Configuration Reference

ForgeRock Identity Gateway 5

Paul Bryan
Mark Craig
Jamie Nelson
Joanne Henry

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

Abstract

Reference documentation for the ForgeRock® Identity Gateway.

Table of Contents

Preface

ForgeRock Identity Platform™ is the only offering for access management, identity management, user-managed access, directory services, and an identity gateway, designed and built as a single, unified platform.

The platform includes the following components that extend what is available in open source projects to provide fully featured, enterprise-ready software:

- ForgeRock Access Management (AM)
- ForgeRock Identity Management (IDM)
- ForgeRock Directory Services (DS)
- ForgeRock Identity Gateway (IG)

1. About This Guide

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

For API specifications, see the appropriate Javadoc.

2. Reserved Routes

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

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

To change the base for administrative routes, edit `admin.json`. For more information, see AdminHttpApplication(5).

3. 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`.

4. 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 evaluated at configuration time, when routes are loaded.

Configuration expressions can refer to the system heap properties, the built-in functions listed in Functions(5), the `${env['variable']}`, and `${system['property']}`. Because configuration expressions are evaluated before any requests are made, they cannot refer to the runtime properties, `request`, `response`, or `context`. For more information, see Expressions(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, and negative durations are not supported. The following units can be used in durations:

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

expression

See Expressions(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.

5. About ForgeRock Common REST

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

5.1. Common REST Resources

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

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

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

5.2. Common REST Verbs

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

Create

Add a new resource.

This verb maps to HTTP PUT or HTTP POST.

For details, see Section 5.6, "Create".

Read

Retrieve a single resource.

This verb maps to HTTP GET.

For details, see Section 5.7, "Read".

Update

Replace an existing resource.

This verb maps to HTTP PUT.

For details, see Section 5.8, "Update".

Delete

Remove an existing resource.

This verb maps to HTTP DELETE.

For details, see Section 5.9, "Delete".

Patch

Modify part of an existing resource.

This verb maps to HTTP PATCH.

For details, see Section 5.10, "Patch".

Action

Perform a predefined action.

This verb maps to HTTP POST.

For details, see Section 5.11, "Action".

Query

Search a collection of resources.

This verb maps to HTTP GET.

For details, see Section 5.12, "Query".

5.3. Common REST Parameters

Common REST reserved query string parameter names start with an underscore, `_`.

Reserved query string parameters include, but are not limited to, the following names:

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

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

Continue reading for details about how to use each parameter.

5.4. Common REST Extension Points

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

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

5.5. Common REST API Documentation

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

Use the following query string parameters to retrieve API descriptors:

`_api`

Serves an API descriptor that complies with the OpenAPI specification.

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

`_crestapi`

Serves a native Common REST API descriptor.

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

> **Note**
>
> Consider limiting access to API descriptors in production environments in order to avoid unnecessary traffic.
>
> To provide documentation in production environments, see Procedure 1, "To Publish OpenAPI Documentation" instead.

Procedure 1. To Publish OpenAPI Documentation

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

Use the OpenAPI-compliant descriptors to provide API reference documentation for your developers as described in the following steps:

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

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

2. Retrieve the OpenAPI-compliant descriptor.

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

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

3. If necessary, edit the descriptor.

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

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

4. Publish the descriptor using a tool such as Swagger UI.

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

5.6. Create

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

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

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

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

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

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

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

Parameters

You can use the following parameters:

`_prettyPrint=true`

Format the body of the response.

_fields=*field*[,*field*...]

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

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

5.7. Read

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

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

Parameters

You can use the following parameters:

_prettyPrint=true

Format the body of the response.

_fields=*field*[,*field*...]

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

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

_mimeType=*mime-type*

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

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

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

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

5.8. Update

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

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

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

Parameters

You can use the following parameters:

`_prettyPrint=true`

Format the body of the response.

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

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

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

5.9. Delete

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

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

Parameters

You can use the following parameters:

`_prettyPrint=true`

Format the body of the response.

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

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

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

5.10. Patch

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

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

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

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

PATCH operations apply to three types of targets:

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

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

5.10.1. Patch Operation: Add

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

If the target field is single-valued, then the value you include in the PATCH replaces the value of the target. Examples of a single-valued field include: object, string, boolean, or number.

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

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

5.10.2. Patch Operation: Copy

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

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

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

If the source field value and the target field value are configured as arrays, the result depends on whether the array has list semantics or set semantics, as described in Section 5.10.1, "Patch Operation: Add".

5.10.3. Patch Operation: Increment

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

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

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

5.10.4. Patch Operation: Move

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

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

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

To apply a move operation on an array, you need a compatible single-value, list semantic array, or set semantic array on both the source and the target. For details, see the criteria described in Section 5.10.1, "Patch Operation: Add".

5.10.5. Patch Operation: Remove

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

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

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

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

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

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

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

5.10.6. Patch Operation: Replace

The replace operation removes any existing value(s) of the targeted field, and replaces them with the provided value(s). It is essentially equivalent to a remove followed by a add operation. If the arrays are used, the criteria is based on Section 5.10.1, "Patch Operation: Add". However, indexed updates are not allowed, even when the target is an array.

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

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

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

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

Apply the following operations on that resource:

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

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

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

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

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

5.10.7. Patch Operation: Transform

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

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

5.10.8. Patch Operation Limitations

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

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

Parameters

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

`_prettyPrint=true`

Format the body of the response.

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

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

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

5.11. Action

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

The standard action indicated by `_action=create` is described in Section 5.6, "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"`.

5.12. Query

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

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

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

Parameters

You can use the following parameters:

`_queryFilter=`*`filter-expression`*

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

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

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

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

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

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

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

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

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

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

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

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

For presence, use *json-pointer pr* to match resources where the JSON pointer is present.

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"`.

5.13. HTTP Status Codes

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

200 OK

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

201 Created

The request succeeded and the resource was created.

204 No Content

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

304 Not Modified

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

400 Bad Request

The request was malformed.

401 Unauthorized

The request requires user authentication.

403 Forbidden

Access was forbidden during an operation on a resource.

404 Not Found

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

405 Method Not Allowed

The HTTP method is not allowed for the requested resource.

406 Not Acceptable

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

409 Conflict

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

410 Gone

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

412 Precondition Failed

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

415 Unsupported Media Type

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

428 Precondition Required

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

500 Internal Server Error

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

501 Not Implemented

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

503 Service Unavailable

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

6. 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.");
    }
}
```

7. Accessing Documentation Online

ForgeRock publishes comprehensive documentation online:

- The ForgeRock Knowledge Base offers a large and increasing number of up-to-date, practical articles that help you deploy and manage ForgeRock software.

- ForgeRock core 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.

Core documentation therefore follows a three-phase review process designed to eliminate errors:

- Product managers and software architects review project documentation design with respect to the readers' software lifecycle needs.

- Subject matter experts review proposed documentation changes for technical accuracy and completeness with respect to the corresponding software.
- Quality experts validate implemented documentation changes for technical accuracy, completeness in scope, and usability for the readership.

The review process helps to ensure that documentation published for a ForgeRock release is technically accurate and complete.

Fully reviewed, published core documentation is available at http://backstage.forgerock.com/. Use this documentation when working with a ForgeRock Identity Platform release.

8. Joining the ForgeRock Community

Visit the Community resource center where you can find information about each project, download trial builds, browse the resource catalog, ask and answer questions on the forums, find community events near you, and find the source code for open source software.

9. Getting Support and Contacting ForgeRock

ForgeRock provides support services, professional services, classes through ForgeRock University, and partner services to assist you in setting up and maintaining your deployments. For a general overview of these services, see https://www.forgerock.com.

ForgeRock has staff members around the globe who support our international customers and partners. For details, visit https://www.forgerock.com, or send an email to ForgeRock at info@forgerock.com.

Required Configuration

The configuration of OpenIG is split into the following parts:

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

Table of Contents

Name

AdminHttpApplication — initialize OpenIG configuration

Description

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

The configuration is loaded from a JSON-encoded configuration file, expected by default at `$HOME/.openig/config/admin.json`. If you don't provide an `admin.json`, OpenIG provides a file with the following defaults:

- The base of the administrative route is `/openig`. The product version and build information for a running instance of OpenIG are available at the `/openig/api/info` endpoint.

- OpenIG runs in development mode.

The AdminHttpApplication creates the following objects by default:

- 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).

Important

The AdminHttpApplication does not declare default configurations for objects such as the ClientHandler or ForgeRockClientHandler. If you add an `admin.json` file to your configuration, consider whether you need to add these objects to its heap.

The AdminHttpApplication 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 `admin.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.

The AdminHttpApplication creates a default TemporaryStorage object named TemporaryStorage. To change the default values, add a TemporaryStorage object named TemporaryStorage and use non-default values.

Usage

```
{
   "heap": [ configuration object, ... ],
   "mode": enumeration,
   "prefix" : string,
   "properties": JSON object,
   "temporaryStorage": TemporaryStorage reference
}
```

Properties

`mode`: *operating mode, optional*

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

Development mode

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

In development mode, by default all endpoints are open and accessible. You can create, edit, and deploy routes through OpenIG Studio, and manage routes through Common REST, without authentication or authorization.

To protect specific endpoints in development mode, configure an ApiProtectionFilter in `admin.json` and add it to the OpenIG configuration.

Production mode

After you have developed your configuration, switch to production mode to test the configuration, to run the software in pre-production or production, or to run multiple instances of the software with the same configuration.

In production mode, the `/routes` endpoint is not exposed or accessible. OpenIG Studio is effectively disabled, and you cannot manage, list, or even read routes through Common REST.

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

Default: `development`

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

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

You can omit an empty array.

`"prefix"`: *Base for the administrative route, optional*

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

Default: `openig`

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

Configuration parameters declared as property variables for use in the configuration. See also Properties(5).

Default: none

`"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.

See also TemporaryStorage(5).

Example

The following example shows an `admin.json` file configured to override the default `ApiProtectionFilter` that protects the reserved administrative route. This example is used in Procedure 11.5, "To Set Up OpenIG As a UMA Resource Server" in the *Gateway Guide*.

