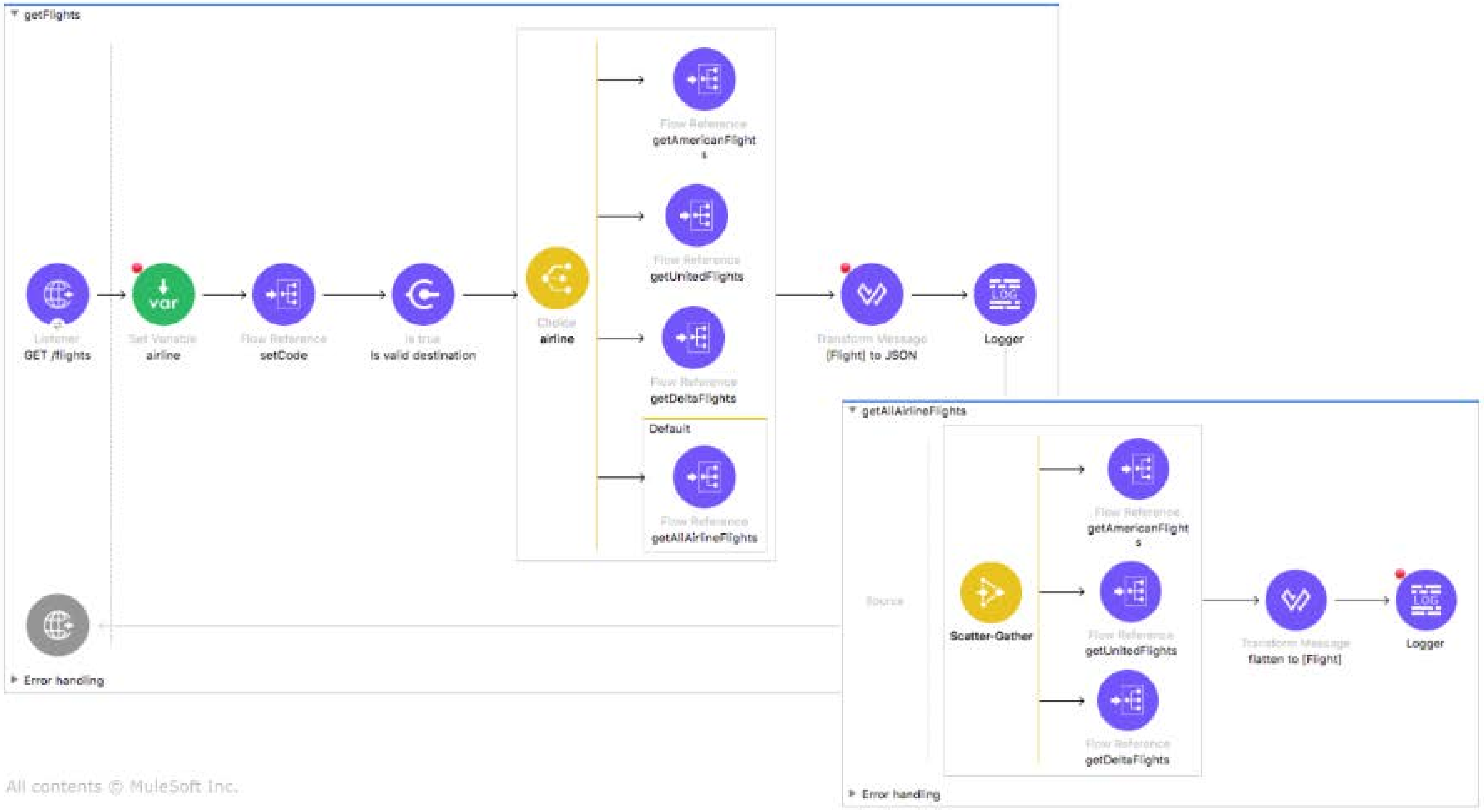
MuleSoft®

Module 9: Controlling Event Flow

Goal



getFlights
Listener
GET /flights
Set Variable
airline
Flow Reference
setCode
Is true
Is valid destination
Choice
airline
Flow Reference
getAmericanFlights
Flow Reference
getUnitedFlights
Flow Reference
getDeltaFlights
Default
Flow Reference
getAllAirlineFlights
Transform Message
[Flight] to JSON
Logger
Error handling
getAllAirlineFlights
Source
Scatter-Gather
Flow Reference
getAmericanFlights
Flow Reference
getUnitedFlights
Flow Reference
getDeltaFlights
Transform Message
flatten to [Flight]
Logger
Error handling

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

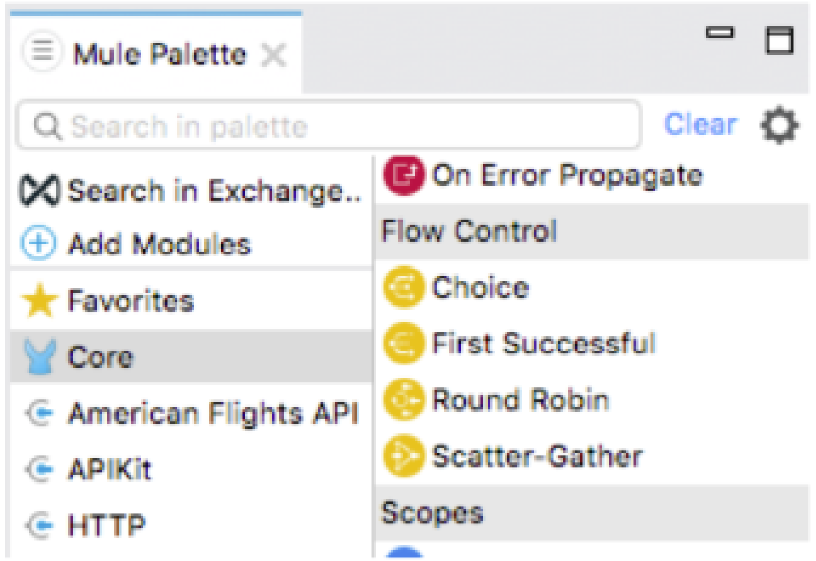
- Multicast events
- Route events based on conditions
- Validate events

Routing events

Routers

- Routers send events to one or more groups of event processors (routes)

- **Choice**
  - One route executed based on conditional logic
- First Successful
  - Routes executed sequentially until one is successfully executed
- Round Robin
  - One route executed, which one is selected by iterating through a list maintained across executions
- **Scatter-Gather**
  - All routes executed concurrently



Multicasting events

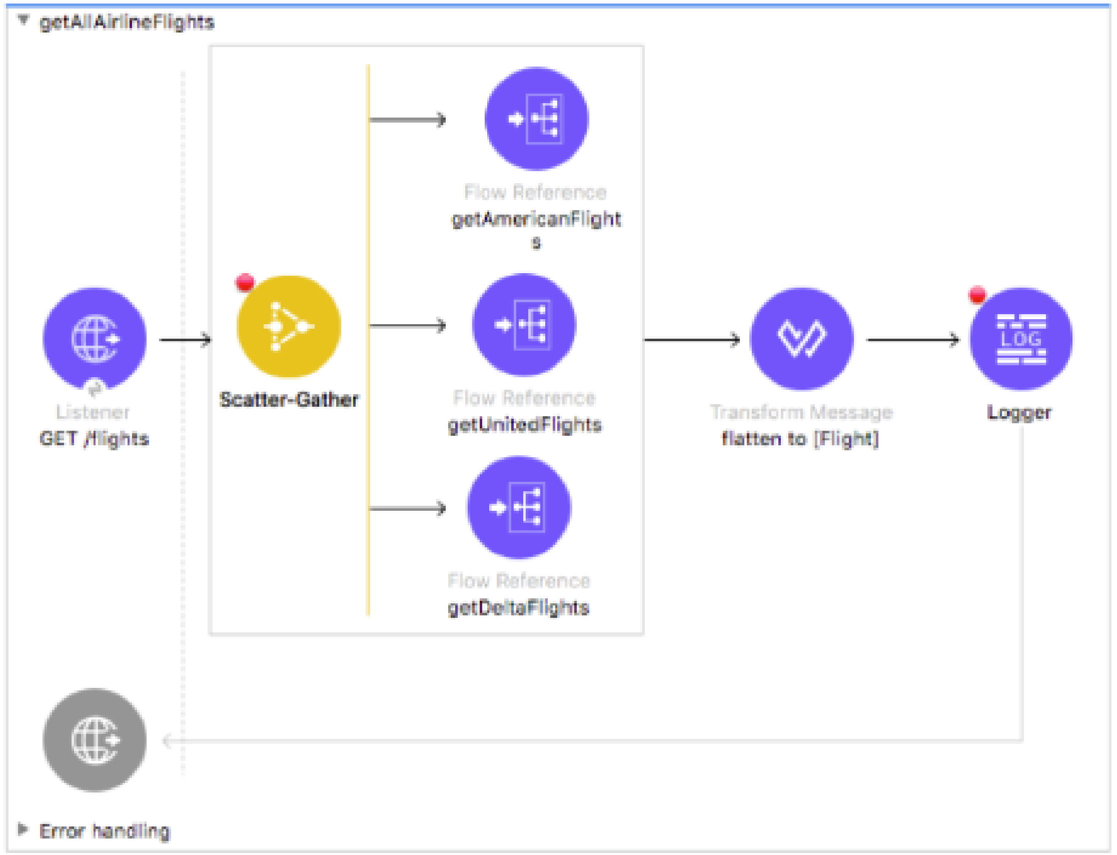
The Scatter-Gather router

- Scatter-Gather sends the event to each route concurrently and returns a collection of all results
- Collection is an object of objects
  - Each object contains attributes and payload from each Mule event returned from a flow

```
{
  "0": {
    "exceptionPayload": null,
    "inboundAttachmentNames":  [ ],
    "outboundPropertyNames":   [ ],
    "inboundPropertyNames":    [ ],
    "attributes": { },
    "outboundAttachmentNames": [ ],
    "payload": [
      {
        "airline": "Delta",
        "flightCode": "A1B2C3",
        "fromAirportCode": "MUA",
        "toAirportCode": "SFO",
        "departureDate": "2015/03/20",
        "emptySeats": "40",
        "price": "400.0",
        "planeType": "Boing 737"
      }
    ],
  "1": {
    "exceptionPayload": null,
    "inboundAttachmentNames":  [ ],
    "outboundPropertyNames":   [ ],
    "inboundPropertyNames":    [ ],
    "attributes": { },
    "outboundAttachmentNames": [ ],
    "payload": "A Payload"
}
```

Walkthrough 9-1: Multicast an event

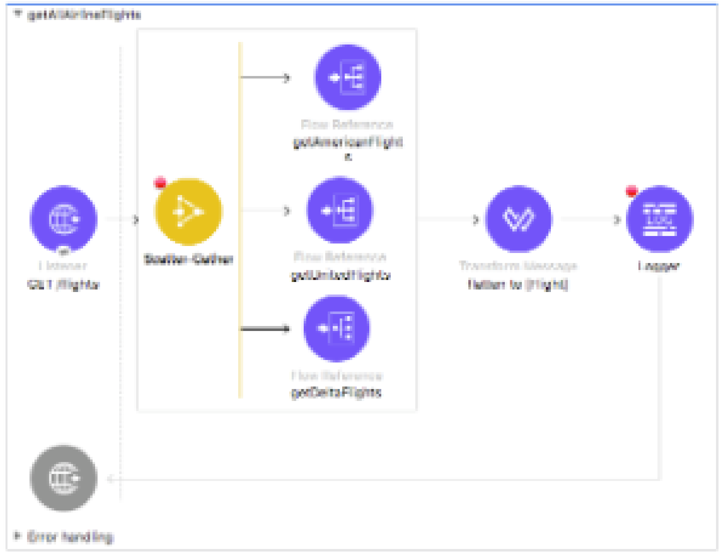
- Use a Scatter-Gather router to concurrently call all three flight services
- Use DataWeave to flatten multiple collections into one collection



Walkthrough 9-1: Multicast an event

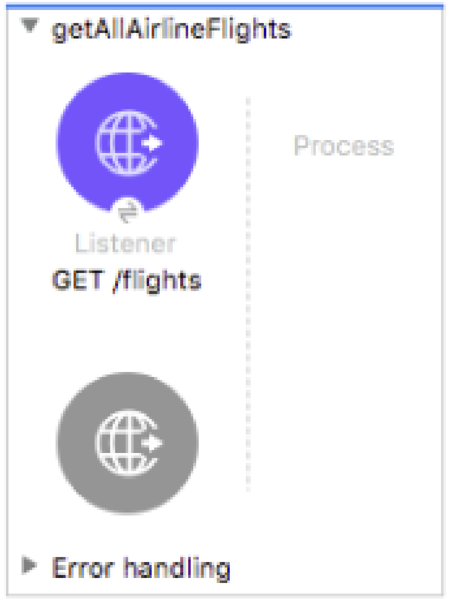
In this walkthrough, you create a flow that calls each of the three airline services and combines the results. You will:

- Use a Scatter-Gather router to concurrently call all three flight services.
- Use DataWeave to flatten multiple collections into one collection.



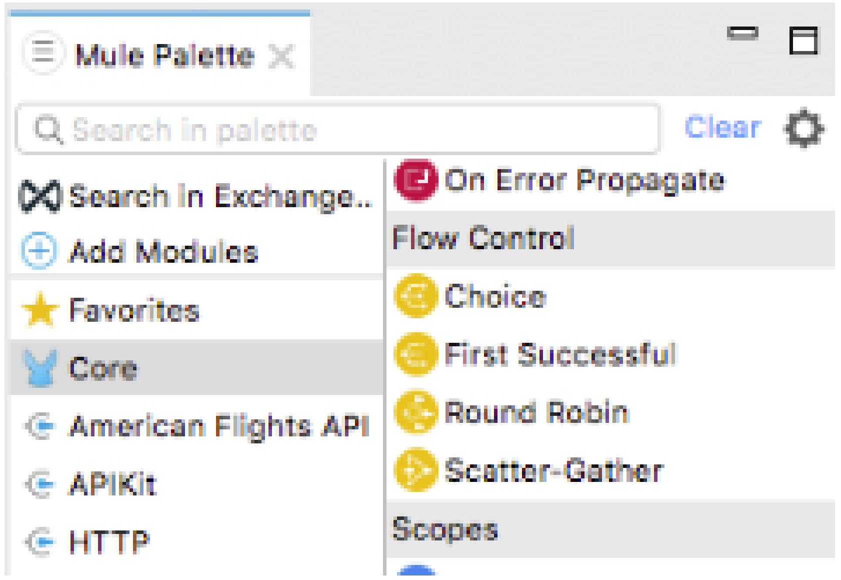
Create a new flow

1. Return to implementation.xml.
2. From the Mule Palette, drag an HTTP Listener and drop it at the top of the canvas.
3. Change the flow name to getAllAirlineFlights.
4. In the Listener properties view, set the display name to GET /flights.
5. Set the connector configuration to the existing HTTP_Listener_config.
6. Set the path to /flights and the allowed methods to GET.



