

Module 3: Consuming Web Services

The screenshot displays the MuleSoft IDE interface. The top pane shows a REST client configuration for 'localhost:8081/united'. The bottom pane shows a REST client configuration for 'localhost:8081/delta'. The right pane shows a flow diagram for 'getUnitedFlightsFlow' and 'getDeltaFlightsFlow'.

REST Client: localhost:8081/united

```
{
  "flights": [
    {
      "airlineName": "United",
      "price": 400,
      "departureDate": "2015/03/20",
      "planeType": "Boeing 737",
      "origin": "MUA",
      "code": "ER38ed",
      "emptySeats": 0,
      "destination": "SFO"
    },
    {
      "airlineName": "United",
      "price": 345.99,
      "departureDate": "2015/02/11",
      "planeType": "Boeing 737",
      "origin": "MUA",
      "code": "ER45if",
      "emptySeats": 52,
      "destination": "LAX"
    },
    {
      "airlineName": "United",
      "price": 346,
      "departureDate": "2015/04/11",
      "planeType": "Boeing 777",
      "origin": "MUA",
      "code": "ER45jd",
      "emptySeats": 12,
      "destination": "LAX"
    },
    {
      "airlineName": "United",
      "price": 423,
      "departureDate": "2015/06/11",
      "planeType": "Boeing 707",
      "origin": "MUA",
      "code": "ER0945",
      "emptySeats": 0,
      "destination": "LAX"
    },
    {
      "airlineName": "United",
      "price": 845,
      "departureDate": "2015/07/11",
      "planeType": "Boeing 727",
      "origin": "MUA",
      "code": "ER9fje",
      "emptySeats": 32,
      "destination": "CLE"
    },
    {
      "airlineName": "United",
      "price": 245,
      "departureDate": "2015/08/11",
      "planeType": "Boeing 747",
      "origin": "MUA",
      "code": "ER3kfd",
      "emptySeats": 13,
      "destination": "CLE"
    }
  ]
}
```

Flow Diagram: getUnitedFlightsFlow

```
graph LR
    HTTP1[HTTP] --> UR[United REST Request]
    UR --> HTTP2[HTTP]
    subgraph ErrorHandling [Error handling]
        HTTP2 --> HTTP1
    end
```

REST Client: localhost:8081/delta

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<ns2:listAllFlightsResponse xmlns:ns2="http://soap.training.mulesoft.com/">
  <return>
    <airlineName>Delta</airlineName>
    <code>A1B2C</code>
    <departureDate>2015/03/20</departureDate>
    <destination>SFO</destination>
    <emptySeats>40</emptySeats>
    <origin>MUA</origin>
    <planeType>Boeing 737</planeType>
    <price>409.9</price>
  </return>
  <return>
    <airlineName>Delta</airlineName>
    <code>A1B2C4</code>
    <departureDate>2015/03/11</departureDate>
    <destination>LAX</destination>
    <emptySeats>10</emptySeats>
    <origin>MUA</origin>
    <planeType>Boeing 737</planeType>
    <price>199.99</price>
  </return>
  <return>
    <airlineName>Delta</airlineName>
    <code>A134DS</code>
    <departureDate>2015/04/11</departureDate>
  </return>
</listAllFlightsResponse>
```

Flow Diagram: getDeltaFlightsFlow

```
graph LR
    HTTP1[HTTP] --> DR[Delta SOAP Request]
    DR --> Logger[Logger]
    Logger --> HTTP2[HTTP]
    subgraph ErrorHandling [Error handling]
        HTTP2 --> HTTP1
    end
```

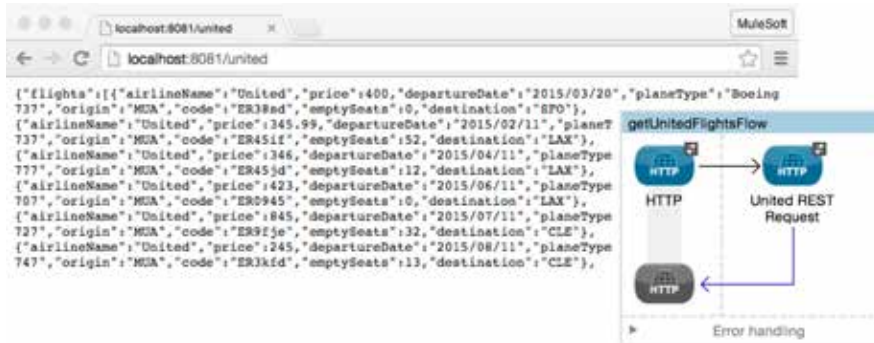
In this module, you will learn:

- About RESTful and SOAP based web services.
- What RAML is and how it can be used.
- To consume RESTful web services with and without RAML definitions.
- To consume SOAP web services.

Walkthrough 3-1: Consume a RESTful web service

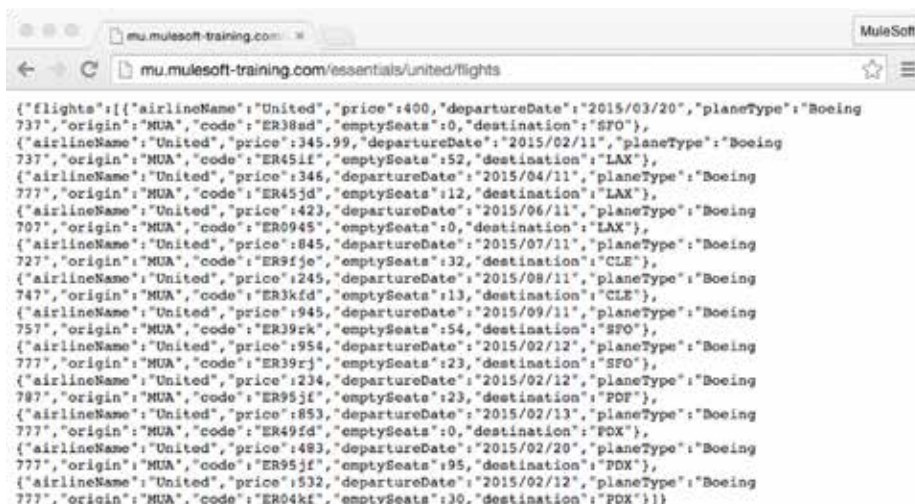
In this walkthrough and many others, you will work on building a Mule United Airline (MUA) application that returns flights for Delta, United, and American airlines. In this walkthrough, you will consume a RESTful web service that returns a list of all United flights as JSON. You will:

- Create a second flow and rename flows.
- Add an HTTP Listener connector endpoint to receive requests at <http://localhost:8081/united>.
- Add an HTTP Request connector endpoint to consume a RESTful web service for United flight data.



Make a request to the web service

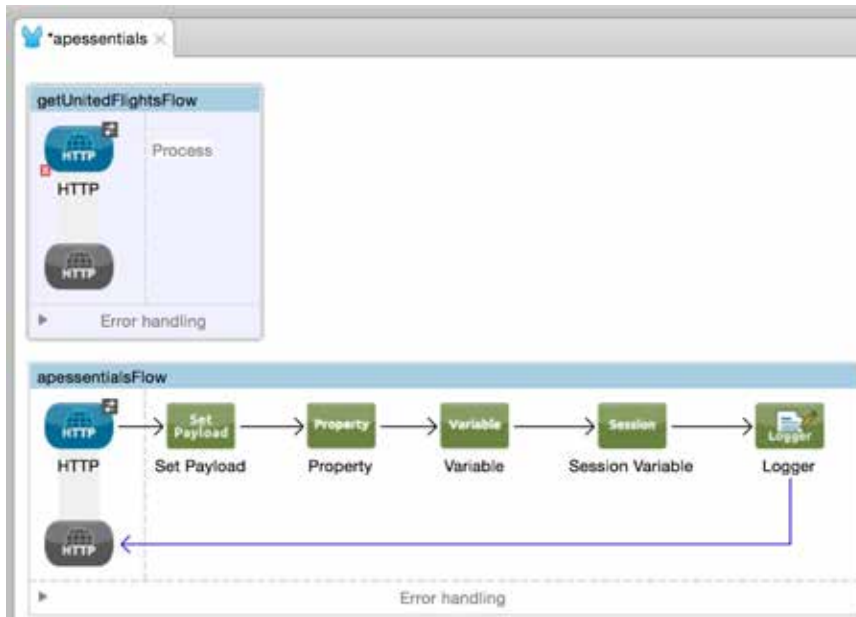
1. In your computer's system explorer, navigate to the student files folder for the course: `MUEssentials3.7_studentFiles_{date}`.
2. Open the course `snippets.txt` file.
3. In a browser window, make a request to the United RESTful web service URL listed in the course `snippets.txt` file; you should see JSON data for the United flights returned.
4. Look at the destination values; you should see SFO, LAX, CLE, PDX, and PDF.



Add a new flow with an HTTP Listener connector endpoint

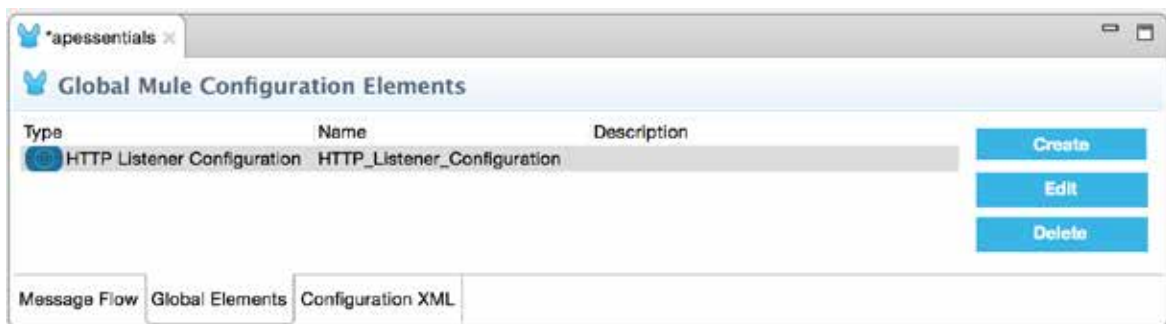
5. Return to apessentials.xml.
6. Drag out another HTTP connector and drop it in the canvas above the existing apessentialsFlow.
7. Double-click the name of the flow in the canvas and give it a new name of getUnitedFlightsFlow.

Note: You can set the name in the Properties view or directly in the blue banner.