```
{
  "heap": [
    {
      "name": "ClientHandler",
      "type": "ClientHandler"
    },
    {
      "name": "ApiProtectionFilter",
      "type": "ScriptableFilter",
      "config": {
        "type": "application/x-groovy",
        "file": "CorsFilter.groovy"
      }
    }
  ],
  "prefix": "openig"
}
```

Javadoc

org.forgerock.openig.http.AdminHttpApplication

Name

GatewayHttpApplication — configure OpenIG

Description

The GatewayHttpApplication is the entry point for all incoming gateway requests. It is responsible for initializing a heap of objects, described in Heap Objects(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`.

If you provide a `config.json`, the OpenIG configuration is loaded from that file. If there is no file, the default configuration is loaded.

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

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

Important

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

The GatewayHttpApplication creates the following objects by default:

- 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, ... ],
  "properties": JSON object,
  "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.

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

Configuration parameters declared as property variables for use in the configuration. See also Properties(5).

Default: none

`"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

Name

Heap Objects — configure and initialize objects, with dependency injection

Description

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

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

The heap configuration is included as an object in the configuration of the GatewayHttpApplication and AdminHttpApplication. For information, see AdminHttpApplication(5) and 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 required configuration objects. To override an automatically created object, create a heap object with the same name. The following objects are automatically created:

`"ApiProtectionFilter"`

The default filter used to protect administrative APIs on reserved routes. Reserved routes are described in Section 2, "Reserved Routes".

To override this filter, declare a different filter with the same name in `admin.json`. For more information, see AdminHttpApplication(5).

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

`"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).

Javadoc

org.forgerock.openig.Heap

Name

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.

Table of Contents

Name

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

Description

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

Unlike `ChainOfFilters`, `Chain` finishes by dispatching the request to a handler. For more information, see ChainOfFilters(5).

Usage

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

Properties

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

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

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

See also Filters.

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

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

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

See also Handlers.

Example

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

Javadoc

org.forgerock.openig.filter.ChainHandlerHeaplet

Name

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),
         "temporaryStorage": string
     }
}
```

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, and negative durations are not supported. The following units can be used in durations:

- `indefinite`, `infinity`, `undefined`, `unlimited`: unlimited duration
- `zero`, `disabled`: zero-length duration
- `days`, `day`, `d`: days

- `hours`, `hour`, `h`: hours
- `minutes`, `minute`, `min`, `m`: minutes
- `seconds`, `second`, `sec`, `s`: seconds
- `milliseconds`, `millisecond`, `millisec`, `millis`, `milli`, `ms`: milliseconds
- `microseconds`, `microsecond`, `microsec`, `micros`, `micro`, `us`, `µs`: microseconds
- `nanoseconds`, `nanosecond`, `nanosec`, `nanos`, `nano`, `ns`: nanoseconds

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, and negative durations are not supported. The following units can be used in durations:

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

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).

"temporaryStorage": *string, optional*

Specifies the heap object to use for temporary buffer storage.

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

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

Name

DesKeyGenHandler — generate a DES key

Description

Generates a DES key for use with OpenAM as described in Procedure 5.5, "To Configure Password Capture in OpenAM" in the *Gateway Guide*.

Usage

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

Javadoc

org.forgerock.openig.handler.DesKeyGenHandler

Name

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

Name

Route — JSON configuration files to handle requests and their context

Description

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

- Manually into the filesystem. Many of the examples in this guide, use this method.
- Through Common REST commands. For information, see Section 12.3, " Creating and Editing Routes Through Common REST " in the *Gateway Guide*.
- For some OpenIG features, through OpenIG Studio. For information, see Section 12.4, "Creating Routes Through OpenIG Studio " in the *Gateway Guide*.

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

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

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

For examples of route configurations see Chapter 12, "*Configuring Routers and 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, and 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 at a Common REST endpoint. For more information, see "The REST API for Monitoring".

For information about API descriptors for monitoring endpoints, see Section 1.10, "Understanding OpenIG APIs With API Descriptors" in the *Gateway Guide*. For information about Common REST, see Section 5, "About ForgeRock Common REST".

Default: `false`, with percentiles `0.999`, `0.9999`, and `0.99999`.

```
{
    "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 ]`

The following examples show the formats of the `monitor` attribute:

```
{
    "monitor": true
}
```

```
{
   "monitor": {
       "enabled": false
   }
}
```

```
{
   "monitor": {
       "enabled": "${true}",
       "percentiles": [ 0.1, 0.75, 0.99, 0.999 ]
   }
}
```

`"name"`: *string, optional*

Name for the route. This parameter is used by the router to order the routes in the configuration. If the route is not named, the route ID is used.

When you add a route manually to the routes folder or add it through Common REST, the route ID is the same as the filename of the route you add. When you add a route through OpenIG Studio, you can edit the default route ID.

Default: route ID

`"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 a route has `"monitor": true`, monitoring statistics are exposed at a Common REST endpoint. OpenIG logs the paths to the HTTP endpoints when the log level is `INFO` or finer.

The monitoring REST API supports only read (HTTP GET).

OpenIG does not reset or persist monitoring statistics - when the OpenIG container stops, collected statistics are discarded.

For information about API descriptors for monitoring endpoints, see Section 1.10, "Understanding OpenIG APIs With API Descriptors" in the *Gateway Guide*. For information about Common REST, see Section 5, "About ForgeRock Common REST".

Accessing Monitoring Endpoints

To know the endpoints in a system, look in the system logs for messages like this:

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

To access a monitoring endpoint over HTTP or HTTPS, prefix the endpoint with the OpenIG scheme, host, and port to obtain a full URL, such as: `http://openig.example.com:8080/openig/api/system/objects/main-router/routes/00-monitor/monitoring`. When you access the URL, a JSON monitoring resource displays statistics for requests and responses.

A JSON resource with default percentiles has the following form, where the 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
        }
    }
}
```

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

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
```

Name

Router — Route request processing to separate routes

Description

A router is a handler that performs the following tasks:

- Defines the routes directory and loads routes into the configuration. Routes are loaded into the configuration in lexicographic order by the route name. If a route is not named, then the route ID is used instead. For more information, see Route(5).
- Depending on the scanning interval, periodically scans the routes directory and updates the OpenIG configuration when routes are added, removed, or changed. The router updates the OpenIG configuration without needing to restart OpenIG or access the route.
- Routes requests to the first route in the configuration whose condition is satisfied.

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

  If a request satisfies the condition of more than one route, it is routed to the first route in the configuration whose condition is satisfied. Even if it matches a later route in the configuration, it might never reach that route. Because the configuration orders routes lexicographically by route name, name your routes with this in mind.

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

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

Usage

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

An alternative value for type is RouterHandler.

Properties

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

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

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

Default: If no default route is set either here or in the route configurations, 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 Section 3.3, "Installing OpenIG" in the *Gateway Guide*.

The following example changes the file system location to look for routes:

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

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

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

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

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

Default: 10 seconds

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

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

Javadoc

org.forgerock.openig.handler.router.RouterHandler

Name

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:

- *`localName`* is the name of the attribute set in the session
- *`incomingName`* is the name of the attribute set in the incoming assertion

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

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

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

`"redirectURI"`: *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. Because the dot character (`.`) serves as a query separator in expressions, do not use dot characters in the field name.

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

As an example, if you set `"authnContext": "myAuthnContext"`, then 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. Because the dot character (`.`) serves as a query separator in expressions, do not use dot characters in the field name.

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

As an example, if you set `"sessionIndexMapping": "mySessionIndex"`, then 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. Because the dot character (`.`) serves as a query separator in expressions, do not use dot characters in the field name.

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

As an example, if you set `"subjectMapping": "mySubjectName"`, then 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

Name

ScriptableHandler — handle a request by using a script

Description

Handles a request by using a script.

The script must return either a Promise<Response, NeverThrowsException> 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.

The values for script arguments can be defined as expressions. The expressions are evaluated at configuration time, and cannot refer to `context` and `request`. The `context` and `request` variables can be accessed directly within the scripts.

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

All heap objects

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

`openig`

An implicit object that provides access to the environment when expressions are evaluated.

`attributes`

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

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

`context`

The processing context.

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

`contexts`

a map<string, context> object. For information, see Contexts(5).

`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.

`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 array of strings   // or "source", but not both
        "args": object,
        "clientHandler": Handler reference
    }
}
```

Properties

`"type"`: *string, required*

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

`"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 Section 3.3, "Installing OpenIG" in the *Gateway Guide*.

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

`"source"`: *string or array of strings, required if `"file"` is not used*

The script as a string or array of strings; mutually exclusive with `"file"`.

The following example shows the source of a script as an array of strings:

```
"source" : [
    "Response response = new Response(Status.OK)",
    "response.entity = 'foo'",
    "return response"
]
```

"args": ***map, optional***

Parameters passed from the configuration to the script.

The 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

Name

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

Name

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.

Table of Contents

Name

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

Name

ConditionalFilter — dispatch request to a delegate filter if a condition is met

Description

Verifies that a specified condition is met. If the condition is met, the request is dispatched to a delegate Filter. If the condition is not met, the delegate Filter is skipped.

Use `ConditionalFilter` to easily use or skip a Filter depending on whether a condition is met.

To easily use or skip a set of Filters, use a `ChainOfFilters` as the delegate Filter and define a set of Filters. For information, see ChainOfFilters(5).

Usage

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

Properties

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

Expression that evaluates to `true` or `false`, to determine whether a request should be dispatched to the delegate Filter.

See also Expressions.

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

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

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

See also Filters.

Example

The following example tests whether a request contains a header named `username`. If it does, the request is dispatched to the delegate Filter. Otherwise, the delegate Filter is skipped and the request is passed directly to the `ClientHandler`.

```
{
  "type": "Chain",
  "config": {
    "filters": [{
      "type": "ConditionalFilter",
      "config": {
        "condition": "${not empty request.headers['username']}",
        "delegate": "myDelegateFilter"
      }
    }],
    "handler": "ClientHandler"
  }
}
```

Javadoc

org.forgerock.http.filter.ConditionalFilter

Name

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

Name

ChainOfFilters — dispatch the request to ordered list of filters

Description

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

A `ChainOfFilters` can be used as a filter inside a chain.

Unlike `Chain`, `ChainOfFilters` does not finish by dispatching the request to a handler. For more information, see Chain(5).

Usage

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

Properties

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

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

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

See also Filters.

Example

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

Javadoc

org.forgerock.openig.filter.ChainFilterHeaplet

Name

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

Name

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

Name

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

Name

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 Section 4.2, "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

Name

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

Name

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

Name

LocationHeaderFilter — rewrites Location headers

Description

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

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

In the following example, OpenIG listens on `https://openig.example.com:443` and the application it protects listens on `http://app.example.com:8081`. The filter rewrites redirects that would normally take the user to locations under `http://app.example.com:8081` to go instead to locations under `https://openig.example.com:443`.

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

Javadoc

org.forgerock.openig.filter.LocationHeaderFilter

Name

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.

