Reference

OpenIG 3.1

Paul Bryan
Mark Craig
Jamie Nelson

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

Abstract

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

Table of Contents

Preface

This reference covers OpenIG configuration.

1. Who Should Use this Reference

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

For API specifications, see the appropriate Javadoc.

2. 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" or "com.example.myDecorator".

3. Field Value Conventions

OpenIG configuration uses JSON notation.

This reference uses the following terms when referring to the values that configuration object fields take.

array

JSON array.

boolean

Either `true` or `false`.

configuration expression

Expression for which no exchange is available.

A configuration expression is independent of the exchange, so do not use expressions that reference exchange properties. You can, however, use `${env['variable']}`, `${system['property']}`, and all the built-in functions.

duration

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

Durations are not case sensitive.

Negative durations are not supported.

The following units can be used in durations.

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

expression

See expression.

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.

4. 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.");
    }
}
```

5. 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.

  While many articles are visible to community members, ForgeRock customers have access to much more, including advanced information for customers using ForgeRock software in a mission-critical capacity.

- ForgeRock product documentation, such as this document, aims to be technically accurate and complete with respect to the software documented. It is visible to everyone and covers all product features and examples of how to use them.

6. Using the ForgeRock.org Site

The ForgeRock.org site has links to source code for ForgeRock open source software, as well as links to the ForgeRock forums and technical blogs.

If you are a *ForgeRock customer*, raise a support ticket instead of using the forums. ForgeRock support professionals will get in touch to help you.

Required Configuration

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

Table of Contents

Name

Gateway Servlet — entry point for all incoming requests

Description

The gateway servlet is the entry point for all incoming requests. It is responsible for initializing a heap of objects, and dispatching all requests to a configured handler. The configuration of the gateway servlet is loaded from a JSON-encoded configuration file, expected by default at `$HOME/.openig/config/config.json`.

The gateway servlet creates the following objects by default.

- An AuditDecorator that you can use to trigger notification for audit events. The default AuditDecorator is named "audit".
- A CaptureDecorator that you can use to capture requests and response messages, though not the entity body or the exchange. The default CaptureDecorator is named "capture".
- A TimerDecorator that you can use to record time spent within Filters and Handlers. The default TimerDecorator is named "timer".

The gateway servlet 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`. In you do that, however, stored session information must fit the constraints for storage in a JWT and in a cookie, as described in the reference documentation for JwtSession. If no such object is found, `exchange.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, ... ],
  "baseURI": string,
  "logSink":  LogSink reference,
  "temporaryStorage": TemporaryStorage reference
}
```

Properties

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

Dispatch all requests to this handler.

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

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

The heap object configuration.

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

`"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 the request URI untouched.

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

Send log messages to this LogSink.

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

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

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

Cache content during processing based on this TemporaryStorage configuration.

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

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

Javadoc

org.forgerock.openig.servlet.GatewayServlet

Name

Heap Objects — configure and initialize objects, with dependency injection

Description

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

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

When a heap is first created, it is automatically populated with some objects, without required configuration. An automatically created object can be overridden by creating a heap object with the same name.

"LogSink"

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

Default: a ConsoleLogSink object with default values.

"TemporaryStorage"

The default object to use for managing temporary buffers.

Default: a TemporaryStorage object with default values.

Implicit Properties

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

"logSink": *string*

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

Default: "LogSink".

"temporaryStorage": *string*

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

Default: "TemporaryStorage".

Handlers

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

Table of Contents

Name

Chain — dispatch exchange to ordered list of filters

Description

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

Usage

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

Properties

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

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

The chain dispatches to dispatch the exchange to these Filters in the order they appear in the array.

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

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

The chain dispatches to this handler once the exchange has traversed all of the specified filters.

Example

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

Javadoc

org.forgerock.openig.filter.Chain

Name

ClientHandler — submit exchange requests to remote servers

Description

Submits exchange requests to remote servers.

Usage

```
{
     "name": string,
     "type": "ClientHandler",
     "config": {
         "httpClient": HttpClient reference
     }
}
```

Properties

"httpClient": *HttpClient reference, optional*

HttpClient used when making HTTP requests to other servers.

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

Default: use the heap object named "HttpClient", which matches the name of the HttpClient object created by the gateway servlet by default with default settings

If you overload that name by configuring your own HttpClient object named "HttpClient", then use that heap object.

Example

The following object configures a `ClientHandler` named `Client` that uses the HTTP client configuration specified in the configuration object with name `MyHttpClient`.

```
{
  "name": "Client",
  "type": "ClientHandler",
  "config": {
    "httpClient": "MyHttpClient"
  }
}
```

Javadoc

org.forgerock.openig.handler.ClientHandler

Name

DispatchHandler — dispatch to one of a list of handlers

Description

Dispatches to one of a list of handlers. When an exchange is handled, each handler's `condition` is evaluated. If a condition expression yields `true`, then the exchange 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.

`"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.

`"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 exchange session, then the user has not authenticated with the SAML 2.0 Identity Provider, so OpenIG dispatches to a "SPInitiatedSSORedirectHandler" to initiate SAML 2.0 single sign-on from the Service Provider, which is OpenIG. All other requests go through a "LoginChain" handler.

```
{
    "name": "DispatchHandler",
    "type": "DispatchHandler",
    "config": {
        "bindings": [
            {
                "condition": "${matches(exchange.request.uri.path, '^/saml')}",
                "handler": "SamlFederationHandler"
            },
            {
                "condition": "${empty exchange.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

MonitorEndpointHandler — return basic audit statistics in JSON format

Description

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

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

Usage

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

Example

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

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

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

