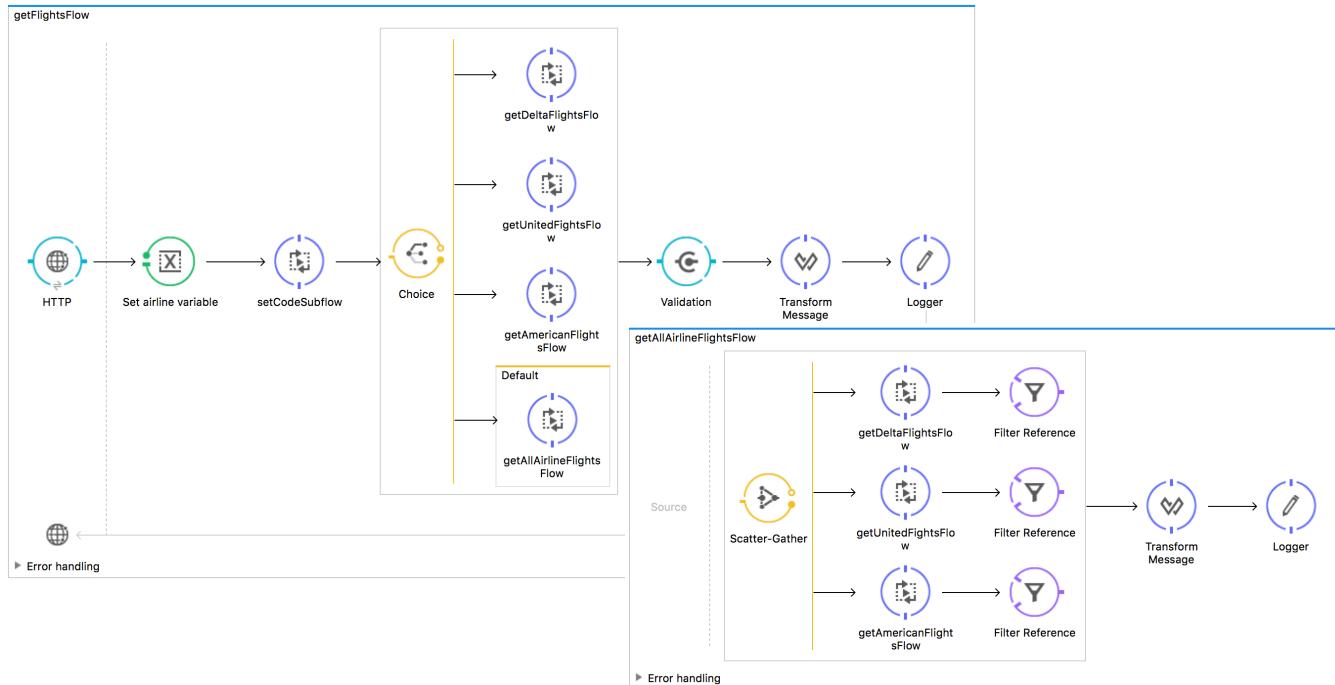


Module 9: Controlling Message Flow



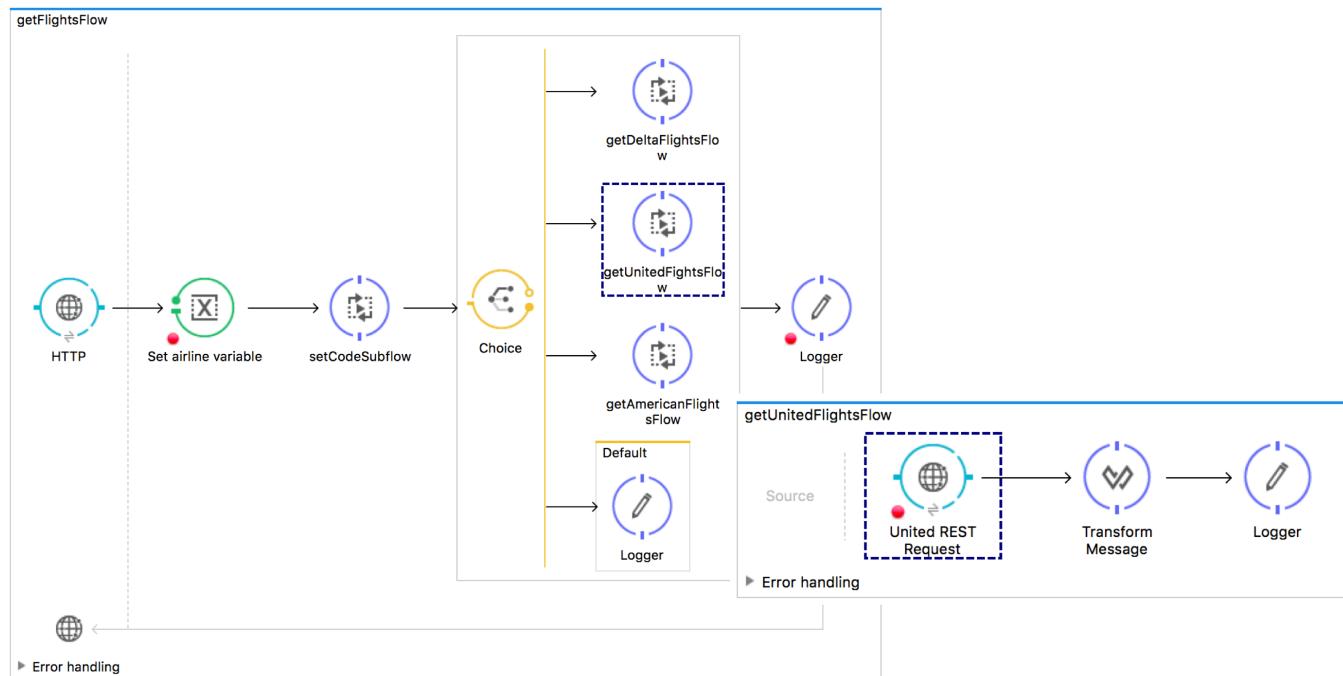
Objectives:

- Route messages based on conditions.
- Multicast messages.
- Filter messages.
- Validate messages.

Walkthrough 9-1: Route messages based on conditions

In this walkthrough, you modify gtAllFlightsFlow to route messages to either the United, Delta, or American flows based on the value of an airline query parameter. You will:

- Use a Choice router.
- Set the router paths.



Look at possible airline values specified in the API

1. Return to the apdev-flights-ws project in Anypoint Studio.
2. Go to the API Sync view and open `api.raml`.
3. Locate the airline query parameter and its possible values.
4. Run the project.

Test the application

5. In Postman, make a request to `http://localhost:8081/flights?code=CLE`; you should see only Delta flights to CLE.
6. Add a query parameter called `airline` and set it equal to `united`.

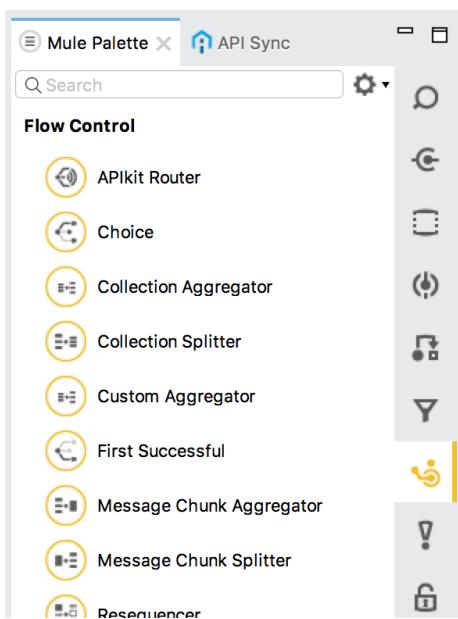
7. Send the request again; you should still only see the Delta flights.

The screenshot shows a REST client interface. At the top, there's a header bar with 'GET' dropdown, URL 'localhost:8081/flights?code=CLE&airline=united', 'Params' button, 'Send' button, and 'Save' button. Below the header are tabs: 'Body' (selected), 'Cookies', 'Headers (3)', and 'Tests'. On the right, it says 'Status: 200 OK' and 'Time: 185 ms'. Under the 'Body' tab, there are buttons for 'Pretty', 'Raw', 'Preview', 'JSON' (selected), and a copy icon. The JSON response is displayed in a code editor-like area:

```
12 {
13   "airline": "Delta",
14   "price": "308.0",
15   "departureDate": "2015/07/11",
16   "plane": "Boeing 777"
```

Browse the flow control elements in the Mule Palette

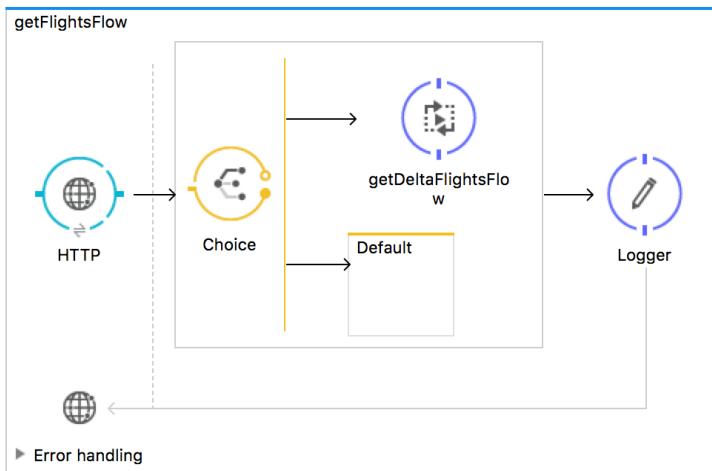
8. Return to Anypoint Studio.
9. In the Mule Palette, select the Flow Control tab.
10. View the available flow control processors.



Add a Choice router

11. Drag a Choice flow control element from the Mule Palette and drop it in getFlightsFlow before getDeltaFlightsFlow.

12. Drag the getDeltaFlightsFlow flow reference into the router.

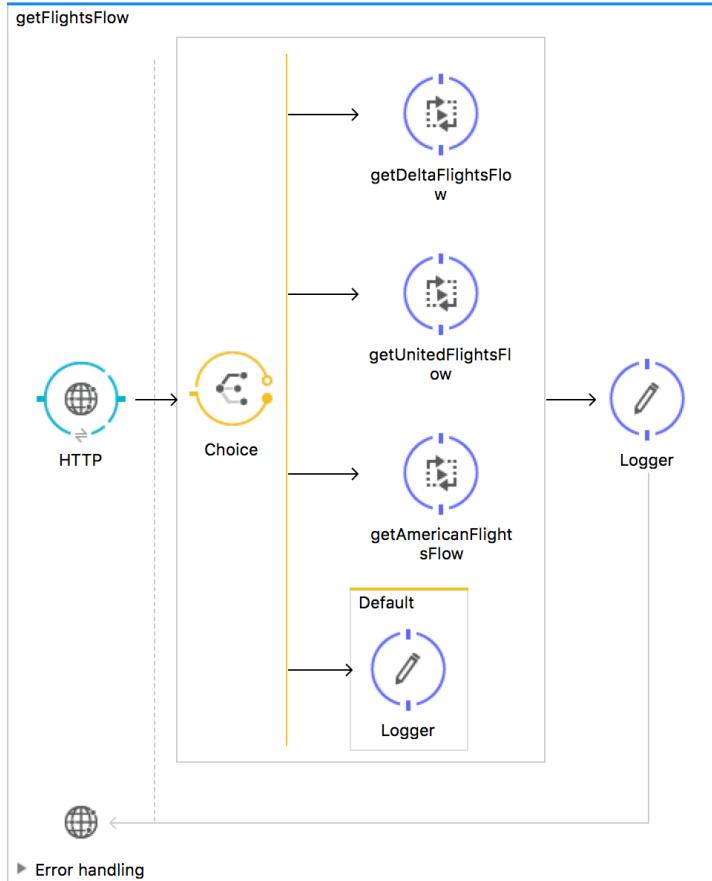


13. Add two additional Flow Reference components to the Choice router.

14. In the Properties view for the first flow reference, set the flow name to getUnitedFlightsFlow.

15. In the Properties view for the second flow reference, set the flow name to getAmericanFlightsFlow.

16. Drag a Logger component from the Mule Palette and drop it in the default branch.



Store the airline query parameter in a flow variable

17. Add a Variable transformer before the Choice router.
18. Change its display name to Set airline variable.
19. Set the flow variable name to airline and set it equal to a query parameter called airline.

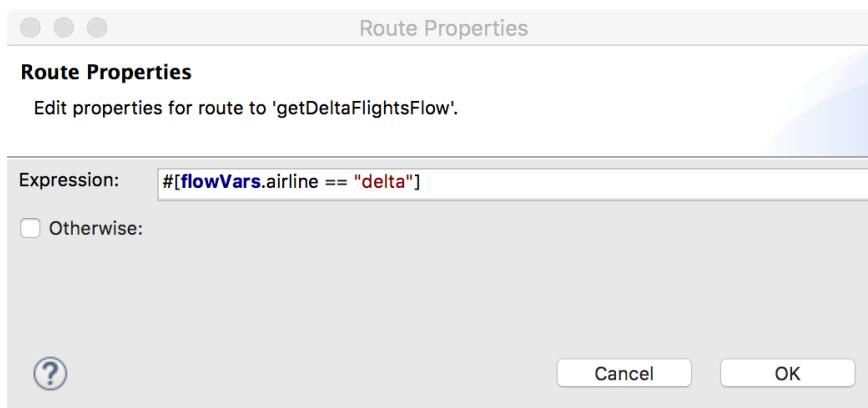
```
##[message.inboundProperties.'http.query.params'.airline]
```

The screenshot shows the Mule Studio interface. At the top, there's a message flow diagram with components: HTTP, Set airline variable (highlighted with a green border), Choice, getUnitedFlightsFlow, and a logger. Below the diagram are tabs for 'Message Flow', 'Global Elements', and 'Configuration XML'. A secondary window titled 'Set airline variable' is open, showing route properties. It has tabs for 'General' (selected), 'Notes', and 'Metadata'. Under 'General', it says 'There are no errors.' and shows fields for 'Name' (set to 'airline') and 'Value' (set to '##[message.inboundProperties.'http.query.params'.airline]').