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

***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": uriexpression,
    "defaultLogoutGoto": uriexpression,
    "requireHttps": boolean,
    "requireLogin": boolean,
    "issuerRepository": Issuer repository reference
  }
}
```

Properties

`"clientEndpoint"`: *expression, required*

Base URI for the filter.

For example, if you set `"clientEndpoint": "/openid"`, the service URIs for this filter on your OpenIG server are `/openid/login`, `/openid/logout`, and `/openid/callback`. These endpoints are implicitly reserved. Attempts to access them directly can cause undefined errors.

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.

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

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

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

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

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

The route in "Example Configuration For Multiple Identity Providers" includes a login handler that allows the user to choose from two client registrations, `openam` and `google`. For an example of a login handler where no client registrations are defined, see Procedure 9.6, "Preparing OpenIG for Discovery and Dynamic Registration" in the *Gateway Guide*.

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, and negative durations are not supported. The following units can be used in durations:

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

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*

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

- If the authorization process completes successfully, the OAuth2ClientFilter injects the authorization state data into the target.

  In the following example, a downstream StaticRequestFilter retrieves the username and password from the target to log the user in to the sample application.

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

For information about setting up this example, see Section 9.7, "Adapting the Configuration to Authenticate Automatically to the Sample Application" in the *Gateway Guide*.

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

Default: `${attributes.openid}`

See also Expressions(5).

`"defaultLoginGoto"`: *expression, optional*

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

Default: return an empty page.

See also Expressions(5).

`"defaultLogoutGoto"`: *expression, optional*

The URI to which a request is redirected if the user accesses the `clientEndpoint/logout` endpoint without providing a goto URL.

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.

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

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

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

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

See also IssuerRepository(5).

Example Configuration For Multiple Identity Providers

The following example builds on the OpenAM and OpenIG configuration in Example 9.1, "Route for OpenIG As a Relying Party" in the *Gateway Guide*.

The client registration for the OpenAM provider is now declared in the heap. A second client registration defines Google as an alternative identity provider. The NascarPage helps the user to choose an identity provider.

When a request is made to `/home/id_token/login`, the OAuth2ClientFilter directs the user to the NascarPage to choose an identity provider, and then handles negotiation for authorization with the chosen provider.

If the authorization process completes successfully, the filter injects the authorization state data into `attributes.openid`. The ClientHandler then passes the request to the home page of the sample application.

If a request is made to `/home/id_token/logout` without specifying a goto URL, the request is redirected to www.forgerock.com by the `defaultLogoutGoto` property.

```
{
  "heap": [
    {
      "name": "openam",
      "type": "ClientRegistration",
      "config": {
        "clientId": "oidc_client",
        "clientSecret": "password",
        "issuer": {
          "name": "Issuer",
          "type": "Issuer",
          "config": {
            "wellKnownEndpoint": "http://openam.example.com:8088/openam/oauth2/.well-known/openid-
configuration"
          }
        },
        "scopes": [
          "openid",
          "profile",
          "email"
```

```
       ],
       "tokenEndpointAuthMethod": "client_secret_basic"
     }
   },
   {
     "name": "google",
     "type": "ClientRegistration",
     "config": {
       "clientId": "myClientId",
       "clientSecret": "myClientSecret",
       "issuer": {
         "name": "accounts.google.com",
         "type": "Issuer",
         "config": {
           "wellKnownEndpoint": "https://accounts.google.com/.well-known/openid-configuration"
         }
       },
       "scopes": [
         "openid",
         "profile"
       ]
     }
   },
   {
     "name": "NascarPage",
     "type": "StaticResponseHandler",
     "config": {
       "status": 200,
       "entity": "<html><body>
                  <p><a href='/home/id_token/login?registration=openam&goto=
                  ${urlEncodeQueryParameterNameOrValue('http://openig.example.com:8080/home')}'>OpenAM
Login</a></p>
                  <p><a href='/home/id_token/login?registration=google&goto=
                  ${urlEncodeQueryParameterNameOrValue('http://openig.example.com:8080/home')}'>Google
Login</a></p>
                  </body></html>"
     }
   }
 ],
 "handler": {
   "type": "Chain",
   "config": {
     "filters": [
       {
         "type": "OAuth2ClientFilter",
         "config": {
           "target" : "${attributes.openid}",
           "clientEndpoint": "/home/id_token",
           "loginHandler" : "NascarPage",
           "registrations": [ "openam", "google" ],
           "defaultLogoutGoto" : "http://www.forgerock.com",
           "requireHttps": false,
           "requireLogin": true,
           "failureHandler": {
             "type": "StaticResponseHandler",
             "config": {
               "comment": "Trivial failure handler for debugging only",
               "status": 500,
               "reason": "Error",
                "entity": "${attributes.openid}"
              }
            }
          }
        }
      ],
      "handler": "ClientHandler"
    }
  },
  "condition": "${matches(request.uri.path, '^/home/id_token')}"
}
```

Use the following hints to set up Google credentials for this example:

1. Go to https://console.cloud.google.com/apis/credentials.

2. Create credentials for an OAuth 2.0 client ID with the following options:

   - Application type: `Web application`

   - Authorized redirect URI: `http://openig.example.com:8080/home/id_token/callback`

3. In the OpenIG route, replace `myClientId` and `myClientSecret` with the returned credentials.

For information about configuring 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

Name

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.

> **Tip**
>
> To facilitate auditing, configure this handler with a `ForgeRockClientHandler`, which sends a ForgeRock Common Audit transaction ID when it communicates with protected applications.
>
> Alternatively, configure this handler as a chain containing a `TransactionIdOutboundFilter`, as in the following configuration:
>
> ```
> providerHandler : {
>   "type" : "Chain",
>   "config" : {
>     "handler": "MySecureClientHandler",
>     "filters": [ "TransactionIdOutboundFilter" ]
>   }
> }
> ```

Default: ForgeRockClientHandler

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, and negative durations are not supported. The following units can be used in durations:

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

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

Name

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 Chapter 13, "*Configuration Templates*" in the *Gateway Guide*, and the Javadoc for the PasswordReplayFilter class.

Javadoc

org.forgerock.openig.filter.PasswordReplayFilterHeaplet

Name

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.

After a policy decision, OpenIG continues to process requests as follows:

- If the request is allowed, processing continues.
- If the request is denied with advice, OpenIG checks whether it can respond to the advice. If OpenIG can respond to the advice, it processes the advice.
- If the request is denied without advice, or if OpenIG cannot respond to the advice, OpenIG forwards the request to a `failureHandler` declared in the `PolicyEnforcementFilter`. If there is no `failureHandler`, OpenIG returns a 403 Forbidden.
- If an error occurs during the process, OpenIG returns 500 Internal Server Error.

Attributes and advices returned by the policy decision are stored in the `attributes` and `advices` properties of the `${contexts.policyDecision}` context. They are available to the PolicyEnforcementFilter's downstream filters and handlers.

The subject can be specified by an SSO token, an OpenID Connect id_token that is encoded as JWT, or the decoded claims from a JWT.

> **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,
        "cache": object,
        "amHandler": handler reference,
        "realm": string,
        "ssoTokenHeader": string,
        "application": string,
        "environment": map or expression,
        "executor": executor
        "failureHandler": handler reference
   }
}
```

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.

Tip

To facilitate auditing, configure this handler with a `ForgeRockClientHandler`, which sends a ForgeRock Common Audit transaction ID when it communicates with protected applications.

Alternatively, configure this handler as a chain containing a `TransactionIdOutboundFilter`, as in the following configuration:

```
amHandler : {
  "type" : "Chain",
  "config" : {
    "handler": "MySecureClientHandler",
    "filters": [ "TransactionIdOutboundFilter" ]
  }
}
```

Default: `ForgeRockClientHandler`

See also Handlers, ClientHandler(5).

`"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 policy set to use when requesting policy decisions.

Default: `Default Policy Set`

`cache`: *object, optional*

The caching of OpenAM policy decisions. By default, policy decisions are not cached.

> **Note**
>
> Policy decisions that contain advices are never cached.

The following code example caches OpenAM policy decisions without advices for these times:

- One hour, when the policy decision doesn't provide a `ttl` value.
- The duration specified by the `ttl`, when `ttl` is shorter than one day.
- One day, when `ttl` is longer than one day.

```
"cache": {
  "enabled": true,
  "defaultTimeout": "1 hour",
  "maxTimeout": "1 day"
}
```

`enabled`: *boolean, optional*

Enable or disable caching of policy decisions.

Default: `false`

`defaultTimeout`: *duration, optional*

The default duration for which to cache OpenAM policy decisions.

If an OpenAM policy decision provides a valid `ttl` value to specify the time until which the policy decision remains valid, OpenIG uses that value or the `maxTimeout`.

Default: `1 minute`

`maxTimeout`: *duration, optional*

The maximum duration for which to cache OpenAM policy decisions.

If the `ttl` value provided by the OpenAM policy decision is after the current time plus the `maxTimeout`, OpenIG uses the `maxTimeout`.

The duration cannot be `zero` or `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, and negative durations are not supported. The following units can be used in durations:

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

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

A list of strings to forward to OpenAM with the request for a policy decision, that represent the environment (or context) of the request. Forward any HTTP header or any value that the OpenAM policy definition can use.

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

If this property is a map, the structure must have the format `Map<String, List<String>>`. In the following example, the `PolicyEnforcementFilter` forwards standard and non-standard request headers, an ID token, and the IP address of the subject making the request:

```
"environment": {
  "H-Via": "${request.headers['Via']}",
  "H-X-Forwarded-For": "${request.headers['X-Forwarded-For']}",
  "H-myHeader": "${request.headers['myHeader']}",
  "id_token": [
    "${attributes.openid.id_token}"
  ],
    "IP": [
      "${contexts.client.remoteAddress}"
  ]
}
```

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

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

`"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).

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

Handler to treat requests when they are denied by the policy decision.

In the following example, the `failureHandler` is a chain with a scriptable filter. If there are some advices with the policy decision, the script recovers the advices for processing. Otherwise, it passes the request to the `StaticResponseHandler` to display a message.

```
"failureHandler":{
  "type": "Chain",
  "config": {
    "filters": [
      {
        "type": "ScriptableFilter",
        "config": {
          "type": "application/x-groovy",
          "source": [
            "if (contexts.policyDecision.advices['MyCustomAdvice'] != null) {",
            "  return handleCustomAdvice(context, request)",
            "} else {",
            "  return next.handle(context, request)",
            "}"
          ]
        }
      }
    ],
    "handler" : {
      "type": "StaticResponseHandler",
      "config": {
        "status": 403,
        "entity": "Restricted area. You do not have sufficient privileges."
      }
    }
  }
}
```

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

See also Handlers.

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

Example

In the following example, the PolicyEnforcementFilter retrieves a cookie value from the SsoTokenContext to identify the subject making the request, and then calls to OpenAM for a policy decision.

If the request is denied by the policy decision, the failure handler displays a message along with any advices from the policy decision. By default, the policy decision is not cached—each request requires a new policy decision.