Look at the global elements

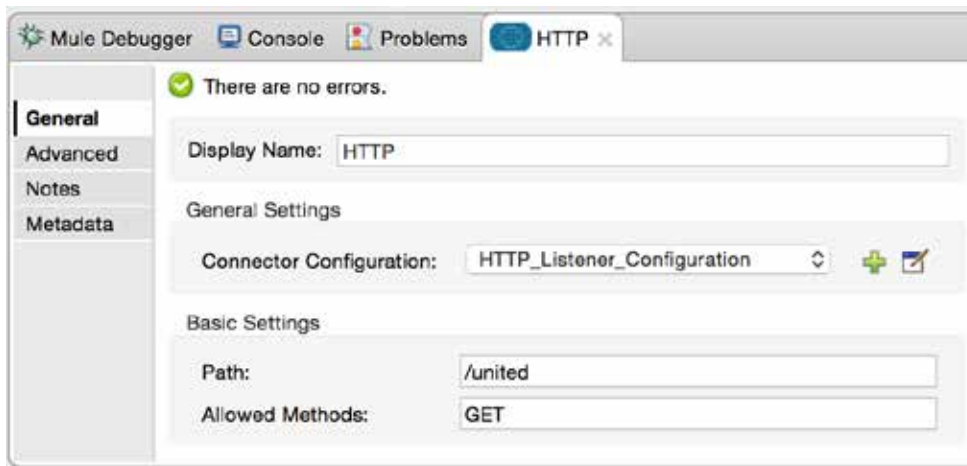
8. Click the Global Element tabs at the bottom of the canvas.
9. Select the HTTP Listener Configuration and click Edit (or double-click it).



10. In the Global Element Properties dialog box, review the HTTP_Listener_Configuration and click OK.

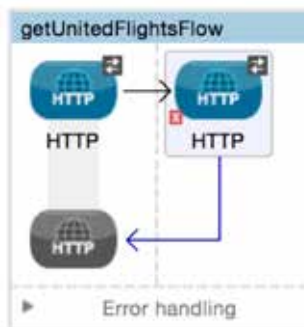
Configure the HTTP Listener connector endpoint

11. Click the Message Flow tab.
12. Double-click the new HTTP Listener connector endpoint.
13. Set the connector configuration to the existing HTTP_Listener_Configuration.
14. Set the path to /united.
15. Set the allowed methods to GET.



Add an HTTP Request connector endpoint

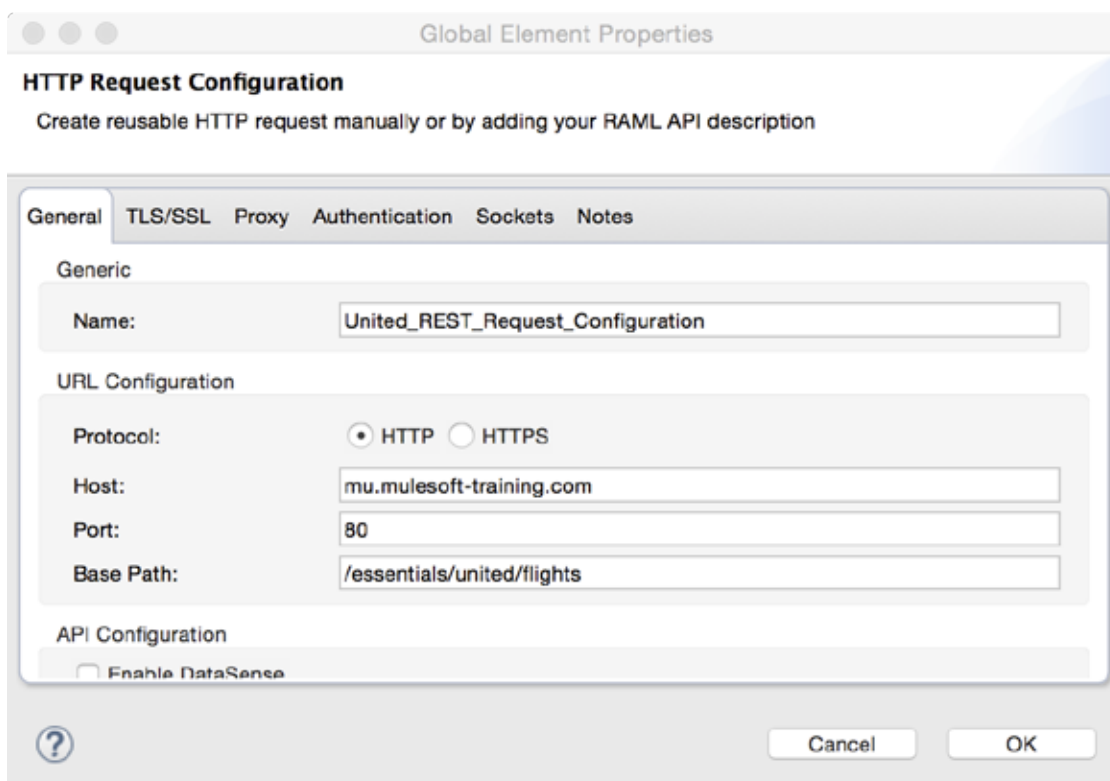
16. Drag out another HTTP connector and drop it into the process section of the flow.



17. In the Properties view, change the display name to United REST Request.
18. Click the Add button next to connector configuration.

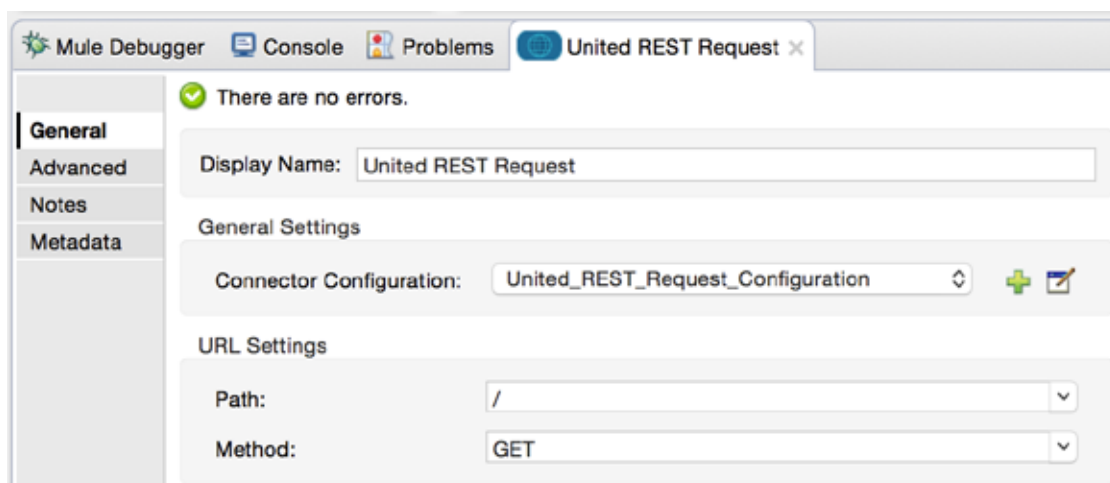
19. In the Global Element Properties dialog box, set the following values and click OK.

- Name: United_REST_Request_Configuration
- Host: Use the value for the United web service host listed in the course snippets.txt file.
- Port: Use the value for the United web service port listed in the course snippets.txt file.
- Base Path: Use the value for the United web service base path listed in the course snippets.txt file.



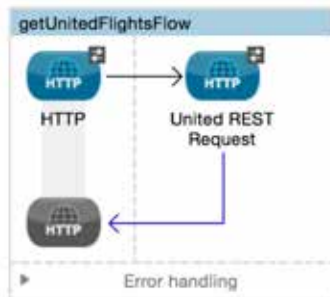
The image shows the 'Global Element Properties' dialog box with the 'HTTP Request Configuration' tab selected. The 'Generic' section has the 'Name' field set to 'United_REST_Request_Configuration'. The 'URL Configuration' section has the 'Protocol' set to 'HTTP', 'Host' set to 'mu.mulesoft-training.com', 'Port' set to '80', and 'Base Path' set to '/essentials/united/flights'. The 'API Configuration' section has the 'Enable DataSense' checkbox unchecked. The dialog has 'Cancel' and 'OK' buttons at the bottom right.

20. In the Properties view, set the path to / and the method to GET.



The image shows the 'Properties view' for the 'United REST Request' element. The 'General' tab is selected. The 'Display Name' is 'United REST Request'. Under 'General Settings', the 'Connector Configuration' is set to 'United_REST_Request_Configuration'. Under 'URL Settings', the 'Path' is set to '/' and the 'Method' is set to 'GET'. The 'Mule Debugger' tab is also visible at the top.