```
{
    "ForgeRock.com route": {
        "in progress": 0,
        "completed": 6,
        "internal errors": 0
    },
    "ForgeRock.org route": {
        "in progress": 0,
        "completed": 15,
        "internal errors": 0
    },
    "Monitor route": {
        "in progress": 1,
        "completed": 1,
        "internal errors": 0
    },
    "Static login route": {
        "in progress": 0,
        "completed": 12,
        "internal errors": 0
    },
    "HTTP Basic route": {
        "in progress": 0,
        "completed": 21,
        "internal errors": 3
    }
}
```

Javadoc

org.forgerock.openig.audit.monitor.MonitorEndpointHandler

Name

Route — Configuration for handling a specified Exchange condition

Description

In OpenIG, a route is represented by a separate JSON configuration file and that handles an Exchange when a specified condition is met.

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

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

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

Usage

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

Properties

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

For this route, dispatch the exchange to this handler.

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

`"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.

`"baseURI"`: *string, optional*

URI on which to rebase the request URL.

If this is set, the request URL is rebased using the value. Rebasing changes the scheme, host, and port of the request URL. Rebasing does not affect the path, query string, or fragment.

Default: the request URL remains unchanged.

`"condition"`: *expression, optional*

Whether the route accepts to handle the Exchange.

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

`"name"`: *string, optional*

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

Default: Route configuration file name

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

Session storage implementation used by this route, such as a JwtSession.

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 `exchange.session`.

Name

Router — Route processing to distinct configurations

Description

A Router is a handler that routes Exchange processing to separate configuration files. Each separate configuration file then defines a Route. See the *Description* section of the Route reference for details.

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

Usage

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

An alternative value for "type" is "RouterHandler".

Properties

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

Default handler for this Router.

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

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

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

`"directory"`: *expression, optional*

Base directory from which to load configuration files for routes.

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

`"scanInterval"`: *integer, optional*

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

Default: 10 (seconds)

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

Javadoc

org.forgerock.openig.handler.router.RouterHandler

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).

Usage

```
{
    "name": string,
    "type": "SamlFederationHandler",
    "config": {
        "assertionMapping": object,
        "redirectURI": string,
        "assertionConsumerEndpoint": string,
        "authnContext": string,
        "authnContextDelimiter": string,
        "logoutURI": string,
        "sessionIndexMapping": string,
        "singleLogoutEndpoint": string,
        "singleLogoutEndpointSoap": string,
        "SPinitiatedSLOEndpoint": string,
        "SPinitiatedSSOEndpoint": string,
        "subjectMapping": string
    }
}
```

Properties

`"assertionMapping"`: *object, required*

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

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

The following shows an example of an "assertionMapping" object.

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

If the incoming assertion contains the statement:

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

```
mailPassword = costanza
```

Then the following values are set in the session:

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

```
password = costanza
```

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

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

`"redirectURI"`: *string, required*

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

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

`"assertionConsumerEndpoint"`: *string, optional*

Default: "fedletapplication" (same as the Fedlet)

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

`"authnContext"`: *string, optional*

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

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

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

Default: map to `session.authnContext`

`"authnContextDelimiter"`: *string, optional*

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

Default: `|`

`"logoutURI"`: *string, optional*

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

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

`"sessionIndexMapping"`: *string, optional*

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

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

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

Default: map to `session.sessionIndex`

`"singleLogoutEndpoint"`: *string, optional*

Default: "fedletSLORedirect" (same as the Fedlet)

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

`"singleLogoutEndpointSoap"`: *string, optional*

Default: "fedletSloSoap" (same as the Fedlet)

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

`"SPinitiatedSLOEndpoint"`: *string, optional*

Default: "SPInitiatedSLO"

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

`"SPinitiatedSSOEndpoint"`: *string, optional*

Default: "SPInitiatedSSO"

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

`"subjectMapping"`: *string, optional*

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

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

As an example, if you set `"subjectMapping": "mySubjectName"`, then OpenIG sets `exchange.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 has access to the following global objects.

Any parameters passed as "args"

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

Take care when naming keys in the "args" object. If you reuse the name of another global object, then you overwrite the global object value with the specified "args" value.

`exchange`

The exchange provides access to the HTTP request and response.

The request is created and populated before calling the handler. The handler is responsible for creating and for populating the response in the exchange.

`globals`

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

`http`

The http object provides an embedded HTTP client.

Use this client to perform outbound HTTP requests.

`ldap`

The ldap object provides an embedded LDAP client.

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

`logger`

The logger object provides access to the server log sink.

Usage

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

Properties

`"type"`: *string, required*

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

`"file"`: *string*

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 the section, *Installing OpenIG* in the *Gateway Guide*.

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

`"source"`: *string*

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

`"args"`: *map, optional*

Parameters passed from the configuration to the script.

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

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

The script can then access the "args" parameters in the same way as other global objects. The following example sets the response reason to "I'm a teapot".

```
exchange.response.reason = reason[1]
```

Javadoc

org.forgerock.openig.handler.ScriptableHandler

Name

SequenceHandler — process exchange through sequence of handlers

Description

Processes an exchange 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.

`"postcondition"`: *expression, optional*

Evaluated to determine if the sequence continues. Default: unconditional.

Javadoc

org.forgerock.openig.handler.SequenceHandler

Name

StaticResponseHandler — create static response in HTTP exchange

Description

Creates a static response in an HTTP exchange.

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`.

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 perform filtering of the request and response of an HTTP exchange.

Table of Contents

Name

AssignmentFilter — conditionally assign values to expressions

Description

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

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

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

`"condition"`: *expression, optional*

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

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

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

`"value"`: *expression, optional*

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

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

Javadoc

org.forgerock.openig.filter.AssignmentFilter

Name

CaptureFilter — capture request and response messages

Description

Captures request and response messages for further analysis.

Interface Stability: Deprecated. Use a CaptureDecorator instead.

Usage

```
{
     "name": string,
     "type": "CaptureFilter",
     "config": {
         "file": expression,
         "charset": string,
         "condition": expression,
         "captureEntity": boolean
     }
}
```

Properties

`"file"`: *expression, required*

The path of the file where captured output should be written.

`"charset"`: *string, optional*

The character set to encode captured output with. Default: `"UTF-8"`.

`"condition"`: *expression, optional*

The condition to evaluate to determine whether to capture an exchange. Default: unconditional.

`"captureEntity"`: *boolean, optional*

Whether the message entity should be captured.

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

Default: true

Examples

Log the entire request and response:

```
{
    "name": "LogToTemporaryFile",
    "type": "CaptureFilter",
    "config": {
        "file": "/tmp/gateway.log"
    }
}
```

Log the request and response. Do not log the entity:

```
{
    "name": "LogToTemporaryFile",
    "type": "CaptureFilter",
    "config": {
        "file": "/tmp/gateway.log",
        "captureEntity": false
    }
}
```

You can use the CaptureFilter to capture the exchange before sending the request and when receiving the response as in the following example.