For information about how to set up and test this example, see Section 6.2, "Enforcing Policy Decisions From OpenAM" in the *Gateway Guide*

```
{
  "name": "04-pep",
  "baseURI": "http://app.example.com:8081",
  "monitor": false,
  "condition": "${matches(request.uri.path, '^/home/pep')}",
  "handler": {
    "type": "Chain",
    "config": {
      "filters": [
        {
          "type": "SingleSignOnFilter",
          "name": "SingleSignOn",
          "config": {
            "openamUrl": "http://openam.example.com:8088/openam",
            "realm": "/",
            "cookieName": "iPlanetDirectoryPro"
          }
        },
        {
          "type": "PolicyEnforcementFilter",
          "name": "PEPFilter",
          "config": {
            "openamUrl": "http://openam.example.com:8088/openam",
            "pepUsername": "policyAdmin",
            "pepPassword": "password",
            "pepRealm": "/",
            "realm": "/",
            "application": "PEP policy set",
            "ssoTokenSubject": "${contexts.ssoToken.value}"
          }
        }
      ],
      "handler": "ClientHandler"
    }
  }
}
```

Javadoc

org.forgerock.openig.openam.PolicyEnforcementFilter

org.forgerock.openig.openam.PolicyDecisionContext

See Also

OpenAM Authorization Guide

Name

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, NeverThrowsException> 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.

The values for script arguments can be defined as expressions. The expressions are evaluated at configuration time, and cannot refer to `context` and `request`. The `context` and `request` variables can be accessed directly within the scripts.

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

All heap objects

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

`openig`

An implicit object that provides access to the environment when expressions are evaluated.

`attributes`

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

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

`context`

The processing context.

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

`contexts`

a map<string, context> object. For information, see Contexts(5).

`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.

`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 array of strings,   // or "source", but not both.
        "args": object,
        "clientHandler": Handler reference
    }
}
```

Properties

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

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

`"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 Section 3.3, "Installing OpenIG" in the *Gateway Guide*.

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

`"source"`: *string or array of strings, required if `"file"` is not used*

The script as a string or array of strings; mutually exclusive with `"file"`.

The following example shows the source of a script as an array of strings:

```
"source" : [
    "Response response = new Response(Status.OK)",
    "response.entity = 'foo'",
    "return response"
]
```

`"args"`: *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 Section 14.6.2, "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

Name

SingleSignOnFilter — require OpenAM authentication before processing a request

Description

This filter tests for the presence and validity of an SSO token in the cookie header of a request:

- If an SSO token is present, the SingleSignOnFilter calls OpenAM to validate the SSO token. If the SSO token is valid, the request continues along the chain. The SingleSignOnFilter creates an `SsoTokenContext` to store the token value and additional information in endpoints such as `${contexts.ssoToken.value}` and `${contexts.ssoToken.info}`.

- If the SSO token is not present, or if it is empty or invalid, the filter redirects the user agent to the OpenAM login page for authentication.

> **Note**
>
> Each time the SingleSignOnFilter validates an SSO token on OpenAM, the idle timeout for the OpenAM stateful session is reset. As a result, validating a session token triggers a write operation to the OpenAM Core Token Service token store.
>
> For information about token validation in OpenAM, see Using OpenAM Sessions in the *OpenAM Authentication and Single Sign-On Guide*.

Usage

```
{
  "name": string,
  "type": "SingleSignOnFilter",
    "config": {
      "amHandler": handler,
      "cookieName": string,
      "defaultLogoutLandingPage": uriExpression,
      "logoutEndpoint": pattern,
      "openamUrl": uriExpression,
      "realm": string
  }
}
```

Properties

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

The handler to use when communicating with OpenAM to validate the token in the incoming request.

In production, use a ClientHandler that can make HTTPS connections to OpenAM.

Tip

To facilitate auditing, configure this handler with a `ForgeRockClientHandler`, which sends a ForgeRock Common Audit transaction ID when it communicates with protected applications.

Alternatively, configure this handler as a chain containing a `TransactionIdOutboundFilter`, as in the following configuration:

```
amHandler : {
  "type" : "Chain",
  "config" : {
    "handler": "MySecureClientHandler",
    "filters": [ "TransactionIdOutboundFilter" ]
  }
}
```

Default: `ForgeRockClientHandler`

See also Handlers, ClientHandler(5).

`"cookieName"` ***string, optional***

The name of the cookie to use for authentication.

Default: `iPlanetDirectoryPro`

`"defaultLogoutLandingPage"`: *URI expression, optional*

The URI to which a request is redirected after the user logs out of OpenAM.

This parameter is effective only when `logoutEndpoint` is specified.

Default: None. Processing continues.

`"logoutEndpoint"` ***pattern, optional***

A string denoting a regular expression pattern for a URL. When a request matches the pattern, OpenIG performs the logout process and the OpenAM authentication token for the end user is revoked.

If a `defaultLogoutLandingPage` is specified, the request is redirected to that page. Otherwise, the request continues to be processed.

Default: Logout is not managed by this filter.

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

The URL of an OpenAM service to use for authentication. For example, use a URL such as `https://openam.example.com:8443/openam/`.

See also Expressions(5).

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

The OpenAM realm to use for authentication.

Default: `/` (Top Level Realm)

Example

In the following example, a SingleSignOnFilter tests whether a request has a valid iPlanetDirectoryPro cookie:

- If the request does not have a valid iPlanetDirectoryPro cookie, the filter redirects the user-agent to OpenAM for authentication.
- If the request already has a valid iPlanetDirectoryPro cookie, or after authenticating with OpenAM to get a valid iPlanetDirectoryPro cookie, the request is forwarded to the PolicyEnforcement filter.

```
{
  "name": "04-pep",
  "baseURI": "http://app.example.com:8081",
  "monitor": false,
  "condition": "${matches(request.uri.path, '^/home/pep')}",
  "handler": {
    "type": "Chain",
    "config": {
      "filters": [
        {
          "type": "SingleSignOnFilter",
          "name": "SingleSignOn",
          "config": {
            "openamUrl": "http://openam.example.com:8088/openam",
            "realm": "/",
            "cookieName": "iPlanetDirectoryPro"
          }
        },
        {
          "type": "PolicyEnforcementFilter",
          "name": "PEPFilter",
          "config": {
            "openamUrl": "http://openam.example.com:8088/openam",
            "pepUsername": "policyAdmin",
            "pepPassword": "password",
            "pepRealm": "/",
            "realm": "/",
            "application": "PEP policy set",
            "ssoTokenSubject": "${contexts.ssoToken.value}"
          }
        }
      ],
      "handler": "ClientHandler"
    }
  }
}
```

For information about how to set up this example, see Chapter 6, "*Enforcing Policy Decisions and Supporting Session Upgrade*" in the *Gateway Guide*. For information about the PolicyEnforcementFilter, see PolicyEnforcementFilter(5).

Javadoc

org.forgerock.openig.openam.SingleSignOnFilter

org.forgerock.openig.openam.SsoTokenContext

Name

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

Name

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`.

In the following example, the field values are hardcoded in the form:

```
"form": {
  "username": [
    "demo"
  ],
  "password": [
    "changeit"
  ]
```

In the following example, the field values are evaluated to the first value of `username` and `password` provided in the session:

```
"form": {
  "username": [
    "${session.username[0]}"
  ],
  "password": [
    "${session.password[0]}"
  ]
```

`"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

In the following example, OpenIG replaces the browser's original HTTP GET request with an HTTP POST login request containing credentials to authenticate to the sample application. For information about how to set up and test this example, see Section 2.6, "Try the Installation" in the *Gateway Guide*.

```
{
  "handler": {
    "type": "Chain",
    "config": {
      "filters": [
        {
          "type": "StaticRequestFilter",
          "config": {
            "method": "POST",
            "uri": "http://app.example.com:8081/login",
            "form": {
              "username": [
                "demo"
              ],
              "password": [
                "changeit"
              ]
            }
          }
        }
      ],
      "handler": "ClientHandler"
    }
  },
  "condition": "${matches(request.uri.path, '^/static')}"
}
```

Javadoc

org.forgerock.openig.filter.StaticRequestFilter

Name

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

Name

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 REST Security Token Service (STS) APIs to transform an OpenID Connect ID Token (`id_token`) into a SAML 2.0 assertion. The subject confirmation method is Bearer, as described in *Profiles for the OASIS Security Assertion Markup Language (SAML) V2.0*.

The TokenTransformationFilter makes the result of the token transformation available to downstream handlers in the `issuedToken` property of the `${contexts.sts}` context.

The TokenTransformationFilter configuration references a REST STS instance that must be set up in OpenAM before the TokenTransformationFilter can be used. The REST STS instance exposes a preconfigured transformation under a specific REST endpoint. For information about setting up a REST STS instance, see the OpenAM documentation.

Errors that occur during the token transformation cause a error response to be returned to the client and an error message to be logged for the OpenIG administrator.

Usage

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

Properties

`"openamUri"`: *URI expression, required*

The base URL to an OpenAM service, such as `https://openam.example.com:8443/openam/`. This expression is evaluated when the route is initialized.

Authentication and REST STS requests are made to this service.

`"realm"`: *expression, 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. This expression is evaluated when the route is initialized.

Default: `/` (Top Level Realm)

`"username"`: *expression, required*

The username for authenticating OpenIG as an OpenAM REST STS client. This expression is evaluated when the route is initialized.

See also Expressions(5).

`"password"`: *expression, required*

The password for authenticating OpenIG as an OpenAM REST STS client. This expression is evaluated when the route is initialized.

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).

`"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.

> **Tip**
>
> To facilitate auditing, configure this handler with a `ForgeRockClientHandler`, which sends a ForgeRock Common Audit transaction ID when it communicates with protected applications.
>
> Alternatively, configure this handler as a chain containing a `TransactionIdOutboundFilter`, as in the following configuration:

```
amHandler : {
  "type" : "Chain",
  "config" : {
    "handler": "MySecureClientHandler",
    "filters": [ "TransactionIdOutboundFilter" ]
  }
}
```

Default: `ForgeRockClientHandler`

See also Handlers, ClientHandler(5).

`"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 Chapter 10, "*Transforming OpenID Connect ID Tokens Into SAML Assertions*" in the *Gateway Guide*.

Javadoc

org.forgerock.openig.openam.TokenTransformationFilter

org.forgerock.openig.openam.StsContext

Name

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 additional heap objects that let you extend what another object can do. For example, a *CaptureDecorator* extends the capability of filters and handlers to log requests and responses.

1. Predefined Decorators

OpenIG defines the following decorators: `audit`, `baseURI`, `capture`, and `timer`. You can use these decorators without configuring them explicitly.

2. Guidelines for Naming Decorators

To prevent unwanted behavior, avoid naming decorators as follows:

- For 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`, avoid decorators named `logSink` and `temporaryStorage`.
- In general, avoid decorators named `comment` or `comments`.

Use names that do not clash with field names for the decorated objects. The best way to avoid a clash 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`.

3. Local, Route, and Global Decorators

OpenIG applies decorations in this order:

1. Local decoration - decorators declared on an object
2. globalDecoration - decorators declared on a parent route are applied before decorators declared in the current route.
3. Route decoration: decorators declared on a route handler

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.

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"
}
```

3.1. Decorating Individual Objects

To decorate individual objects, add the decorator's name value as a top-level field of the object, next to `type` and `config`.