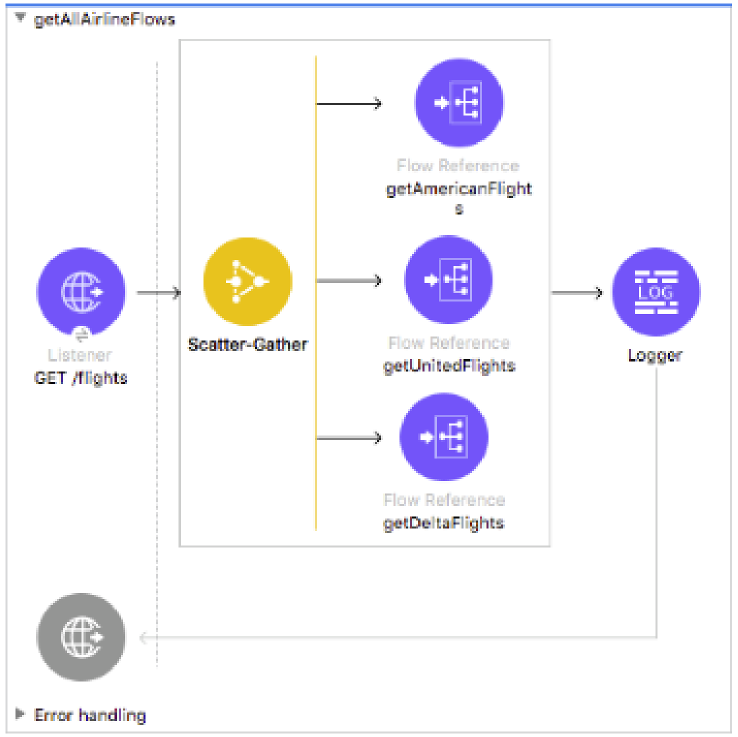
Browse the flow control elements in the Mule Palette

7. In the Core section of the Mule Palette, locate the Flow Control elements.



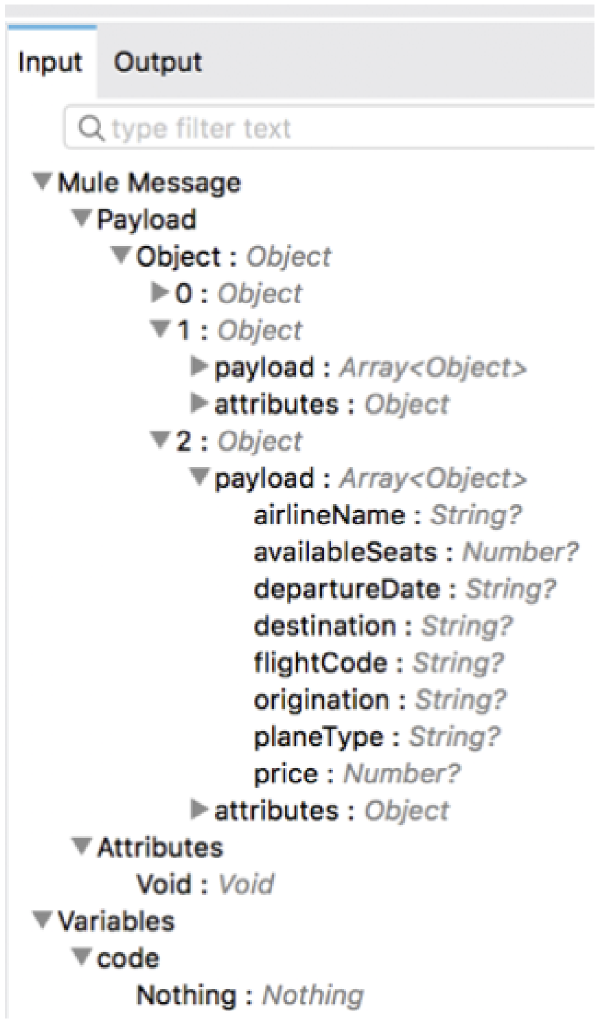
Add a Scatter-Gather to call all three airline services

8. Drag a Scatter-Gather flow control element from the Mule Palette and drop it in the process section of getAllAirlineFlights.
9. Add three Flow Reference components to the Scatter-Gather router.
10. In the first Flow Reference properties view, set the flow name and display name to getAmericanFlights.
11. Set the flow name and display name of the second Flow Reference to getUnitedFlights.
12. Set the flow name and display name of the third Flow Reference to getDeltaFlights.
13. Add a Logger after the Scatter-Gather.



Review the metadata for the Scatter-Gather output

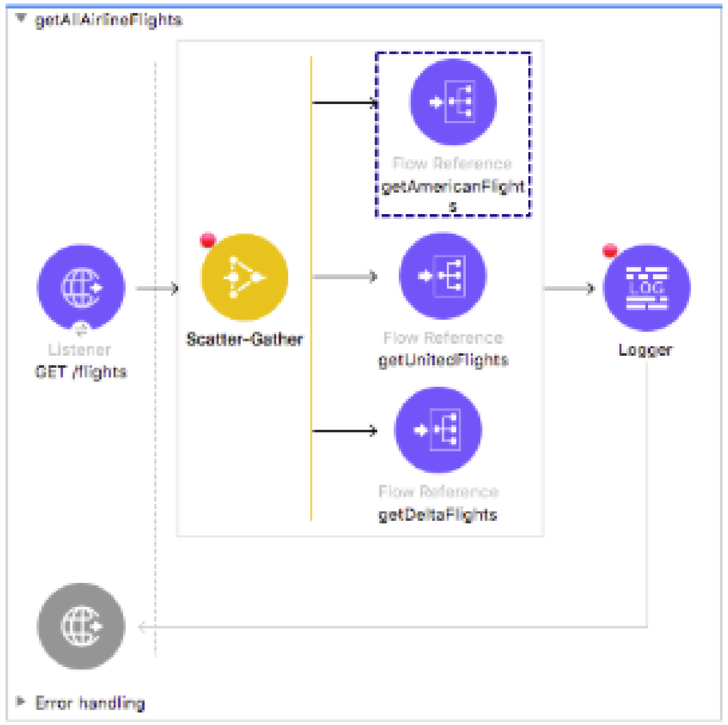
14. In the Logger properties view, explore the input payload structure in the DataSense Explorer.



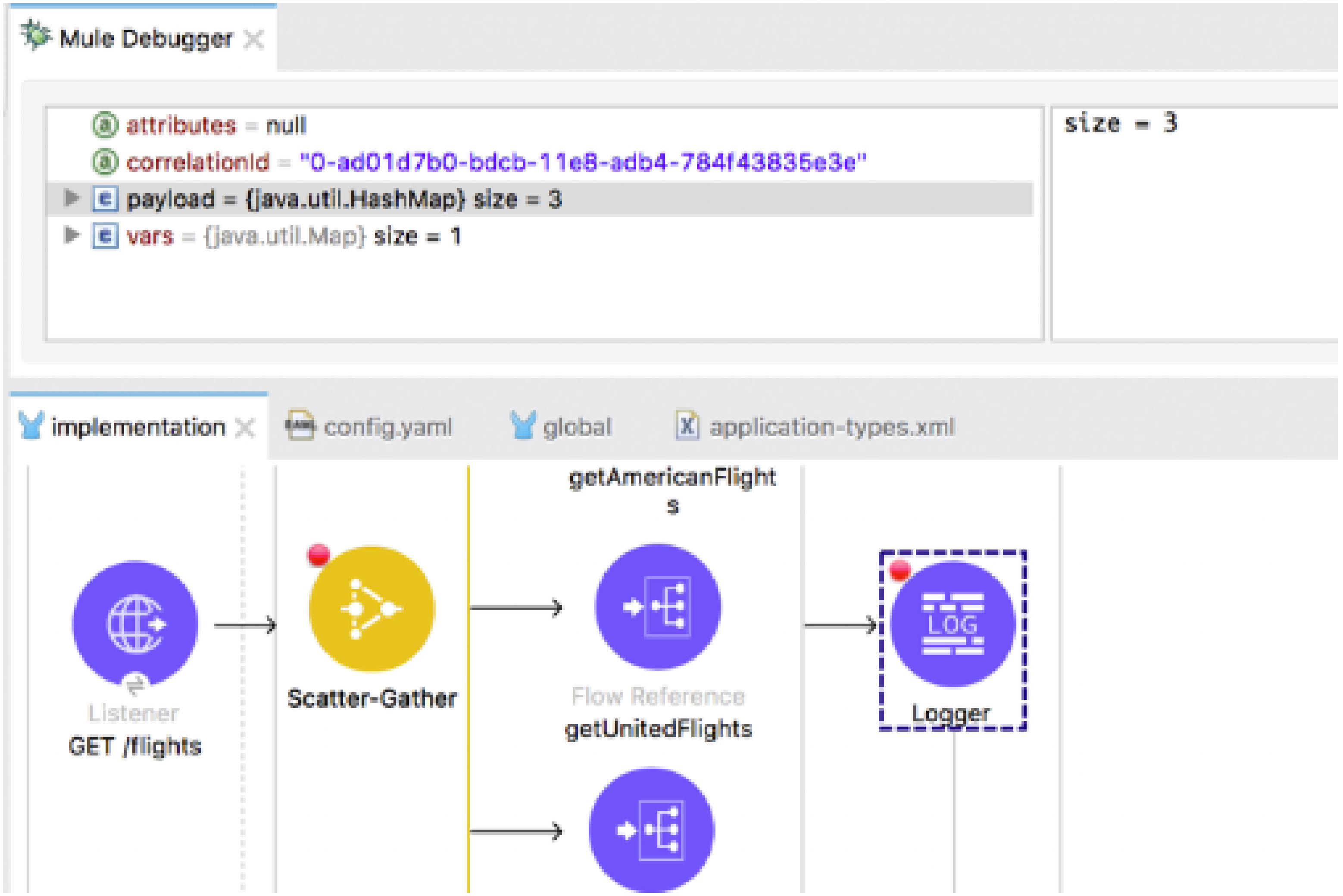
```
▼ Mule Message
  ▼ Payload
    ▼ Object : Object
      ▶ 0 : Object
      ▼ 1 : Object
        ▶ payload : Array<Object>
        ▶ attributes : Object
      ▼ 2 : Object
        ▼ payload : Array<Object>
            airlineName : String?
            availableSeats : Number?
            departureDate : String?
            destination : String?
            flightCode : String?
            origination : String?
            planeType : String?
            price : Number?
        ▶ attributes : Object
  ▼ Attributes
      Void : Void
▼ Variables
  ▼ code
      Nothing : Nothing
```

Debug the application

15. Add a breakpoint to the Scatter-Gather.
16. Add a breakpoint to the Logger.
17. Save the file to redeploy the project in debug mode.
18. In Advanced REST Client, change the URL to make a request to http://localhost/flights.
19. In the Mule Debugger, step through the application; you should step through each of the airline flows.

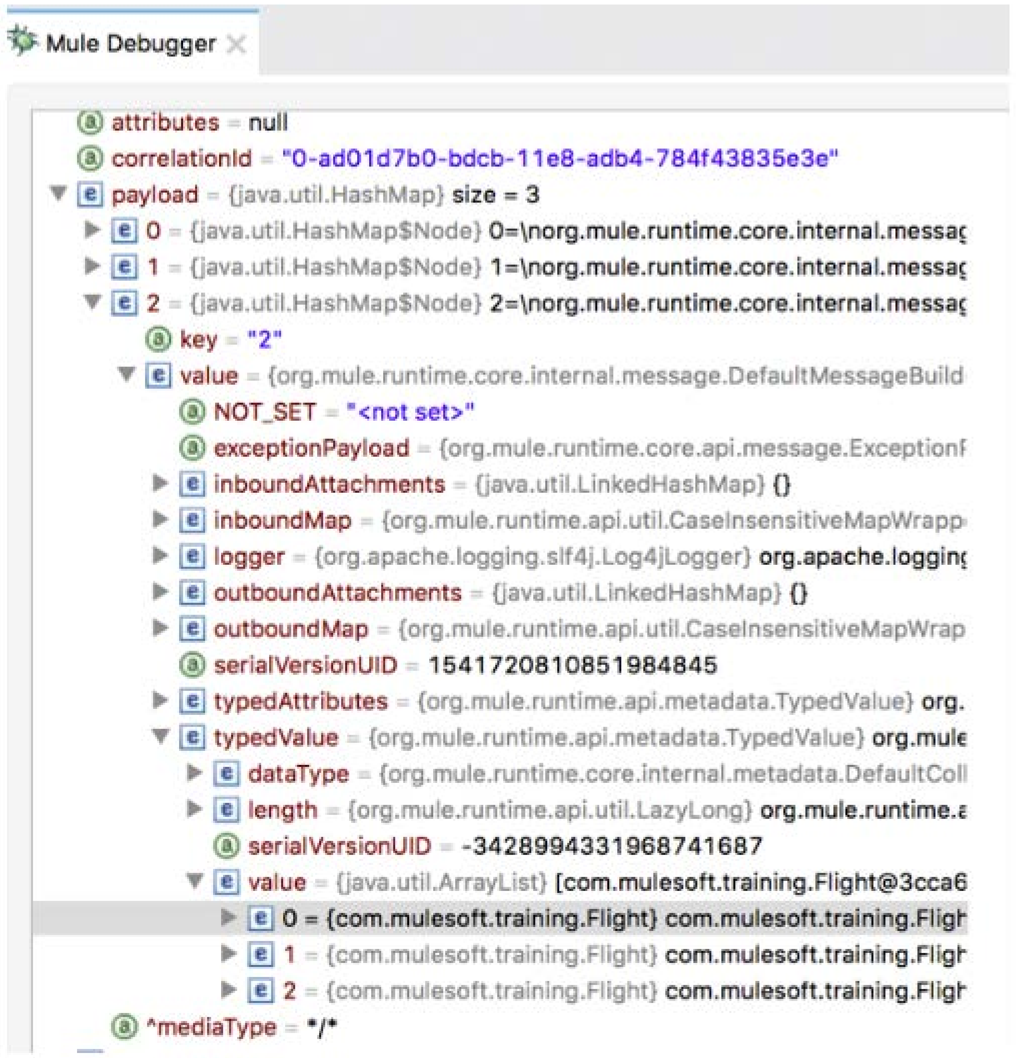


20. Stop at the Logger after the Scatter-Gather and explore the payload.


```
attributes = null
correlationId = "0-ad01d7b0-bdcb-11e8-adb4-784f43835e3e"
▶ payload = {java.util.HashMap} size = 3
▶ vars = {java.util.Map} size = 1
```

```
size = 3
```