21. Click the Global Elements tab at the bottom of the canvas and see the new global configuration element.
22. Return to the Message Flow view.



Test the application

23. Save the file and run the application.
24. Make a request to <http://localhost:8081/united>; you should see JSON flight data returned.

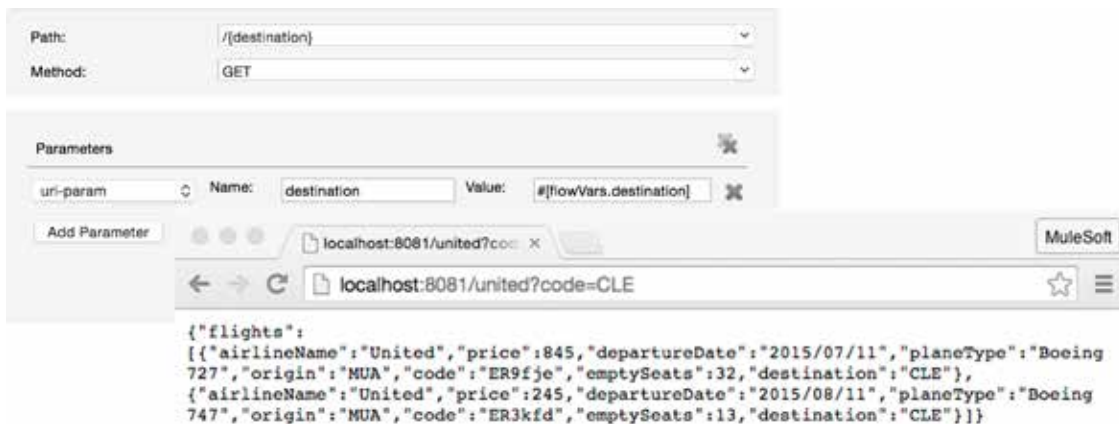
```
{
  "flights": [
    {
      "airlineName": "United",
      "price": 400,
      "departureDate": "2015/03/20",
      "planeType": "Boeing 737",
      "origin": "MUA",
      "code": "ER38sd",
      "emptySeats": 0,
      "destination": "SFO"
    },
    {
      "airlineName": "United",
      "price": 345.99,
      "departureDate": "2015/02/11",
      "planeType": "Boeing 737",
      "origin": "MUA",
      "code": "ER45if",
      "emptySeats": 52,
      "destination": "LAX"
    },
    {
      "airlineName": "United",
      "price": 346,
      "departureDate": "2015/04/11",
      "planeType": "Boeing 777",
      "origin": "MUA",
      "code": "ER45jd",
      "emptySeats": 12,
      "destination": "LAX"
    },
    {
      "airlineName": "United",
      "price": 423,
      "departureDate": "2015/06/11",
      "planeType": "Boeing 707",
      "origin": "MUA",
      "code": "ER0945",
      "emptySeats": 0,
      "destination": "LAX"
    },
    {
      "airlineName": "United",
      "price": 845,
      "departureDate": "2015/07/11",
      "planeType": "Boeing 727",
      "origin": "MUA",
      "code": "ER9fje",
      "emptySeats": 32,
      "destination": "CLE"
    },
    {
      "airlineName": "United",
      "price": 245,
      "departureDate": "2015/08/11",
      "planeType": "Boeing 747",
      "origin": "MUA",
      "code": "ER3kfd",
      "emptySeats": 13,
      "destination": "CLE"
    },
    {
      "airlineName": "United",
      "price": 945,
      "departureDate": "2015/09/11",
      "planeType": "Boeing 757",
      "origin": "MUA",
      "code": "ER39rk",
      "emptySeats": 54,
      "destination": "SFO"
    },
    {
      "airlineName": "United",
      "price": 954,
      "departureDate": "2015/02/12",
      "planeType": "Boeing 777",
      "origin": "MUA",
      "code": "ER39rj",
      "emptySeats": 23,
      "destination": "SFO"
    },
    {
      "airlineName": "United",
      "price": 234,
      "departureDate": "2015/02/12",
      "planeType": "Boeing 787",
      "origin": "MUA",
      "code": "ER95jf",
      "emptySeats": 23,
      "destination": "PDF"
    },
    {
      "airlineName": "United",
      "price": 853,
      "departureDate": "2015/02/13",
      "planeType": "Boeing 777",
      "origin": "MUA",
      "code": "ER49fd",
      "emptySeats": 0,
      "destination": "PDX"
    },
    {
      "airlineName": "United",
      "price": 483,
      "departureDate": "2015/02/20",
      "planeType": "Boeing 777",
      "origin": "MUA",
      "code": "ER95jf",
      "emptySeats": 95,
      "destination": "PDX"
    },
    {
      "airlineName": "United",
      "price": 532,
      "departureDate": "2015/02/12",
      "planeType": "Boeing 777",
      "origin": "MUA",
      "code": "ER04kf",
      "emptySeats": 30,
      "destination": "PDX"
    }
  ]
}
```

Walkthrough 3-2: Pass arguments to a RESTful web service

In this walkthrough, you will retrieve United flights for a specific destination by setting the destination as a URI parameter. You will:

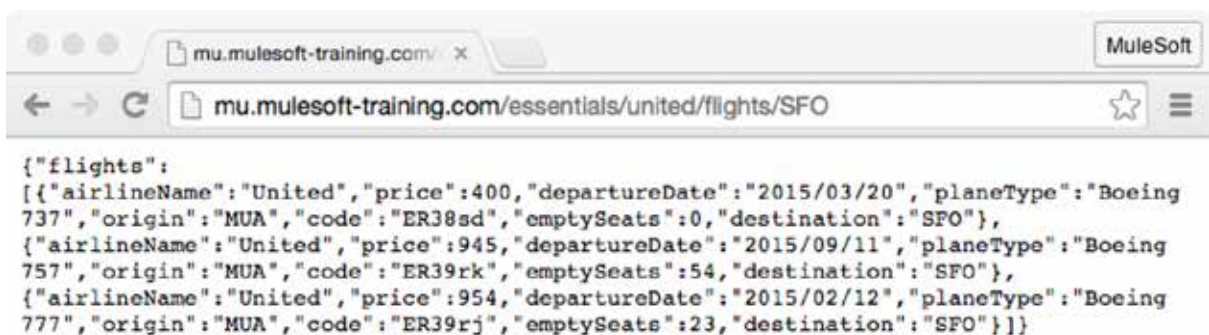
- Modify the HTTP Request connector endpoint to use a URI parameter for the destination.
- Set the destination to a static value.
- Set the destination to a dynamic query parameter value.
- Create a variable to set the destination.

Note: In a later module, you will add an HTML form to the application for destination selection.



Make a request to the web service in a browser or another tool

1. Make a request to the United RESTful web service URL for a destination listed in the course snippets.txt file; you should see JSON data for only the flights to SFO.



2. Make additional requests for destinations of LAX, CLE, PDX, or PDF.

Add a URI parameter with a static value

3. Return to `apessentials.xml`.
4. Double-click the United REST Request endpoint.

5. In the Properties view, click the Add Parameter button.
6. Set the following parameter values.
 - Parameter type: uri-param
 - Name: destination
 - Value: SFO
7. Change the United REST Request endpoint path to `/{"destination"}`.

The screenshot shows the Mule Debugger interface with the 'United REST Request' component selected. The 'General' tab is active, displaying the following settings:

- Display Name:** United REST Request
- Connector Configuration:** United_HTTP_Request_Configuration
- URL Settings:**
 - Path:** `/{"destination"}`
 - Method:** GET
- Parameters:**
 - Parameter type: uri-param
 - Name: destination
 - Value: SFO

Buttons for 'Add Parameter' and 'Remove Parameter' are visible at the bottom of the parameters section.

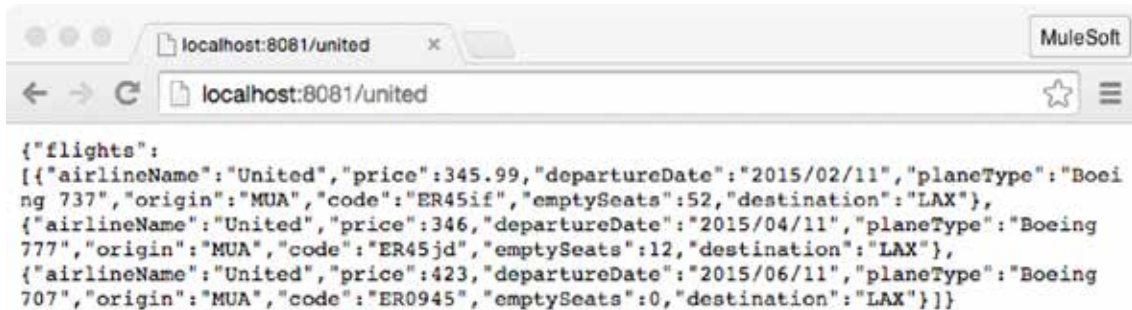
Test the application

8. Save the application to redeploy the application and make a request to `http://localhost:8081/united/`; you should only get the SFO flights.

The screenshot shows a web browser window with the address bar set to `localhost:8081/united`. The response is a JSON array of flight objects, all with the destination 'SFO':

```
{"flights": [{"airlineName": "United", "price": 400, "departureDate": "2015/03/20", "planeType": "Boeing 737", "origin": "MUA", "code": "ER38sd", "emptySeats": 0, "destination": "SFO"}, {"airlineName": "United", "price": 945, "departureDate": "2015/09/11", "planeType": "Boeing 757", "origin": "MUA", "code": "ER39rk", "emptySeats": 54, "destination": "SFO"}, {"airlineName": "United", "price": 954, "departureDate": "2015/02/12", "planeType": "Boeing 777", "origin": "MUA", "code": "ER39rj", "emptySeats": 23, "destination": "SFO"}]}
```


9. Modify the United REST Request endpoint and set the URI parameter value to LAX.
10. Save and redeploy the application and make another request to <http://localhost:8081/united/>; you should now only get the LAX flights.



