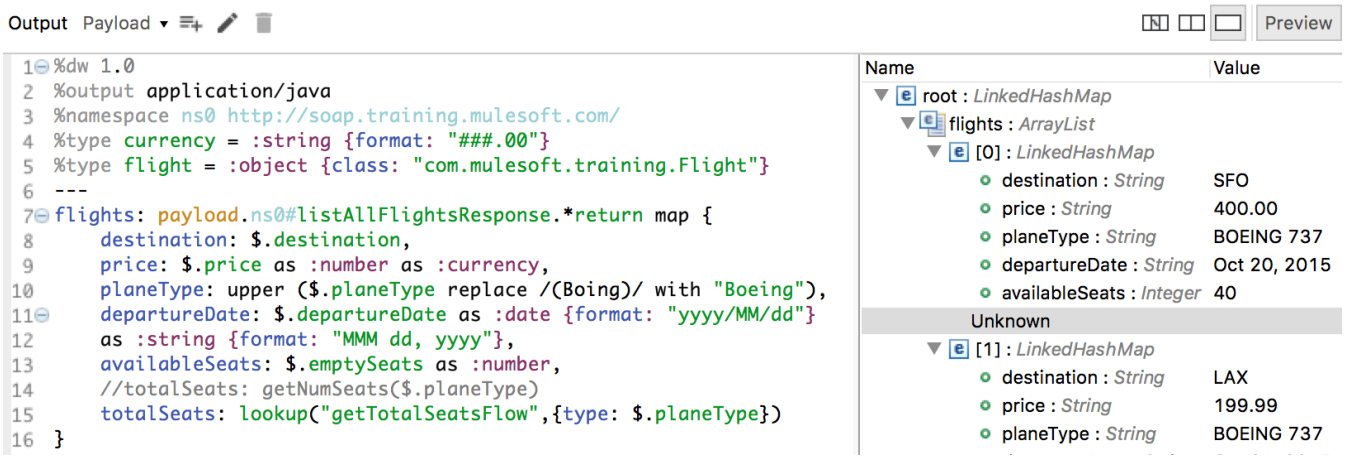
Module 10: Writing DataWeave Transformations



Objectives:

- Write DataWeave expressions for basic XML, JSON, and Java transformations.
- Store DataWeave transformations in external files.
- Write DataWeave transformations for complex data structures with repeated elements.
- Coerce and format strings, numbers, and dates.
- Use DataWeave operators.
- Define and use custom data types.
- Call MEL functions and Mule flows from DataWeave transformations.

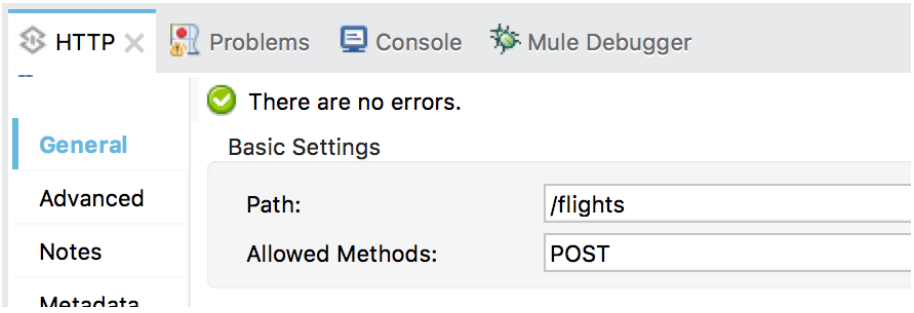
Walkthrough 10-1: Write your first DataWeave transformation

In this walkthrough, you work with JSON data for a flight posted to a flow. You will:

- Create a new flow that receives POST requests.
- Write a DataWeave expression to transform the data to Java, XML, or JSON.
- Add sample data and use live preview.
- Save the transformation in an external file.

Create a new flow

1. Return to implementation.xml.
2. Drag an HTTP connector from the Mule Palette to the bottom of the canvas.
3. Change the name of the new flow to postFlightFlow.
4. In the HTTP properties view, set the connector configuration to the existing HTTP_Listener_Configuration.
5. Set the path to /flights and the allowed methods to POST.

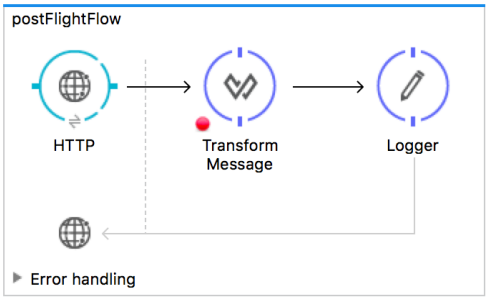


6. Add a Logger to the flow.

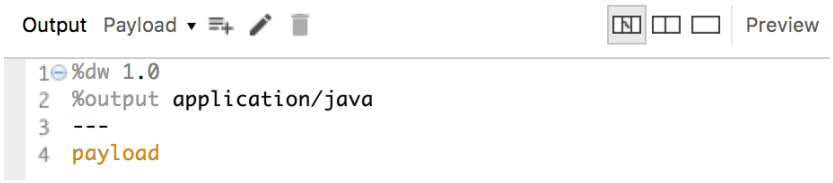
Add a DataWeave Transform Message component

7. Add a DataWeave Transform Message component before the Logger.

8. Add a breakpoint to it.

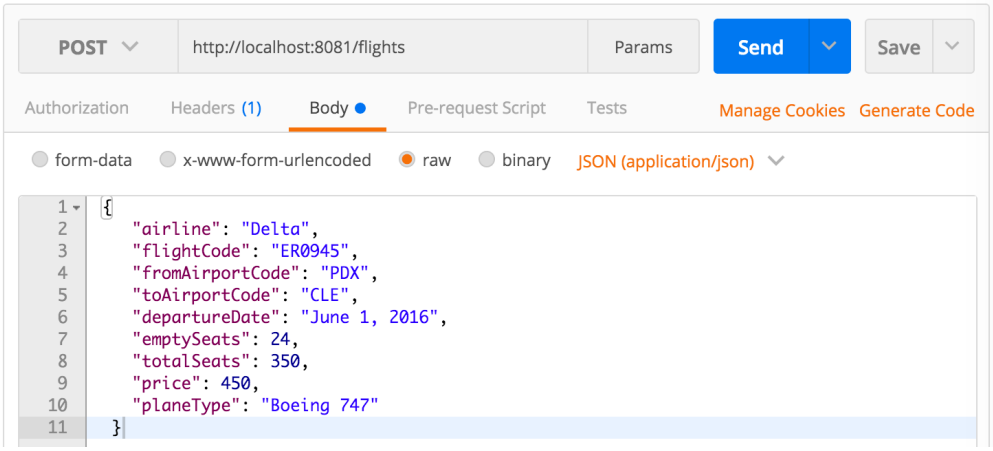


9. In the Transform Message properties view, locate the drop-down menu at the top of the view that sets the output type; it should be set to Payload.
10. Beneath it, locate the directive that sets the output to application/java.
11. Delete the existing DataWeave expression (the curly braces) and set it to payload.

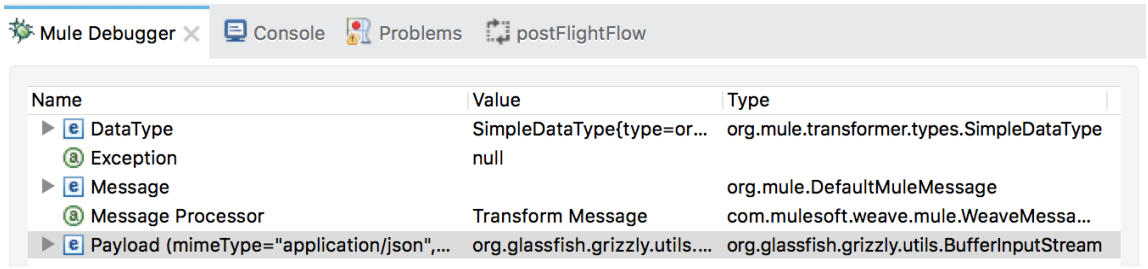


Post data to the flow

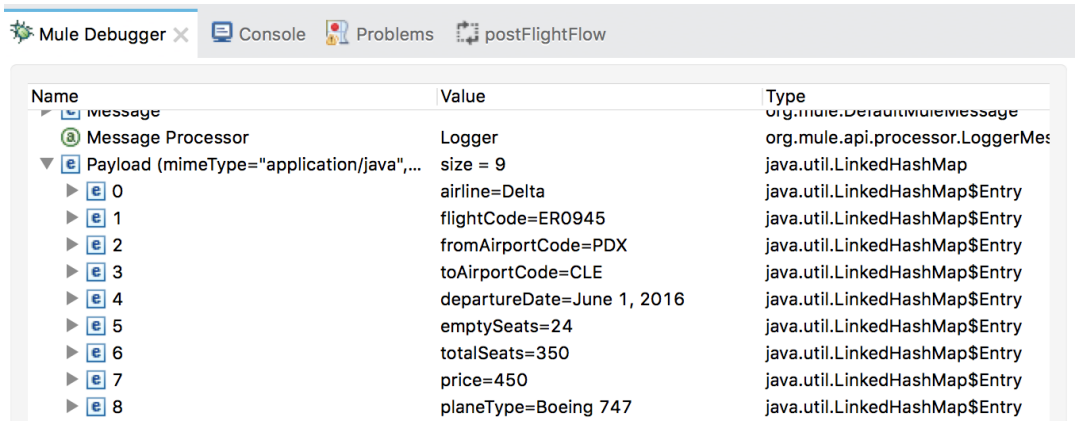
12. Open api.raml in src/main/api.
13. Locate the post method and copy the example for the post body.
14. Debug the project.
15. In Postman, change the method to POST and remove any query parameters.
16. Add a request header called Content-Type and set it equal to application/json.
17. For the request body, select raw and paste the value you coped from api.raml.
18. Make the request to http://localhost:8081/flights.