```
{
    "name": "OutgoingChain",
    "type": "Chain",
    "config": {
        "filters": [ "LogToTemporaryFile" ],
        "handler": "ClientHandler"
    }
},
{
    "name": "LogToTemporaryFile",
    "type": "CaptureFilter",
    "config": {
        "captureEntity": false,
        "file": "/tmp/gateway.log"
    }
}
```

Javadoc

org.forgerock.openig.filter.CaptureFilter

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 in the exchange 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.

`"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, 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 in the exchange 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']}`.

`"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.

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

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

Default: store the match result itself.

Examples

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

```
{
    "name": "WikiNoncePageExtract",
    "type": "EntityExtractFilter",
    "config": {
        "messageType": "response",
        "target": "${exchange.wikiNonce}",
        "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 `loginPage.found = true` to be used in a SwitchFilter to post the login credentials.

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

Javadoc

org.forgerock.openig.filter.EntityExtractFilter

Name

ExceptionFilter — catch exceptions when handling request

Description

Catches any exceptions thrown during handling of a request. This allows friendlier error pages to be displayed than would otherwise be displayed by the container. Caught exceptions are logged with a log level of `WARNING` and the exchange is diverted to the specified exception handler.

Usage

```
{
     "name": string,
     "type": "ExceptionFilter",
     "config": {
         "handler": Handler reference
     }
}
```

Properties

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

Dispatch to this handler in the event of caught exceptions.

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

Javadoc

org.forgerock.openig.filter.ExceptionFilter

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 exchange-scoped 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 the section, *Login With Credentials From a File* in the *Gateway Guide*.

Properties

`"file"`: *expression, required*

The file containing the record to be read.

`"charset"`: *string, optional*

The character set the file is encoded in. Default: `"UTF-8"`.

`"header"`: *boolean, optional*

Indicates the first line of the file contains the set of defined field keys. Default: `true`.

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

Explicit field keys in the order they appear in a record, overriding any existing field header. Default: use field header.

`"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 ${exchange.credentials} and the record has a "username" field and a "password" field mentioned in the "fields" list, Then you can access the user name as `${exchange.credentials.username}` and the password as `${exchange.credentials.password}`.

`"key"`: *string, required*

The name of the field in the file to perform the lookup on.

`"value"`: *expression, required*

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

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 in the exchange 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 the exchange 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.

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

Expression that yields the password to supply during authentication.

`"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.

`"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

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

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

Usage

```
{
    "name": string,
    "type": "LocationHeaderFilter",
    "config": {
        "baseURI": expression
    }
}
```

An alternative value for "type" is "RedirectFilter".

Properties

"baseURI": *expression*, *required*

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

Example

```
{
     "name": "LocationRewriter",
     "type": "LocationHeaderFilter",
     "config": {
         "baseURI": "https://proxy.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 filter configuration includes the client credentials that are used to authenticate to identity providers. The client credentials can be included directly in the configuration, or retrieved in some other way using an expression.

In the case where all users share the same identity provider, you can configure the filter as a client of a single provider. 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/provider?goto=url`

Redirect the end user for authorization with the specified provider.

The provider then authenticates the end user and obtains authorization consent from the end user 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 exchange and redirects the user-agent to the specified URL.

`clientEndpoint/logout?goto=url`

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

`clientEndpoint/callback`

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

If the callback is handled successfully, the filter saves the authorization state in the exchange 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,
    "loginHandler": Handler reference,
    "cacheExpiration": duration string,
    "providerHandler": Handler reference,
    "providers": [ provider configuration object, ... ],
    "target": expression,
    "defaultLoginGoto": expression,
    "defaultLogoutGoto": expression,
    "requireHttps": boolean,
    "requireLogin": boolean,
    "scopes": [ expression, ...]
  }
}
```

Properties

`"clientEndpoint"`: *expression, required*

Base URI for the filter.

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

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

Invoke this Handler if authentication fails.

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

If this handler is invoked, then the "target" in the exchange is populated with information about the provider, and the error

The failure object in the "target" is a simple map. It has the following layout.

```
{
    "provider": "provider 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"
}
```

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"

`"loginHandler"`: *Handler reference, required when multiple providers are configured*

Invoke this Handler the user must choose a provider.

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

This handler allows the user to choose a provider, as in the following example that allows the user to choose between "openam" and "google".

```
{
   "name": "NascarPage",
   "type": "StaticResponseHandler",
   "config": {
       "status": 200,
       "entity": "<html><p><a
                  href='/openid/login?provider=openam&goto=${urlEncode(exchange.originalUri)}'
                  >OpenAM Login</a></p>
                  <p><a
                  href='/openid/login?provider=google&goto=${urlEncode(exchange.originalUri)}'
                  >Google Login</a></p>
                  </html>"
   }
}
```

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

Duration for which to cache user-info resources.

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

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

Durations are not case sensitive.

Negative durations are not supported.

The following units can be used in durations.

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

Default: 20 seconds

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

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

Invoke this HTTP client handler to communicate with the provider.

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.

`"providers"`: *array of provider configuration objects, required*

One or more provider configuration objects that indicate how the client communicates with authorization providers.

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 user info endpoint URL.

Provider configuration objects have the following layout:

```
{
    "name": string,
    "clientId": expression,
    "clientSecret": expression,
    "wellKnownConfiguration": URL string,
    "authorizeEndpoint": URI expression,
    "tokenEndpoint": URI expression,
    "userInfoEndpoint": URI expression,
    "scopes": [ "expression", ... ]
}
```

The provider configuration object properties are as follows.

`"name"`: *string, required*

A name for the provider configuration.

`"clientID"`: *expression, required*

The `client_id` obtained when registering with the provider.

`"clientSecret"`: *expression, required*

The `client_secret` obtained when registering with the provider.

`"wellKnownConfiguration"`: *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 "wellKnownConfiguration"*

The URL to the provider's OAuth 2.0 authorization endpoint.

`"tokenEndpoint"`: *expression, required unless obtained through "wellKnownConfiguration"*

The URL to the provider's OAuth 2.0 token endpoint.