Configure the Choice router

20. In the Choice properties view, double-click the getDeltaFlightsFlow route.
21. In the Route Properties dialog box, set the expression to true if the airline flow variable is equal to delta and click OK.

```
##[flowVars.airline == "delta"]
```



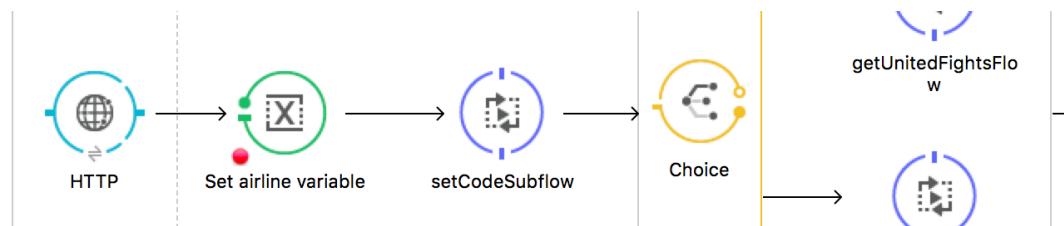
22. Set a similar expression for the United route, routing to it when flowVars.airline is equal to united.

23. Set a similar expression for the American route, routing to it when flowVars.airline is equal to american.

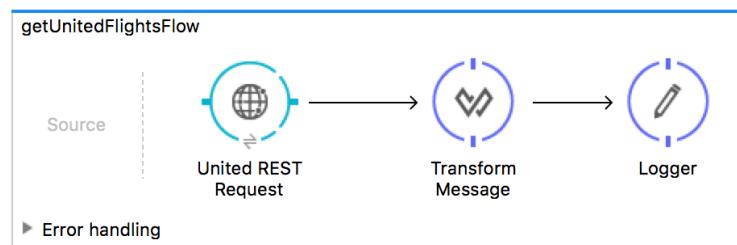
The screenshot shows the 'Choice' properties configuration in Anypoint Studio. The 'When' section contains three conditions: '#[flowVars.airline == "delta"]', '#[flowVars.airline == "united"]', and '#[flowVars.airline == "american"]'). The 'Route Message to' section lists four options: 'getDeltaFlightsFlow', 'Logger', 'getUnitedFlightsFlow', and 'getAmericanFlightsFlow'. The 'getAmericanFlightsFlow' option is highlighted with a grey background.

Route all requests through the router

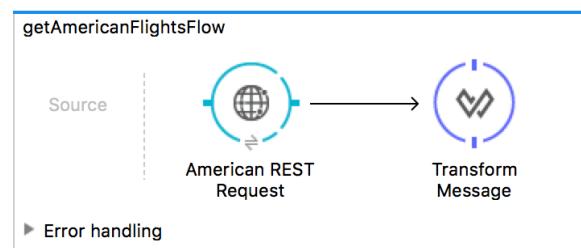
24. From getUnitedFlightsFlow, drag the setCodeSubflow flow reference into getFlightsFlow before the choice router.



25. In getUnitedFlightsFlow, delete the HTTP Listener endpoint.



26. In getAmericanFlightsFlow, delete the HTTP Listener endpoint and the setCodeSubflow flow reference.

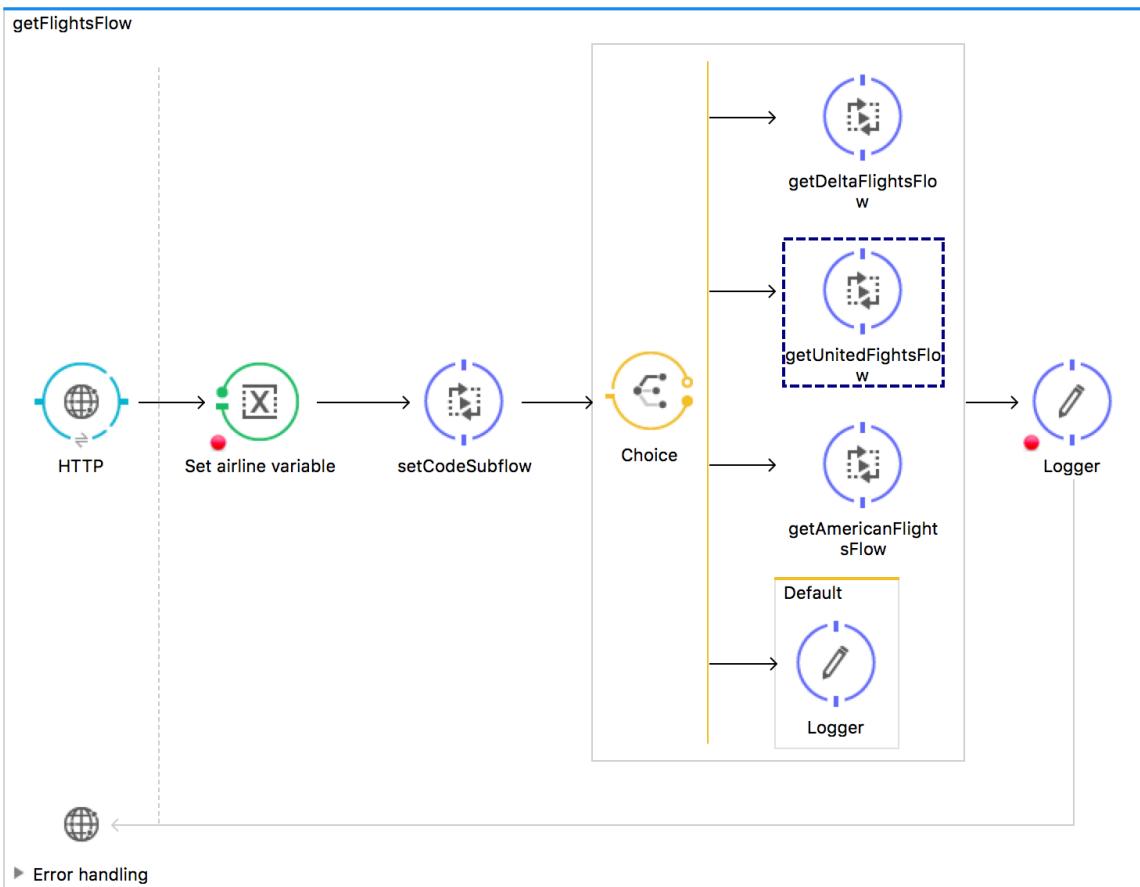


27. In getDeltaFlightsFlow, delete the HTTP Listener endpoint and the setCodeSubflow flow reference.

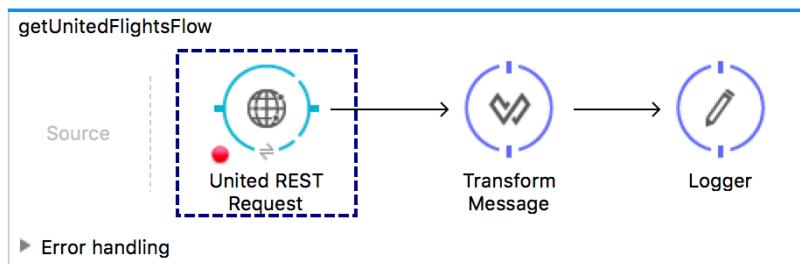


Test the application

28. In getFlightsFlow, make sure there is a breakpoint on one of the processors before the Choice router.
29. Debug the project.
30. In Postman, make the same request to <http://localhost:8081/flights?code=CLE&airline=united>.
31. In the Mule Debugger, step through the application; you should see the Choice router pass the message to the United branch.



32. Step through the rest of the application; you should see the message passed to getUnitedFlightsFlow and then back to getFlightsFlow.



33. Return to Postman; you should see only United flights to CLE returned.

A screenshot of the Postman application interface. The top bar shows a 'GET' method, the URL 'localhost:8081/flights?code=CLE&airline=united', and a 'Send' button. Below the URL, tabs for 'Body', 'Cookies', 'Headers (3)', and 'Tests' are visible, with 'Body' being the active tab. The 'Body' section shows a JSON response with line numbers on the left. The response content is:

```
2 {  
3   "airline": "United",  
4   "price": 845,  
5   "departureDate": "2015/07/11",  
6   "plane": "Boeing 727",  
7   "origination": "MUA",  
8   "code": "ER9fje",  
9   "emptySeats": 32,  
10  "destination": "CLE"  
11 } ,
```

The status bar at the bottom indicates 'Status: 200 OK' and 'Time: 38960 ms'.

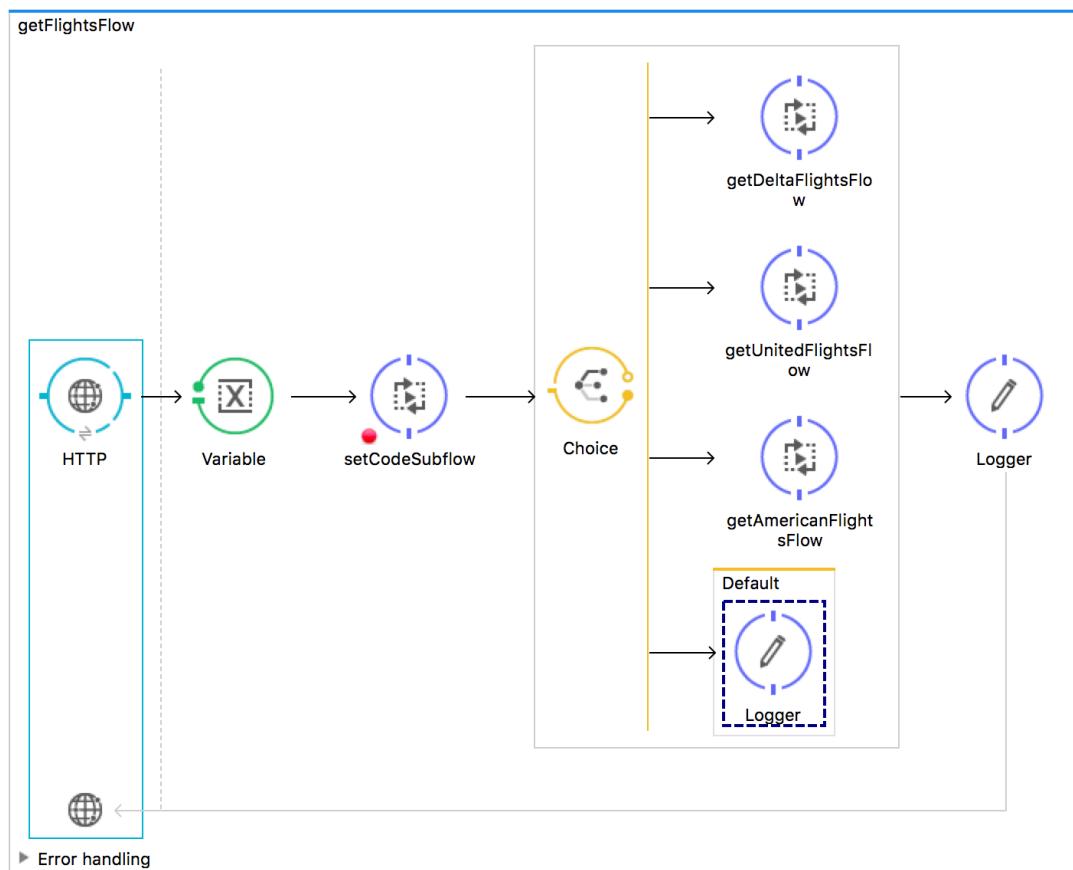
34. Change the airline to delta and the code to LAX.

35. Send the request.

36. Step through the application; the message should be routed to the Delta branch.

37. Return to Postman; you should see only Delta flights to LAX are returned.

38. In Postman, remove both parameters and make a request to <http://localhost:8081/flights>.
39. In the Mule Debugger, step through the application; you should see the Choice router pass the message to the default branch.



40. Click the Resume button.
41. Return to Postman; no flights are returned.

The screenshot shows a Postman request configuration for a GET request to <http://localhost:8081/flights>. The response status is 200 OK with a time of 60431 ms. The body response is empty, showing only the number 1.

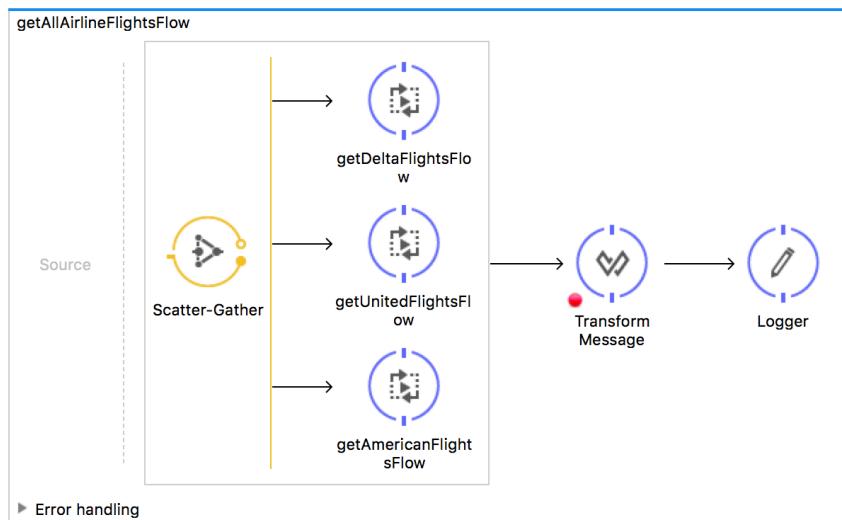
Note: If the next walkthrough, you will change this behavior so that if no airline is specified, flights for all airlines will be returned.

