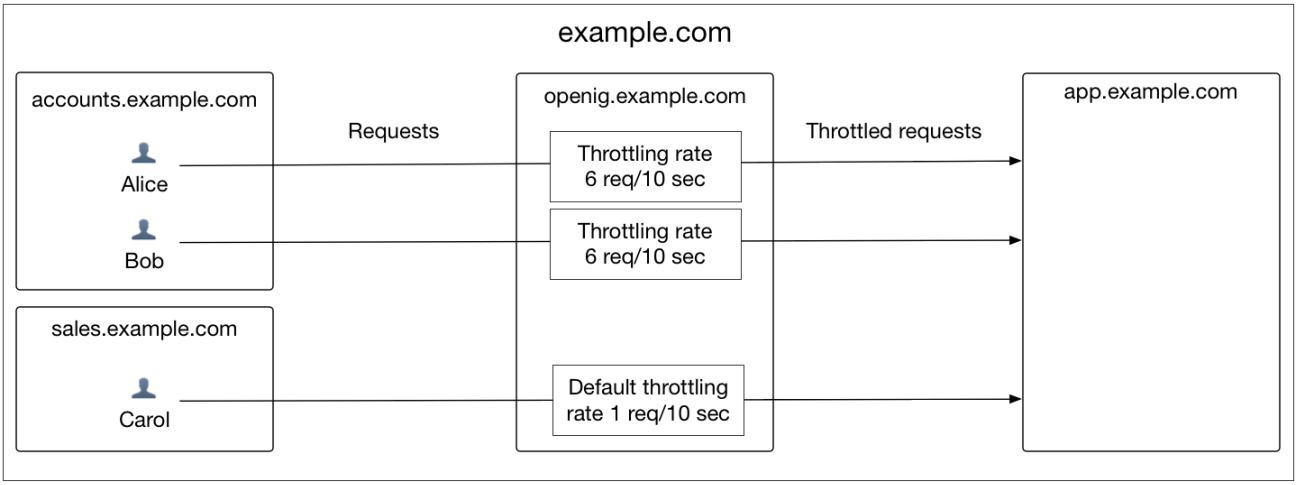
The script as a string.

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 Configuring a Scriptable Throttling Filter in the *Gateway Guide*.

The script applies a throttling rate of 6 requests/10 seconds to requests from the accounts department of `example.com`. For all other requests, the script returns `null`. When the script returns `null`, the default rate of 1 request/10 seconds is applied.

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, query an external database, or read the mapping from a file.




```
{
  "handler": {
    "type": "Chain",
    "config": {
      "filters": [
        {
          "type": "ThrottlingFilter",
          "config": {
            "requestGroupingPolicy": "${request.headers['UserId'][0]}",
            "throttlingRatePolicy": {
              "type": "DefaultRateThrottlingPolicy",
              "config": {
                "delegateThrottlingRatePolicy": {
                  "type": "ScriptableThrottlingPolicy",
                  "config": {
                    "type": "application/x-groovy",
                    "file": "ThrottlingScript.groovy"
                  }
                },
                "defaultRate": {
                  "numberOfRequests": 1,
                  "duration": "10 seconds"
                }
              }
            }
          }
        }
      ],
      "handler": "ClientHandler"
    }
  },
  "condition": "${matches(request.uri.path, '^/throttle-scriptable')}"
}
```

The groovy script maps a throttling rate for the accounts department of `example.com`. Other requests receive the default throttling rate.

```
/**
 * ThrottlingScript.groovy
 *
 * Script to throttle access for requests from the accounts department
 * of example.com. Other requests return null.
 */

if (request.headers['X-Forwarded-For'].values[0]  == 'accounts.example.com') {
    return new ThrottlingRate(6, '10 seconds')
} else {
    return null
}
```

Javadoc

org.forgerock.openig.filter.throttling.ScriptableThrottlingPolicy.Heaplet

---

DefaultRateThrottlingPolicy — default policy for throttling rate

Description

Provides a default throttling rate if the delegating throttling policy returns `null`.

Usage

```
{
   "type": "ThrottlingFilter",
   "config": {
       "requestGroupingPolicy": expression,
       "throttlingRatePolicy": {
           "type": "DefaultRateThrottlingPolicy",
           "config": {
               "delegateThrottlingRatePolicy" : reference or inline declaration,
               "defaultRate": {
                   "numberOfRequests": integer,
                   "duration": duration string
               }
           }
       }
   }
}
```

Properties

`"delegateThrottlingRatePolicy"`: *reference, required*

The policy to which the default policy delegates the throttling rate. The `DefaultRateThrottlingPolicy` delegates management of throttling to the policy specified by `delegateThrottlingRatePolicy`.

If `delegateThrottlingRatePolicy` returns `null`, the `defaultRate` is used.

For information about policies to use, see MappedThrottlingPolicy(5) and ScriptableThrottlingPolicy(5).

`"defaultRate"`: *object, required*

The default throttling rate to apply if the delegating policy returns `null`.

`"numberOfRequests"`: *integer, required*

The number of requests allowed through the filter in the time specified by `"duration"`.

`"duration"`: *duration string, required*

A time interval during which the number of requests passing through the filter is counted.

A duration is a lapse of time expressed in English, such as `23 hours 59 minutes and 59 seconds`.

Durations are not case sensitive.