```
{
  "flights": [
    {
      "airlineName": "United",
      "price": 345.99,
      "departureDate": "2015/02/11",
      "planeType": "Boeing 737",
      "origin": "MUA",
      "code": "ER45if",
      "emptySeats": 52,
      "destination": "LAX"
    },
    {
      "airlineName": "United",
      "price": 346,
      "departureDate": "2015/04/11",
      "planeType": "Boeing 777",
      "origin": "MUA",
      "code": "ER45jd",
      "emptySeats": 12,
      "destination": "LAX"
    },
    {
      "airlineName": "United",
      "price": 423,
      "departureDate": "2015/06/11",
      "planeType": "Boeing 707",
      "origin": "MUA",
      "code": "ER0945",
      "emptySeats": 0,
      "destination": "LAX"
    }
  ]
}
```


Add a URI parameter with a dynamic value

11. Return to the Properties view for the United REST Request endpoint.
12. Change the value of the uri-param from LAX to an expression for the value of a query parameter called code.

```
#[message.inboundProperties.'http.query.params'.code]
```

Test the application

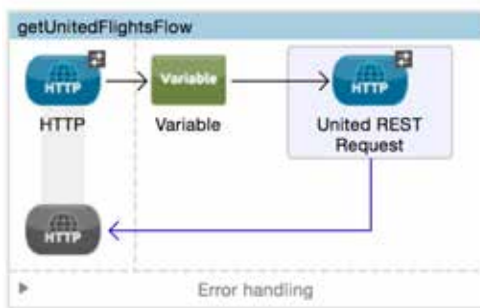
13. Save and redeploy the application and make a request to <http://localhost:8081/united/?code=CLE>; you should only get the CLE flights.



```
{
  "flights": [
    {
      "airlineName": "United",
      "price": 845,
      "departureDate": "2015/07/11",
      "planeType": "Boeing 727",
      "origin": "MUA",
      "code": "ER9fje",
      "emptySeats": 32,
      "destination": "CLE"
    },
    {
      "airlineName": "United",
      "price": 245,
      "departureDate": "2015/08/11",
      "planeType": "Boeing 747",
      "origin": "MUA",
      "code": "ER3kfd",
      "emptySeats": 13,
      "destination": "CLE"
    }
  ]
}
```

Create a variable to set the destination

14. Add a Variable transformer before the United REST Request endpoint.



15. In the Variable Properties view, change the display name to Set Destination.
16. Set the operation to Set Variable and the name to destination.
17. Use a ternary expression to set the value to 'SFO' or the value of a query parameter called code.

```
#[(message.inboundProperties.'http.query.params'.code == empty) ?  
'SFO' : message.inboundProperties.'http.query.params'.code]
```

Operation: ☒ Set Variable ☐ Remove Variable

Name:

Value:

18. Navigate to the Properties view for the United REST connector endpoint.
19. Modify the URI parameter to use the new destination variable.

Parameters

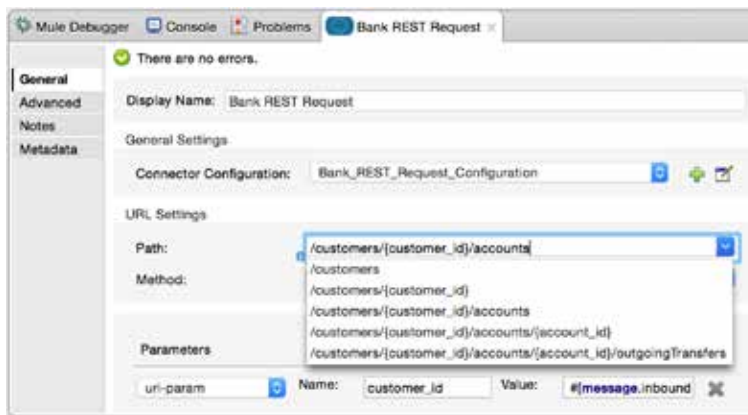
uri-param	Name: destination	Value: #[flowVars.destination]
-----------	-------------------	--------------------------------

20. Save and redeploy the application and make a request to <http://localhost:8081/united>; you should see only flights to SFO again.
21. Make another request to <http://localhost:8081/united?code=PDX>; you should now see flights to PDX.

Walkthrough 3-3: Consume a RESTful web service that has a RAML definition

In this walkthrough, you will consume a RESTful web service containing some simple bank account data that has a RAML definition file. You will:

- Add a third flow to the application.
- Add an HTTP Listener connector endpoint to receive requests at <http://localhost:8081/bank>.
- Add an HTTP Request connector endpoint to consume a RESTful web service defined with a RAML file.



Add a new flow with an HTTP Listener connector endpoint

1. Return to apessentials.xml.
2. Drag out another HTTP connector and drop it in the canvas between the two existing flows.
3. Rename the flow to getBankAccountsFlow.



4. In the Properties view for the endpoint, set the connector configuration to the existing HTTP_Listener_Configuration.

5. Set the path to /bank.
6. Set the allowed methods to GET.

Add an HTTP Request connector with a RAML location

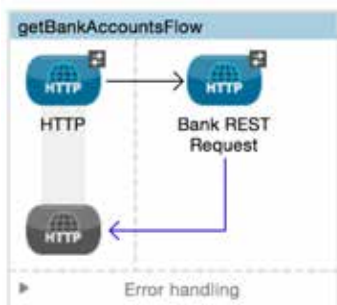
7. Drag out another HTTP connector and drop it in the process section of the new flow.
8. In the Properties view, change the name to Bank REST Request.
9. Click the Add button next to connector configuration.
10. In the Global Element Properties dialog box, change the name to Bank_REST_Request_Configuration.
11. Set the RAML location to the Banking RAML URL listed in the course snippets.txt file.
12. Wait for the RAML to be parsed and the host, port, and base path fields to be populated and then click OK.

Note: If the fields did not populate, click the Reload RAML button next to the RAML location.

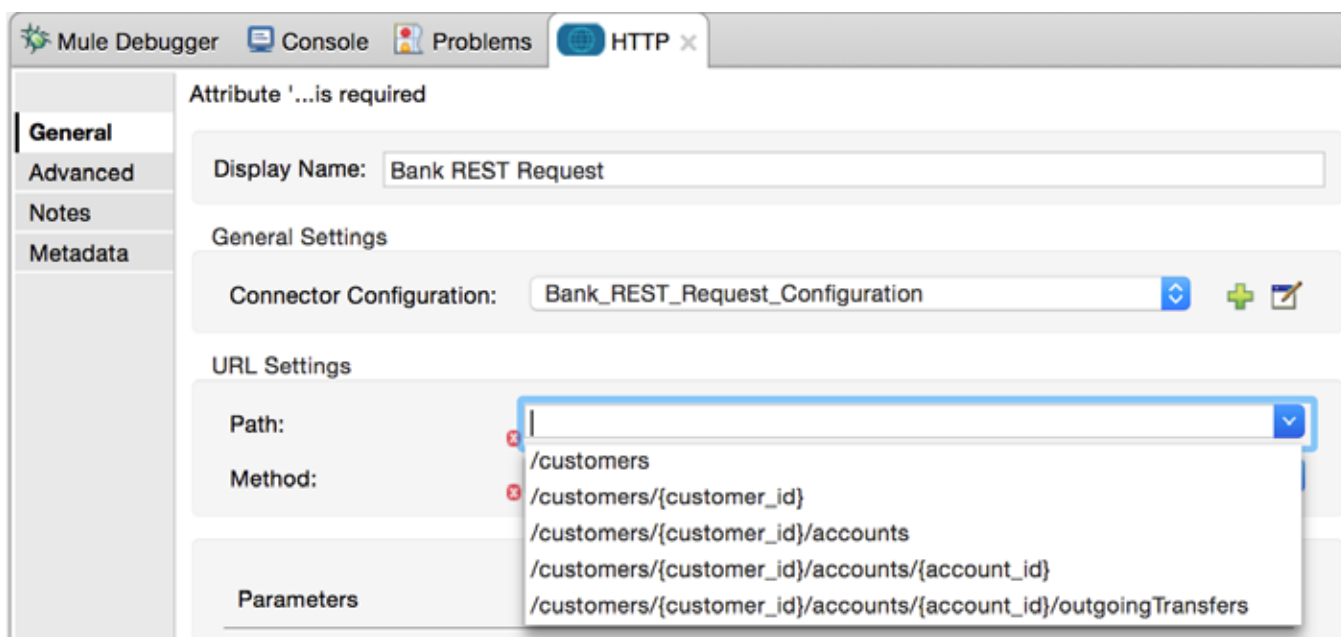
The screenshot shows the 'Global Element Properties' dialog box for an 'HTTP Request Configuration'. The dialog has a title bar with three window control buttons and the text 'Global Element Properties'. Below the title bar is the section 'HTTP Request Configuration' with the subtitle 'Create reusable HTTP request manually or by adding your RAML API description'. The dialog is divided into several tabs: 'General', 'TLS/SSL', 'Proxy', 'Authentication', 'Sockets', and 'Notes'. The 'General' tab is selected. Inside the 'General' tab, there are three main sections: 'Generic', 'URL Configuration', and 'API Configuration'. The 'Generic' section has a 'Name' field with the value 'Bank_REST_Request_Configuration'. The 'URL Configuration' section has a 'Protocol' section with radio buttons for 'HTTP' (selected) and 'HTTPS'. Below this are fields for 'Host' (mu.mulesoft-training.com), 'Port' (80), and 'Base Path' (/build/banking/rest). The 'API Configuration' section has a checkbox for 'Enable DataSense' which is unchecked. Below this is a 'RAML Location' field with the value 'https://anypoint.mulesoft.com/apiplatform/repository/v2/organiz' and a 'Browse' button. At the bottom right of the 'API Configuration' section is a link 'Search RAML in Exchange'. At the bottom of the dialog are a help icon (question mark in a circle) and two buttons: 'Cancel' and 'OK'.