42. Return to Anypoint Studio and stop the project.

Walkthrough 9-2: Multicast a message

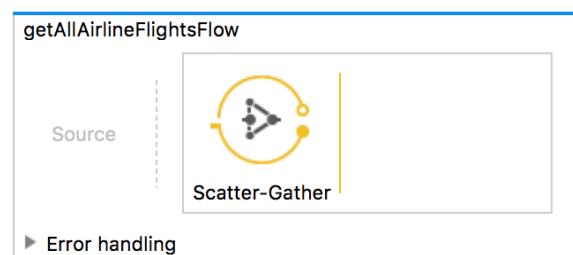
In this walkthrough, you create a flow that calls each of the three airline services and combines the results. You will:

- Use a Scatter-Gather router to concurrently call all three flight services.
- Use DataWeave to flatten multiple collections into one collection.
- Use DataWeave to sort the flights by price and return them as JSON.
- (Optional) Modify the airline flows to use DataWeave to each return a Java collection of Flight objects.



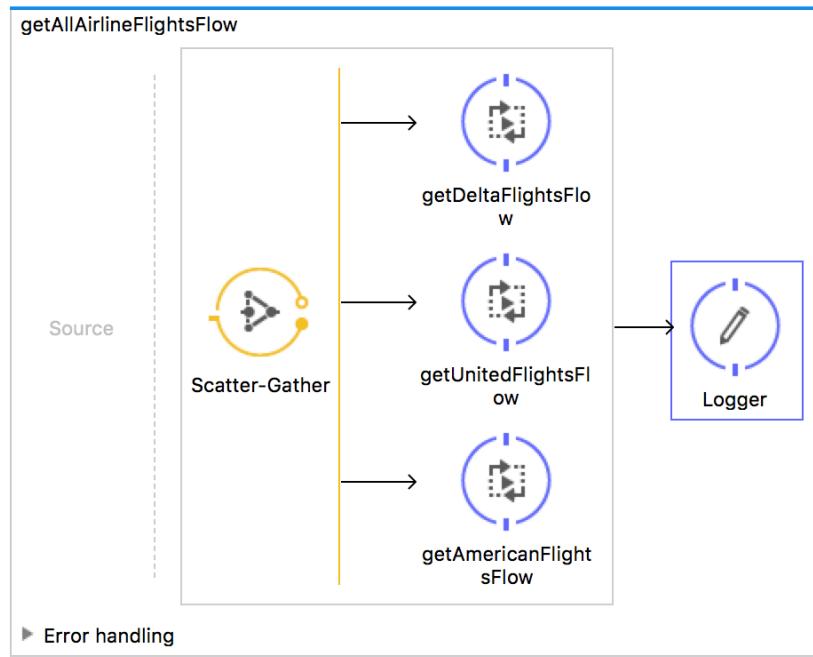
Create a flow to use a router to call all three airline services

1. Return to implementation.xml.
2. Drag a Scatter-Gather flow control element from the Mule Palette and drop it at the bottom of the canvas.
3. Change the name of the flow to getAllAirlineFlightsFlow.



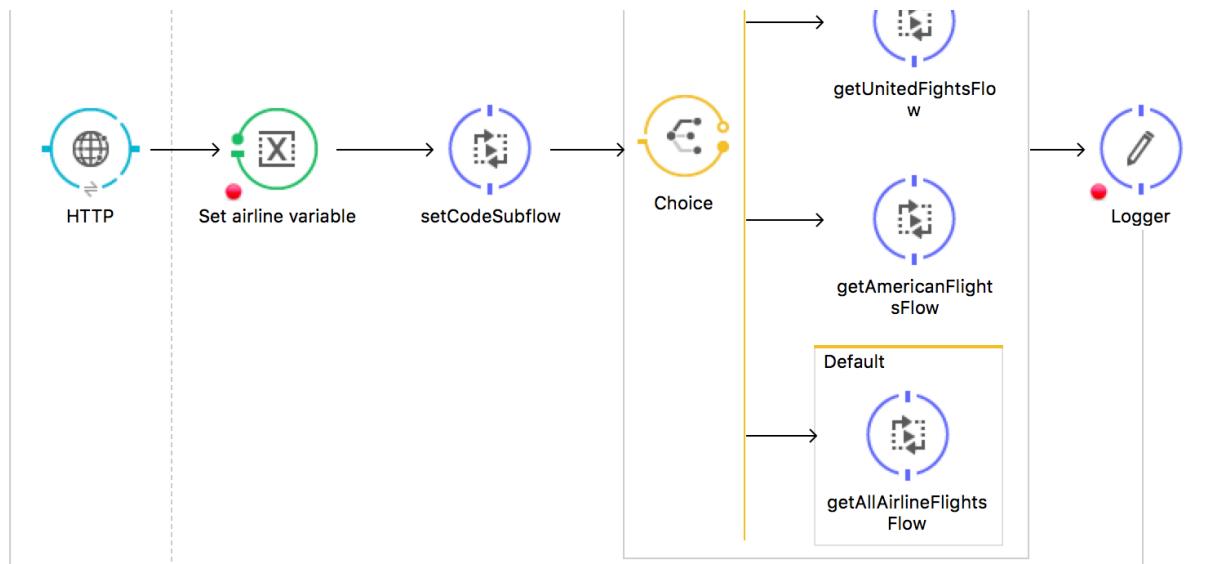
4. Drag three Flow Reference components into the Scatter-Gather router.
5. Using the Properties view, set the flow name of the first Flow Reference to getDeltaFlightsFlow.

6. Set the flow name of the second Flow Reference to getUnitedFlightsFlow.
7. Set the flow name of the third Flow Reference to getAmericanFlightsFlow.
8. Add a Logger after the router.



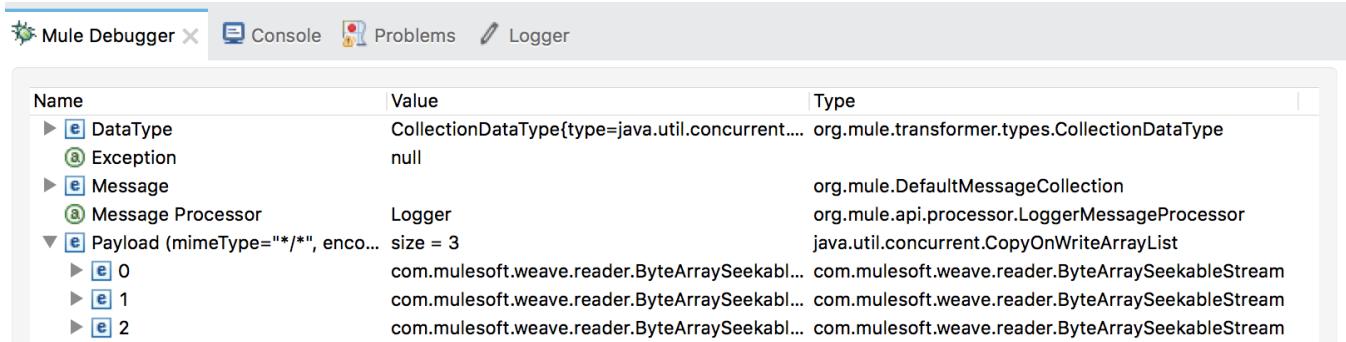
Call this flow if no airline is specified

9. Return to getFlightsFlow at the top of the canvas.
10. Delete the Logger in the default branch.
11. Add a Flow Reference to the default branch and set its flow name to getAllAirlineFlightsFlow.



Test the application

12. Debug the project.
13. In Postman, make the same request to <http://localhost/flights>.
14. In the Mule Debugger, step through the application to the default branch of the choice router, then into the Scatter-Gather, and then into each of the airline flows.
15. Stop at the Logger back in getFlightsFlow and look at the payload.



The screenshot shows the Mule Debugger interface with the 'Payload' table open. The table has three columns: Name, Value, and Type. The 'Payload' row is expanded, showing three sub-rows labeled 0, 1, and 2, each representing a com.mulesoft.weave.reader.ByteString object.

Name	Value	Type
(DataType)	CollectionDataType{type=java.util.concurrent.... org.mule.transformer.types.CollectionDataType}	
(Exception)	null	
(Message)		org.mule.DefaultMessageCollection
(Message Processor)	Logger	org.mule.api.processor.LoggerMessageProcessor
Payload (mimeType="*/*", enco...)	size = 3	java.util.concurrent.CopyOnWriteArrayList
(0)	com.mulesoft.weave.reader.ByteString	com.mulesoft.weave.reader.ByteString
(1)	com.mulesoft.weave.reader.ByteString	com.mulesoft.weave.reader.ByteString
(2)	com.mulesoft.weave.reader.ByteString	com.mulesoft.weave.reader.ByteString

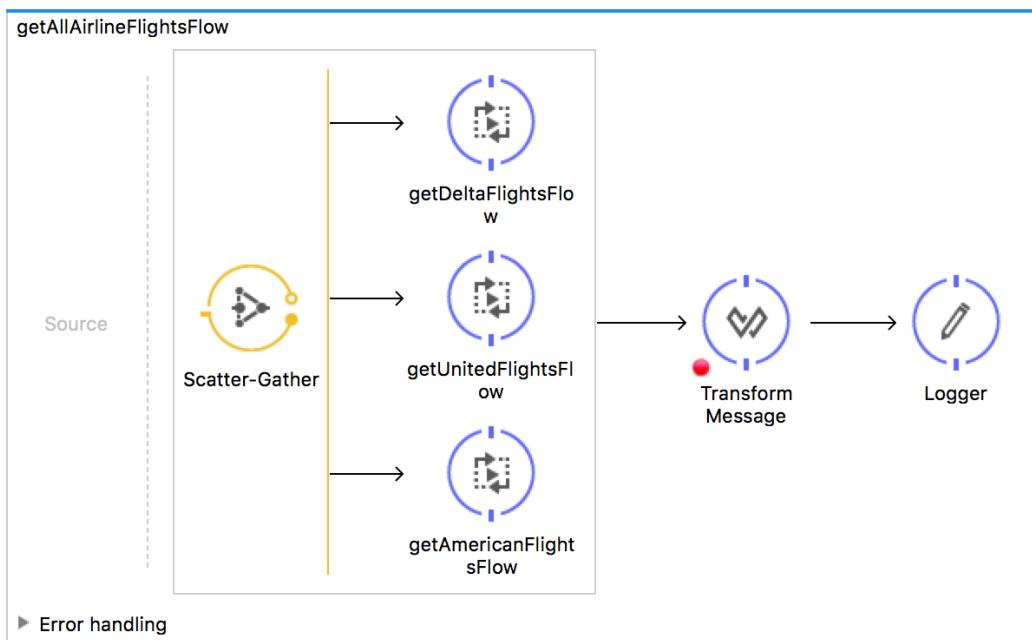
Note: You get three different DataWeave stream objects. You want to combine the three collections and then sort them by price and return them as JSON. Because the airline flight flows were written first and already return JSON, you can just flatten the collections into one and then order by price. More typically, however, you would have each of the airline flows return data of a common canonical format – in this case, a collection of Flight Java objects – and then flatten, order, and transform that data to the JSON specified by the API.

16. Click the Resume button.
17. Stop the project.

Flatten the combined results

18. Return to getAllAirlineFlightsFlow.

19. Add a Transform Message component before the Logger.



20. In the Properties view, set the DataWeave expression to flatten the payload.

`flatten payload`

```
1 %dw 1.0
2 %output application/java
3 ---
4 flatten payload
```

Note: You will learn about DataWeave operators in the later module Writing DataWeave expressions.