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`: microseconds
- `nanoseconds`, `nanosecond`, `nanosec`, `nanos`, `nano`, `ns`: nanoseconds

Example

For an example of how this policy is used, see "Example of a Scriptable Throttling Policy" .

Javadoc

org.forgerock.openig.filter.throttling.DefaultRateThrottlingPolicyHeaplet

Miscellaneous Heap Objects

ClientRegistration — Hold OAuth 2.0 client registration information

Description

A ClientRegistration holds information about registration with an OAuth 2.0 authorization server or OpenID Provider.

The configuration includes the client credentials that are used to authenticate to the identity provider. The client credentials can be included directly in the configuration, or retrieved in some other way using an expression, described in Expressions(5).

Usage

```
{
  "name": string,
  "type": "ClientRegistration",
  "config": {
    "clientId": expression,
    "clientSecret": expression,
    "issuer": Issuer reference,
    "registrationHandler": Handler reference,
    "scopes": [ expression, ...],
    "tokenEndpointUseBasicAuth": boolean
  }
}
```

Properties

The client registration configuration object properties are as follows:

`"name"`: ***string, required***

A name for the client registration.

`"clientId"`: ***expression, required***

The `client_id` obtained when registering with the authorization server.

See also Expressions(5).

`"clientSecret"`: ***expression, required***

The `client_secret` obtained when registering with the authorization server.

See also Expressions(5).

`"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(5).

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

Invoke this HTTP client handler to communicate with the authorization server.

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

Usually set this to the name of a ClientHandler configured in the heap, or a chain that ends in a ClientHandler.

Default: OpenIG uses the default ClientHandler.

See also Handlers, ClientHandler(5).

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

OAuth 2.0 scopes to use with this client registration.

See also Expressions(5).

`"tokenEndpointUseBasicAuth"`: ***boolean, optional***

Whether to perform client authentication to the provider using HTTP Basic authentication when sending a request to the provider's OAuth 2.0 token endpoint.

When set to `true`, the client credentials are sent using HTTP Basic authentication as in the following example request:

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

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

When set to `false`, the client credentials are sent in HTTP POST form data as in the following example request:

```
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=...
```

Some providers accept both authentication methods. For providers that strictly enforce how the client must authenticate, such as recent versions of OpenAM, you must align the configuration with that of the provider.

If the provider does not support the configured authentication method, then according to RFC 6749 The OAuth 2.0 Authorization Framework the provider sends an HTTP 400 Bad Request response with an `invalid_client` error message as in the following example response:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store
Pragma: no-cache

{
  "error":"invalid_client"
}
```

Default: `true`

Example

The following example shows a client registration for OpenAM. In this example client credentials are replaced with . In the actual configuration either include the credentials and protect the configuration file or obtain the credentials from the environment in a safe manner:

```
{
    "name": "OpenIDConnectRelyingParty",
    "type": "ClientRegistration",
    "config": {
        "clientId": "**********",
        "clientSecret": "**********",
        "issuer": "openam",
        "redirect_uris": [
            "https://openig.example.com:8443/openid/callback"
        ],
        "scopes": [
            "openid",
            "profile"
        ]
    }
}
```

Javadoc

org.forgerock.openig.filter.oauth2.client.ClientRegistration

See Also

Issuer(5), OAuth2ClientFilter(5)

The OAuth 2.0 Authorization Framework

OAuth 2.0 Bearer Token Usage

OpenID Connect

JwtSession — store sessions in encrypted JWT cookies

Description

A JwtSession object holds settings for storing session information in encrypted JSON Web Token (JWT) cookies.

In this context, *encrypted JWT cookie* means an HTTP cookie whose value is an encrypted JWT. The payload of the encrypted JWT is a JSON representation of the session information.

The JWT cookie lifetime is Session (not persistent), meaning the user-agent deletes the JWT cookie when it shuts down.

When using this storage implementation, you must use data types for session information that can be mapped to JavaScript Object Notation (JSON). JSON allows strings, numbers, `true`, `false`, `null`, as well as arrays and JSON objects composed of the same primitives. Java and Groovy types that can be mapped include Java primitive types and `null`, String and CharSequence objects, as well as List and Map objects.

As browser cookie storage capacity is limited to 4 KB, and encryption adds overhead, take care to limit the size of any JSON that you store. Rather than store larger data in the session information, consider storing a reference instead.

When a request enters a route that uses a new session type, the scope of the session information becomes limited to the route. OpenIG builds a new session object and does not propagate any existing session information to the new object. `session` references the new session object. When the response then exits the route, the session object is closed, and serialized to a JWT cookie in this case, and `session` references the previous session object. Session information set inside the route is no longer available.

An HTTP client that performs multiple requests in a session that modify the content of its session can encounter inconsistencies in the session information. This is because OpenIG does not share JwtSessions across threads. Instead, each thread has its own JwtSession objects that it modifies as necessary, writing its own session to the JWT cookie regardless of what other threads do.

Usage

