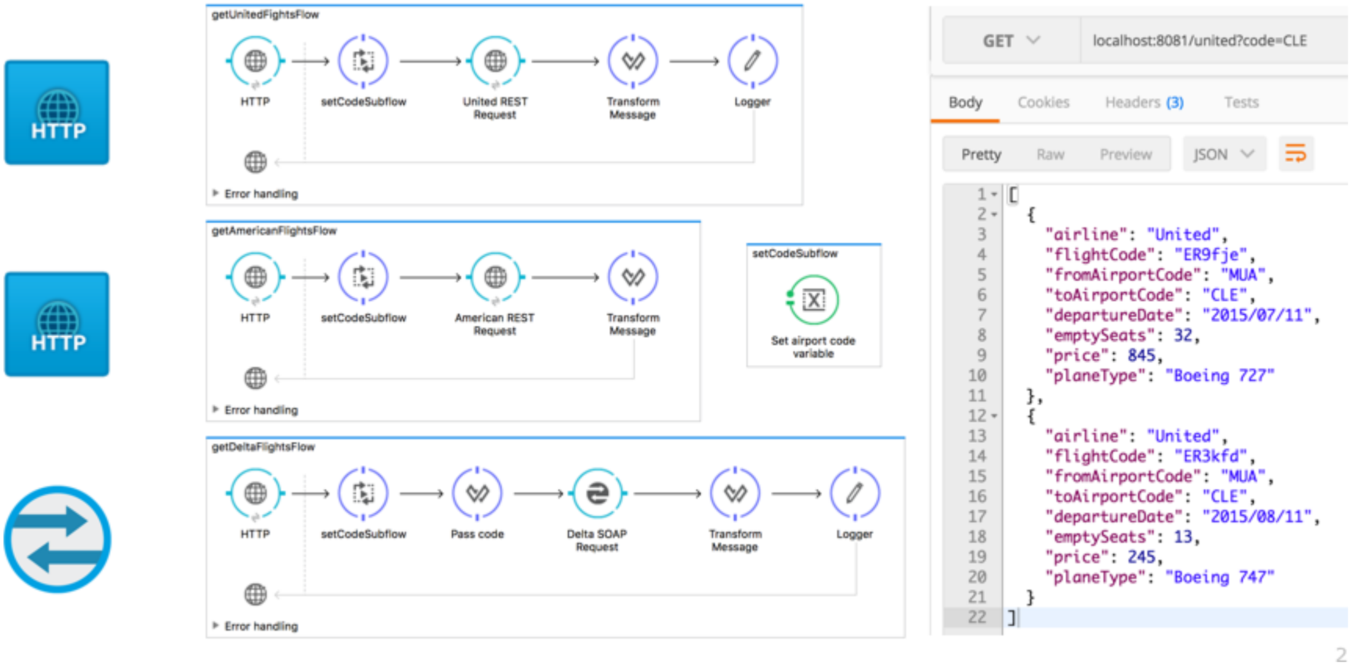
Module 8: Consuming Web Services



```
[
  {
    "airline": "United",
    "flightCode": "ER9fje",
    "fromAirportCode": "MUA",
    "toAirportCode": "CLE",
    "departureDate": "2015/07/11",
    "emptySeats": 32,
    "price": 845,
    "planeType": "Boeing 727"
  },
  {
    "airline": "United",
    "flightCode": "ER3kfd",
    "fromAirportCode": "MUA",
    "toAirportCode": "CLE",
    "departureDate": "2015/08/11",
    "emptySeats": 13,
    "price": 245,
    "planeType": "Boeing 747"
  }
]
```

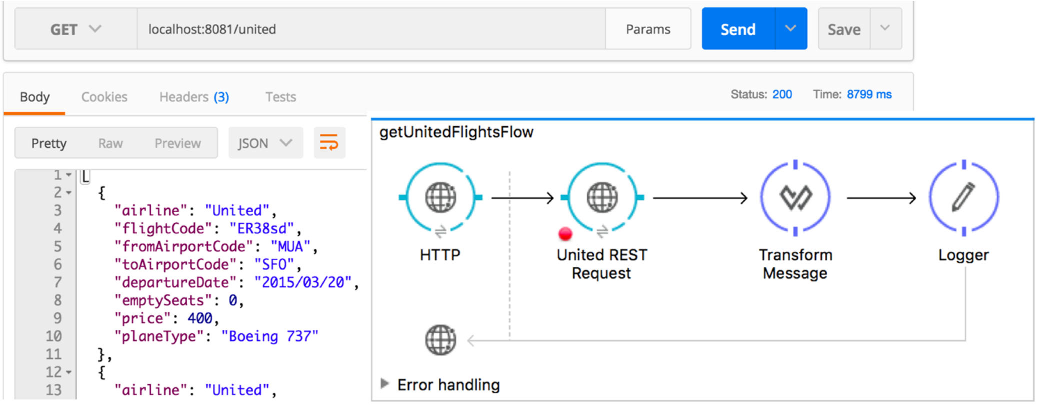
At the end of this module, you should be able to:

- Consume RESTful web services with and without parameters.
- Consume RESTful web services that have RAML definitions.
- Consume SOAP web services.
- Use DataWeave to pass parameters to SOAP web services.

Walkthrough 8-1: Consume a RESTful web service

In this walkthrough, you consume a RESTful web service that returns a list of all United flights as JSON. You will:

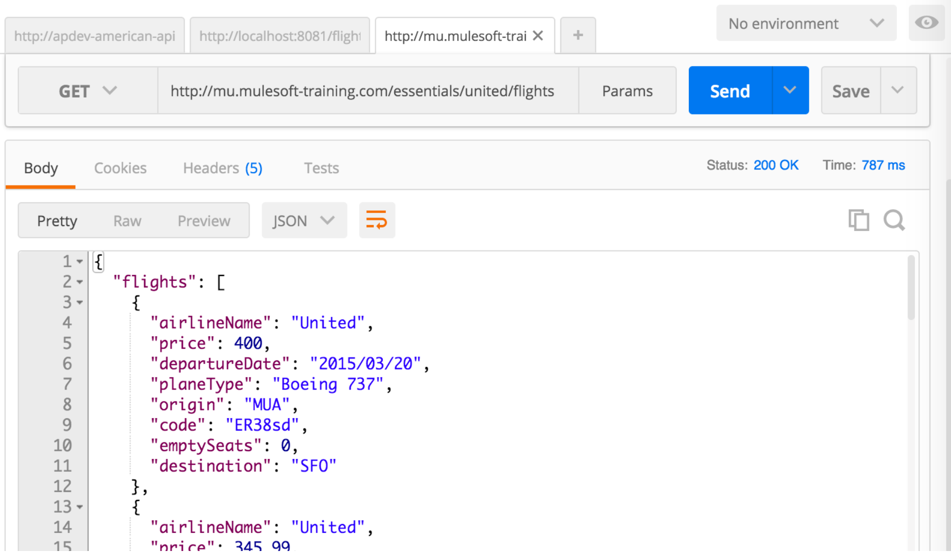
- Create a new flow to call a RESTful web service.
- Use an HTTP Request endpoint to consume a RESTful web service.
- Use DataWeave to transform the JSON response into JSON specified by an API.



```
[
  {
    "airline": "United",
    "flightCode": "ER38sd",
    "fromAirportCode": "MUA",
    "toAirportCode": "SFO",
    "departureDate": "2015/03/20",
    "emptySeats": 0,
    "price": 400,
    "planeType": "Boeing 737"
  },
  {
    "airline": "United",
```

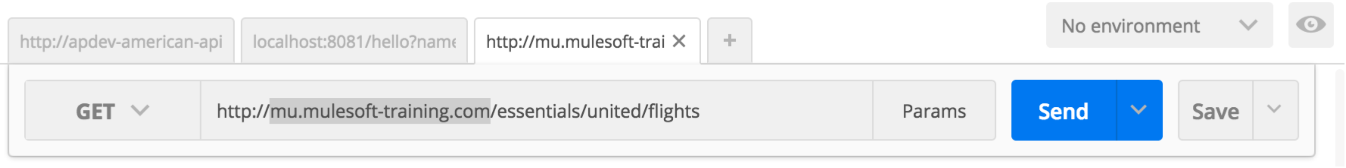
Make a request to the web service

1. Return to the course snippets.txt file.
2. Locate and copy the United RESTful web service URL.
3. In Postman, make a new tab.
4. Make a GET request to this URL; you should see JSON data for the United flights as a response.



```
{
  "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
```

5. Look at the destination values; you should see SFO, LAX, CLE, PDX, and PDF.
6. Copy the host name from the URL; this should be something like mu.mulesoft-training.com.



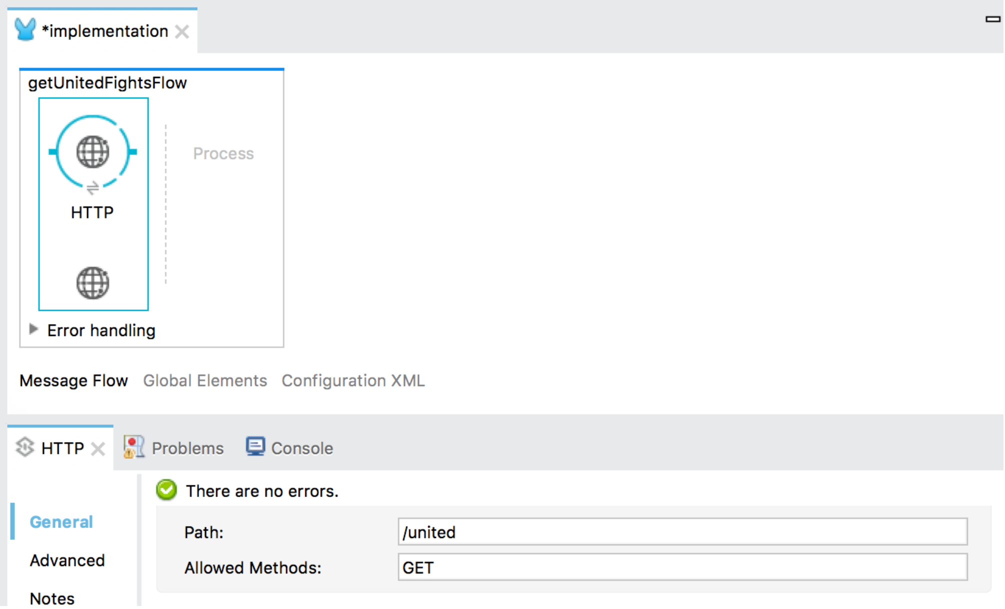
GET http://mu.mulesoft-training.com/essentials/united/flights

Add a new flow with an HTTP Listener endpoint

7. Return to implementation.xml in Anypoint Studio.
8. Drag out an HTTP connector and drop it in the canvas.
9. Double-click the name of the flow in the canvas and give it a new name of getUnitedFlightsFlow.

Configure the HTTP Listener endpoint

10. In the HTTP properties view, set the connector configuration to the existing HTTP_Listener_Configuration.
11. Set the path to /united.
12. Set the allowed methods to GET.



getUnitedFightsFlow

There are no errors.

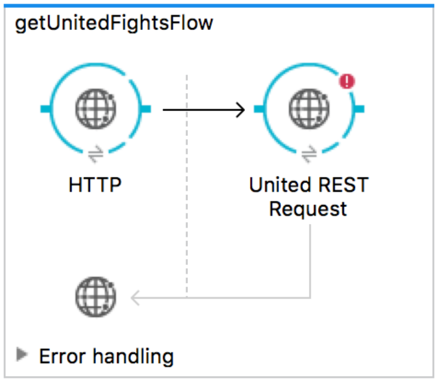
Path: /united

Allowed Methods: GET

Add an HTTP Request endpoint

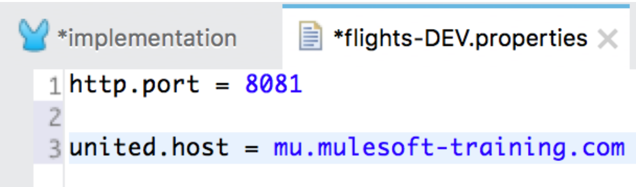
13. Drag out another HTTP connector and drop it into the process section of the flow.

14. Change the HTTP Request endpoint display name to United REST Request.



Configure the HTTP Request connector

15. Return to flights-DEV.properties in src/main/resources.
16. Create a property called united.host and set it equal to the value you copied from the URL.



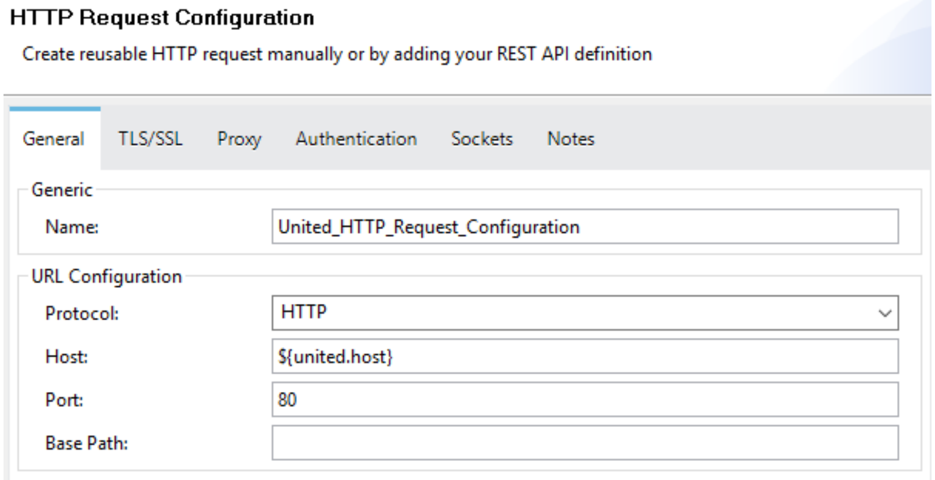
```
http.port = 8081

united.host = mu.mulesoft-training.com
```

17. Save the file.
18. Return to global.xml.
19. In the Global Elements view, click Create.
20. In the Choose Global Type dialog box, select Connector Configuration > HTTP Request Configuration and click OK.

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

- Name: United_REST_Request_Configuration
- Host: ${united.host}
- Port: *80*
- Base Path: *Leave blank*



HTTP Request Configuration

Create reusable HTTP request manually or by adding your REST API definition

Generic

Name: United_HTTP_Request_Configuration

URL Configuration

Protocol: HTTP

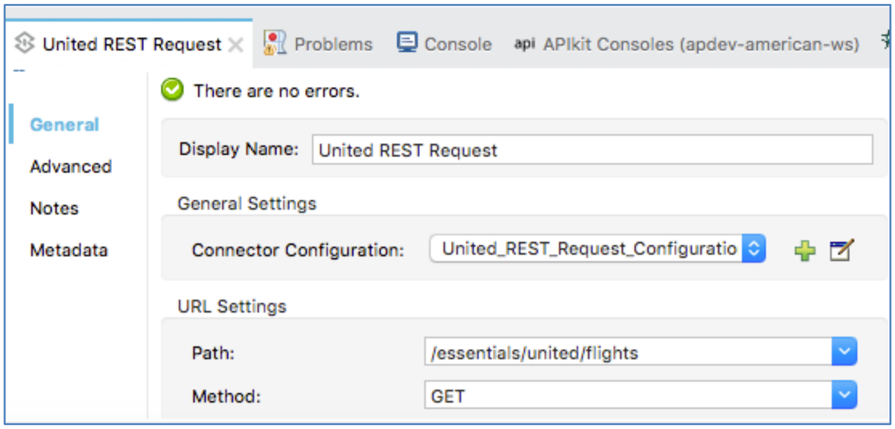
Host: ${united.host}

Port: 80

Base Path:

Configure the HTTP Request endpoint

22. Return to implementation.xml.
23. In the United REST Request properties view, set the connector configuration to the existing United_REST_Request_Configuration.
24. Set the path to /essentials/united/flights.
25. Set the method to GET.



There are no errors.

Display Name: United REST Request

General Settings

Connector Configuration: United_REST_Request_Configuratio

URL Settings

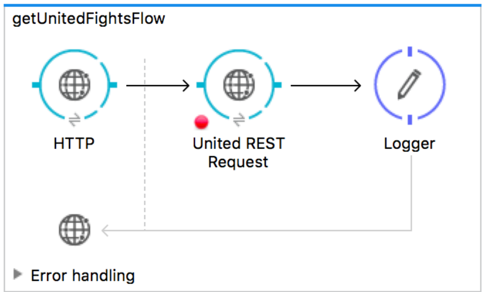
Path: /essentials/united/flights

Method: GET

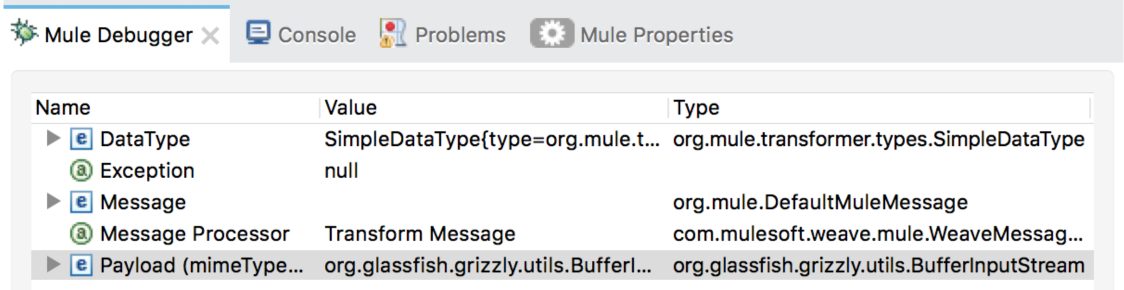
Test the application

26. Make sure the United REST Request endpoint has a breakpoint.

27. Add a Logger after the United REST Request endpoint.

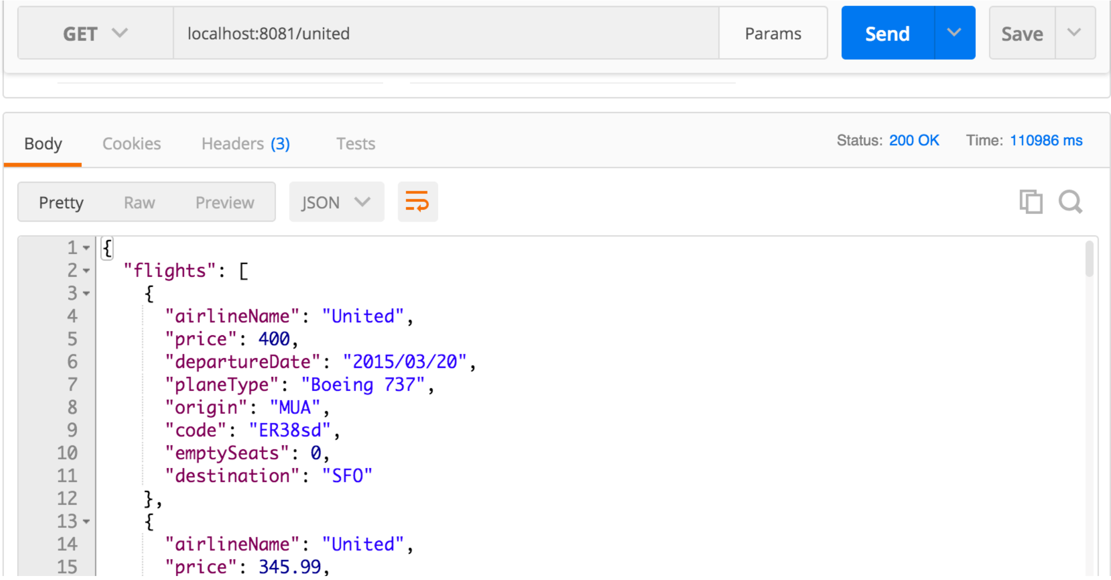


28. Debug the project.
29. In Postman, return to the tab with the localhost request.
30. Make a request to http://localhost:8081/united.
31. In the Mule Debugger, step to the Logger and look at the payload; it should be of type BufferInputStream.



