Generating Synapse Code to be run in WSO2 Micro Integrator

1. **Overview**

WSO2 API Manager 4.2.0 is shipped with an integration runtime (Micro Integrator) with comprehensive enterprise integration capabilities. Therefore, you can now use WSO2 API Manager to develop complex integration services and expose them as managed APIs in an API marketplace. This allows you to enable API-led connectivity across your business using a single platform.

Integration Strategy¶

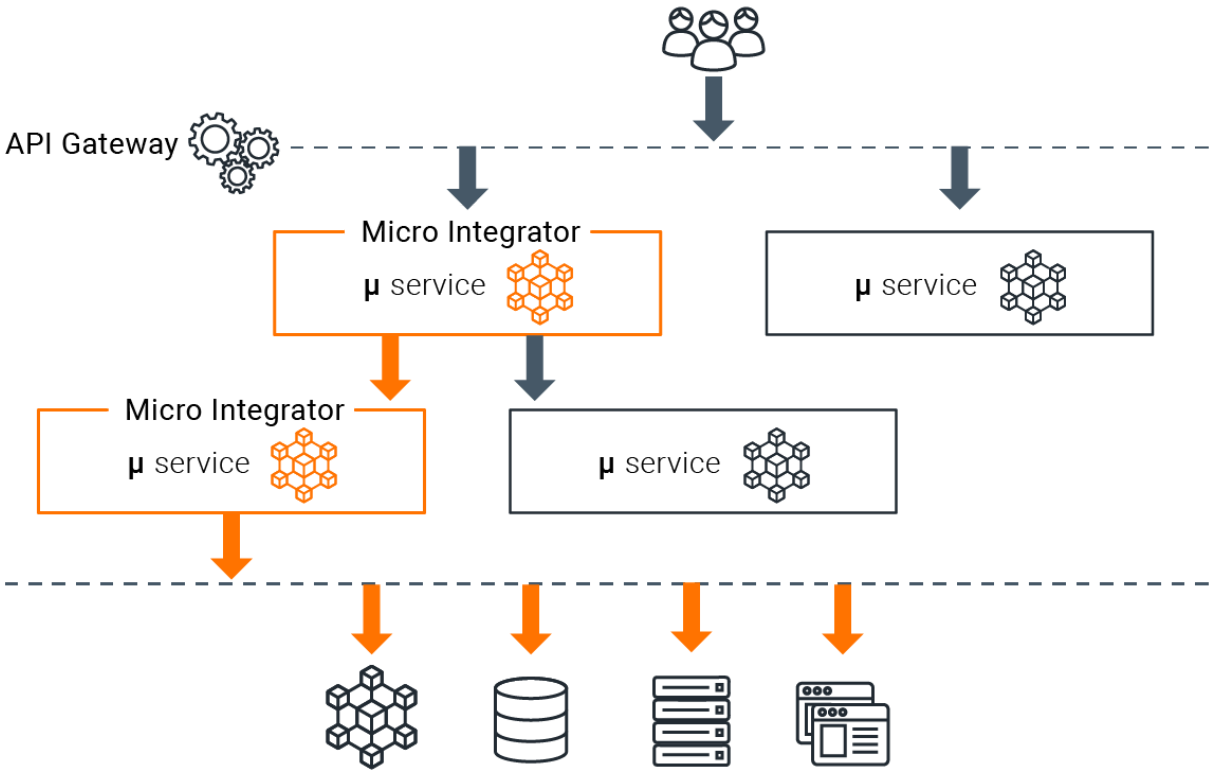
You can now leverage the integration capabilities as well as the API management capabilities of the product to implement any of the following integration strategies.

API-led Integration¶

WSO2 API Manager consists of an API management layer as well as an integration layer, which enables API-led integration through a single platform. The integration layer (Micro Integrator) is used for running the integration APIs, which are developed using WSO2 Integration Studio. The API management layer is used for converting the integration APIs into experience APIs and making them discoverable to developers.

Microservices Integration¶

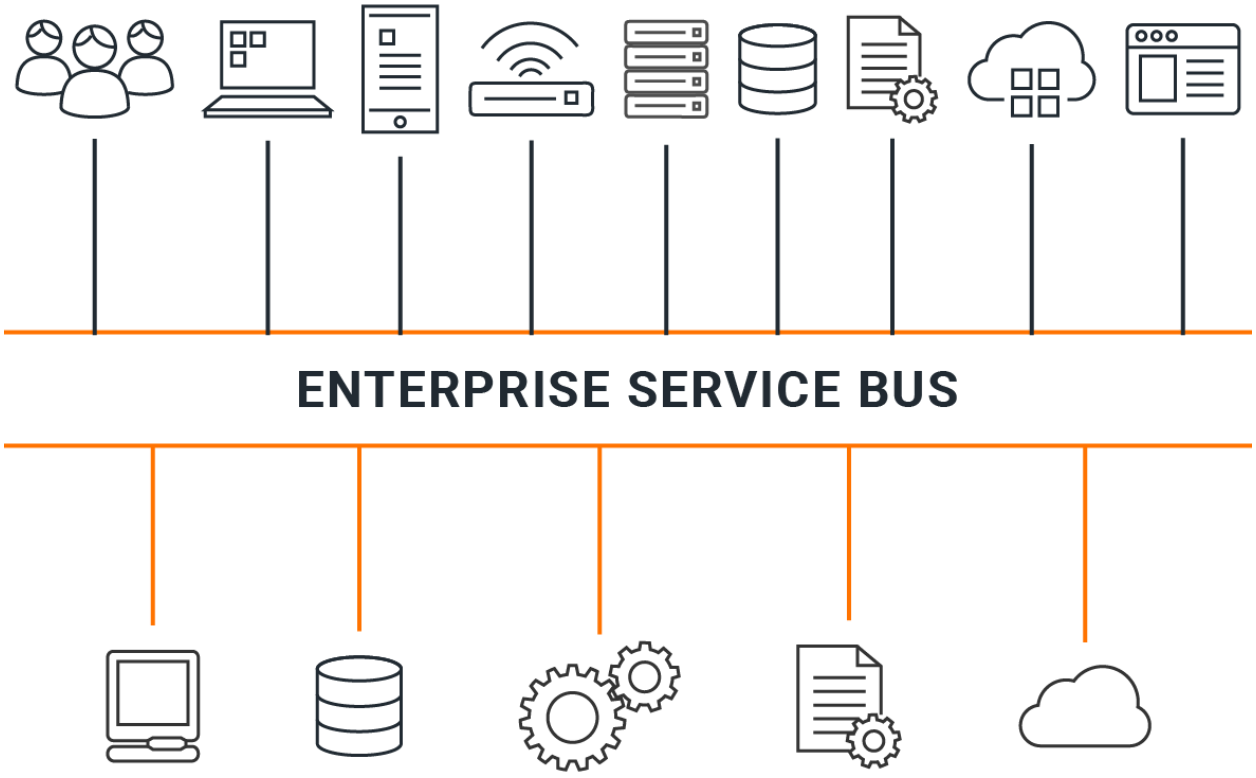
The Micro Integrator is lightweight and container friendly. This allows you to leverage the comprehensive enterprise messaging capabilities of the Micro Integrator in your decentralized, cloud-native integrations.



If your organization is running on a decentralized, cloud-native, integration architecture where microservices are used for integrating the various APIs, events, and systems, the Micro Integrator can easily function as your Integration microservices and API microservices.

Centralized Integration (Enterprise Service Bus)¶

At the heart of the Micro Integrator server is an event-driven, standards-based messaging engine (the Bus). This ESB supports message routing, message transformations, and other types of messaging use cases. If your organization uses an API-driven, centralized, integration architecture, the Micro Integrator can be used as the central integration layer that implements the message mediation logic connecting all the systems, data, events, APIs, etc. in your integration ecosystem.



2. Schema and Details for Synapse Configurations

REST API

Introduction¶

A REST API in WSO2 Micro Integrator is analogous to a web application deployed in the web container. Each API is anchored at a user-defined URL context, much like how a web application deployed in a servlet container is anchored at a fixed URL context. An API will only process requests that fall under its URL context. An API is made of one or more Resources, which are logical components of an API that can be accessed by making a particular type of HTTP call.

A REST API resource is used by the Micro Integrator to process messages before forwarding them to the relevant endpoint. Just as Proxy Services and Inbound Endpoints, the REST API resource uses mediators and sequences to define the mediation logic for processing messages. The API resource mediates incoming requests, forwards them to a specified endpoint, mediates the responses from the endpoint, and sends the responses back to the client that originally requested them. The In sequence handles incoming requests and sends them to the back-end service, and the Out sequence handles the responses from the back-end service and sends them to the requesting client. You can also define a fault sequence to handle any errors that may occur while mediating a message through a resource.

We can create an API resource to process defined HTTP request methods. Furthermore, you can configure REST endpoints in an API by directly specifying HTTP verbs (such as POST and GET), URI templates, and URL mappings. Alternatively, you can use the HTTP Endpoint to define REST endpoints using URI templates.

Properties¶

See the topics given below for the list of properties that can be configured when creating a REST API artifact.

REST API Properties (Required)¶

The following properties are required in order to create the REST API artifact.

Name	A unique name for the new API.
Context URL	An API in WSO2 Micro Integrator is analogous to a web application deployed in the Micro Integrator. Each API is anchored at a user-defined URL context, much like how a web application deployed in a servlet container is anchored at a fixed URL context. An API only processes requests that fall under its URL context. For example, if a particular API is anchored at the context `/test`, the API only handles HTTP requests with a URL path that starts with `/test`.

REST API Properties (Optional)¶

The following properties are optional properties you can configure when creating a REST API artifact.

Hostname	The Host at which the API is anchored. If you do not enter a value, `localhost` is considered the default hostname. If required, you can bind a given API to a user-defined hostname.
Port	The Port of the REST API. If you do not enter a value, `8280` is considered the default hostname. If required, you can bind a given API to a user-defined port number.
Version	The Micro Integrator identifies each API by its unique context name. If you introduce a version in the API context (e.g., /Service 1.0.0), you can update it when you upgrade the same API (e.g., /Service 1.0.1). Version your APIs as early as possible in the development cycle.

Path to Swagger Definition	**The path to a custom Swagger definition (YAML/JSON file) that is stored in a registry project in your workspace.** **Once this API is created and deployed in the Micro Integrator, users will be able to access this custom Swagger definition and not the default Swagger definition of the API. See the instructions on using Swagger documents for more information.**

REST API Resource Properties¶

When you creating a REST API artifact, you need to configure the API resource. Listed below are the properties you can configure when defining an API resource.