```
{
```

```
    "name": string,
    "type": "JwtSession",
    "config": {
        "keystore": KeyStore reference,
        "alias": string,
        "password": configuration expression,
        "cookieName": string,
        "sessionTimeout": duration,
        "sharedSecret": string
    }
}
```

An alternative value for type is JwtSessionFactory.

Properties

"keystore": *KeyStore reference, optional*

The keystore holding the key pair with the private key used to decrypt 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, OpenIG generates a unique key pair, and stores the key pair in memory. With JWTs encrypted using a unique key pair generated at runtime, OpenIG cannot decrypt the JWTs after a restart, nor can it decrypt such JWTs encrypted by another OpenIG server.

See also KeyStore(5).

"alias": *string, required when keystore is used*

Alias for the private key.

"password": *configuration expression, required when keystore is used*

The password to read the private key from the keystore.

A configuration expression, described in Expressions(5) is independent of the request, response, and contexts, so do not use expressions that reference their properties. You can, however, use `${env['variable']}`, `${system['property']}`, and all the built-in functions listed in Functions(5).

"cookieName" *string, optional*

The name of the JWT cookie stored on the user-agent.

Default: `openig-jwt-session`

"sessionTimeout" *duration, optional*

The amount of time before the cookie session expires.

A duration is a lapse of time expressed in English, such as `23 hours 59 minutes and 59 seconds`.

Durations are not case sensitive.

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`: microseconds
- `nanoseconds`, `nanosecond`, `nanosec`, `nanos`, `nano`, `ns`: nanoseconds

Default: 30 minutes

A zero duration for session timeout is not a valid setting. The maximum session timeout duration is 3650 days (approximately 10 years). If you set a longer duration, OpenIG truncates the duration to the maximum value.

`"sharedSecret"` ***string, optional***

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 OpenIG instance can verify only the sessions it has created.

Example

The following example defines a JwtSession for storing session information in a JWT token cookie named `OpenIG`. The JWT is encrypted with a private key that is recovered using the alias `private-key`, and stored in the keystore. The password is both the password for the keystore and also the private key:

```
{
    "name": "JwtSession",
    "type": "JwtSession",
    "config": {
        "keystore": {
            "type": "KeyStore",
            "config": {
                "url": "file://${env['HOME']}/keystore.jks",
                "password": "${system['keypass']}"
            }
```

```
        },
        "alias": "private-key",
        "password": "${system['keypass']}",
        "cookieName": "OpenIG"
    }
}
```

Javadoc

org.forgerock.openig.jwt.JwtSessionManager

KeyManager — configure a Java Secure Socket Extension KeyManager

Description

This represents the configuration for a Java Secure Socket Extension KeyManager, which manages the keys used to authenticate an SSLSocket to a peer. The configuration references the keystore that actually holds the keys.

Usage

```
{
    "name": string,
    "type": "KeyManager",
    "config": {
        "keystore": KeyStore reference,
        "password": expression,
        "alg": string
    }
}
```

Properties

"keystore": ***KeyStore reference, optional***

The keystore that references the store for the actual keys.

Provide either the name of the KeyStore object defined in the heap, or the inline KeyStore configuration object inline.

See also KeyStore(5).

"password": ***expression, required***

The password to read private keys from the keystore.

`"alg"` ***string, optional***

The certificate algorithm to use.

Default: the default for the platform, such as `SunX509`.

See also Expressions(5).

Example

The following example configures a key manager that depends on a KeyStore configuration. The keystore takes a password supplied as a Java system property when starting the container where OpenIG runs, as in `-Dkeypass=password`. This configuration uses the default certificate algorithm:

```
{
    "name": "MyKeyManager",
    "type": "KeyManager",
    "config": {
        "keystore": {
            "type": "KeyStore",
            "config": {
                "url": "file://${env['HOME']}/keystore.jks",
                "password": "${system['keypass']}"
            }
        },
        "password": "${system['keypass']}"
    }
}
```

Javadoc

org.forgerock.openig.security.KeyManagerHeaplet

See Also

JSSE Reference Guide, KeyStore(5), TrustManager(5)

KeyStore — configure a Java KeyStore

Description

This represents the configuration for a Java KeyStore, which stores cryptographic private keys and public key certificates.

Usage

```
{
```

```
    "name": name,
    "type": "KeyStore",
    "config": {
        "url": expression,
        "password": expression,
        "type": string
    }
}
```

Properties

"url": *expression, required*

URL to the keystore file.

See also Expressions(5).

"password": *expression, optional*

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.

See also Expressions(5).

"type": *string, optional*

The keystore format.

Default: the default for the platform, such as `JKS`.

Example

The following example configures a keystore that references a Java Keystore file, `$HOME/keystore.jks`. The keystore takes a password supplied as a Java system property when starting the container where OpenIG runs, as in `-Dkeypass=password`. As the keystore file uses the default format, no type is specified:

```
{
    "name": "MyKeyStore",
    "type": "KeyStore",
    "config": {
        "url": "file://${env['HOME']}/keystore.jks",
        "password": "${system['keypass']}"
    }
}
```

Javadoc

org.forgerock.openig.security.KeyStoreHeaplet

See Also

JSSE Reference Guide, KeyManager(5), TrustManager(5)

Issuer — Describe an Authorization Server or OpenID Provider

Description

An Issuer describes an OAuth 2.0 Authorization Server or an OpenID Provider that OpenIG can use as a OAuth 2.0 client or OpenID Connect relying party.

An Issuer is generally referenced from a ClientRegistration, described in ClientRegistration(5).

Usage

```
{
  "name": string,
  "type": "Issuer",
  "config": {
    "wellKnownEndpoint": URL string,
    "authorizeEndpoint": URI expression,
    "registrationEndpoint": URI expression,
    "tokenEndpoint": URI expression,
    "userInfoEndpoint": URI expression,
    "issuerHandler": Handler reference,
    "supportedDomains": [ domain 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"`: ***URL string, required unless authorizeEndpoint and tokenEndpoint are specified***

The URL to the well-known configuration resource as described in OpenID Connect 1.0 Discovery.