13. Click the Global Elements tab at the bottom of the canvas and see the new global configuration element.

14. Return to the Message Flow view.

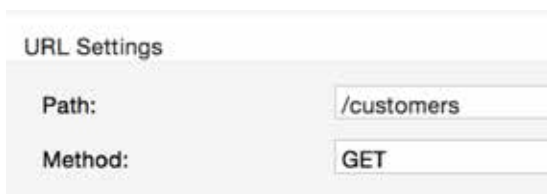


15. In the Bank REST Request Properties view, click the drop-down button for the path; you should see the available resources (paths) for the RESTful web service defined by the RAML file.



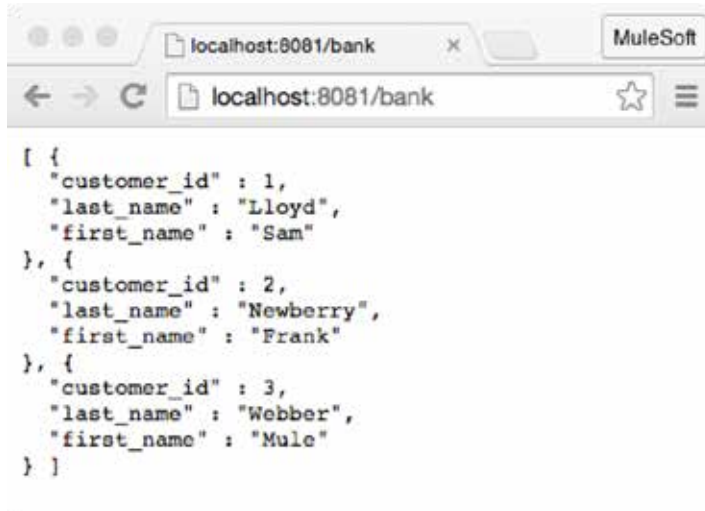
16. Select the /customers path.

17. Set the method to GET.



Test the application

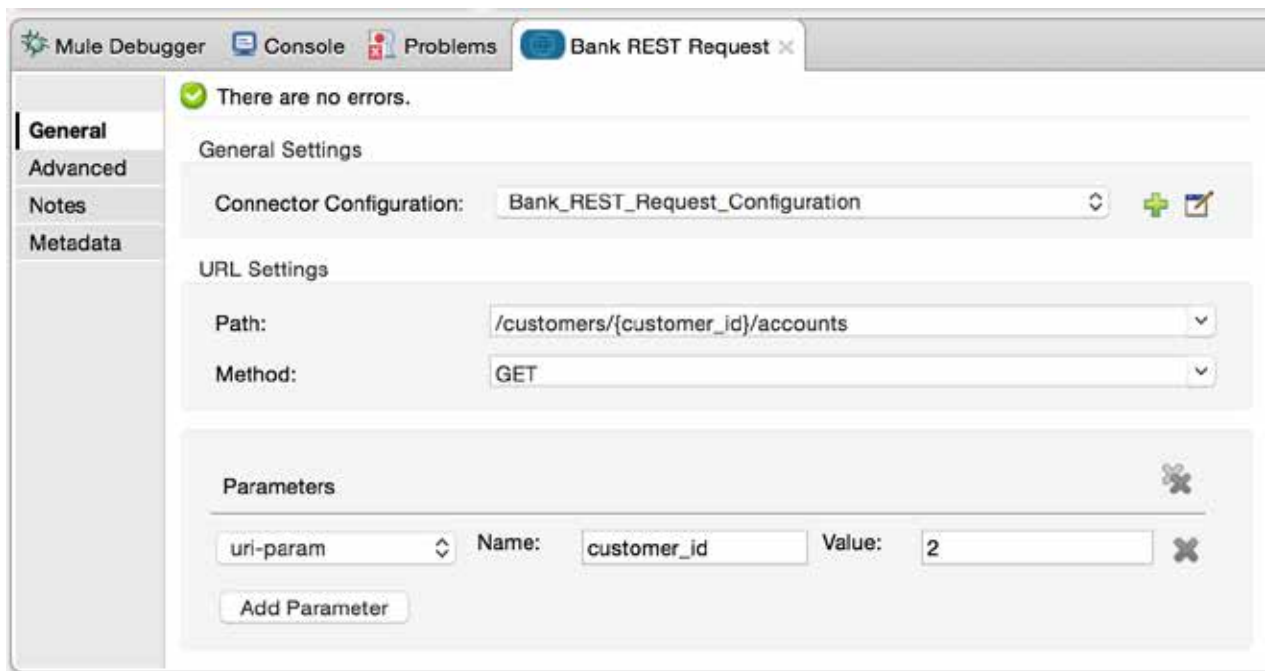
18. Save to redeploy the application and make a request to <http://localhost:8081/bank>; you should see the customers returned as JSON.



```
[ {
  "customer_id" : 1,
  "last_name" : "Lloyd",
  "first_name" : "Sam"
}, {
  "customer_id" : 2,
  "last_name" : "Newberry",
  "first_name" : "Frank"
}, {
  "customer_id" : 3,
  "last_name" : "Webber",
  "first_name" : "Mule"
} ]
```

Modify the request to use a path requiring a URI parameter

19. Return to the Properties view for the Bank REST Request endpoint.
20. Change the path to `/customers/{customer_id}/accounts` and the method to GET; a URI parameter with the name `customer_id` should have been created automatically.
21. Set the `customer_id` parameter to a value of 2.



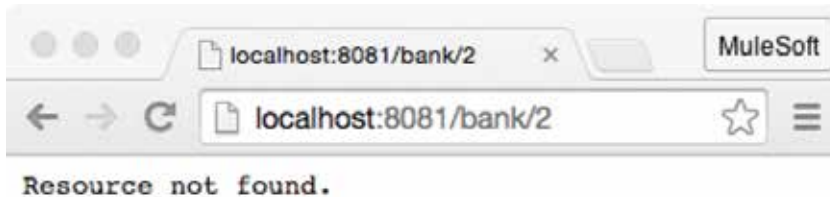
Test the application

22. Save to redeploy the application make a request to <http://localhost:8081/bank>; you should see the account info for the customer with an ID of 2 returned as JSON.



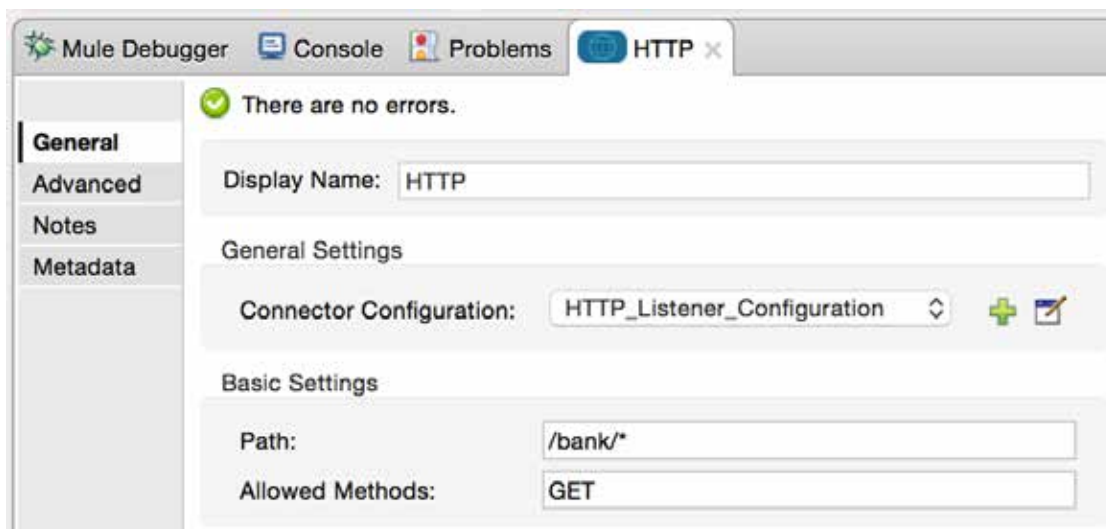
```
[ {
  "balance" : 40000.00,
  "customer_id" : 2,
  "account_id" : 587
}, {
  "balance" : 5000.29,
  "customer_id" : 2,
  "account_id" : 611
} ]
```

23. Make a request and try to pass the customer ID as a URI parameter:
<http://localhost:8081/bank/2>; you should get a Resource not found response.

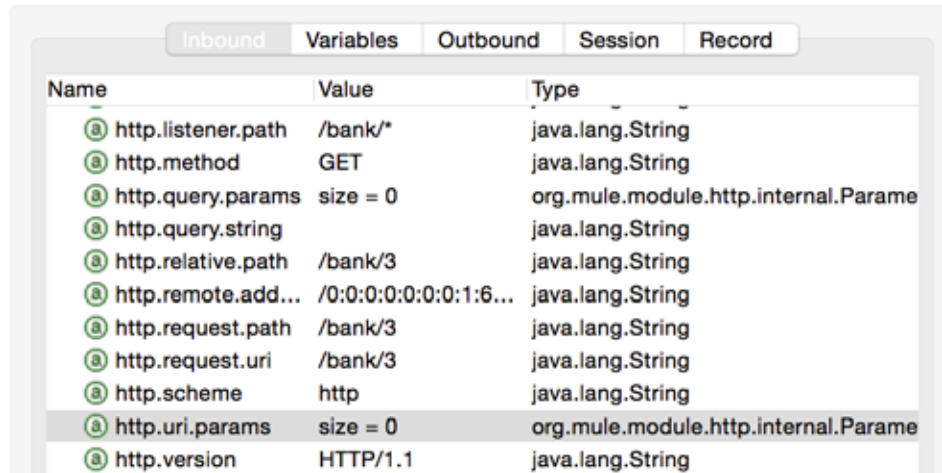


Get the value of an HTTP Listener endpoint URI parameter