21. Drill-down into one of the objects in the payload.

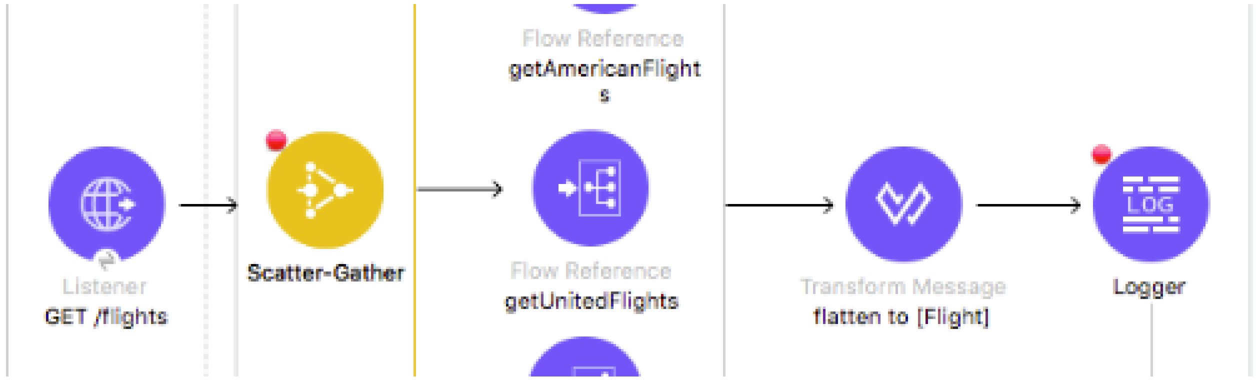


```
Mule Debugger

attributes = null
correlationId = "0-ad01d7b0-bdcb-11e8-adb4-784f43835e3e"
▼ payload = {java.util.HashMap} size = 3
  ▶ 0 = {java.util.HashMap$Node} 0=\norg.mule.runtime.core.internal.messag
  ▶ 1 = {java.util.HashMap$Node} 1=\norg.mule.runtime.core.internal.messag
  ▼ 2 = {java.util.HashMap$Node} 2=\norg.mule.runtime.core.internal.messag
      key = "2"
    ▼ value = {org.mule.runtime.core.internal.message.DefaultMessageBuild
        NOT_SET = "<not set>"
        exceptionPayload = {org.mule.runtime.core.api.message.Exception
      ▶ inboundAttachments = {java.util.LinkedHashMap} {}
      ▶ inboundMap = {org.mule.runtime.api.util.CaseInsensitiveMapWrapp
      ▶ logger = {org.apache.logging.slf4j.Log4jLogger} org.apache.logging
      ▶ outboundAttachments = {java.util.LinkedHashMap} {}
      ▶ outboundMap = {org.mule.runtime.api.util.CaseInsensitiveMapWrap
        serialVersionUID = 1541720810851984845
      ▶ typedAttributes = {org.mule.runtime.api.metadata.TypedValue} org.
      ▼ typedValue = {org.mule.runtime.api.metadata.TypedValue} org.mule
        ▶ dataType = {org.mule.runtime.core.internal.metadata.DefaultColl
        ▶ length = {org.mule.runtime.api.util.LazyLong} org.mule.runtime.a
          serialVersionUID = -3428994331968741687
        ▼ value = {java.util.ArrayList} [com.mulesoft.training.Flight@3cca6
          ▶ 0 = {com.mulesoft.training.Flight} com.mulesoft.training.Fligh
          ▶ 1 = {com.mulesoft.training.Flight} com.mulesoft.training.Fligh
          ▶ 2 = {com.mulesoft.training.Flight} com.mulesoft.training.Fligh
    ^mediaType = */*
```

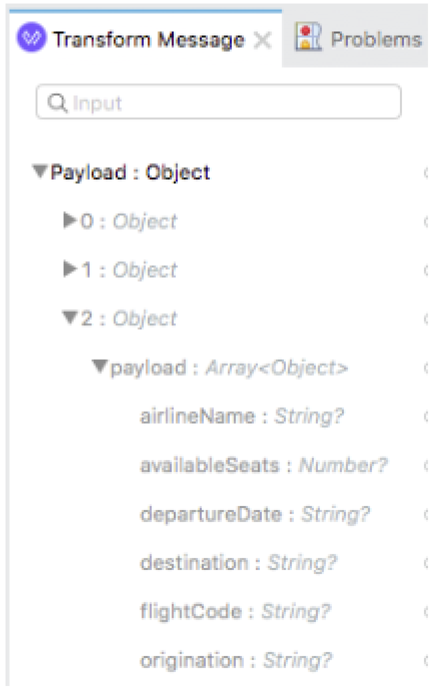
22. Step through the rest of the application and switch perspectives.
23. Return to Advanced REST Client and review the response; although the result is Java and you see a lot of characters, you should also see the Java for flight data for the three airlines.

Flatten the combined results

24. Return to Anypoint Studio.
25. In getAllAirlineFlights, add a Transform Message component before the Logger.
26. Change the display name to flatten to [Flight].



27. In the input section of the Transform Message properties view, review the payload data structure.



Payload : Object
- 0 : *Object*
- 1 : *Object*
- 2 : *Object*
  - payload : *Array<Object>*
    - airlineName : *String?*
    - availableSeats : *Number?*
    - departureDate : *String?*
    - destination : *String?*
    - flightCode : *String?*
    - origination : *String?*

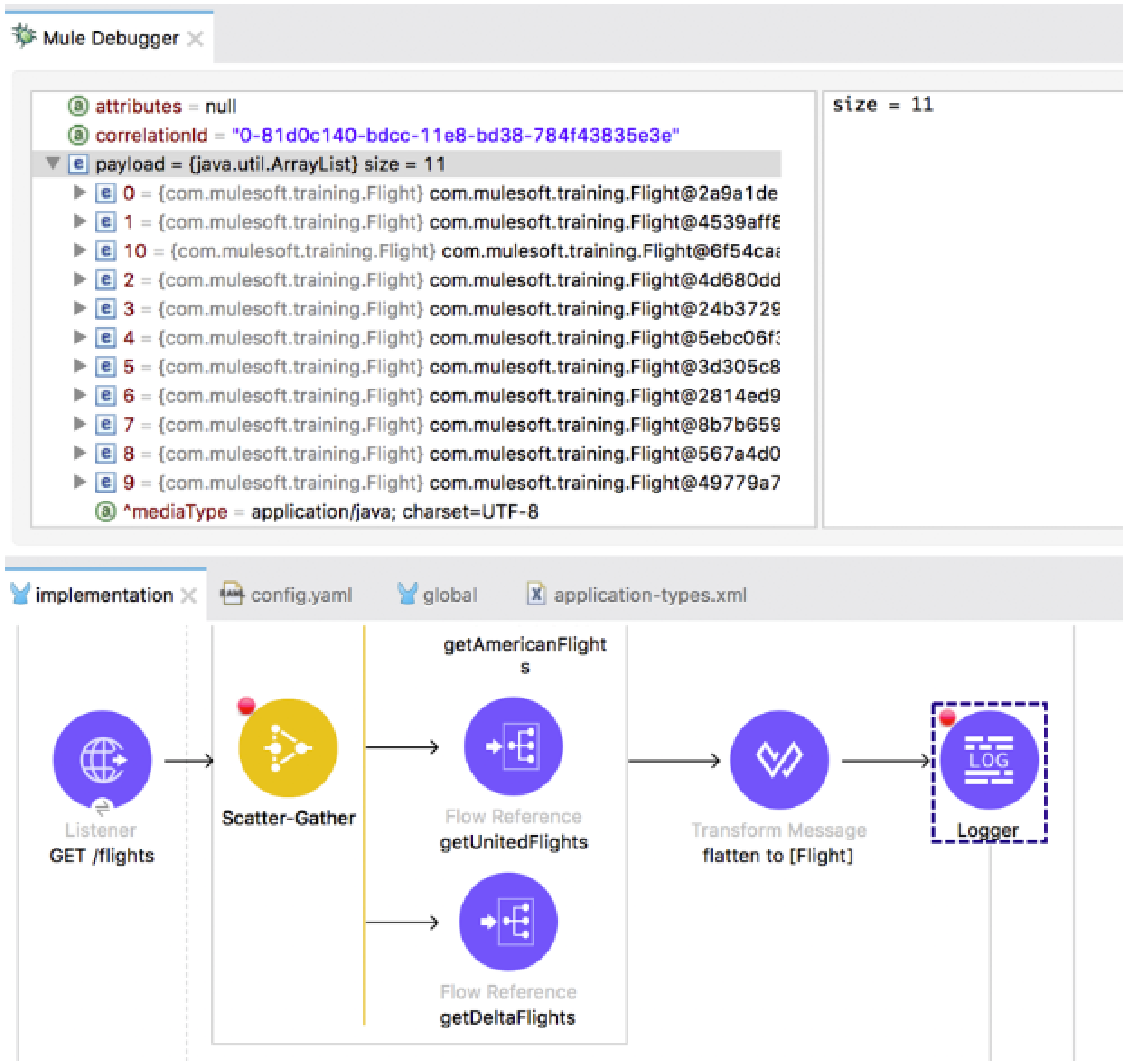
28. In the expression section, use the DataWeave flatten function to flatten the collection of objects into a single collection.

```
%dw 2.0
output application/java
---
flatten(payload..payload)
```

Debug the application

29. Save the file to redeploy the project.
30. In Advanced REST Client, make the same request to http://localhost:8081/flights.

31. In the Mule Debugger, press Resume until you are stopped at the Logger at the end of the Scatter-Gather; you should see the payload is now one ArrayList of Flights.



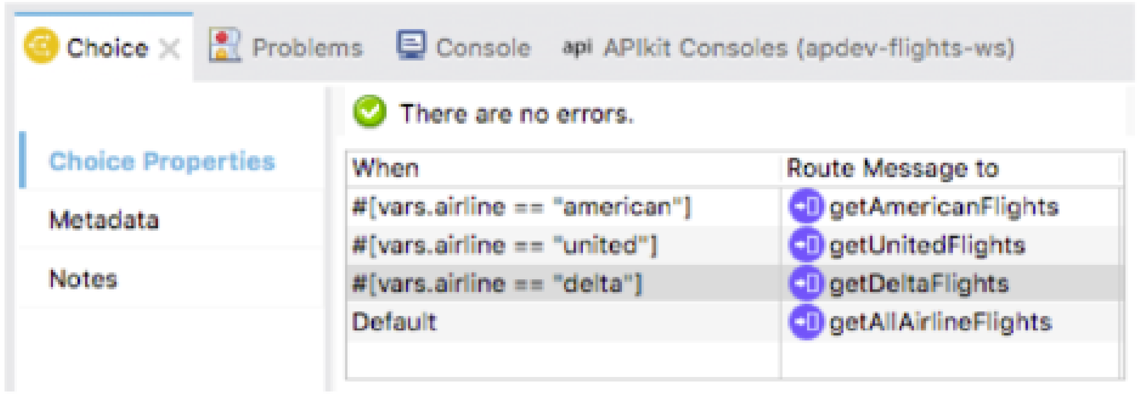
```
Mule Debugger

  attributes = null
  correlationId = "0-81d0c140-bdcc-11e8-bd38-784f43835e3e"
▼ payload = {java.util.ArrayList} size = 11
  ▶ 0 = {com.mulesoft.training.Flight} com.mulesoft.training.Flight@2a9a1de
  ▶ 1 = {com.mulesoft.training.Flight} com.mulesoft.training.Flight@4539aff8
  ▶ 10 = {com.mulesoft.training.Flight} com.mulesoft.training.Flight@6f54ca
  ▶ 2 = {com.mulesoft.training.Flight} com.mulesoft.training.Flight@4d680dd
  ▶ 3 = {com.mulesoft.training.Flight} com.mulesoft.training.Flight@24b3729
  ▶ 4 = {com.mulesoft.training.Flight} com.mulesoft.training.Flight@5ebc06f
  ▶ 5 = {com.mulesoft.training.Flight} com.mulesoft.training.Flight@3d305c8
  ▶ 6 = {com.mulesoft.training.Flight} com.mulesoft.training.Flight@2814ed9
  ▶ 7 = {com.mulesoft.training.Flight} com.mulesoft.training.Flight@8b7b659
  ▶ 8 = {com.mulesoft.training.Flight} com.mulesoft.training.Flight@567a4d0
  ▶ 9 = {com.mulesoft.training.Flight} com.mulesoft.training.Flight@49779a7
    ^mediaType = application/java; charset=UTF-8

size = 11
```

32. Step through the rest of the application and switch perspectives.

Routing events based on conditions

The Choice router

- Sends the event to one route based on conditional logic
- The conditions are written with DataWeave