`"authorizeEndpoint"`: ***expression, required unless obtained through wellKnownEndpoint***

The URL to the provider's OAuth 2.0 authorization endpoint.

See also Expressions(5).

`"registrationEndpoint"`: ***expression, optional***

The URL to the provider's OpenID Connect dynamic registration endpoint.

See also Expressions(5).

`"tokenEndpoint"`: ***expression, required unless obtained through wellKnownEndpoint***

The URL to the provider's OAuth 2.0 token endpoint.

See also Expressions(5).

`"userInfoEndpoint"`: ***expression, optional***

The URL to the provider's OpenID Connect UserInfo endpoint.

Default: no UserInfo is obtained from the provider.

See also Expressions(5).

`"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: OpenIG uses the default ClientHandler.

See also Handlers, ClientHandler(5).

`"supportedDomains"`: ***array of patterns, optional***

List of patterns matching domain names handled by this issuer, used as a shortcut for OpenID Connect discovery before performing OpenID Connect dynamic registration.

In summary when the OpenID Provider is not known in advance, it might be possible to discover the OpenID Provider Issuer based on information provided by the user, such as an email address. The OpenID Connect discovery specification explains how to use WebFinger to discover the issuer. OpenIG can discover the issuer in this way. As a shortcut OpenIG can also use supported domains lists to find issuers already described in the OpenIG configuration.

To use this shortcut, OpenIG 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. OpenIG 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 OpenAM issuer configuration for OpenAM. OpenAM exposes a well-known endpoint for the provider configuration, but this example demonstrates use of the other fields:

```
{
    "name": "openam",
    "type": "Issuer",
    "config": {
        "authorizeEndpoint":
          "https://openam.example.com:8443/openam/oauth2/authorize",
        "registration_endpoint":
          "https://openam.example.com:8443/openam/oauth2/connect/register",
        "tokenEndpoint":
          "https://openam.example.com:8443/openam/oauth2/access_token",
        "userInfoEndpoint":
          "https://openam.example.com:8443/openam/oauth2/userinfo",
        "supportedDomains": [ "mail.example.*", "docs.example.com:8443" ]
    }
}
```

The following example shows an issuer configuration for Google:

```
{
    "name": "google",
    "type": "Issuer",
    "config": {
        "wellKnownEndpoint":
          "https://accounts.google.com/.well-known/openid-configuration",
        "supportedDomains": [ "gmail.*", "googlemail.com:8052" ]
    }
}
```

Javadoc

org.forgerock.openig.filter.oauth2.client.Issuer

ScheduledExecutorService — schedule the execution of tasks

Description

An executor service to schedule tasks for execution after a delay or for repeated execution with a fixed interval of time in between each execution. You can configure the number of threads in the executor service and how the executor service is stopped.

The `ScheduledExecutorService` is shared by all downstream components that use an executor service.

Usage

```
{
    "name": string,
    "type": "ScheduledExecutorService",
    "config": {
        "corePoolSize":  integer or expression<integer>,
        "gracefulStop":  boolean or expression<boolean>,
        "gracePeriod" :  duration string or expression<duration string>
    }
}
```

Properties

`"corePoolSize"`: ***integer or expression<integer>, optional***

The minimum number of threads to keep in the pool. If this property is an expression, the expression is evaluated as soon as the configuration is read.

The value must be an integer greater than zero.

Default: 1

`"gracefulStop"`: ***boolean or expression<boolean> , optional***

Defines how the executor service stops. If this property is an expression, the expression is evaluated as soon as the configuration is read.

If true, the executor service does the following:

- Blocks the submission of new jobs.
- Allows running jobs to continue.
- If a grace period is defined, waits for up to that maximum time for running jobs to finish before it stops.

If false, the executor service does the following:

- Blocks the submission of new jobs.
- Removes submitted jobs without running them.
- Attempts to end running jobs.
- If a grace period is defined, ignores it.

Default: true

`"gracePeriod"`: ***duration string or expression<duration string>, 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 stops anyway and prints a message.

When `gracefulStop` is `false`, the grace period is ignored.

A duration is a lapse of time expressed in English, such as `23 hours 59 minutes and 59 seconds`.

Durations are not case sensitive.

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`: microseconds
- `nanoseconds`, `nanosecond`, `nanosec`, `nanos`, `nano`, `ns`: nanoseconds

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"
    }
}
```

Javadoc

org.forgerock.openig.thread.ScheduledExecutorServiceHeaplet

TemporaryStorage — cache streamed content

Description

Allocates temporary buffers for caching streamed content during request processing. Initially uses memory; when the memory limit is exceeded, switches to a temporary file.

Usage

```
{
     "name": string,
     "type": "TemporaryStorage",
     "config": {
         "initialLength": number,
         "memoryLimit": number,
         "fileLimit": number,
         "directory": string
     }
}
```

Properties

`"initialLength"`: ***number, optional***

The initial length of memory buffer byte array. Default: 8192 (8 KiB).

`"memoryLimit"`: ***number, optional***

The length limit of the memory buffer. Exceeding this limit results in promotion from memory to file. Default: 65536 (64 KiB).

`"fileLimit"`: ***number, optional***

The length limit of the file buffer. Exceeding this limit results in a thrown exception. Default: 1048576 (1 MiB).

`"directory"`: ***string, optional***

The directory where temporary files are created. If omitted, then the system-dependent default temporary directory is used (typically `"/tmp"` on Unix systems). Default: use system-dependent default.

Javadoc

org.forgerock.openig.io.TemporaryStorage

TrustManager — configure a Java Secure Socket Extension TrustManager

Description

This represents the configuration for a Java Secure Socket Extension TrustManager, which manages the trust material (typically X.509 public key certificates) used to decide whether to accept the credentials presented by a peer. The configuration references the keystore that actually holds the trust material.

Usage