3.2. Decorating the Handler for a Route

To decorate the handler for a route, add the decorator as a top-level field of the route. The following route includes a timer 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",
    "timer": true
}
```

3.3. Decorating All Objects in a Route's Heap

To decorate all applicable objects defined in 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.

In the following example, the route has audit and capture decorations that apply to the Chain, HeaderFilter, and StaticResponseHandler. The decorations apply to all objects in this route's heap:

```
{
    "globalDecorators": {
        "timer": true,
        "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')}"
}
```

3.4. Decorating Individual Uses of a Named Filter or Handler

When a named filter or handler is configured in `config.json` or in the heap, it can be used many times in the configuration. To decorate each use of the filter or handler individually, use a delegate filter or handler, and decorate the delegate. In this way you can decorate the filter or handler differently each time you use it in the configuration.

```
{
    "filter or handler": {
        "type": "Delegate",
        "config": {
            "delegate": [object reference]
        },
        [decorator reference, ...]
    }
}
```

A delegate can be used to decorate a filter, a handler, or any other object type.

In the following example, the policy enforcement filter configures an `amHandler` to request policy decisions from OpenAM. This handler delegates the task to `ForgeRockClientHandler` and adds a decorator to capture all requests and responses passing through the handler:

```
{
    "type": "PolicyEnforcementFilter",
    "config": {
        ...,
        "amHandler": {
            "type": "Delegate",
            "config": {
                "delegate": "ForgeRockClientHandler"
            },
            "capture": "all"
        }
    }
}
```

You can use the same `ForgeRockClientHandler` in another part of the configuration, in a different route for example, without adding a capture decorator. Requests and responses that pass through that use of the handler are not captured.

Table of Contents

Name

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

Name

CaptureDecorator — capture request and response messages

Description

Captures request and response messages for further analysis.

Decorator Usage

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

Captured information is written to SLF4J logs, and named in this format:

```
<className>.<decoratorName>.<decoratedObjectName>
```

If the decorated object is not named, the object path is used in the log.

The decorator configuration has these properties:

`"captureEntity"`: *boolean, optional*

Whether the message entity should be captured. The message entity is the body of the HTTP message, which can be a JSON document, XML, HTML, image, or other information.

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

Default: false

`"captureContext"`: *boolean, optional*

Whether the context should be captured as JSON. The context chain is used when processing the request inside OpenIG in the filters and handlers.

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><body><p>Hello, World!</p></body></html>"
                }
            }
        }
    },
    "capture": [
        "filtered_request",
        "filtered_response"
    ]
}
```

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

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

Javadoc

org.forgerock.openig.decoration.capture.CaptureDecorator

Name

TimerDecorator — record times to process filters and handlers

Description

Records time in milliseconds to process filters and handlers. The following times are recorded:

- For handlers, the total time to process the request and response.
- For filters,
  - the total time to process the request and response,
  - the time to process the request and response inside the filter,
  - and the time to process the request and response in downstream filters and handlers.

Recorded information is written to SLF4J logs, and named in this format:

```
<className>.<decoratorName>.<decoratedObjectName>
```

If the decorated object is not named, the object path is used in the log.

Decorator Usage

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

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

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

Unit of time used in the decorator output. The unit of time can be any unit allowed in the `duration` field. For information, see duration.

Default: `ms`

Decorated Object Usage

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

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

The unique name of the object, 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 For a ClientHandler

To record the elapsed time for operations traversing a ClientHandler, use a configuration like this:

```
{
    "handler": {
        "type": "ClientHandler"
    },
    "timer": true
}
```

This configuration could produce the following log message:

```
11:23:16:870 | INFO  | I/O dispatcher 2 | o.f.o.d.t.T.t.top-level-handler | Elapsed time: 5 ms
```

To Record Times for Individual Filters and Handlers

To record times spent for individual filters and handlers in a route, individually decorate each filter and handler in the route. Alternatively, configure `globalDecorators` in the top-level field of the route.

To Deactivate the Timers

To deactivate timer, set the value to `false`:

```
{
    "timer": false
}
```

Javadoc

org.forgerock.openig.decoration.timer.TimerDecorator

Audit Framework

OpenIG uses the ForgeRock common audit framework to log system boundary events using an implementation that is common across the ForgeRock platform.

Table of Contents

Name

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.openig.handler.router.AuditServiceObject

Name

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, and negative durations are not supported. The following units can be used in durations:

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

`"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, and negative durations are not supported. The following units can be used in durations:

- `indefinite`, `infinity`, `undefined`, `unlimited`: unlimited duration
- `zero`, `disabled`: zero-length duration
- `days`, `day`, `d`: days
- `hours`, `hour`, `h`: hours
- `minutes`, `minute`, `min`, `m`: minutes
- `seconds`, `second`, `sec`, `s`: seconds
- `milliseconds`, `millisecond`, `millisec`, `millis`, `milli`, `ms`: milliseconds

- `microseconds`, `microsecond`, `microsec`, `micros`, `micro`, `us`, `µs`: microseconds
- `nanoseconds`, `nanosecond`, `nanosec`, `nanos`, `nano`, `ns`: nanoseconds

"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, and negative durations are not supported. The following units can be used in durations:

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

"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, and negative durations are not supported. The following units can be used in durations:

- `indefinite`, `infinity`, `undefined`, `unlimited`: unlimited duration
- `zero`, `disabled`: zero-length duration
- `days`, `day`, `d`: days

- `hours`, `hour`, `h`: hours
- `minutes`, `minute`, `min`, `m`: minutes
- `seconds`, `second`, `sec`, `s`: seconds
- `milliseconds`, `millisecond`, `millisec`, `millis`, `milli`, `ms`: milliseconds
- `microseconds`, `microsecond`, `microsec`, `micros`, `micro`, `us`, `µs`: microseconds
- `nanoseconds`, `nanosecond`, `nanosec`, `nanos`, `nano`, `ns`: nanoseconds

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 Procedure 15.2, "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

Name

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, and negative durations are not supported. The following units can be used in durations:

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

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

`"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 Procedure 15.3, " 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"
    ]
  }
}
```

Name

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, and negative durations are not supported. The following units can be used in durations:

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

`"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

Name

JmsAuditEventHandler — record messages between a JMS message broker and clients

Description

The Java Message Service (JMS) is a Java API for sending asynchronous messages between clients. It wraps audit events in JMS messages and publishes them in a JMS broker, which then delivers the messages to the appropriate destination.

The JMS API architecture includes a *JMS provider* and *JMS clients*, and supports the *publish/ subscribe* messaging pattern. For more information, see *Basic JMS API Concepts*.

The JMS audit event handler does not support queries. To support queries, also enable a second handler that supports queries.

The ForgeRock JMS audit event handler supports JMS communication, based on the following components:

- JMS message broker, to provide clients with connectivity, message storage, and message delivery functionality.
- JMS messages, which follow the format in "Example".
- Destinations, maintained by a message broker. A destination can be a JMS topic, using publish/ subscribe to take the ForgeRock JSON for an audit event, wrap it into a JMS TextMessage, and send it to the broker.
- JMS clients, to produce and/or receive JMS messages.

Depending on the configuration, some or all of these components are included in JMS audit log messages.

Important

The example in this section is based on *Apache ActiveMQ*, but you can choose a different JMS message broker.

Make sure that the `.jar` files required by the JMS message broker are available in the OpenIG web container.

Declare the configuration in an audit service, as described in AuditService(5). To define an `JmsAuditEventHandler`, create the following object in the heap:

```
{
  "name": string,
  "type": "AuditService",
  "config": {},
}
```

Usage

```
{
  "type": "AuditService",
  "config": {
    "config": {},
    "event-handlers": [
    {
      "class": "org.forgerock.audit.handlers.jms.JmsAuditEventHandler",
      "config": {
        "name": string,
        "topics": array of strings,
        "deliveryMode": string,
        "sessionMode": string,
        "jndi": {
          "contextProperties": map,
          "topicName": string,
          "connectionFactoryName": string
        }
      }
    }]
  }
}
```

The values in this configuration object can use configuration expressions, as described in "Configuration and Runtime Expressions".

Configuration

For a full list of properties in the `"config"` object, see Configuration Properties for the JMS Audit Event Handler in the *OpenIDM Integrator's Guide*.

`"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" ]`.

`"deliveryMode"`: *string, required*

Delivery mode for messages from a JMS provider. Set to `PERSISTENT` or `NON_PERSISTENT`.

`"sessionMode"`: *string, required*

Acknowledgement mode in sessions without transactions. Set to `AUTO`, `CLIENT`, or `DUPS_OK`.

"contextProperties": ***map, optional***

Settings with which to populate the initial context.

The following properties are required when ActiveMQ is used as the message broker:

- `java.naming.factory.initial`

  For example, `"org.apache.activemq.jndi.ActiveMQInitialContextFactory"`.

  To substitute a different JNDI message broker, change the JNDI context properties.

- `java.naming.provider.url`

  For example, `"tcp://127.0.0.1:61616"`.

  To configure the message broker on a remote system, substitute the associated IP address.

  To set up SSL, set up keystores and truststores, and change the value of the `java.naming.provider.url` to:

  ```
  ssl://127.0.0.1:61617?daemon=true&socket.enabledCipherSuites=SSL_RSA_WITH_RC4_128_SHA
  ,SSL_DH_anon_WITH_3DES_EDE_CBC_SHA
  ```

- `topic.audit`

  For example, `"audit"`.

To use the JMS resources provided by your application server, leave this field empty. The values for `topicName` and `connectionFactoryName` are then JNDI names that depend on the configuration of your application server.

"topicName": ***string, required***

JNDI lookup name for the JMS topic.

For ActiveMQ, this property must be consistent with the value of `topic.audit` in `contextProperties`.

"connectionFactoryName": ***string, required***

JNDI lookup name for the JMS connection factory.

Example

In the following example, a JMS audit event handler delivers audit events in batches. The handler is configured to use the ActiveMQ JNDI message broker, on port 61616. For an example of setting up and testing this configuration, see Procedure 15.4, "To Record Audit Events With a JMS Audit Event Handler" in the *Gateway Guide*.