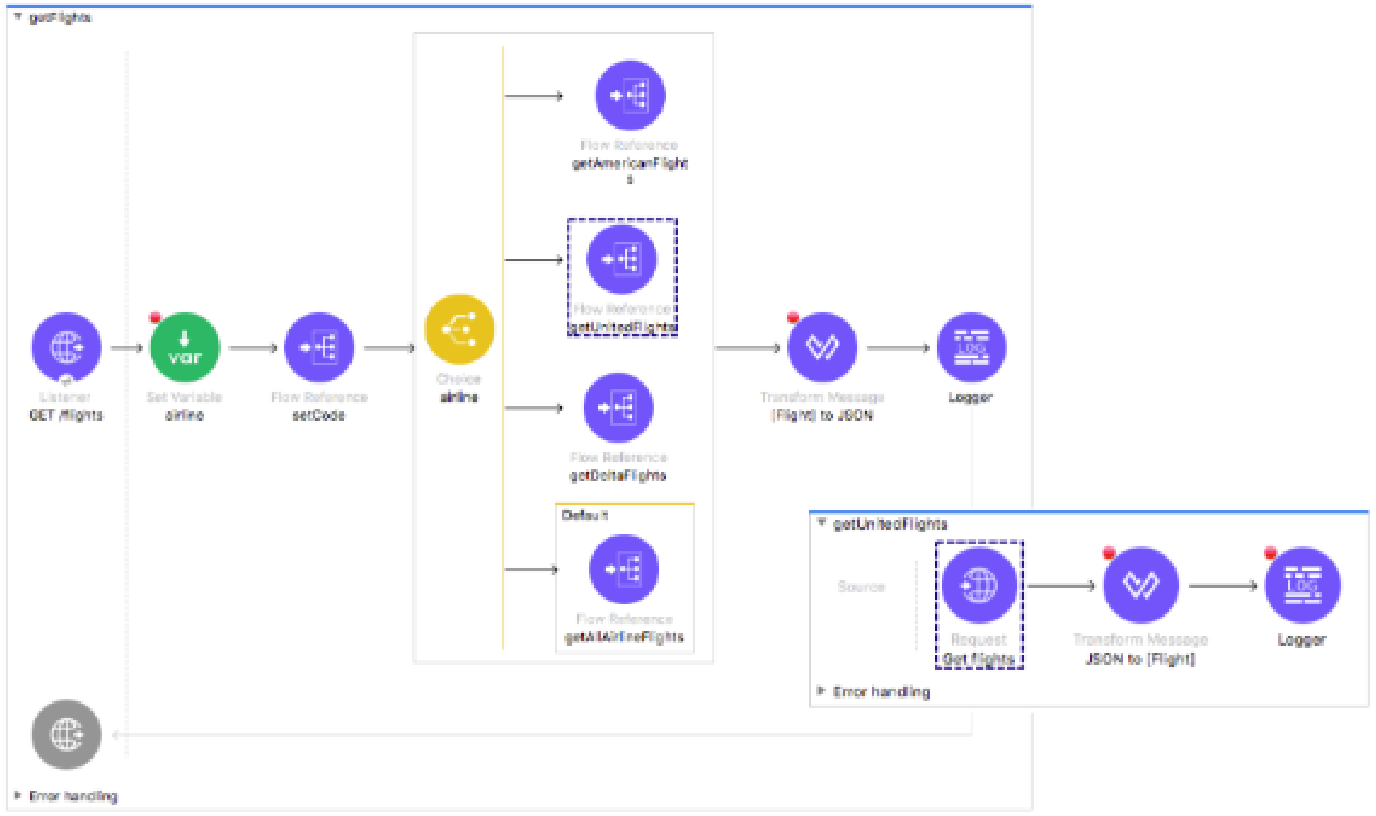


There are no errors.

When	Route Message to
#[vars.airline == "american"]	getAmericanFlights
#[vars.airline == "united"]	getUnitedFlights
#[vars.airline == "delta"]	getDeltaFlights
Default	getAllAirlineFlights

Walkthrough 9-2: Route events based on conditions

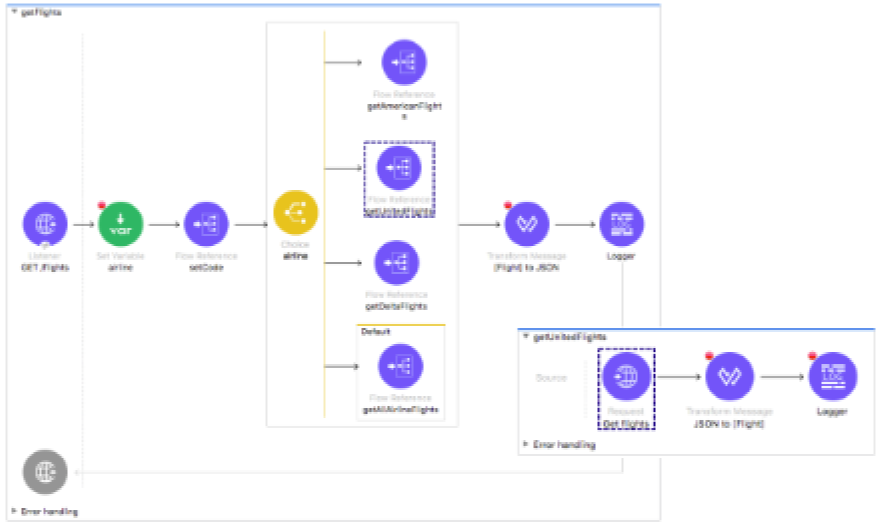
- Use a Choice router
- Use DataWeave expressions to set the router paths
- Route all flight requests through the router



Walkthrough 9-2: Route events based on conditions

In this walkthrough, you create a flow to route events to either the American, United, Delta, or get all airline flows based on the value of an airline query parameter. You will:

- Use a Choice router.
- Use DataWeave expressions to set the router paths.
- Route all flight requests through the router.



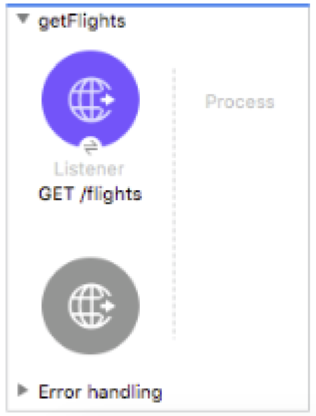
Look at possible airline values specified in the API

1. Return to the apdev-flights-ws project in Anypoint Studio.
2. Open mua-flights-api.raml in src/main/resources/api.
3. Locate the airline query parameter and its possible values.

```
airline:
  displayName: Airline
  required: false
  enum:
    - united
    - delta
    - american
```

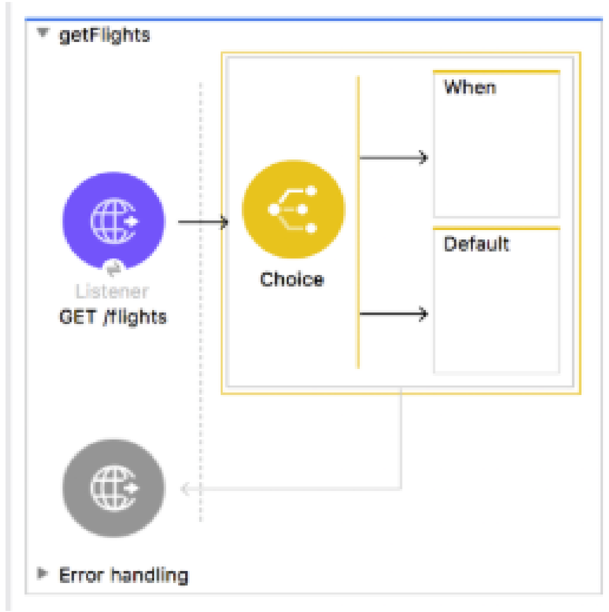
Create a new flow

4. Return to implementation.xml.
5. Drag a Flow scope from the Mule Palette and drop it at the top of the canvas above all the other flows.
6. Change the name of the flow to getFlights.
7. Move the GET /flights HTTP Listener from getAllAirlineFlights to the source section of getFlights.



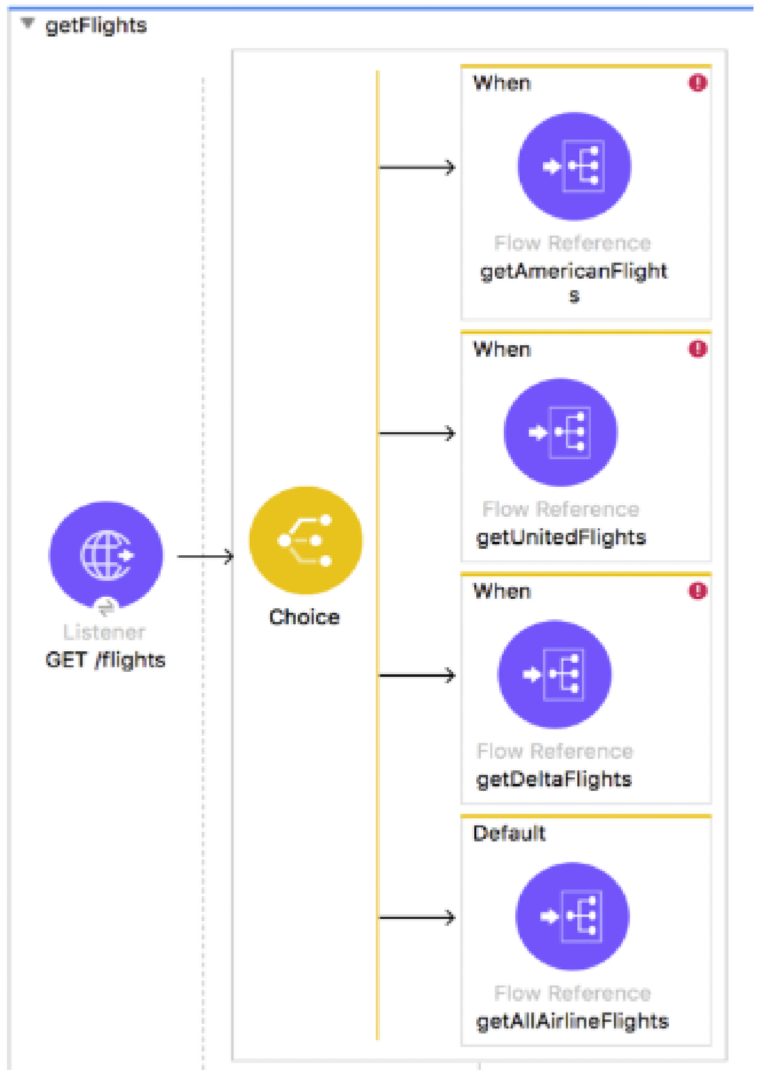
Add a Choice router

8. Drag a Choice flow control element from the Mule Palette and drop it in process section of getFlights.



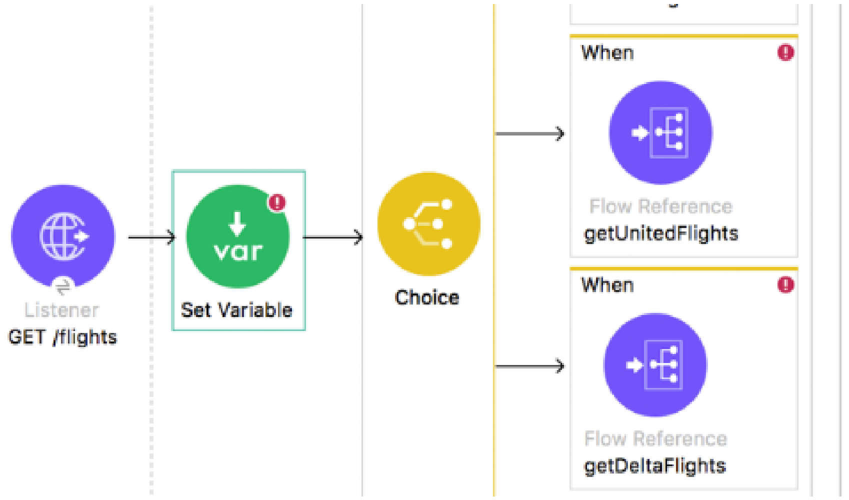
9. Add three Flow Reference components to the Choice router.
10. Add a Flow Reference component to the default branch of the router.
11. In the first Flow Reference properties view, set the flow name and display name to getAmericanFlights.
12. Set the flow names and display names for the other two Flow References to getUnitedFlights and getDeltaFlights.

13. For the Flow Reference in the default branch, set the flow name and display name to getAllAirlineFlights.



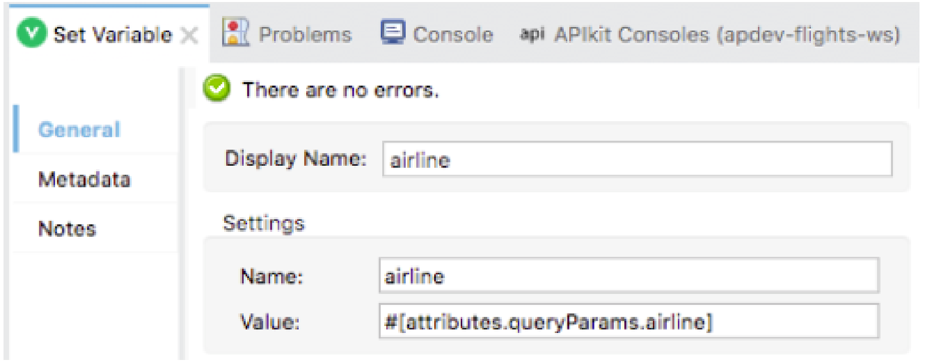
Store the airline query parameter in a variable

14. Add a Set Variable transformer before the Choice router.



15. In the Set Variable properties view, set the display name and name to airline.
16. Set the value to a query parameter called airline.

```
#[attributes.queryParams.airline]
```



There are no errors.