19. In the Mule Debugger, you should see the payload has a mime-type of application/json and is of type BufferInputStream.



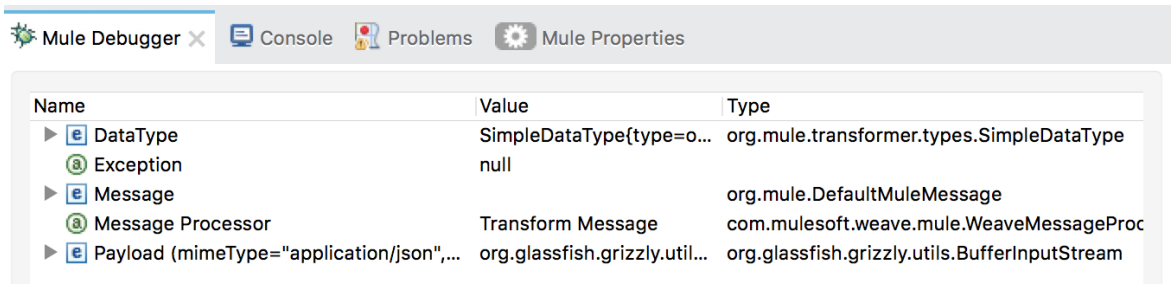
20. Look at the inbound properties; you should see the content-type is set to application/json.
21. Step to the Logger; you should see the payload now has a mime-type of application/java and is a LinkedHashMap.



22. Click the Resume button.

Change output data type to JSON

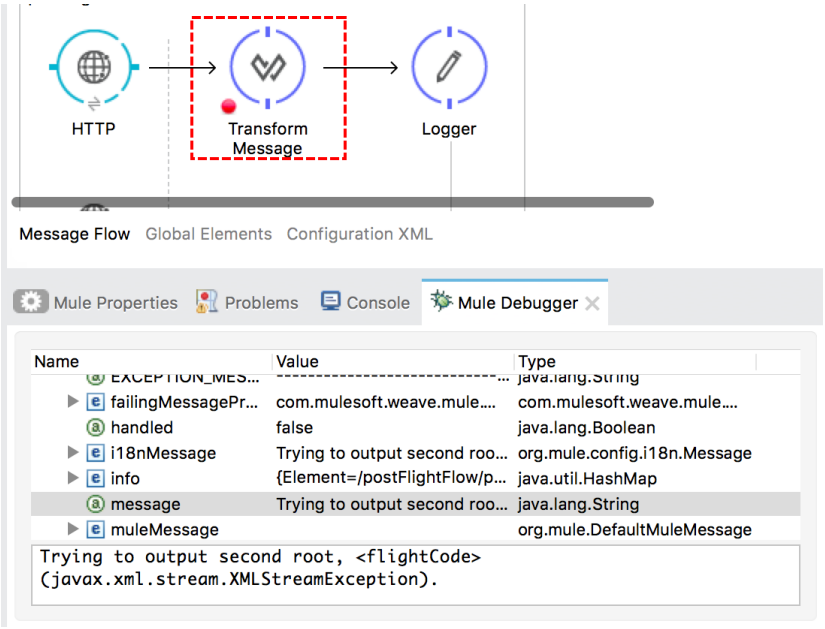
23. In the Transform Message properties view, change the output directive to application/json.
24. Debug the project.
25. In Postman, post the same request to http://localhost:8081/flights.
26. In the Mule Debugger, step to the Logger; you should see the payload has a mime-type of application/json and is a Weave ByteArraySeekableStream.



27. Step to the end of the application.

Change output data type to XML

28. In the Transform Message properties view, change the output directive to application/xml.
29. Debug the project.
30. In Postman, post the same request to http://localhost:8081/flights.
31. In the Mule Debugger, click Next processor; you should get an exception.
32. Drill-down into the exceptionThrown and locate the message; you should see there is a DataWeave error when trying to output the second root.

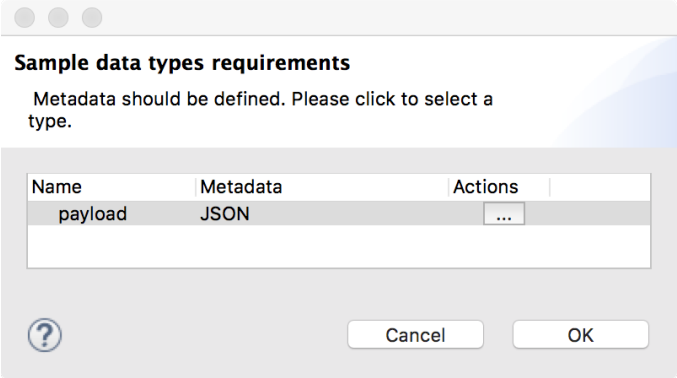


33. Click Resume.
34. Stop the project.

Add input metadata and sample data

35. In the Transform Message properties view, change the output type from application/xml back to application/json.
36. Click the Preview button.
37. Click the Create required sample data to execute preview link.
38. In the Sample data types requirements dialog box, click the word Unknown under Metadata.

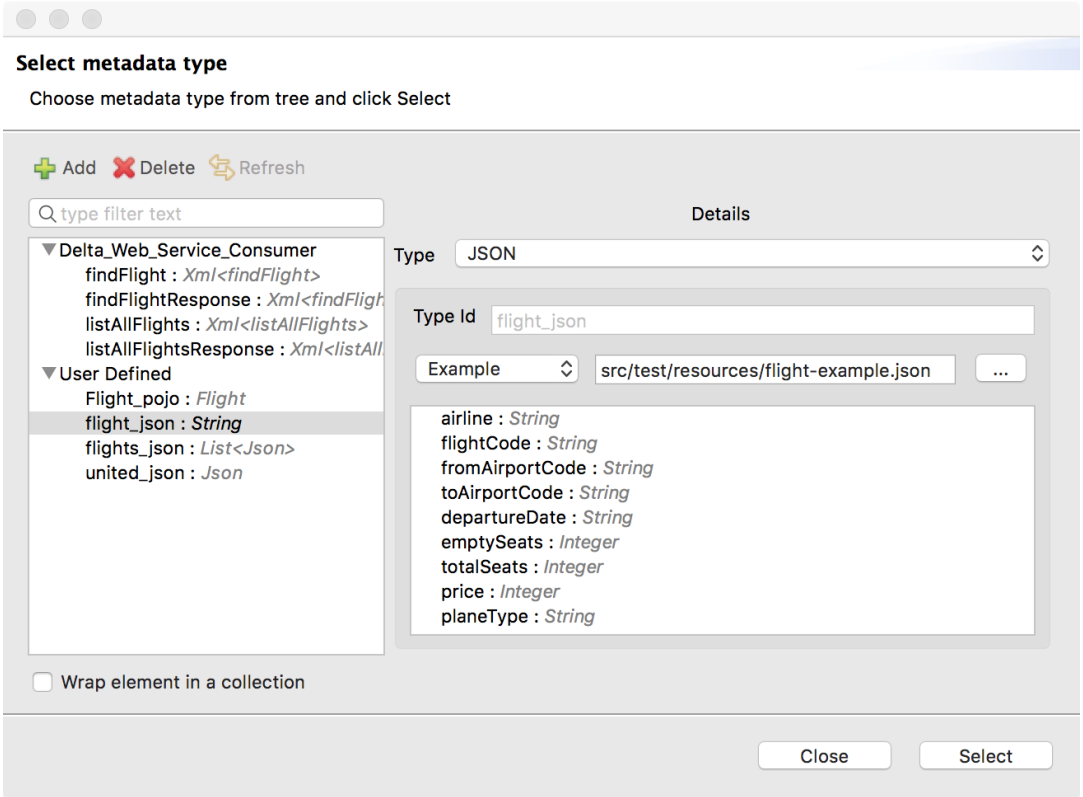
39. In the drop-down menu that appears, select JSON.