URI Style	**This allows us to restrict the type of HTTP requests processed by a particular resource. A resource can be associated with a user-defined URL mapping or a URI template.**

URI Template	**This property is enabled only if URI Template is specified as the URI Style.**
	
	A URI template represents a class of URIs using patterns and variables. When a resource is associated with a URI template, all requests that match the template will be processed by the resource. Some examples of valid URI templates are as follows:
	
	`/order/{orderId}/dictionary/{char}/{word}`
	
	The identifiers within curly braces are considered variables. For example, a URL that matches the template "/order/{orderId}" is as follows:
	
	`/order/A0001`
	
	In the above URL instance, the variable "orderId" has been assigned the value "A0001". Similarly, the following URL adheres to the template "/dictionary/{char}/{word}":
	
	`/dictionary/c/cat`

	In this case, the variable “char” has the value “c” and the variable “word” is given the value “cat”. It is possible to retrieve the exact values of the two variables using the "get-property" XPath extension of the Micro Integrator and prefixing the variable with "uri.var".
URL Mapping	**This property is enabled only if URL Mapping is specified as the URI Style.** **When a resource is defined with a URL mapping, only those requests that match the given pattern will be processed by the resource. There are three types of URL mappings:** • **Path mappings (`/test/`, `/foo/bar/`)** • **Extension mappings (`.jsp` , `.do`)** • **Exact mappings (`/test`, `/test/foo`)** **For example, if the URL mapping is “/foo/” and the HTTP method is “GET”, the resource will only process GET requests with a URL path that matches the pattern “/foo/”. Therefore, the following example requests will be processed by the resource:** `GET /test/foo/bar` `GET /test/foo/a?arg1=hello` **The following example requests would not be handled by this resource:** `GET /test/food/bar` **(URL pattern does not match)** `POST /test/foo/bar` **(HTTP verb does not match)**
Protocol	**Specify the HTTP methods that the resource should handle. This provides additional control over what requests are handled by a given resource.**

```
* limitations under the License.
-->

<xs:schema
       xmlns:xs="http://www.w3.org/2001/XMLSchema"
       elementFormDefault="qualified"
       targetNamespace="http://ws.apache.org/ns/synapse"
       xmlns="http://ws.apache.org/ns/synapse">

   <xs:include schemaLocation="misc/common.xsd" />
   <xs:include schemaLocation="misc/resource.xsd" />

   <xs:element name="api" type="API">
      <xs:annotation>
         <xs:documentation source="description">
            REST API element in the Synapse Configuration
         </xs:documentation>
      </xs:annotation>
   </xs:element>

   <xs:complexType name="API">
      <xs:annotation>
         <xs:documentation source="description">
            This is the element type representing the REST API in the Synapse Configuration
         </xs:documentation>
      </xs:annotation>
      <xs:sequence>
         <xs:element name="resource" type="APIResource" minOccurs="1"
maxOccurs="unbounded" />
         <xs:element name="handlers" minOccurs="0">
            <xs:complexType>
               <xs:sequence>
                  <xs:element name="handler" minOccurs="0" maxOccurs="unbounded">
                     <xs:complexType>
                        <xs:sequence>
                           <xs:element name="property" minOccurs="0" maxOccurs="unbounded">
                              <xs:complexType>
                                 <xs:attribute name="name" type="xs:string" use="required" />
                                 <xs:attribute name="value" type="xs:string" use="required" />
                              </xs:complexType>
                           </xs:element>
                        </xs:sequence>
                        <xs:attribute name="class" type="xs:string" use="required" />
                     </xs:complexType>
```

```
                    </xs:element>
                </xs:sequence>
            </xs:complexType>
        </xs:element>
    </xs:sequence>
    <xs:attributeGroup ref="monitoringAspect" />
    <xs:attribute name="name" type="xs:string" use="required" />
    <xs:attribute name="context" type="xs:string" use="required" />
    <xs:attribute name="hostname" type="xs:string" use="optional" />
    <xs:attribute name="port" type="xs:string" use="optional" />
    <xs:attribute name="version" type="xs:string" use="optional" />
    <xs:attribute name="version-type" use="optional">
        <xs:simpleType>
            <xs:restriction base="xs:string">
                <xs:enumeration value="context" />
                <xs:enumeration value="url" />
            </xs:restriction>
        </xs:simpleType>
    </xs:attribute>
    <xs:attribute name="publishSwagger" type="xs:string" use="optional" />
    <xs:attribute name="description" type="xs:string" use="optional" />
  </xs:complexType>

</xs:schema>
```

Endpoint

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!--
 ~  Licensed to the Apache Software Foundation (ASF) under one
 ~  or more contributor license agreements.  See the NOTICE file
 ~  distributed with this work for additional information
 ~  regarding copyright ownership.  The ASF licenses this file
 ~  to you under the Apache License, Version 2.0 (the
 ~  "License"); you may not use this file except in compliance
 ~  with the License.  You may obtain a copy of the License at
 ~
 ~   http://www.apache.org/licenses/LICENSE-2.0
 ~
 ~  Unless required by applicable law or agreed to in writing,
 ~  software distributed under the License is distributed on an
 ~   * "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
```

```
 ~  KIND, either express or implied.  See the License for the
 ~  specific language governing permissions and limitations
 ~  under the License.
 -->
<xs:schema
      xmlns:xs="http://www.w3.org/2001/XMLSchema"
      elementFormDefault="qualified"
      targetNamespace="http://ws.apache.org/ns/synapse"
      xmlns="http://ws.apache.org/ns/synapse">

   <xs:include schemaLocation="misc/common.xsd" />

   <xs:element name="endpoint" type="NamedEndpoint">
      <xs:annotation>
         <xs:documentation source="description">
            This is a named endpoint which will come on the top level synapse configuration
         </xs:documentation>
      </xs:annotation>
   </xs:element>

   <xs:complexType name="NamedEndpoint">
      <xs:complexContent>
         <xs:extension base="Endpoint">
            <xs:attribute name="name" type="xs:string" use="optional" />
         </xs:extension>
      </xs:complexContent>
   </xs:complexType>

   <xs:complexType name="Endpoint">
      <xs:annotation>
         <xs:documentation source="description">
            This is a representation of an endpoint
         </xs:documentation>
      </xs:annotation>
      <xs:sequence>
         <xs:choice>
            <xs:group ref="leafEndpoints" minOccurs="0" maxOccurs="1" />
            <xs:group ref="aggregatedEndpoints" minOccurs="0" maxOccurs="1" />
            <xs:group ref="otherEndpoints" minOccurs="0" maxOccurs="1" />
         </xs:choice>
         <xs:element name="property" minOccurs="0" maxOccurs="unbounded">
            <xs:complexType>
               <xs:complexContent>
                  <xs:extension base="mediatorProperty">
```

```
                    <xs:attribute name="scope" use="optional">
                        <xs:simpleType>
                            <xs:restriction base="xs:string">
                                <xs:enumeration value="transport" />
                                <xs:enumeration value="axis2" />
                                <xs:enumeration value="axis2-client" />
                            </xs:restriction>
                        </xs:simpleType>
                    </xs:attribute>
                </xs:extension>
            </xs:complexContent>
        </xs:complexType>
    </xs:element>
    <xs:element name="parameter" minOccurs="0" maxOccurs="unbounded">
        <xs:complexType>
            <xs:attribute name="name" type="xs:string" use="required" />
            <xs:attribute name="value" type="xs:string" use="required" />
        </xs:complexType>
    </xs:element>
    <xs:element name="description" type="xs:string" minOccurs="0" maxOccurs="1" />
  </xs:sequence>
  <xs:attribute name="key" type="xs:string" use="optional" />

  <!--key-expression-->

  <xs:attribute name="template" type="xs:string" use="optional" />
  <xs:attribute name="uri" type="xs:string" use="optional" />
</xs:complexType>

<!--done-->
<xs:group name="leafEndpoints">
  <xs:annotation>
    <xs:documentation source="description">
      This group represents the set of leaf level endpoints
    </xs:documentation>
  </xs:annotation>
  <xs:choice>
    <xs:element name="default" type="DefaultEndpoint" />

    <xs:element name="http">
      <xs:annotation>
        <xs:documentation source="description">
          HTTP endpoint representation of the synapse configuration
        </xs:documentation>
```

```
            </xs:annotation>
            <xs:complexType>
               <xs:choice maxOccurs="3">

                  <xs:group ref="QoS" minOccurs="0" maxOccurs="1" />
                  <xs:group ref="commonEndpointTags" minOccurs="0" maxOccurs="1" />

                  <xs:element name="authentication" minOccurs="0" maxOccurs="1">
                     <xs:complexType>
                        <xs:choice>
                           <xs:element name="oauth" minOccurs="0" maxOccurs="1">
                              <xs:complexType>
                                 <xs:choice>
                                    <xs:element name="authorizationCode" minOccurs="0"
maxOccurs="1">
                                       <xs:complexType>
                                          <xs:group ref="oauthParams" minOccurs="1"
maxOccurs="1" />
                                       </xs:complexType>
                                    </xs:element>
                                    <xs:element name="clientCredentials" minOccurs="0"
maxOccurs="1">
                                       <xs:complexType>
                                          <xs:group ref="oauthParams" minOccurs="1"
maxOccurs="1" />
                                       </xs:complexType>
                                    </xs:element>
                                    <xs:element name="passwordCredentials" minOccurs="0"
maxOccurs="1">
                                       <xs:complexType>
                                          <xs:sequence>
                                             <xs:group ref="oauthParams" minOccurs="1"
maxOccurs="1" />
                                             <xs:element name="username" type="xs:string"
minOccurs="1" maxOccurs="1"/>
                                             <xs:element name="password" type="xs:string"
minOccurs="1" maxOccurs="1"/>
                                          </xs:sequence>
                                       </xs:complexType>
                                    </xs:element>
                                 </xs:choice>
                              </xs:complexType>
                           </xs:element>
                           <xs:element name="basicAuth" minOccurs="0" maxOccurs="1">
```

```
                        <xs:complexType>
                           <xs:sequence>
                              <xs:element name="username" type="xs:string" minOccurs="1"
maxOccurs="1" />
                              <xs:element name="password" type="xs:string" minOccurs="1"
maxOccurs="1" />
                           </xs:sequence>
                        </xs:complexType>
                     </xs:element>
                  </xs:choice>
               </xs:complexType>
            </xs:element>
         </xs:choice>
         <xs:attribute name="uri-template" type="xs:string" use="optional" />
         <xs:attribute name="method">
            <xs:simpleType>
               <xs:restriction base="xs:string">
                  <xs:enumeration value="get" />
                  <xs:enumeration value="post" />
                  <xs:enumeration value="patch" />
                  <xs:enumeration value="delete" />
                  <xs:enumeration value="put" />
                  <xs:enumeration value="options" />
                  <xs:enumeration value="head" />
               </xs:restriction>
            </xs:simpleType>
         </xs:attribute>
         <xs:attributeGroup ref="monitoringAspect" />
      </xs:complexType>
   </xs:element>