Display Name: airline

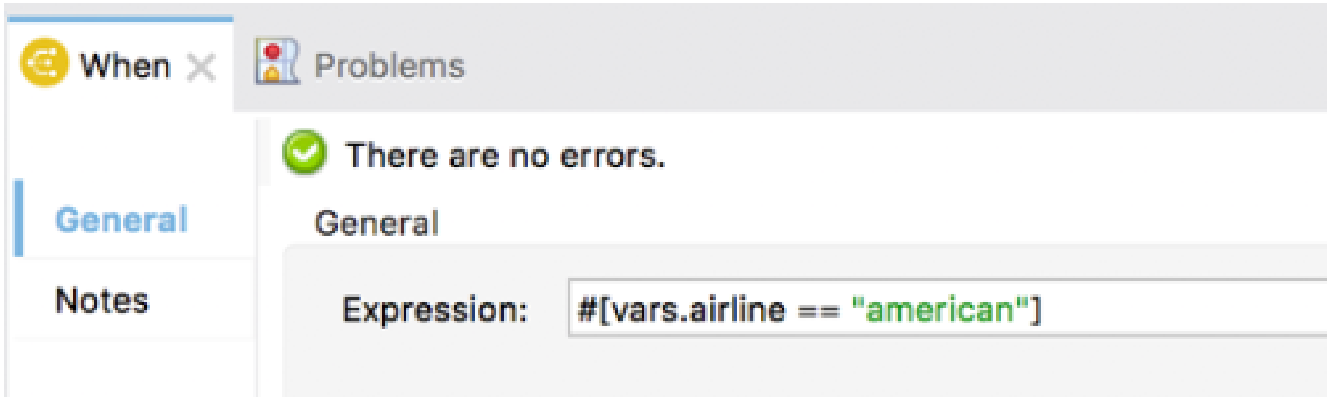
Name: airline

Value: #[attributes.queryParams.airline]

Configure the Choice router

17. In the Choice router getAmericanFlights flow reference branch, click the When scope.
18. In the When properties view, add an expression to check if the airline variable is equal to american.

    ```
    #[vars.airline == "american"]
    ```

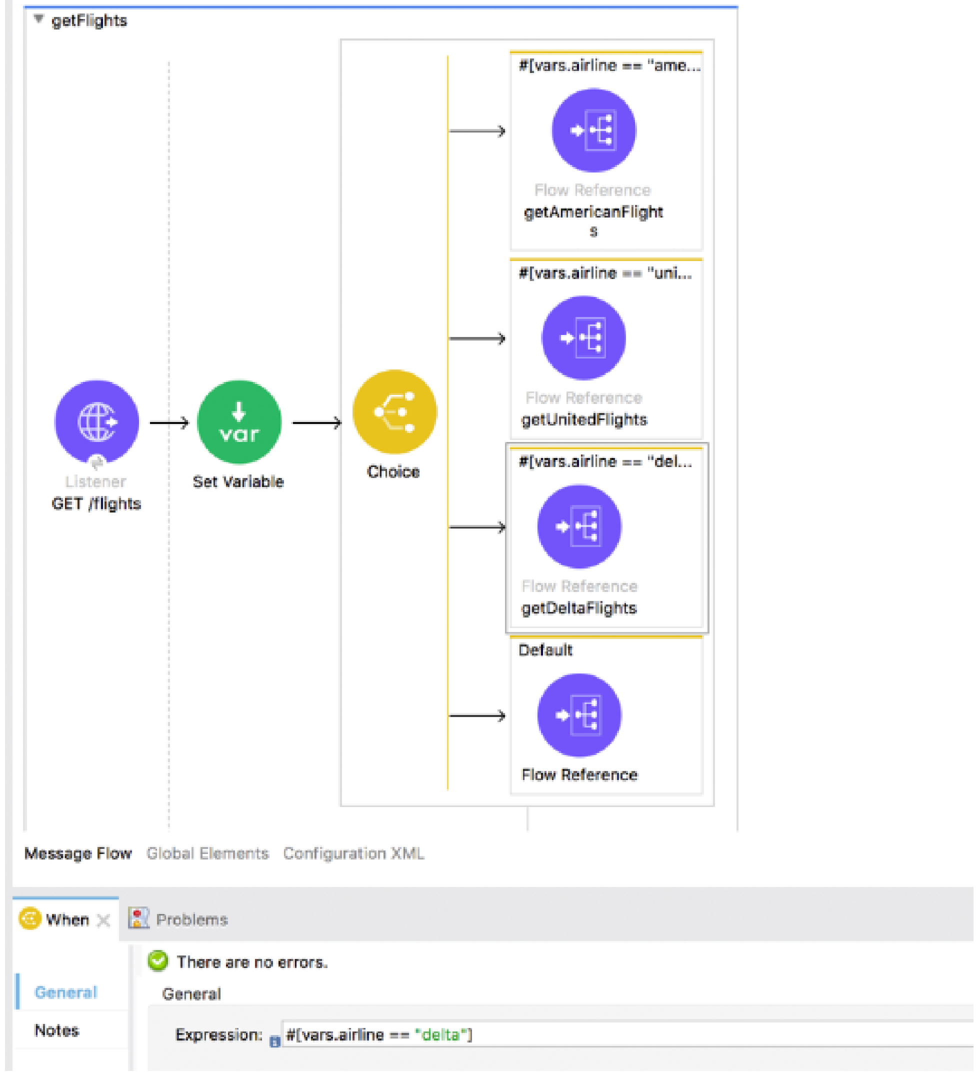
    There are no errors.

    General

    Expression: #[vars.airline == "american"]

19. Set a similar expression for the United route, routing to it when vars.airline is equal to united.

20. Set a similar expression for the Delta route, routing to it when vars.airline is equal to delta.



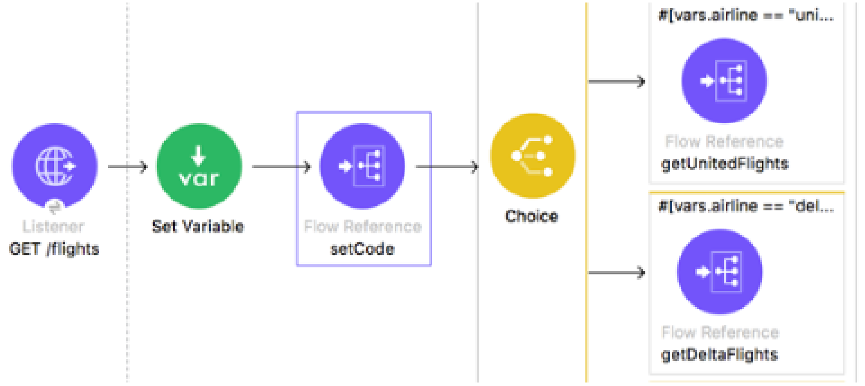
There are no errors.

General

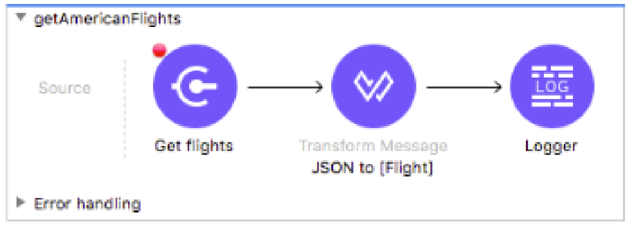
Expression: #[vars.airline == "delta"]

Route all requests through the router

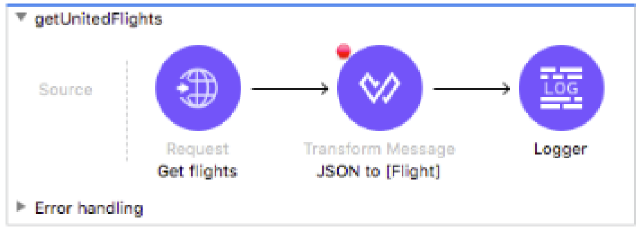
21. Add a Flow Reference to the flow before the Choice router.
22. Set the flow name and display name to setCode.



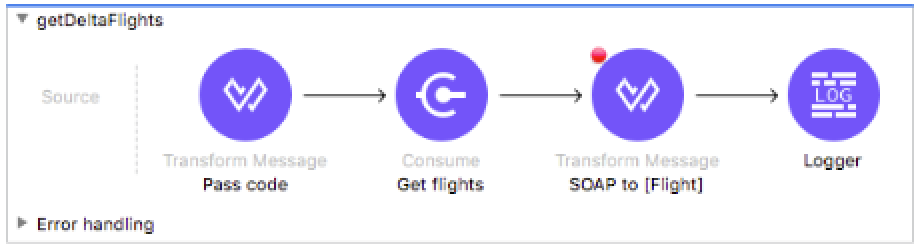
23. In getAmericanFlights, delete the HTTP Listener and the setCode Flow Reference.



24. In getUnitedFlights, delete the HTTP Listener and the setCode Flow Reference.

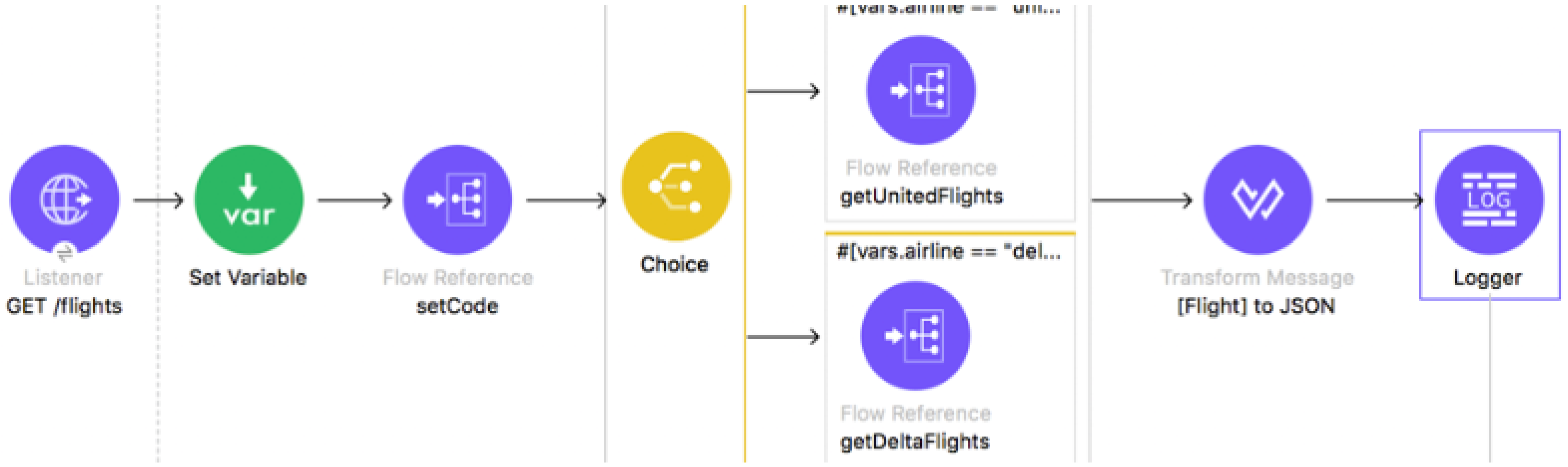


25. In getDeltaFlights, delete the HTTP Listener and the setCode Flow Reference.



Return JSON from the flow

26. Add a Transform Message component after the Choice router.
27. Change its display name to [Flight] to JSON.
28. Add a Logger at the end of the flow.

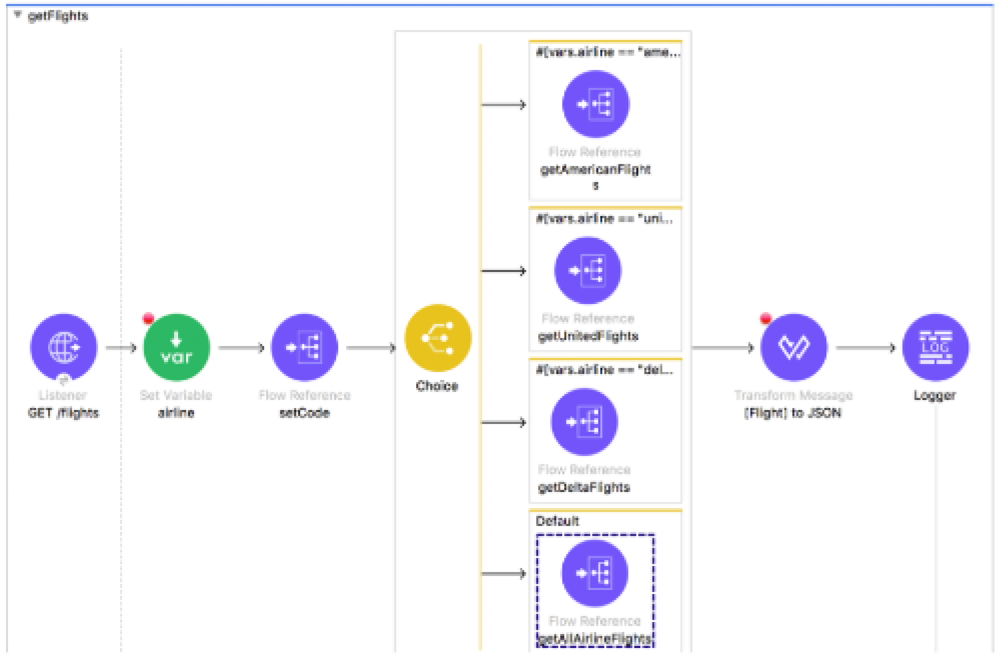


29. In the Transform Message properties view, set the output type to JSON and the output to payload.

```
%dw 2.0
output application/json
---
payload
```

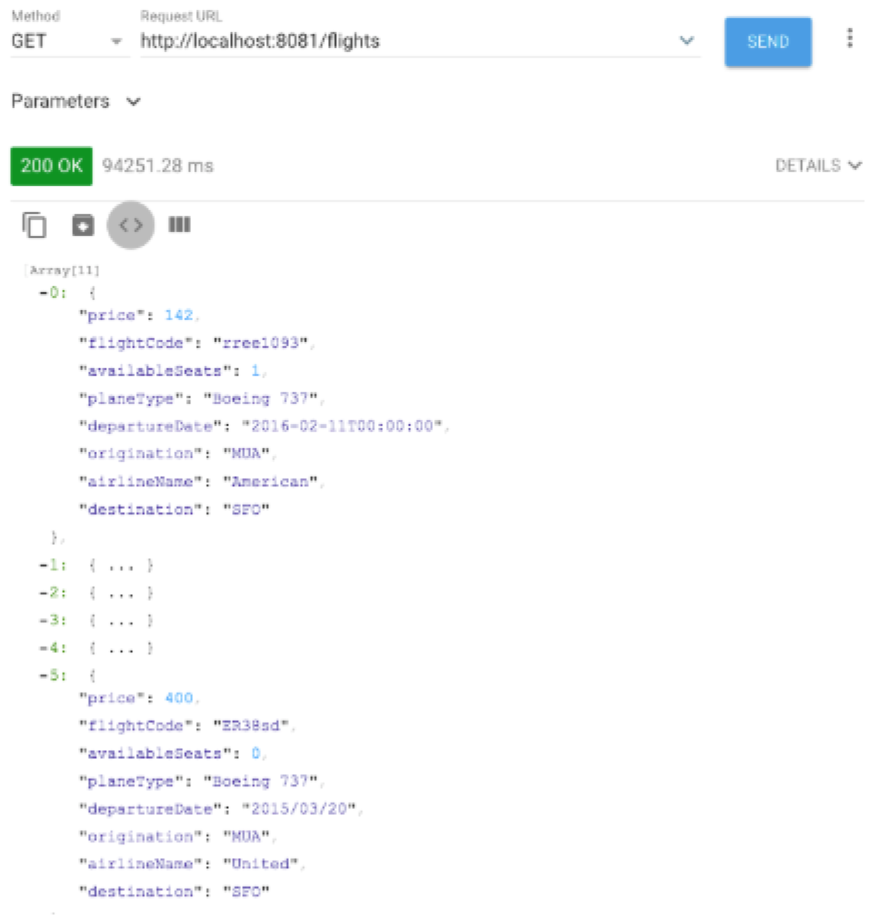
Debug the application

30. Add a breakpoint to the Set Variable transformer at the beginning of getFlights.
31. Add a breakpoint to the Transform Message component after the Choice router.
32. Save the file to redeploy the project in debug mode.
33. In Advanced REST Client, make a request to http://localhost:8081/flights.
34. In the Mule Debugger, step through the application; you should see the Choice router pass the event to the default branch.