Test the application

21. Save the file to redeploy the application.

22. In Postman, make the same request to <http://localhost:8081/flights>.

23. In the Mule Debugger, step through the application to the Logger after the Choice router; you should see the payload is now one ArrayList of HashMaps.

```

▼ [E] Payload (mimeType="appli... size = 11          java.util.ArrayList
  ► [E] 0           {airline=Delta, flightCode=A1B2...  java.util.LinkedHashMap
  ► [E] 1           {airline=Delta, flightCode=A1BT...  java.util.LinkedHashMap
  ► [E] 10          {airline=American, flightCode=rr...  java.util.LinkedHashMap
  ► [E] 2           {airline=Delta, flightCode=A142...  java.util.LinkedHashMap
  ► [E] 3           {airline=United, flightCode=ER3...  java.util.LinkedHashMap
  ► [E] 4           {airline=United, flightCode=ER3...  java.util.LinkedHashMap
  ► [E] 5           {airline=American, flightCode=rr...  java.util.LinkedHashMap
  ► [E] 6           {airline=American, flightCode=rr...  java.util.LinkedHashMap
  ► [E] 7           {airline=American, flightCode=ee...  java.util.LinkedHashMap
  ► [E] 8           {airline=American, flightCode=ff...  java.util.LinkedHashMap
  ► [E] 9           {airline=American, flightCode=ee...  java.util.LinkedHashMap
    ► [E] 0           airline=American          java.util.LinkedHashMap$Entry
    ► [E] 1           flightCode=eefd3000       java.util.LinkedHashMap$Entry
    ► [E] 2           fromAirportCode=MUA        java.util.LinkedHashMap$Entry
    ► [E] 3           toAirportCode=SFO         java.util.LinkedHashMap$Entry
    ► [E] 4           departureDate=2016-02-01T00...  java.util.LinkedHashMap$Entry
    ► [E] 5           emptyvSeats=0            java.util.LinkedHashMap$Entry

```

24. Step through the rest of the application and stop the project.

25. In Postman, you should get a representation of Java objects returned.

The screenshot shows a Postman request to `localhost:8081/flights`. The response body is a JSON object representing an ArrayList of flight details:

```

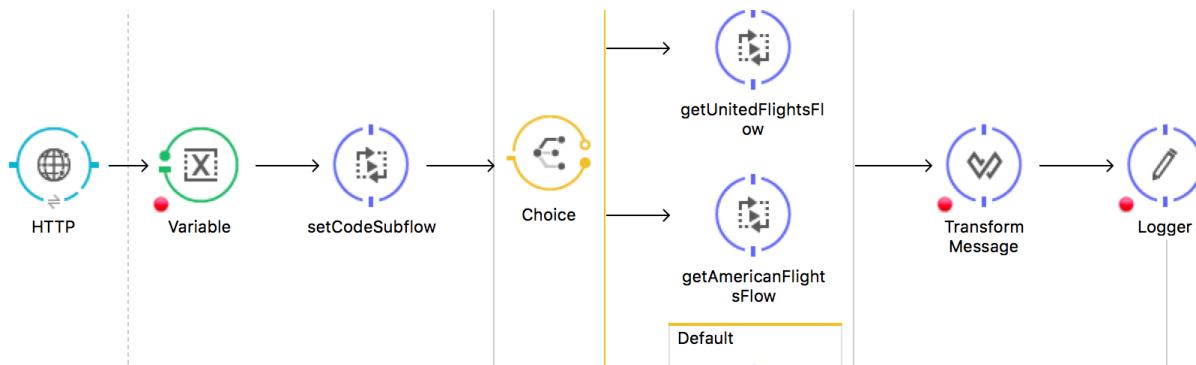
[{"id": 1, "airline": "Delta", "flightCode": "A1B2C3", "fromAirportCode": "MUA", "toAirportCode": "SFO", "departureDate": "2015/03/20", "emptySeats": 40, "price": 400.0, "planeType": "Boing 737x sq ~"}, {"id": 2, "airline": "Delta", "flightCode": "A1BT", "fromAirportCode": "MUA", "toAirportCode": "SFO", "departureDate": "2015/03/20", "emptySeats": 40, "price": 400.0, "planeType": "Boing 737x sq ~"}, {"id": 3, "airline": "American", "flightCode": "rr", "fromAirportCode": "MUA", "toAirportCode": "SFO", "departureDate": "2015/03/20", "emptySeats": 40, "price": 400.0, "planeType": "Boing 737x sq ~"}, {"id": 4, "airline": "Delta", "flightCode": "A142", "fromAirportCode": "MUA", "toAirportCode": "SFO", "departureDate": "2015/03/20", "emptySeats": 40, "price": 400.0, "planeType": "Boing 737x sq ~"}, {"id": 5, "airline": "United", "flightCode": "ER3", "fromAirportCode": "MUA", "toAirportCode": "SFO", "departureDate": "2015/03/20", "emptySeats": 40, "price": 400.0, "planeType": "Boing 737x sq ~"}, {"id": 6, "airline": "United", "flightCode": "ER3", "fromAirportCode": "MUA", "toAirportCode": "SFO", "departureDate": "2015/03/20", "emptySeats": 40, "price": 400.0, "planeType": "Boing 737x sq ~"}, {"id": 7, "airline": "American", "flightCode": "rr", "fromAirportCode": "MUA", "toAirportCode": "SFO", "departureDate": "2015/03/20", "emptySeats": 40, "price": 400.0, "planeType": "Boing 737x sq ~"}, {"id": 8, "airline": "American", "flightCode": "ee", "fromAirportCode": "MUA", "toAirportCode": "SFO", "departureDate": "2015/03/20", "emptySeats": 40, "price": 400.0, "planeType": "Boing 737x sq ~"}, {"id": 9, "airline": "American", "flightCode": "ff", "fromAirportCode": "MUA", "toAirportCode": "SFO", "departureDate": "2015/03/20", "emptySeats": 40, "price": 400.0, "planeType": "Boing 737x sq ~"}, {"id": 10, "airline": "American", "flightCode": "ee", "fromAirportCode": "MUA", "toAirportCode": "SFO", "departureDate": "2015/03/20", "emptySeats": 40, "price": 400.0, "planeType": "Boing 737x sq ~"}]

```

Return the data as JSON and sort by price

26. Return to `getFlightsFlow`.

27. Add a Transform Message component after the Choice router.



28. In the Transform Message properties view, change the output to application/json.

29. Change the transformation expression to order the payload by price.

```
Output Payload ▾ ⌂
```

```
1 %dw 1.0
2 %output application/json
3 ---
4 payload orderBy $.price
```

Note: You will learn about DataWeave operators in the later module Writing DataWeave expressions.

Test the application

30. Run the project.

31. In Postman, send the same request to <http://localhost:8081/flights>; you should get all the flights to SFO returned and they should be sorted by price.

The screenshot shows the Postman interface. At the top, there is a header bar with 'GET' selected, the URL 'localhost:8081/flights', and buttons for 'Params', 'Send', and 'Save'. Below the header, there are tabs for 'Body', 'Cookies', 'Headers (3)', and 'Tests'. The 'Body' tab is active, showing a JSON response. The response is a JSON array with two elements, each representing a flight record. The flights are sorted by price, with the first flight having a price of 142 and the second having a price of 294.0. The JSON is displayed in a 'Pretty' format with line numbers on the left.

```
1 [ [ { "airline": "American", "flightCode": "A321093", "fromAirportCode": "MUA", "toAirportCode": "SFO", "departureDate": "2016-02-11T00:00:00", "emptySeats": 1, "totalSeats": 150, "price": 142, "planeType": "Boeing 737" }, { "airline": "Delta", "flightCode": "A14244", "fromAirportCode": "MUA", "toAirportCode": "SFO", "departureDate": "2015/02/12", "emptySeats": "10", "price": "294.0", "planeType": "Boeing 787" } ] ]
```

32. Add an airline parameter equal to united and send the request:

<http://localhost:8081/flights?airline=united>; you should get united flights to SFO sorted by price.

33. Add a code parameter equal to PDF and send the request:

<http://localhost:8081/flights?airline=united&code=PDF>; you should get one united flight to PDF.

The screenshot shows a REST client interface with the following details:

- Method: GET
- URL: localhost:8081/flights?airline=united&code=PDF
- Headers: (3)
- Status: 200 Time: 249 ms
- Body tab selected
- Pretty JSON view:

```
1 [  
2 {  
3   "airline": "United",  
4   "flightCode": "ER95jf",  
5   "fromAirportCode": "MUA",  
6   "toAirportCode": "PDF",  
7   "departureDate": "2015/02/12",  
8   "emptySeats": 23,  
9   "price": 234,  
10  "planeType": "Boeing 787"  
11 }]  
12 ]
```
- Buttons: Params, Send, Save

34. Change the airline to delta and send the request:

<http://localhost:8081/flights?airline=delta&code=PDF>; you should see the message that there are no flights to PDF – which is correct.

The screenshot shows a REST client interface with the following details:

- Method: GET
- URL: localhost:8081/flights?airline=delta&code=PDF
- Headers: (3)
- Status: 400 Time: 489 ms
- Body tab selected
- HTML view:

```
i 1 NO FLIGHTS to PDF  
2 com.mulesoft.weave.mule.exception.WeaveExecutionException: Exception while executing:  
3 payload orderBy $.price
```
- Buttons: Params, Send, Save

35. Remove the airline parameter and send the request: <http://localhost:8081/flights?code=PDF>;

you should get the one United flight to PDF, but instead you get the message that there are no flights.

The screenshot shows a REST client interface with the following details:

- Method: GET
- URL: localhost:8081/flights?code=PDF
- Headers: (3)
- Status: 400 Time: 1409 ms
- Body tab selected
- HTML view:

```
i 1 NO FLIGHTS to PDF  
2 com.mulesoft.weave.mule.exception.WeaveExecutionException: Exception while executing:  
3 payload orderBy $.price
```
- Buttons: Params, Send, Save

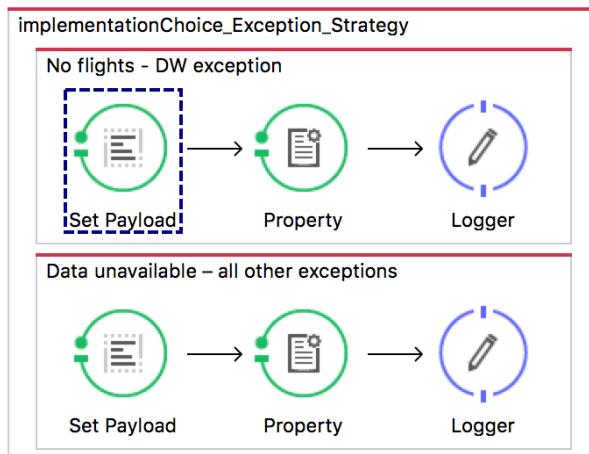
36. Return to Anypoint Studio and stop the project.

Debug the application

37. Debug the project.

38. In Postman, send the same request to <http://localhost:8081/flights?code=PDF>.

39. In the Mule Debugger, step through the application until an exception is thrown – and handled by the default global exception handler.



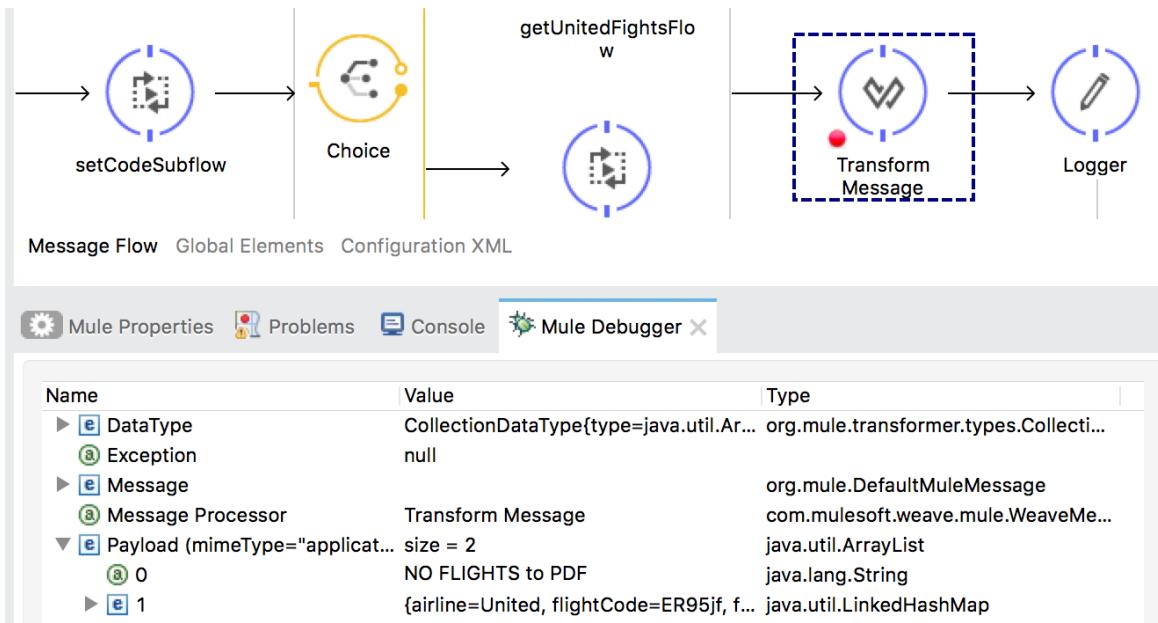
40. Continue to step through the application until you reach the Transform Message component in getAllAirlineFlightsFlow; you should see different types of objects returned.

The screenshot shows the Mule Debugger interface with the following components and details:

- Message Flow:** The flow consists of a 'Source' (dashed line), a 'Scatter-Gather' component (yellow circle with a network icon), a 'getUnitedFlightsFlow' component (blue circle with a gear icon), and a 'Transform Message' component (blue circle with a heart icon). A 'Logger' component (pen icon) is connected to the end of the flow.
- Global Elements:** A 'Configuration XML' tab is visible at the bottom.
- Mule Properties:** A table showing the current state of variables:

Name	Value	Type
▶ e DataType	CollectionDataType{type=java.u...}	org.mule.transformer.types.Coll...
ⓐ Exception	null	
▶ e Message		org.mule.DefaultMessageCollection
ⓐ Message Processor	Transform Message	com.mulesoft.weave.mule.Weav...
▼ e Payload (mimeType="/*/*")	size = 3 NO FLIGHTS to PDF com.mulesoft.weave.reader.Byte... com.mulesoft.weave.reader.Byte... com.mulesoft.weave.reader.Byte...	java.util.concurrent.CopyOnWrit...
ⓐ 0	NO FLIGHTS to PDF	java.lang.String
▶ e 1	com.mulesoft.weave.reader.Byte...	com.mulesoft.weave.reader.Byt...
▶ e 2	com.mulesoft.weave.reader.Byte...	com.mulesoft.weave.reader.Byt...
- Problems:** A tab showing any errors or warnings.
- Console:** A tab for viewing logs and console output.
- Mule Debugger:** The active tab, showing the current state of the application's data.

41. Step to the Transform Message component in getFlightsFlow; you should see the payload still contains the error message in addition to the valid flight results to PDF.



42. Step to the end of the application; you should not get the United results to PDF.

The screenshot shows a Postman request configuration and a response body.

Request:

- Method: GET
- URL: localhost:8081/flights?code=PDF
- Params: None
- Send button is highlighted.
- Save button is present.

Response:

- Status: 400
- Time: 203856 ms
- Body tab is selected.
- Pretty, Raw, Preview, and HTML buttons are available.
- Body content:

```
i 1 NO FLIGHTS to PDF
2 com.mulesoft.weave.mule.exception.WeaveExecutionException: Exception while executing:
3 payload orderBy $.price
```

Note: You could write a custom aggregation strategy for the Scatter-Gather router with Java to handle this situation, but instead you will use the simpler approach of filtering out the exception messages in the next walkthrough.