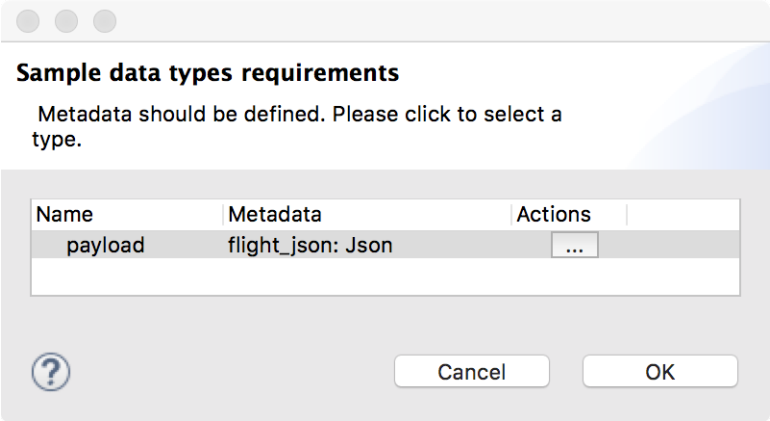
40. Click the button under Actions.
41. In the Select metadata type dialog box, click the Add button.
42. In the Create new type dialog box, set the type id to flight_json and click Create type.
43. In the Select metadata dialog box, set the type to JSON.
44. Select Example and browse to and select the flight-example.json file in src/main/resources.

    Note: Be sure to select flight-example.son (singular) and not flights-example.json (plural).

45. In the Select metadata type dialog box, click Select.

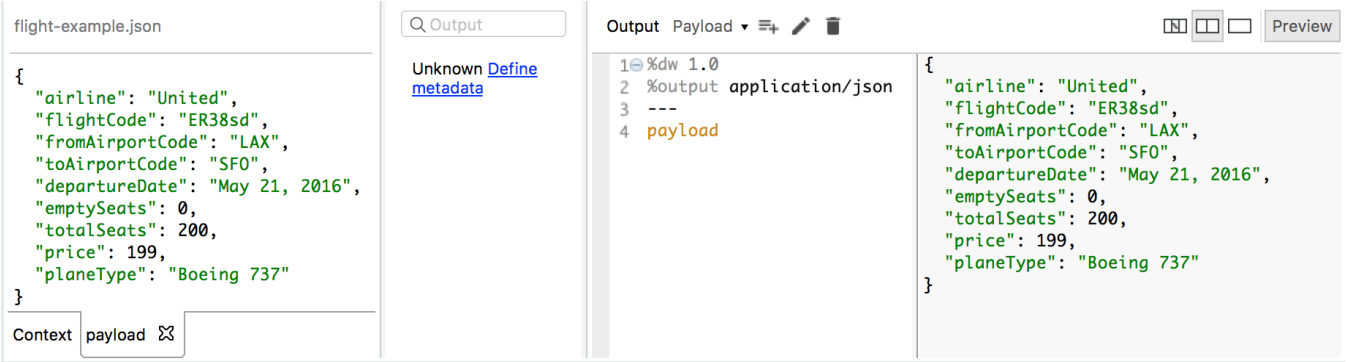


46. In the Sample data types requirements dialog box, locate the new metadata type value; it should be flight_json: Json.
47. Click OK.



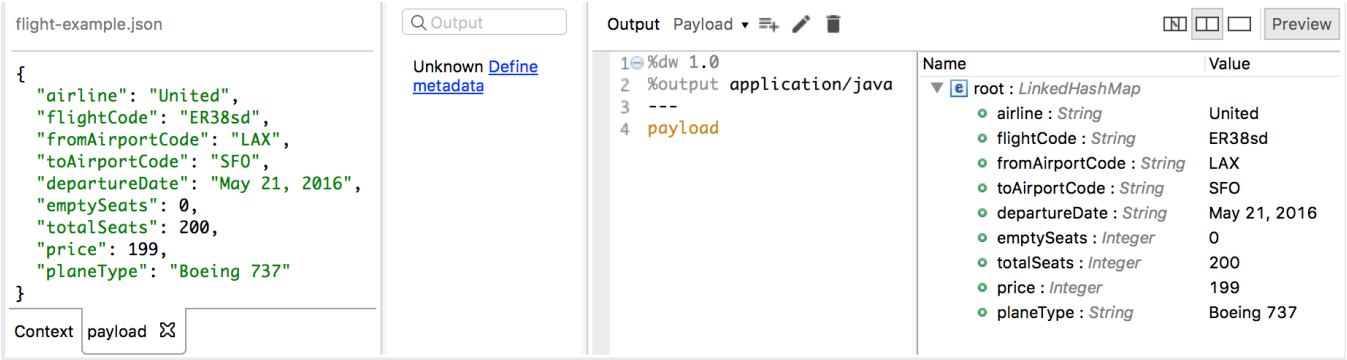
Preview sample data and sample output

48. In the Transform Message properties view, look at the input section; you should see a new tab called payload and it should contain the sample data.
49. Click the Show Source With Trees button to the left of the Preview button.
50. Look at the preview section; you should see the sample output there of type JSON.

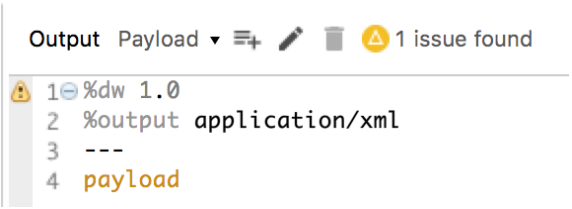


Change the output type

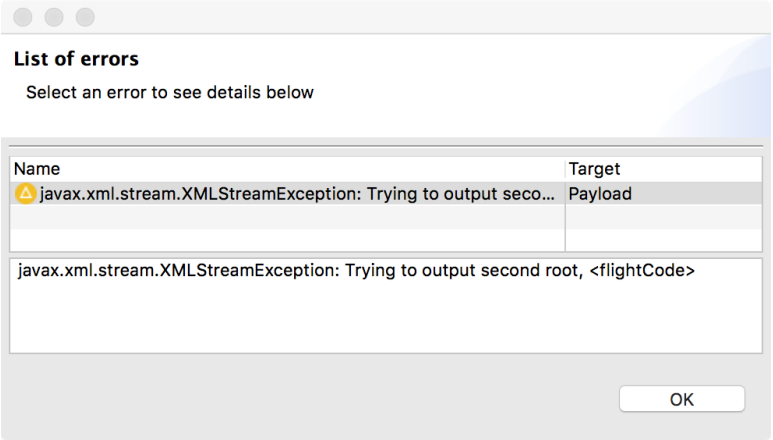
51. In the transform section, change the output type from application/json to application/java.
52. Look at the preview section; you should see the sample output there is now a LinkedHashMap object populated with the sample data.



53. In the transform section, change the output type from application/java to application/xml.
54. Locate the warning icons indicating that there is a problem.



55. Mouse over the icon located to the left of the code; you should see a message that there was an exception trying to output the second root <flightCode>.
56. Click the icon above the code; a List of errors dialog box should open.



57. In the List of errors dialog box, click OK.
58. Change the output from application/xml to application/java for now.

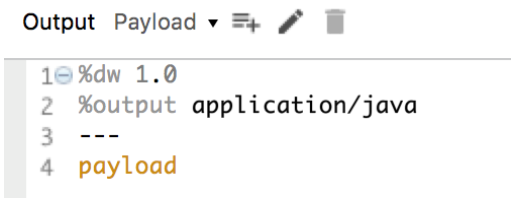
Note: You will learn how to successfully transform to XML in the next walkthrough.