35. Resume to the Transform Message component after the Choice router; the payload should be an ArrayList of Flight objects
36. Step to the Logger; the payload should be JSON.
37. Step to the end of the application.

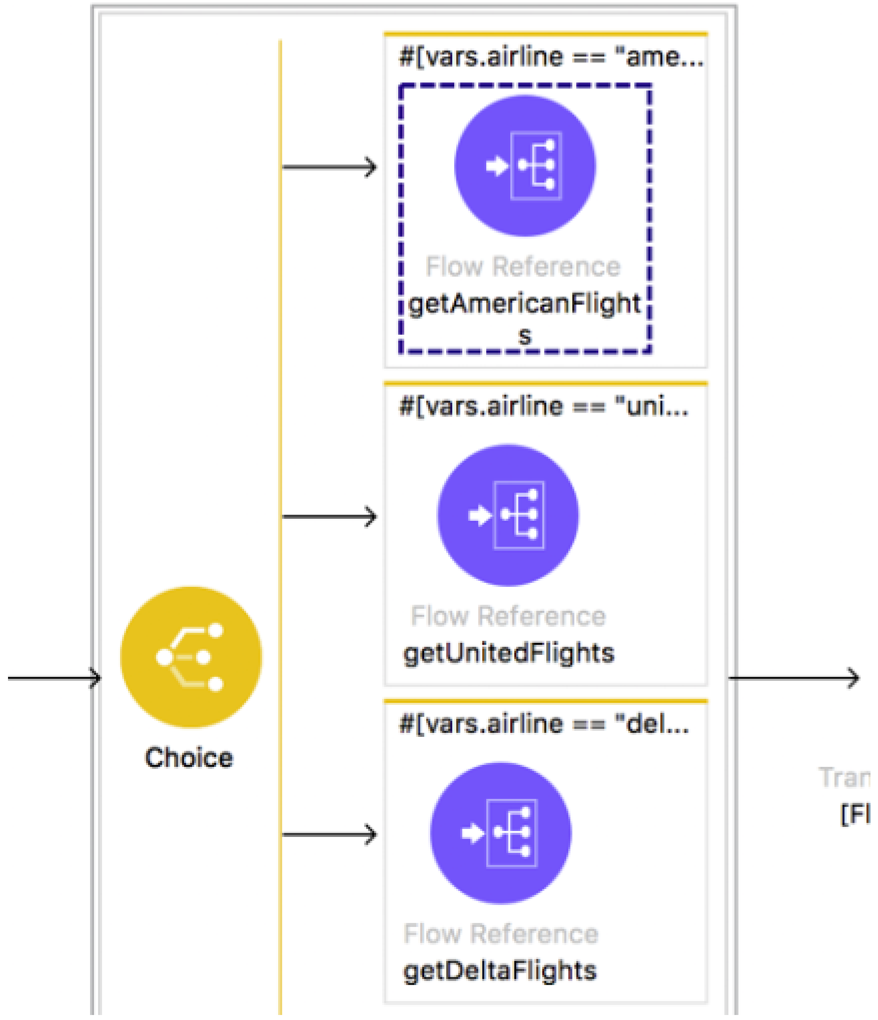
38. Return to Advanced REST Client; you should see American, United, and Delta flights to SFO (the default airport code).



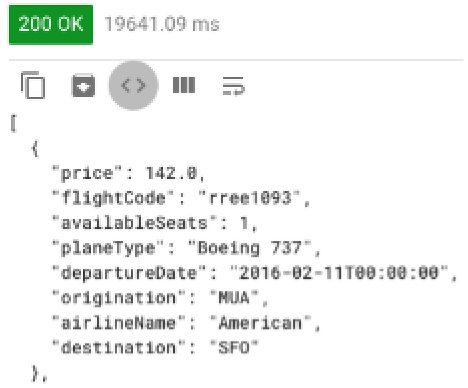
```
Array[11]
 -0:  {
     "price": 142,
     "flightCode": "rree1093",
     "availableSeats": 1,
     "planeType": "Boeing 737",
     "departureDate": "2016-02-11T00:00:00",
     "origination": "MUA",
     "airlineName": "American",
     "destination": "SFO"
   },
 -1:  { ... }
 -2:  { ... }
 -3:  { ... }
 -4:  { ... }
 -5:  {
     "price": 400,
     "flightCode": "ER38sd",
     "availableSeats": 0,
     "planeType": "Boeing 737",
     "departureDate": "2015/03/20",
     "origination": "MUA",
     "airlineName": "United",
     "destination": "SFO"
```

39. Add an airline query parameter set to american and send the request: http://localhost:8081/flights?airline=american.

40. In the Mule Debugger, step through the rest of the application; you should see the message passed to getAmericanFlights and then back to getFlights.



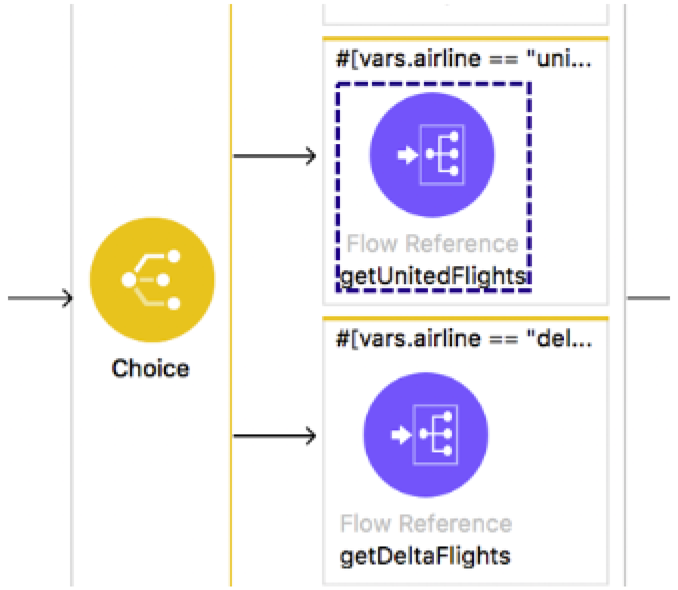
41. Return to Advanced REST Client; you should see only American flights to SFO returned.



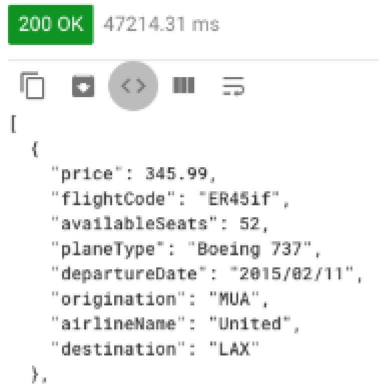
```
[
  {
    "price": 142.0,
    "flightCode": "rree1093",
    "availableSeats": 1,
    "planeType": "Boeing 737",
    "departureDate": "2016-02-11T00:00:00",
    "origination": "MUA",
    "airlineName": "American",
    "destination": "SFO"
  },
```

42. Change the airline to delta and add a second query parameter code set to LAX: http://localhost:8081/flights?airline=united&code=LAX.
43. Send the request.

44. In the Mule Debugger, step through the application; the message should be routed to the United branch.



45. Return to Advanced REST Client; you should see only United flights to LAX are returned.

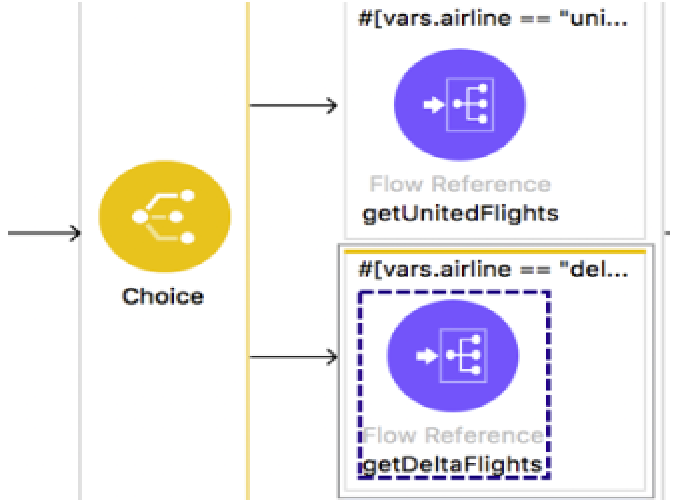


```
[
  {
    "price": 345.99,
    "flightCode": "ER45if",
    "availableSeats": 52,
    "planeType": "Boeing 737",
    "departureDate": "2015/02/11",
    "origination": "MUA",
    "airlineName": "United",
    "destination": "LAX"
  },
```

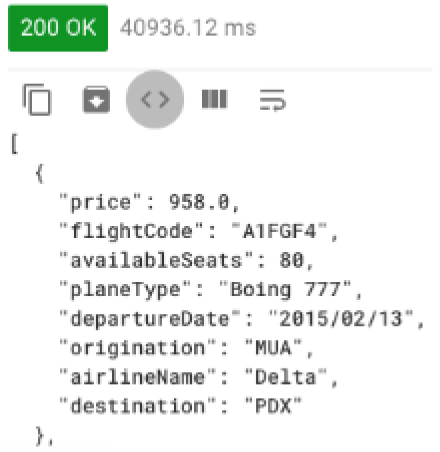
46. Change the airline to delta and the code to PDX: http://localhost:8081/flights?airline=united&code=PDX.

47. Send the request.

48. In the Mule Debugger, step through the application; the message should be routed to the Delta branch.



49. Return to Advanced REST Client; you should see only Delta flights to PDX are returned.

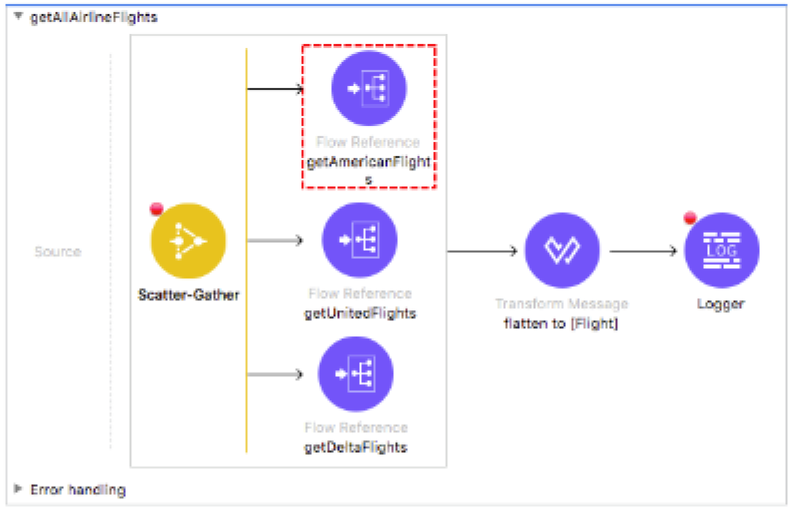


```
[
  {
    "price": 958.0,
    "flightCode": "A1FGF4",
    "availableSeats": 80,
    "planeType": "Boing 777",
    "departureDate": "2015/02/13",
    "origination": "MUA",
    "airlineName": "Delta",
    "destination": "PDX"
  },
```

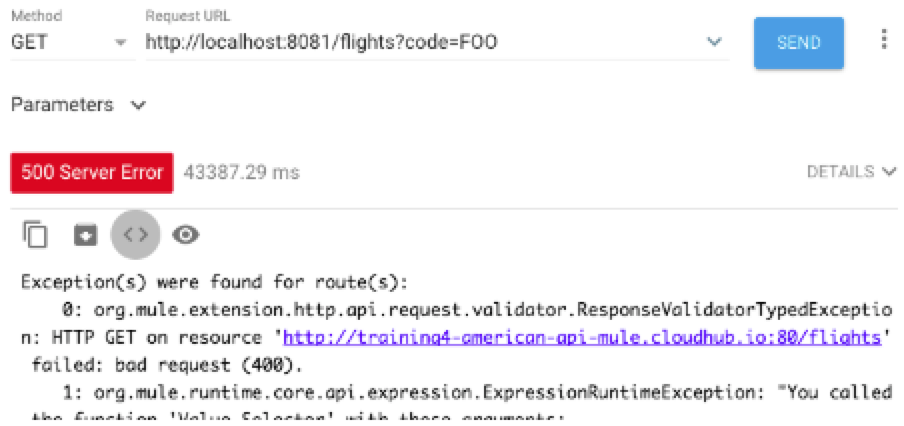
Note: This JSON structure does not match that specified in the API: mua-flights-api.raml in src/main/resources/api. The data ***should****, of course, be transformed so the response from the API matches this specification – but we are going to hold off doing that right now so that the scenario stays simpler for the error handling module (there is one less error to handle at the beginning).*

Test the application with a destination that has no flights

50. Remove the airline parameter and set the code to FOO: http://localhost:8081/flights?code=FOO.
51. In the Mule Debugger, step through the application; you should see multiple errors.



