PayloadFactory Mediator

The **PayloadFactory Mediator** transforms or replaces the contents of a message. Each argument in the mediator configuration can be a static value, or you can specify an XPath or JSON expression to get the value at runtime by evaluating the provided expression against the existing SOAP message. You can configure the format of the request or response and map it to the arguments provided.

The PayloadFactory mediator is a content aware mediator. The ability to add inline expressions have been removed from this mediator.

Syntax | Configuration

**Syntax**

```xml
<payloadFactory media-type="xml | json">
  <format ../>
  <args>
    <arg (value="string" | expression= " {xpath} | {json} | {text} ")/>*
  </args>
</payloadFactory>
```

The `media-type` attribute specifies whether to format the message in XML, JSON, or text. If no media type is specified, the message is formatted in XML. If you want to change the payload type of the outgoing message, such as to change it to JSON, add the `messageType` property after the `<payloadFactory>` tag. For example:

```xml
... 
<payloadFactory>
  <property name="messageType" value="application/json" scope="axis2"/>
</payloadFactory>
```

**Configuration**

Parameters available to configure the PayloadFactory mediator are as follows:

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Payload Media-Type</strong></td>
<td>This parameter is used to specify whether the message payload should be created in JSON, XML, or text.</td>
</tr>
</tbody>
</table>
Payload Format

**Define Inline:** If this is selected, the payload format can be defined within the PayloadFactory mediator configuration by entering it in the text field that appears. To add content to the payload, enter variables for each value you want to add using the format $n$ (starting with 1 and incrementing with each additional variable, i.e., $1$, $2$, etc.). You will then create arguments in the same order as the variables to specify each variable’s actual value.

**Pick from Registry:** If this is selected, an existing payload format that is saved in the Registry can be selected. Click either **Governance Registry** or **Configuration Registry** as relevant to select the payload format from the resource tree.

Arguments

This section is used to add an argument that defines the actual value of each variable in the format definition. The arguments must be entered in the same order as the variables in the format, so that the first argument defines the value for variable $1$, the second argument defines the value for variable $2$, etc. An argument can specify a literal string (e.g., "John") or an XPath or JSON expression that extracts the value from the content in the incoming payload.

Examples

- Example 1: XML
- Example 2: JSON
- Example 3: Adding arguments
- Example 4: Suppressing the namespace
- Example 5: Including a complete SOAP envelope as the format
- Example 6: Uploading a file to an HTTP endpoint via a multipart request
- Example 7: Adding a literal argument
- Example 8: Adding a custom SOAP header
- Samples

This section provides examples of using PayloadFactory mediator to generate XML and JSON messages.

Example 1: XML

This example is based on Sample 17: Transforming / Replacing Message Content with PayloadFactory Mediator.
<definitions xmlns="http://ws.apache.org/ns/synapse">
  <sequence name="main">
    <in>
      <!-- using payloadFactory mediator to transform the request message -->
      <payloadFactory media-type="xml">
        <format>
          <m:getQuote xmlns:m="http://services.samples">
            <m:request>
              <m:symbol>$1</m:symbol>
            </m:request>
          </m:getQuote>
        </format>
        <args>
          <arg xmlns:m0="http://services.samples" expression="//m0:Code"/>
        </args>
      </payloadFactory>
    </in>
    <out>
      <!-- using payloadFactory mediator to transform the response message -->
      <payloadFactory media-type="xml">
        <format>
          <m:CheckPriceResponse xmlns:m="http://services.samples/xsd">
            <m:Code>$1</m:Code>
            <m:Price>$2</m:Price>
          </m:CheckPriceResponse>
        </format>
        <args>
          <arg xmlns:m0="http://services.samples/xsd" expression="//m0:symbol"/>
          <arg xmlns:m0="http://services.samples/xsd" expression="//m0:last"/>
        </args>
      </payloadFactory>
    </out>
    <send/>
  </sequence>
</definitions>