```
{
  "MyCapture" : "all",
```

```
  "auditService" : {
    "config" : {
      "event-handlers" : [
        {
          "class" : "org.forgerock.audit.handlers.jms.JmsAuditEventHandler",
          "config" : {
            "name" : "jms",
            "topics" : [ "access" ],
            "deliveryMode" : "NON_PERSISTENT",
            "sessionMode" : "AUTO",
            "jndi" : {
              "contextProperties" : {
                "java.naming.factory.initial" : "org.apache.activemq.jndi.ActiveMQInitialContextFactory",
                "java.naming.provider.url" : "tcp://openam.example.com:61616",
                "topic.audit" : "audit"
              },
              "topicName" : "audit",
              "connectionFactoryName" : "ConnectionFactory"
            }
          }
        }
      ],
      "config" : { }
    },
    "type" : "AuditService"
  },
  "handler" : {
    "type" : "StaticResponseHandler",
    "config" : {
      "status" : 200,
      "headers" : {
        "content-type" : [ "text/plain" ]
      },
      "reason" : "found",
      "entity" : "Message from audited route"
    }
  },
  "monitor" : true,
  "condition" : "${request.uri.path == '/activemq_event_handler'}"
}
```

The following code shows an example message returned by the JMS audit event handler.

```
{
  "auditTopic": "access",
  "event": {
    "eventName": "OPENIG-HTTP-ACCESS",
    "timestamp": "2016-11-28T14:39:30.004Z",
    "transactionId": "882918f9-f7c3-47ee-9f87-5e3cfcfb98be-37",
    "server": {
      "ip": "0:0:0:0:0:0:0:1",
      "port": 8080
    },
    "client": {
      "ip": "0:0:0:0:0:0:0:1",
      "port": 56095
    },
    "http": {
      "request": {
        "secure": false,
        "method": "GET",
        "path": "http://openig.example.com:8080/activemq_event_handler",
        "queryParameters": {},
        "headers": {
          "accept": ["*/*"],
          "accept-encoding": ["gzip, deflate"],
          "Connection": ["keep-alive"],
          "host": ["openig.example.com:8080"],
          "user-agent": ["python-requests/2.9.1"]
        },
        "cookies": {}
      },
      "response": {
        "headers": {
          "Content-Length": ["26"],
          "Content-Type": ["text/plain"]
        }
      }
    },
    "response": {
      "status": "SUCCESSFUL",
      "statusCode": "200",
      "elapsedTime": 73,
      "elapsedTimeUnits": "MILLISECONDS"
    },
    "_id": "882918f9-f7c3-47ee-9f87-5e3cfcfb98be-38"
  }
}
```

Name

JsonAuditEventHandler — log events as JSON objects to a set of JSON files

Description

The JSON audit event handler logs events as JSON objects to a set of JSON files. This is the preferred file-based audit event handler.

Declare the configuration in an audit service, as described in AuditService(5). To define an `JsonAuditEventHandler`, create the following object in the heap:

```
{
  "name": string,
  "type": "AuditService",
  "config": {},
}
```

Usage

```
{
  "type": "AuditService",
  "config": {
    "config": {},
    "event-handlers": [
    {
      "class": "org.forgerock.audit.handlers.json.JsonAuditEventHandler",
      "config": {
        "name": string,
        "logDirectory" : expression,
        "topics": array of strings
      }
    }]
  }
}
```

Configuration

For a full list of properties in the `config` object, see Configuration Properties for the JSON Audit Event Handler in the *OpenIDM Integrator's Guide*. The `config` object includes 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" ]`.

Examples

Example 1. Configuration for JSON Audit Event Handler

The following example configures a JSON audit event handler to log audit events for access:

```
{
  "handler": "ForgeRockClientHandler",
  "baseURI": "http://app.example.com:8081",
  "condition": "${matches(request.uri.path, '^/home/json-audit')}",
  "auditService": {
    "type": "AuditService",
    "config": {
      "config": {},
      "event-handlers": [{
        "class": "org.forgerock.audit.handlers.json.JsonAuditEventHandler",
        "config": {
          "name": "json",
          "logDirectory": "/tmp/logs",
          "topics": [
            "access"
          ]
        }
      }]
    }
  }
}
```

Example 2. Log File

The following excerpt of `access.audit.json` file shows the log message format produced by the previous route:

```
{
  "eventName": "OPENIG-HTTP-ACCESS",
  "timestamp": "2016-11-08T15:39:59.128Z",
  "transactionId": "a386a21c-0ceb-4c6b-af77-167bd71f0161-1",
  "server": {
    "ip": "0:0:0:0:0:0:0:1",
    "port": 8080
  },
  "client": {
    "ip": "0:0:0:0:0:0:0:1",
    "port": 34066
  },
  "http": {
    "request": {
      "secure": false,
      "method": "GET",
      "path": "http://openig.example.com:8080/home/json-audit",
      "queryParameters": {},
      "headers": {
        "accept": ["text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8"],
        "accept-encoding": ["gzip, deflate"],
        "Accept-Language": ["en-US;q=1"],
        "cache-control": ["max-age=0"],
        "Connection": ["keep-alive"],
        "host": ["openig.example.com:8080"],
        "upgrade-insecure-requests": ["1"],
        "user-agent": ["Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_6)
                       AppleWebKit/602.2.14 (KHTML, like Gecko) Version/10.0.1
                       Safari/602.2.14"]
      },
      "cookies": {
        "i18next": "en"
      }
    },
    "response": {
      "status": "SUCCESSFUL",
      "statusCode": "200",
      "elapsedTime": 104,
      "elapsedTimeUnits": "MILLISECONDS"
    },
    "_id": "a386a21c-0ceb-4c6b-af77-167bd71f0161-2"
  }
}
```

Name

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

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.

Table of Contents

Name

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*.

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/home/throttle-scriptable HTTP/1.1

    . . .

    HTTP/1.1 429 Too Many Requests
    Retry-After: 10
```

Usage

```
{
    "name": string,
    "type": "ThrottlingFilter",
    "config": {
        "strategy": string,
        "requestGroupingPolicy": expression,
        "throttlingRatePolicy": reference or inline declaration,  //Use either "throttlingRatePolicy"
        "rate": {                                                 //or "rate", but not both.
            "numberOfRequests": integer,
            "duration": duration string
        },
        "cleaningInterval": duration string,
        "executor": executor
    }
}
```

Properties

"strategy": *string, optional*

The name of the throttling strategy. Use one of the following values:

- `bursty`: Implements a throttling rate that allows some unevenness or bursts, based on the token bucket algorithm.

  The throttling rate can be exceeded momentarily - 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.

- `constant`: Implements a constant throttling rate, without allowing bursts. The throttling rate can never be higher than the defined limit.

Default: `bursty`

`"requestGroupingPolicy"`: ***expression, optional***

An expression to identify the partition to use for the request. In many cases the partition identifies an individual client that sends requests, but it can also identify a group that sends requests. The expression can evaluate to the client IP address or user ID, or an OpenID Connect subject/issuer.

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, and negative durations are not supported. The following units can be used in durations:

- `indefinite`, `infinity`, `undefined`, `unlimited`: unlimited duration
- `zero`, `disabled`: zero-length duration

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

"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 Section 16.1, " Configuring a Simple Throttling Filter " in the *Gateway Guide*.

```
{
  "name": "00-throttle-simple",
  "baseURI": "http://app.example.com:8081",
  "monitor" : false,
  "condition": "${matches(request.uri.path, '^/home/throttle-simple')}",
  "handler": {
    "type": "Chain",
    "config": {
      "filters": [
        {
          "type": "ThrottlingFilter",
          "name": "Throttling",
          "config": {
            "rate": {
              "numberOfRequests": 6,
              "duration": "10 s"
            }
          }
        }
      ],
      "handler": "ClientHandler"
    }
  }
}
```

Javadoc

org.forgerock.openig.filter.throttling.ThrottlingFilterHeaplet

Name

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, and negative durations are not supported. The following units can be used in durations:

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

Example of a Mapped Throttling Policy

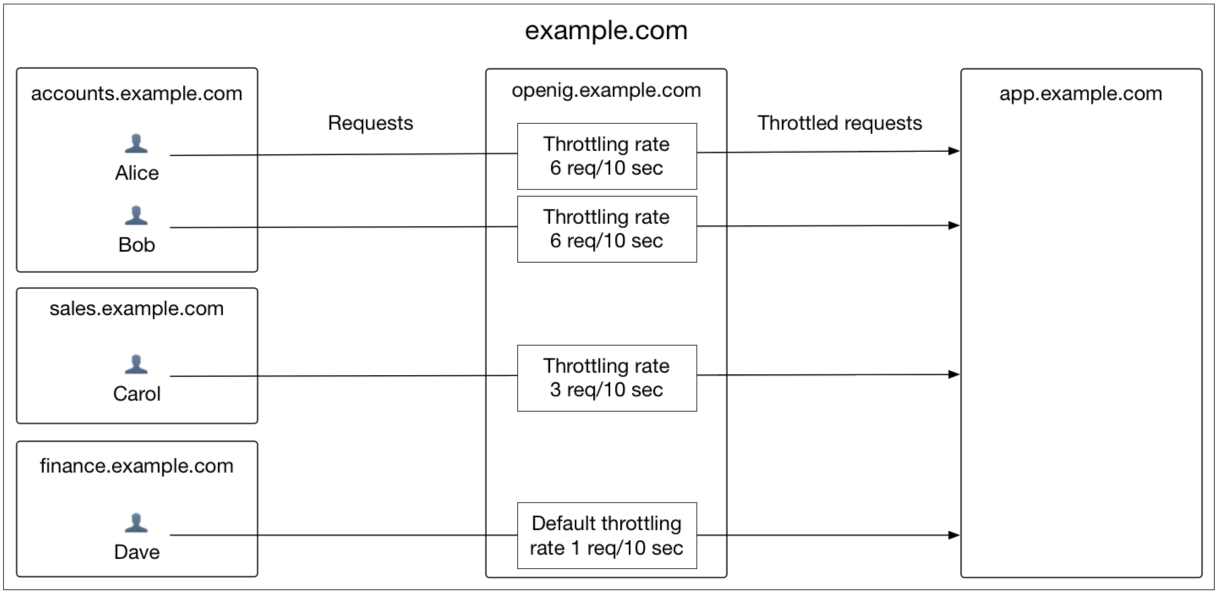
In the following example, requests from users in the accounts and sales departments of `example.com` are mapped to different throttling rates. Requests from other departments use the default throttling rate. For information about how to set up and test this example, see Section 16.2, " Configuring a 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.