`"userInfoEndpoint"`: *expression, optional*

The URL to the provider's OpenID Connect UserInfo endpoint.

Default: no UserInfo is obtained from the provider.

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

Overrides the list of scopes specified globally for the filter.

Default: use the list of scopes specified globally for the filter.

`"target"`: *expression, optional*

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

Default: `${exchange.openid}`

`"defaultLoginGoto"`: *expression, optional*

The URI to redirect to after successful authentication and authorization.

Default: return an empty page.

`"defaultLogoutGoto"`: *expression, optional*

The URI to redirect to after successful logout.

Default: return an empty page.

`"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.

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

Expression that yields the scope strings to request of any providers contacted by this filter.

Instead of or in addition to specifying scopes globally here, you can also specify a list of scopes per provider in each provider's configuration. Per-provider scope lists then override this list.

Default: do not specify scopes.

Example

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

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

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

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

```
{
    "name": "OpenIDConnectClient",
    "type": "OAuth2ClientFilter",
    "config": {
        "target"                : "${exchange.openid}",
        "scopes"                : ["openid","profile","email"],
        "clientEndpoint"        : "/openid",
        "loginHandler"          : "NascarPage",
        "failureHandler"        : "Dump",
        "providerHandler"       : "ClientHandler",
        "defaultLoginGoto"      : "/dump",
        "defaultLogoutGoto"     : "/unprotected",
        "requireHttps"          : false,
        "requireLogin"          : true,
        "providers"             : [
            {
                "name"          : "openam",
                "wellKnownConfiguration"
                                :
                  "http://openam.example.com:8080/openam/.well-known/openid-configuration",
                "clientId"      : "********",
                "clientSecret"  : "********"
            },
            {
                "name"          : "google",
                "wellKnownConfiguration"
```

```
                                :
                  "https://accounts.google.com/.well-known/openid-configuration",
                "clientId"      : "**************.apps.googleusercontent.com",
                "clientSecret"  : "***************"
            }
        ]
    }
}
```

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

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 an Exchange containing an OAuth 2.0 access token

Description

An OAuth2ResourceServerFilter is a filter that validates an exchange 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 includes the token info from the authorization server response as JSON in the exchange at the location specified by the "target" setting. Subsequent filters and handlers can access the token info through the exchange.

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,
    "requireHttps": boolean,
    "realm": string,
    "target": expression
  }
}
```

An alternative value for "type" is "OAuth2RSFilter".

Properties

"providerHandler": *Handler reference, required*

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.

Usually set this to the name of a ClientHandler configured in the heap.

"scopes": *array of expressions, required*

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

"tokenInfoEndpoint": *URL string, required*

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

"cacheExpiration": *duration string, optional*

Duration for which to cache OAuth 2.0 access tokens.

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

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

Durations are not case sensitive.

Negative durations are not supported.

The following units can be used in durations.

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

Default: 1 minute

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

`"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

`"target"`: *expression, optional*

Where to store the OAuth 2.0 access token in the exchange, such as `${exchange.token}`.

Default: `${exchange.oauth2AccessToken}`

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.OAuth2ResourceServerFilter

See Also

The OAuth 2.0 Authorization Framework

OAuth 2.0 Bearer Token Usage

Name

ScriptableFilter — process exchange by using a script

Description

Processes an exchange by using a script.

The script has access to the following global objects.

Any parameters passed as "args"

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

Take care when naming keys in the "args" object. If you reuse the name of another global object, then you overwrite the global object value with the specified "args" value.

`exchange`

The exchange provides access to the HTTP request and response.

`globals`

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

`http`

The http object provides an embedded HTTP client.

Use this client to perform outbound HTTP requests.

`ldap`

The ldap object provides an embedded LDAP client.

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

`logger`

The logger object provides access to the server log sink.

`next`

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

When finished processing the request, call the `next.handle(exchange)` method to call the next filter in the current chain. Everything in the script subsequent to this method call deals with the exchange response.

Usage

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

Properties

`"type"`: *string, required*

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

`"file"`: *string*

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 the section, *Installing OpenIG* in the *Gateway Guide*.

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

`"source"`: *string*

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

`"args"`: *object, optional*

Parameters passed from the configuration to the script.

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

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

The script can then access the "args" parameters in the same way as other global objects. The following example sets the response reason to "I'm a teapot".

```
exchange.response.reason = reason[1]
```

Javadoc

org.forgerock.openig.filter.ScriptableFilter

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 exchange-scoped 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.

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

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

Example

Using the users sessionid from a cookie, query the database to find the user logged in and set the profile attributes in the exchange:

```
{
        "name": "SqlAttributesFilter",
        "type": "SqlAttributesFilter",
        "config": {
              "target": "${exchange.sql}",
              "dataSource": "java:comp/env/jdbc/mysql",
              "preparedStatement": "SELECT f.value AS 'first', l.value AS
                'last', u.mail AS 'email', GROUP_CONCAT(CAST(r.rid AS CHAR)) AS
                'roles'
                FROM sessions s
                INNER JOIN users u
                ON ( u.uid = s.uid AND u.status = 1 )
                LEFT OUTER JOIN profile_values f
                ON ( f.uid = u.uid AND f.fid = 1 )
                LEFT OUTER JOIN profile_values l
                ON ( l.uid = u.uid AND l.fid = 2 )
                LEFT OUTER JOIN users_roles r
                ON ( r.uid = u.uid )
                WHERE (s.sid = ? AND s.uid <> 0) GROUP BY s.sid;",
              "parameters": [ "${exchange.request.cookies
                [keyMatch(exchange.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 within exchange object

Description

Creates a new request within the exchange object. It replaces any request that may already be present in the exchange. 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.

Usage

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

`"restore"`: *boolean, optional*

Whether to restore the original request after the exchange is handled.

Default: false

`"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.