   <xs:element name="address">
      <xs:complexType>
         <xs:complexContent>
            <xs:extension base="DefaultEndpoint">
               <xs:attribute name="uri" type="xs:anyURI" use="required" />
            </xs:extension>
         </xs:complexContent>
      </xs:complexType>
   </xs:element>
   <xs:element name="wsdl" type="WSDLEndpoint" />
 </xs:choice>
</xs:group>
```

```
   <xs:group name="aggregatedEndpoints">
      <xs:annotation>
         <xs:documentation source="description">
            This group represents the secondary endpoints like load balance and fail over
         </xs:documentation>
      </xs:annotation>
      <xs:choice>
         <xs:group ref="lbEndpoints" minOccurs="1" maxOccurs="1" />
         <xs:group ref="failoverEndpoints" minOccurs="1" maxOccurs="1" />
      </xs:choice>
   </xs:group>

   <!--done-->
   <xs:group name="lbEndpoints">
      <xs:annotation>
         <xs:documentation source="description">
            This group represents the secondary load balance endpoints
         </xs:documentation>
         <xs:appinfo>
            <sch:pattern name="Extended_all" xmlns:sch="http://purl.oclc.org/dsdl/schematron">
               <sch:rule context="endpoint">
                  <sch:assert test="count(session) = 1">You must have exactly 1 session
elements.</sch:assert>
                  <sch:assert test="count(loadbalance) = 1">You must have exactly 1 loadbalance
elements.</sch:assert>
               </sch:rule>
            </sch:pattern>
         </xs:appinfo>
      </xs:annotation>

      <xs:sequence>
         <xs:element name="loadbalance" minOccurs="1" maxOccurs="1">
            <xs:complexType>
               <xs:choice maxOccurs="unbounded">
                  <xs:element name="endpoint" minOccurs="1" maxOccurs="unbounded">
                     <xs:complexType>
                        <xs:choice>
                           <xs:group ref="leafEndpoints" minOccurs="0" maxOccurs="1" />
                           <xs:group ref="aggregatedEndpoints" minOccurs="0" maxOccurs="1" />
                        </xs:choice>
                        <xs:attribute name="name" type="xs:string" use="optional" />
                        <xs:attribute name="key" type="xs:string" use="optional" />
                     </xs:complexType>
                  </xs:element>
```

```
                <xs:element name="member" minOccurs="2" maxOccurs="unbounded">
                  <xs:complexType>
                    <xs:attribute name="hostName" type="xs:string" use="required" />
                    <xs:attribute name="httpPort" type="xs:string" use="optional" />
                    <xs:attribute name="httpsPort" type="xs:string" use="optional" />
                  </xs:complexType>
                </xs:element>
              </xs:choice>
              <xs:attribute name="algorithm" type="xs:string"
default="org.apache.synapse.endpoints.algorithms.LoadbalanceAlgorithm" use="optional" />
              <xs:attribute name="failover" type="xs:boolean" default="true" use="optional" />
              <xs:attribute name="policy" type="xs:string" use="optional" />
              <xs:attribute name="buildMessage" type="xs:boolean" use="optional" />
            </xs:complexType>
          </xs:element>
          <xs:element name="session" minOccurs="0" maxOccurs="1">
            <xs:complexType>
              <xs:sequence>
                <xs:element name="sessionTimeout" type="xs:long" minOccurs="0"
maxOccurs="1" />
              </xs:sequence>
              <xs:attribute name="type" use="optional" default="http">
                <xs:simpleType>
                  <xs:restriction base="xs:string">
                    <xs:enumeration value="simpleClientSession" />
                    <xs:enumeration value="http" />
                    <xs:enumeration value="soap" />
                  </xs:restriction>
                </xs:simpleType>
              </xs:attribute>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:group>

      <!--done-->
      <xs:group name="failoverEndpoints">
        <xs:annotation>
          <xs:documentation source="description">
            This group represents the secondary fail over endpoints
          </xs:documentation>
        </xs:annotation>
        <xs:sequence>
          <xs:element name="failover" minOccurs="1" maxOccurs="1">
```

```
            <xs:complexType>
               <xs:sequence>
                  <xs:element name="endpoint" minOccurs="1" maxOccurs="unbounded">
                     <xs:complexType>
                        <xs:choice>
                           <xs:group ref="leafEndpoints" minOccurs="0" maxOccurs="1" />
                           <xs:group ref="lbEndpoints" minOccurs="0" maxOccurs="1" />
                        </xs:choice>
                        <xs:attribute name="name" type="xs:string" use="optional" />
                        <xs:attribute name="key" type="xs:string" use="optional" />
                     </xs:complexType>
                  </xs:element>
               </xs:sequence>
               <xs:attribute name="dynamic" type="xs:boolean" use="optional" />
               <xs:attribute name="buildMessage" type="xs:boolean" use="optional" />
            </xs:complexType>
         </xs:element>
      </xs:sequence>
   </xs:group>

   <!--done-->
   <xs:group name="dynamicLB">
      <xs:annotation>
         <xs:documentation source="description">
            This group represents the dynamic loadbalance endpoint
         </xs:documentation>
      </xs:annotation>
      <xs:all>
         <xs:element name="dynamicLoadbalance" minOccurs="1" maxOccurs="1">
            <xs:complexType>
               <xs:sequence>
                  <xs:element name="membershipHandler" minOccurs="1" maxOccurs="1">
                     <xs:complexType>
                        <xs:sequence>
                           <xs:element name="property" minOccurs="0" maxOccurs="unbounded">
                              <xs:complexType>
                                 <xs:attribute name="name" type="xs:string" use="required" />
                                 <xs:attribute name="value" type="xs:string" use="required" />
                              </xs:complexType>
                           </xs:element>
                        </xs:sequence>
                        <xs:attribute name="class" type="xs:string" use="optional"
default="org.apache.synapse.core.axis2.Axis2LoadBalanceMembershipHandler" />
                     </xs:complexType>
```

```
                </xs:element>
             </xs:sequence>
             <xs:attribute name="algorithm" type="xs:string" use="optional"
default="org.apache.synapse.endpoints.algorithms.RoundRobin" />
             <xs:attribute name="failover" type="xs:boolean" use="optional" default="true" />
             <xs:attribute name="policy" type="xs:string" use="optional" default="roundRobin" />
          </xs:complexType>
       </xs:element>
    </xs:all>
 </xs:group>

 <xs:group name="otherEndpoints">
    <xs:sequence>
       <xs:element name="recipientlist">
          <xs:complexType>
             <xs:annotation>
                <xs:documentation source="description">
                   This contains the recipientlist endpoint of the synapse configuration
                </xs:documentation>
             </xs:annotation>
             <xs:sequence>
                <xs:element name="endpoint" minOccurs="1" maxOccurs="unbounded">
                   <xs:complexType>
                      <xs:choice minOccurs="1">
                         <xs:group ref="leafEndpoints" minOccurs="0" maxOccurs="1" />
                         <xs:group ref="aggregatedEndpoints" minOccurs="0" maxOccurs="1" />
                      </xs:choice>
                      <xs:attribute name="name" type="xs:string" use="optional" />
                      <xs:attribute name="key" type="xs:string" use="optional" />
                   </xs:complexType>
                </xs:element>
             </xs:sequence>
          </xs:complexType>
       </xs:element>
    </xs:sequence>
 </xs:group>

 <!--done-->
 <xs:complexType name="DefaultEndpoint">
    <xs:annotation>
       <xs:documentation source="description">
          This is the type declaration of the default endpoint
       </xs:documentation>
    </xs:annotation>
```

```
    <xs:sequence>
      <xs:group ref="QoS" minOccurs="0" maxOccurs="1" />
      <xs:group ref="commonEndpointTags" minOccurs="0" maxOccurs="1" />
    </xs:sequence>
    <xs:attributeGroup ref="commonEndpoint" />
</xs:complexType>

<!--done-->
<xs:complexType name="WSDLEndpoint">
    <xs:annotation>
      <xs:documentation source="description">
        This is the type declaration of the wsdl endpoint
      </xs:documentation>
    </xs:annotation>
    <xs:sequence>
      <xs:group ref="inlineWsdl" minOccurs="0" maxOccurs="1" />
      <xs:group ref="QoS" minOccurs="0" maxOccurs="1" />
      <xs:group ref="commonEndpointTags" minOccurs="0" maxOccurs="1" />
    </xs:sequence>
    <xs:attributeGroup ref="commonEndpoint" />
    <xs:attribute name="uri" type="xs:anyURI" use="required" />
    <xs:attribute name="service" type="xs:string" use="required" />
    <xs:attribute name="port" type="xs:string" use="required" />
</xs:complexType>

<!--done-->
<xs:attributeGroup name="commonEndpoint">
    <xs:annotation>
      <xs:documentation source="description">
        This group of attributes represents the common endpoint attribute set
      </xs:documentation>
    </xs:annotation>
    <xs:attribute name="format" use="optional">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="soap11" />
          <xs:enumeration value="soap12" />
          <xs:enumeration value="pox" />
          <xs:enumeration value="get" />
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
    <xs:attribute name="optimize" use="optional">
      <xs:simpleType>
```

```
            <xs:restriction base="xs:string">
               <xs:enumeration value="mtom" />
               <xs:enumeration value="swa" />
            </xs:restriction>
         </xs:simpleType>
      </xs:attribute>
      <xs:attribute name="encoding" type="xs:string" use="optional" />
      <xs:attributeGroup ref="monitoringAspect" />
   </xs:attributeGroup>