```
{
    "name": string,
    "type": "TrustManager",
    "config": {
        "keystore": KeyStore reference,
        "alg": string
    }
}
```

Properties

`"keystore"`: ***KeyStore reference, optional***

The KeyStore that references the store for public key certificates.

Provide either the name of the KeyStore object defined in the heap, or the inline KeyStore configuration object inline.

See also KeyStore(5).

`"alg"` ***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://${env['HOME']}/keystore.jks",
                "password": "${system['keypass']}"
            }
        }
    }
}
```

Javadoc

org.forgerock.openig.security.TrustManagerHeaplet

See Also

JSSE Reference Guide, KeyManager(5), KeyStore(5)

TrustAllManager — a TrustManager that blindly trusts all servers

Description

The TrustAllManager blindly trusts all server certificates presented the servers for protected applications. It can be used instead of a TrustManager(5) 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(5) instead.

Usage

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

Example

The following example configures a client handler that blindly trusts server certificates when OpenIG connects to servers over HTTPS:

```
{
    "name": "BlindTrustClientHandler",
    "type": "ClientHandler",
    "config": {
        "trustManager": {
            "type": "TrustAllManager"
        }
    }
}
```

Javadoc

org.forgerock.openig.security.TrustAllManager

UmaService — represent an UMA resource server configuration

Description

An UmaService represents a User-Managed Access (UMA) resource server. Each service is statically registered as an OAuth 2.0 client of a single UMA authorization server.

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

```
{
    "type": "UmaService",
    "config": {
        "protectionApiHandler": Handler reference,
        "authorizationServerUri": URI string,
        "clientId": expression,
        "clientSecret": expression,
        "resources": [ resource, ... ]
    }
}
```

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 details, see Handlers.

`"authorizationServerUri"`: *URI string, required*

The URI to the UMA authorization server.

`"clientId"`: *expression, required*

An expression that evaluates to the OAuth 2.0 `client_id` registered with the UMA authorization server.

`"clientSecret"`: *expression, required*

An expression that evaluates to the OAuth 2.0 `client_secret` registered with the UMA authorization server.

`"resources"`: *array of resources, required*

Resource objects matching the resources the resource owner wants to share.

Each resource object has the following form:

```
{
    "pattern": resource pattern,
    "actions": [
        {
            "scopes": [ scope string, ... ],
            "condition": boolean expression
        },
        {
            ...
        }
    ]
}
```

Each resource pattern can be seen to represent an application, or a consistent set of endpoints that share scope definitions. The actions map each request to the associated scopes. This configuration serves to set the list of scopes in the following ways:

1. When registering a resource set, OpenIG uses the list of actions to provide the aggregated, exhaustive list of all scopes that can be used.
2. When responding to an initial request for a resource, OpenIG derives the scopes for the ticket based on the scopes that apply according to the request.
3. When verifying the RPT, OpenIG checks that all required scopes are encoded in the RPT.

A description of each field follows:

`"pattern"`: ***resource pattern, required***

A pattern matching resources to be shared by the resource owner, such as `. to match any resource path, and /photos/.` to match paths starting with `/photos/`.

See also Patterns(5).

`"actions"`: ***array of action objects, optional***

A set of actions on matching resources that the resource owner can authorize.

When granting permission, the resource owner specifies the action scope. Conditions specify what the scopes mean in concrete terms. A given scope matches a requesting party operation when the corresponding condition evaluates to `true`.

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

Scope strings to identify permissions.

For example, `#read` (read access on a resource).

`"condition"`: ***boolean expression, required***

A boolean expression representing the meaning of a scope.

For example, `${request.method == 'GET'}` (true when reading a resource).

See also Expressions(5).

The REST API for Shares

The REST API for UMA shares is exposed at a registered endpoint. OpenIG 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-handler/routes/00-uma/objects/umaservice/share'
```

To access the endpoint over HTTP or HTTPS, prefix the path with the OpenIG scheme, host, and port to obtain a full URL, such as `http://localhost:8080/openig/api/system/objects/router-handler/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, OpenIG does not have a mechanism for persisting shares. When the OpenIG container stops, the shares are discarded.

A share object has the following form:

```
{
    "path": pattern,
    "pat": UMA protection API token (PAT) string,
```

```
    "id": unique identifier string,
    "resource_set_id": unique identifier string,
    "user_access_policy_uri": URI string
}
```

The fields are as follows:

`"path"`: ***pattern, required***

A pattern matching the path to protected resources, such as `/photos/.*`.

This pattern must match a pattern defined in the UmaService for this API.

See also Patterns(5).

`"pat"`: ***PAT string, required***

A PAT granted by the UMA authorization server given consent by the resource owner.

In the present implementation, OpenIG 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_set_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.

See Also

User-Managed Access (UMA) Profile of OAuth 2.0

org.forgerock.openig.uma.UmaSharingService

Expressions

Many configuration parameters support dynamic expressions.

Expressions — expression configuration parameter values

Description

Expressions are specified as configuration parameter values for a number of built-in objects. Such expressions conform to the Universal Expression Language as specified in JSR-245.

General Syntax

All expressions follow standard Universal Expression Language syntax: `${expression}`. The expression can be a simple reference to a value, a function call, or arbitrarily complex arithmetic, logical, relational and conditional operations. When supplied within a configuration parameter, an expression is always a string enclosed in quotation marks, for example: `"${request.method}"`.

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.

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.

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.

Operations

Universal Expression Language supports arbitrarily complex arithmetic, logical, relational and conditional operations. They are, in order of precedence:

- Index property value: `[]`, `.`
- Change precedence of operation: `()`
- Unary negative: `-`

- Logical operations: `not`, `!`, `empty`
- Arithmetic operations: `*`, `/`, `div`, `%`, `mod`
- Binary arithmetic operations: `+`, `-`
- Relational operations: `<`, `>`, `<=`, `>=`, `lt`, `gt`, `le`, `ge`, `==`, `!=`, `eq`, `ne`
- Logical operations: `&&`, `and`, `||`, `or`
- Conditional operations: `?`, `:`

System Properties and Environment Variables

You can use expressions to retrieve Java system properties, and to retrieve environment variables.

For system properties, `${system['property']}` yields the value of *property*, or `null` if there is no value for *property*. For example, `${system['user.home']}` yields the home directory of the user running the application server for OpenIG.

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 OpenIG.

Functions

A number of built-in functions described in Functions(5) can be called within an expression.

Syntax is `${function(parameter, ···)}`, where zero or more parameters are supplied to the function. For example, `"${toLowerCase(request.method)}"` yields the method of the request, converted to lower case. Functions can be operands for operations, and can yield parameters for other function calls.

Escaping Literal Expressions

Use the backslash `\` character as the escape character. For example, `${true}` as an expression normally evaluates to `true`. To include the string `${true}` in an expression, write `${true}`.

You can also escape literal expressions by single-quoting the initial characters. For example, `${'${'}true}` evaluates to `${true}`. To include a single backslash `\` character, write `${'\\'}`. To include a double backslash, write `${'\\\\'}`.

Embedding Expressions

Although an expression cannot be embedded as `${expression}` inside another expression, embedding system property, environment variable, and function expressions within each other is fine. Do not enclose the embedded elements in `${}`.

The following single line example embeds an `env` environment variable expression and the Java `String.concat()` method in the argument to a `read()` function:

```
"entity" : "${read(env['OPENIG_BASE'].concat('/html/defaultResponse.html'))}"
```

In the example the entity property value is set to the contents of the file `$OPENIG_BASE/html/defaultResponse.html`.

Extensions

OpenIG offers a plugin interface for extending expressions. See Key Extension Points in the *Gateway Guide*.

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.form['action'][0] != 'logout'}"

"${request.uri.host == 'wiki.example.com'}"

"${request.cookies[keyMatch(request.cookies,'^SESS.*')][0].value}"

"${toString(request.uri)}"

"${request.method == 'POST' and request.uri.path == '/wordpress/wp-login.php'}"

"${request.method != 'GET'}"

"${request.headers['cookie'][0]}"

"${request.uri.scheme == 'http'}"

"${not (response.status.code == 302 and not empty session.gotoURL)}"

"${response.headers['Set-Cookie'][0]}"

"${request.headers['host'][0]}"

"${not empty system['OPENIG_BASE'] ? system['OPENIG_BASE'] :
'/path/to'}/logs/gateway.log"
```

See Also

Contexts(5), Functions(5), Request(5), Response(5)

Functions — built-in functions to call within expressions

Description

A set of built-in functions that can be called from within expressions, which are described in Expressions(5).

array

```
array(strings...)
```

Returns an array of the strings given as argument. .Parameters

strings
: the strings to put in the array.

Returns

array
: the resulting array of containing the given strings.

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
: the object to be searched for the presence of.

value
: the value to be searched 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

The base64-encoded string to decode.

Returns

string

The base64-decoded string.

encodeBase64

```
encodeBase64(string)
```

Returns the base64-encoded string, or `null` if the string is `null`. .Parameters

string

The string to encode into Base64.

Returns

string

The base64-encoded string.

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

the parameter name or value

Returns

string

The 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

the parameter name or value

Returns

string

The 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 within a string of the first occurrence of a specified substring. .Parameters

string

the string in which to search for the specified substring.

substring

the value to search for within the string.

Returns

number

the index of the first instance of substring, or -1 if not found.

The index count starts from 1, not 0.

join

```
join(strings, separator)
```

Joins an array of strings into a single string value, with a specified separator. .Parameters

separator

the separator to place between joined elements.

strings

the array of strings to be joined.

Returns

string

the string containing the joined strings.

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

the map whose keys are to be searched.

pattern

a string containing the regular expression pattern to match.

Returns

string

the 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. .Parameters

object

the object whose length is to be determined.

Returns

number

the length of the object, or 0 if length could not be determined.

matchingGroups

```
matchingGroups(string, pattern)
```

Returns an array of matching groups for the specified regular expression pattern applied to the specified string, or `null` if no such match is found. The first element of the array is the entire match, and each subsequent element correlates to any capture group specified within the regular expression. .Parameters

string

the string to be searched.

pattern

a string containing the regular expression pattern to match.