Example

```
{
     "name": "LoginRequestFilter",
     "type": "StaticRequestFilter",
     "config": {
         "method": "POST",
         "uri": "http://10.10.0.2:8080/wp-login.php",
         "form": {
             "log": [ "george" ],
             "pwd": [ "bosco" ],
             "rememberme": [ "forever" ],
             "redirect_to": [ "http://portal.example.com:8080/wp-admin/" ],
             "testcookie": [ "1" ]
         }
     }
}
```

Javadoc

org.forgerock.openig.filter.StaticRequestFilter

Name

SwitchFilter — divert exchange to other handler

Description

Conditionally diverts the exchange to another handler. If a `condition` evaluates to `true`, then the exchange 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 exchange is handled.

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

Conditions to test (and handler to dispatch to, if `true`) after the exchange is handled.

`"condition"`: *expression, optional*

Condition to evaluate to determine if exchange should be dispatched to handler.

Default: unconditional dispatch to handler.

`"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.

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": "${exchange.response.status == 200}",
                "handler": "LoginRequestHandler"
            }
        ]
    }
}
```

Javadoc

org.forgerock.openig.filter.SwitchFilter

Decorators

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

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

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

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

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

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

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

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

```
{
```

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

Decorations are inherited as follows.

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

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

OpenIG defines some decorators, such as "audit", "capture", and "timer". You can use these without configuring them explicitly. For details see the reference section, *Gateway Servlet*.

Take care when defining decorator names not to use names that unintentionally clash with field names for the decorated objects. For all heap objects, avoid decorators named "config", "name", and "type". For Routes, avoid decorators named "baseURI", "condition", "globalDecorators", "heap", "handler", "name", and "session". In `config.json`, also avoid "logSink" and "temporaryStorage". In addition, avoid decorators named "comment" or "comments". The best way to avoid a clash with other field names is to avoid OpenIG reserved field names, which include all purely alphanumeric field names. Instead use dots or dashes in your decorator names, such as "my-decorator".

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

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

OpenIG applies decorations in this order.

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

Interface Stability: Evolving

Table of Contents

Name

AuditDecorator — trigger notification of audit events for Filters and Handlers

Description

Triggers notification of audit events for applicable Filters and Handlers.

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

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

`exchange`

A reference to the Exchange involved in the event.

Note that does not represent a copy of the exchange at event creation. If notification receivers process the event asynchronously, the content of the exchange can change between the time the event is created and the time the receiver processes the audit event.

`source`

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

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

`tags`

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

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

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

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

  When decorating a Filter, realize that the filter returns after handling the response, even if it only filters the request and so does nothing to the response but pass it along.
- `completed`: This event happens when the processing unit under audit has successfully handled the exchange. This tag always complements a `response` tag.

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

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

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

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

`timestamp`

Timestamp indicating when the event happened, with millisecond precision.

Decorated Object Usage

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

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

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

`"type"`: string, required

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

`"config"`: object, required unless empty

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

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

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

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

Examples

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

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

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

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

To observe audit events, use a registered audit agent such as a MonitorEndpointHandler.

Javadoc

org.forgerock.openig.audit.decoration.AuditDecorator

Name

CaptureDecorator — capture request and response messages

Description

Captures request and response messages for further analysis.

Decorator Usage

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

The decorator configuration has these properties.

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

Capture requests and responses to this LogSink.

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

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

`"captureEntity"`: *boolean, optional*

Whether the message entity should be captured.

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

Default: false

`"captureExchange"`: *boolean, optional*

Whether the exchange, excluding the request and response, should be captured as JSON.

Default: false

Decorated Object Usage

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

"name": string, required except for inline objects

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

"type": string, required

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

"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 exchange in JSON format, excluding the request and the response:

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

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

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

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

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

```
                "config": {
                    "status": 200,
                    "reason": "OK",
                    "entity": "<html><p>Hello, World!</p></html>"
                }
            }
        }
    },
    "capture": [
        "filtered_request",
        "filtered_response"
    ]
}
```

To capture the exchange 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": {
                "captureExchange": 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 applicable Filters and Handlers. OpenIG writes the records to the LogSink configured for the decorated heap object. If no LogSink is defined for the decorated heap object, then OpenIG writes to the LogSink configured for the heap. Records include the time elapsed while processing the exchange, and for Filters the elapsed time spent processing the exchanged within the Filter itself.

OpenIG records times at log level `STAT`.

The TimerDecorator is not applicable to the *Gateway Servlet*, as the Gateway Servlet is not declared in the heap. Instead, OpenIG always records Gateway Servlet times at log level `STAT`.

Decorator Usage

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

A TimerDecorator does not have configurable properties.

The Gateway Servlet creates a default TimerDecorator named "timer" at startup time in the top-level heap, so you can use "timer" as the decorator name without adding the decorator declaration explicitly.

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

`"config"`: object, required unless empty

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

***`decorator name`*: boolean, required**

OpenIG looks for the presence of the *decorator name* field for the TimerDecorator.

To activate the timer, set the value of the *decorator name* field to `true`.

To deactivate the TimerDecorator temporarily, set the value to `false`.

Examples

To record times spent within the client handler, and elapsed time for operations traversing the client handler, use a configuration such as the following.

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

This configuration could result in the following log messages.

```
TUE DEC 02 17:20:08 CET 2014 (STAT) @Timer[top-level-handler]
Started
------------------------------
TUE DEC 02 17:20:08 CET 2014 (STAT) @Timer[top-level-handler]
Elapsed time: 40 ms
```

When you decorate a Filter with a TimerDecorator, OpenIG can record two timer messages in the LogSink: the elapsed time for operations traversing the Filter, and the elapsed time spent within the Filter.

To record times spent within all Filters and the "handler", decorate the Route as in the following example.

```
{
    "handler": {
        "type": "Chain",
        "config": {
            "filters": [
                {
                    "type": "OAuth2ResourceServerFilter",
                    "config": {
                        "providerHandler": "ClientHandler",
```

```
                        "scopes": [
                            "mail",
                            "employeenumber"
                        ],
                        "tokenInfoEndpoint":
                            "http://openam.example.com:8088/openam/oauth2/tokeninfo",
                        "requireHttps": false,
                        "target": "${exchange.token}"
                    },
                    "timer": true
                },
                {
                    "type": "ScriptableFilter",
                    "config": {
                        "type": "application/x-groovy",
                        "source":
                            "import org.forgerock.json.fluent.JsonValue;
                             logger.info(exchange.token.asJsonValue() as String);
                             exchange.username = exchange.token.info.mail;
                             exchange.password = exchange.token.info.employeenumber;
                             next.handle(exchange)"
                    },
                    "timer": true
                },
                {
                    "type": "StaticRequestFilter",
                    "config": {
                        "method": "POST",
                        "uri": "http://www.example.com:8081",
                        "form": {
                            "username": [
                                "${exchange.username}"
                            ],
                            "password": [
                                "${exchange.password}"
                            ]
                        }
                    },
                    "timer": true
                }
            ],
            "handler": "ClientHandler"
        }
    },
    "condition": "${matches(exchange.request.uri.path, '^/rs')}",
    "timer": true
}
```

This configuration could result in the following log messages.