Figure 1. Mapped Throttling Policy



```
{
  "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, '^/home/throttle-mapped')}"
}
```

Javadoc

org.forgerock.openig.filter.throttling.MappedThrottlingPolicyHeaplet

Name

ScriptableThrottlingPolicy — script to map throttling rates

Description

Uses a script to look up throttling rates to apply to groups of requests.

The values for script arguments can be defined as expressions. The expressions are evaluated at configuration time, and cannot refer to `context` and `request`. The `context` and `request` variables can be accessed directly within the scripts.

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.

The values for script arguments can be defined as expressions. The expressions are evaluated at configuration time, and cannot refer to `context` and `request`. The `context` and `request` variables can be accessed directly within the scripts.

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

All heap objects

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

`openig`

An implicit object that provides access to the environment when expressions are evaluated.

`attributes`

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

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

`context`

The processing context.

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

`contexts`

a map<string, context> object. For information, see Contexts(5).

`request`

The HTTP request.

`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

```
{
   "type": "ThrottlingFilter",
   "config": {
       "requestGroupingPolicy": expression,
       "throttlingRatePolicy": {
           "type": "ScriptableThrottlingPolicy",
           "config": {
               "type": string,
               "file": string,                         // Use either "file"
               "source": string or array of strings   // 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 Section 3.3, "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 or array of strings, required if "file" is not used*

The script as a string or array of strings; mutually exclusive with `"file"`.

For an example of `"source"` as an array of strings, see "Example of a Scriptable Throttling Policy".

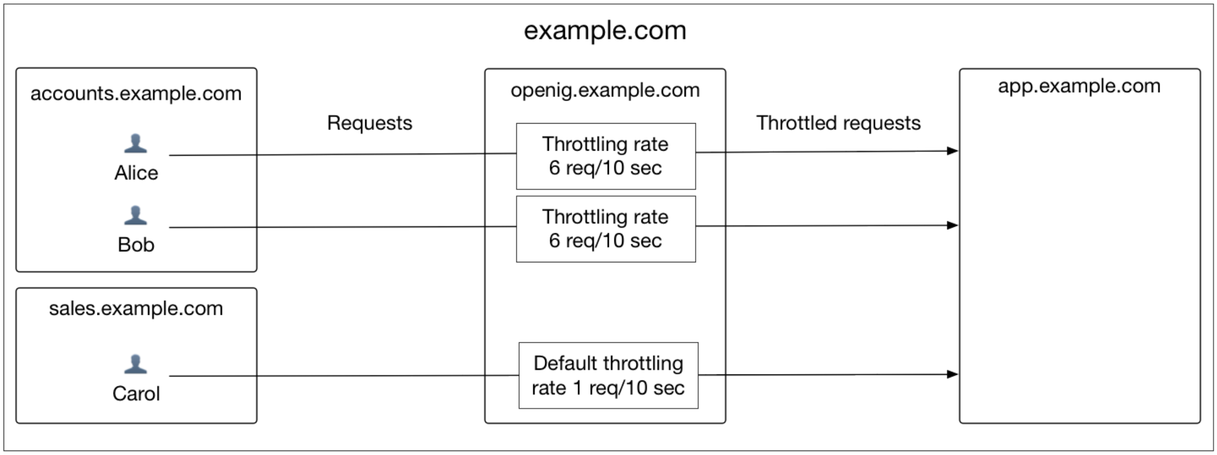
Example of a Scriptable Throttling Policy

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 Section 16.3, " 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.

Figure 2. Scriptable Throttling Policy



```
{
  "handler": {
    "type": "Chain",
    "config": {
      "filters": [
        {
          "type": "ThrottlingFilter",
          "config": {
            "requestGroupingPolicy": "${request.headers['UserId'][0]}",
            "throttlingRatePolicy": {
              "type": "DefaultRateThrottlingPolicy",
              "config": {
                "delegateThrottlingRatePolicy": {
                  "type": "ScriptableThrottlingPolicy",
                  "config": {
                    "type": "application/x-groovy",
                    "source": [
                      "if (request.headers['X-Forwarded-For'].values[0]  == 'accounts.example.com') {",
                      "  return new ThrottlingRate(6, '10 seconds')",
                      "} else {",
                      "  return null",
                      "}"
                    ]
                  }
                },
                "defaultRate": {
                  "numberOfRequests": 1,
                  "duration": "10 seconds"
                }
              }
            }
          }
        }
      ],
      "handler": "ClientHandler"
    }
  },
  "condition": "${matches(request.uri.path, '^/home/throttle-scriptable')}"
}
```

The groovy script maps a throttling rate for the accounts department of `example.com`. Other requests receive the default throttling rate.

Javadoc

org.forgerock.openig.filter.throttling.ScriptableThrottlingPolicy.Heaplet

Name

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, and negative durations are not supported. The following units can be used in durations:

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

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

Table of Contents

Name

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, ...],
    "tokenEndpointAuthMethod": enumeration
  }
}
```

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).

`"tokenEndpointAuthMethod"`: ***enumeration, optional***

Method used by clients to authenticate to the OAuth 2.0 authorization server or OpenID Provider when using the provider's OAuth 2.0 token endpoint. The following methods are allowed:

- `client_secret_basic`: 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=...
```

- `client_secret_post`: Client credentials are sent in HTTP POST form data. The credentials are included in the request body, as in the following example:

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

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

Some providers accept both authentication methods. Where a provider strictly enforces how the client must authenticate, such as recent versions of OpenAM, align the configuration with that of the provider.

If the configured method is not supported by the provider, then according to RFC 6749 *The OAuth 2.0 Authorization Framework*, section 5.2 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"
}
```

If the configured method is invalid, an `IllegalArgumentException` is thrown.

Default: `client_secret_basic`

For information about client authentication methods, see OpenID Client Authentication.

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 way:

```
{
    "name": "registration",
    "type": "ClientRegistration",
    "config": {
        "clientId": "**********",
        "clientSecret": "**********",
        "issuer": {
            "type": "Issuer",
            "config": {
                "wellKnownEndpoint": "http://openam.example.com:8088/openam/oauth2/.well-known/openid-
configuration"
            }
        },
        "scopes": [
          "openid",
          "profile",
          "email"
        ]
    }
}
```

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

Name

JwtSession — store sessions in encrypted JWT cookies

Description

A JwtSession object holds settings for storing session information in JSON Web Tokens (JWT) that are encrypted and signed, and then placed in a cookie.

A JWT session is built or rebuilt when there is a write operation in the session (for example, when a user successfully logs in with SAML, assertions are stored in the session). The JWT contains a JSON representation of the session information. The supporting cookie can be configured as a session cookie or persistent cookie. By default it is a session cookie.

The JWT session information contains the session attributes (serialized as JSON), and a marker that OpenIG uses to control the session timeout. When OpenIG receives a JWT cookie, it checks that the current date and time are before the session timeout.

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.

Important

The security of your system depends on how well your keys and keystores are secured.

Usage

```
{
    "name": string,
    "type": "JwtSession",
    "config": {
        "keystore": KeyStore reference,
        "alias": string,
        "password": configuration expression,
        "cookieName": string,
        "cookieDomain": string,
        "sessionTimeout": duration,
        "persistentCookie": boolean,
        "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 encrypt the JWT.

Provide either the name of the KeyStore object defined in the heap, or the inline KeyStore configuration object inline.

Default: When no keystore is specified, 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.

Configuration expressions can refer to the system heap properties, the built-in functions listed in Functions(5), the `${env['variable']}`, and `${system['property']}`. Because configuration expressions are evaluated before any requests are made, they cannot refer to the runtime properties, `request`, `response`, or `context`. For more information, see Expressions(5).

`"cookieName"` ***string, optional***

The name of the JWT cookie stored on the user-agent.

Default: `openig-jwt-session`

`"cookieDomain"` ***string, optional***

The name of the domain from which the JWT cookie can be accessed.

When the domain is specified, a JWT cookie can be accessed from different hosts in that domain.

Default: The domain is not specified. The JWT cookie can be accessed only from the host where the cookie was created.

`"sessionTimeout"` ***duration, optional***

The duration for which a JWT session is valid. If the supporting cookie is persistent, this property also defines the expiry of the cookie.

The value must be above zero. The maximum value is 3650 days (approximately 10 years). If you set a longer duration, OpenIG truncates the duration to 3650 days.

Default: 30 minutes

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

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

`"persistentCookie"` ***boolean, optional***

Whether or not the supporting cookie is persistent:

- `true`: the supporting cookie is a persistent cookie. Persistent cookies are reemitted by the user-agent until their expiration date or until they are deleted.

- `false`: the supporting cookie is a session cookie. OpenIG does not specify an expiry date for session cookies. The user-agent is responsible for deleting them when it considers that the session is finished (for example, when the browser is closed).

Default: `false`

`"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 cookie named `OpenIG`, which can be accessed from the domain `.example.com`. 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",
        "cookieDomain": ".example.com"
    }
}
```

Javadoc

org.forgerock.openig.jwt.JwtSessionManager

Name

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)

Name

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)

Name

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,
    "issuerRepository": Issuer repository 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).

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

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

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

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

See also IssuerRepository(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

Name

IssuerRepository — Store discovered and built OAuth2 issuers in a repository

Description

A repository to store OAuth2 issuers that are discovered or built from the configuration.

It is not normally necessary to change this object. Change it only for the following tasks:

- To isolate different repositories in the same route.
- To view the interactions of the well-known endpoint, for example, if the `issuerHandler` is delegating to another handler.

Usage

```
{
   "name": string,
   "type": "IssuerRepository",
   "config": {
     "issuerHandler": Handler reference
   }
}
```

Properties

The object properties are as follows:

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

The default handler to fetch OAuth2 issuer configurations from the well-known endpoint.

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

Default: ForgeRockClientHandler

Javadoc

org.forgerock.openig.filter.oauth2.client.IssuerRepository

Name

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, and negative durations are not supported. The following units can be used in durations:

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

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

Name

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*

Initial size of the memory buffer.

Default: 8192 (8 KB). Maximum: The value of `"memoryLimit"`.

`"memoryLimit"`: *number, optional*

Maximum size of the memory buffer. When the memory buffer is full, the content is transferred to a temporary file.

Default: 65536 (64 KB). Maximum: 2147483647 (2 GB).

`"fileLimit"`: *number, optional*

Maximum size of the temporary file. If the downloaded file is bigger than this value, an OverflowException is thrown.

Default: 1073741824 (1 GB). Maximum: 2147483647 (2 GB).

`"directory"`: *string, optional*

The directory where temporary files are created.

Default: The value of the system property `java.io.tmpdir`, typically `/tmp` on Unix systems, or `/var/tmp` on Linux.

Javadoc

org.forgerock.openig.io.TemporaryStorage

Name

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)

Name

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

Name

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/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/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 Section 5, "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.