Save the transformation in an external file

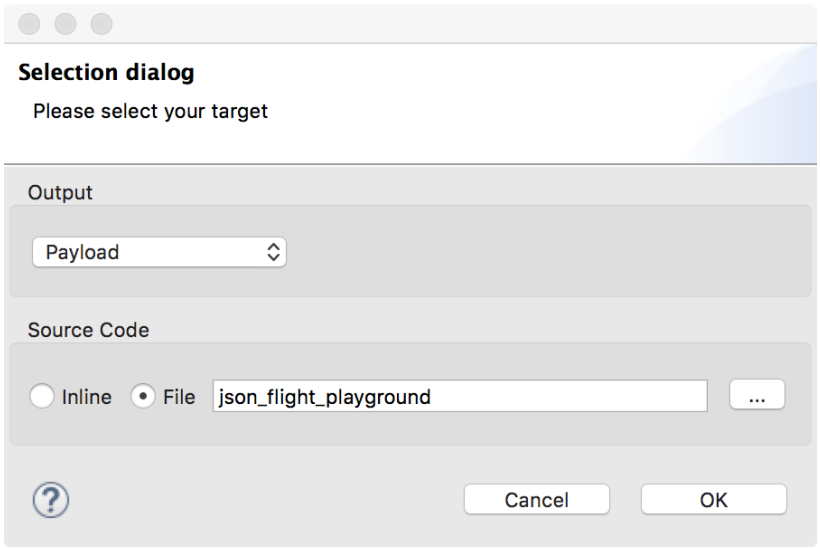
59. Switch the canvas to the Configuration XML view.
60. Locate and review the code for the DataWeave transformation.

```
148     </flow>
149     <flow name="postFlightFlow">
150         <http:listener config-ref="HTTP_Listener_Configuration" path="/flights" allowedM
151         <dw:transform-message metadata:id="5ded8a12-3d01-4c39-98f1-395a9ac25b31" doc:nam
152             <dw:input-payload doc:sample="flight-example.json"/>
153             <dw:set-payload><![CDATA[%dw 1.0
154 %output application/java
155 ---
156 payload]]></dw:set-payload>
157         </dw:transform-message>
158         <logger level="INFO" doc:name="Logger"/>
159     </flow>
```

61. Switch back to the Message Flow view.
62. In the Transform Message properties view, click the Edit current target button (the pencil) above the code.



63. In the Selection dialog dialog box, change the source code selection form inline to file.
64. Set the file name to json_flight_playground.



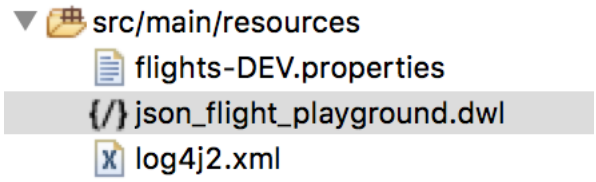
65. Click OK.

Locate and review the code

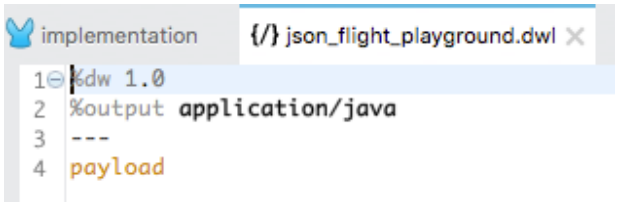
66. Switch to Configuration XML view.
67. Scroll down and look at the new code for the transformation in postFlightFlow.

```
148     </flow>
149     <flow name="postFlightFlow">
150         <http:listener config-ref="HTTP_Listener_Configuration" path="/flights" allowe
151         <dw:transform-message metadata:id="5ded8a12-3d01-4c39-98f1-395a9ac25b31" doc:n
152             <dw:input-payload doc:sample="flight-example.json"/>
153             <dw:set-payload resource="classpath:json_flight_playground.dwl"/>
154         </dw:transform-message>
155         <logger level="INFO" doc:name="Logger"/>
156     </flow>
```

68. In the Package Explorer, expand src/main/resources.



69. Open json_flight_playground.dwl.
70. Review and then close the file.



71. Return to Message Flow view in implementation.xml.
72. Save the file.

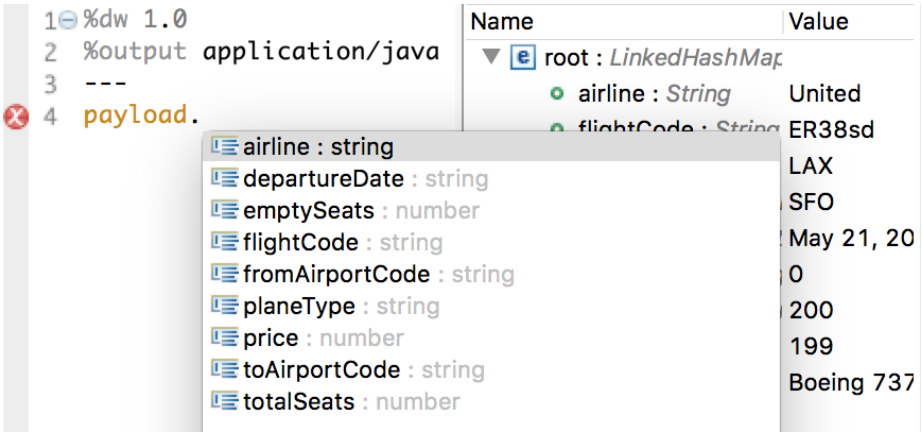
Walkthrough 10-2: Transform basic Java, JSON, and XML data structures

In this walkthrough, you continue to work with the JSON flight data posted to the flow. You will:

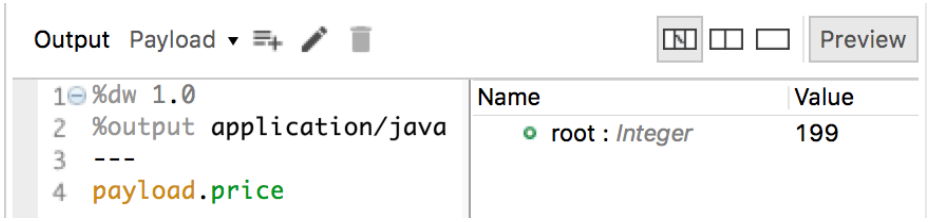
- Write expressions to transform the JSON payload to various Java structures.
- Create a second transformation to store a transformation output in a flow variable.
- Write expressions to transform the JSON payload to various XML structures.

Write expressions to transform JSON to various Java structures

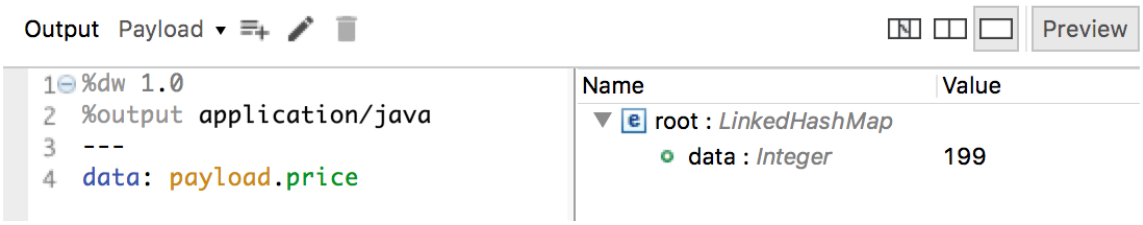
1. Return to the Transform Message properties view in postFlightFlow.
2. Type a period after payload and select price from the pop-up menu.



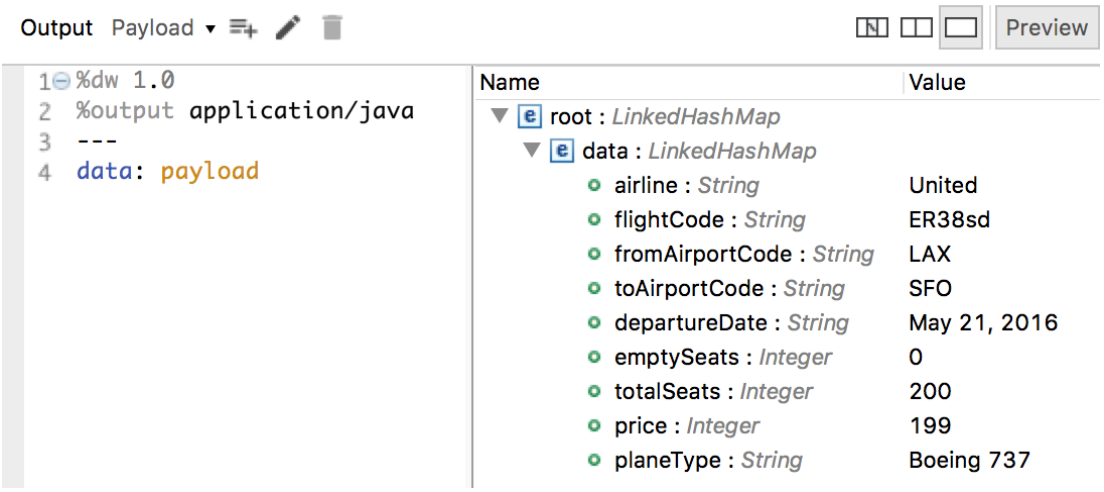
3. Look at the preview; you should see the output is an integer.