```
TUE DEC 02 17:26:43 CET 2014 (STAT) @Timer[top-level-handler]
Started
------------------------------
TUE DEC 02 17:26:43 CET 2014 (STAT) @Timer[{OAuth2ResourceServerFilter}/handler/config/filters/0]
Started
```

```
------------------------------
TUE DEC 02 17:26:43 CET 2014 (STAT) @Timer[{ScriptableFilter}/handler/config/filters/1]
Started
------------------------------
...
------------------------------
TUE DEC 02 17:26:43 CET 2014 (STAT) @Timer[{StaticRequestFilter}/handler/config/filters/2]
Started
------------------------------
TUE DEC 02 17:26:43 CET 2014 (STAT) @Timer[{StaticRequestFilter}/handler/config/filters/2]
Elapsed time: 5 ms
------------------------------
TUE DEC 02 17:26:43 CET 2014 (STAT) @Timer[{StaticRequestFilter}/handler/config/filters/2]
Elapsed time (within the object): 0 ms
------------------------------
TUE DEC 02 17:26:43 CET 2014 (STAT) @Timer[{ScriptableFilter}/handler/config/filters/1]
Elapsed time: 24 ms
------------------------------
TUE DEC 02 17:26:43 CET 2014 (STAT) @Timer[{ScriptableFilter}/handler/config/filters/1]
Elapsed time (within the object): 18 ms
------------------------------
TUE DEC 02 17:26:43 CET 2014 (STAT) @Timer[{OAuth2ResourceServerFilter}/handler/config/filters/0]
Elapsed time: 37 ms
------------------------------
TUE DEC 02 17:26:43 CET 2014 (STAT) @Timer[{OAuth2ResourceServerFilter}/handler/config/filters/0]
Elapsed time (within the object): 13 ms
------------------------------
TUE DEC 02 17:26:43 CET 2014 (STAT) @Timer[top-level-handler]
Elapsed time: 37 ms
```

You can then deactivate the timer by setting the values to `false`.

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

Javadoc

org.forgerock.openig.decoration.timer.TimerDecorator

Miscellaneous Heap Objects

Table of Contents

Name

ConsoleLogSink — log to standard error

Description

A log sink that writes log entries to the standard error stream.

Usage

```
{
     "name": string,
     "type": "ConsoleLogSink",
     "config": {
         "level": string,
         "stream": string
    }
}
```

Properties

`"level"`: *string, optional*

The level of log entries to display in the console.

Must be one of the following settings. These are ordered from most verbose to least verbose. `ALL` (log all messages), `TRACE` (log low-level tracing information), `DEBUG` (log debugging information), `STAT` (log performance measurement statistics), `CONFIG` (log configuration information), `INFO` (log general information), `WARNING` (log potential problems), `ERROR` (log serious failures), `OFF` (log no messages).

Default: `INFO`.

`"stream"`: *string, optional*

The standard output to use to display logs in the console.

Must be one of the following settings. `ERR` (use standard error: System.err), `OUT` (use standard output: System.out), `AUTO` (select standard error or output depending on the message log level: TRACE, DEBUG, STAT, CONFIG, INFO print to System.out, WARNING and ERROR print to System.err).

Default: `ERR`.

Example

```
{
     "name": "LogSink",
     "comment": "Default sink for logging information.",
     "type": "ConsoleLogSink",
     "config": {
         "level": "DEBUG",
         "stream": "AUTO"
     }
}
```

Javadoc

org.forgerock.openig.log.ConsoleLogSink

Name

FileLogSink — log to a file

Description

A log sink that writes log entries to a file using the UTF-8 character set.

Usage

```
{
     "name": string,
     "type": "FileLogSink",
     "config": {
         "file": configuration expression,
         "level": string
     }
}
```

Properties

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

The path to the log file.

A configuration expression is independent of the exchange, so do not use expressions that reference exchange properties. You can, however, use `${env['variable']}`, `${system['property']}`, and all the built-in functions.

`"level"`: *string, optional*

The level of log entries to display in the console.

Must be one of the following settings. These are ordered from most verbose to least verbose. `ALL` (log all messages), `TRACE` (log low-level tracing information), `DEBUG` (log debugging information), `STAT` (log performance measurement statistics), `CONFIG` (log configuration information), `INFO` (log general information), `WARNING` (log potential problems), `ERROR` (log serious failures), `OFF` (log no messages).

Default: `INFO`.

Example

```
{
     "name": "LogSink",
     "type": "FileLogSink",
     "config": {
         "file": "${system['log'] ? system['log'] : '/tmp/proxy.log'}",
         "level": "DEBUG"
     }
}
```

Javadoc

org.forgerock.openig.log.FileLogSink

Name

HttpClient — group settings and submit requests to remote servers

Description

Groups settings and submits requests to remote servers.

You configure `HttpClient` objects in order to specify HTTP client settings for ClientHandler objects.

Usage

```
{
  "name": string,
  "type": "HttpClient",
  "config": {
    "connections": number,
    "disableReuseConnection": boolean,
    "disableRetries": boolean,
    "hostnameVerifier": string,
    "soTimeout": duration string,
    "connectionTimeout": duration string,
    "keyManager": KeyManager reference(s),
    "trustManager": TrustManager reference(s),
  }
}
```

Properties

`"connections"`: *number, optional*

The maximum number of connections to open, from 1-64 inclusive.

Default: 64

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

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

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

Durations are not case sensitive.

Negative durations are not supported.

The following units can be used in durations.

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

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

Default: 10 seconds

`"disableRetries"`: *boolean, optional*

Whether to disable automatic retries for failed requests.

Default: `true`

`"disableReuseConnection"`: *boolean, optional*

Whether to disable connection reuse.

Default: `true`

`"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.
- `BROWSER_COMPATIBLE`: 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 all subdomains, so `*.example.com` matches `www.example.com` and `some.host.example.com`.
- `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`

`"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.

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

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

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

Durations are not case sensitive.

Negative durations are not supported.

The following units can be used in durations.

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

Default: 10 seconds

`"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 trust store. The default Java trust store depends on the Java environment. For example, `$JAVA_HOME/lib/security/cacerts`.

Example

