Weather Exercise

By Jason Tomlins

Development Notes

Background

The aim of the exercise was to wrap a simple SOAP service in a RESTful API

The SOAP service GlobalWeather provides two services:

GetWeather – based on providing the city and country

GetCitiesByCountry – based on providing the country

The URL of the web service is <http://www.webservicex.com/globalweather.asmx?WSDL>, however this site was not available for the exercise and so the provided docker container was used

Solution

To make the RESTFull HTTP requests the required parameters are provided as part of the URL

[*http://localhost:8081/api/GetWeather/{cityName}/{countryName}*](http://localhost:8081/api/GetWeather/%7bcityName%7d/%7bcountryName%7d)

[*http://localhost:8081/api/GetCitiesByCountry/{countryName}*](http://localhost:8081/api/GetCitiesByCountry/%7bcountryName%7d)

The specification is described in the process-weather.raml file.

Mulesoft ESB extracts the URI parameters and then, using Dataweave, creates the corresponding SOAP payload with which to call the service.

The XML response is transformed, using Dataweave, into a JSON response.

Error handling is by way of Try/Catch with a message returned to the client.

Design Decisions

I opted for URI parameters as opposed to requiring a JSON payload to invoke the service for ease of use and can easily be used from any browser, command line etc.

The RAML file describes and validates the URI parameters so no validation is required in the actual flows.

Structure & Naming Conventions Used

process-weather.xml – this file was auto-generated from the RAML file and, using APIKIT, routes the client requests to the appropriate flow

impl-weather.xml – this is the actual implementation for the RESTful wrapper

impl-weather:\getWeather is the flow within impl-weather.xml file to implement the GetWeather service

impl-weather:\getCitiesByCountry is the flow within impl-weather.xml to implement the GetCitiesByCountry service

Error Handling

As a result of the service not actually working I did not fully implement extensive error handling.

Outcome

As the actual web service was down I used the provided docker mock service which I had to deploy and run on a linux box as Windows Professional edition was required to run Docker on a Windows machine.

Unfortunately, I was unable to successfully call the web service end point despite the payload generating a successful response when using SOAP UI

INFO 2019-06-17 20:37:08,724 [[MuleRuntime].cpuLight.15: [process-weather].impl-weather:\getWeather.CPU\_LITE @30d16f6a] [event: db94ff00-90eb-11e9-85d9-d43d7e548382] org.mule.runtime.core.internal.processor.LoggerMessageProcessor: Creating SOAP payload using city Calgary, country Canada

INFO 2019-06-17 20:37:08,726 [[MuleRuntime].cpuIntensive.05: [process-weather].impl-weather:\getWeather.CPU\_INTENSIVE @76bfe605] [event: db94ff00-90eb-11e9-85d9-d43d7e548382] org.mule.runtime.core.internal.processor.LoggerMessageProcessor: <?xml version='1.0' encoding='UTF-8'?>

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">

<soapenv:Body>

<web:GetWeather xmlns:web="http://www.webserviceX.NET">

<web:CityName>Calgary</web:CityName>

<web:CountryName>Canada</web:CountryName>

</web:GetWeather>

</soapenv:Body>

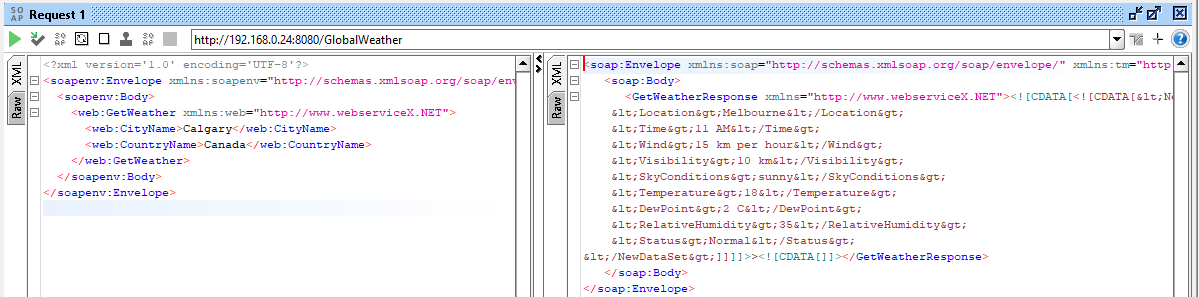
</soapenv:Envelope>

WARN 2019-06-17 20:37:08,773 [[MuleRuntime].io.07: [process-weather].impl-weather:\getWeather.BLOCKING @58509d93] [event: db94ff00-90eb-11e9-85d9-d43d7e548382] org.apache.cxf.phase.PhaseInterceptorChain: Interceptor for {http://client.internal.soap.mule.org/}ProxyService#{http://client.internal.soap.mule.org/}invoke has thrown exception, unwinding now

org.apache.cxf.binding.soap.SoapFault: Error reading XMLStreamReader: Unexpected character 'T' (code 84) in prolog; expected '<'

at [row,col {unknown-source}]: [1,1]

Example of using same payload in SOAPUI



\*Note that since this is a mock service the response does not represent the actual parameters specified.

Challenges Encountered

The first challenge I had was deploying the Docker instance. You need Windows Professional so I had to use Linux as I only have Windows Home edition. I have never worked with Docker before so there was a learning curve to overcome.

Second challenge was having not used Mule 4 and the new Anypoint Studio. There have been quite a few changes, again another learning curve.

Biggest challenge though was getting the call to the web service to work, which ultimately I could not resolve as detailed in the Outcome section.