52. Return to Advanced REST Client; you should get a 500 Server Response and an exception.



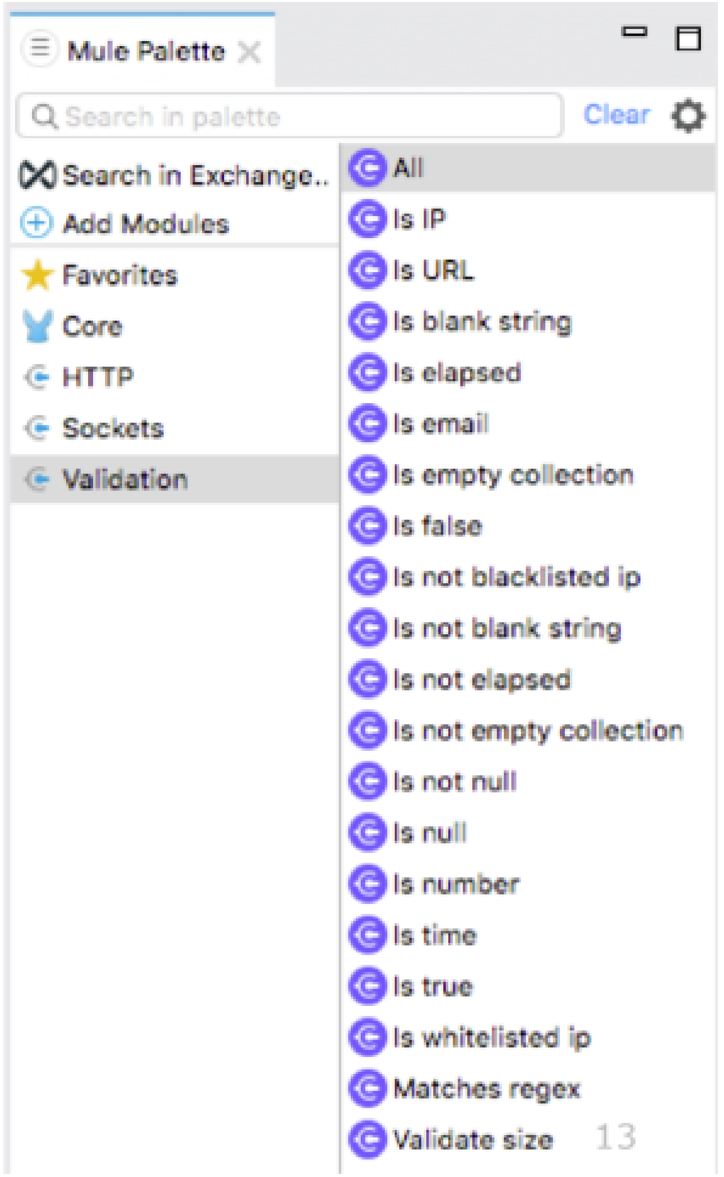
```
Exception(s) were found for route(s):
    0: org.mule.extension.http.api.request.validator.ResponseValidatorTypedExceptio
n: HTTP GET on resource 'http://training4-american-api-mule.cloudhub.io:80/flights'
 failed: bad request (400).
    1: org.mule.runtime.core.api.expression.ExpressionRuntimeException: "You called
```

Validating events

Validators

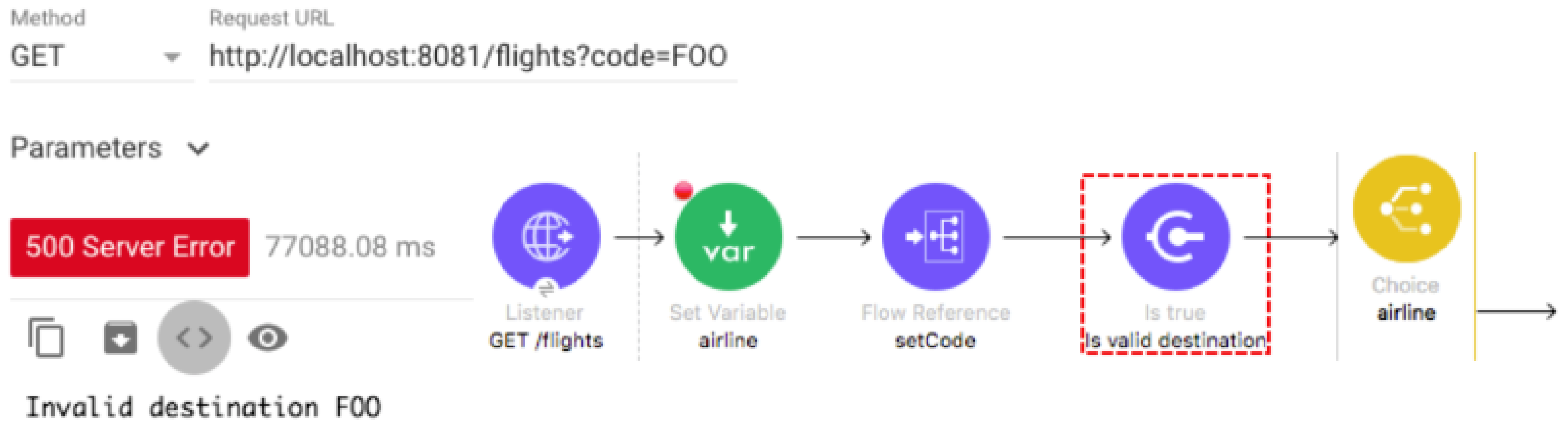
- Provide a way to test some conditions are met and throw an error if the validation fails

- To use
  - Add the Validation module to a project
  - Select a validation operation



Walkthrough 9-3: Validate events

- Add the Validation module to a project
- Use an Is true validator to check if a query parameter called code with a value of SFO, LAX, CLE, PDX, or PDF is sent with a request
- Return a custom error message is the condition is not met



http://localhost:8081/flights?code=FOO

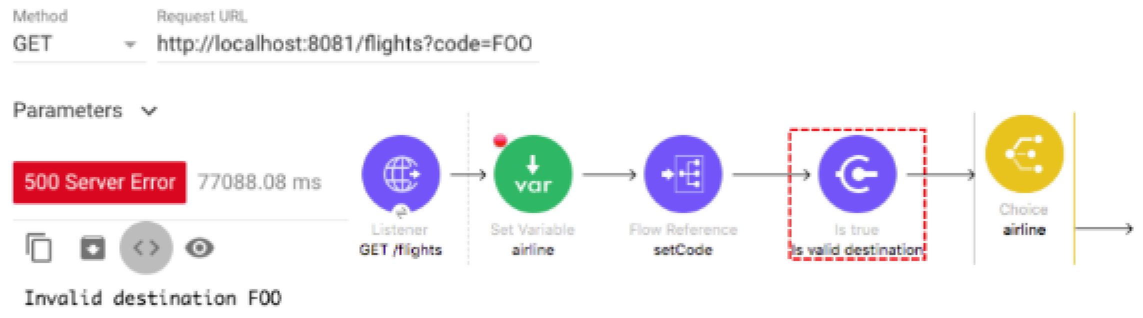
500 Server Error 77088.08 ms

```
Invalid destination FOO
```

Walkthrough 9-3: Validate events

In this walkthrough, you use a validator to check if a query parameter called code with a value of SFO, LAX, CLE, PDX, or PDF is sent with a request and to throw an error if it is not. You will:

- Add the Validation module to a project.
- Use an Is true validator to check if a query parameter called code with a value of SFO, LAX, CLE, PDX, or PDF is sent with a request.
- Return a custom error message is the condition is not met.

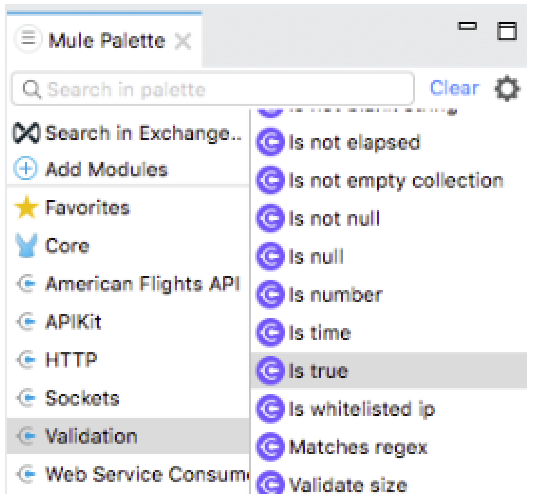

Method GET
Request URL http://localhost:8081/flights?code=FOO

500 Server Error 77088.08 ms

```
Invalid destination FOO
```

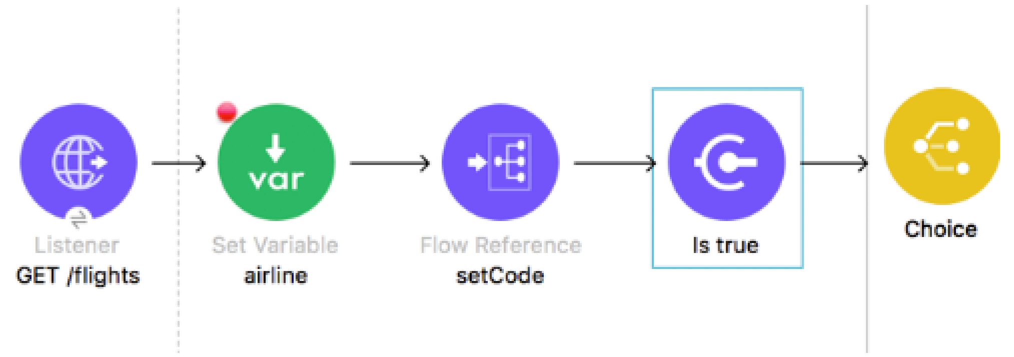
Add the Validation module to the project

1. Return to implementation.xml.
2. In the Mule Palette, select Add Modules.
3. Select the Validation module in the right side of the Mule Palette and drag and drop it into the left side.
4. If you get a Select module version dialog box, select the latest version and click Add.



Use the Is true validator to check for a valid destination code

5. Locate the Is true validator in the right side of the Mule Palette.
6. Drag and drop the Is true validator after the setCode Flow Reference in the getFlights flow.



7. In the Is true properties view, set the display name to: Is valid destination.
8. Change the expression from False (Default) to Expression.
9. Add a DataWeave expression to check if the code variable is one of the five destination codes.

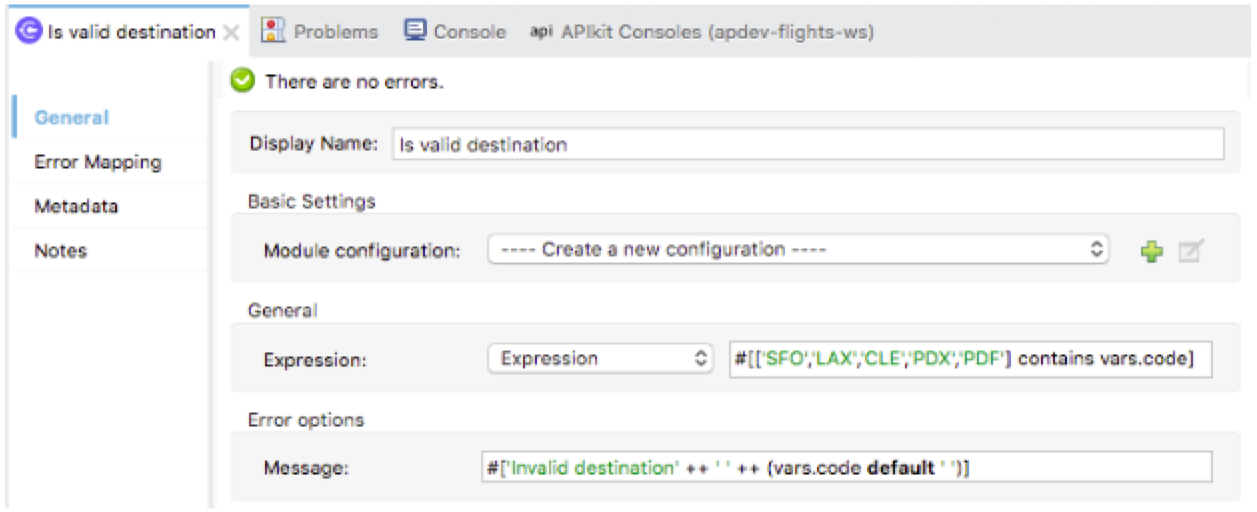
```
#[['SFO','LAX','CLE','PDX','PDF'] contains vars.code]
```

Note: You can copy this expression from the course snippets.txt file.

10. Set the error message to Invalid destination followed by the provided code.

```
#['Invalid destination' ++ ' ' ++ (vars.code default ' ')]
```

Note: You can copy this expression from the course snippets.txt file.



Is valid destination | Problems | Console | APIkit Consoles (apdev-flights-ws)

There are no errors.