```
{
    "name": "HttpClient",
    "type": "HttpClient",
    "config": {
        "connections": 64,
        "disableReuseConnection": true,
        "disableRetries": true,
        "hostnameVerifier": "BROWSER_COMPATIBLE",
        "soTimeout": "10 seconds",
        "connectionTimeout": "10 seconds",
        "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.http.HttpClient

Name

JwtSession — store sessions in encrypted JWT cookies

Description

A JwtSession object holds settings for storing session information in encrypted JSON Web Token (JWT) cookies.

In this context "encrypted JWT cookie" means an HTTP cookie whose value is an encrypted JWT. The payload of the encrypted JWT is a JSON representation of the session information.

The JWT cookie lifetime is "Session" (not persistent), meaning the user-agent deletes the JWT cookie when it shuts down.

When using this storage implementation, you must use data types for session information that can be mapped to JavaScript Object Notation (JSON). JSON allows strings, numbers, `true`, `false`, `null`, as well as arrays and JSON objects composed of the same primitives. Java and Groovy types that can be mapped include Java primitive types and `null`, String and CharSequence objects, as well as List and Map objects.

As browser cookie storage capacity is limited to 4 KB, and encryption adds overhead, take care to limit the size of any JSON that you store. Rather than store larger data in the session information, consider storing a reference instead.

When an exchange 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. `exchange.session` references the new session object. When the exchange then exits the route, the session object is closed, and serialized to a JWT cookie in this case, and `exchange.session` references the previous session object. Session information set inside the route is no longer available.

An HTTP client that performs multiple requests in a session that modify the content of its session can encounter inconsistencies in the session information. This is because OpenIG does not share JwtSessions across threads. Instead, each thread has its own JwtSession objects that it modifies as necessary, writing its own session to the JWT cookie regardless of what other threads do.

Usage

```
{
    "name": string,
    "type": "JwtSession",
    "config": {
        "keystore": KeyStore reference,
        "alias": string,
        "password": configuration expression,
        "cookieName": string
    }
}
```

An alternative value for "type" is "JwtSessionFactory".

Properties

`"keystore"`: *KeyStore reference, optional*

The key store 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.

`"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 key store.

A configuration expression is independent of the exchange, so do not use expressions that reference exchange properties. You can, however, use `${env['variable']}`, `${system['property']}`, and all the built-in functions.

`"cookieName"` *string, optional*

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

Default: `openig-jwt-session`

Example

The following example defines a JwtSession for storing session information in a JWT token cookie named `OpenIG`. The JWT is encrypted with a private key that is recovered using the alias `private-key`, and stored in the key store. The password is both the password for the key store and also the private key.

```
{
    "name": "JwtSession",
    "type": "JwtSession",
    "config": {
        "keystore": {
            "type": "KeyStore",
            "config": {
                "url": "file://${env['HOME']}/keystore.jks",
                "password": "${system['keypass']}"
            }
        },
        "alias": "private-key",
        "password": "${system['keypass']}",
        "cookieName": "OpenIG"
    }
}
```

Javadoc

org.forgerock.openig.jwt.JwtSessionFactory

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 key store 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.

`"password"`: *expression, required*

The password to read private keys from the key store.

`"alg"` *string, optional*

The certificate algorithm to use.

Default: the default for the platform, such as "SunX509"

Example

The following example configures a key manager that depends on a "KeyStore" configuration. The key store 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, TrustManager

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 key store file.

"password": *expression, optional*

The password to read private keys from the key store.

If the key store is used as a trust store 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.

"type": *string, optional*

The key store format.

Default: the default for the platform, such as "JKS"

Example

The following example configures a key store that references a Java Key Store file, `$HOME/keystore.jks`. The key store takes a password supplied as a Java system property when starting the container

where OpenIG runs, as in `-Dkeypass=password`. As the key store 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, TrustManager

Name

NullLogSink — discards log messages

Description

A log sink that discards all log messages.

Usage

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

Example

```
{
     "name": "LogSink",
     "type": "NullLogSink"
}
```

Javadoc

org.forgerock.openig.log.NullLogSink

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*

The initial length of memory buffer byte array. Default: 8192 (8 KiB).

`"memoryLimit"`: *number, optional*

The length limit of the memory buffer. Exceeding this limit results in promotion from memory to file. Default: 65536 (64 KiB).

`"fileLimit"`: *number, optional*

The length limit of the file buffer. Exceeding this limit results in a thrown exception. Default: 1048576 (1 MiB).

`"directory"`: *string, optional*

The directory where temporary files are created. If omitted, then the system-dependent default temporary directory is used (typically `"/tmp"` on Unix systems). Default: use system-dependent default.

Javadoc

org.forgerock.openig.io.TemporaryStorage

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.

"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, KeyStore

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: `${expression}`. The expression can be a simple reference to a value, a function call, or arbitrarily complex arithmetic, logical, relational and conditional operations. When supplied within a configuration parameter, an expression is always a string enclosed in quotation marks, for example: `"${exchange.request.method}"`.

Value Expressions

A value expression references a value relative to the scope supplied to the expression. In the current version of OpenIG, the supplied scope is always the HTTP exchange object. For example `"${exchange.request.method}"` references the method of an incoming HTTP request in the exchange scope.

An *lvalue-expression* is a specific type of value expression that references a value to be written. For example, `"${exchange.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 `"${exchange.request}"` and `"${exchange['request']}"` are equivalent.

In the case of arrays, the index of an element in the array is expressed as a number in brackets. For example, `"${exchange.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 & 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 can be called within an expression.