For information about API descriptors for the UMA share endpoint, see Section 1.10, "Understanding OpenIG APIs With API Descriptors" in the *Gateway Guide*. For information about Common REST, see Section 5, "About ForgeRock Common REST".

A share object has the following form:

```
{
    "path": pattern,
    "pat": UMA protection API token (PAT) string,
    "id": unique identifier string,
    "resource_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.

Table of Contents

Name

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}"`.

Configuration and Runtime Expressions

Expressions are evaluated at configuration time (when routes are loaded), or at runtime (when OpenIG is running).

When expressions are evaluated, they access the current environment through the implicit object `openig`. The object has the following properties:

- `baseDirectory`, the path to the base location for OpenIG files. The default location is `$HOME/.openig` (`%appdata%\OpenIG`).

- `configDirectory`, the path to the OpenIG configuration files. The default location is `$HOME/.openig/config` (`%appdata%\OpenIG\config`).

- `tempDirectory`, the path to the OpenIG temporary files. The default location is `$HOME/.openig/tmp` (`%appdata%\OpenIG\tmp`).

For information about how to change the default values, see Section 3.4, "Changing the Default Location of the Configuration Folders" in the *Gateway Guide*.

configuration expression

Expression evaluated at configuration time, when routes are loaded.

Configuration expressions can refer to the system heap properties, the built-in functions listed in Functions(5), the `${env['`*`variable`*`']}`, and `${system['`*`property`*`']}`. Because configuration expressions are evaluated before any requests are made, they cannot refer to the runtime properties, `request`, `response`, or `context`. For more information, see Expressions(5).

runtime expression

Expression evaluated at runtime, for each request and response.

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

- `request`, a org.forgerock.http.protocol.Request object. For information, see Request(5).
- `context`, a org.forgerock.services.context.Context object.
- `contexts`, a map<string, context> object. For information, see Contexts(5).
- `response`, a org.forgerock.http.protocol.Response object, available only when the expression is intended to be evaluated on the response flow. For information, see Response(5).

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(openig.baseDirectory.path.concat('/html/defaultResponse.html'))}"
```

In the example the entity property value is set to the contents of the file `$HOME/.openig/html/defaultResponse.html`.

Extensions

OpenIG offers a plugin interface for extending expressions. See Section 14.6.1, "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['my-variable'] ? system['my-variable'] : '/path/to'}/logs/gateway.log"
```

See Also

Contexts(5), Functions(5), Request(5), Response(5)

Name

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.

boolean

```
boolean(string)
```

Returns a Boolean with a value represented by the specified string.

The returned Boolean represents a true value if the string argument is not `null` and is equal to the string `"true"`, ignoring case.

Parameters

string

the string containing the boolean representation.

Returns

Boolean

the Boolean value represented by the string.

contains

```
contains(object, value)
```

Returns `true` if the object contains the specified value. If the object is a string, a substring is searched for the value. If the object is a collection or array, its elements are searched for the value.

Parameters

object

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.

fileToUrl

```
fileToUrl(file)
```

Converts a `java.io.File` into a string representation for the URL of the file or directory.

Parameters

file

The file or directory for which to build the URL.

For example, `${fileToUrl(openig.configDirectory)}/myProperties.json`.

Returns

file

The string representation for the URL of the file or directory, or `null` if the file or directory is `null`.

For example, `file:///home/gcostanza/.openig/config/myProperties.json`.

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.

integer

```
integer(string)
```

Transforms the string parameter into an integer. If the parameter is not a valid number in radix 10, returns null.

Parameters

string

the string containing the integer representation.

Returns

integer

the integer value represented by the string.

integerWithRadix

```
integer(string, radix)
```

Uses the radix as the base for the string, and transforms the string into a base-10 integer. For example:

- `("20", 8)`: transforms `20` in base `8`, and returns `16`
- `("11", 16)` transforms `11` in base `16`, and returns `17`

If either parameter is not a valid number, returns null.

Parameters

string

a string containing the integer representation, and an integer containing the radix representation.

Returns

integer

the integer value in base-10

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.

pathToUrl

```
pathToUrl(path)
```

Converts the given path into the string representation of its URL.

Parameters

path

The path of a file or directory as a string.

For example, `${pathToUrl(system['java.io.tmpdir'])}`.

Returns

string

The string representation for the URL of the path, or `null` if the path is `null`.

For example, `file:///var/tmp`.

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.

Name

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

Properties

Configuration parameters can be declared as properties in the OpenIG configuration or in an external JSON file.

Table of Contents

Name

Properties — declare configuration parameters as property variables.

Description

Configuration parameters, such as host names, port numbers, and directories, can be declared as property variables in the OpenIG configuration or in an external JSON file. The variables can then be used in expressions in routes and in `config.json` to set the value of configuration parameters.

Properties can be inherited across the router, so a property defined in `config.json` can be used in any of the routes in the configuration.

Storing the configuration centrally and using variables for parameters that can be different for each installation makes it easier to deploy OpenIG in different environments without changing a single line in your route configuration.

Usage

Simple Property Configured Inline

```
{
    "properties": {
        "<variable name>": "valid JSON value"
    }
}
```

Group Property Configured Inline

```
{
    "properties": {
        "<group name>": {
            ["<variable name>": "valid JSON value", ... ]
        }
    }
}
```

Properties Configured in an External File

```
{
    "properties": {
        "$location": expression
    }
}
```

Properties

"<variable name>": ***string***

The name of a variable to use in the OpenIG configuration. The variable can be used in expressions in routes or in `config.json` to assign the value of a configuration parameter.

The value assigned to the variable can be any valid JSON value: string, number, boolean, array, object, or null.

- In the following example from `config.json`, the URL of an application is declared as a property variable named `appLocation`. The variable is then used by the `baseURI` parameter of the handler, and can be used again in other routes in the configuration.

```
{
    "properties": {
        "appLocation": "http://app.example.com:8081"
    },
    "handler": {
        "type": "Router",
        "baseURI": "${appLocation}",
        "capture": "all"
    }
}
```

- The following example adds the property variable `ports` to define an array of port numbers used by the configuration. The `ports` variable is referenced in the `appLocation` variable, and is resolved at runtime with the value in the ports array:

```
{
    "properties" : {
        "ports": [8080, 8081, 8088],
        "appLocation": "http://app.example.com:${ports[1]}"
    },
    "handler": {
        "type": "Router",
        "baseURI": "${appLocation}",
        "capture": "all"
    }
}
```

"<group name>": ***string, required***

The name of a group of variables to use in the OpenIG configuration. The group name and variable name are combined using dot notation in an expression.

In the following example from `config.json`, the property group `directories` contains two variables that define the location of files:

```
{
    "properties" : {
        "directories": {
            "config": "${openig.configDirectory.path}",
            "auditlog": "/tmp/logs"
        }
    }
}
```

The group name and variable name are combined using dot notation in the following example to define the directory where the audit log is stored:

```
{
    "type": "AuditService",
    "config": {
        "event-handlers": [
            {
            "class": "org.forgerock.audit.handlers.csv.CsvAuditEventHandler",
            "config": {
                "name": "csv",
                "logDirectory": "${directories.auditlog}",
                . . .
```

`"$location"`: *expression, required*

The location and name of a file where the property variables are configured.

In the following example, the location of the file that contains the property variables is defined as an expression:

```
{
    "properties" : {
        "$location": "${fileToUrl(openig.configDirectory)}/myProperties.json"
    }
}
```

In the following example, the location of the file that contains the property variables is defined as a string:

```
{
    "properties" : {
        "$location": "file:/Users/user-id/.openig/config/myProperties.json"
    }
}
```

The file `myProperties.json` contains the base URL of an OpenAM service and the port number of an application.

```
{
    "openamLocation": "http://openam.example.com:8088/openam/",
    "appPortNumber": 8081
}
```

Requests, Responses, and Contexts

This part of the reference describes the OpenIG object model. The top-level objects are request, response, and contexts.

Table of Contents

Name

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

Name

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

Name

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

Name

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.

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

Javadoc

org.forgerock.http.protocol.Request

org.forgerock.http.protocol.Entity

Name

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.

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

Javadoc

org.forgerock.http.protocol.Response

org.forgerock.http.protocol.Entity

Name

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. The session is a map with attributes that are name-value pairs of the format `Map<String, Object>`.

For web container sessions, any object can be stored in the session. For JwtSession, only JSON compatible types (such as primitive JSON structures, lists, arrays, and maps) can be stored in the session.

By default, sessions are web container sessions.

Javadoc

org.forgerock.services.context.SessionContext

Name

Status — HTTP response status

Description

Represents an HTTP response status. For details, see *RFC 7231: HTTP/1.1 Semantics and Content*, Section 6.1. Overview of Status Codes.

Properties

`"code"`: ***integer***

Three-digit integer reflecting the HTTP status code.

`"family"`: ***enum***

Family Enum value representing the class of response that corresponds to the code:

`Family.INFORMATIONAL`

Status code reflects a provisional, informational response: 1xx.

`Family.SUCCESSFUL`

The server received, understood, accepted and processed the request successfully. Status code: 2xx.

`Family.REDIRECTION`

Status code indicates that the client must take additional action to complete the request: 3xx.

`Family.CLIENT_ERROR`

Status code reflects a client error: 4xx.

`Family.SERVER_ERROR`

Status code indicates a server-side error: 5xx.

`Family.UNKNOWN`

Status code does not belong to one of the known families: 600+.

`"reasonPhrase"`: ***string***

The human-readable reason-phrase corresponding to the status code.

For details, see *RFC 7231: HTTP/1.1 Semantics and Content*, Section 6.1. Overview of Status Codes.

`"isClientError"`: ***boolean***

True if Family.CLIENT_ERROR.

`"isInformational"`: ***boolean***

True if Family.INFORMATIONAL.

`"isRedirection"`: ***boolean***

True if Family.REDIRECTION.

`"isServerError"`: ***boolean***

True if Family.SERVER_ERROR.

`"isSuccessful"`: ***boolean***

True if Family.SUCCESSFUL.

Javadoc

org.forgerock.http.protocol.Status

Name

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

Name

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.services.context.UriRouterContext

Appendix A. Release Levels and Interface Stability

This appendix includes ForgeRock definitions for product release levels and interface stability.

A.1. 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

Release Label	Version Numbers	Characteristics
		• 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

A.2. ForgeRock Product Interface Stability

ForgeRock 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.

ForgeRock acknowledges that you invest in these interfaces, and therefore must know when and how ForgeRock expects them to change. For that reason, ForgeRock defines interface stability labels and uses these definitions in ForgeRock products.

Table A.2. 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 ForgeRock 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 ForgeRock support or email info@forgerock.com to discuss your needs.

Index