Name	Value	Type
▶ DataType	SimpleDataType{type=org.mule.t...	org.mule.transformer.types.SimpleDataType
Exception	null	
▶ Message		org.mule.DefaultMuleMessage
Message Processor	Transform Message	com.mulesoft.weave.mule.WeaveMessag...
▶ Payload (mimeType...	org.glassfish.grizzly.utils.Bufferl...	org.glassfish.grizzly.utils.BufferInputStream

32. Step through the application.
33. Return to Postman; you should see the JSON flight data returned.
34. Examine the data structure of the JSON response.



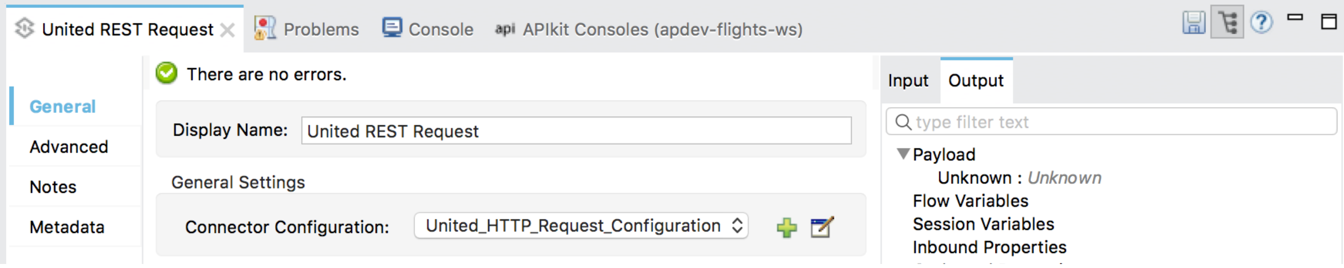
localhost:8081/united

Status: 200 OK Time: 110986 ms

```
{
  "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,
```

Add metadata for the United REST Request response

35. Return to Anypoint Studio and switch perspectives.
36. In the Properties view for the United REST Request, click the Output tab; you should see the payload is of type unknown.



There are no errors.

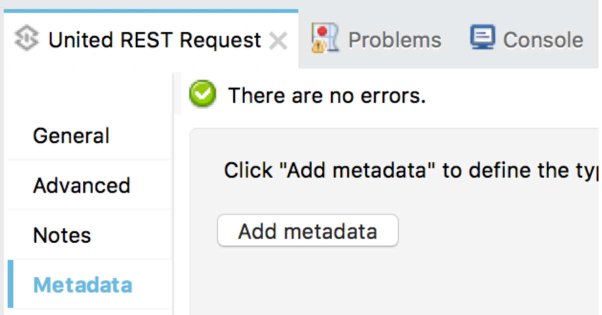
Display Name: United REST Request

General Settings

Connector Configuration: United_HTTP_Request_Configuration

Payload
Unknown : *Unknown*
Flow Variables
Session Variables
Inbound Properties

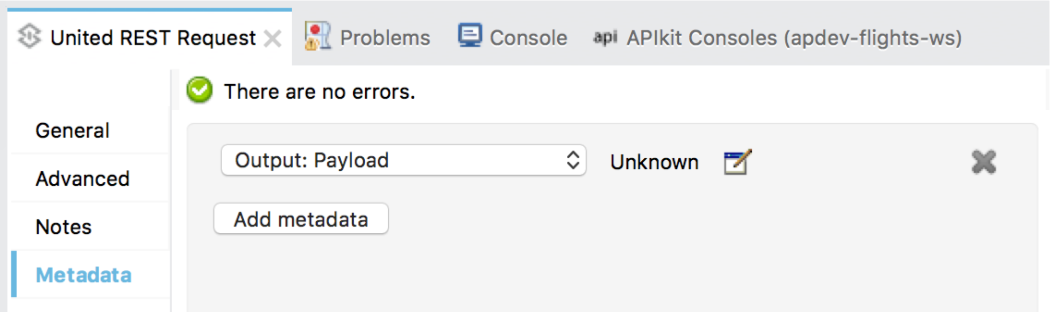
37. In the left-side navigation, click the Metadata link.
38. Click the Add metadata button.



There are no errors.

Click "Add metadata" to define the ty

39. Change the drop-down menu to Output: Payload.
40. Click the Edit button.

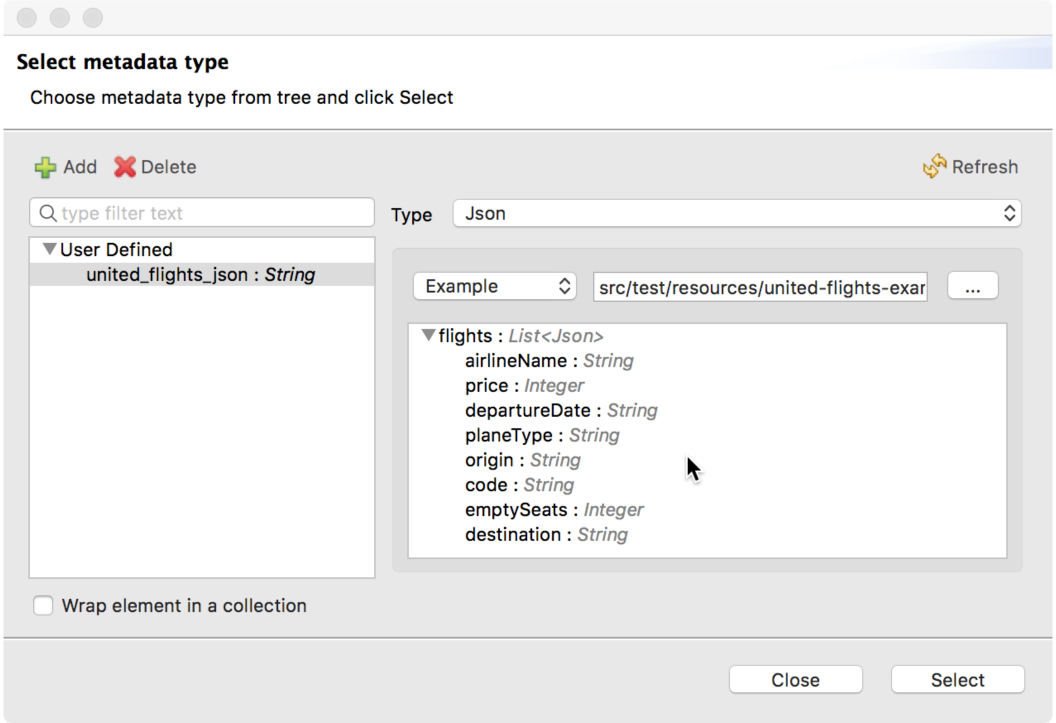


There are no errors.

Output: Payload Unknown

41. In the Select metadata type dialog box, click the Add button.
42. In the Create new type dialog box, set the type id to united_flights_json.
43. Click Create type.
44. In the Select metadata type dialog box, set the type to JSON.
45. Change the Schema selection to Example.
46. Click the browse button and navigate to the projects's src/test/resources folder.

47. Select united_flights-example.json and click Open; you should see the example data for the metadata type.



Select metadata type

Choose metadata type from tree and click Select

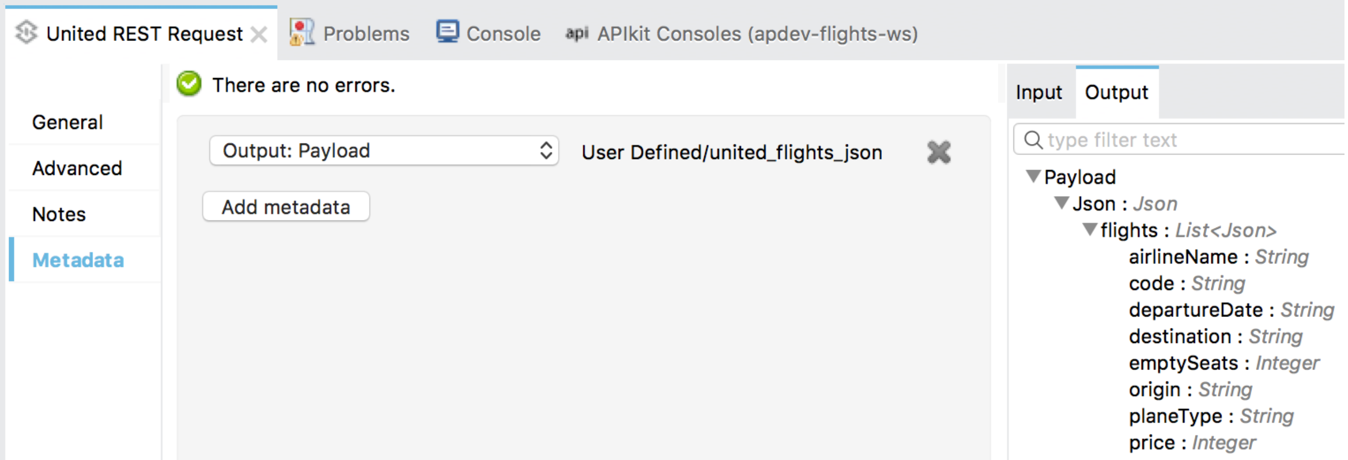
Type: Json

Example: src/test/resources/united-flights-exar

```
▼flights : List<Json>
    airlineName : String
    price : Integer
    departureDate : String
    planeType : String
    origin : String
    code : String
    emptySeats : Integer
    destination : String
```

☐ Wrap element in a collection

48. Click Select.
49. In the Properties view for the United REST Request, click the Output tab; you should see the payload is of type Json.
50. Expand the Json object and its flights object; you should see the properties of the return objects.



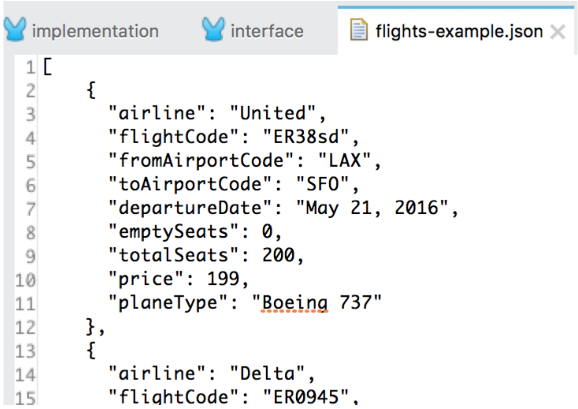
There are no errors.

Output: Payload — User Defined/united_flights_json

```
▼Payload
  ▼Json : Json
    ▼flights : List<Json>
        airlineName : String
        code : String
        departureDate : String
        destination : String
        emptySeats : Integer
        origin : String
        planeType : String
        price : Integer
```

Review the structure for the desired JSON response

51. Open flights-example.json in src/test/resources and review the sample.

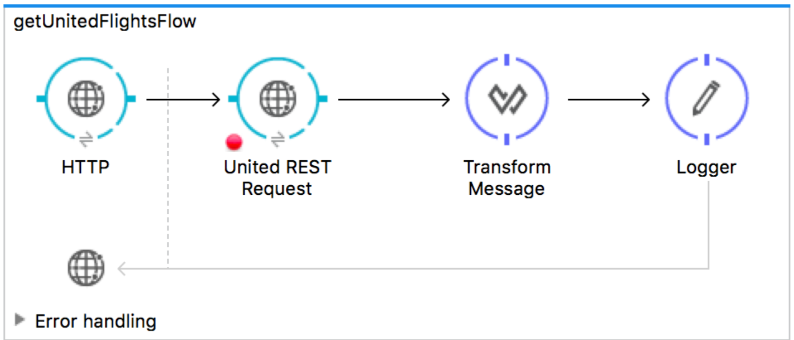


```
[
    {
      "airline": "United",
      "flightCode": "ER38sd",
      "fromAirportCode": "LAX",
      "toAirportCode": "SFO",
      "departureDate": "May 21, 2016",
      "emptySeats": 0,
      "totalSeats": 200,
      "price": 199,
      "planeType": "Boeing 737"
    },
    {
      "airline": "Delta",
      "flightCode": "ER0945",
```

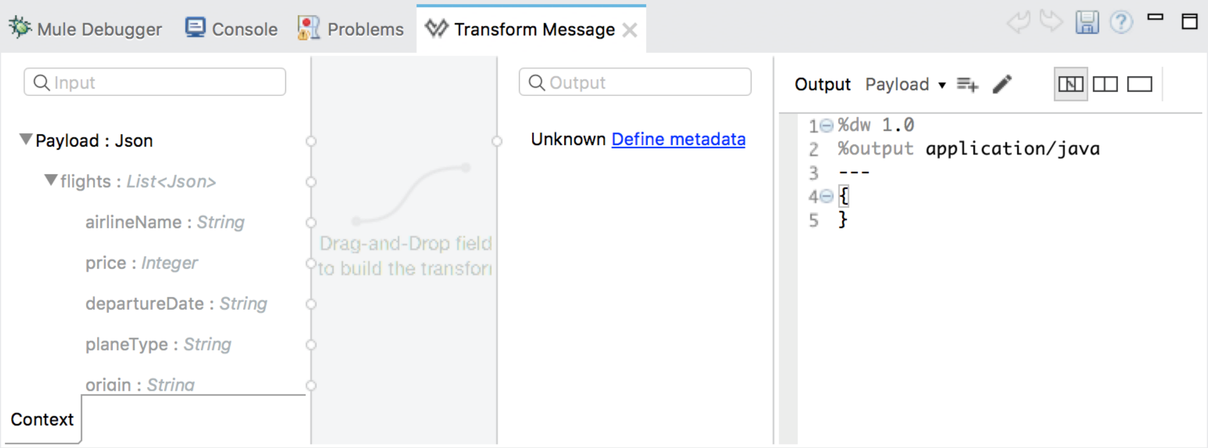
Add a Transform Message component

52. Return to implementation.xml.

53. Add a Transform Message component after the United REST Request endpoint.



54. Look at the input section; you should see metadata for the output from the United REST Request.



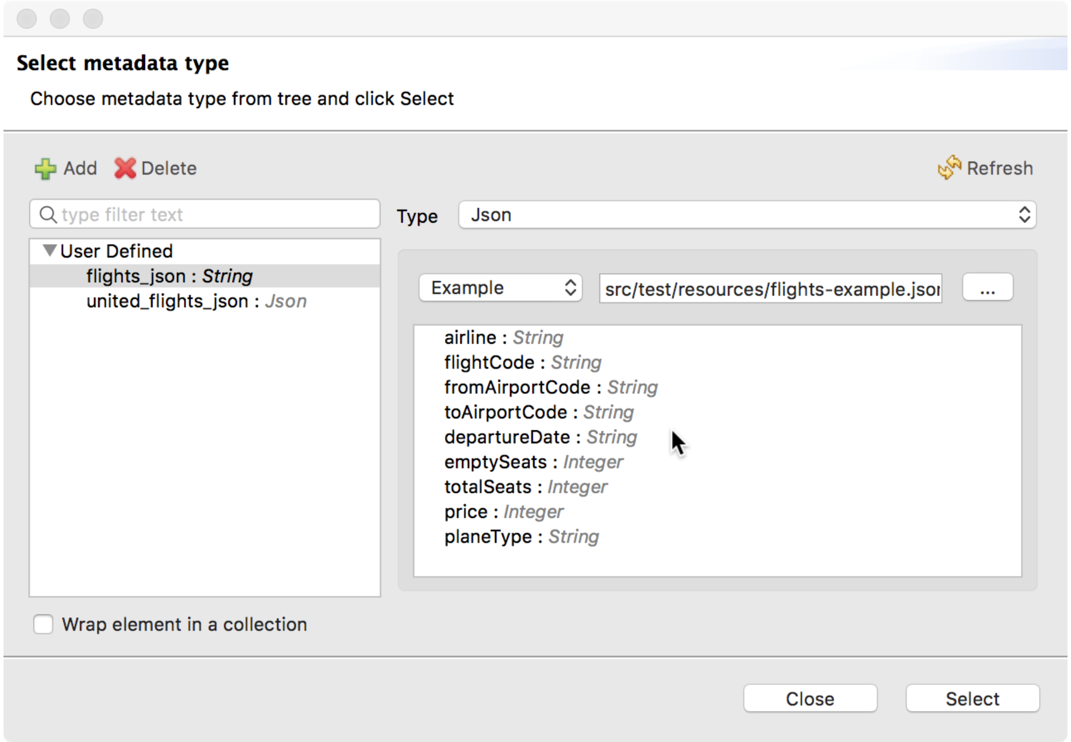
- Payload : Json
  - flights : *List<Json>*
    - airlineName : *String*
    - price : *Integer*
    - departureDate : *String*
    - planeType : *String*
    - origin : *String*

```
%dw 1.0
%output application/java
---
{
}
```

Add output metadata for the transformation

55. In the output section of the Transform Message properties view, click the Define metadata link.
56. In the Select metadata type dialog box, click the Add button.
57. In the Create new type dialog box, set the type id to flights_json.
58. Click Create type.
59. In the Select metadata type dialog box, set the type to JSON.
60. Change the Schema selection to Example.
61. Click the browse button and navigate to the projects's src/test/resources folder.
62. Select flights-example.json and click Open; you should see the example data for the metadata type.

    Note: Be sure to select the JSON file with flights plural, not singular.