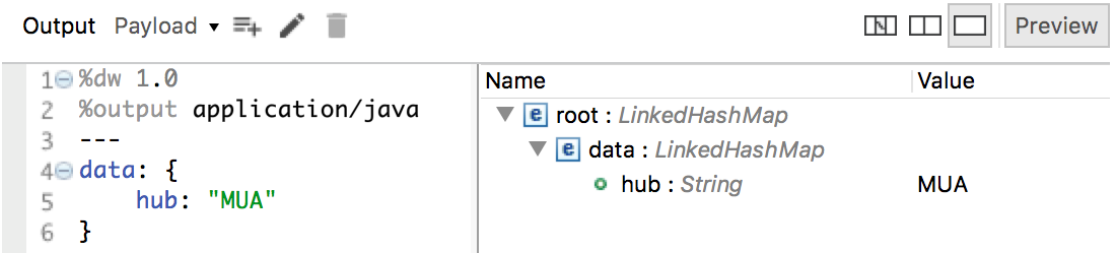
4. Change the DataWeave expression to data: payload.price.



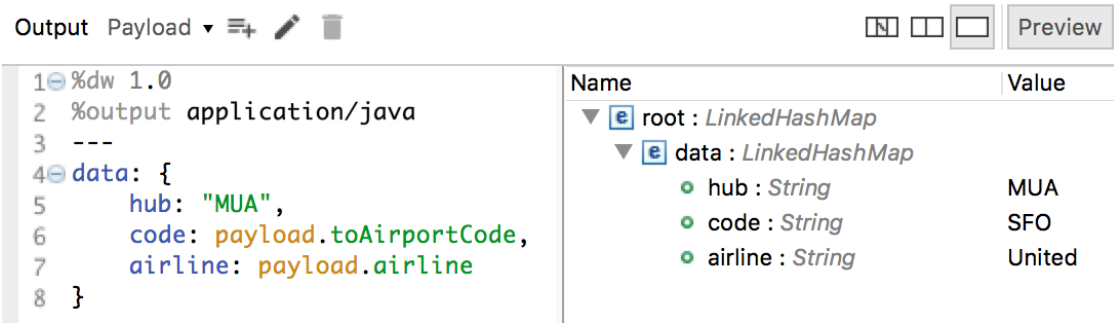
5. Change the DataWeave expression to data: payload.



6. Change the DataWeave expression to data: {}.
7. Add a field called hub and set it to “MUA”.

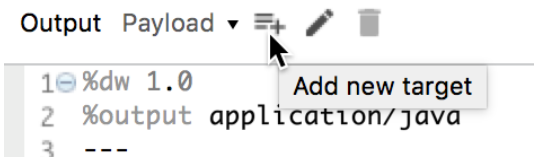


8. Add a field called code and set it to the toAirportCode property of the payload.
9. Add a field called airline and set it to the airline property of the payload.

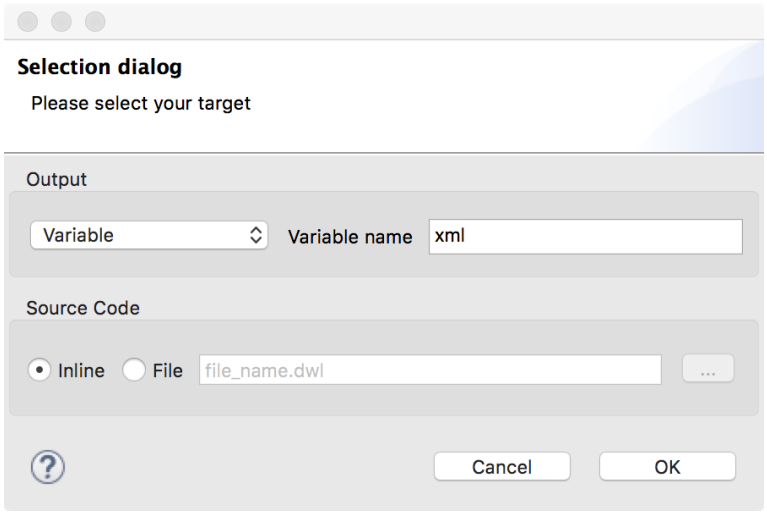


Create a second transformation with the same component

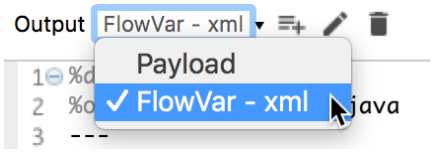
10. Click the Add new target button.



11. In the Selection dialog dialog box, leave the output set to variable.
12. Set the variable name to xml.



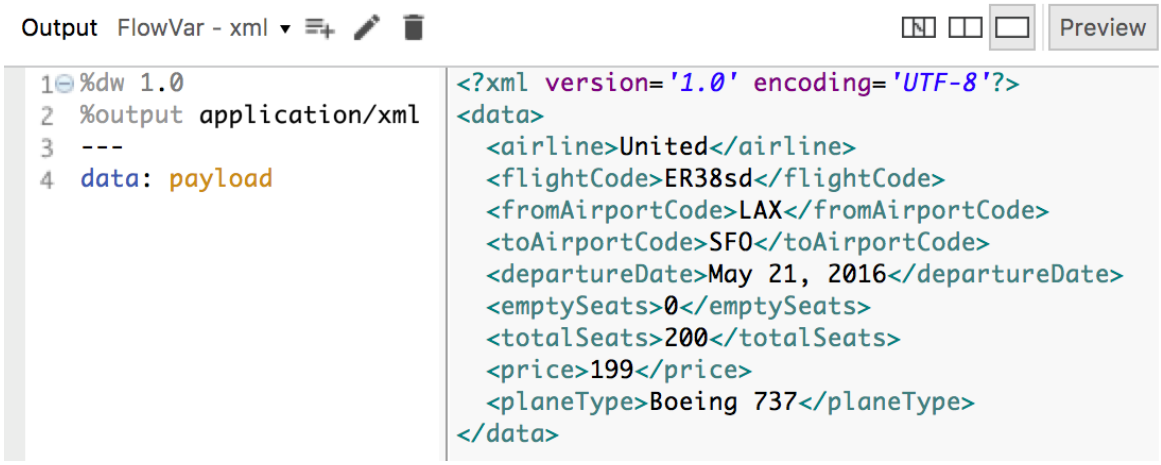
13. Click OK.
14. In the Transform Message properties view, click the drop-down menu button for the output; you should see and can switch between the two transformations.



Write an expression to output data as XML

15. For the FlowVar – xml transformation, click the Preview button.
16. Set the DataWeave expression to payload.
17. Change the output type to application/xml; you should get an issue displayed.
18. Mouse over the warning icon and read the message; it should be the same one you saw in the last walkthrough.
19. Change the DataWeave expression to data: payload.

20. Look at the preview; you should now see XML.



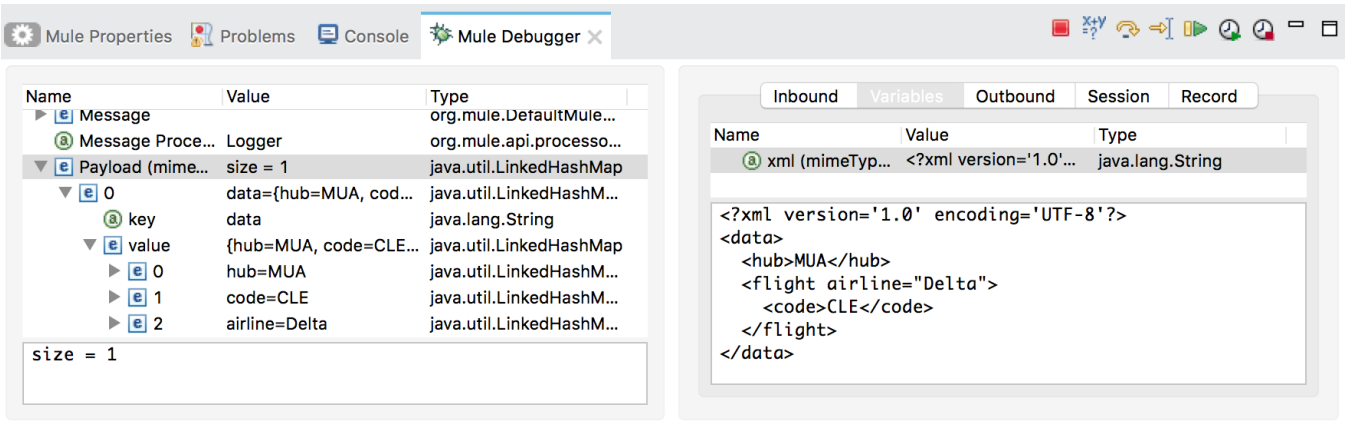
21. In the output section menu bar, click the drop-down arrow next to FlowVar – xml and select Payload.
22. Copy the DataWeave expression you just wrote.
23. Switch back to FlowVar – xml and replace the DataWeave expression with the one you just copied.