24. Return to the Properties view for the HTTP Listener endpoint.
25. Change the path to use a wildcard to specify any path starting with /bank/.

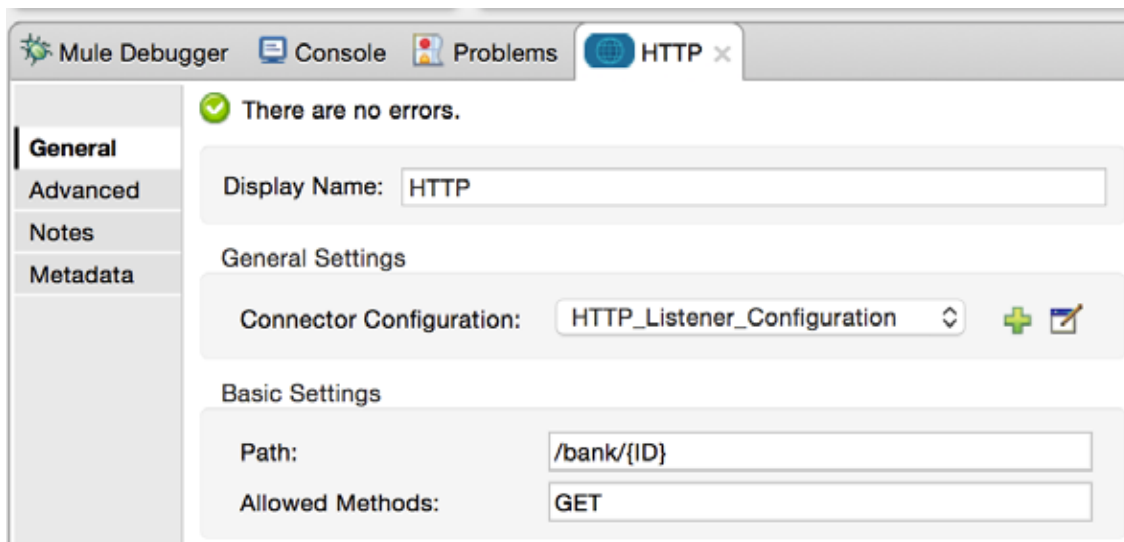


26. Add a breakpoint to the Bank REST Request endpoint.
27. Save the file and debug the application.
28. Make a request to <http://localhost:8081/bank/3>.
29. In the Mule Debugger view, locate the `http.request.uri` and `http.uri.params` inbound properties.



Name	Value	Type
<code>http.listener.path</code>	<code>/bank/*</code>	<code>java.lang.String</code>
<code>http.method</code>	<code>GET</code>	<code>java.lang.String</code>
<code>http.query.params</code>	<code>size = 0</code>	<code>org.mule.module.http.internal.Parame</code>
<code>http.query.string</code>		<code>java.lang.String</code>
<code>http.relative.path</code>	<code>/bank/3</code>	<code>java.lang.String</code>
<code>http.remote.add...</code>	<code>/0:0:0:0:0:0:1:6...</code>	<code>java.lang.String</code>
<code>http.request.path</code>	<code>/bank/3</code>	<code>java.lang.String</code>
<code>http.request.uri</code>	<code>/bank/3</code>	<code>java.lang.String</code>
<code>http.scheme</code>	<code>http</code>	<code>java.lang.String</code>
<code>http.uri.params</code>	<code>size = 0</code>	<code>org.mule.module.http.internal.Parame</code>
<code>http.version</code>	<code>HTTP/1.1</code>	<code>java.lang.String</code>

30. Step through the rest of the application; you should still get the account info for the customer with an ID of 2.
31. Return to the Properties view for the HTTP Listener endpoint.
32. Change the path to specify a URI parameter called ID: `/bank/{ID}`.





Mule Debugger Console Problems HTTP x

There are no errors.

General

Display Name:

General Settings

Connector Configuration:  

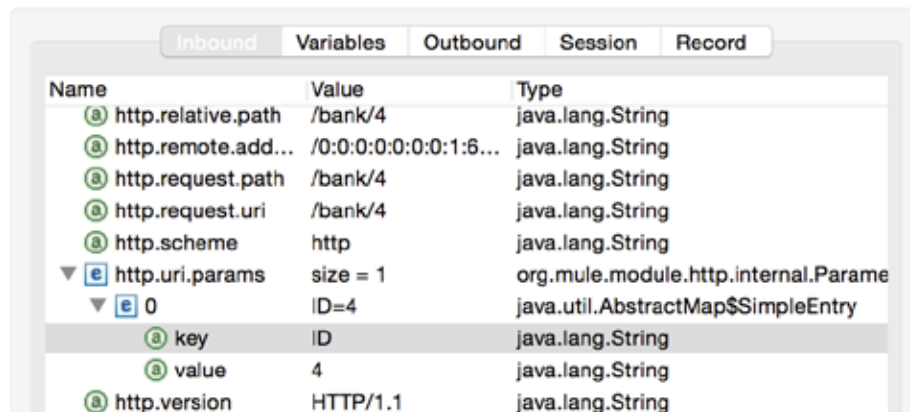
Basic Settings

Path:

Allowed Methods:

33. Save the file to redeploy the application in debug mode.
34. Make a request to <http://localhost:8081/bank/4>.

35. In the Mule Debugger view, locate the `http.request.uri` and `http.uri.params` inbound properties.



Name	Value	Type
<code>http.relative.path</code>	<code>/bank/4</code>	<code>java.lang.String</code>
<code>http.remote.add...</code>	<code>/0:0:0:0:0:0:1:6...</code>	<code>java.lang.String</code>
<code>http.request.path</code>	<code>/bank/4</code>	<code>java.lang.String</code>
<code>http.request.uri</code>	<code>/bank/4</code>	<code>java.lang.String</code>
<code>http.scheme</code>	<code>http</code>	<code>java.lang.String</code>
▼ <code>http.uri.params</code>	<code>size = 1</code>	<code>org.mule.module.http.internal.Parame</code>
▼ <code>0</code>	<code>ID=4</code>	<code>java.util.AbstractMap\$SimpleEntry</code>
<code>key</code>	<code>ID</code>	<code>java.lang.String</code>
<code>value</code>	<code>4</code>	<code>java.lang.String</code>
<code>http.version</code>	<code>HTTP/1.1</code>	<code>java.lang.String</code>

36. Step through the rest of the application; you should still get the account info for the customer with an ID of 2

Set the HTTP Request endpoint URI parameter to a dynamic value

37. Return to the Properties view for the Bank REST Request endpoint.

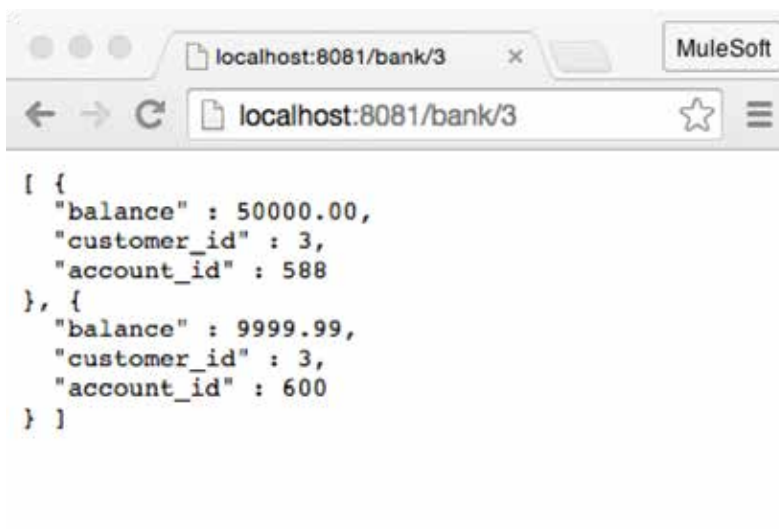
38. Change the value of the `customer_id` uri-param from 2 to an expression for the value of an HTTP Listener endpoint URI parameter called `ID`.

```
#[message.inboundProperties.'http.uri.params'.ID]
```

Test the application

39. Save the file and run the application.

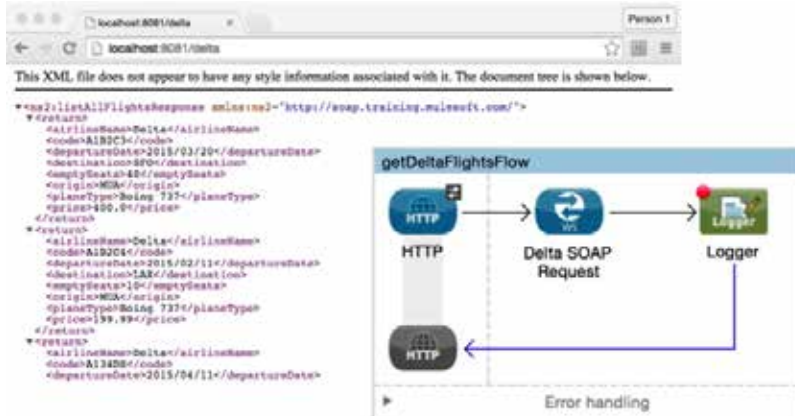
40. Make another request to <http://localhost:8081/bank/3>; you should now see the account info for the customer with an ID of 3.