Select metadata type

Choose metadata type from tree and click Select

Type: Json

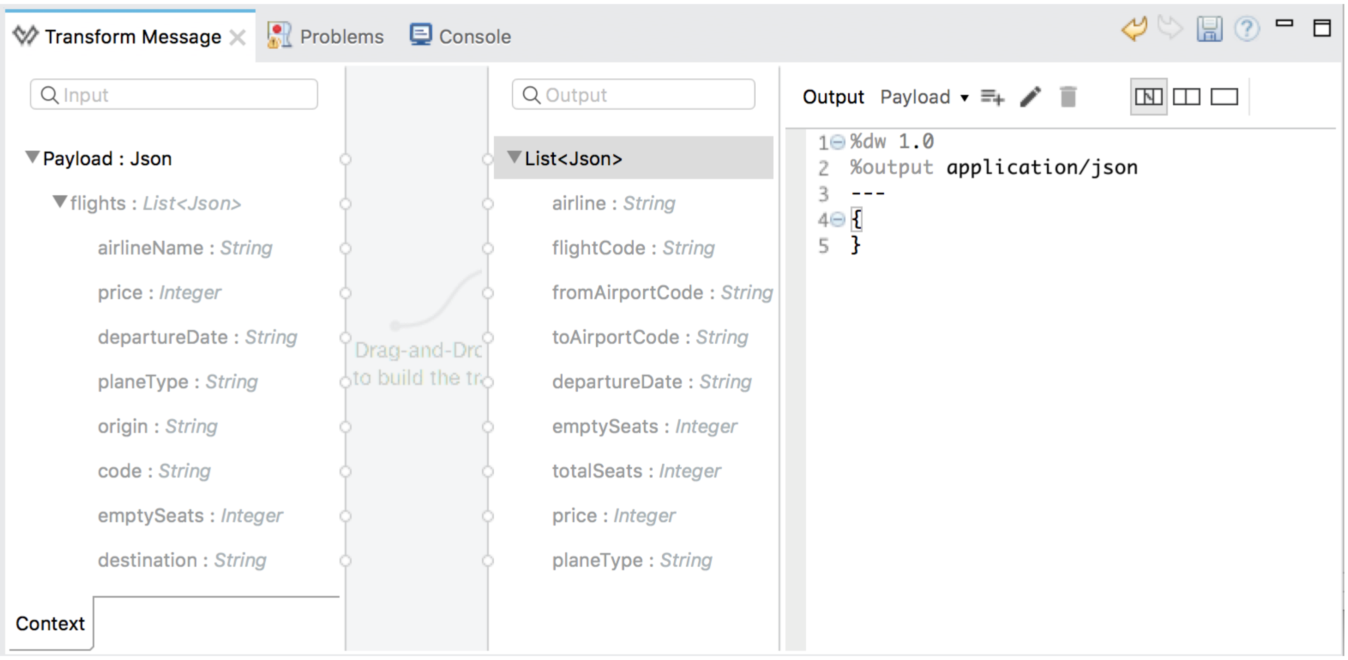
User Defined
- flights_json : *String*
- united_flights_json : *Json*

Example: src/test/resources/flights-example.jsor

airline : *String*
flightCode : *String*
fromAirportCode : *String*
toAirportCode : *String*
departureDate : *String*
emptySeats : *Integer*
totalSeats : *Integer*
price : *Integer*
planeType : *String*

☐ Wrap element in a collection

63. Click Select; you should now see output metadata in the output section of the Transform Message properties view.

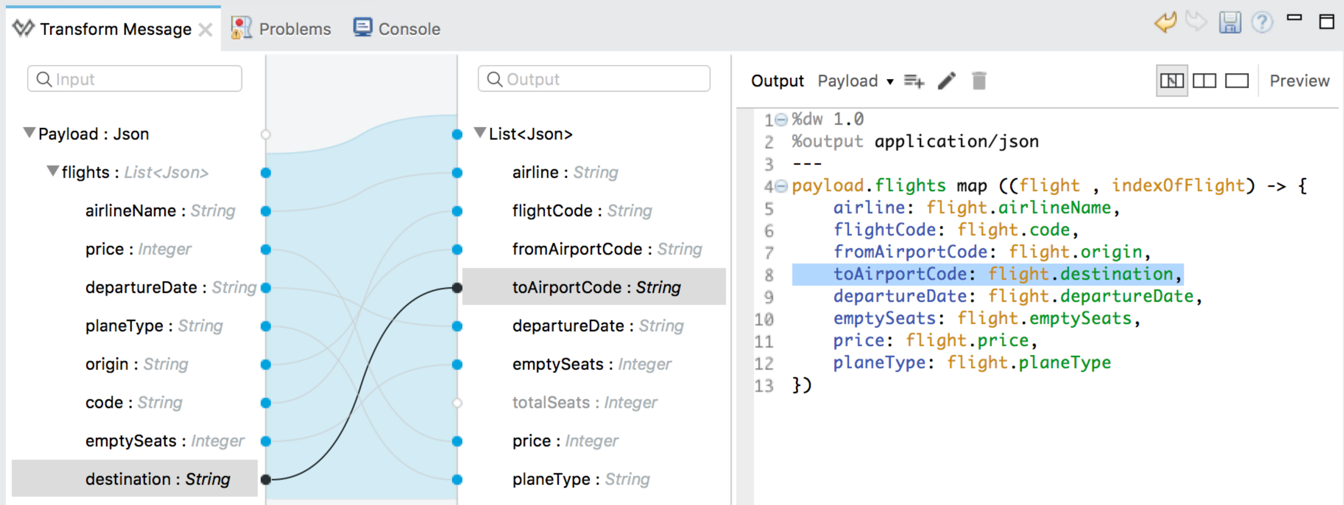


```
%dw 1.0
%output application/json
---
{
}
```

Create the transformation

64. Map fields by dragging them from the input section and dropping them on the corresponding field in the output section.

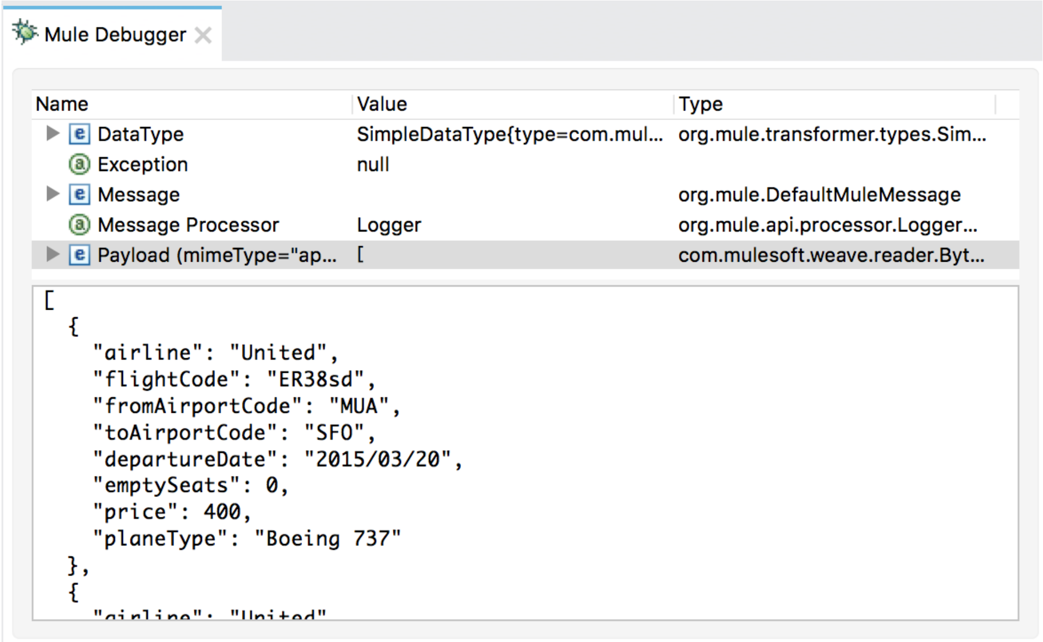
Note: There is no input field to map to totalSeats.



```
%dw 1.0
%output application/json
---
payload.flights map ((flight , indexOfFlight) -> {
    airline: flight.airlineName,
    flightCode: flight.code,
    fromAirportCode: flight.origin,
    toAirportCode: flight.destination,
    departureDate: flight.departureDate,
    emptySeats: flight.emptySeats,
    price: flight.price,
    planeType: flight.planeType
})
```

Test the application

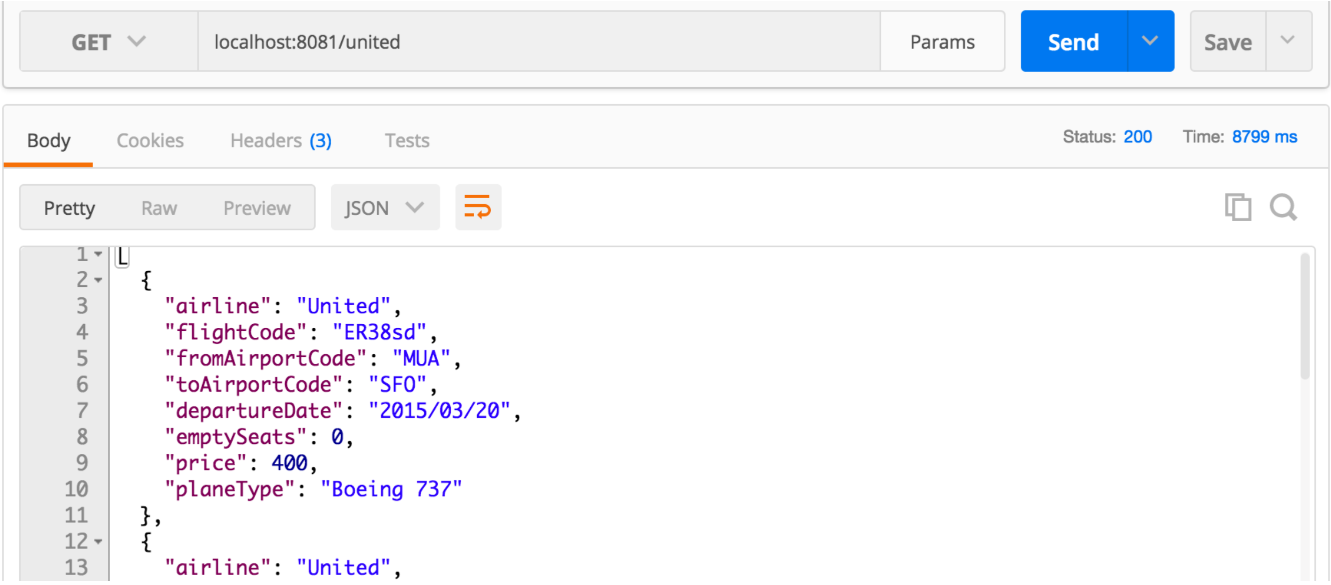
65. Debug the project.
66. In Postman, make another request to http://localhost:8081/united.
67. In the Mule Debugger, step to the Logger; you should see the payload is now a DataWeave ByteArraySeekableStream.



Name	Value	Type
▶ DataType	SimpleDataType{type=com.mul…	org.mule.transformer.types.Sim…
Exception	null	
▶ Message		org.mule.DefaultMuleMessage
Message Processor	Logger	org.mule.api.processor.Logger…
▶ Payload (mimeType="ap…	[	com.mulesoft.weave.reader.Byt…

```
[
  {
    "airline": "United",
    "flightCode": "ER38sd",
    "fromAirportCode": "MUA",
    "toAirportCode": "SFO",
    "departureDate": "2015/03/20",
    "emptySeats": 0,
    "price": 400,
    "planeType": "Boeing 737"
  },
  {
    "airline": "United",
```

68. Step through the application.
69. Return to Postman; you should see the flight data with the different JSON structure.



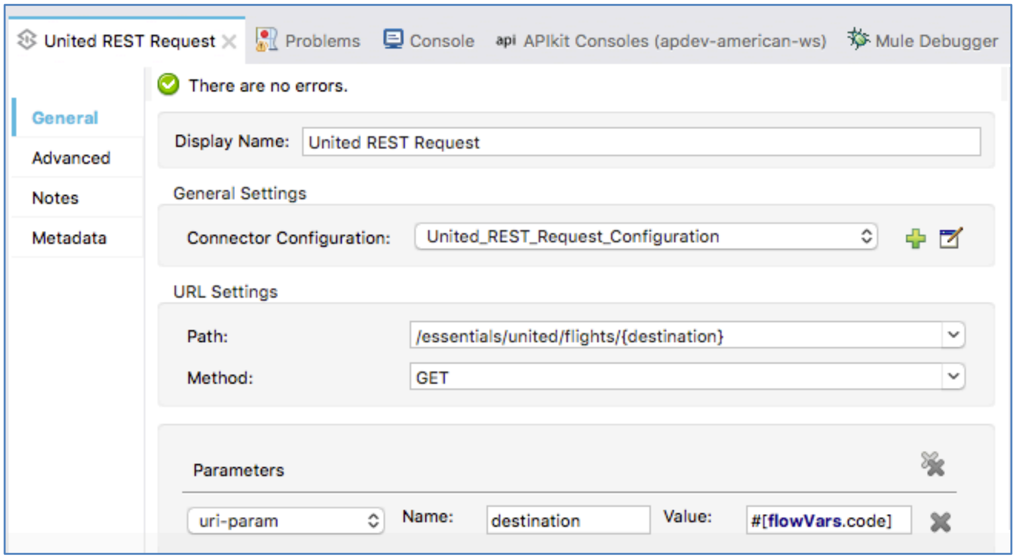
```
[
  {
    "airline": "United",
    "flightCode": "ER38sd",
    "fromAirportCode": "MUA",
    "toAirportCode": "SFO",
    "departureDate": "2015/03/20",
    "emptySeats": 0,
    "price": 400,
    "planeType": "Boeing 737"
  },
  {
    "airline": "United",
```

70. Return to Anypoint Studio, stop the project, and switch to the Mule Design perspective.

Walkthrough 8-2: Pass arguments to a RESTful web service

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

- Modify the HTTP Request endpoint to use a URI parameter for the destination.
- Set the destination to a static value.
- Create a flow variable to store the value of a query parameter with an airport code value.
- Set the destination to the dynamic value of this flow variable.



There are no errors.

Display Name: United REST Request

General Settings

Connector Configuration: United_REST_Request_Configuration

URL Settings

Path: /essentials/united/flights/{destination}

Method: GET

Parameters

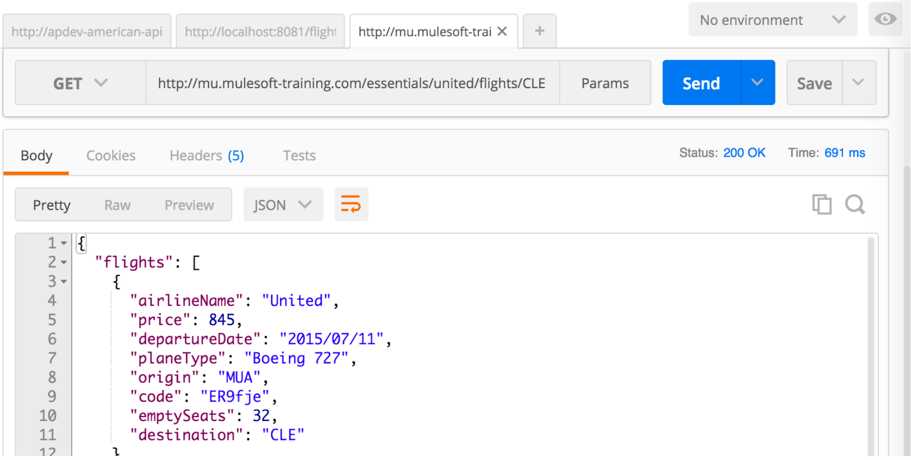
uri-param Name: destination Value: #[flowVars.code]

Make a request to the web service specifying a destination

1. Return to Postman and click the third tab, the one with the request to the United web service.
2. In the URL field, add the destination CLE as a URI parameter: http://mu.mulesoft-training.com/essentials/united/flights/CLE.

   Note: The API you are building for flights.raml has a query parameter called code for the destination; this existing United web service uses a URI parameter for the destination.

3. Send the request; you should see JSON data for only the flights to CLE.



http://mu.mulesoft-training.com/essentials/united/flights/CLE

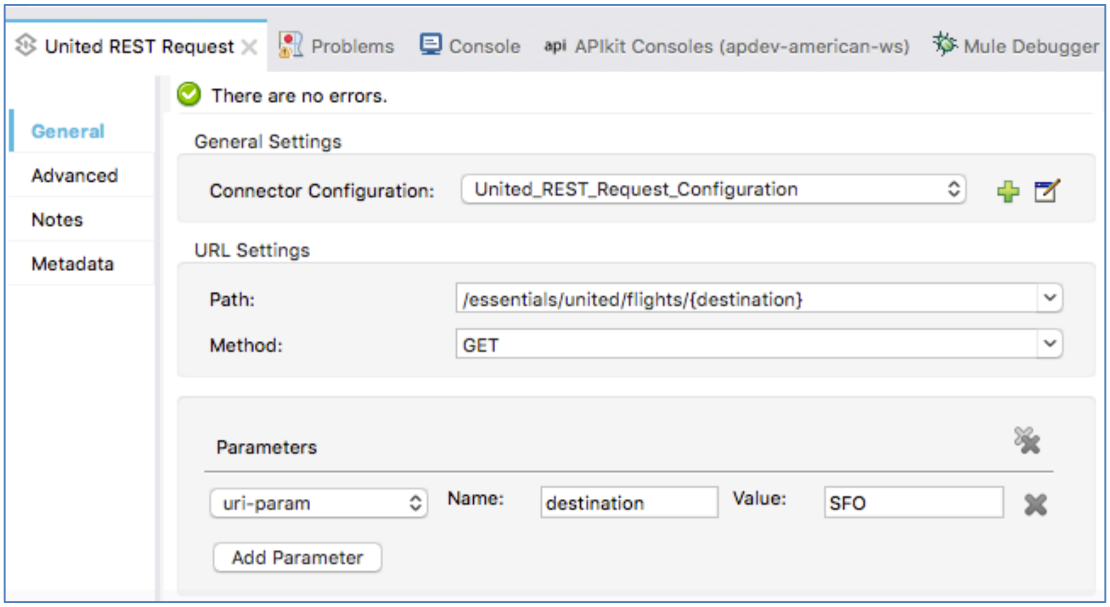
Status: 200 OK Time: 691 ms

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

4. Make additional requests for destinations of LAX, SFO, PDX, or PDF.

Add a URI parameter with a static value

5. Return to implementation.xml in Anypoint Studio.
6. Double-click the United REST Request endpoint.
7. In the Properties view, locate the parameters section; there should be no parameters listed.
8. Change the United REST Request path to /essentials/united/flights/{destination}.
9. Click the Add Parameter button.
10. Select a parameter type of uri-param.
11. Set the name to destination.
12. Set the parameter value to SFO.



There are no errors.

General Settings

Connector Configuration: United_REST_Request_Configuration

URL Settings

Path: /essentials/united/flights/{destination}

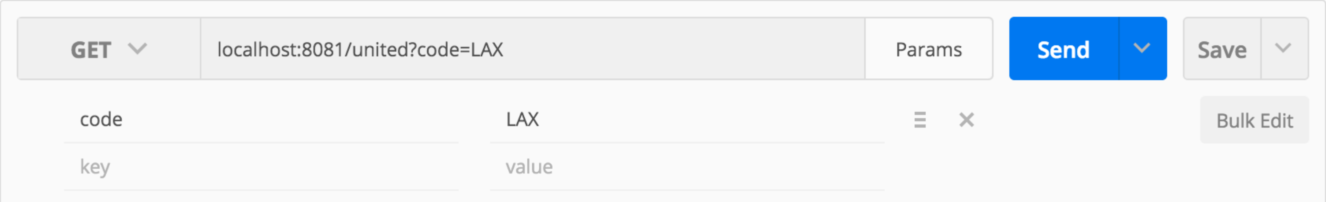
Method: GET

Parameters

uri-param Name: destination Value: SFO

Test the application

13. Run the project.
14. In Postman, return to the tab with the localhost requests.
15. Make a request to http://localhost:8081/united/; you should only get the flights to SFO.
16. Add a query parameter called code and set it to LAX.