24. Modify the expression so the code and airline properties are child elements of a new element called flight.

25. Modify the expression so the airline is an attribute of the flight element.

Debug the application

26. Debug the project.
27. In Postman, post the same request to http://localhost:8081/flights.
28. In the Mule Debugger, step to the Logger and click the Variables tab; you should now see a flow variable.

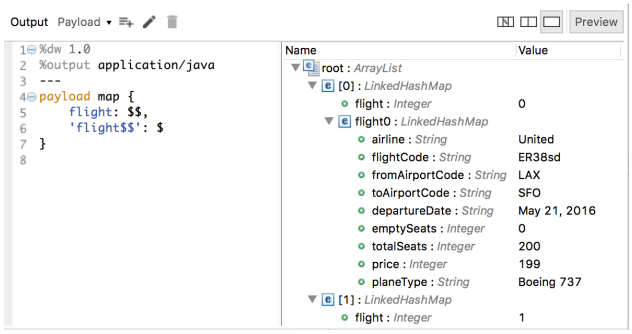


29. Stop the project.

Walkthrough 10-3: Transform complex data structures with arrays

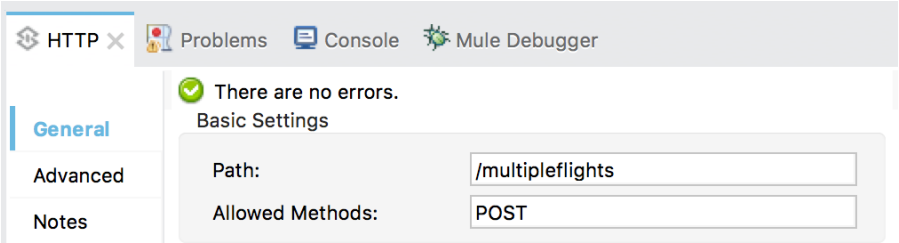
In this walkthrough, you work with JSON data for multiple flights posted to a flow. You will:

- Create a new flow that receives POST requests of a JSON array of objects.
- Transform a JSON array of objects to Java.



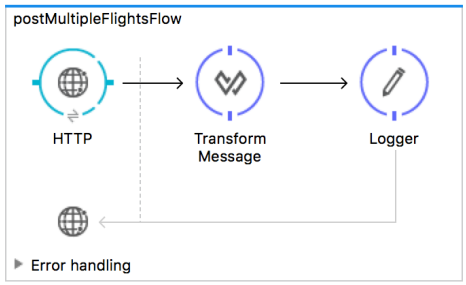
Create a new flow

1. Return to implementation.xml.
2. Drag out an HTTP connector to the bottom of the canvas.
3. Change the flow name to postMultipleFlightsFlow
4. In the HTTP Properties view, set the connector configuration to the existing HTTP_Listener_Configuration.
5. Set the path to /multipleflights and set the allowed methods to POST.



Transform the payload to Java

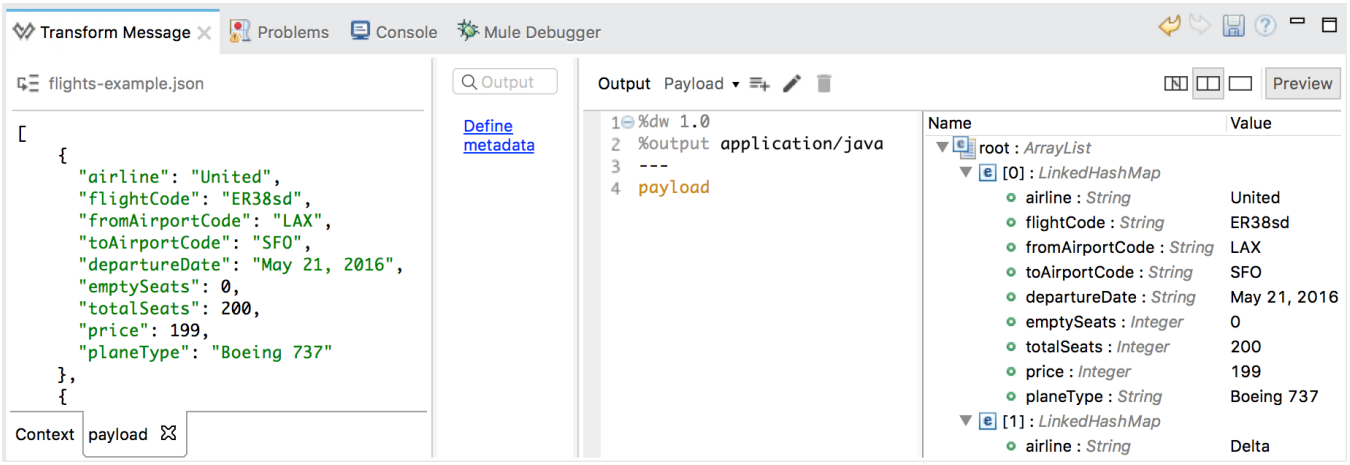
6. Add a Transform Message component and a Logger to the flow.



7. In the Transform Message properties view, set the DataWeave expression to payload.
8. Click the Preview button.

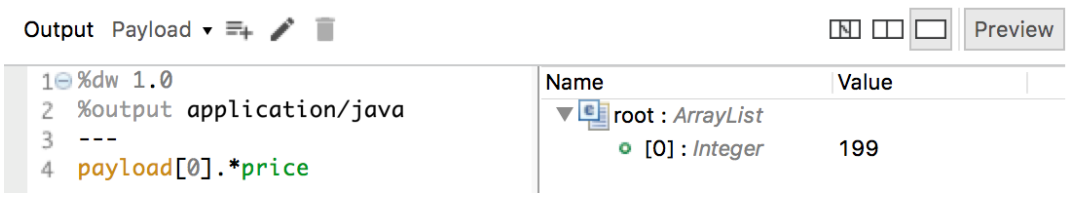
Add input metadata and sample data

9. In the preview section, click the Create required sample data to execute preview link.
10. In the Sample data types requirements dialog box, set the metadata type to JSON.
11. Click the Actions button.
12. In the Select metadata dialog box, select flights_json and click Select.
13. In the Sample data types requirements dialog box, click OK.
14. Look at the preview section; the sample output should be an ArrayList of LinkedHashMaps.



Return values for the first object in the collection

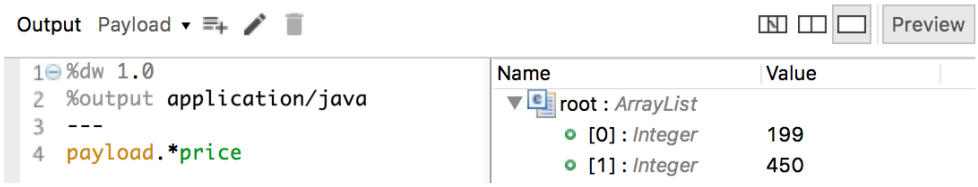
15. Change the DataWeave expression to payload[0]; in the preview section, you should see one LinkedHashMap object.
16. Change the DataWeave expression to payload[0].price; in the preview section, you should get 199, the first price value.
17. Change the DataWeave expression to payload[0].*price; in the preview section, you should see an ArrayList with one integer value of 400 – the first price value.



Return collections of values

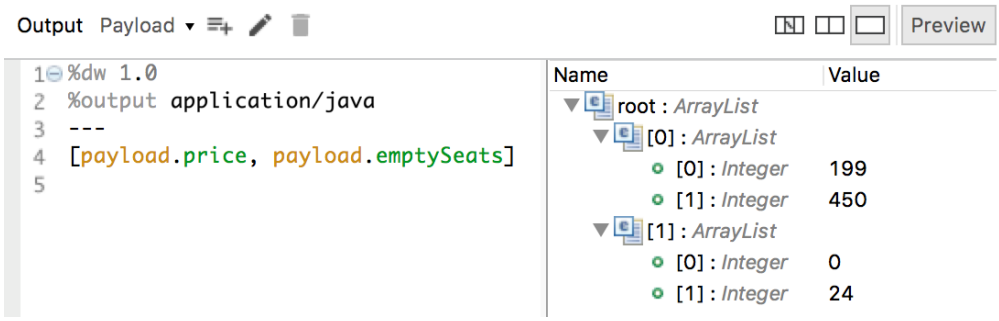
18. Change the DataWeave expression to return a collection of all the prices; in the preview section, you should see two prices in an ArrayList.