Walkthrough 3-4: Consume a SOAP web service

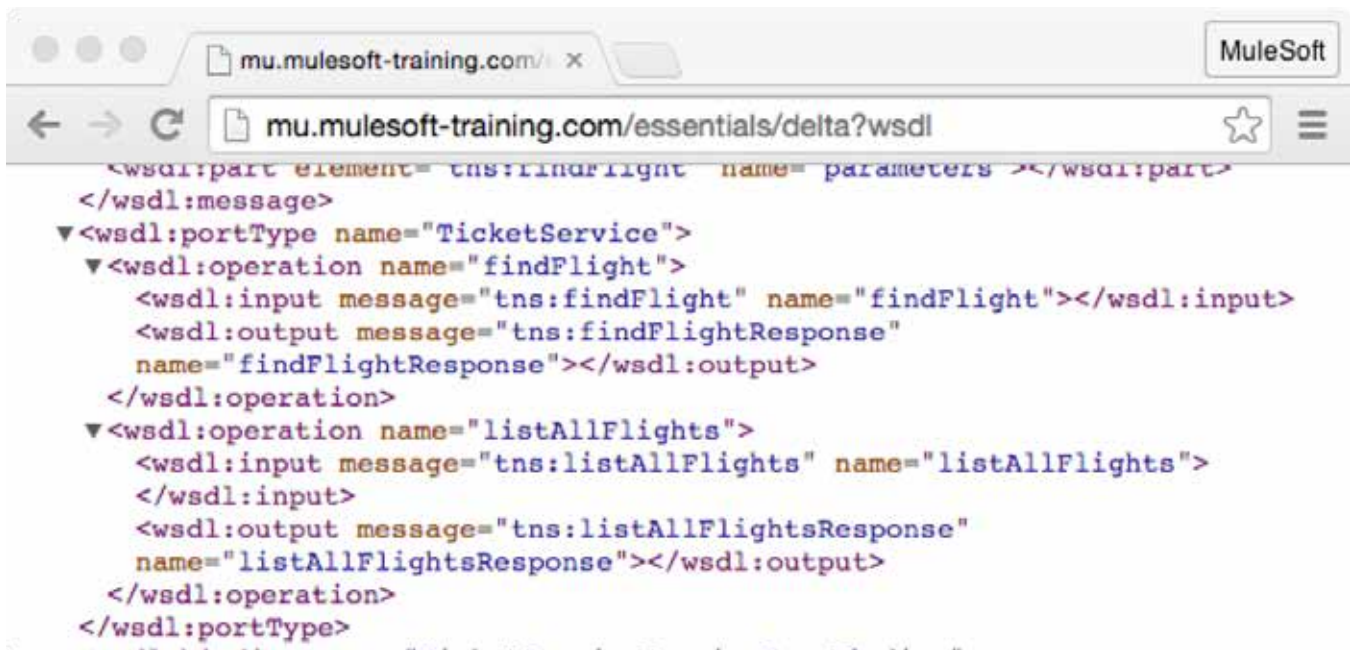
In this walkthrough, you will consume a SOAP web service that returns a list of flights for Delta airlines. You will:

- Create a fourth flow with an endpoint to receive requests at <http://localhost:8081/delta>.
- Add a Web Service Consumer connector to consume a SOAP web service for Delta flight data.
- Use the DOM to XML transformer to display the SOAP response.



Browse the WSDL

1. Make a request to the Delta SOAP web service WSDL listed in the course snippets.txt file; you should see the web service WSDL returned.
2. Browse the WSDL; you should find references to operations `listAllFlights` and `findFlight`.

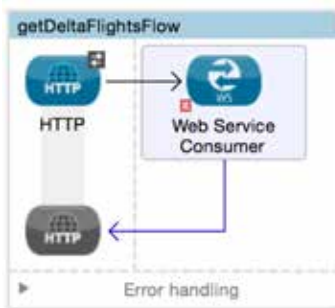


Create a new flow with an HTTP Listener connector endpoint

3. Return to apessentials.xml.
4. Drag out another HTTP connector and drop it in the canvas above the existing flows.
5. Give the flow a new name of getDeltaFlightsFlow.
6. In the Properties view for the endpoint, set the connector configuration to the existing HTTP_Listener_Configuration.
7. Set the path to /delta.
8. Set the allowed methods to GET.

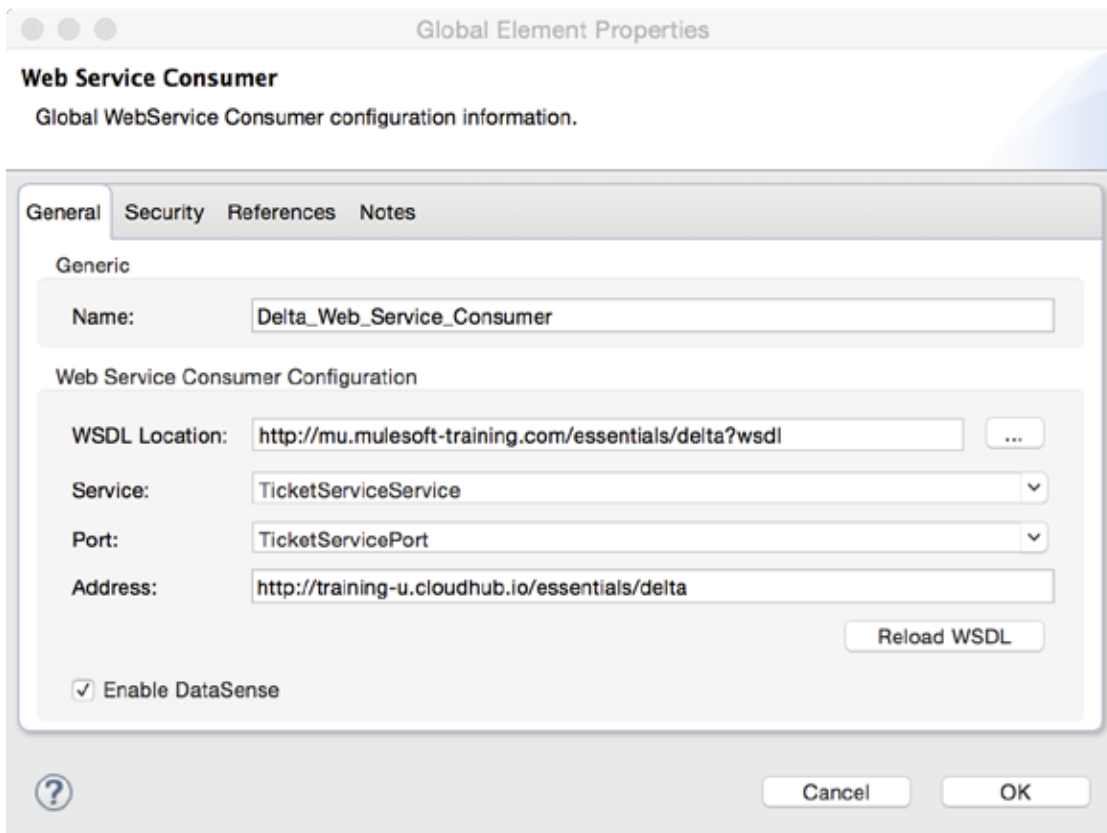
Add a Web Service Consumer connector endpoint

9. Drag out a Web Service Consumer connector and drop it in the process section of the flow.

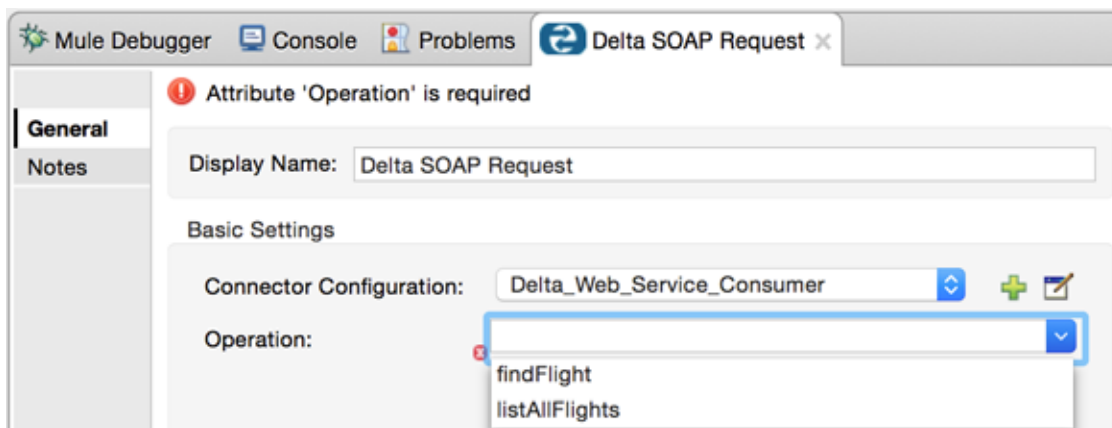


10. In the Properties view, set the display name to Delta SOAP Request.
11. Click the Add button next to connector configuration.
12. In the Global Element Properties dialog box, change the name to Delta_Web_Service_Consumer.
13. Set the WSDL Location to the Delta SOAP web service WSDL listed in the course snippets.txt file.

- Wait for the WSDL to be parsed and the service, port, and address fields to be automatically populated.



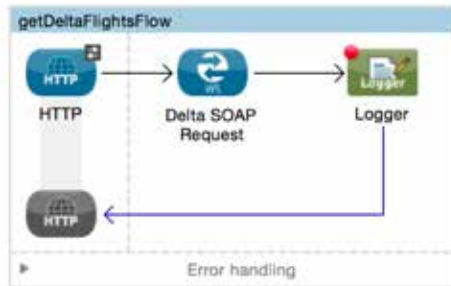
- Click OK.
- Switch to the Global Elements view and see the new global configuration element.
- Return to the Message Flow view.
- In the Delta SOAP Request Properties view, click the operation drop-down button; you should see all of the web service operations listed.



- Select the listAllFlights operation.