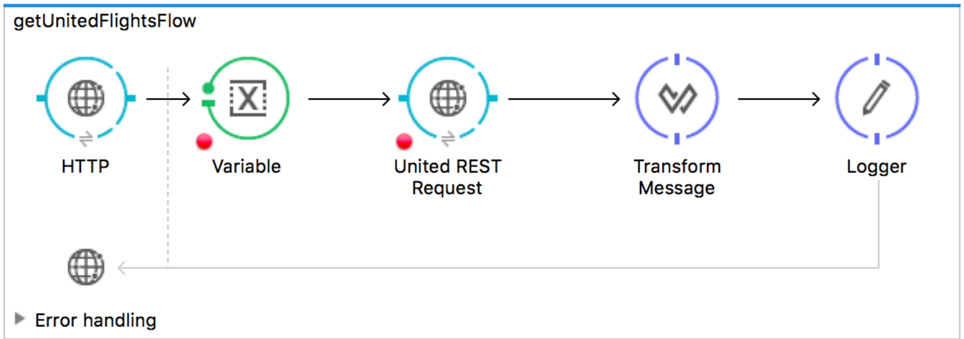
localhost:8081/united?code=LAX

key	value
code	LAX

17. Send the request; of course, you should still get only flights to SFO.

Create a variable to set the destination airport code

18. Return to Anypoint Studio.
19. Add a Variable transformer before the United REST Request endpoint.

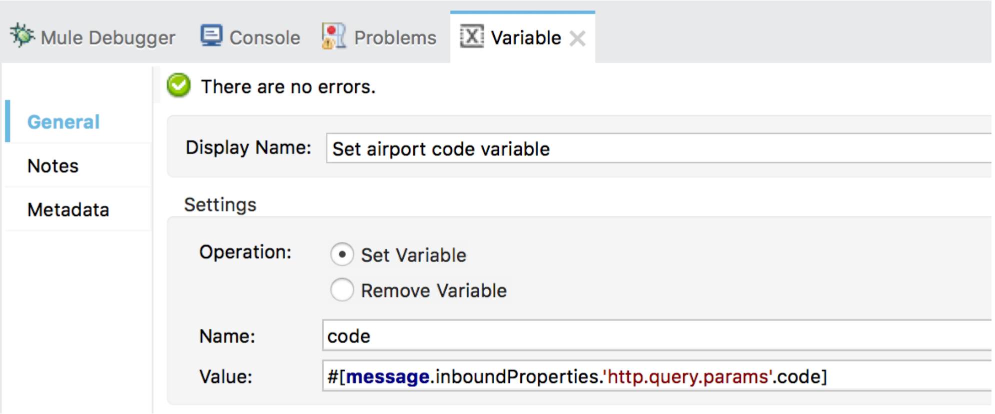


20. In the Variable Properties view, change the display name to Set airport code variable.

Set the operation to Set Variable and the name to code.

21. Set the value to a query parameter called code.

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



There are no errors.

Display Name: Set airport code variable

Settings

Operation: ● Set Variable ○ Remove Variable

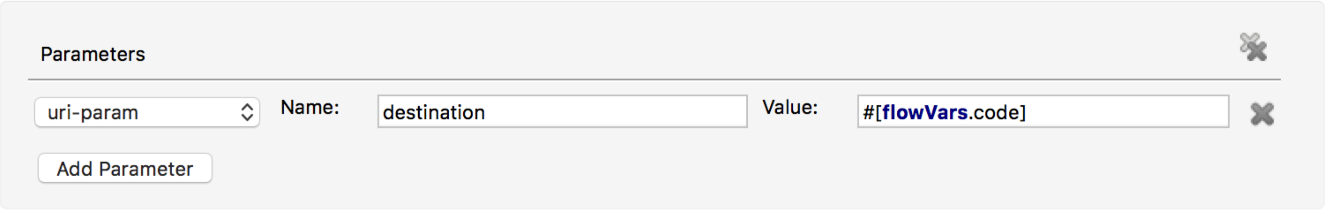
Name: code

Value: #[message.inboundProperties.'http.query.params'.code]

Change the REST request URI parameter to a dynamic value

22. In the United REST Request properties view, change the value of the uri-param from SFO to the value of the flow variable containing the airport code.

```
#[flowVars.code]
```

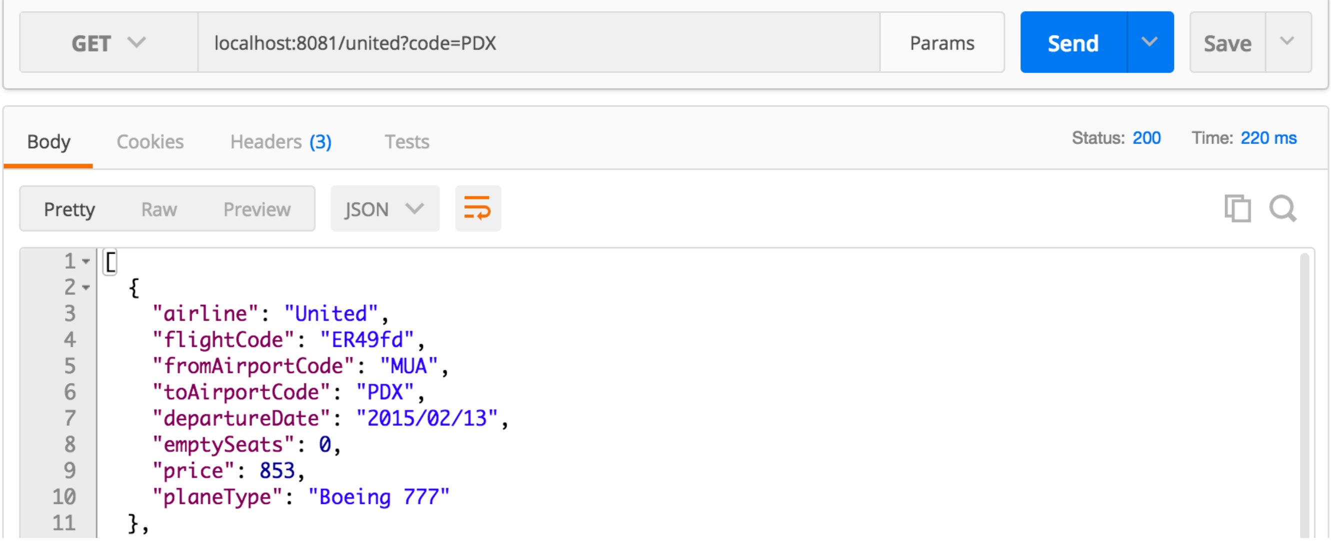


uri-param Name: destination Value: #[flowVars.code]

Test the application

23. Save and redeploy the application.
24. In Postman, send the same request again; this time you should only get flights to LAX.

25. Change the code to PDX and make the request; you should now see flights to PDX.

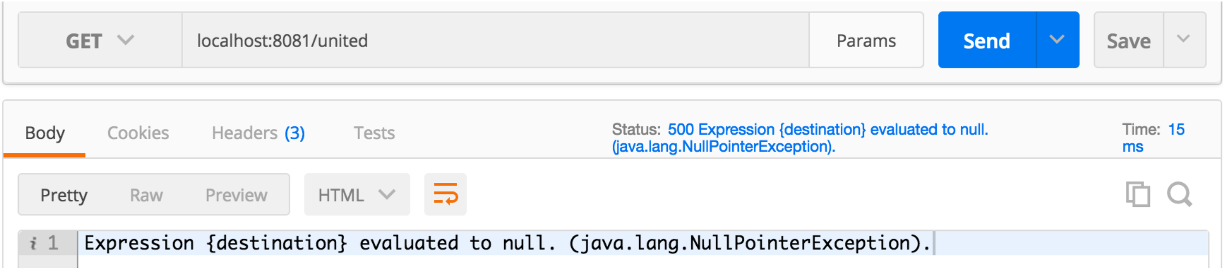


localhost:8081/united?code=PDX

Status: 200 Time: 220 ms

```
[
  {
    "airline": "United",
    "flightCode": "ER49fd",
    "fromAirportCode": "MUA",
    "toAirportCode": "PDX",
    "departureDate": "2015/02/13",
    "emptySeats": 0,
    "price": 853,
    "planeType": "Boeing 777"
  },
```

26. Remove the code parameter and make the request; you should get a 500 responses and an exception.



localhost:8081/united

Status: 500 Expression {destination} evaluated to null. (java.lang.NullPointerException). Time: 15 ms

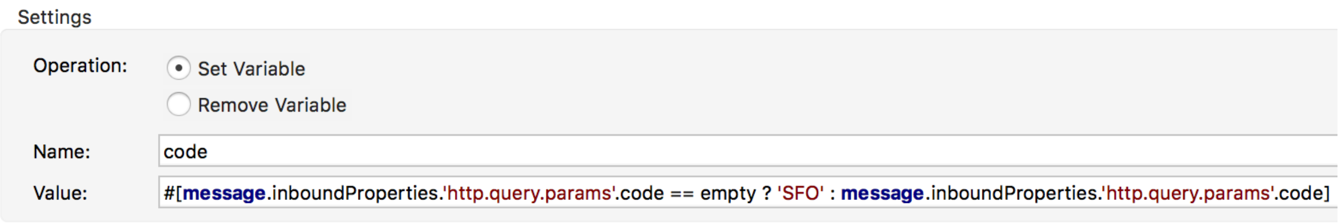
```
Expression {destination} evaluated to null. (java.lang.NullPointerException).
```

Modify the set airport code flow variable to assign a default value

27. Return to Anypoint Studio.

28. Modify the Set variable transformer to use a ternary expression to assign a default value of SFO if no query parameter is passed to the flow.

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



Settings

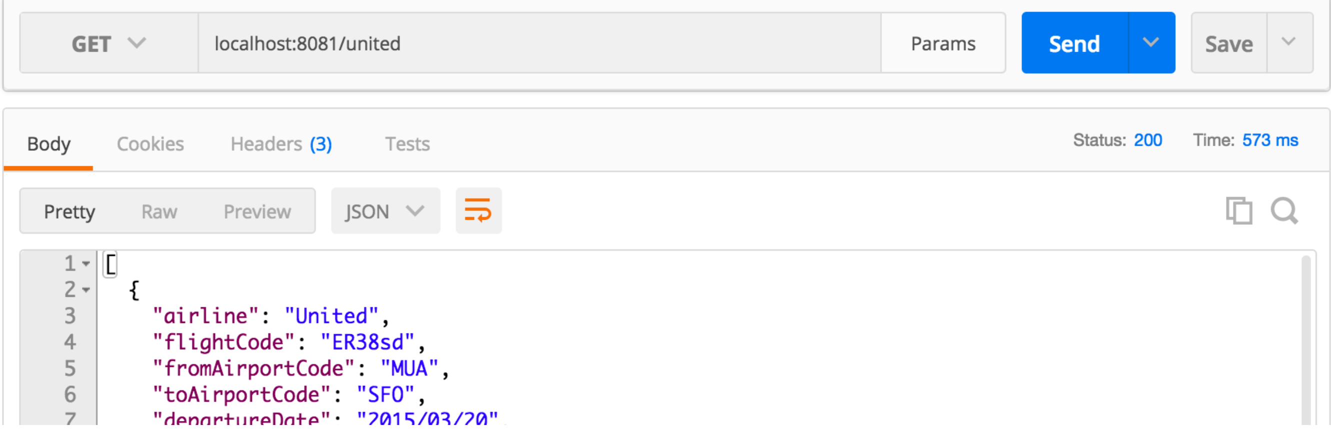
Operation: ● Set Variable ○ Remove Variable

Name: code

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

Test the application

29. Save and redeploy the application.
30. In Postman, send the same request again with no query parameter; this time you should get flights to SFO instead of an exception.



localhost:8081/united

Status: 200 Time: 573 ms

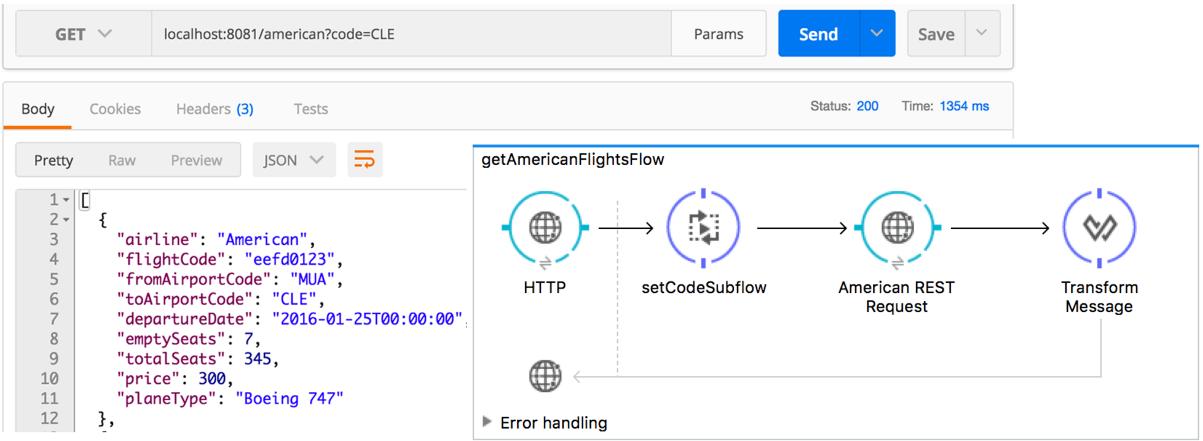
```
[
  {
    "airline": "United",
    "flightCode": "ER38sd",
    "fromAirportCode": "MUA",
    "toAirportCode": "SFO",
    "departureDate": "2015/03/20",
```

31. Return to Anypoint Studio and stop the project.

Walkthrough 8-3: Consume a RESTful web service that has a RAML definition

In this walkthrough, you consume the American flights RESTful web service that you built and deployed to the cloud. You will:

- Create a new flow to call a RESTful web service that has a RAML definition.
- Select the web service resource from the list provided by Anypoint Studio from the RAML file.
- Use DataWeave to transform the JSON response into JSON specified by an API.



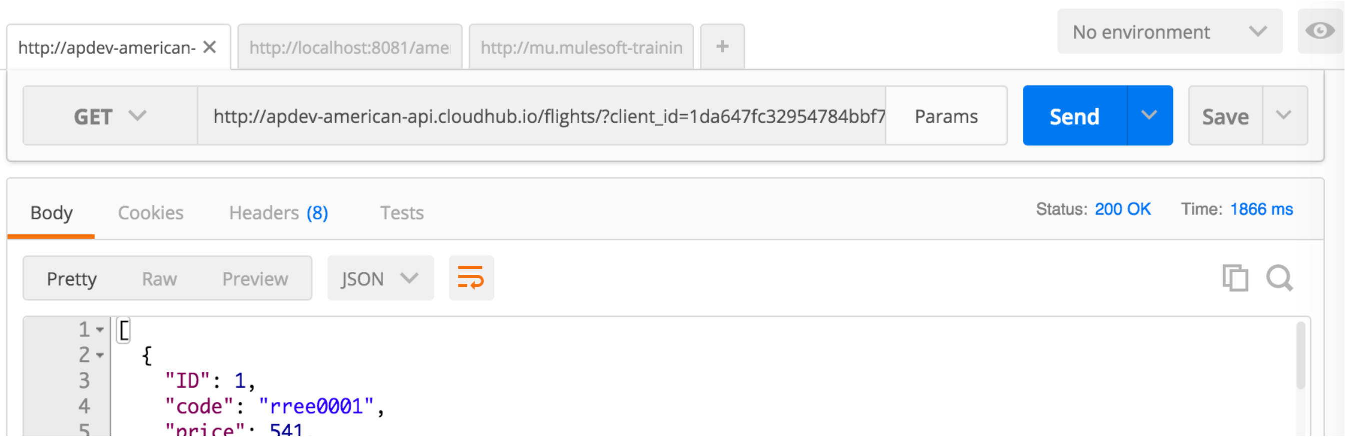
localhost:8081/american?code=CLE

Status: 200 Time: 1354 ms

```
[
  {
    "airline": "American",
    "flightCode": "eefd0123",
    "fromAirportCode": "MUA",
    "toAirportCode": "CLE",
    "departureDate": "2016-01-25T00:00:00",
    "emptySeats": 7,
    "totalSeats": 345,
    "price": 300,
    "planeType": "Boeing 747"
  },
```

Make a request to the web service

1. In Postman, return to the first tab – the one with the request to your American Flights API http://training-american-api-{lastname}.cloudhub.io/flights and that passes a client_id and client_secret.
2. Send the request; you should still see JSON data for the American flights as a response.



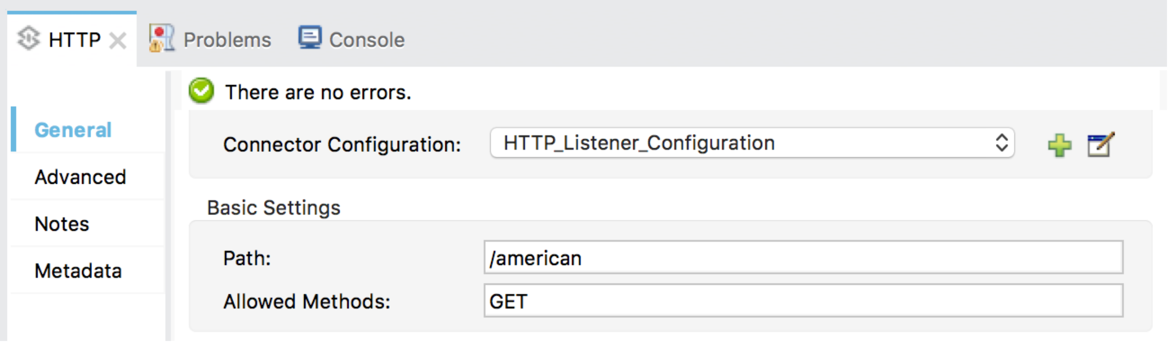
http://apdev-american-api.cloudhub.io/flights/?client_id=1da647fc32954784bbf7

Status: 200 OK Time: 1866 ms

```
[
  {
    "ID": 1,
    "code": "rree0001",
    "price": 541
```

Add a new flow with an HTTP Listener endpoint

3. Return to implementation.xml.
4. Drag out another HTTP connector and drop it in the canvas.
5. Rename the flow to getAmericanFlightsFlow.
6. In the Properties view, set the connector configuration to the existing HTTP_Listener_Configuration.
7. Set the path to /american.
8. Set the allowed methods to GET.