Syntax is `${function(parameter, ...)}`, where zero or more parameters are supplied to the function. For example, `"${toLowerCase(exchange.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.

Examples

```
"${exchange.request.uri.path == '/wordpress/wp-login.php'
   and exchange.request.form['action'][0] != 'logout'}"

"${exchange.request.uri.host == 'wiki.example.com'}"

"${exchange.request.cookies[keyMatch(exchange.request.cookies,'^SESS.*')][0].value}"

"${toString(exchange.request.uri)}"

"${exchange.request.method == 'POST' and exchange.request.uri.path == '/wordpress/wp-login.php'}"

"${exchange.request.method != 'GET'}"

"${exchange.request.headers['cookie'][0]}"

"${exchange.request.uri.scheme == 'http'}"

"${not (exchange.response.status == 302 and not empty exchange.session.gotoURL)}"

"${exchange.response.headers['Set-Cookie'][0]}"

"${exchange.request.headers['host'][0]}"
```

See Also

Exchange

Functions

Name

Functions — built-in functions to call within expressions

Description

A set of built-in functions that can be called from within expressions.

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.

indexOf

```
indexOf(string, substring)
```

Returns the index within a string of the first occurrence of a specified substring.

Parameters

string

the string to be searched.

substring

the value to search for within the string.

Returns

the index of the first instance of substring, or -1 if not found.

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

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

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

the length of the object, or 0 if length could not be determined.

matchingGroups

```
matchingGroups(string, pattern)
```

Returns an array of matching groups for a regular expression pattern against a 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

an array of matching groups, or `null` if no such match is found.

matches

```
matches(string, pattern)
```

Returns `true` if the string contains the specified regular expression pattern.

Parameters

string

the string to be searched.

pattern

a string containing the regular expression pattern to find.

Returns

true

if the string contains the specified regular expression pattern.

read

```
read(string)
```

Takes a file name as a `string`, and returns the content of the file as a plain string, or `null` on error (due to the file not being found, for example).

Either provide the absolute path to the file, or a path relative to the location of the Java system property `user.dir`.

Parameters

string

The name of the file to read.

Returns

string

The content of the file or `null` on error.

readProperties

```
readProperties(string)
```

Takes a Java Properties file name as a `string`, and returns the content of the file as a key/value map of properties, or `null` on error (due to the file not being found, for example).

Either provide the absolute path to the file, or a path relative to the location of the Java system property `user.dir`.

For example, to get the value of the `key` property in the properties file `/path/to/my.properties`, use `${readProperties('/path/to/my.properties')['key']}`.

Parameters

string

The name of the Java Properties file to read.

Returns

object

The key/value map of properties or `null` on error.

split

```
split(string, pattern)
```

Splits a string into an array of substrings around matches of the given regular expression pattern.

Parameters

string

the string to be split.

pattern

the regular expression to split substrings around.

Returns

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

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

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

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

the string with leading and trailing white space omitted.

trim

```
urlDecode(string)
```

Returns the URL decoding of the provided string.

Parameters

string

The string to be URL decoded, which may be `null`.

Returns

The URL decoding of the provided string, or `null` if string was `null`.

trim

```
urlEncode(string)
```

Returns the URL encoding of the provided string.

Parameters

string

The string to be URL encoded, which may be `null`.

Returns

The URL encoding of the provided string, or `null` if string was `null`.

Javadoc

org.forgerock.openig.el.Functions

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

Exchange Object Model

Expressions are evaluated within an exchange object model scope.

Table of Contents

Name

ClientInfo — HTTP Exchange client information

Description

Information about the client sending the request in the exchange object model

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`.

`"remoteAddress"`: string

The IP address of the client (or the last proxy) that sent the request

`"remoteHost"`: string

The fully qualified host name 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.openig.http.ClientInfo

Name

Exchange — HTTP exchange of request and response

Description

The root object for the exchange object model: an HTTP exchange of request and response. The exchange object model parallels the document object model, exposing elements of the exchange. It supports this by exposing a set of fixed properties and allowing arbitrary properties to be added.

Properties

`"clientInfo"`: object

Information about the client making the request in this HTTP exchange.

`"exchange"`: object

Self-referential property to make this the root object in the exchange object model.

`"originalUri"`: URI

The original target URI for the request, as received by the web container.

The value of this field is read-only.

`"request"`: object

The request portion of the HTTP exchange.

`"response"`: object

The response portion of the HTTP exchange.

`"principal"`: object

The principal associated with the request, or `null` if unknown.

`"session"`: object

Session context associated with the remote client. Exposes session attributes as name-value pairs, where both name and value are strings.

Javadoc

org.forgerock.openig.http.Exchange

Name

Request — HTTP exchange request

Description

An HTTP request message in an exchange object model.

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"`.

`"version"`: *string*

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

`"headers"`: *object*

Exposes message header fields as name-value pairs, where name is header name and value is an array of header values.

`"cookies"`: *object*

Exposes incoming request cookies as name-value pairs, where name is cookie name and value is an array of string cookie values.

`"form"`: *object*

Exposes query parameters and/or `application/x-www-form-urlencoded` entity as name-value pairs, where name is the field name and value is an array of string values.

`"entity"`: *object*

The message entity body (no accessible properties).

Javadoc

org.forgerock.openig.http.Request

Name

Principal — user principal in HTTP exchange

Description

Represents a user principal in an exchange object model, containing the name of the current authenticated user.

Properties

`"name"`: *string*

The name of the principal, or `null` if principal is undefined (user has not been authenticated).

Javadoc

java.security.Principal

Name

Response — HTTP exchange response

Description

An HTTP response message in an exchange object model.

Properties

`"status"`: *number*

The response status code. Example: 200.

`"reason"`: *string*

The response status reason. Example: `"OK"`.

`"version"`: *string*

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

`"headers"`: *object*

Exposes message header fields as name-value pairs, where name is header name and value is an array of header values.

`"entity"`: *object*

The message entity body (no accessible properties).

Javadoc

org.forgerock.openig.http.Response

Name

URI — Uniform Resource Identifier in HTTP exchange

Description

Represents a Uniform Resource Identifier (URI) reference in an exchange object model.

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.openig.util.MutableUri

Appendix A. Release Levels & 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