   <!--done-->
   <xs:group name="commonEndpointTags">
      <xs:sequence>
         <xs:element name="timeout" minOccurs="0" maxOccurs="1">
            <xs:complexType mixed="true">
               <xs:all>
                  <xs:element name="duration" minOccurs="0" maxOccurs="1" type="xs:string" />
                  <xs:element name="responseAction" type="xs:string" minOccurs="0"
maxOccurs="1" default="discard" />
               </xs:all>
            </xs:complexType>
         </xs:element>
         <xs:element name="suspendOnFailure" minOccurs="0" maxOccurs="1">
            <xs:complexType>
               <xs:all>
                  <xs:element name="errorCodes" minOccurs="0" maxOccurs="1" type="xs:string"
/>
                  <xs:element name="initialDuration" minOccurs="0" maxOccurs="1"
type="xs:string" />
                  <xs:element name="progressionFactor" minOccurs="0" maxOccurs="1"
type="xs:string" />
                  <xs:element name="maximumDuration" minOccurs="0" maxOccurs="1"
type="xs:string" />
               </xs:all>
            </xs:complexType>
         </xs:element>
         <xs:element name="markForSuspension" minOccurs="0" maxOccurs="1">
            <xs:complexType>
               <xs:all>
                  <xs:element name="errorCodes" minOccurs="0" maxOccurs="1" type="xs:string"
/>
                  <xs:element name="retriesBeforeSuspension" minOccurs="0" maxOccurs="1"
type="xs:string" />
```

```
                <xs:element name="retryDelay" minOccurs="0" maxOccurs="1" type="xs:string"
/>
              </xs:all>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
    </xs:group>

    <xs:group name="oauthParams">
      <xs:sequence>
        <xs:element name="clientId" type="xs:string" minOccurs="1" maxOccurs="1" />
        <xs:element name="clientSecret" type="xs:string" minOccurs="1" maxOccurs="1" />
        <xs:element name="refreshToken" type="xs:string" minOccurs="0" maxOccurs="1" />
        <xs:element name="tokenUrl" type="xs:string" minOccurs="1" maxOccurs="1" />
        <xs:element name="requestParameters" minOccurs="0" maxOccurs="1">
          <xs:complexType>
            <xs:sequence>
              <xs:element ref="parameter" minOccurs="0" maxOccurs="unbounded" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="authMode" type="xs:string" minOccurs="1" maxOccurs="1" />
      </xs:sequence>
    </xs:group>

    <xs:group name="QoS">
      <xs:sequence>
        <xs:element name="enableSec" minOccurs="0" maxOccurs="1">
          <xs:complexType>
            <xs:attribute name="inboundPolicy" type="xs:string" use="optional" />
            <xs:attribute name="outboundPolicy" type="xs:string" use="optional" />
          </xs:complexType>
        </xs:element>
        <xs:element name="enableRM" minOccurs="0" maxOccurs="1">
          <xs:complexType>
            <xs:attribute name="policy" type="xs:string" use="required" />
          </xs:complexType>
        </xs:element>
        <xs:element name="enableAddressing" minOccurs="0" maxOccurs="1">
          <xs:complexType>
            <xs:attribute name="version" use="optional">
              <xs:simpleType>
                <xs:restriction base="xs:string">
                  <xs:enumeration value="final" />
```

```
                        <xs:enumeration value="discard" />
                        <xs:enumeration value="fault" />
                    </xs:restriction>
                </xs:simpleType>
            </xs:attribute>
            <xs:attribute name="separateListener" type="xs:boolean" use="optional" />
        </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:group>

</xs:schema>
```

Mediators

There are mediators which can help build the Integration flow. The schema for mediators are as follows;

Call Mediator

The Call mediator is used to send messages out of the Micro Integrator to an endpoint. You can invoke services either in blocking or non-blocking manner.

When you invoke a service in non-blocking mode, the underlying worker thread returns without waiting for the response. In blocking mode, the underlying worker thread gets blocked and waits for the response after sending the request to the endpoint. Call mediator in blocking mode is very much similar to the Callout mediator.

In both blocking and non-blocking modes, Call mediator behaves in a synchronous manner. Hence, mediation pauses after the service invocation, and resumes from the next mediator in the sequence when the response is received. Call mediator allows you to create your configuration independent from the underlying architecture.

Non-blocking mode of the Call mediator leverages the non-blocking transports for better performance. Therefore, it is recommended to use it in non-blocking mode as much as possible. However, there are scenarios where you need to use the blocking mode. For example, when you implement a scenario related to JMS transactions, it is vital to use the underlying threads in blocking mode.

You can obtain the service endpoint for the Call mediator as follows:

- **Pick from message-level information**
- **Pick from a pre-defined endpoint**

If you do not specify an endpoint, the Call mediator tries to send the message using the `WSA:TO` address of the message. If you specify an endpoint, the Call mediator sends the message based on the specified endpoint.

The endpoint type can be Leaf Endpoint (i.e. Address/WSDL/Default/HTTP) or Group Endpoint (i.e. Failover/Load balance/Recipient list). Group Endpoint is only supported in non-blocking mode.

Info

The Call mediator is a content-unaware mediator.

Enabling mutual SSL in the blocking mode¶

When using the Call mediator in the blocking mode (blocking=true), enable the mediator to handle mutual SSL by adding the following JVM settings to the `MI_HOME/bin/micro-integrator.sh` file:

```
-Djavax.net.ssl.keyStore="$CARBON_HOME/repository/resources/security/wso2carbon.jks" \
-Djavax.net.ssl.keyStorePassword="wso2carbon" \
-Djavax.net.ssl.keyPassword="wso2carbon" \

-Drampart.axiom.parser.pool=false \
```

Syntax¶

```
<call [blocking="true"] />
```

Note

The call mediator in blocking mode (blocking=true) builds the message from the response payload. If a response message is not expected by the client, you can set the OUT_ONLY property before the call mediator to avoid building the response payload. You may set the allowEmptyBodyForHttp2xx system property to true if you want to receive a response without the body for the following HTTP status codes in blocking mode: 200, 201, and 202.

```
<property name="OUT_ONLY" value="true"/>
```

If the message is to be sent to one or more endpoints, use the following syntax:

```
<call [blocking="true"]>
   (endpointref | endpoint)+

</call>
```

- The `endpointref` token refers to the following:
- `<endpoint key="name"/>`
- The `endpoint` token refers to an anonymous endpoint definition.

Configuration¶

Select one of the following options to define the endpoint to which the message should be delivered.

None	**Select this option if you do not want to provide an endpoint. The Call mediator will send the message using its `wsa:to` address.**
Define Inline	**If this is selected, the endpoint to which the message should be sent can be included within the Call mediator configuration. Click Add to add the required endpoint. For more information on Adding an endpoint, see Adding an Endpoint .**
Pick From Registry	**If this is selected, the message can be sent to a pre-defined endpoint, which is currently saved as a resource in the registry. Click either Configuration Registry or Governance Registry as relevant to select the required endpoint from the resource tree.**
XPath	**If this is selected, the endpoint to which the message should be sent will be derived via an XPath expression. You are required to enter the relevant XPath expression in the text field that appears when this option is selected.** **Note:** **You can click NameSpaces to add namespaces if you are providing an expression. Then the Namespace Editor panel would appear where you can provide any number of namespace prefixes and URLs used in the XPath expression.**
Blocking	**If set to `true`, you can use the call mediator in blocking mode.**

Example¶

Example 1 - Service orchestration¶

In this example, the Call mediator invokes a backend service. An Enrich mediator stores the response received for that service invocation.

The Filter Mediator added after the Call mediator carries out a filter to determine whether the first call has been successful. If it is successful, second backend service is invoked. The payload of the request to the second backend is the response of the first service invocation .

After a successful second backend service invocation, response of the first service is retrieved by the Enrich mediator from the property where it was formerly stored. This response is sent to the client by the Respond mediator .

If it is not successful, a custom JSON error message is sent with HTTP 500. If the first call itself is not successful, the output is just sent back with the relevant error code.

```
<target>
      <inSequence>
         <log/>
         <call>
            <endpoint>
               <http method="get"
uri-template="http://192.168.1.10:8088/mockaxis2service"/>
            </endpoint>
         </call>
         <enrich>
            <source type="body" clone="true"/>
            <target type="property" action="child"
property="body_of_first_call"/>
         </enrich>
         <filter source="get-property('axis2', 'HTTP_SC')"
regex="200">
            <then>
               <log level="custom">
                  <property name="switchlog" value="Case: first call
successful"/>
               </log>
               <call>
                  <endpoint>
                     <http method="get"
uri-template="http://localhost:8080/MockService1"/>
                  </endpoint>
               </call>
```

```
            <filter source="get-property('axis2', 'HTTP_SC')"
regex="200">
               <then>
                  <log level="custom">
                     <property name="switchlog" value="Case: second
call successful"/>
                  </log>
                  <enrich>
                     <source type="property" clone="true"
property="body_of_first_call"/>
                     <target type="body"/>
                  </enrich>
                  <respond/>
               </then>
               <else>
                  <log level="custom">
                     <property name="switchlog" value="Case: second
call unsuccessful"/>
                  </log>
                  <property name="HTTP_SC" value="500"
scope="axis2"/>
                  <payloadFactory media-type="json">
                     <format>{ "status": "ERROR!"}</format>
                     <args/>
                  </payloadFactory>
                  <respond/>
               </else>
            </filter>
         </then>
         <else>
            <log level="custom">
               <property name="switchlog" value="Case: first call
unsuccessful"/>
            </log>
            <respond/>
         </else>
      </filter>
   </inSequence>
```

```
   </target>
```

Example 2 - Continuing mediation without waiting for responses¶

In this example, the message will be cloned by the Clone Mediator and sent via the Call mediator. The Drop mediator drops the response so that no further mediation is carried out for the cloned message. However, since the `continueParent` attribute of the Clone mediator is set to `true` , the original message is mediated in parallel. Therefore, the Log Mediator at the end of the configuration will log the `After call mediator` log message without waiting for the Call mediator response.

```
...
<log level="full"/>
<clone continueParent="true">
<target>
<sequence>
<call>
<endpoint>
<address uri="http://localhost:8080/echoString"/>
</endpoint>
</call>
<drop/>
</sequence>
</target>
</clone>
<log level="custom">
<property name="MESSAGE" value="After call mediator"/>
</log>

...
```

Example 3 - Call mediator in blocking mode¶

In the following sample configuration, the Header Mediator is used to add the action, the PayloadFactory Mediator is used to store the the request message and the Call mediator is used to invoke a backend service. You will see that the payload of the request and header action are sent to the backend. After successful backend service invocation, you will see that the response of the service is retrieved by the Micro Integrator and sent to the client as the response using the Respond Mediator.