There are no errors.

Connector Configuration: HTTP_Listener_Configuration

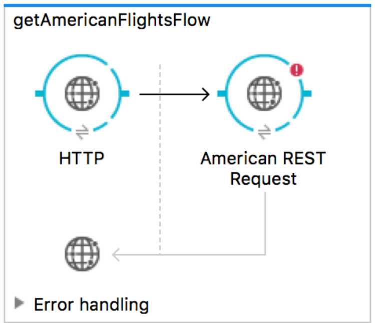
Basic Settings

Path: /american

Allowed Methods: GET

Add an HTTP Request endpoint for a web service with a RAML definition

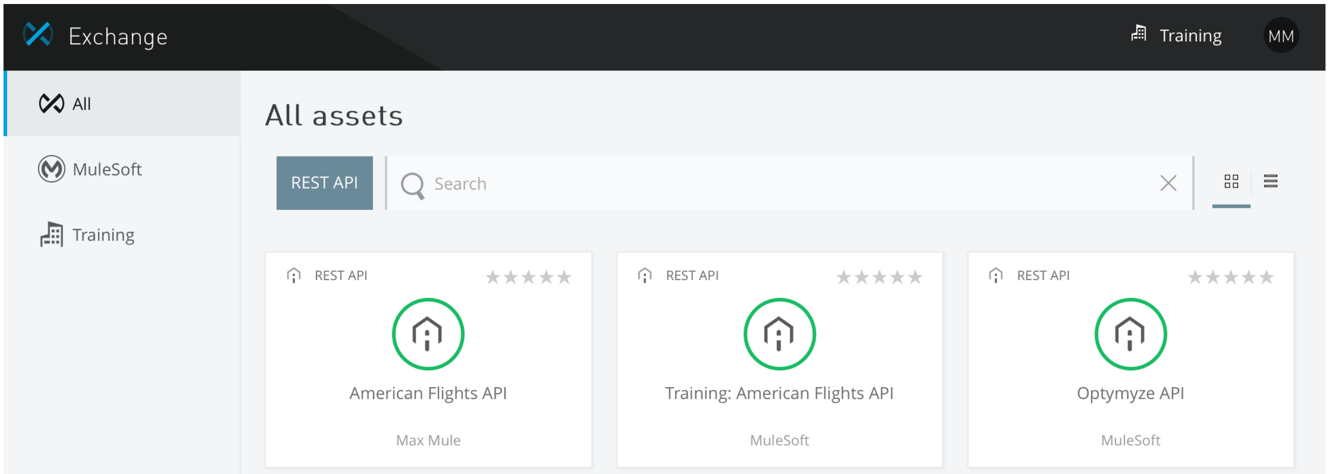
9. Drag out another HTTP connector and drop it in the process section of the new flow.
10. Change the endpoint display name to American REST Request.



Configure the HTTP Request connector

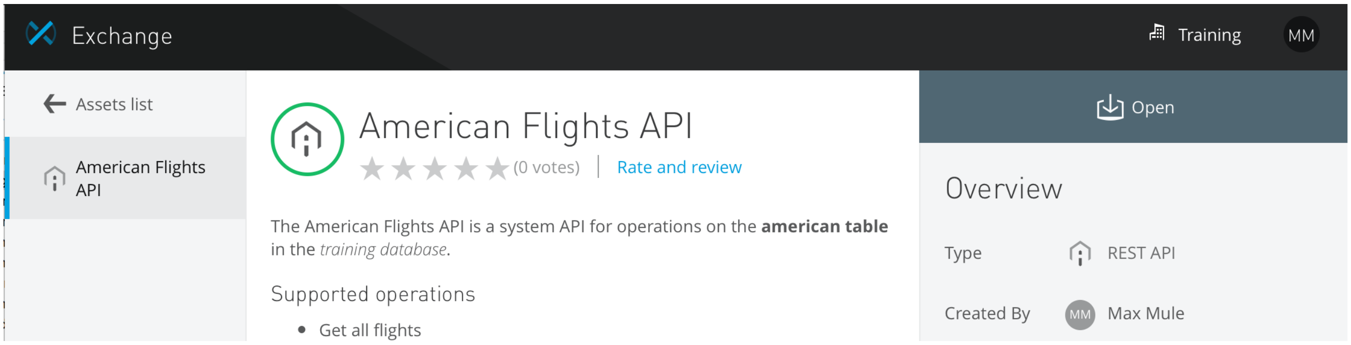
11. Return to global.xml.
12. In the Global Elements view, click Create.
13. In the Choose Global Type dialog box, select Connector Configuration > HTTP Request Configuration and click OK.

14. In the Global Element Properties dialog box, change the name to American_HTTP_Request_Configuration.
15. Click the Search in Exchange link next to REST API Location.
16. In the Exchange window that opens, locate your American Flights RAML and click it.



Note: If you do not have a functional API implementation for the American Flights API, you can select the Training: American Flights API instead.

17. Click the Open button; the Exchange window should close.

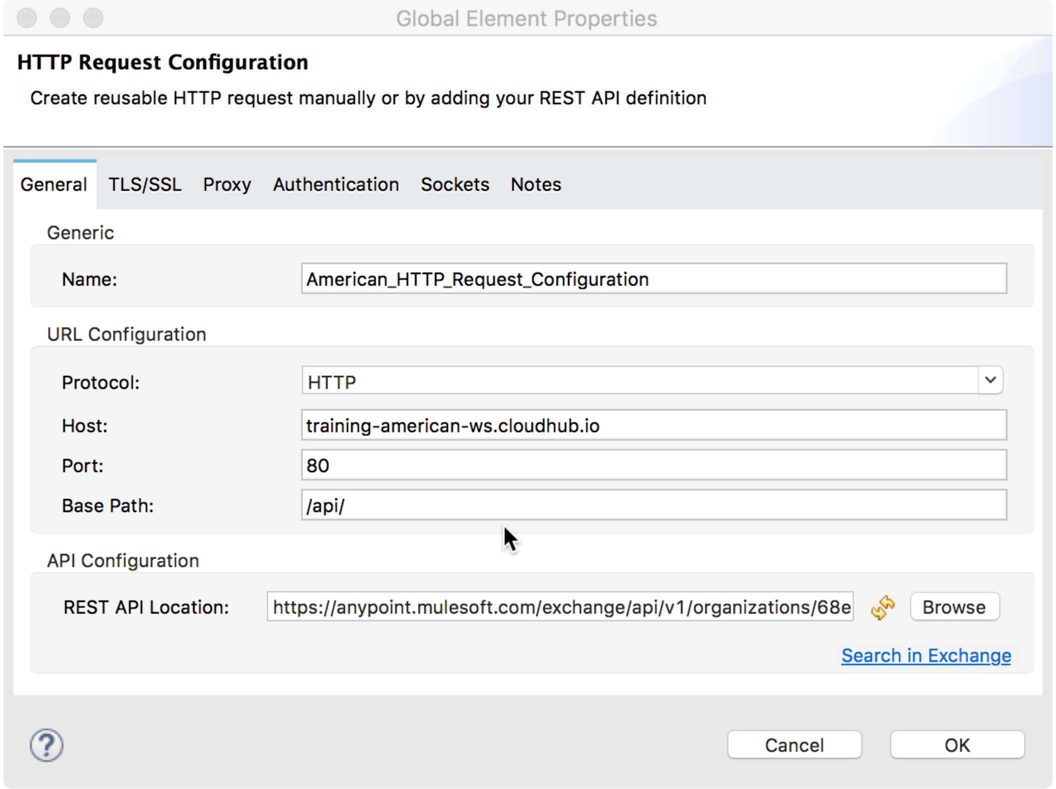


The American Flights API is a system API for operations on the **american table** in the *training database*.

Supported operations

- Get all flights

18. Back in the Global Element Properties dialog box, wait for the RAML to be parsed and the host, port, and base path fields to be populated and then click OK.



HTTP Request Configuration

Create reusable HTTP request manually or by adding your REST API definition

Generic

Name: American_HTTP_Request_Configuration

URL Configuration

Protocol: HTTP

Host: training-american-ws.cloudhub.io

Port: 80

Base Path: /api/

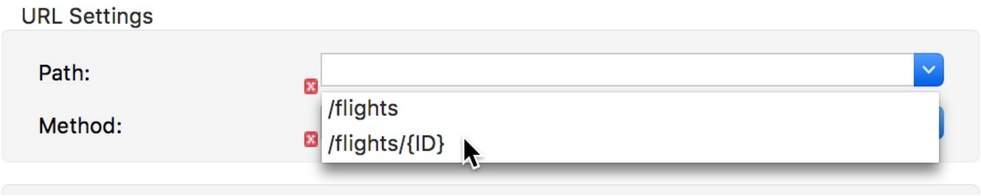
API Configuration

REST API Location: https://anypoint.mulesoft.com/exchange/api/v1/organizations/68e

19. Click OK.

Configure the HTTP Request endpoint

20. Return to implementation.xml.
21. In the American REST Request properties view, set the connector configuration to the existing American_HTTP_Request_Configuration.
22. Click the expand button for the path field; you should see all the available resources for the RESTful web service defined by the RAML file listed – in this case, there are two.
23. Select /flights/{ID}.

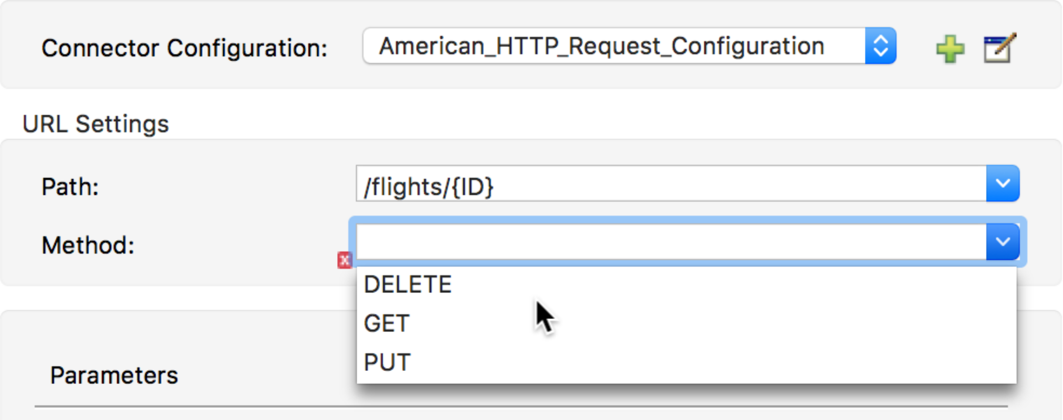


URL Settings

Path:

Method:

24. Look at the method drop-down menu; you should see DELETE, GET, and PUT.



Connector Configuration: American_HTTP_Request_Configuration

URL Settings

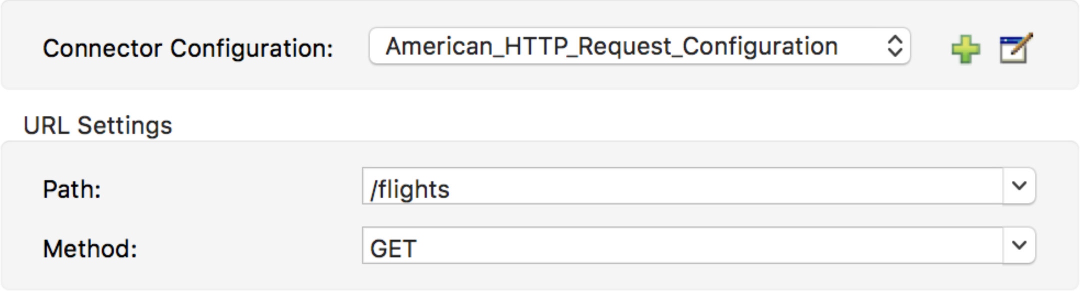
Path: /flights/{ID}

Method:

25. Change the path to /flights.

26. Look at the method drop-down menu; you should see GET and POST.

27. Set the method to GET.



Connector Configuration: American_HTTP_Request_Configuration

URL Settings

Path: /flights

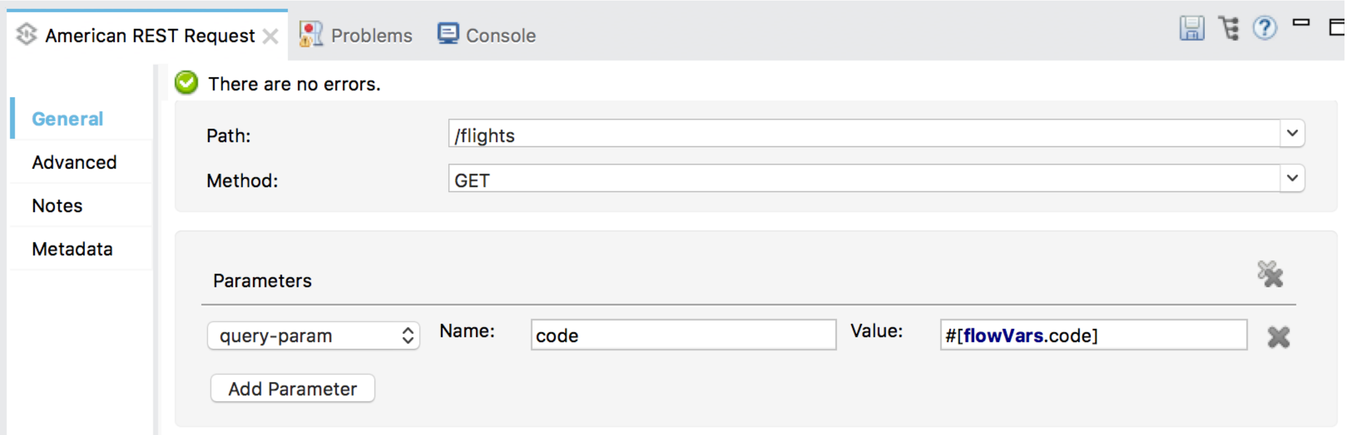
Method: GET

28. Scroll down and check to see if a query parameter called code has been added.

29. If the destination parameter was not automatically created, create it.

30. Set the value to a flow variable called code; you will add this to the flow next.

```
#[flowVars.code]
```



There are no errors.

Path: /flights

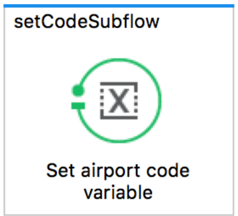
Method: GET

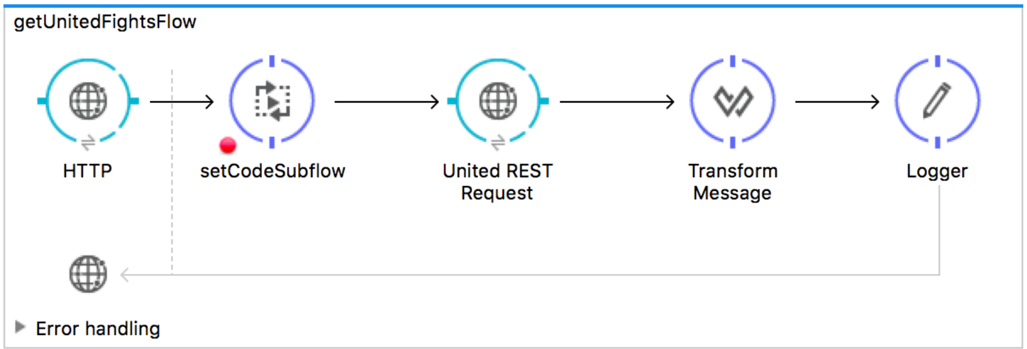
query-param Name: code Value: #[flowVars.code]

Extract the Set airport code variable processor into a subflow and use it

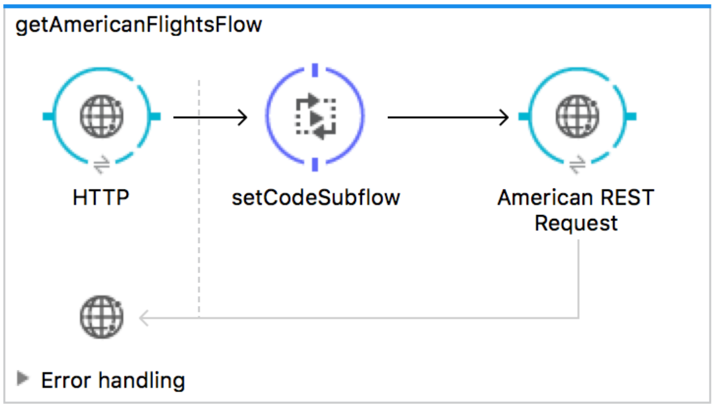
31. Right-click the Set airport code variable processor in getUnitedFlightsFlow and select Extract to > Sub Flow.
32. In the Extract Flow dialog box, set the flow name to setCodeSubflow and click OK.
33. Double-click the new Flow Reference in getUnitedFlightsFlow; the display name should update.
34. Locate the new subflow.

    Note: If you do not see the subflow in the canvas, switch to the Configuration XML view and then back to the Message Flow view.



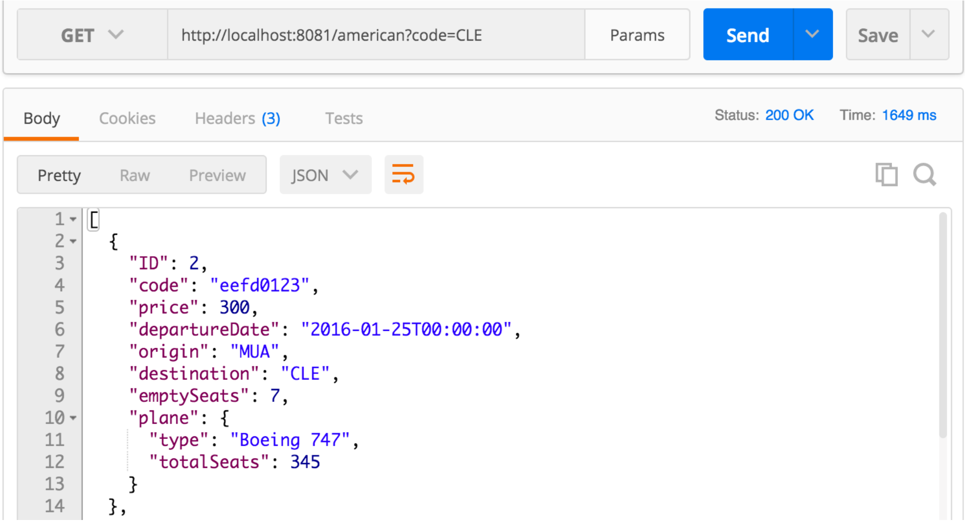


35. Drag a Flow Reference component from the Mule Palette before the American REST Request in getAmericanFlightsFlow.
36. In the Flow Reference properties view, set the flow name to setCodeSubflow.