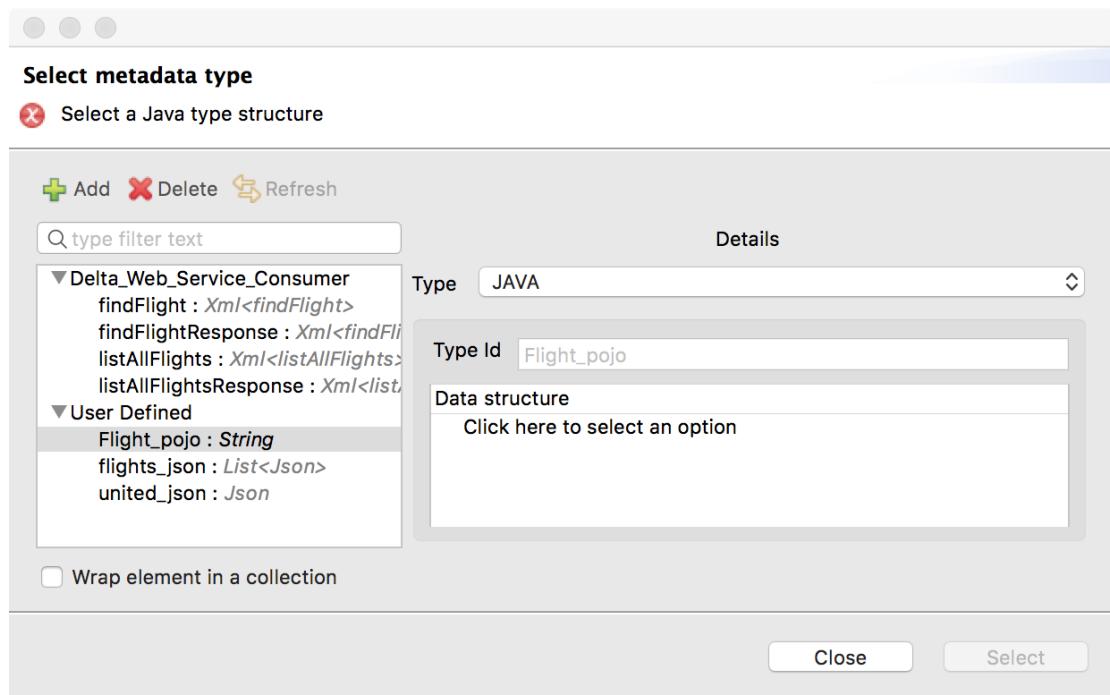
43. Return to Anypoint Studio and stop the project.

(Optional) Modify the airline flows to return Java Flight objects instead of JSON

Note: The rest of the walkthrough is optional. It contains steps to modify each of the airline flows to return data of a common canonical format – in this case, a collection of Flight Java objects.

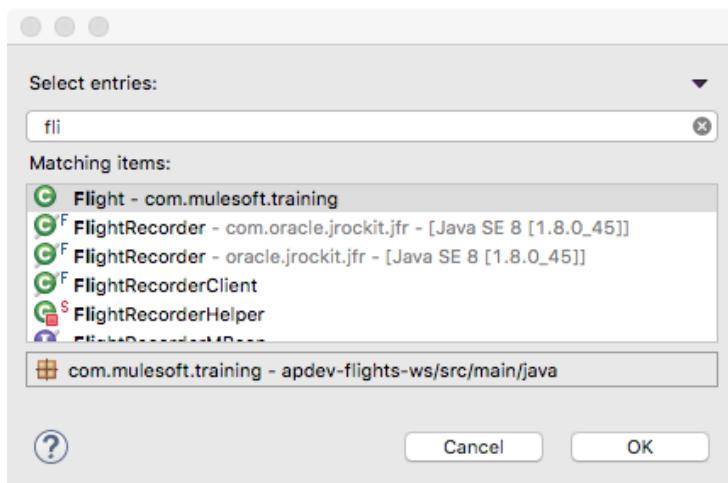
Change the United airline flows to return Java Flight objects instead of JSON

44. Return to getUnitedFlightsFlow in implementation.xml.
45. In the Transform Message properties view, right-click List <Json> in the output section and select Clear Metadata.
46. Click the Define metadata link.
47. In the Select metadata type dialog box, click the Add button.
48. In the Create new type dialog box, set the name to Flight_pojo and click Create type.
49. In the Select metadata type dialog box, change the type to JAVA.
50. In the Data structure section, click the Click here to select an option link.



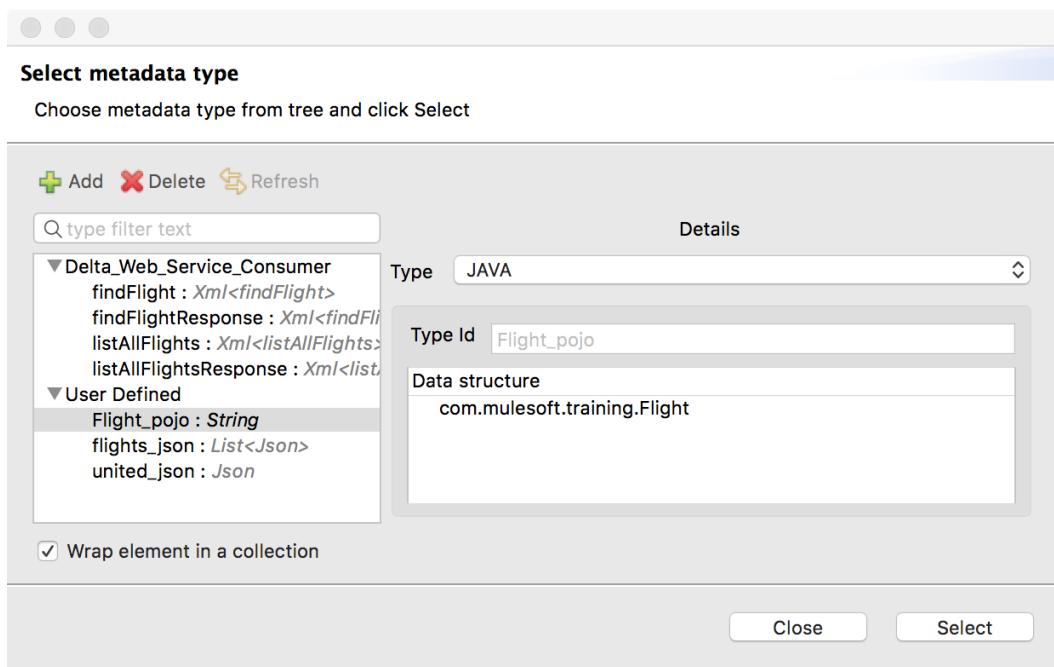
51. In the drop-down menu that appears, select Java object.

52. In the dialog box that opens, type fli and then in the list of classes that appears, select Flight – com.mulesoft.training.com.



53. Click OK.

54. In the Select metadata type dialog box, select Wrap element in a collection in the lower-left corner.



55. Click Select.

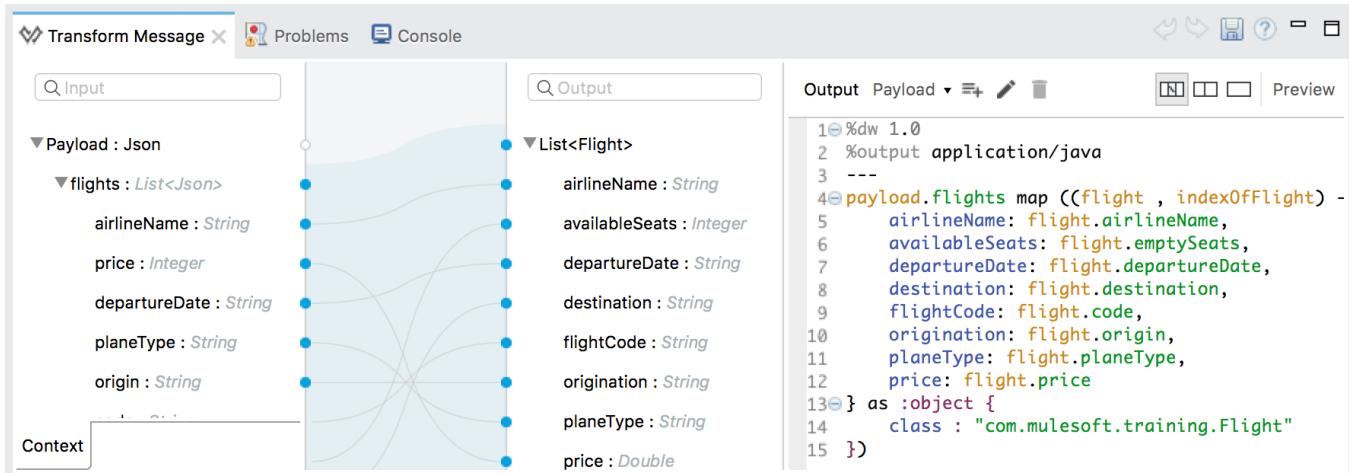
56. In the Transform Message properties view, replace the current transformation expression with an empty object.

```

Output Payload ▾ + 🖊️ 🗑️ | Preview
1 %dw 1.0
2 %output application/java
3 ---
4 {}

```

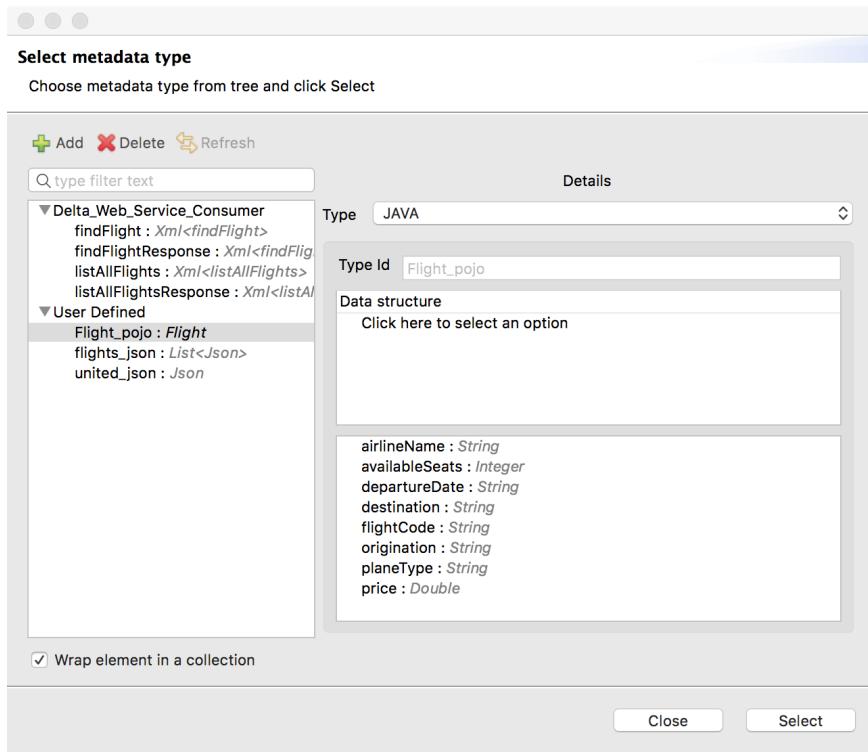
57. Use the graphical editor to map the fields appropriately.



Change the American flow to return Java Flight objects instead of JSON

58. Return to getAmericanFlightsFlow.
59. In the Transform Message properties view, right-click List <Json> in the output section and select Clear Metadata.
60. Click the Define metadata link.
61. In the Select metadata type dialog box, select Flight_pojo.

62. Select the Wrap element in a collection checkbox in the lower-left corner.



63. Click Select.

64. In the Transform Message properties view, replace the current transformation expression with an empty object.

65. Use the graphical editor to map the fields appropriately.

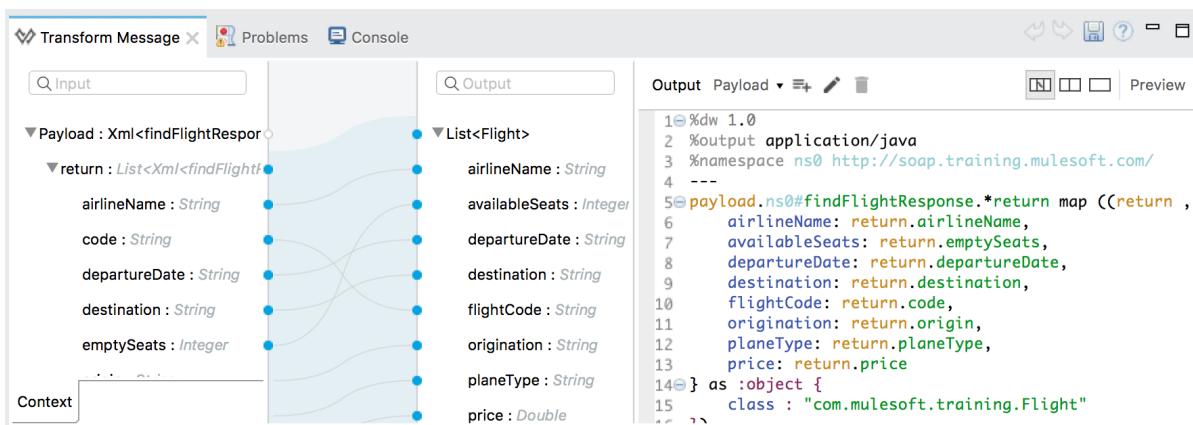
66. In the output section of the graphical editor, double-click airlineName; the field should be added to the DataWeave expression.