```
<target>
   <inSequence>
```

```
      <header name="Action" value="urn:getQuote" />
      <payloadFactory media-type="xml">
         <format>
            <m0:getQuote xmlns:m0="http://services.samples">
               <m0:request>
                  <m0:symbol>WSO2</m0:symbol>
               </m0:request>
            </m0:getQuote>
         </format>
         <args />
      </payloadFactory>
      <call blocking="true">
         <endpoint>
            <address
uri="http://localhost:9000/services/SimpleStockQuoteService" />
         </endpoint>
      </call>
      <respond />
   </inSequence>

</target>
```

Example 4 - Receiving response headers in blocking mode¶

If you want to receive the response message headers, when you use the Call mediator in blocking mode, add the `BLOCKING_SENDER_PRESERVE_REQ_HEADERS` property within the proxy service, or in a sequence as shown in the sample proxy configuration below.

Info

Set the value of the `BLOCKING_SENDER_PRESERVE_REQ_HEADERS` property to `false` to receive the response message headers. If you set it to `true`, you cannot get the response headers, but the request headers will be preserved.

```
<proxy xmlns="http://ws.apache.org/ns/synapse"
       name="sample"
       transports="https"
       statistics="enable"
       trace="enable"
```

```
        startOnLoad="true">
   <target>
      <inSequence>
         <property name="FORCE_ERROR_ON_SOAP_FAULT"
                   value="true"
                   scope="default"
                   type="STRING"/>
         <property name="HTTP_METHOD" value="POST" scope="axis2"
type="STRING"/>
         <property name="messageType" value="text/xml" scope="axis2"
type="STRING"/>
         <property name="BLOCKING_SENDER_PRESERVE_REQ_HEADERS"
value="false"/>
         <call blocking="true">
            <endpoint>
               <address uri="https://localhost:8243/services/sampleBE"
                        trace="enable"
                        statistics="enable"/>
            </endpoint>
         </call>

      </inSequence>
      <outSequence/>
   </target>
   <description/>

</proxy>
```

```
*
* Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
* limitations under the License.
-->

<xs:schema
        xmlns:xs="http://www.w3.org/2001/XMLSchema"
        elementFormDefault="qualified"
        targetNamespace="http://ws.apache.org/ns/synapse"
        xmlns="http://ws.apache.org/ns/synapse">

    <xs:include schemaLocation="../../endpoint.xsd" />

    <xs:element name="call">
        <xs:annotation>
            <xs:documentation source="description">
                Invokes an external service (in blocking or non-blocking mode). The service response
is received by the
                next mediator in the mediation flow.
            </xs:documentation>
        </xs:annotation>
        <xs:complexType>
            <xs:all>
                <xs:element name="source" minOccurs="0" maxOccurs="1">

                    <xs:complexType mixed="true">
                        <xs:sequence>
                            <xs:any minOccurs="0" processContents="skip" />
                        </xs:sequence>
                        <xs:attribute name="contentType" type="xs:string" use="required" />
                        <xs:attribute name="type" use="required">
                            <xs:simpleType>
                                <xs:restriction base="xs:string">
                                    <xs:enumeration value="custom" />
                                    <xs:enumeration value="inline" />
                                    <xs:enumeration value="property" />
                                    <xs:enumeration value="body" />
                                </xs:restriction>
                            </xs:simpleType>
                        </xs:attribute>
                    </xs:complexType>
```

```
            </xs:element>
            <xs:element name="target" minOccurs="0" maxOccurs="1">
               <xs:complexType mixed="true">
                  <xs:attribute name="type" use="required">
                     <xs:simpleType>
                        <xs:restriction base="xs:string">
                           <xs:enumeration value="property" />
                           <xs:enumeration value="body" />
                        </xs:restriction>
                     </xs:simpleType>
                  </xs:attribute>
               </xs:complexType>
            </xs:element>
            <xs:element name="endpoint" type="NamedEndpoint" minOccurs="1" maxOccurs="1"
/>
         </xs:all>
         <xs:attribute name="blocking" type="xs:boolean" use="optional" default="true" />
         <xs:attribute name="description" type="xs:string" use="optional" />
      </xs:complexType>
   </xs:element>

</xs:schema>
```

Log Mediator

The Log mediator is used to log mediated messages. For more information on logging, see Monitoring Logs.

Info

The Log mediator is a conditionally content aware mediator.

Syntax¶

The log token refers to a `<log>` element, which may be used to log messages being mediated.

```
<log [level="string"] [separator="string"]>
   <property name="string" (value="literal" |
expression="[XPath|json-eval(JSON Path)]")/>*
```

```
</log>
```

Configuration¶

The general parameters available to configure the Log mediator are as follows.

Log Category	**This parameter is used to specify the log category. Possible values are as follows. Following log levels correspond to the ESB profile service level logs.** • **TRACE - This designates fine-grained informational events than the DEBUG.** • **DEBUG - This designates fine-grained informational events that are most useful to debug an application.** • **INFO - This designates informational messages that highlight the progress of the application at coarse-grained level.** • **WARN - This designates potentially harmful situations.** • **ERROR - This designates error events that might still allow the application to continue running.** • **FATAL - This designate s very severe error events that will presumably lead the application to abort.**

Log Level	**This parameter is used to specify the log level. The possible values are as follows.** • **Full : If this is selected, all the standard headers logged at the Simple level as well as the full payload of the message will be logged. This log level causes the message content to be parsed and hence incurs a performance overhead.** • **Simple : If this is selected, the standard headers (i.e. `To` , `From` , `WSAction` , `SOAPAction` , `ReplyTo` , and `MessageID`) will be logged.** • **Headers : If this is selected, all the SOAP header blocks are logged.** • **Custom : If this is selected, only the properties added to the Log mediator configuration will be logged.** **The properties included in the Log mediator configuration will be logged regardless of the log level selected.**

Log Separator	**This parameter is used to specify a value to be used in the log to separate attributes. The , comma is default.** **Use only the Source View to add a tab (i.e., by defining the `separator=" "` parameter in the syntax) or a new line (i.e., by defining the `separator="
"` parameter in the syntax) as the Log Separator , since the Design View does not support this.**

The parameters available to configure a property are as follows:

Property Name	**The name of the property to be logged.**
Property Value	**The possible values for this parameter are as follows:** • **Value: If this is selected, a static value would be considered as the property value and this value should be entered in the Value/Expression parameter.** • **Expression: If this is selected, the property value will be determined during mediation by evaluating an expression. This expression should be entered in the Value/Expression parameter.**
Value/Expression	**This parameter is used to enter a status value as the property value, or to enter an expression to evaluate the property value based on the what you entered for the Property Value parameter. When specifying a JSONPath, use the format `json-eval(<JSON_PATH>)` , such as `json-eval(getQuote.request.symbol)`.**
Action	**This parameter allows the property to be deleted.**

Examples¶

Using Full log¶

In this example, everything is logged including the complete SOAP message.

```
<log level="full" xmlns="http://ws.apache.org/ns/synapse"/>
```

Using Custom logs¶

In this example, the log level is `custom` . A property with an XPath expression which is used to get a stock price from a message is included. This results in logging the stock, price which is a dynamic value.

```
<log level="custom" xmlns="http://ws.apache.org/ns/synapse">
    <property name="text"
              expression="fn:concat('Stock price -
',get-property('stock_price'))"/>

</log>
```

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!--
 ~  Licensed to the Apache Software Foundation (ASF) under one
 ~  or more contributor license agreements.  See the NOTICE file
 ~  distributed with this work for additional information
 ~  regarding copyright ownership.  The ASF licenses this file
 ~  to you under the Apache License, Version 2.0 (the
 ~  "License"); you may not use this file except in compliance
 ~  with the License.  You may obtain a copy of the License at
 ~
 ~   http://www.apache.org/licenses/LICENSE-2.0
 ~
 ~  Unless required by applicable law or agreed to in writing,
 ~  software distributed under the License is distributed on an
 ~   * "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
 ~  KIND, either express or implied.  See the License for the
 ~  specific language governing permissions and limitations
 ~  under the License.
 -->

<xs:schema
      xmlns:xs="http://www.w3.org/2001/XMLSchema"
      elementFormDefault="qualified"
      targetNamespace="http://ws.apache.org/ns/synapse"
      xmlns="http://ws.apache.org/ns/synapse">

   <xs:include schemaLocation="../../misc/common.xsd" />
```

```
   <xs:element name="log">
      <xs:annotation>
         <xs:documentation source="description">
            Generates logs for messages (mediated by a sequence or proxy service). By default,
only the minimum
            details are logged. If required, you can log the full message payload, headers, and
even custom
            user-defined properties.
         </xs:documentation>
      </xs:annotation>
      <xs:complexType>
         <xs:sequence>
            <xs:element name="property" type="mediatorProperty" minOccurs="0"
maxOccurs="unbounded" />
         </xs:sequence>
         <xs:attribute name="level" type="logLevel" use="optional" default="simple" />
         <xs:attribute name="separator" type="xs:string" use="optional" default=", " />
         <xs:attribute name="category" type="logCategory" use="optional" default="INFO" />
         <xs:attribute name="description" type="xs:string" use="optional" />
      </xs:complexType>
   </xs:element>

   <xs:simpleType name="logLevel">
      <xs:annotation>
         <xs:documentation source="description">
            This simple type represents the possible values for
            the header mediator action attribute
         </xs:documentation>
      </xs:annotation>
      <xs:restriction base="xs:string">
         <xs:enumeration value="simple" />
         <xs:enumeration value="headers" />
         <xs:enumeration value="full" />
         <xs:enumeration value="custom" />
      </xs:restriction>
   </xs:simpleType>

   <xs:simpleType name="logCategory">
      <xs:annotation>
         <xs:documentation source="description">
            This simple type represents the possible values for
            the header mediator action attribute
         </xs:documentation>
      </xs:annotation>
```

```
      <xs:restriction base="xs:string">
         <xs:enumeration value="INFO" />
         <xs:enumeration value="FATAL" />
         <xs:enumeration value="ERROR" />
         <xs:enumeration value="WARN" />
         <xs:enumeration value="DEBUG" />
         <xs:enumeration value="TRACE" />
      </xs:restriction>
   </xs:simpleType>

</xs:schema>
```

Property Mediator

The Property Mediator has no direct impact on the message, but rather on the message context flowing through Synapse. You can retrieve the properties set on a message later through the Synapse XPath Variables or the `get-property()` extension function. A property can have a defined scope for which it is valid. If a property has no defined scope, it defaults to the Synapse message context scope. Using the property element with the action specified as `remove`, you can remove any existing message context properties.

Info

The Property mediator is a conditionally content aware mediator.