Test the application

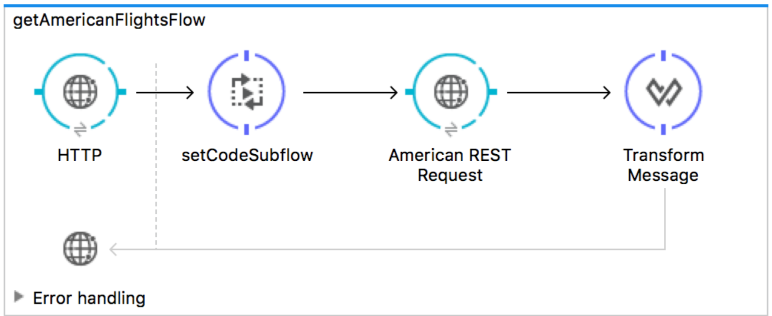
37. Save all the files to redeploy the application.
38. In Postman, return to the tab with the local requests.
39. Make a request to http://localhost:8081/american; you should get the American flights to SFO.
40. Add a query parameter called code with a value of CLE.
41. Send the request; you should get just the flights to CLE.



```
[
  {
    "ID": 2,
    "code": "eefd0123",
    "price": 300,
    "departureDate": "2016-01-25T00:00:00",
    "origin": "MUA",
    "destination": "CLE",
    "emptySeats": 7,
    "plane": {
      "type": "Boeing 747",
      "totalSeats": 345
    }
  },
```

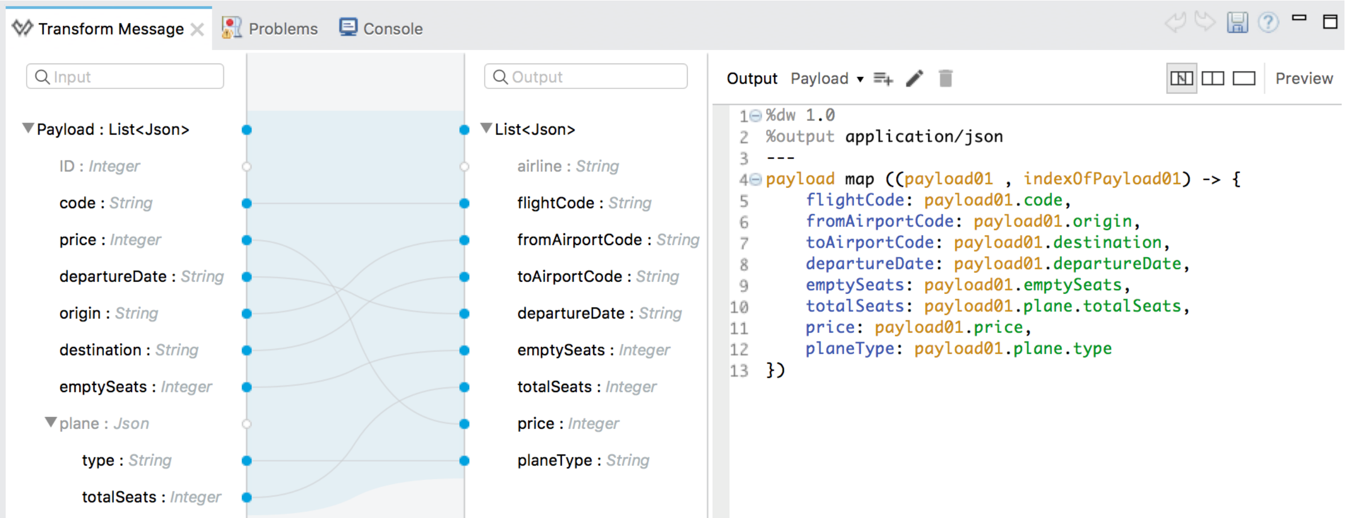
Transform the data

42. Return to getAmericanFlightsFlow.
43. Add a Transform Message component after the American REST Request endpoint.



44. Double-click the Transform Message component.
45. Look at the input section; you should see metadata already defined.
46. In the output section of the Transform Message properties view, click the Define metadata link.
47. In the Define metadata type dialog box, select flights_json.
48. Click Select; you should now see output metadata in the output section of the Transform Message properties view.

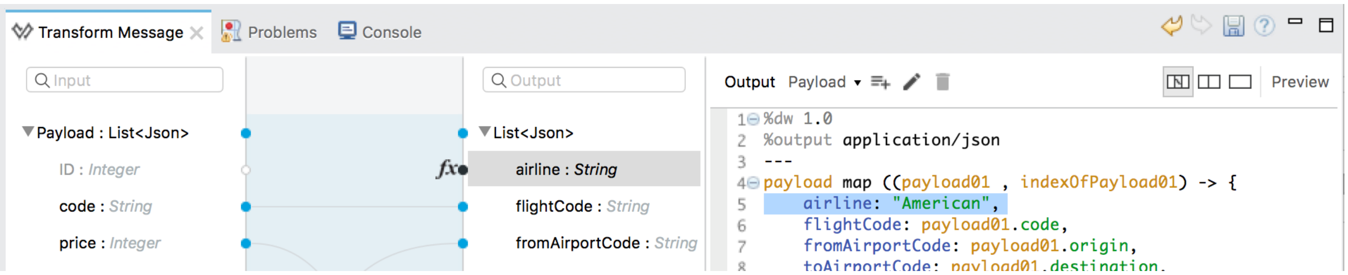
49. Map fields (except ID and airline) by dragging them from the input section and dropping them on the corresponding field in the output section.



```
%dw 1.0
%output application/json
---
payload map ((payload01 , indexOfPayload01) -> {
    flightCode: payload01.code,
    fromAirportCode: payload01.origin,
    toAirportCode: payload01.destination,
    departureDate: payload01.departureDate,
    emptySeats: payload01.emptySeats,
    totalSeats: payload01.plane.totalSeats,
    price: payload01.price,
    planeType: payload01.plane.type
})
```

50. Double-click the airline field in the output section.

51. In the generated DataWeave expression, change the airline value from null to "American".

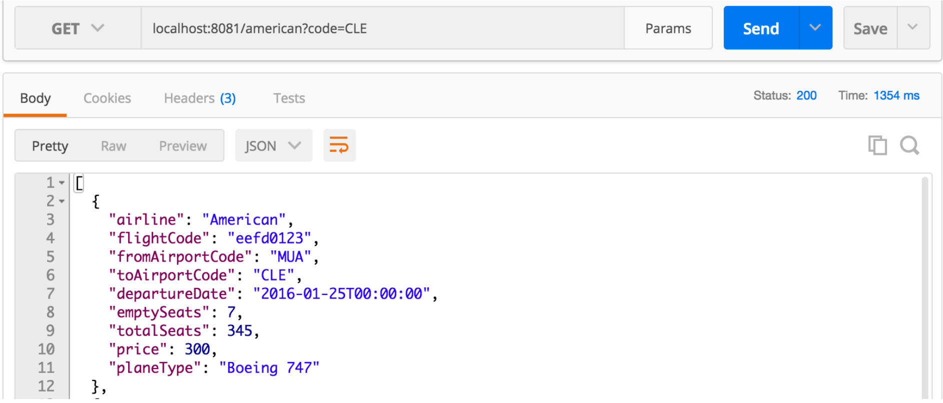


```
%dw 1.0
%output application/json
---
payload map ((payload01 , indexOfPayload01) -> {
    airline: "American",
    flightCode: payload01.code,
    fromAirportCode: payload01.origin,
    toAirportCode: payload01.destination,
```

Test the application

52. Save to redeploy the project.

53. In Postman, make a request to http://localhost:8081/american; you should see all the flight data as JSON again but now with a different structure.



localhost:8081/american?code=CLE

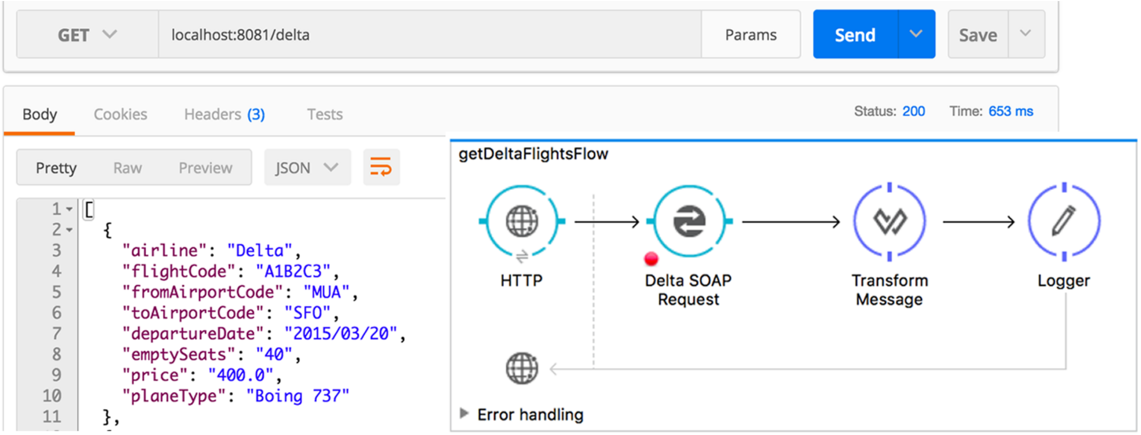
Status: 200 Time: 1354 ms

```
[
  {
    "airline": "American",
    "flightCode": "eefd0123",
    "fromAirportCode": "MUA",
    "toAirportCode": "CLE",
    "departureDate": "2016-01-25T00:00:00",
    "emptySeats": 7,
    "totalSeats": 345,
    "price": 300,
    "planeType": "Boeing 747"
  },
```

Walkthrough 8-4: Consume a SOAP web service

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

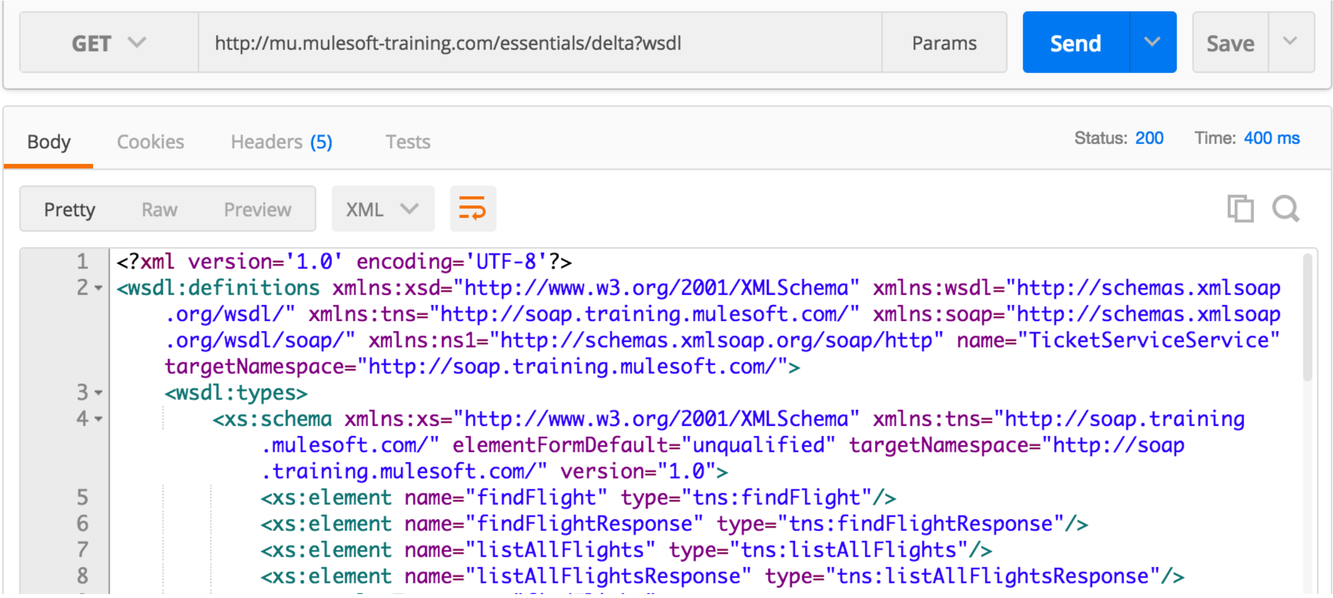
- Create a new flow to call a SOAP web service.
- Use a Web Service Consumer endpoint to consume a SOAP web service for Delta flight data.
- Use DataWeave to transform the XML response into JSON specified by the MUA Flights API.



```
[
  {
    "airline": "Delta",
    "flightCode": "A1B2C3",
    "fromAirportCode": "MUA",
    "toAirportCode": "SFO",
    "departureDate": "2015/03/20",
    "emptySeats": "40",
    "price": "400.0",
    "planeType": "Boing 737"
  },
```

Browse the WSDL

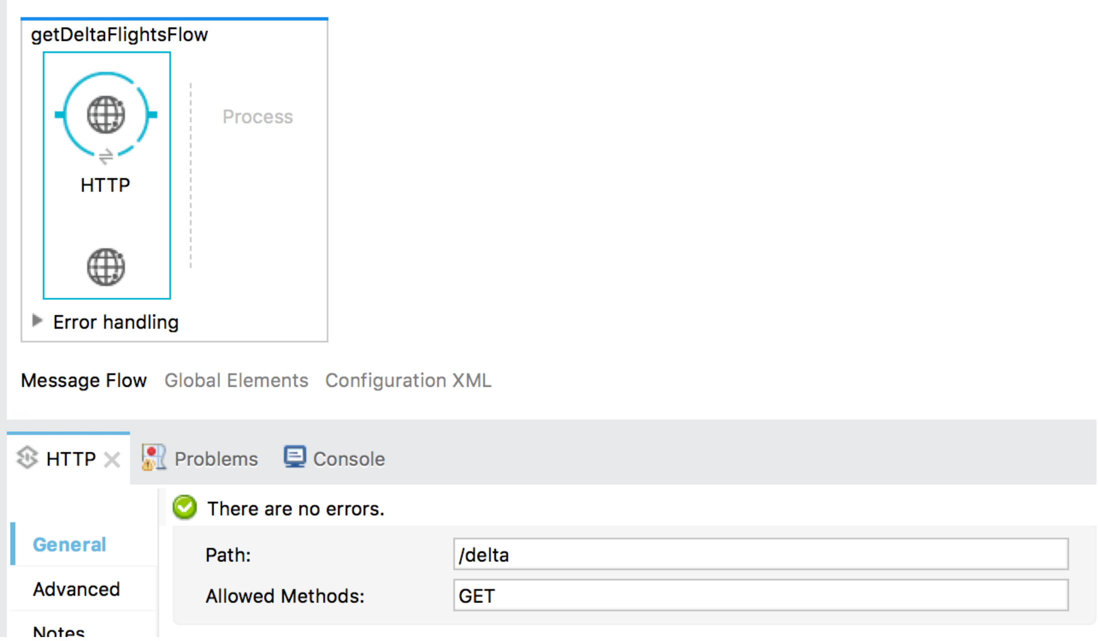
1. Return to the course snippets.txt file and copy the WSDL URL for the Delta SOAP web service.
2. In Postman, return to the third tab, the one with the mulesoft-training request.
3. Paste the URL and send the request; you should see the web service WSDL returned.
4. Browse the WSDL; you should find references to operations listAllFlights and findFlight.



```
<?xml version='1.0' encoding='UTF-8'?>
<wsdl:definitions xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:wsdl="http://schemas.xmlsoap
    .org/wsdl/" xmlns:tns="http://soap.training.mulesoft.com/" xmlns:soap="http://schemas.xmlsoap
    .org/wsdl/soap/" xmlns:ns1="http://schemas.xmlsoap.org/soap/http" name="TicketServiceService"
    targetNamespace="http://soap.training.mulesoft.com/">
    <wsdl:types>
        <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:tns="http://soap.training
            .mulesoft.com/" elementFormDefault="unqualified" targetNamespace="http://soap
            .training.mulesoft.com/" version="1.0">
            <xs:element name="findFlight" type="tns:findFlight"/>
            <xs:element name="findFlightResponse" type="tns:findFlightResponse"/>
            <xs:element name="listAllFlights" type="tns:listAllFlights"/>
            <xs:element name="listAllFlightsResponse" type="tns:listAllFlightsResponse"/>
```

Create a new flow with an HTTP Listener connector endpoint

5. Return to Anypoint Studio.
6. Drag out another HTTP connector and drop it in the canvas after the existing flows.
7. Rename the flow to getDeltaFlightsFlow.
8. In the Properties view for the endpoint, set the connector configuration to the existing HTTP_Listener_Configuration.
9. Set the path to /delta.
10. Set the allowed methods to GET.



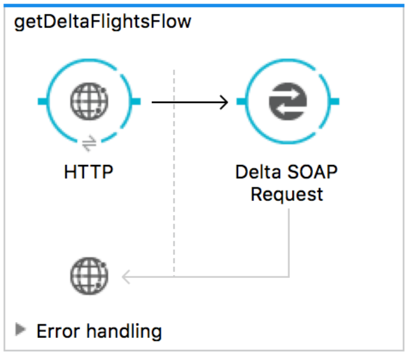
There are no errors.

Path: /delta

Allowed Methods: GET

Add a Web Service Consumer connector endpoint

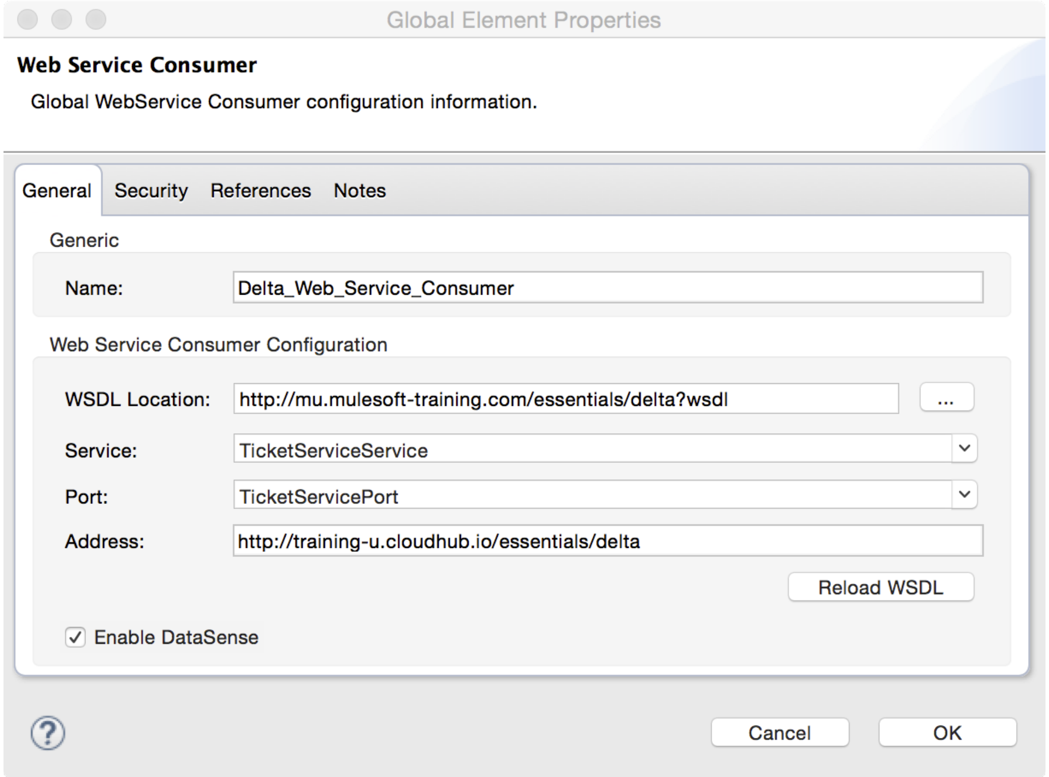
11. Drag out a Web Service Consumer connector and drop it in the process section of the flow.
12. Change its display name to Delta SOAP Request.