67. Set its value to "American".

```
1 %dw 1.0
2 %output application/java
3 ---
4 payload map ((payload01 , indexOfPayload01) -> {
5     airlineName: "American",
6     availableSeats: payload01.emptySeats,
7     departureDate: payload01.departureDate,
8     destination: payload01.destination,
9     flightCode: payload01.code,
10    origination: payload01.origin,
11    planeType: payload01.plane.type,
12    price: payload01.price
13 } as :object {
14     class : "com.mulesoft.training.Flight"
15 })
```

Change the Delta flow to return Java Flight objects instead of JSON

68. Return to getDeltaFlightsFlow.
69. In the Transform Message properties view for the component after the Delta SOAP Request, right-click List <Json> in the output section and select Clear Metadata.
70. Click the Define metadata link.
71. In the Select metadata type dialog box, select Flight_pojo.
72. Select the Wrap element in a collection checkbox in the lower-left corner.
73. Click Select.
74. In the Transform Message properties view, replace the current transformation expression with an empty object.
75. Use the graphical editor to map the fields appropriately.



Test the application

76. Debug the project.
77. In Postman, remove the parameters and send a request to <http://localhost:8081/flights>.
78. In the Mule Debugger, step through the application to the Transform Message component in getAllAirlineFilghtsFlow; the payload should be a collection of three ArrayLists of Flight objects.

Payload (mimeType="*/*", encoding="UT...")	size = 3	java.util.concurrent.CopyOnWriteArrayList
▶ [E] 0	[com.mulesoft.training.Flight@24222585, com....]	java.util.ArrayList
▶ [E] 1	[com.mulesoft.training.Flight@5979a9c0, com....]	java.util.ArrayList
▼ [E] 2	[com.mulesoft.training.Flight@736e92d2, com....]	java.util.ArrayList
▶ [E] 0	com.mulesoft.training.Flight@736e92d2	com.mulesoft.training.Flight
▶ [E] 1	com.mulesoft.training.Flight@45cf35b4	com.mulesoft.training.Flight
▶ [E] 2	com.mulesoft.training.Flight@37b9ada5	com.mulesoft.training.Flight
▶ [E] 3	com.mulesoft.training.Flight@4174ecb8	com.mulesoft.training.Flight
▼ [E] 4	com.mulesoft.training.Flight@468b0e46	com.mulesoft.training.Flight
@ airlineName	American	java.lang.String
@ availableSeats	100	java.lang.Integer
@ departureDate	2016-01-20T00:00:00	java.lang.String
@ destination	SFO	java.lang.String
@ flightCode	rree4567	java.lang.String
@ origination	MUA	java.lang.String
@ planeType	Boeing 737	java.lang.String
@ price	456.0	java.lang.Double

79. Step to the Logger; the payload should now be one ArrayList of Flight objects.

Payload (mimeType="... size = 11		java.util.ArrayList
► [e] 0	com.mulesoft.training.Flight...	com.mulesoft.training.Flight
► [e] 1	com.mulesoft.training.Flight...	com.mulesoft.training.Flight
► [e] 10	com.mulesoft.training.Flight...	com.mulesoft.training.Flight
► [e] 2	com.mulesoft.training.Flight...	com.mulesoft.training.Flight
► [e] 3	com.mulesoft.training.Flight...	com.mulesoft.training.Flight
► [e] 4	com.mulesoft.training.Flight...	com.mulesoft.training.Flight
► [e] 5	com.mulesoft.training.Flight...	com.mulesoft.training.Flight
► [e] 6	com.mulesoft.training.Flight...	com.mulesoft.training.Flight
► [e] 7	com.mulesoft.training.Flight...	com.mulesoft.training.Flight
► [e] 8	com.mulesoft.training.Flight...	com.mulesoft.training.Flight
► [e] 9	com.mulesoft.training.Flight...	com.mulesoft.training.Flight
@ airlineName	American	java.lang.String
@ availableSeats	0	java.lang.Integer
@ departureDate	2016-02-01T00:00:00	java.lang.String

80. Step through to the end of the application.

Test the application

Note: You completed the rest of the walkthrough steps already before modifying the airline flows to return collections of Flight objects. You are repeating them as a lead in to the next walkthrough.

81. Run the project.

82. In Postman, send the same request to <http://localhost:8081/flights>; you should get all the flights to SFO returned and they should be sorted by price.

The screenshot shows the Postman interface with a GET request to 'localhost:8081/flights'. The response status is 200 and the time taken is 1810 ms. The response body is a JSON array containing two flight objects:

```
1 | [
2 |   {
3 |     "flightCode": "rree1093",
4 |     "availableSeats": 1,
5 |     "destination": "SFO",
6 |     "planeType": "Boeing 737",
7 |     "origination": "MUA",
8 |     "price": 142,
9 |     "departureDate": "2016-02-11T00:00:00",
10 |     "airlineName": "American"
11 |   },
12 |   {
13 |     "flightCode": "A14244",
14 |     "availableSeats": 10,
15 |     "destination": "SFO",
16 |     "planeType": "Boing 787",
17 |     "origination": "MUA",
18 |     "price": 294,
19 |     "departureDate": "2015/02/12".
```

83. Add an airline parameter equal to united and send the request:

<http://localhost:8081/flights?airline=united>; you should get united flights to SFO sorted by price.

84. Add a code parameter equal to PDF and send the request:

<http://localhost:8081/flights?airline=united&code=PDF>; you should get one united flight to PDF.

The screenshot shows a REST client interface. At the top, there's a header bar with 'GET' dropdown, URL 'localhost:8081/flights?airline=united&code=PDF', 'Params' button, 'Send' button, and 'Save' button. Below the header are tabs for 'Body', 'Cookies', 'Headers (3)', and 'Tests'. The 'Body' tab is selected, showing a status of 'Status: 200' and time '277 ms'. Under 'Body', there are tabs for 'Pretty', 'Raw', 'Preview', and 'JSON'. The 'Pretty' tab is selected, displaying a JSON object with flight details. The JSON output is as follows:

```
1 | {
2 |   "flightCode": "ER95jf",
3 |   "availableSeats": 23,
4 |   "destination": "PDF",
5 |   "planeType": "Boeing 787",
6 |   "origination": "MUA",
7 |   "price": 234,
8 |   "departureDate": "2015/02/12",
9 |   "airlineName": "United"
10| }
11| ]
12| ]
```

85. Change the airline to delta and send the request:

<http://localhost:8081/flights?airline=delta&code=PDF>; you should see the message that there are no flights to PDF – which is correct.

The screenshot shows a REST client interface. At the top, there's a header bar with 'GET' dropdown, URL 'localhost:8081/flights?airline=delta&code=PDF', 'Params' button, 'Send' button, and 'Save' button. Below the header are tabs for 'Body', 'Cookies', 'Headers (3)', and 'Tests'. The 'Body' tab is selected, showing a status of 'Status: 400' and time '363 ms'. Under 'Body', there are tabs for 'Pretty', 'Raw', 'Preview', and 'HTML'. The 'Pretty' tab is selected, displaying an error message. The output is as follows:

```
1 | NO FLIGHTS to PDF
2 | com.mulesoft.weave.mule.exception.WeaveExecutionException: Exception while executing:
3 | payload orderBy $.price
```

86. Remove the airline parameter and send the request: <http://localhost:8081/flights?code=PDF>; you should get the one United flight to PDF, but instead you get the message that there are no flights.

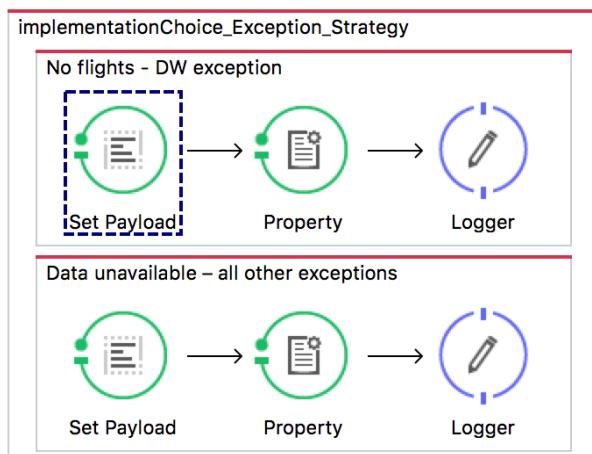
The screenshot shows a Postman request for `localhost:8081/flights?code=PDF`. The response status is 400, and the body contains the following error message:

```
i 1 NO FLIGHTS to PDF
2 com.mulesoft.weave.mule.exception.WeaveExecutionException: Exception while executing:
3 payload orderBy $.price
```

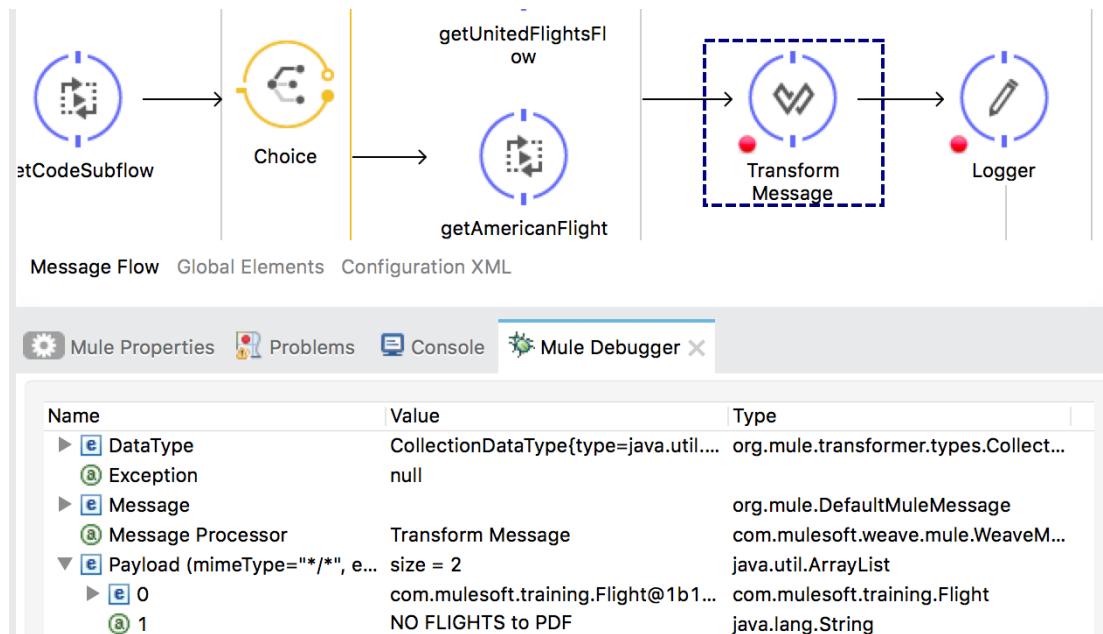
87. Return to Anypoint Studio and stop the project.

Debug the application

88. Debug the project.
89. In Postman, send the same request to <http://localhost:8081/flights?code=PDF>.
90. In the Mule Debugger, step through the application until an exception is thrown – and handled by the default global exception handler.



91. Continue to step through the application until you reach the Transform Message component in getFlightsFlow; you should see the payload still contains the error message in addition to the valid flight results to PDF.



92. Step to the end of the application; you should not get the United results to PDF.

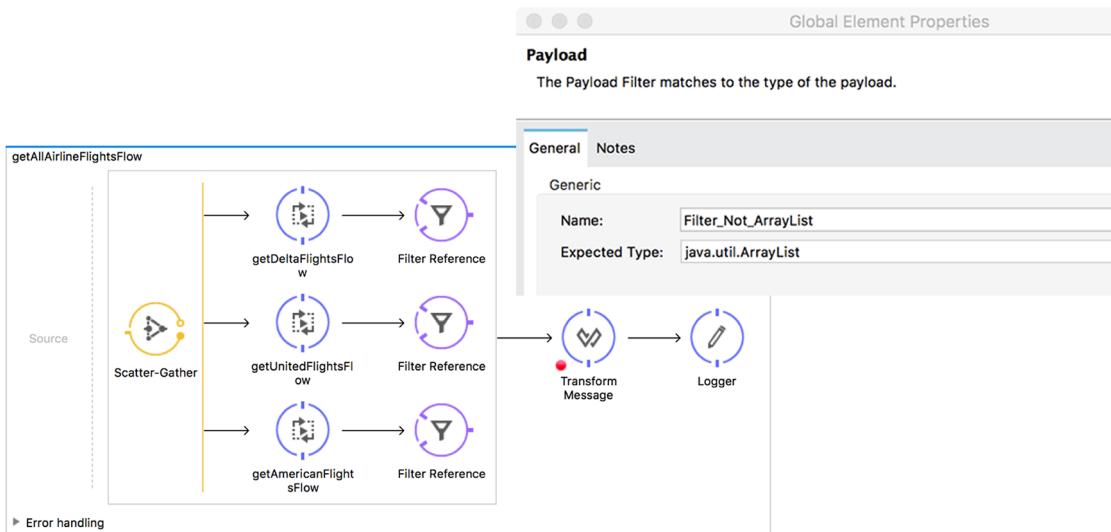
Note: You could write a custom aggregation strategy for the Scatter-Gather router with Java to handle this situation, but instead you will use the simpler approach of filtering out the exception messages in the next walkthrough.