Returns

array

an array of matching groups, or `null` if no such match is found.

matches

```
matches(string, pattern)
```

Returns `true` if the string contains a match for the specified regular expression pattern. .Parameters

string

the string to be searched.

pattern

a string containing the regular expression pattern to find.

Returns

true

if the string contains the specified regular expression pattern.

read

```
read(string)
```

Takes a file name as a `string`, and returns the content of the file as a plain string, or `null` on error (due to the file not being found, for example).

Either provide the absolute path to the file, or a path relative to the location of the Java system property `user.dir`. .Parameters

string

The name of the file to read.

Returns

string

The content of the file or `null` on error.

readProperties

```
readProperties(string)
```

Takes a Java Properties file name as a `string`, and returns the content of the file as a key/value map of properties, or `null` on error (due to the file not being found, for example).

Either provide the absolute path to the file, or a path relative to the location of the Java system property `user.dir`.

For example, to get the value of the `key` property in the properties file `/path/to/my.properties`, use `${readProperties('/path/to/my.properties')['key']}`. .Parameters

string

The name of the Java Properties file to read.

Returns

object

The key/value map of properties 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

the string to be split.

pattern

the regular expression to split substrings around.

Returns

array

the resulting array of split substrings.

toLowerCase

```
toLowerCase(string)
```

Converts all of the characters in a string to lower case. .Parameters

string

the string whose characters are to be converted.

Returns

string

the string with characters converted to lower case.

toString

```
toString(object)
```

Returns the string value of an arbitrary object. .Parameters

object
: the object whose string value is to be returned.

Returns

string
: the string value of the object.

toUpperCase

```
toUpperCase(string)
```

Converts all of the characters in a string to upper case. .Parameters

string
: the string whose characters are to be converted.

Returns

string
: the string with characters converted to upper case.

trim

```
trim(string)
```

Returns a copy of a string with leading and trailing whitespace omitted. .Parameters

string
: the string whose white space is to be omitted.

Returns

string
: the 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
: The string to be URL decoded, which may be `null`.

Returns

string

The 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

The string to be URL encoded, which may be `null`.

Returns

string

The 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

the fragment

Returns

string

The 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

The 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

the parameter name or value

Returns

string

The 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

the userInfo

Returns

string

The 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

the 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

the path element

Returns

string

The 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

the parameter name or value

Returns

string

The 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

the userInfo

Returns

string

The string resulting from URL encoding the provided userInfo as per RFC 3986.

Javadoc

Some functions are provided by org.forgerock.openig.el.Functions.

Other functions are provided by org.forgerock.http.util.Uris.

Patterns — regular expression patterns

Description

Patterns in configuration parameters and expressions use the standard Java regular expression Pattern class. For more information on regular expressions, see Oracle's tutorial on Regular Expressions.

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.

See Also

Java Pattern class

Regular Expressions tutorial

Requests, Responses, and Contexts

This part of the reference describes the OpenIG object model. The top-level objects are request, response, and contexts.

Attributes — context for arbitrary information

Description

Provides a map for arbitrary context information.

This is one of the contexts described in Contexts(5).

Properties

`"attributes"`: map

Map of arbitrary information where the keys are strings, and the values are objects.

This is never `null`.

Javadoc

org.forgerock.services.context.AttributesContext

Client — HTTP client context information

Description

Provides information about the client sending the request.

This is one of the contexts described in Contexts(5).

Properties

`"certificates"`: array

List of X.509 certificates presented by the client

If the client does not present any certificates, OpenIG returns an empty list.

This is 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 OpenIG with a non-default deployment descriptor that secures the OpenIG web application.

`"userAgent"`: **string**

The value of the User-Agent HTTP header in the request if any, otherwise `null`

Javadoc

org.forgerock.services.context.ClientContext

Contexts — HTTP request contexts

Description

The root object for request context information.

Contexts is a map of available contexts, which implement the Context interface. The contexts map's keys are strings and the values are context objects. A context holds type-safe information useful for processing requests and responses. The `contexts` map is populated dynamically when creating bindings for evaluation of expressions and scripts.

All context objects have the following properties:

`"contextName"`: **string**

Name of the context.

`"id"`: **string**

Read-only string uniquely identifying the context object.

`"rootContext"`: **boolean**

True if the context object is a RootContext (has no parent).

`"parent"`: **Context object**

Parent of this context object.

Properties

The contexts object provides access to the following contexts:

`"attributes"`: **AttributesContext object**

Arbitrary state information.

OpenIG can use this to inject arbitrary state information into the context.

See also Attributes(5).

`"client"`: **ClientContext object**

Information about the client making the request.

See also Client(5).

`"router"`: **UriRouterContext object**

Routing information associated with the request.

See also UriRouterContext(5).

`"session"`: **SessionContext object**

Session context associated with the remote client.

See also Session(5).

Javadoc

org.forgerock.services.context.Context

Request — HTTP request

Description

An HTTP request message.

Properties

`"method"`: ***string***

The method to be performed on the resource. Example: `"GET"`.

`"uri"`: ***object***

The fully-qualified URI of the resource being accessed. Example: `"http://www.example.com/resource.txt"`.

See also URI(5).

`"version"`: ***string***

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

`"headers"`: ***object***

Exposes message header fields as name-value pairs, where name is header name and value is an array of header values.

`"cookies"`: ***object***

Exposes incoming request cookies as name-value pairs, where name is cookie name and value is an array of string cookie values.