Configure the Web Service Consumer connector

13. Return to global.xml.
14. Click Create.
15. In the Choose Global Type dialog box, select Connector Configuration > Web Service Consumer and click OK.
16. In the Global Element Properties dialog box, change the name to Delta_Web_Service_Consumer.
17. Set the WSDL location to the value you copied from the course snippets.txt file.
18. Wait for the service, port, and address fields to populate.

Note: If the fields do not populate, click the Reload WSDL button. If the fields still do not auto-populate, select TicketServiceService from the service drop-down menu and select TicketServicePort from the port drop-down menu.



Web Service Consumer

Global WebService Consumer configuration information.

Generic

Name: Delta_Web_Service_Consumer

Web Service Consumer Configuration

WSDL Location: http://mu.mulesoft-training.com/essentials/delta?wsdl

Service: TicketServiceService

Port: TicketServicePort

Address: http://training-u.cloudhub.io/essentials/delta

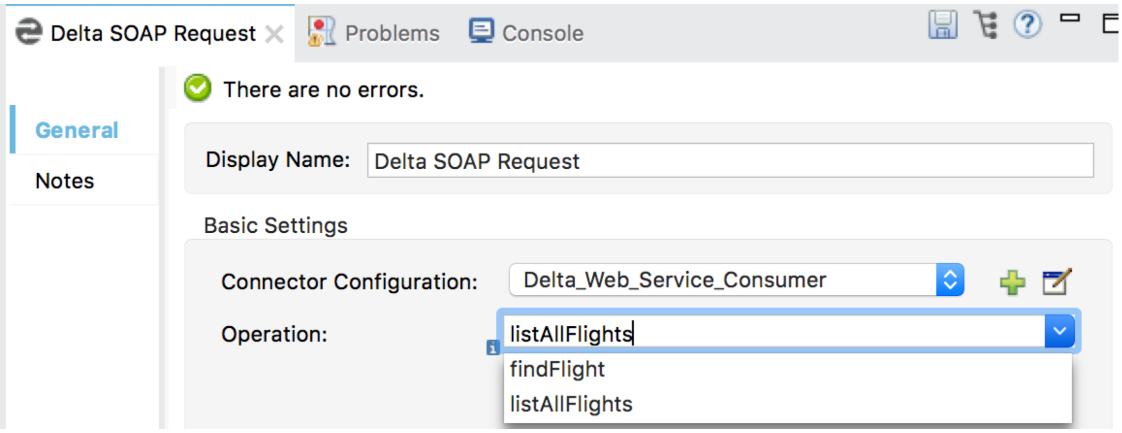
☑ Enable DataSense

19. Click OK.

Configure the Web Service Consumer endpoint

20. Return to implementation.xml.

21. In the Delta SOAP Request properties view, set the connector configuration to the existing Delta_Web_Service_Consumer.
22. Click the operation drop-down menu button; you should see all of the web service operations listed.



There are no errors.

Display Name: Delta SOAP Request

Basic Settings

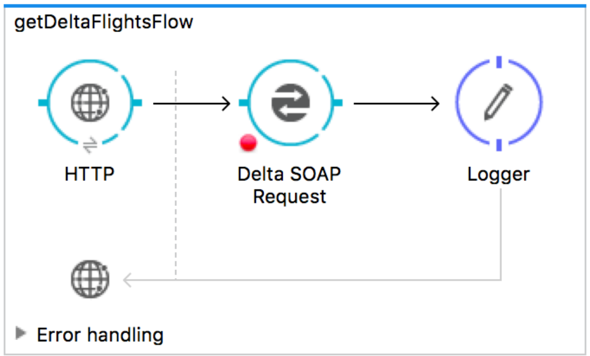
Connector Configuration: Delta_Web_Service_Consumer

Operation: listAllFlights

23. Select the listAllFlights operation.

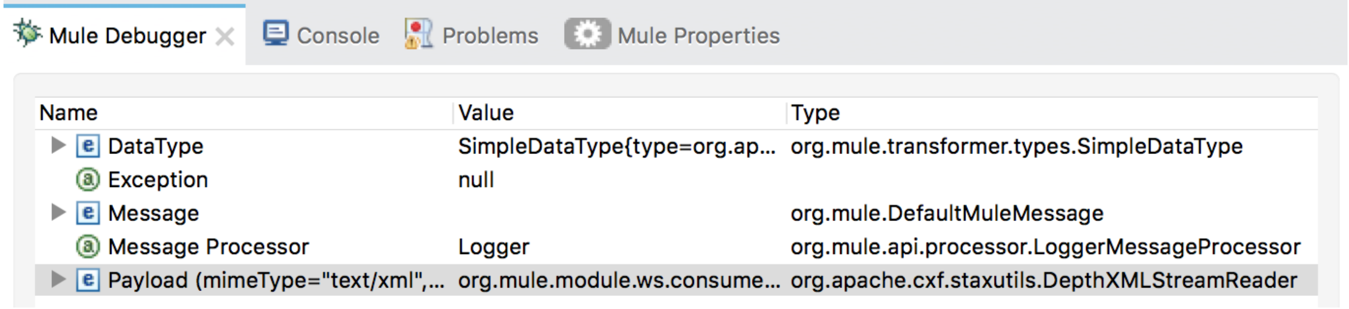
Test the application

24. Add a breakpoint to the Delta SOAP Request endpoint.
25. Add a Logger to the end of the flow.



26. Debug the project.
27. In Postman, return to the middle tab – the one with the localhost requests.
28. Make a request to http://localhost:8081/delta.

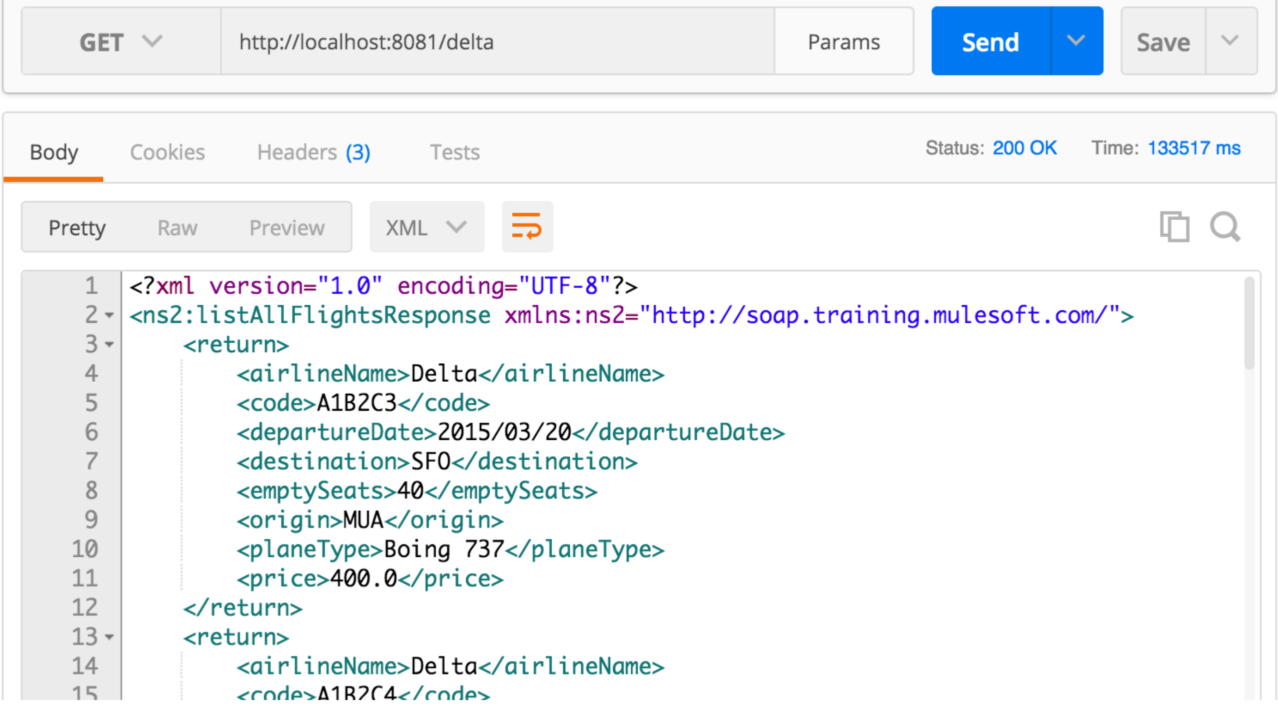
29. In the Mule Debugger, step to the Logger and look at the payload; it should be of type DepthXMLStreamReader.



Name	Value	Type
▶ e DataType	SimpleDataType{type=org.ap...	org.mule.transformer.types.SimpleDataType
a Exception	null	
▶ e Message		org.mule.DefaultMuleMessage
a Message Processor	Logger	org.mule.api.processor.LoggerMessageProcessor
▶ e Payload (mimeType="text/xml",...	org.mule.module.ws.consume...	org.apache.cxf.staxutils.DepthXMLStreamReader

30. Step through the application.

31. Return to Postman; you should see the XML flight data returned.

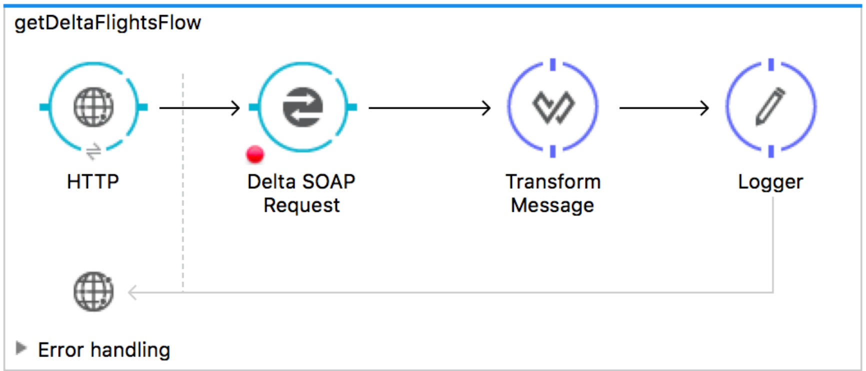


```
<?xml version="1.0" encoding="UTF-8"?>
<ns2:listAllFlightsResponse xmlns:ns2="http://soap.training.mulesoft.com/">
    <return>
        <airlineName>Delta</airlineName>
        <code>A1B2C3</code>
        <departureDate>2015/03/20</departureDate>
        <destination>SFO</destination>
        <emptySeats>40</emptySeats>
        <origin>MUA</origin>
        <planeType>Boing 737</planeType>
        <price>400.0</price>
    </return>
    <return>
        <airlineName>Delta</airlineName>
        <code>A1B2C4</code>
```

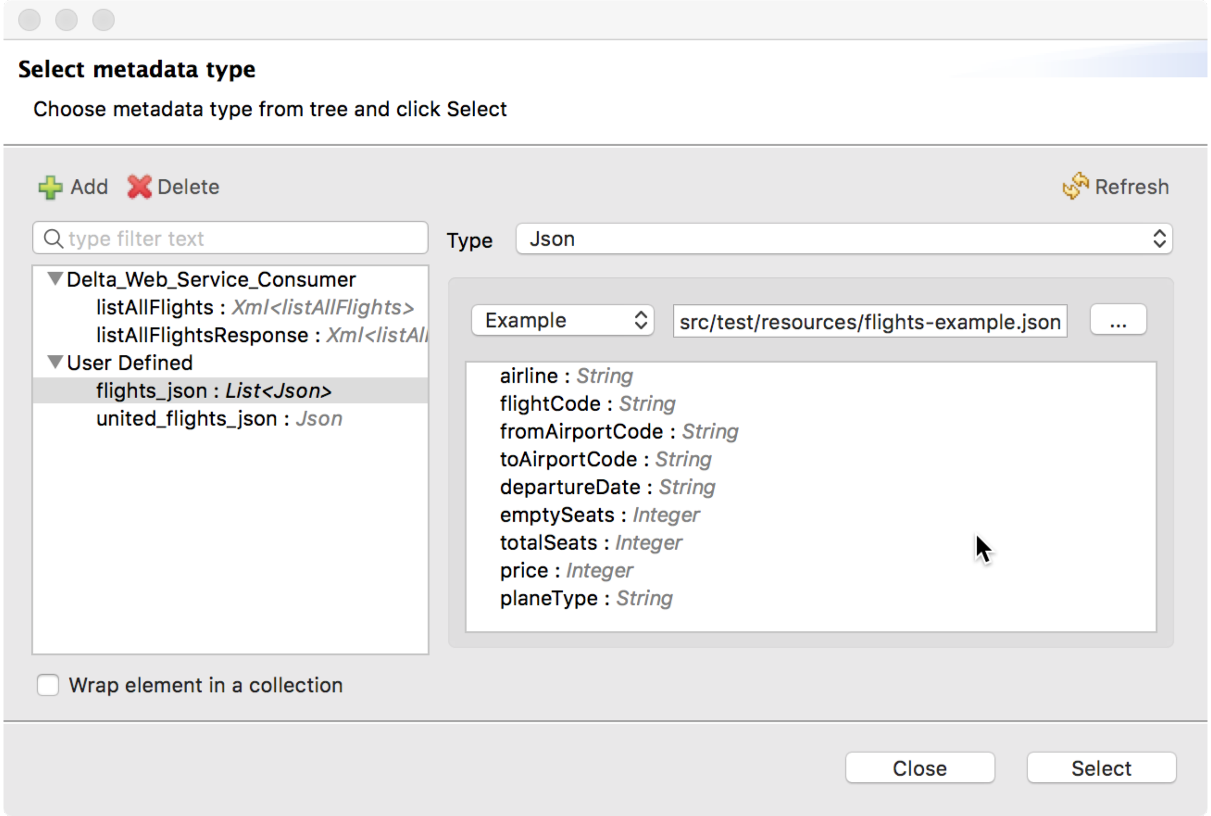
Transform the data

32. Return to Anypoint Studio, stop the project, and switch to the Mule Design perspective.

33. In getDeltaFlightsFlow, add a Transform Message component after the Delta SOAP Request endpoint.



34. Look at the input section of the Transform Message properties view; you should see metadata already defined.
35. In the output section of the Transform Message properties view, click the Define metadata link.
36. In the Define metadata type dialog box, select User Defined > flights_json.



Select metadata type

Choose metadata type from tree and click Select

Type: Json

- Delta_Web_Service_Consumer
  - listAllFlights : *Xml<listAllFlights>*
  - listAllFlightsResponse : *Xml<listAll*
- User Defined
  - flights_json : *List<Json>*
  - united_flights_json : *Json*

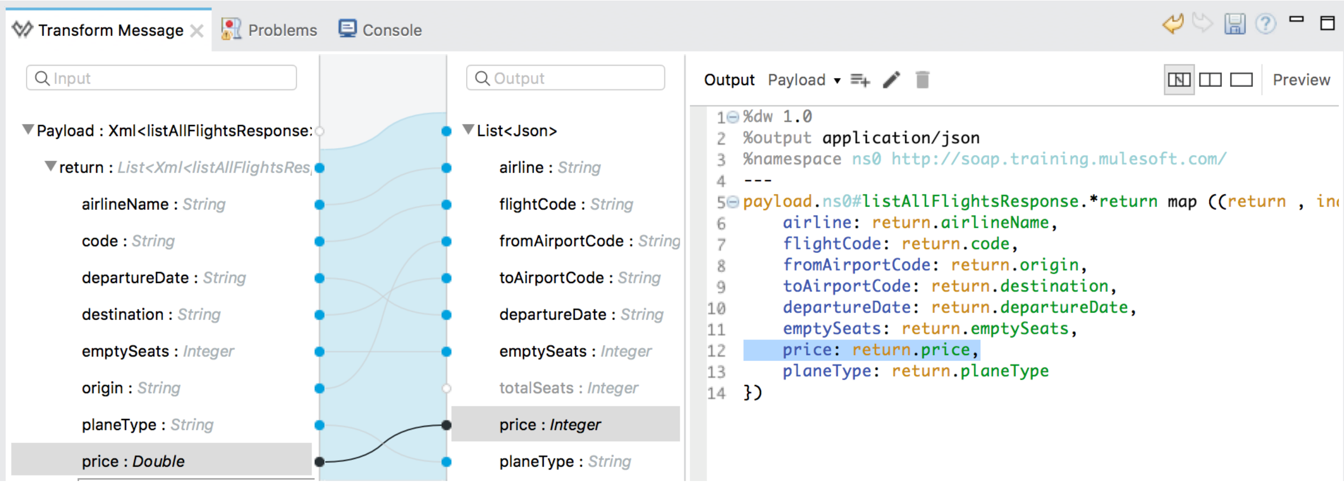
Example: src/test/resources/flights-example.json

airline : *String*
flightCode : *String*
fromAirportCode : *String*
toAirportCode : *String*
departureDate : *String*
emptySeats : *Integer*
totalSeats : *Integer*
price : *Integer*
planeType : *String*

□ Wrap element in a collection

37. Click Select; you should now see output metadata in the output section of the Transform Message properties view.

38. Map fields by dragging them from the input section and dropping them on the corresponding field in the output section.