93. Return to Anypoint Studio and stop the project.

Walkthrough 9-3: Filter messages

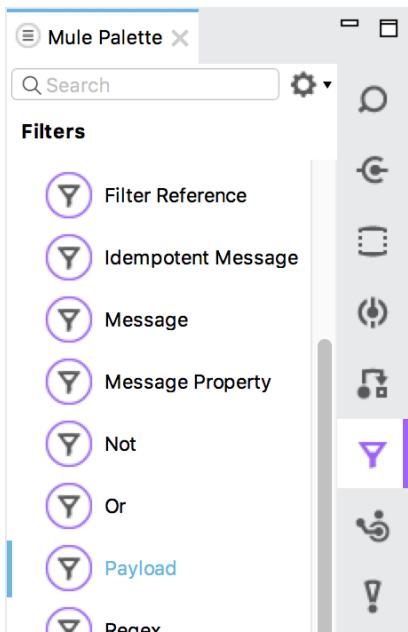
In this walkthrough, you filter the results in the multicast to ensure they are ArrayLists and not exception strings. You will:

- Use the Payload filter.
- Create and use a global filter.



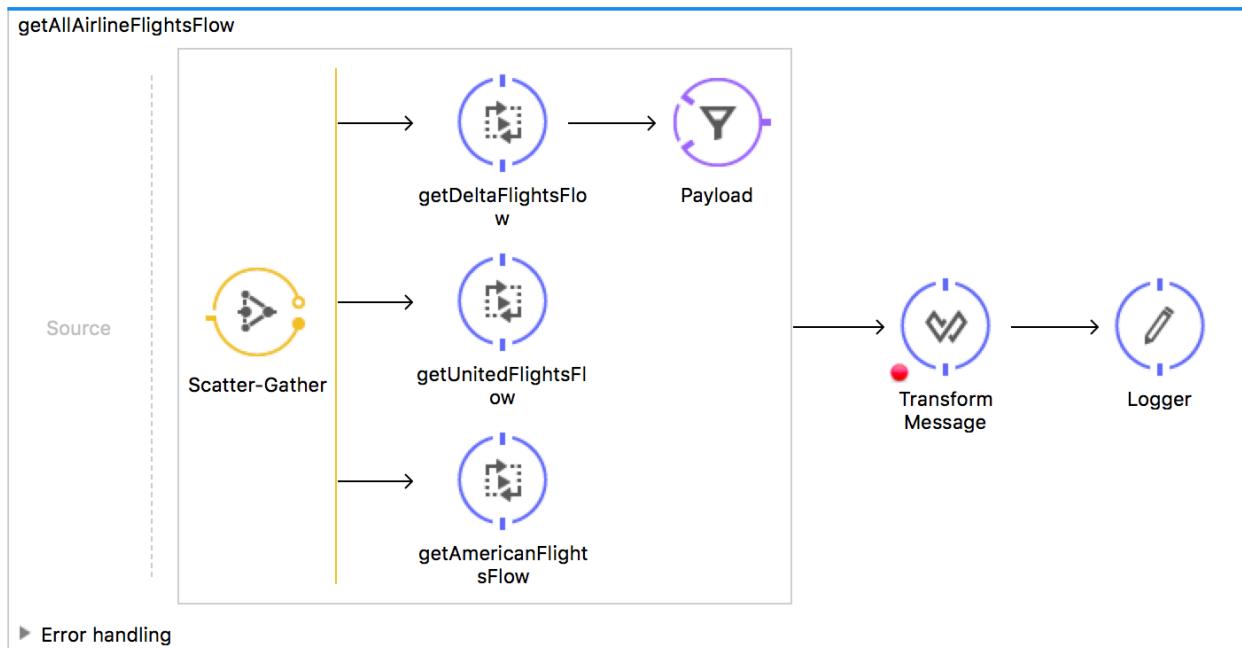
Browse the filter elements in the Mule Palette

1. In the Mule Palette, select the Filters tab.
2. View the available filters.

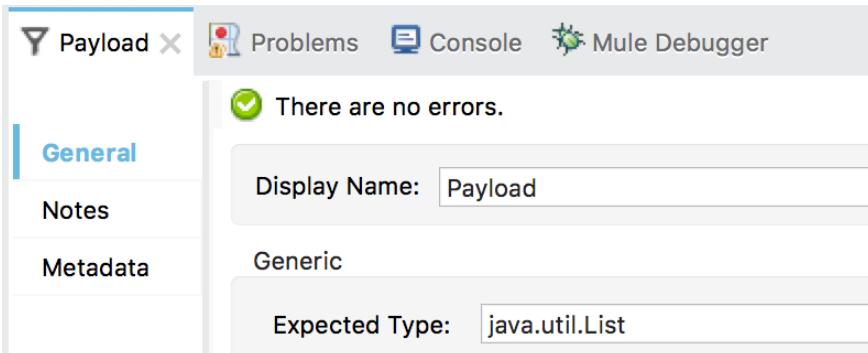


Add a filter

3. Return to getAAirlineFlightsFlow.
4. Drag out a Payload filter from the Mule Palette and drop it after the getDeltaFlightsFlow reference in the Scatter-Gather.



5. In the Payload properties view, set the expected type to java.util.ArrayList.



Test the application

6. Debug the project.
7. In Postman, make the same request to PDF and all airlines.

8. In the Mule Debugger, step to the Transform Message component after the scatter-gather; this time you should see the payload only contains the ArrayLists and not the error string.

Name	Value	Type
Payload (mimeType="/*",...	size = 2	java.util.concurrent.CopyOnWrite...
0	[com.mulesoft.training.Flight@ee...]	java.util.ArrayList
0	com.mulesoft.training.Flight@ee...	com.mulesoft.training.Flight
1	[]	java.util.ArrayList

9. Step to the end of the application.

10. In Postman, view the response; you should see the United results to PDF.

```

1 [ ]
2 {
3   "flightCode": "ER95jf",
4   "availableSeats": 23,
5   "destination": "PDF",
6   "planeType": "Boeing 787",
7   "origination": "MUA",
8   "price": 234,
9   "departureDate": "2015/02/12",
10  "airlineName": "United"
11 }
12 ]
  
```

11. Change the code parameter to FOO and send the request.

12. Click Resume until you step through to the end of the application.

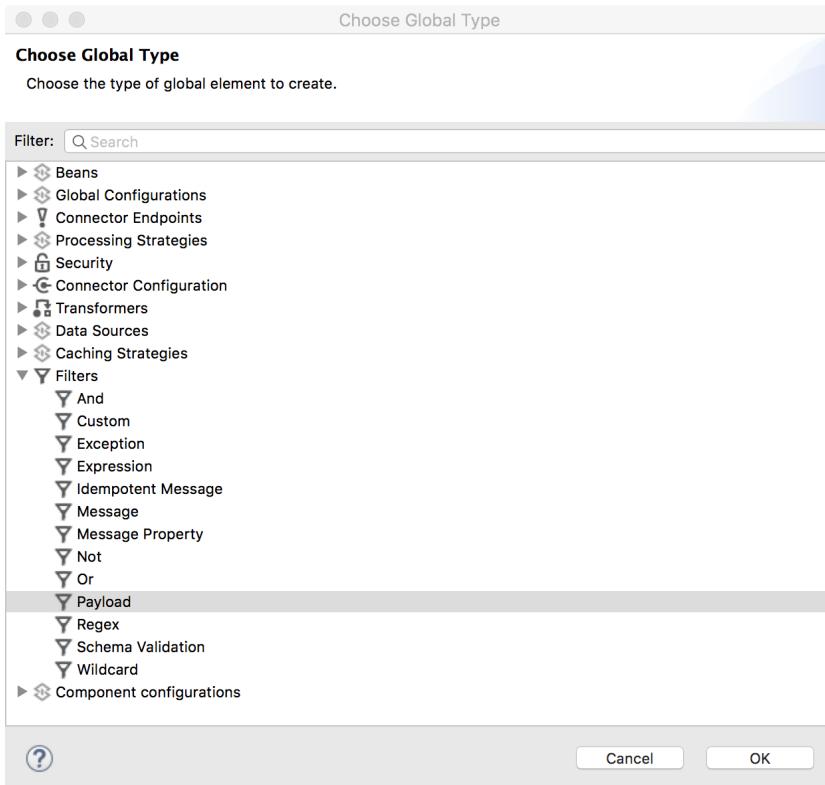
13. In Postman, view the response; you should see an array with the error message.

The screenshot shows the Postman interface. At the top, there's a header with 'GET' selected, the URL 'localhost:8081/flights?code=FOO', and buttons for 'Params', 'Send', and 'Save'. Below the header is a table with tabs for 'Body', 'Cookies', 'Headers (3)', and 'Tests'. The 'Body' tab is active, showing a status of '200' and a time of '8838 ms'. The response body is displayed in JSON format, showing an array with one element containing an error message:

```
[{"error": "NO FLIGHTS to FOO\\ncom.mulesoft.weave.mule.exception.WeaveExecutionException: Exception while executing: \\npayload.flights map ((flight , indexOffFlight) -> {\\n^\\nType mismatch for 'Value Selector' operator\\n      found :binary, :name\\n required :datetime, :name or\\n      required :localdatetime, :name or\\n      required :object, :name or\\n      required :time, :name or\\n      required :array, :name or\\n      required :date, :name or\\n      required :localtime, :name or\\n      required :period, :name")}]
```

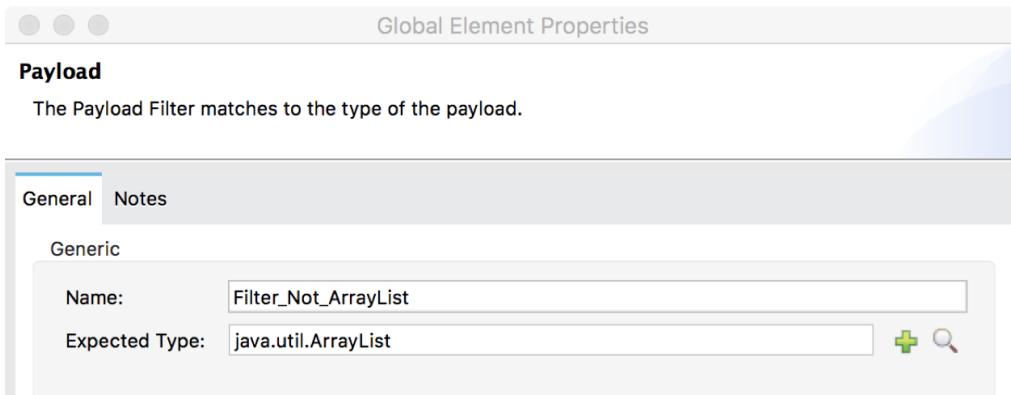
Create a global filter

14. Return to getAllAirlineFlightsFlow.
15. Delete the Payload filter.
16. Return to global.xml.
17. Switch to the Global Elements view and click Create.
18. In the Choose Global Type dialog box, select Filters > Payload and click OK.



19. In the Global Elements dialog box, set the name to Filter_Not_ArrayList.

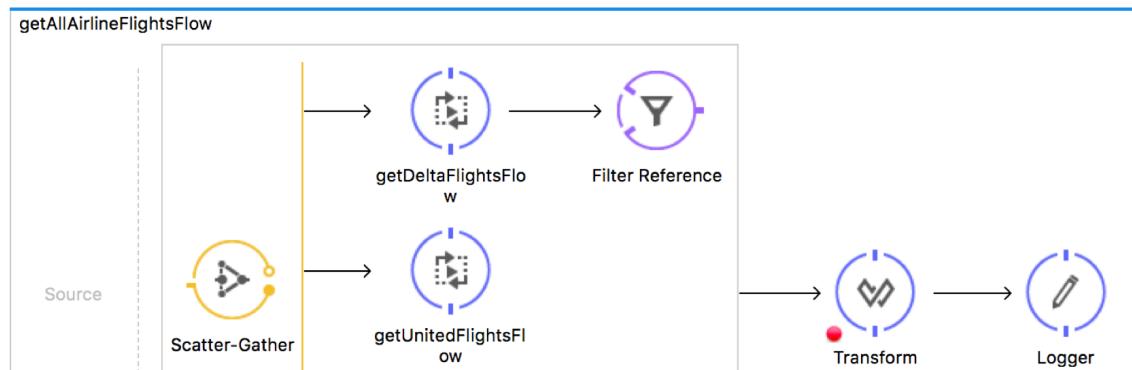
20. Set the expected type to java.util.ArrayList and click OK.