`"form"`: ***object***

Exposes query parameters and/or `application/x-www-form-urlencoded` entity as name-value pairs, where name is the field name and value is an array of string values.

`"entity"`: ***object***

The message entity body (no accessible properties).

Javadoc

org.forgerock.http.protocol.Request

Response — HTTP response

Description

An HTTP response message.

Properties

`"cause"`: ***Exception object***

The cause of an error if the status code is in the range 4xx-5xx. Possibly null.

`"status"`: ***Status object***

The response status.

For details, see Status(5).

`"version"`: ***string***

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

`"headers"`: ***object***

Exposes message header fields as name-value pairs, where name is header name and value is an array of header values.

`"entity"`: ***object***

The message entity body (no accessible properties).

Javadoc

org.forgerock.http.protocol.Response

Session — HTTP session context

Description

Provides access to the HTTP session context.

This is one of the contexts described in Contexts(5).

Properties

`"session"`: **map**

Provides access to the HTTP session, which is a map. Session attributes are name-value pairs, where both keys and value are strings.

Javadoc

org.forgerock.http.session.SessionContext

Status — HTTP response status

Description

Represents an HTTP response status. For details, see RFC 7231: HTTP/1.1 Semantics and Content.

Properties

`"code"`: ***integer***

Three-digit integer reflecting the HTTP status code.

`"family"`: ***enum***

Family Enum value representing the class of response that corresponds to the code:

`Family.INFORMATIONAL`

Status code reflects a provisional, informational response: 1xx.

`Family.SUCCESSFUL`

The server received, understood, accepted and processed the request successfully. Status

code: 2xx.

`Family.REDIRECTION`

Status code indicates that the client must take additional action to complete the request: 3xx.

`Family.CLIENT_ERROR`

Status code reflects a client error: 4xx.

`Family.SERVER_ERROR`

Status code indicates a server-side error: 5xx.

`Family.UNKNOWN`

Status code does not belong to one of the known families: 600+.

`"reasonPhrase"`: ***string***

The human-readable reason-phrase corresponding to the status code.

For details, see RFC 7231: HTTP/1.1 Semantics and Content.

`"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.

Javadoc

org.forgerock.http.protocol.Status

URI — Uniform Resource Identifier

Description

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.

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.

Javadoc

org.forgerock.http.MutableUri

Router — HTTP request routing context information

Description

Provides context information related to HTTP request routing.

This is one of the contexts described in Contexts(5).

Properties

`"matchedUri"`: string

The portion of the request URI that matched the URI template.

`"originalUri"`: URI

The original target URI for the request, as received by the web container.

The value of this field is read-only.

`"remainingUri"`: string

The portion of the request URI that is remaining to be matched.

`"uriTemplateVariables"`: map

An unmodifiable Map where the keys and values are strings. The map contains the parsed URI template variables keyed on the URI template variable name.

Javadoc

org.forgerock.http.routing.UriRouterContext

Appendix A: Release Levels and Interface Stability

This appendix includes Open Identity Platform definitions for product release levels and interface stability.

Appendix B: Release Levels and Interface Stability

This appendix includes ForgeRock definitions for product release levels and interface stability.

ForgeRock Product Release Levels

ForgeRock defines Major, Minor, and Maintenance product release levels. The release level is reflected in the version number. The release level tells you what sort of compatibility changes to expect.

Table A.1. Release Level Definitions

Release Label	Version Numbers	Characteristics
Major	Version: x[.0.0] (trailing 0s are optional)	• Bring major new features, minor features, and bug fixes • Can include changes even to Stable interfaces • Can remove previously Deprecated functionality, and in rare cases remove Evolving functionality that has not been explicitly Deprecated • Include changes present in previous Minor and Maintenance releases
Minor	Version: x.y[.0] (trailing 0s are optional)	• Bring minor features, and bug fixes • Can include backwards-compatible changes to Stable interfaces in the same Major release, and incompatible changes to Evolving interfaces • Can remove previously Deprecated functionality • Include changes present in previous Minor and Maintenance releases
Maintenance	Version: x.y.z	• Bring bug fixes • Are intended to be fully compatible with previous versions from the same Minor release

Open Identity Platform Product Interface Stability

Open Identity Platform products support many protocols, APIs, GUIs, and command-line interfaces. Some of these interfaces are standard and very stable. Others offer new functionality that is continuing to evolve.

Open Identity Platform Community acknowledges that you invest in these interfaces, and therefore must know when and how Open Identity Platform Community expects them to change. For that reason, Open Identity Platform Community defines interface stability labels and uses these definitions in Open Identity Platform products.

Interface Stability Definitions

Stability Label	Definition
Stable	This documented interface is expected to undergo backwards-compatible changes only for major releases. Changes may be announced at least one minor release before they take effect.
Evolving	This documented interface is continuing to evolve and so is expected to change, potentially in backwards-incompatible ways even in a minor release. Changes are documented at the time of product release. While new protocols and APIs are still in the process of standardization, they are Evolving. This applies for example to recent Internet-Draft implementations, and also to newly developed functionality.
Deprecated	This interface is deprecated and likely to be removed in a future release. For previously stable interfaces, the change was likely announced in a previous release. Deprecated interfaces will be removed from Open Identity Platform products.
Removed	This interface was deprecated in a previous release and has now been removed from the product.
Internal/Undocumented	Internal and undocumented interfaces can change without notice. If you depend on one of these interfaces, contact Open Identity Platform Approved Vendors to discuss your needs.