Debug the application

20. Add a Logger component after the Delta SOAP Request endpoint.
21. In the Logger Properties view, set the message to #[payload].
22. Add a breakpoint to the Logger.



23. Save the file and debug the application.
24. Make a request to <http://localhost:8081/delta>.
25. In the Mule Debugger view, drill-down into the payload variable.

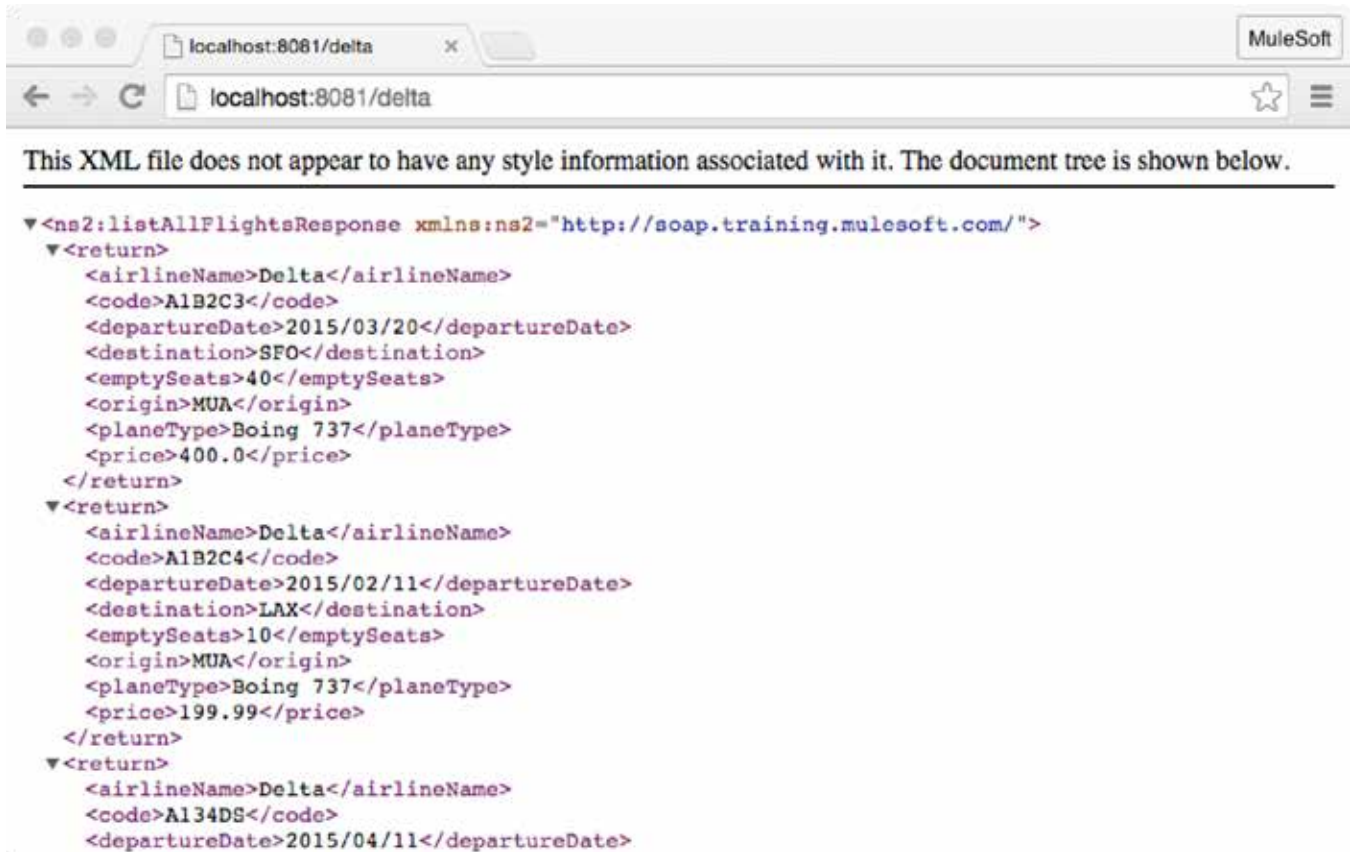
The screenshot shows the 'Mule Debugger View' window. The 'payload' variable is expanded, showing its structure. The 'scope' variable is also expanded, showing its content. The 'attributes' variable is expanded, showing its content.

Name	Value	Type
Message		org.mule.DefaultMuleMessage
Message Processor	Logger	org.mule.api.processor.LoggerMes...
payload	org.mule.module.ws.consumer.Namesp...	org.apache.cxf.staxutils.DepthXML...
depth	0	java.lang.Integer
reader	org.mule.module.ws.consumer.Namesp...	org.mule.module.ws.consumer.Na...
allocator	javanet.staxutils.events.EventAllocato...	javanet.staxutils.events.EventAllocato...
first	false	java.lang.Boolean
namespaces	[xmlns:ns2="http://soap.training.muleso...	java.util.ArrayList
nsBlacklist	[http://schemas.xmlsoap.org/soap/enve...	java.util.ArrayList
reader	org.mule.module.cxf.support.StreamClo...	org.mule.module.cxf.support.StreamClo...
scope	[<soap:Envelope xmlns:soap="http://sc...	java.util.ArrayList
0	<soap:Envelope xmlns:soap="http://sch...	javanet.staxutils.events.StartEleme...
1	<soap:Body>	javanet.staxutils.events.StartEleme...
2	<ns2:listAllFlightsResponse xmlns:ns2=...	javanet.staxutils.events.StartEleme...
attributes	null	java.util.Map
location	[row,col {unknown-source}]: [1,82]	com.ctc.wstx.io.WstxInputLocation
name	{http://soap.training.mulesoft.com/}listA...	javax.xml.namespace.QName

26. Step to the end of the application.
27. Look at the console; you should see the type of object listed.

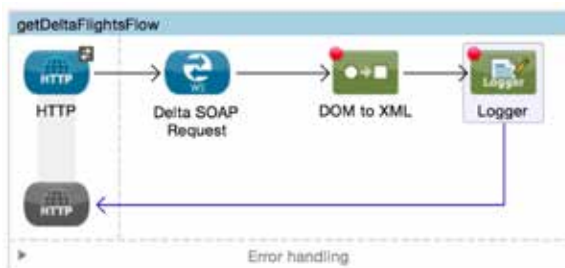
```
INFO 2015-08-19 12:38:25,676 [[apessentials].HTTP_Listener_Configuration.worker.01] org.mule.api.process
or.LoggerMessageProcessor: org.mule.module.ws.consumer.NamespaceRestorerXMLStreamReader@2db53627
```

28. Return to the browser; you should see the XML SOAP response displayed there.



Display the payload as a String

29. Add a DOM to XML transformer before the Logger.

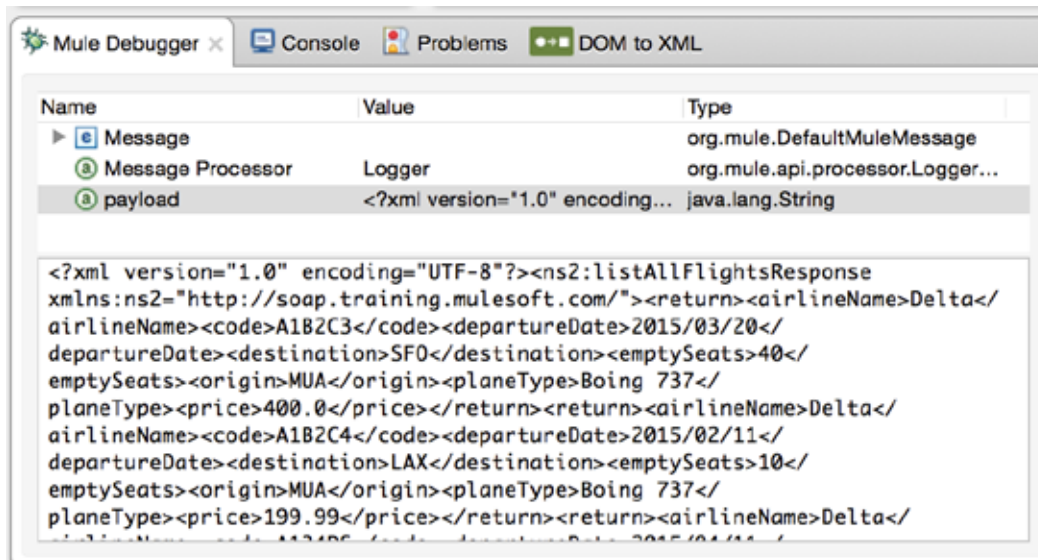


Test the application

30. Save the file to redeploy the application.

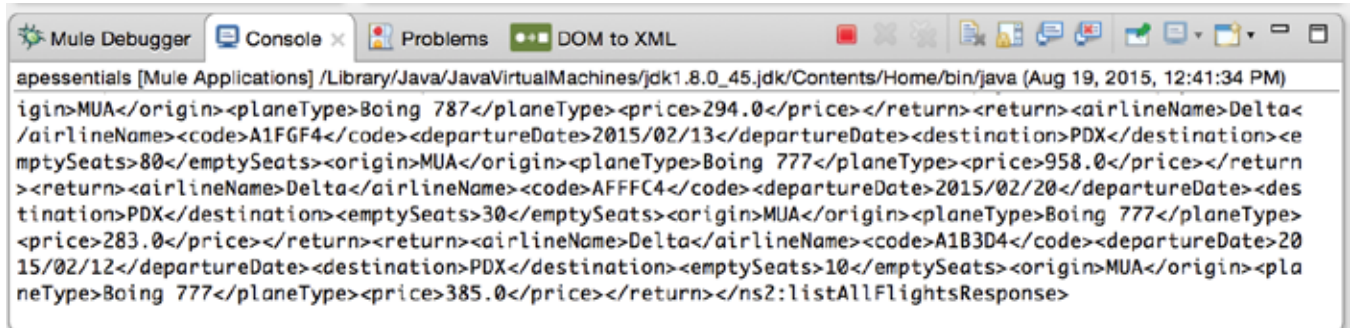
31. Make another request to <http://localhost:8081/delta>.

32. Step to the Logger; you should see the payload is now a String.



33. Resume the application.

34. Look at the console; you should see the XML SOAP response displayed there.



35. Stop the runtime.