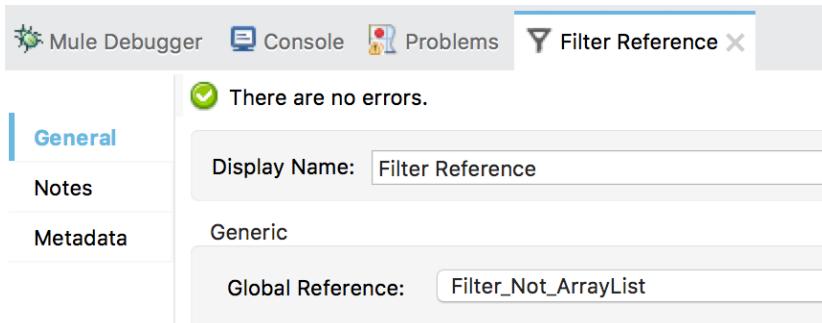
Use the global filter

21. Return to getAllAirlineFlightsFlow in implementation.xml.

22. Drag a Filter Reference from the Mule Palette and drop it after getDeltaFlightsFlow in the Scatter-Gather.



23. In the Filter Reference properties view, set the global reference to Filter_Not_ArrayList.

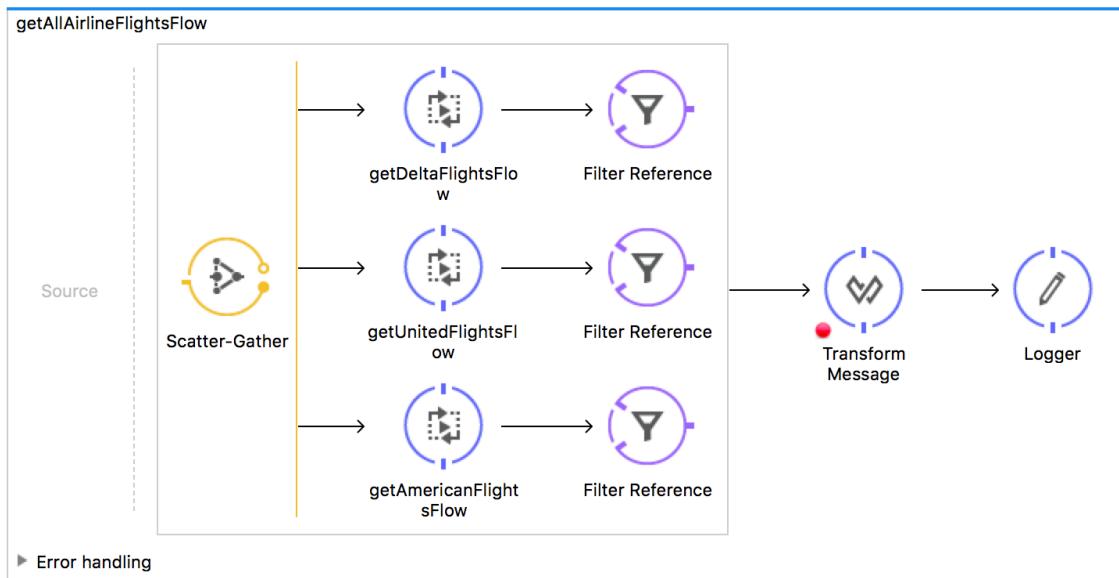


24. Add a Filter Reference after getUnitedFlightsFlow.

25. In the Filter Reference properties view, set the global reference to Filter_Not_ArrayList.

26. Add a Filter Reference after getAmericanFlightsFlow.

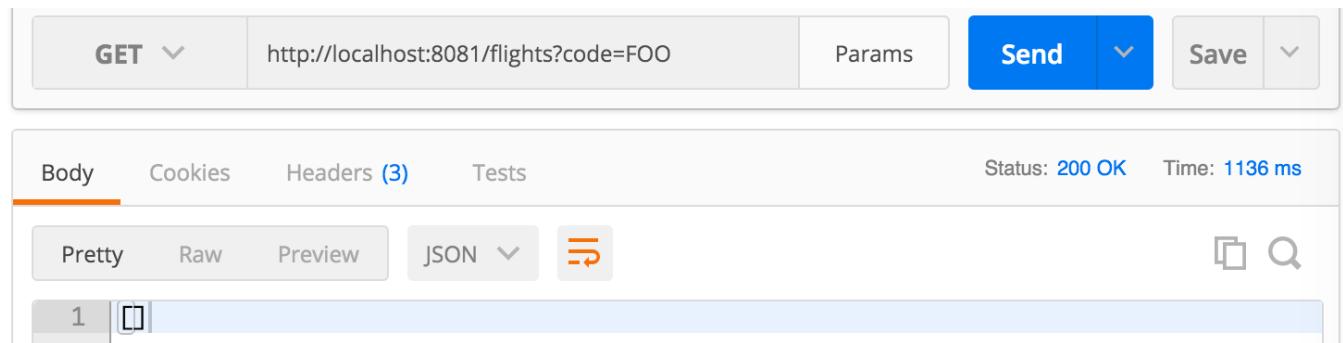
27. In the Filter Reference properties view, set the global reference to Filter_Not_ArrayList.



Test the application

28. Run the project.

29. In Postman, make the same request to <http://localhost:8081/flights?code=FOO>; you should now get no results and an empty array as a result.



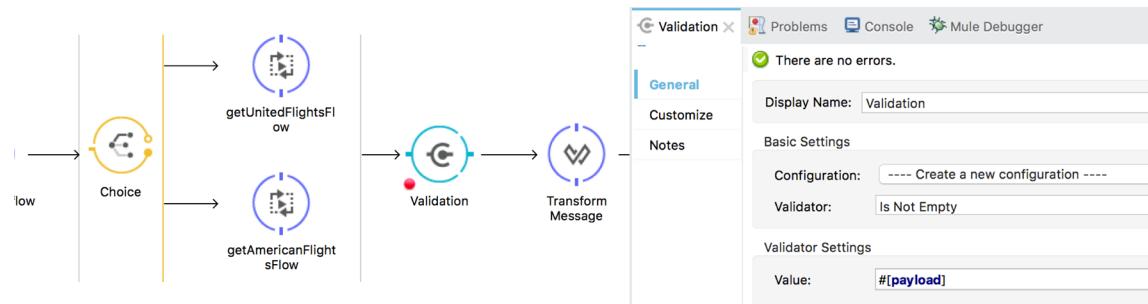
A screenshot of the Postman application interface. The top bar shows a "GET" method, the URL "http://localhost:8081/flights?code=FOO", and a "Send" button. Below the URL, there are tabs for "Body", "Cookies", "Headers (3)", and "Tests". The status bar indicates "Status: 200 OK" and "Time: 1136 ms". The "Body" tab is selected, showing a JSON response with one item. The JSON content is: [{}]. There are buttons for "Pretty", "Raw", "Preview", "JSON", and a copy icon.

30. Return to Anypoint Studio and stop the project.

Walkthrough 9-4: Validate messages

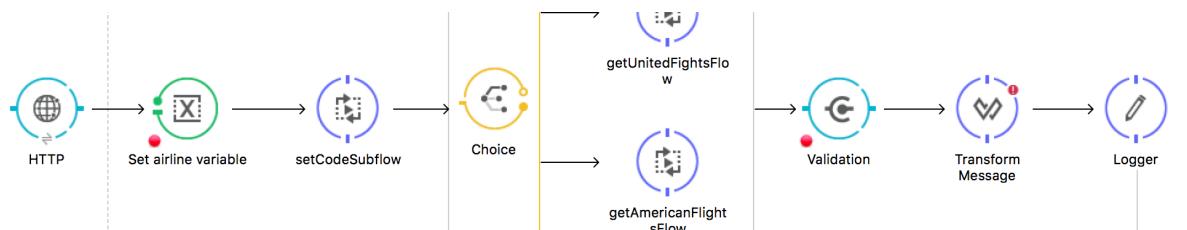
In this walkthrough, you use a validator before the final transformation to check if there are flights and to throw an exception if there are not. You will:

- Use the Validation component to throw an exception.
- Catch the ValidationException in the global exception strategy.



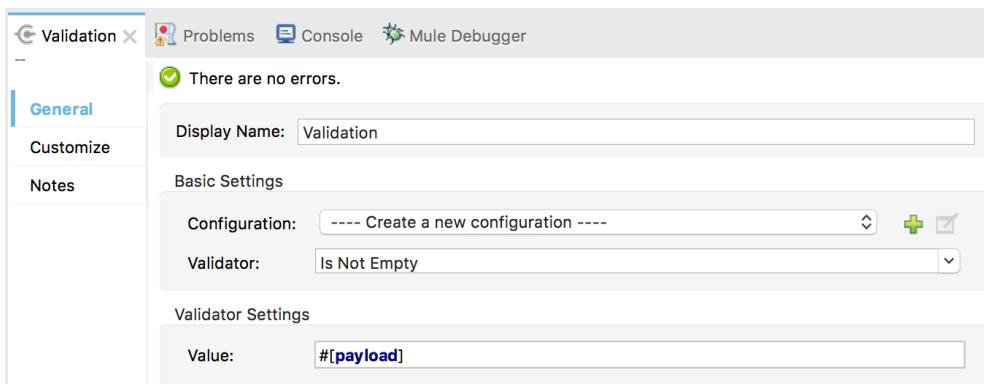
Add a validator

1. Return to getFlightsFlow.
2. Drag out a Validation component from the Mule Palette and drop it before the Transform Message component.

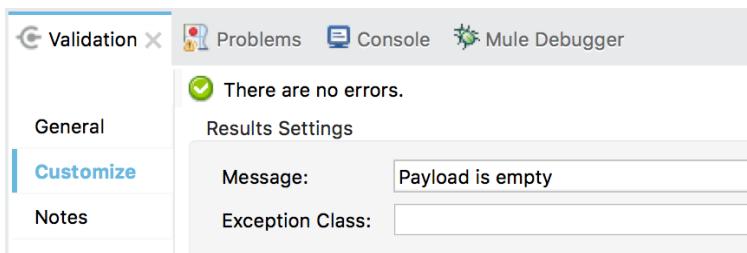


Configure the validator

3. In the Validation properties view, set the validator to Is Not Empty.
4. Set the value to '#[payload].

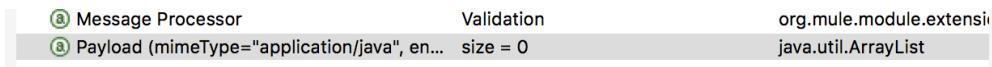


5. In the left-side navigation, click Customize.
6. Set the message to Payload is empty.



Test the application

7. Debug the project.
8. In Postman, make the same request to <http://localhost:8081/flights?code=FOO>.
9. In the Mule Debugger, step to the Validation component after the Choice router; you should see the payload has a size of zero.



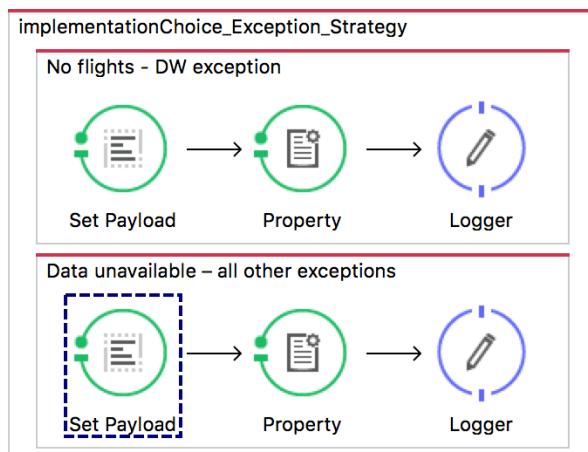
10. Step again and then drill-down into the exceptionThrown element; you should see a ValidationException was thrown.
11. Locate the value you set for the custom message.

The screenshot shows the Mule Studio interface. The top part shows the message flow with components: Variable, setCodeSubflow, Choice, and getUnitedFlightsFlow. The 'Validation' component is highlighted with a red dashed box. The bottom part shows the Mule Debugger window with a table of variables and a message box below it.

Name	Value	Type
exceptionThrown	org.mule.api.MessagingException	org.mule.api.MessagingException
cause	org.mule.extension.validation.api...	org.mule.extension.validation.api...
CAUSE_CAPTION	Caused by:	java.lang.String
causeRollback	false	java.lang.Boolean
detailMessage	org.mule.extension.validation.api...	java.lang.String
EMPTY_THROWABLE_A...	[Ljava.lang.Throwable;@475d2dbc	java.lang.Throwable[]
errorCode	-1	java.lang.Integer
event	MuleEvent: 0-27eceb21-1dd0-1...	org.mule.DefaultMuleEvent
EXCEPTION_MESSAGE...	*****	java.lang.String
EXCEPTION_MESSAGE...	-----	java.lang.String
failingMessageProcessor	org.mule.module.extension.inter...	org.mule.module.extension.inter...
handled	false	java.lang.Boolean
i18nMessage	Payload is empty (org.mule.exten...	org.mule.config.i18n.Message
info	{Payload Type=java.util.ArrayList,...}	java.util.HashMap
message	Payload is empty.	java.lang.String
muleMessage		org.mule.DefaultMuleMessage

Payload is empty.

12. Note that a period has been added to the end of the message.
13. Step again; you should move into the Data unavailable catch exception strategy in global.xml.



14. Step to the end of the application.
15. In Postman, you should see the data unavailable message.

The screenshot shows a Postman request configuration with a 'GET' method and the URL 'http://localhost:8081/flights?code=FOO'. The response details show a status of '500 OK' and a time of '196134 ms'. The 'Body' tab of the response panel displays the error message: 'DATA IS UNAVAILABLE. TRY LATER.' followed by 'org.mule.api.MessagingException: Payload is empty.'.

Modify the global exception handler

16. Return to global.xml.
17. Double-click the No flights catch exception strategy.
18. In the Properties view, modify the execute when expression to also check and see if the exception message is equal to the custom message set by the Validator.

```
# [exception.causeMatches('com.mulesoft.weave.*') ||  
exception.message=='Payload is empty.']}
```

19. Change the display name to No flights.

General

Notes

Display Name: No flights

Settings

Configure conditional execution using an expression

Execute When: #[exception.causeMatches('com.mulesoft.weave.*') || exception.message=='Payload is empty.']}

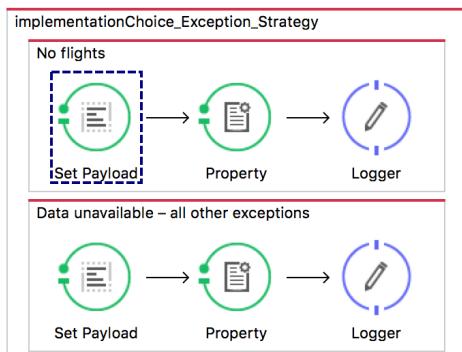
Enable Notifications

Test the application

20. Debug the project.

21. In Postman, make the same request to <http://localhost:8081/flights?code=FOO>.

22. In the Mule Debugger, step past the Validation component into the exception strategy; this time, you should move into the No flights catch exception strategy.



23. Step to the end of the application.

24. In Postman, you should see the no flights message.

GET <http://localhost:8081/flights?code=FOO> Params Send Save

Status: 400 OK Time: 201905 ms

Body Cookies Headers (3) Tests

Pretty Raw Preview HTML

i 1 NO FLIGHTS to FOO
2 org.mule.api.MessagingException: Payload is empty.

25. Return to Anypoint Studio.

26. Close the project.