Syntax¶

```
<property name="string" [action=set|remove] [type="string"]
(value="literal" | expression="xpath")
[scope=default|transport|axis2|axis2-client] [pattern="regex"
[group="integer"]]>
    <xml-element/>?

</property>
```

Configuration¶

The parameters available for configuring the Property mediator are as follows:

Name

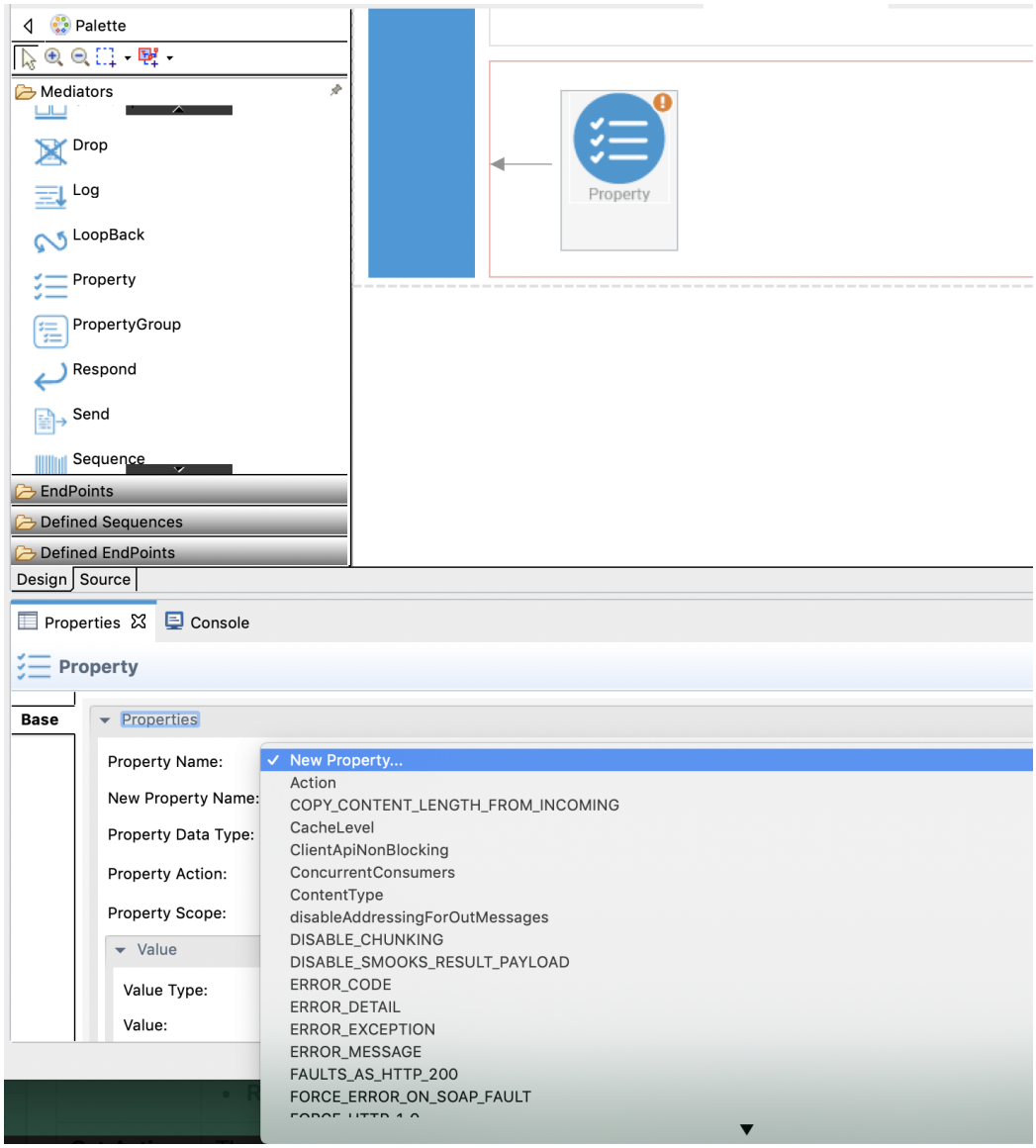
A name for the property.

You can provide a static value or a dynamic value for the property name. A dynamic property name can be retrieved by using an XPath function. You can use any of the XPath functions that you use for the property value or property expression.

Note that the XPath function should be contained within curly brackets ({}) as well as double quotations (""). See the examples given below.

- `property name="{get-property('propertyName')}"`
- `property name="{$ctx:propertyName}"`
- `property name="{json-eval({$ctx:propertyName})}"`

For names of the generic properties that come by default, see Generic Properties . You can select them from the drop-down list if you are adding the Property Mediator as shown below.



Action

The action to be performed for the property.

- **Set: If this is selected, the property will be set in the message context.**
- **Remove: If this is selected, the property will be removed from the message context.**

Set Action As	**The possible values for this parameter are as follows:** • **Value: If this is selected, a static value would be considered as the property value and this value should be entered in the Value parameter.** • **Expression: If this is selected, the property value will be determined during mediation by evaluating an expression. This expression should be entered in the Expression parameter.**
Type	**The data type for the property. Property mediator will handle the property as a property of selected type. Available values are as follows:** • **STRING** • **INTEGER** • **BOOLEAN** • **DOUBLE** • **FLOAT** • **LONG** • **SHORT** • **OM** **String is the default type.** **The `OM` type is used to set xml property values on the message context. This is useful when the expression associated with the property mediator evaluates to an XML node during mediation. When the `OM` type is used, the XML is converted to an AXIOM OMElement before it is assigned to a property.**
Value	**If the Value option is selected for the Set Action As parameter, the property value should be entered as a constant in this parameter.**
Expression	**If the Expression option is selected for the Set Action As parameter, the expression which determines the property value should be entered in this parameter. This expression can be an XPath expression or a JSONPath expression.** **When specifying a JSONPath, use the format `json-eval(<JSON_PATH>)` , such as `json-eval(getQuote.request.symbol)`. In both XPath and JSONPath expressions, you can return the value of another property by calling `get-property(property-name)`. For example, you might create a property called `JSON_PATH` of which the value is `json-eval(pizza.toppings)` , and then you could create another property called `SON_PRINT`**

	of which the value is `get-property('JSON_PATH')`, allowing you to use the value of the JSON_PATH property in the JSON_PRINT property.
Pattern	This parameter is used to enter a regular expression that will be evaluated against the value of the property or result of the XPath/JSON Path expression.
Group	The number (index) of the matching item evaluated using the regular expression entered in the Pattern parameter.
Scope	The scope at which the property will be set or removed from. Possible values are as follows. • Synapse: This is the default scope. The properties set in this scope last as long as the transaction (request-response) exists. • Transport: The properties set in this scope will be considered transport headers. For example, if it is required to send an HTTP header named 'CustomHeader' with an outgoing request, you can use the property mediator configuration with this scope. • Axis2: Properties set in this scope have a shorter life span than those set in the Synapse scope. They are mainly used for passing parameters to the underlying Axis2 engine • axis2-client: This is similar to the Synapse scope. The difference between the two scopes is that the axis2-client scope can be accessed inside the mediate() method of a mediator via a custom mediator created using the Class mediator. • Operation: This scope is used to retrieve a property in the operation context level. • Registry: This scope is used to retrieve properties within the registry . • System: This scope is used to retrieve Java system properties. • Environment: This scope is used to retrieve environment variables ('env'). • File: This scope is used to retrieve properties defined in the `file.properties` configuration file ('file'). For a detailed explanation of each scope, see Accessing Properties with XPath.

Note

There are predefined XPath variables (such as `$ctx`) that you can directly use in the Synapse configuration, instead of using the synapse:get-property() function. These XPath variables get properties of various scopes and have better performance than the `get-property()` function, which can have much lower performance because it does a registry lookup. These XPath variables get properties of various scopes. For more information on these XPath variables, see Accessing Properties with XPath.

Examples¶

Setting and logging and property¶

In this example, we are setting the property symbol and later we can log it using the Log Mediator.

```
<property name="symbol" expression="fn:concat('Normal Stock - ',
//m0:getQuote/m0:request/m0:symbol)"
xmlns:m0="http://services.samples/xsd"/>

<log level="custom">
    <property name="symbol" expression="$ctx:symbol"/>

</log>
```

Sending a fault message based on the Accept http header¶

In this configuration, a response is sent to the client based on the `Accept` header. The PayloadFactory mediator transforms the message contents. Then a Property mediator sets the message type based on the `Accept` header using the `$ctx:accept` expression. The message is then sent back to the client via the Respond mediator.

```
<payloadFactory media-type="xml">
    <format>
        <m:getQuote xmlns:m="http://services.samples">
            <m:request>
                <m:symbol>Error</m:symbol>
            </m:request>
        </m:getQuote>
    </format>
</payloadFactory>
<property name="messageType" expression="$ctx:accept" scope="axis2" />

<respond/>
```

Reading a property stored in the Registry¶

You can read a property that is stored in the Registry by using the `get-property()` method in your Synapse configuration. For example, the following Synapse configuration retrieves the `abc` property of the collection `gov:/data/xml/collectionx`, and stores it in the `regProperty` property.

```
<property name="regProperty" expression="get-property('registry',
'gov:/data/xml/collectionx@abc')"/>
```

Info

You can use the following syntaxes to read properties or resources stored in the `gov` or `conf` Registries. When specifying the path to the resource, do not give the absolute path. Instead, use the `gov` or `conf` prefixes.

Reading a property stored under a collection¶

- `get-property('registry','gov:<path to resource from governance>@<propertyname>')`
- `get-property('registry','conf:<path to resource from config>@<propertyname>')`

Reading a property stored under a resource¶

- `get-property('registry','gov:<path to resource from governance>/@<propertyname>')`
- `get-property('registry','conf:<path to resource from config>/@<propertyname>')`

Reading an XML resource¶

- `get-property('registry','gov:<path to resource from governance>')`
- `get-property('registry','conf:<path to resource from config>')`

Reading a file stored in the Registry¶

Following is an example, in which you read an XML file that is stored in the registry using XPath, to retrieve a value from it. Assume you have the following XML file stored in the Registry (i.e., `gov:/test.xml`).

test.xml

```
<root>
  <book>A Song of Ice and Fire</book>
  <author>George R. R. Martin</author>

</root>
```

Your Synapse configuration should be as follows. This uses XPath to read XML.

reg_xpath.xml

```
<property name="xmlFile"
expression="get-property('registry','gov:/test.xml')" scope="default"
type="OM"></property>
<log level="custom">
    <property name="Book_Name"
expression="$ctx:xmlFile//book"></property>

</log>
```

Your output log will look like this.

```
[2015-09-21 16:01:28,750]  INFO - LogMediator Book_Name = A Song of
Ice and Fire
```

Reading SOAP headers¶

SOAP headers provide information about the message, such as the To and From values. You can use the `get-property()` function of the Property mediator to retrieve these headers. You can also add Custom SOAP Headers using the PayloadFactory mediator and the Script Mediator.