Example 2: JSON

This example sends a JSON message to the back end. For more information on using JSON with the ESB profile, see Working with JSON Message Payloads.

```xml
<payloadFactory media-type="json">
  <format>
    {
```
"coordinates": null,
"created_at": "Fri Jun 24 17:43:26 +0000 2011",
"truncated": false,
"favorited": false,
"id_str": "$1",
"entities": {
  "urls": [
  ],
  "hashtags": [
    {
      "text": "$2",
      "indices": [
        35,
        45
      ]
    }
  ],
  "user_mentions": [
  ]
},
"in_reply_to_user_id_str": null,
"contributors": null,
"text": "$3",
"retweet_count": 0,
"id": "##",
"in_reply_to_status_id_str": null,
"geo": null,
"retweeted": false,
"in_reply_to_user_id": null,

"source": "&lt;a href="http://sites.google.com/site/yorufukurou/" rel="nofollow"&gt;YoruFukurou&lt;/a&gt;",
"in_reply_to_screen_name": null,
"user": {
  "id_str": "##",
  "id": "##"
},
"place": null,
"in_reply_to_status_id": null
</format>
<args>
  <arg expression="$.entities.hashtags[0].text" evaluator="json"/>
  <arg expression="//entities/hashtags/text"/>
  <arg expression="//user/id"/>
  <arg expression="//user/id_str"/>
  <arg expression="$.user.id" evaluator="json"/>
  <arg expression="$.user.id_str" evaluator="json"/>
</args>
<payloadFactory>
<property name="messageType" value="application/json" scope="axis2"/>
</payloadFactory>

Note

By default, JSON messages are converted to XML when they are received by the PayloadFactory mediator. However, if you enable the JSON stream formatter and builder, incoming JSON messages are left in JSON format, which improves performance. To enable them, uncomment the following lines in `<PRODUCT_HOME>/repository/conf/axis2/axis2.xml`:

```
<!--messageFormatter contentType="application/json"
class="org.apache.axis2.json.JSONStreamFormatter"/-->  
<!--messageBuilder contentType="application/json"
class="org.apache.axis2.json.JSONStreamBuilder"/-->  
```

When the JSON stream formatter and builder are enabled, if you specify a JSON expression in the PayloadFactory mediator, you must use the `evaluator` attribute to specify that it is JSON. You can also use the evaluator to specify that an XPath expression is XML, or if you omit the evaluator attribute, XML is assumed by default. For example:

<table>
<thead>
<tr>
<th>XML</th>
<th>JSON</th>
</tr>
</thead>
</table>
| `<arg xmlns:m0=http://sample expression="//m0:symbol" evaluator="xml" />
| `<arg xmlns:m0=http://sample expression="//m0:symbol" />
| `<arg expression="$.user.id" evaluator="json" />

Example 3: Adding arguments

In the following configuration, the values for format parameters `code` and `price` will be assigned with values that are evaluated from arguments given in the specified order.
Example 4: Suppressing the namespace

To prevent the ESB profile from adding the default Synapse namespace in an element in the payload format, use `xmlns=""` as shown in the following example.

```xml
<ser:getPersonByUmid xmlns:ser="http://service.directory.com">
  <umid xmlns="">sagara</umid>
</ser:getPersonByUmid>
```

Example 5: Including a complete SOAP envelope as the format

In the following configuration, an entire SOAP envelope is added as the format defined inline. This is useful when you want to generate the result of the PayloadFactory mediator as a complete SOAP message with SOAP headers.

```xml
<payloadFactory media-type="xml">
  <format>
    <soapenv:Envelope
    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    >
    <soapenv:Body>
    <error>
    <mes>$1</mes>
    </error>
    </soapenv:Body>
    </soapenv:Envelope>
  </format>
  <args>
    <arg value="Your request did not return any results. Please enter a valid EIN and try again"/>
  </args>
</payloadFactory>
```