    ```
    payload.*price
    ```



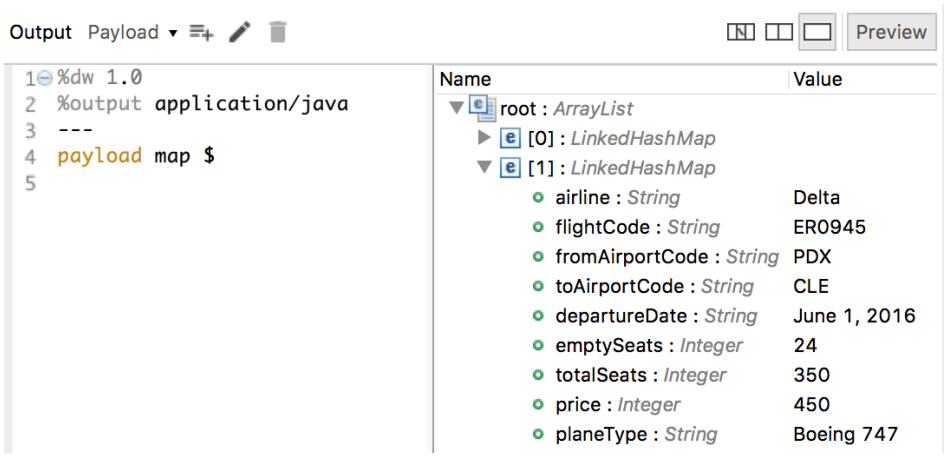
19. Change the DataWeave expression to payload.price; you should get the same result.
20. Change the DataWeave expression to return a collection of prices and available seats; in the preview section, you should see an ArrayList with two ArrayLists.

    ```
    [payload.price, payload.emptySeats]
    ```

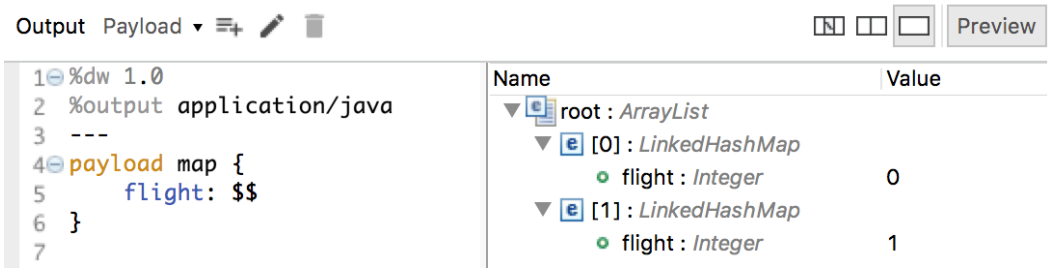


Use the map operator to return object collections with different data structures

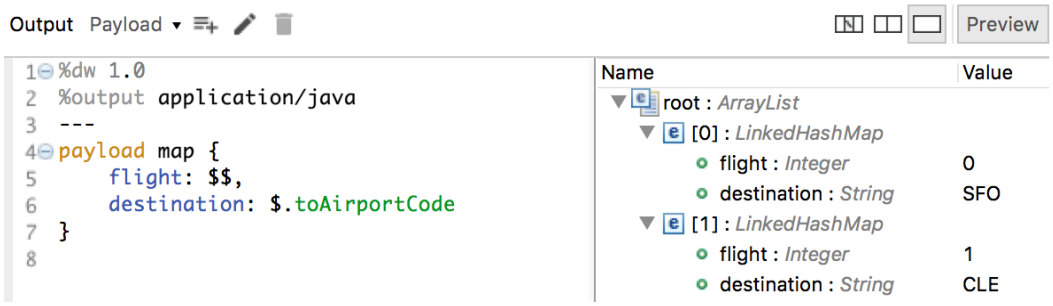
21. Change the DataWeave expression to payload map $; in the preview section, you should see two ArrayList of two LinkedHashMaps again.



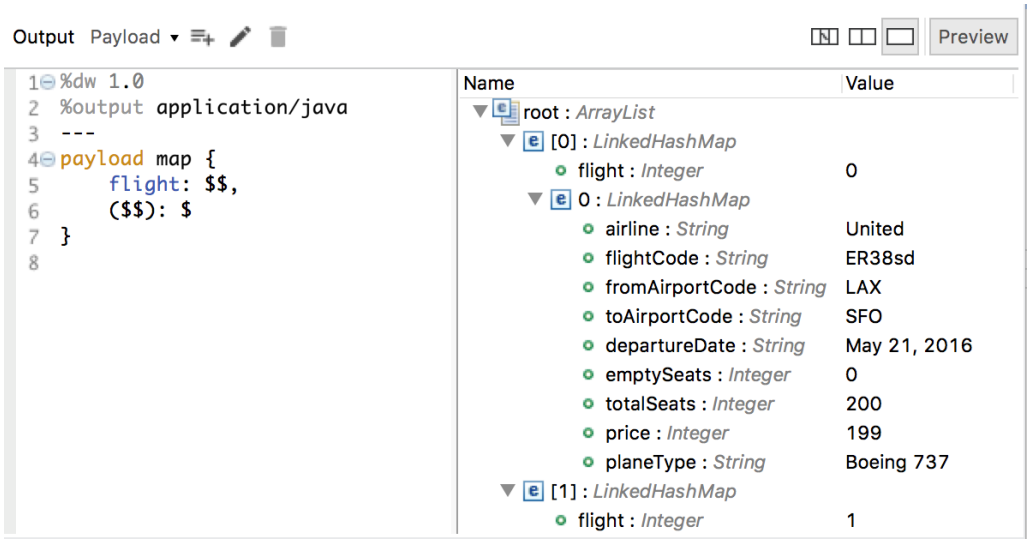
22. Change the DataWeave expression to set a property called flight for each object that is equal to its index value in the collection.



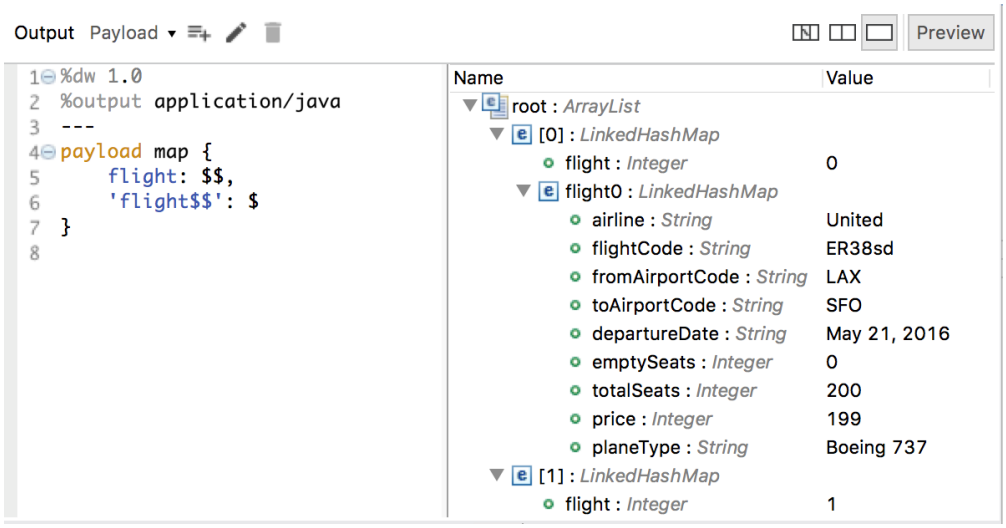
23. Change the DataWeave expression to create a property called destination for each object that is equal to the value of the toAirportCode field in the payload collection.



24. Change the DataWeave expression to dynamically add each of the properties present on the payload objects.



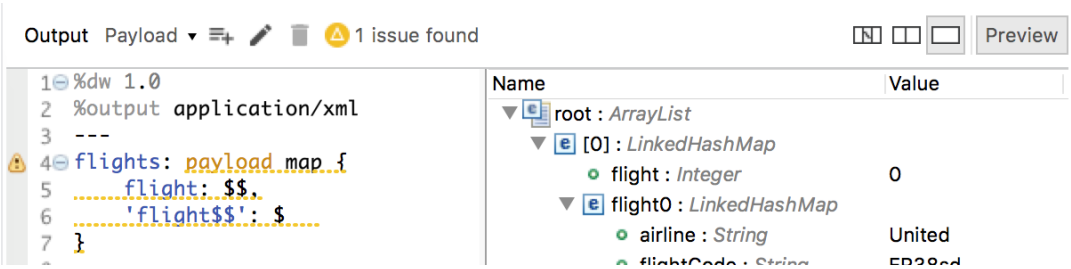
25. Change the DataWeave expression so the name of each object is flight+index and you get flight0, flight1, and flight2.