To¶

Header Name	**To**
Possible Values	**Any URI**
Description	**The To header of the message.**
Example	**get-property("To")**

From¶

Header Name	**From**
Possible Values	**Any URI**
Description	**The From header of the message.**
Example	**get-property("From")**

Action¶

Header Name	**Action**
Possible Values	**Any URI**
Description	**The SOAPAction header of the message.**
Example	**get-property("Action")**

ReplyTo¶

Header Name	**ReplyTo**
Possible Values	**Any URI**

Description	**The ReplyTo header of the message.**
Example	**<header name="ReplyTo" action="remove"/>**

MessageID¶

Header Name	**MessageID**
Possible Values	**UUID**
Description	**The unique message ID of the message. It is not recommended to make alterations to this property of a message.**
Example	**get-property("MessageID")**

RelatesTo¶

Header Name	**RelatesTo**
Possible Values	**UUID**
Description	**The unique ID of the request to which the current message is related. It is not recommended to make changes.**
Example	**get-property("RelatesTo")**

FaultTo¶

Header Name	**FaultTo**
Possible Values	**Any URI**
Description	**The FaultTo header of the message.**
Example	`<header name="FaultTo" value="http://localhost:9000"/>`

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!--
 ~  Licensed to the Apache Software Foundation (ASF) under one
 ~  or more contributor license agreements.  See the NOTICE file
 ~  distributed with this work for additional information
 ~  regarding copyright ownership.  The ASF licenses this file
 ~  to you under the Apache License, Version 2.0 (the
 ~  "License"); you may not use this file except in compliance
 ~  with the License.  You may obtain a copy of the License at
 ~
 ~   http://www.apache.org/licenses/LICENSE-2.0
 ~
 ~  Unless required by applicable law or agreed to in writing,
 ~  software distributed under the License is distributed on an
 ~   * "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
 ~  KIND, either express or implied.  See the License for the
 ~  specific language governing permissions and limitations
 ~  under the License.
 -->

<xs:schema
```

```
      xmlns:xs="http://www.w3.org/2001/XMLSchema"
      elementFormDefault="qualified"
      targetNamespace="http://ws.apache.org/ns/synapse"
      xmlns="http://ws.apache.org/ns/synapse">

   <xs:include schemaLocation="../../misc/common.xsd" />

   <xs:element name="property">
      <xs:annotation>
         <xs:documentation source="description">
            Manipulates the message properties by setting and/or removing property values.
When setting property
            values, the input can be a constant or a variable value generated by an XPath
expression.
         </xs:documentation>
      </xs:annotation>
      <xs:complexType>
         <xs:sequence>
            <xs:any minOccurs="0" maxOccurs="1" processContents="skip" />
         </xs:sequence>
         <xs:attributeGroup ref="nameValueOrExpression" />
         <xs:attribute name="scope" use="optional" default="synapse">
            <xs:simpleType>
               <xs:restriction base="xs:string">
                  <xs:enumeration value="default" />
                  <xs:enumeration value="axis2" />
                  <xs:enumeration value="transport" />
                  <xs:enumeration value="axis2-client" />
                  <xs:enumeration value="operation" />
                  <xs:enumeration value="registry" />
                  <xs:enumeration value="system" />
                  <xs:enumeration value="synapse" />
                  <xs:enumeration value="environment" />
                  <xs:enumeration value="file" />
               </xs:restriction>
            </xs:simpleType>
         </xs:attribute>
         <xs:attribute name="type" default="STRING" use="optional">
            <xs:simpleType>
               <xs:restriction base="xs:string">
                  <xs:enumeration value="STRING" />
                  <xs:enumeration value="INTEGER" />
                  <xs:enumeration value="BOOLEAN" />
                  <xs:enumeration value="DOUBLE" />
```

```
                <xs:enumeration value="FLOAT" />
                <xs:enumeration value="LONG" />
                <xs:enumeration value="SHORT" />
                <xs:enumeration value="OM" />
                <xs:enumeration value="JSON" />
              </xs:restriction>
            </xs:simpleType>
          </xs:attribute>
          <xs:attribute name="pattern" type="xs:string" use="optional" />
          <xs:attribute name="group" type="xs:int" use="optional" />
          <xs:attribute name="description" type="xs:string" use="optional" />
        </xs:complexType>
    </xs:element>

</xs:schema>
```

Switch Mediator

The Switch Mediator is an XPath or JSONPath filter. The XPath or JSONPath is evaluated and returns a string. This string is matched against the regular expression in each switch case mediator, in the specified order. If a matching case is found, it will be executed, and the remaining switch case mediators are not processed. If none of the case statements are matching, and a default case is specified, the default will be executed.

Info

The Switch mediator is a conditionally content aware mediator.

Syntax¶

```
<switch source="[XPath|json-eval(JSON Path)]">
    <case regex="string">
      mediator+
    </case>+
    <default>
      mediator+
    </default>?

 </switch>
```

Configuration¶

The parameters available to configure the Switch mediator are as follows.

Source XPath	**The source XPath or JSONPath to be evaluated. When specifying a JSONPath, use the format `json-eval(<JSON_PATH>)` , such as `json-eval(getQuote.request.symbol)` . If you use namespaces in the expression, click Namespaces and map the namespace prefix to the correct URI.**
Number of cases	**This parameter displays the number of cases currently added to the Switch mediator configuration.**
Specify default case	**Click this link to add a default switch-case mediator. Adding a default switch case mediator is optional. If it is specified, it will be executed if no matching switch-case is identified.**

Switch-case mediator¶

1. **To add a case, click Add case , which adds an empty switch-case mediator under the Switch mediator. A switch-case mediator would appear as a child of the Switch mediator.**
2. **Click Case to configure the switch-case mediator. The page will expand to display the section shown below where a regular expression can be added in the Case Value (Regular Expression) parameter.**
3. **Click Case again and click Add Child , and add the mediator(s) you want to execute when this case matches the switching value.**

Examples¶

In this example the Property mediator sets the local property named `symbol` on the current message depending on the evaluation of the string. It will get the text of symbol element and match it against the values `MSFT` and `IBM` . If the text does not match either of these symbols, the default case will be executed.

```
<switch source="//m0:getQuote/m0:request/m0:symbol"
xmlns:m0="http://services.samples/xsd">
      <case regex="IBM">
          <!-- the property mediator sets a local property on the
*current* message -->
          <property name="symbol" value="Great stock - IBM"/>
      </case>
```

```
      <case regex="MSFT">
          <property name="symbol" value="Are you sure? - MSFT"/>
      </case>
      <default>
          <!-- it is possible to assign the result of an XPath or JSON
Path expression as well -->
          <property name="symbol"
                expression="fn:concat('Normal Stock - ',
//m0:getQuote/m0:request/m0:symbol)"
                xmlns:m0="http://services.samples/xsd"/>
      </default>

  </switch>
```

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!--
 ~  Licensed to the Apache Software Foundation (ASF) under one
 ~  or more contributor license agreements.  See the NOTICE file
 ~  distributed with this work for additional information
 ~  regarding copyright ownership.  The ASF licenses this file
 ~  to you under the Apache License, Version 2.0 (the
 ~  "License"); you may not use this file except in compliance
 ~  with the License.  You may obtain a copy of the License at
 ~
 ~   http://www.apache.org/licenses/LICENSE-2.0
 ~
 ~  Unless required by applicable law or agreed to in writing,
 ~  software distributed under the License is distributed on an
 ~   * "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
 ~  KIND, either express or implied.  See the License for the
 ~  specific language governing permissions and limitations
 ~  under the License.
 -->

<xs:schema
      xmlns:xs="http://www.w3.org/2001/XMLSchema"
      elementFormDefault="qualified"
      targetNamespace="http://ws.apache.org/ns/synapse"
      xmlns="http://ws.apache.org/ns/synapse">

   <xs:include schemaLocation="../mediators.xsd" />
```

```
   <xs:element name="switch">
      <xs:annotation>
         <xs:documentation source="description">
            Creates two or more alternative routes (switch cases) in the message flow. Messages
are filtered by an
            XPath or JSONPath and matched against a regular expression specified in the switch
cases. Messages are
            then routed through the matching switch case.
         </xs:documentation>
      </xs:annotation>
      <xs:complexType>
         <xs:sequence>
            <xs:element name="case" minOccurs="1" maxOccurs="unbounded">
               <xs:annotation>
                  <xs:documentation>
                     If the case criteria matches, the enclosing
                     mediator list will be invoked
                  </xs:documentation>
               </xs:annotation>
               <xs:complexType>
                  <xs:group ref="mediatorList" minOccurs="1" maxOccurs="unbounded" />
                  <xs:attribute name="regex" type="xs:string" use="required" />
               </xs:complexType>
            </xs:element>
            <xs:element name="default" minOccurs="0" maxOccurs="1">
               <xs:annotation>
                  <xs:documentation>
                     If none of the above case criteria matches, the enclosing
                     mediator list will be invoked
                  </xs:documentation>
               </xs:annotation>
               <xs:complexType>
                  <xs:group ref="mediatorList" minOccurs="1" maxOccurs="unbounded" />
               </xs:complexType>
            </xs:element>
         </xs:sequence>
         <xs:attribute name="source" type="xs:string" use="required" />
         <xs:attribute name="description" type="xs:string" use="optional" />
      </xs:complexType>
   </xs:element>

</xs:schema>
```

Respond Mediator

The Respond Mediator stops the processing on the current message and sends the message back to the client as a response.

Syntax¶

The respond token refers to a \< `respond` > element, which is used to stop further processing of a message and send the message back to the client.

```
<respond/>
```

Configuration¶

As with other mediators, after adding the Respond mediator to a sequence, you can click its up and down arrows to move its location in the sequence.