General | Error Mapping | Metadata | Notes

Display Name: Is valid destination

Basic Settings

Module configuration: ---- Create a new configuration ----

General

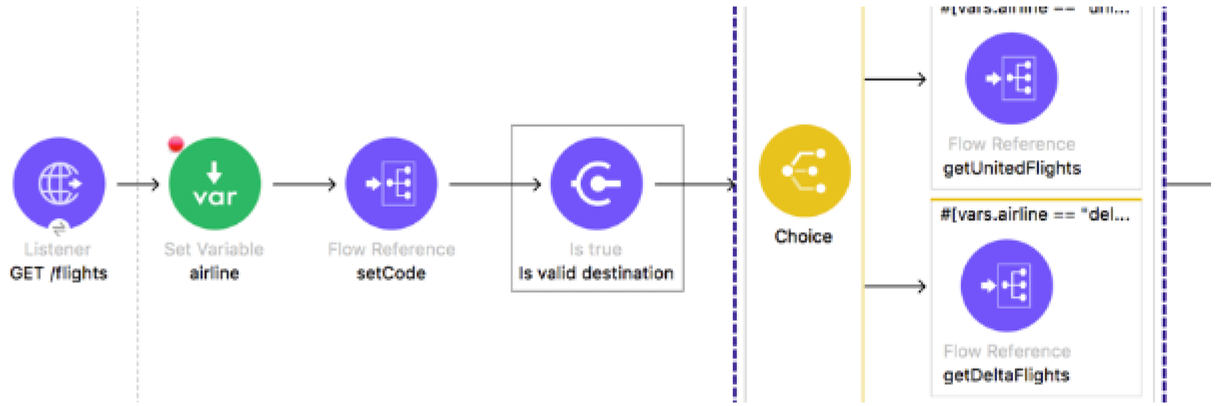
Expression: Expression #[['SFO','LAX','CLE','PDX','PDF'] contains vars.code]

Error options

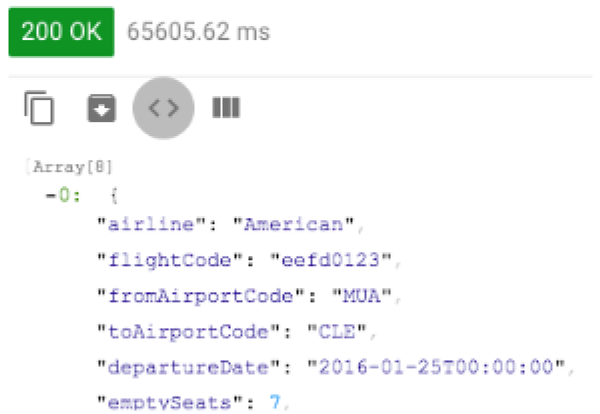
Message: #['Invalid destination' ++ ' ' ++ (vars.code **default** ' ')]

Debug the application

11. Save the file to redeploy the project in debug mode.
12. In Advanced REST Client, change the code to make a request to http://localhost:8081/flights?code=CLE.
13. In the Mule Debugger, step past the validator; you should see the code is valid and application execution steps to the Choice router.



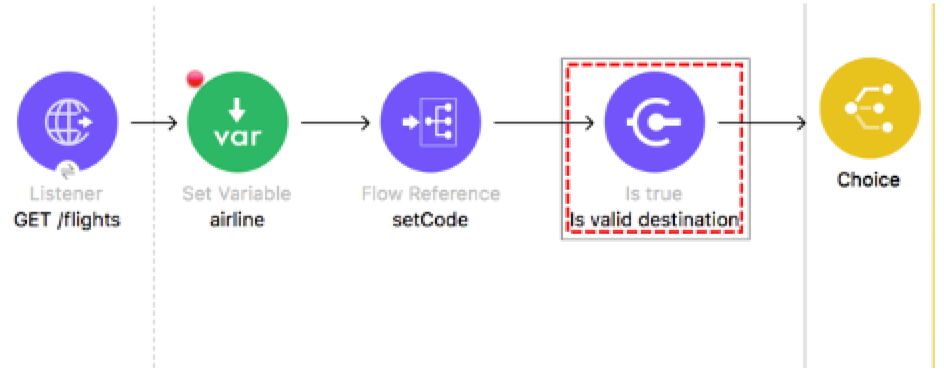
14. Resume through the rest of the application.
15. In Advanced REST Client, you should see flights.



200 OK 65605.62 ms

```
[Array[8]
  -0:  {
      "airline": "American",
      "flightCode": "eefd0123",
      "fromAirportCode": "MUA",
      "toAirportCode": "CLE",
      "departureDate": "2016-01-25T00:00:00",
      "emptySeats": 7,
```

16. Change the code and make a request to http://localhost:8081/flights?code=FOO.
17. In the Mule Debugger, step to the validator; you should see an error.

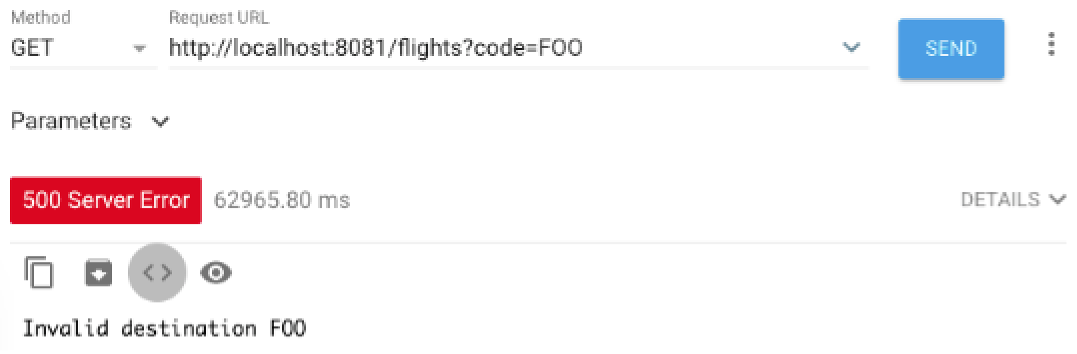


18. Step again; you should see your Invalid destination message in the console.

```
ERROR 2018-04-20 11:55:44,836 [[MuleRuntime].cpuLight.15: [apdev-flights-ws].getFlights.CPU_LITE @65ffdff1] [event: 0-48
bd68f0-44cc-11e8-a62a-8c85900da7e5] org.mule.runtime.core.internal.exception.OnErrorPropagateHandler:
********************************************************************************
Message               : Invalid destination FOO.
Error type            : VALIDATION:INVALID_BOOLEAN
Element               : getFlights/processors/2 @ apdev-flights-ws:implementation.xml:15 (Is valid destination)
Element XML           : <validation:is-true doc:name="Is valid destination" doc:id="e6d96ea1-71ee-45f1-9661-9048ce5382c9
" expression="#[['SFO','LAX','CLE','PDX','PDF'] contains vars.code]" message="#['Invalid destination' ++ ' ' ++ vars.cod
e]"></validation:is-true>
```

19. Resume through the rest of the application.

20. Return to Advanced REST Client; you should get a 500 Server Error with your Invalid destination message.



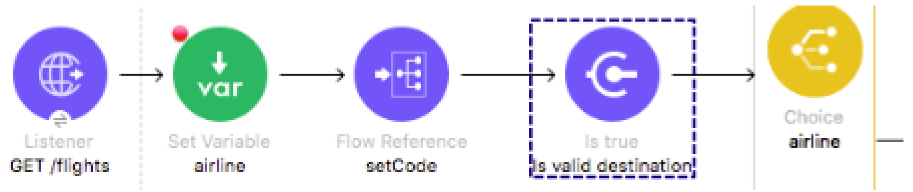
http://localhost:8081/flights?code=FOO

500 Server Error 62965.80 ms

```
Invalid destination FOO
```

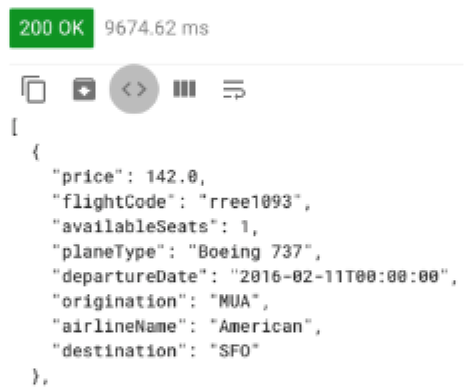
Note: You will catch this error and send a JSON response with a different status code in the next module.

21. Remove the code and make a request to http://localhost:8081/flights.
22. In the Mule Debugger, step through the application; you should not get any errors.



23. Resume through the rest of the application.

24. Return to Advanced REST Client; you should get a 200 response with all airline flights to SFO.

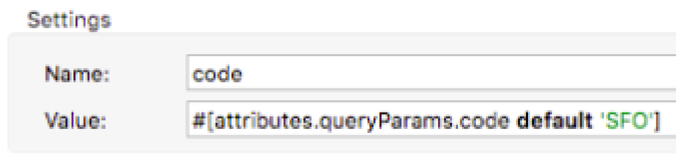


```
[
  {
    "price": 142.0,
    "flightCode": "rree1093",
    "availableSeats": 1,
    "planeType": "Boeing 737",
    "departureDate": "2016-02-11T00:00:00",
    "origination": "MUA",
    "airlineName": "American",
    "destination": "SFO"
  },
```

25. Return to Anypoint Studio and switch to the Mule Design perspective.

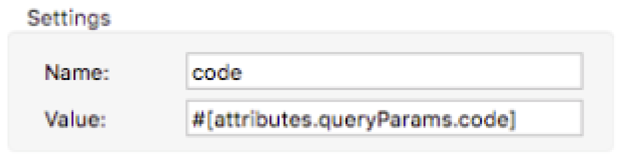
Remove the default destination

26. Locate the setCode subflow.
27. In the Set Variable properties view, review the default value.



Name: code
Value: #[attributes.queryParams.code default 'SFO']

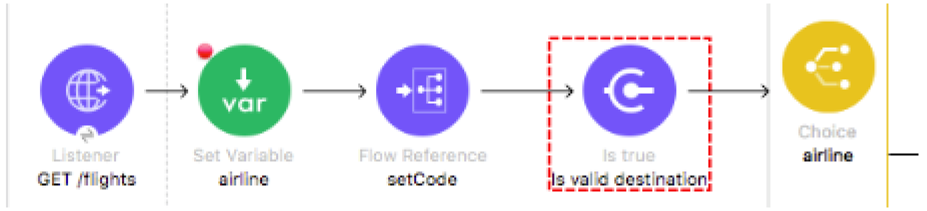
28. Remove the default value from the value.



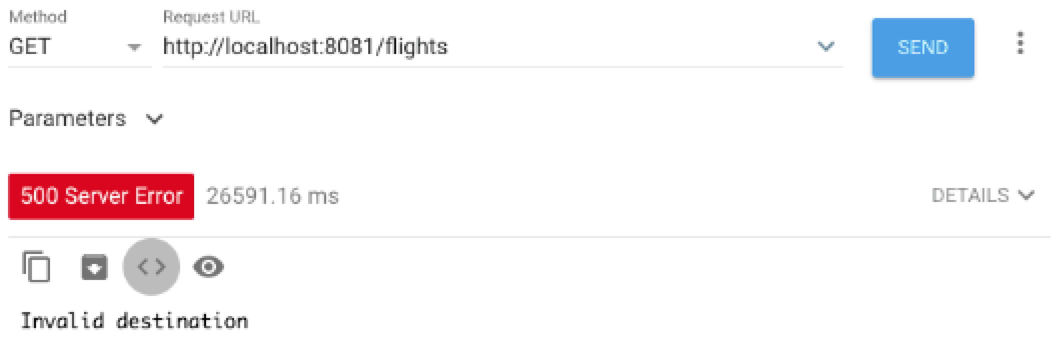
Name: code
Value: #[attributes.queryParams.code]

Debug the application

29. Save the file to redeploy the project.
30. In Advanced REST Client, make another request to http://localhost:8081/flights.
31. In the Mule Debugger, step to the validator; you should now get an error.



32. Resume through the rest of the application.
33. Return to Advanced REST Client; you should get a 500 Server Error with your Invalid destination message.



```
Invalid destination
```

Note: You will catch this error and send a JSON response with a different status code in the next module.

34. Return to Anypoint Studio and stop the project.

Summary

Summary

- Use different routers and validators to control event flow
- Use the **Choice** router to send a message to one route based on conditional logic
- Use the **Scatter-Gather** router to send a message concurrently to multiple routes
  - A collection of all results is returned
  - Use DataWeave to flatten the collection
- Use the **Validation** module to specify whether an event can proceed in a flow

DIY Exercise 9-1: Use validators to validate response schemas

Time estimate: 2 hours

Objectives

In this exercise, you control message flow using validators. You will:

- Validate REST response schemas.
- Validate SOAP response schemas.
- Retrieve data concurrently.
- Filter retrieved data.

Scenario

You are tasked with adding some new functionality to an existing project that involves retrieving data from different sources concurrently. The new requirement is to validate the responses returned from each individual data source and filter the final payload if the payload does meet certain criteria.

Import the starting project

Import /files/module09/route-mod09-starter.jar (in the MUFundamentals_DIYexercises.zip that you can download from the Course Resources) into Anypoint Studio.

Validate the Accounts REST response schema

In the implementation.xml file's accounts flow, add the JSON module and use the Validate schema operation to validate the REST response returned to the HTTP Request. Use the schema src/main/resources/schema/accountsSchema.json to validate against the REST request.

Validate the Flights SOAP response schema

In the implementation.xml file's flights flow, add the XML module and use the Validate schema operation to validate the SOAP response from the web service. Use the schema src/main/resources/schema/flightsSchema.xsd to validate against the SOAP request.

Test the validators

Test the validators by making requests to the following URLs:
http://locahost:8081/accounts
http://localhost:8081/flights

Change the validation rules so the response fails validation

Edit the accountsSchema.json file and the flightsSchema.xsd file to introduce a validation rule that is not met by the accounts and flights calls. For example, modify the XSD Schema file by changing the key name to something different than what is actually returned by the Delta SOAP service response:

```
<xs:sequence>

   <xs:element name="airlineName-BAD" form="unqualified" type="xs:NCName"/>
```

For the JSON schema file, change a key name in the required array value:

```
      "required": ["salesID-BAD","id","firstName", …]
```

After you verify the validation fails for accounts and flights, change the schemas back to the correct schemas.

Concurrently process both requests

In the main.xml file, create a new flow listening on the HTTP endpoint /flights_accounts to concurrently retrieve data from the accounts flow and the flights flow in implementation.xml.

Filter the retrieved data

Add a Choice router to filter out any empty data returned from either the flights flow or the accounts flow. There are three possible empty data scenarios:

- flights is empty
- accounts is empty
- flights and accounts are both empty

Return a message to the client that states that the payload returned from flights, accounts, or both are empty.

Hint: If there are no values returned from the SOAP service, the key listAllFlightsResponse will contain the value null.

Combine data

In the same Choice router, add another branch to return the flights and accounts data as a new JSON data structure to the client if flights and accounts are not empty.

Here is what the new JSON data structure should look like:

```
{
  accounts: {
      attributes: { ... },
      payload: { ... }
},
  flights: {
      attributes: { ... },
      payload: { ... }
}
```

Verify your solution

Load the solution /files/module09/route-mod09-solution.jar (in the MUFundamentals_DIYexercises.zip that you can download from the Course Resources) and compare your solution.