Example 6: Uploading a file to an HTTP endpoint via a multipart request
The below example configuration uses VFS to upload the file in the specified location to the given HTTP endpoint via a HTTP multipart request.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<proxy xmlns="http://ws.apache.org/ns/synapse" name="smooksample" startOnLoad="true" statistics="disable" trace="disable" transports="vfs">
  <target>
    <inSequence>
      <enrich>
        <source clone="true" type="body"/>
        <target property="originalBody" type="property"/>
      </enrich>
      <property name="messageType" scope="axis2" type="STRING" value="multipart/form-data"/>
      <payloadFactory media-type="xml">
        <format>
          <root xmlns="">
            <customFieldOne>$1</customFieldOne>
            <customFieldTwo>$2</customFieldTwo>
          </root>
        </format>
        <args>
          <arg value="Some value 1"/>
          <arg value="Some value 2"/>
          <arg evaluator="xml" expression="Strp:FILE_NAME"/>
          <arg evaluator="xml" expression="$ctx:originalBody"/>
        </args>
      </payloadFactory>
      <header name="Content-Type" scope="transport" value="multipart/form-data"/>
      <property name="messageType" scope="axis2" type="STRING" value="multipart/form-data"/>
      <property name="OUT_ONLY" scope="default" type="STRING" value="true"/>
    </send>
  </inSequence>
</target>
</proxy>
```
</target>
   <parameter name="transport.PollInterval">5</parameter>
   <parameter name="transport.vfs.FileURI">file:///YOUR_FILE_LOCATION</parameter>
   <parameter name="transport.vfs.ContentType">application/octet-stream</parameter>
   <parameter name="transport.vfs.ActionAfterProcess">DELETE</parameter>
   <parameter name="transport.vfs.FileNamePattern">.*\.*</parameter>
In the above example, the following property mediator configuration sets the message type as `multipart/form-data`.

```xml
<property name="messageType"
    scope="axis2"
    type="STRING"
    value="multipart/form-data"/>
```

The below `file` parameter of the payload factory mediator defines the HTTP multipart request.

```xml
    charset="US-ASCII"
    content-type="text/plain"
    filename="$3"
    name="file1">$4</file>
```

Also, the below property mediator configuration sets the content of the uploaded file.

```xml
<header name="Content-Type" scope="transport" value="multipart/form-data"/>
<property name="messageType"
    scope="axis2"
    type="STRING"
    value="multipart/form-data"/>
```

Example 7: Adding a literal argument

The following example adds a literal argument to the Payload Factory mediator, and sets it to true. This allows you to consider the type of the argument value as String and to stop processing it.
Following is a sample payload (i.e., a.json file), which you can process using the above configuration.

```
a.json
{"hello": 
"<pqr>abc</pqr>"}
```

You can use the below sample cURL command to send the request to the above configuration.

```
curl -d @a.json http://localhost:8280/payload -H "Content-Type: application/json" -v
```

You view the below output:

```
{"newValue": "{"pqr":"abc"}"}
```

If you do not add the `literal="true"` within the argument in the Payload Factory mediator of the above configuration, you view the output as follows:

```
{"newValue": "<pqr>abc</pqr>"}
```

**Example 8: Adding a custom SOAP header**

You can add custom SOAP headers to a request by using the PayloadFactory Mediator in a proxy service as shown in the example below.

```
<definitions xmlns="http://ws.apache.org/ns/synapse">
<proxy name="StockQuoteProxy"
transports="https http"
startOnLoad="true"
```
trace="disable">
<description/>
<target>
<endpoint>
<address uri="http://localhost:9001/services/SimpleStockQuoteService"/>
</endpoint>
</inSequence>
<log level="full">
<payloadFactory media-type="xml">
<format>
<soapenv:Envelope xmlns:soapenv="http://www.w3.org/2003/05/soap-envelope" xmlns:xsd="http://services.samples/xsd" xmlns:ser="http://services.samples">
<soapenv:Header>
<ser:authenticationRequest>
<userName xmlns="">$1</userName>
<password xmlns="">$2</password>
</ser:authenticationRequest>
</soapenv:Header>
<soapenv:Body>
<ser:getQuote>
<ser:request>
<xsd:symbol>$3</xsd:symbol>
</ser:request>
</ser:getQuote>
</soapenv:Body>
</soapenv:Envelope>
</format>
</payloadFactory>
</log>
</inSequence>
</target>
<publishWSDL uri="file:repository/samples/resources/proxy/sample_proxy_1.wsdl"/>
</proxy>
</sequence>
</log>
</sequence>
<sequence name="fault">
<log level="full">
<property name="MESSAGE" value="Executing default "fault" sequence"/>
<property name="ERROR_CODE" expression="get-property('ERROR_CODE')"/>
<property name="ERROR_MESSAGE" expression="get-property('ERROR_MESSAGE')"/>
</log>
</sequence>
</sequence>
<sequence name="main">
<log/>
Samples

The following samples demonstrate the use of the PayloadFactory mediator.

- Sample 17: Transforming / Replacing Message Content with PayloadFactory Mediator