Example¶

Assume that you have a configuration that sends the request to the Stock Quote service and changes the response value when the symbol is WSO2 or CRF. Also assume that you want to temporarily change the configuration so that if the symbol is CRF, the ESB profile just sends the message back to the client without sending it to the Stock Quote service or performing any additional processing. To achieve this, you can add the Respond mediator at the beginning of the CRF case as shown below. All the configuration after the Respond mediator is ignored. As a result, the rest of the CRF case configuration is left intact, allowing you to revert to the original behavior in the future by removing the Respond mediator if required.

```
<proxy name="SimpleProxy" transports="http https" startonload="true"
trace="disable" xmlns="http://ws.apache.org/ns/synapse">
    <target>
         <inSequence>
      <switch source="//m0:getQuote/m0:request/m0:symbol"
xmlns:m0="http://services.samples">
        <case regex="WSO2">
          <property name="symbol" value="Great stock - WSO2"/>
          <send>
```

```
            <endpoint>
              <address
uri="http://localhost:9000/services/SimpleStockQuoteService"/>
            </endpoint>
          </send>
        </case>
        <case regex="CRF">

          <respond/>

          <property name="symbol" value="Are you sure? - CRF"/>
            <send>
              <endpoint>
                <address
uri="http://localhost:9000/services/SimpleStockQuoteService"/>
              </endpoint>
            </send>
        </case>
        <default>
          <property name="symbol"
            expression="fn:concat('Normal Stock - ',
//m0:getQuote/m0:request/m0:symbol)"
xmlns:m0="http://services.samples"/>
        </default>
      </switch>
    </inSequence>
    <outSequence>
      <send/>
    </outSequence>
  </target>

</proxy>
```

T

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!--
 Copyright (c) 2019, WSO2 Inc. (http://www.wso2.org) All Rights Reserved.
*
```

```
* Licensed under the Apache License, Version 2.0 (the "License");
* you may not use this file except in compliance with the License.
* You may obtain a copy of the License at
*
* http://www.apache.org/licenses/LICENSE-2.0
*
* Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
* limitations under the License.
-->

<xs:schema
      xmlns:xs="http://www.w3.org/2001/XMLSchema"
      elementFormDefault="qualified"
      targetNamespace="http://ws.apache.org/ns/synapse"
      xmlns="http://ws.apache.org/ns/synapse">

   <xs:element name="respond">
      <xs:annotation>
         <xs:documentation source="description">
            Terminates the processing of the current message flow and returns the message to
the client.
         </xs:documentation>
      </xs:annotation>
      <xs:complexType>
         <xs:attribute name="description" type="xs:string" use="optional" />
      </xs:complexType>
   </xs:element>

</xs:schema>
```

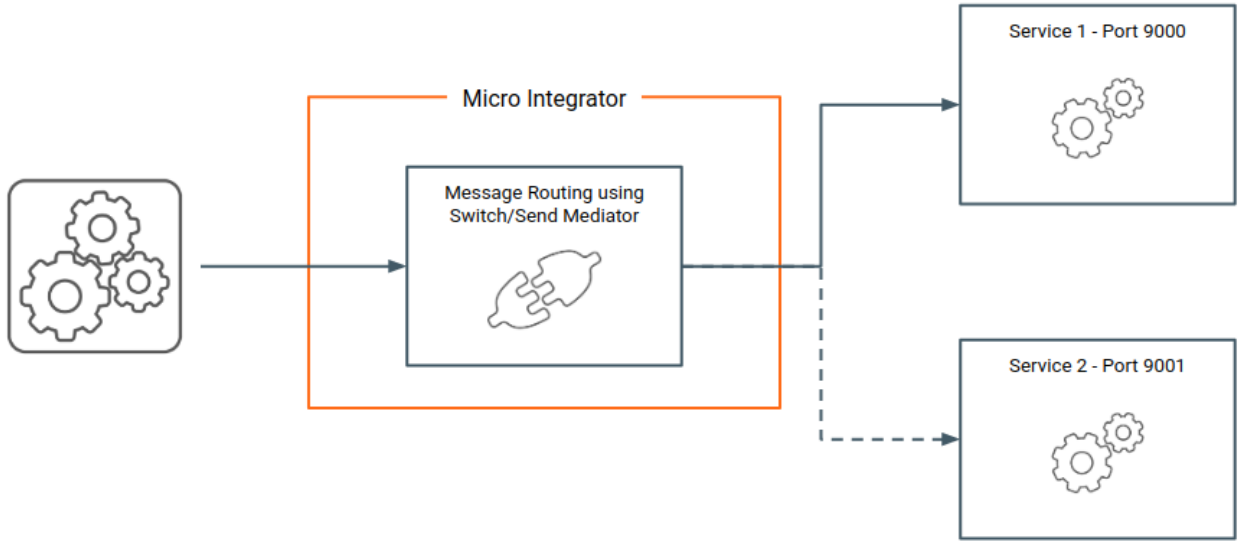
3. Integration Use Cases

3.1. Message Routing

Message routing is one of the most fundamental requirements when integrating systems/services. It considers content-based routing, header-based routing,

rules-based routing, and policy-based routing as ways of routing a message. WSO2 Micro Integrator enables these routing capabilities using the concepts of mediators and endpoints.

The following image depicts a form of message routing where a message is routed through the Micro Integrator to the appropriate service. In this case, the Switch/Send mediator can be used.



4. Example with Code

Consider the following scenario:

When the client sends the appointment reservation request to the Micro Integrator, the message payload of the request contains the name of the hospital where the appointment needs to be confirmed. The HTTP request method that is used for this is POST. Based on the hospital name sent in the request message, the Micro Integrator should route the appointment reservation to the relevant hospital's back-end service.

In this tutorial, we have three hospital services hosted as the backend:

- Grand Oak Community Hospital: `http://localhost:9090/grandoaks/`
- Clemency Medical Center: `http://localhost:9090/clemency/`
- Pine Valley Community Hospital: `http://localhost:9090/pinevalley/`

The request method is POST and the format of the request URL expected by the back-end services is `http://localhost:9090/grandoaks/categories/{category}/reserve`.

First 3 different Endpoints need to be created and the Synapse code for those are as follows;

GrankOakEP.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<endpoint name="GrandOakEP" xmlns="http://ws.apache.org/ns/synapse">
   <http method="post"
uri-template="http://localhost:9090/grandoaks/categories/{uri.var.category}/reserve">
      <suspendOnFailure>
         <initialDuration>-1</initialDuration>
         <progressionFactor>1.0</progressionFactor>
      </suspendOnFailure>
      <markForSuspension>
         <retriesBeforeSuspension>0</retriesBeforeSuspension>
      </markForSuspension>
   </http>
</endpoint>
```

ClemencyEP.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<endpoint name="ClemencyEP" xmlns="http://ws.apache.org/ns/synapse">
   <http method="post"
uri-template="http://localhost:9090/clemency/categories/{uri.var.category}/reserve">
      <suspendOnFailure>
         <initialDuration>-1</initialDuration>
         <progressionFactor>1.0</progressionFactor>
      </suspendOnFailure>
      <markForSuspension>
         <retriesBeforeSuspension>0</retriesBeforeSuspension>
      </markForSuspension>
   </http>
</endpoint>
```

PineValleyEP.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<endpoint name="PineValleyEP" xmlns="http://ws.apache.org/ns/synapse">
   <http method="post"
uri-template="http://localhost:9090/pinevalley/categories/{uri.var.category}/reserve">
      <suspendOnFailure>
         <initialDuration>-1</initialDuration>
         <progressionFactor>1.0</progressionFactor>
      </suspendOnFailure>
      <markForSuspension>
         <retriesBeforeSuspension>0</retriesBeforeSuspension>
      </markForSuspension>
   </http>
</endpoint>
```

Now the REST API needs to be created. In doing so, the appropriate mediators need to be identified to be put into the Integration Flow of the API. Here we first need a Property Mediator to extract the ‘Hospital; Property to check which Hospital’s backend the service needs to be sent to. Then a Switch mediator needs to be added to follow the same scenario as a regular case-switch. In each case, we first will add a ‘Log’ Mediator (although additional) to log that flow is routing to a particular Hospital. There needs to be a Log Mediator added into each Case. Following the Log Mediator, will be the call mediator for each case. The call mediator will call the relevant Endpoint after that. In the default case, only a Log Mediator will be added with a message saying an Error. Finally after the Switch Mediator, a Respond Mediator will be added to return the Response obtained from the Hospital’s Backend.

HealthcareAPI.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<api context="/healthcare" name="HealthcareAPI"
xmlns="http://ws.apache.org/ns/synapse">
   <resource methods="POST" uri-template="/categories/{category}/reserve">
      <inSequence>
         <property expression="json-eval($.hospital)" name="Hospital" scope="default"
type="STRING"/>
         <switch source="get-property('Hospital')">
            <case regex="grand oak community hospital">
               <log description="Grand Oak Log" level="custom" separator="(blank)">
```

```
                    <property expression="fn:concat('Routing to ', get-property('Hospital'))"
name="message"/>
                </log>
                <call>
                    <endpoint key="GrandOakEP"/>
                </call>
            </case>
            <case regex="clemency medical center">
                <log description="Clemency Log" level="custom" separator="(blank)">
                    <property expression="fn:concat('Routing to ', get-property('Hospital'))"
name="message"/>
                </log>
                <call>
                    <endpoint key="ClemencyEP"/>
                </call>
            </case>
            <case regex="pine valley community hospital">
                <log description="Pine Valley Log" level="custom" separator="(blank)">
                    <property expression="fn:concat('Routing to ', get-property('Hospital'))"
name="message"/>
                </log>
                <call>
                    <endpoint key="PineValleyEP"/>
                </call>
            </case>
            <default>
                <log description="Fault Log" level="custom" separator="(blank)">
                    <property expression="fn:concat('Invalid hospital - ',
get-property('Hospital'))" name="message"/>
                </log>
            </default>
        </switch>
        <respond/>
    </inSequence>
    <outSequence/>
    <faultSequence/>
  </resource>
</api>
```

The request shall be sent as:

Method	`POST`
Headers	`Content-Type=application/json`
URL	`http://localhost:8290/healthcare/categories/surgery/reserve` • The URI-Template format that is used in this URL was defined when creating the API resource: `http://host:port/categories/{category}/reserve`.
Body	`{ "patient": { "name": "John Doe", "dob": "1940-03-19", "ssn": "234-23-525", "address": "California", "phone": "8770586755", "email": "johndoe@gmail.com" }, "doctor": "thomas collins", "hospital_id": "grandoaks", "hospital": "grand oak community hospital", "appointment_date": "2025-04-02" }` • This JSON payload contains details of the appointment reservation, which includes patient details, doctor, hospital, and data of appointment.

Expected Response is:


```
{"appointmentNumber":1,
    "doctor":
         {"name":"thomas collins",
          "hospital":"grand oak community hospital",
          "category":"surgery","availability":"9.00 a.m - 11.00 a.m",
```

```
          "fee":7000.0},
    "patient":
        {"name":"John Doe",
         "dob":"1990-03-19",
         "ssn":"234-23-525",
         "address":"California",
         "phone":"8770586755",
         "email":"johndoe@gmail.com"},
    "fee":7000.0,
"confirmed":false,
"appointmentDate":"2025-04-02"}
```


Also, the Console shall display:

```
INFO - LogMediator message = Routing to grand oak community hospital
```

As the log sent by the Log Mediator