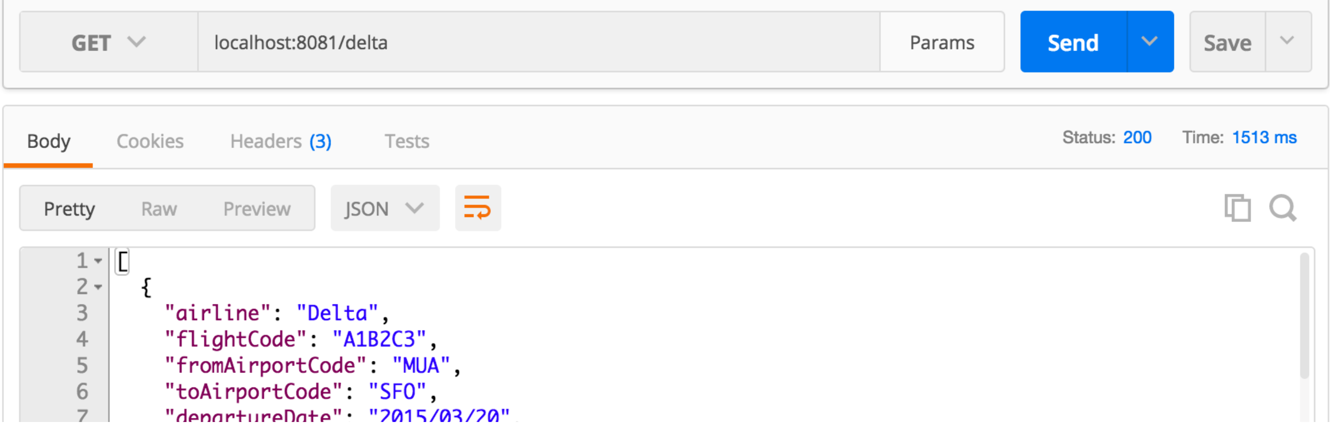


```
%dw 1.0
%output application/json
%namespace ns0 http://soap.training.mulesoft.com/
---
payload.ns0#listAllFlightsResponse.*return map ((return , in
    airline: return.airlineName,
    flightCode: return.code,
    fromAirportCode: return.origin,
    toAirportCode: return.destination,
    departureDate: return.departureDate,
    emptySeats: return.emptySeats,
    price: return.price,
    planeType: return.planeType
})
```

Test the application

39. Run the project.

40. In Postman, make another request to http://localhost:8081/delta; you should see all the flight data but now as JSON.



```
[
  {
    "airline": "Delta",
    "flightCode": "A1B2C3",
    "fromAirportCode": "MUA",
    "toAirportCode": "SFO",
    "departureDate": "2015/03/20",
```

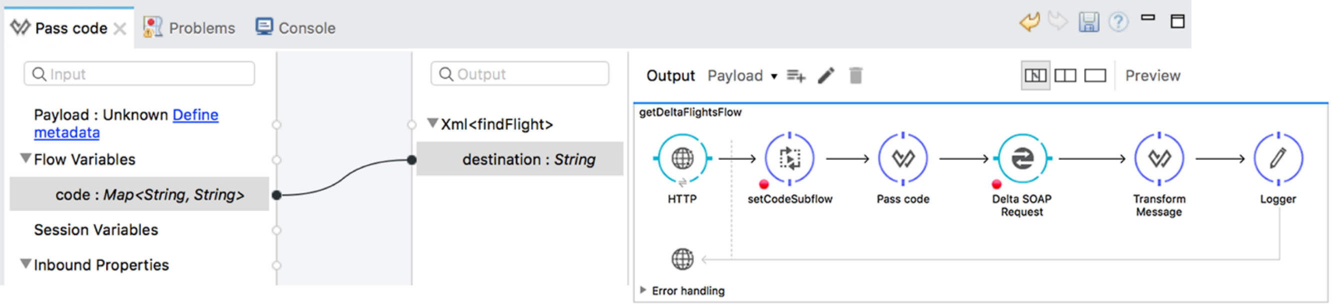
41. Add a query parameter called code and set it equal to LAX.

42. Send the request; you should still get all flights.

Walkthrough 8-5: Pass arguments to a SOAP web service using DataWeave

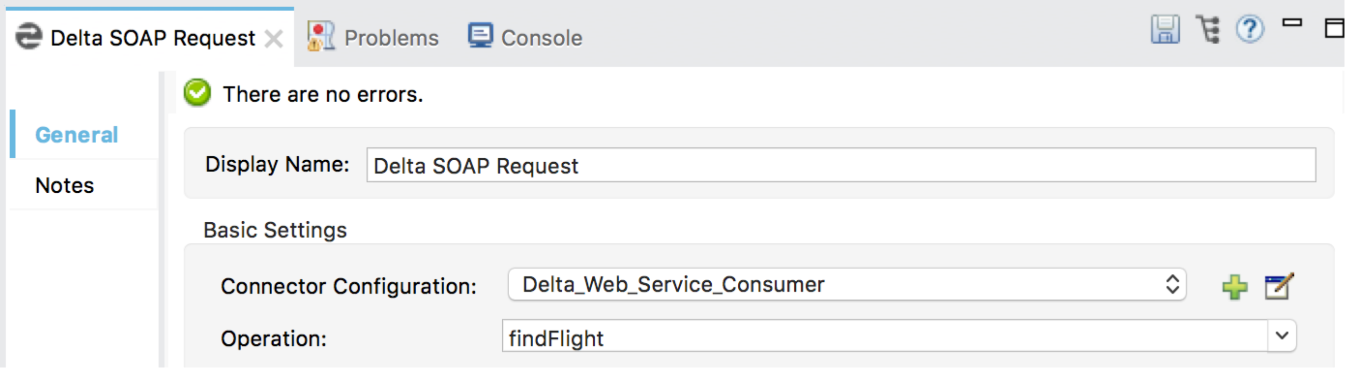
In this walkthrough, you modify the Delta flow to return the flights for a specific destination instead of all the flights. You will:

- Change the web service operation invoked to one that requires a destination as an input argument.
- Set a flow variable to the desired destination.
- Use DataWeave to pass the flow variable to the web service operation.



Call a different web service operation

1. Return to getDeltaFlightsFlow.
2. In the Properties view for the Delta SOAP Request endpoint, change the operation to findFlight.



There are no errors.

Display Name: Delta SOAP Request

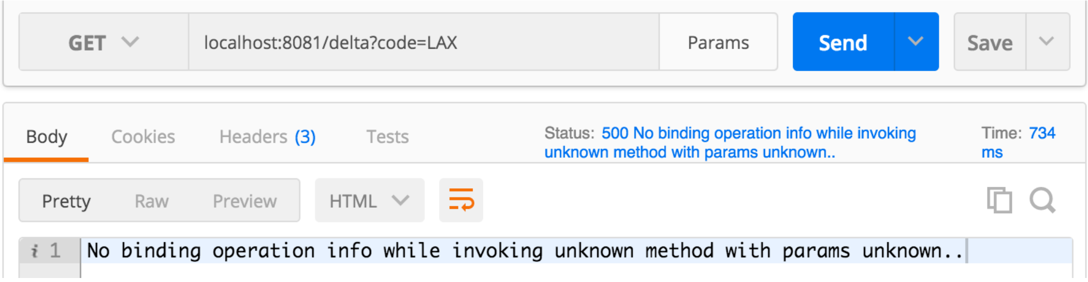
Connector Configuration: Delta_Web_Service_Consumer

Operation: findFlight

Test the application

3. Apply the changes to redeploy the application.

4. In Postman, send the same request with the query parameter; you should get a 500 response with a message about unknown parameters.



localhost:8081/delta?code=LAX

Status: 500 No binding operation info while invoking unknown method with params unknown..

Time: 734 ms

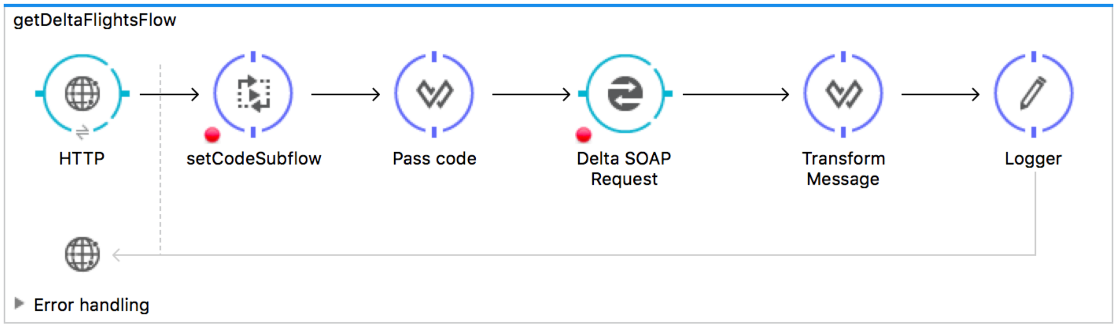
```
No binding operation info while invoking unknown method with params unknown..
```

Use the set airport code subflow

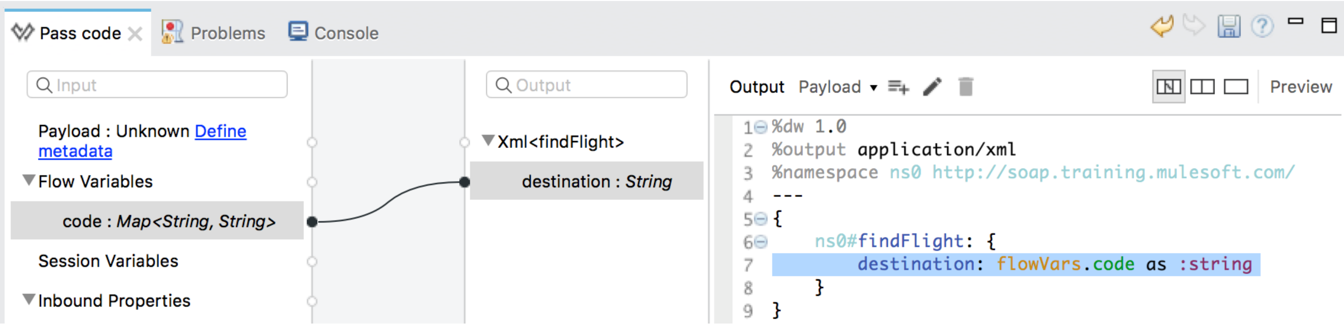
5. Return to getDeltaFlightsFlow.
6. Add a Flow Reference component before the Delta REST Request endpoint.
7. In the Flow Reference properties view, set the flow name to setCodeSubflow.

Use DataWeave to pass parameters to the web service

8. Add a Transform Message component to the left of the Delta SOAP Request endpoint.
9. Change its display name to Pass code.



10. In the Pass code properties view, look at the input and output sections.
11. Drag the code flow variable in the input section to the destination element in the output section.



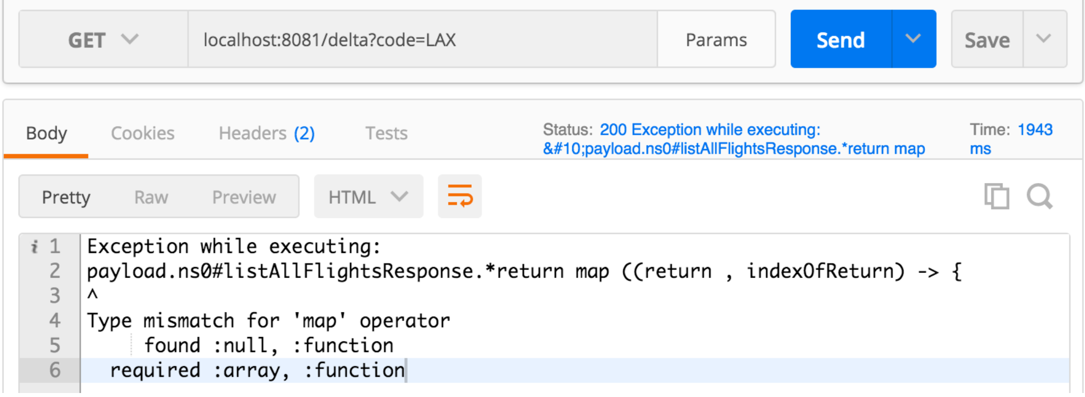
Payload : Unknown Define metadata
Flow Variables
code : *Map<String, String>*
Session Variables
Inbound Properties

Xml<findFlight>
destination : *String*

```
%dw 1.0
%output application/xml
%namespace ns0 http://soap.training.mulesoft.com/
---
{
	ns0#findFlight: {
		destination: flowVars.code as :string
	}
}
```

Test the application

12. Redeploy the application.
13. In Postman, make the same request; you should get a 200 status code with an exception message.



Status: 200 Exception while executing: &#10;payload.ns0#listAllFlightsResponse.*return map

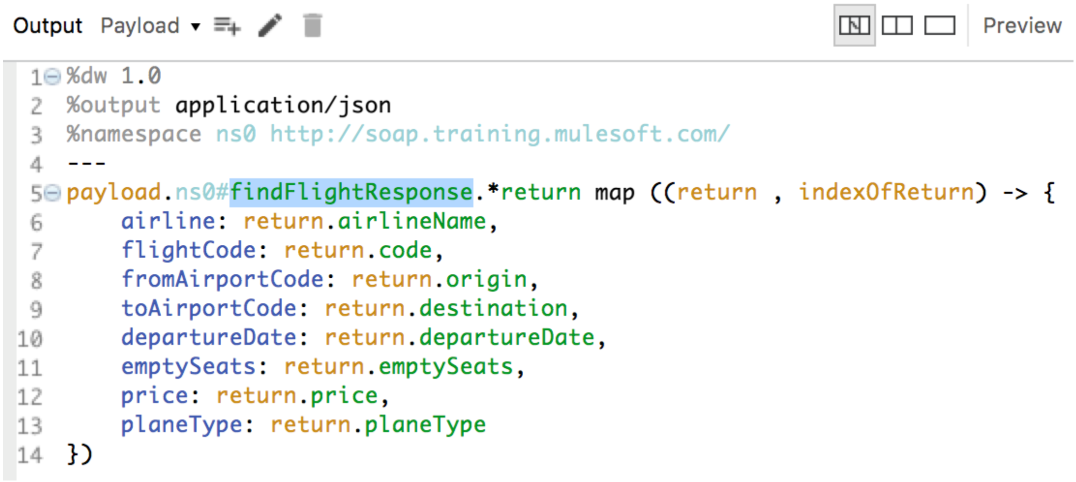
Time: 1943 ms

```
Exception while executing:
payload.ns0#listAllFlightsResponse.*return map ((return , indexOfReturn) -> {
^
Type mismatch for 'map' operator
     found :null, :function
  required :array, :function
```

14. Examine the exception message and figure out what is wrong with the transformation.

Modify the DataWeave expression

15. Return to getDeltaFlightsFlow.
16. Go to the Properties view for the Transform Message component after Delta SOAP Request.
17. Look at the transformation expression.
18. Change ns0#listAllFlightsResponse in the transformation code to ns0#findFlightResponse.

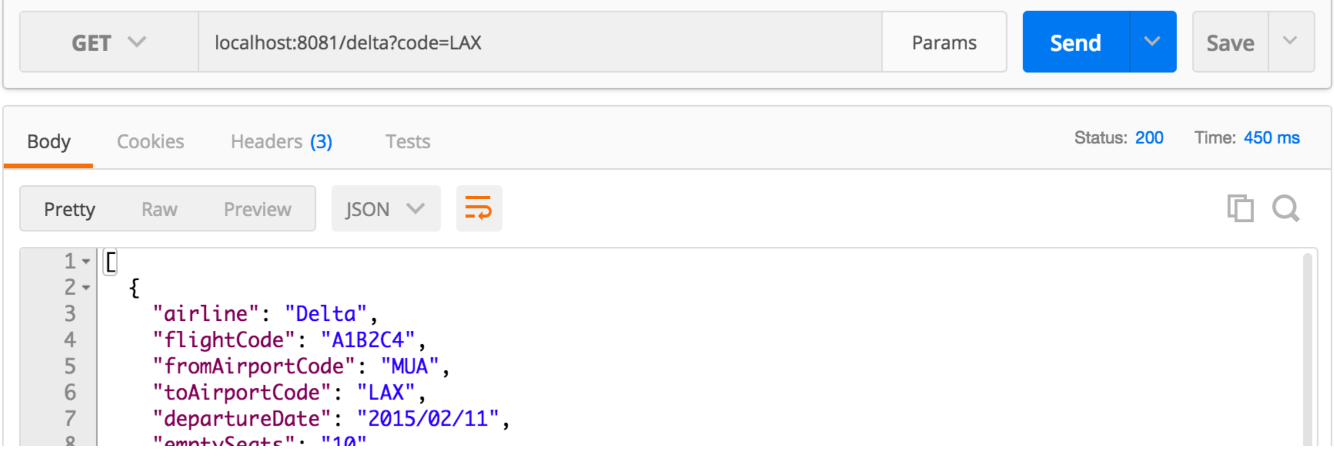


```
%dw 1.0
%output application/json
%namespace ns0 http://soap.training.mulesoft.com/
---
payload.ns0#findFlightResponse.*return map ((return , indexOfReturn) -> {
    airline: return.airlineName,
    flightCode: return.code,
    fromAirportCode: return.origin,
    toAirportCode: return.destination,
    departureDate: return.departureDate,
    emptySeats: return.emptySeats,
    price: return.price,
    planeType: return.planeType
})
```

Test the application

19. Redeploy the application.

20. In Postman, make the same request; you should now get all the flights to LAX.

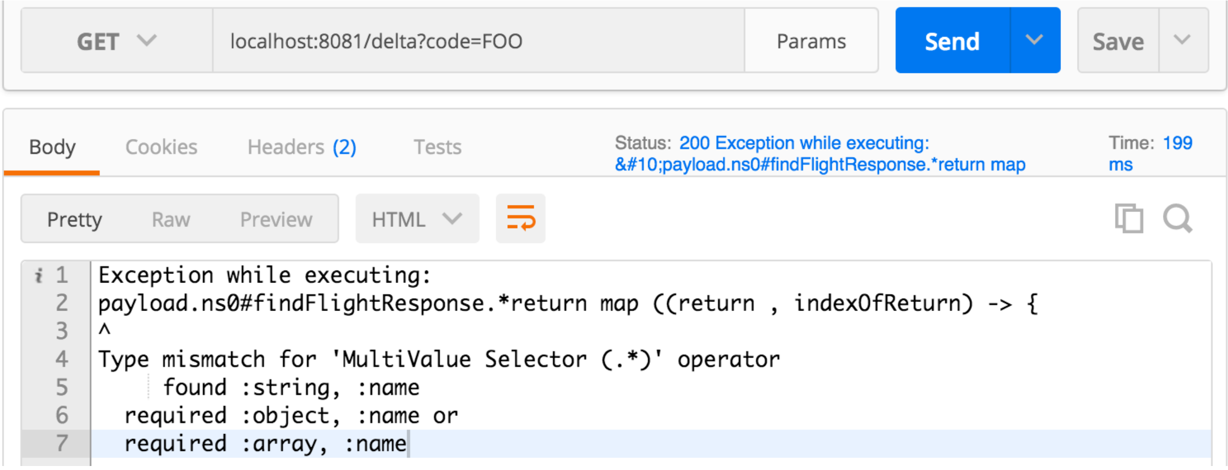


localhost:8081/delta?code=LAX

Status: 200 Time: 450 ms

```
[
  {
    "airline": "Delta",
    "flightCode": "A1B2C4",
    "fromAirportCode": "MUA",
    "toAirportCode": "LAX",
    "departureDate": "2015/02/11",
    "emptySeats": "10"
```

21. Change the code query parameter to have a value of CLE.
22. Send the request; you should now get only flights to CLE.
23. Change the code query parameter to have a value of FOO.
24. Send the request; you should get a 200 status code and a message with an exception.



localhost:8081/delta?code=FOO

Status: 200 Exception while executing: &#10;payload.ns0#findFlightResponse.*return map

Time: 199 ms

```
Exception while executing:
payload.ns0#findFlightResponse.*return map ((return , indexOfReturn) -> {
^
Type mismatch for 'MultiValue Selector (.*)' operator
     found :string, :name
  required :object, :name or
  required :array, :name
```

25. Return to Anypoint Studio and stop the project.