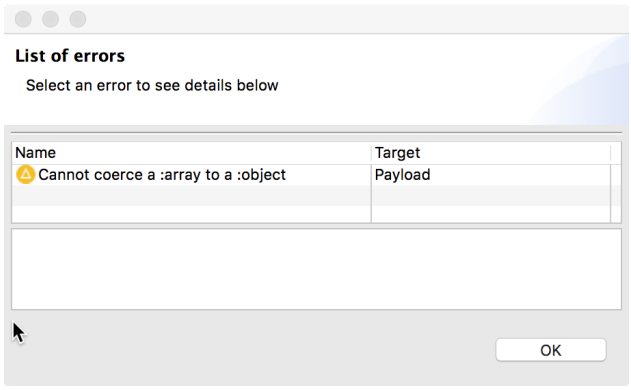
Note: If you want to test the application, change the output type to application/json and then in Postman, change the request URL to http://localhost:8081/multipleflights and replace the request body with JSON from the flights-example.json file, .

Change the expression to output XML

26. Change the DataWeave expression output type from application/java to application/xml; you should get an issue displayed.
27. Change the DataWeave expression to create a root node called flights; you should still get an issue.



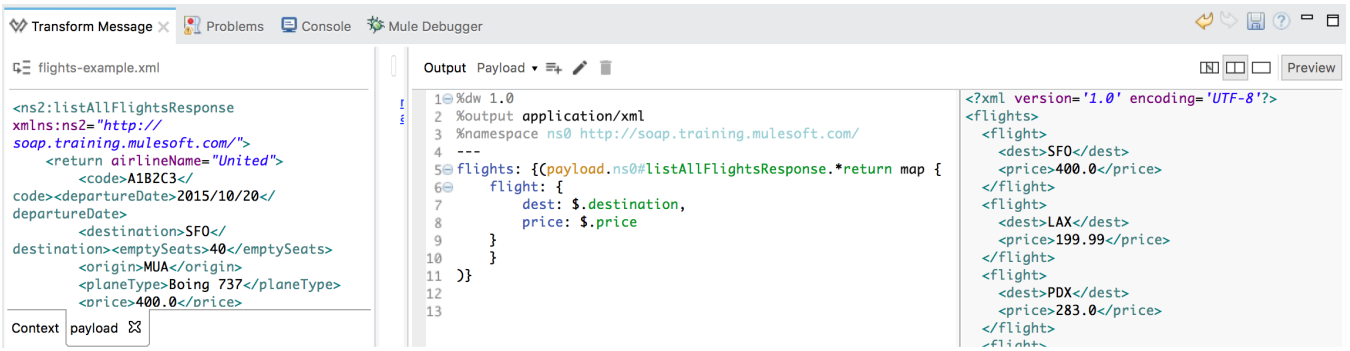
28. Look at the error message.



Walkthrough 10-4: Transform to and from XML with repeated elements

In this walkthrough, you continue to work with the JSON data for multiple flights posted to the flow. You will:

- Transform a JSON array of objects to XML.
- Transform XML with repeated elements to Java.



Transform the JSON array of objects to XML

1. Return to the Transform Message properties view in postMultipleFlightsFlow.
2. Change the expression to map each item in the input array to an XML element.

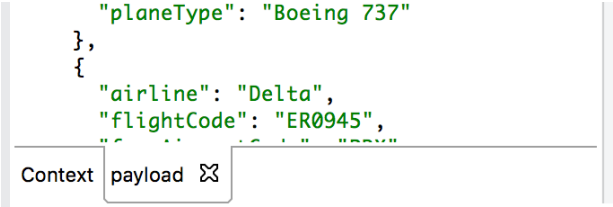
```
flights: {(payload map {
      flight: $$,
      'flight$$':$
    }
)}
```

3. Look at the preview; the JSON should be transformed to XML successfully.

4. Remove the flight property.
5. Modify the expression so the flights object has a single property called flight.

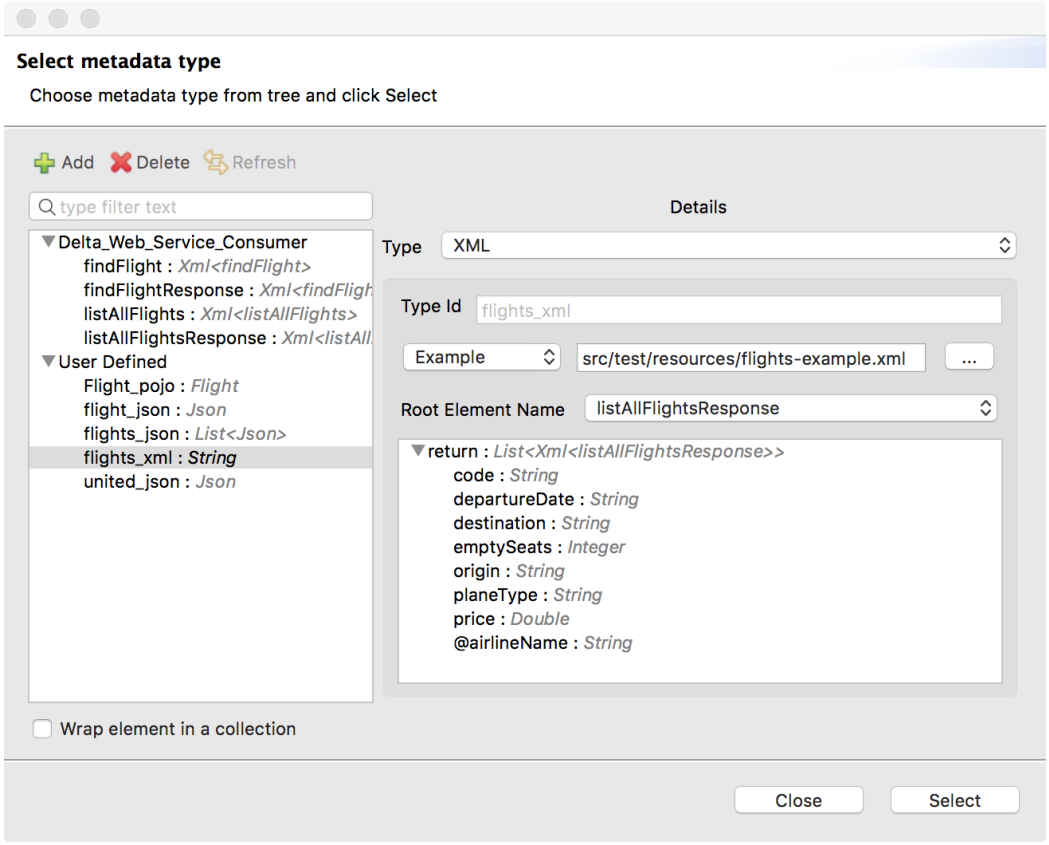
Change input metadata and sample data to XML

6. In the input section, click the x on the payload tab.

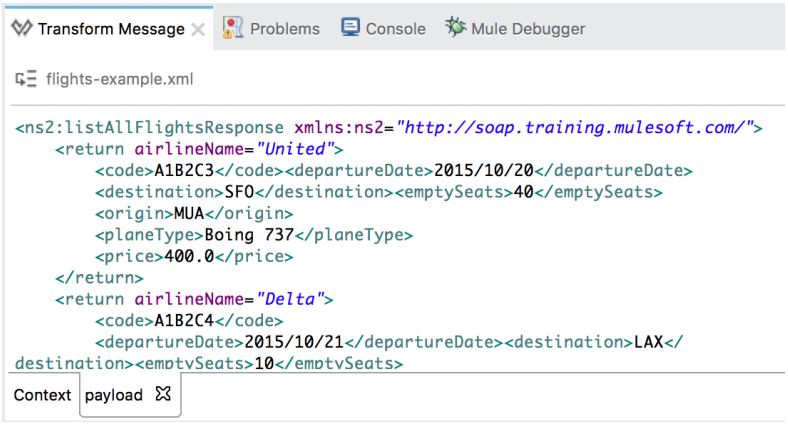


7. In the Close Sample Data dialog box, click Close tab and Keep file.
8. In the input section, right-click Payload: List<Json> and select Clear Metadata.
9. In the preview section, click the link to Create required sample data to execute preview.
10. In the Sample data types requirements dialog box, set the metadata type to XML.
11. Click the Actions button.
12. In the Select metadata type dialog box, click the Add button.
13. In the Create new type dialog box, set the type id to flights_xml and click Create type.
14. In the Select metadata dialog box, set the type to XML.
15. Select Example and browse to and select the flights-example.xml file in src/main/resources.

16. Click Select.



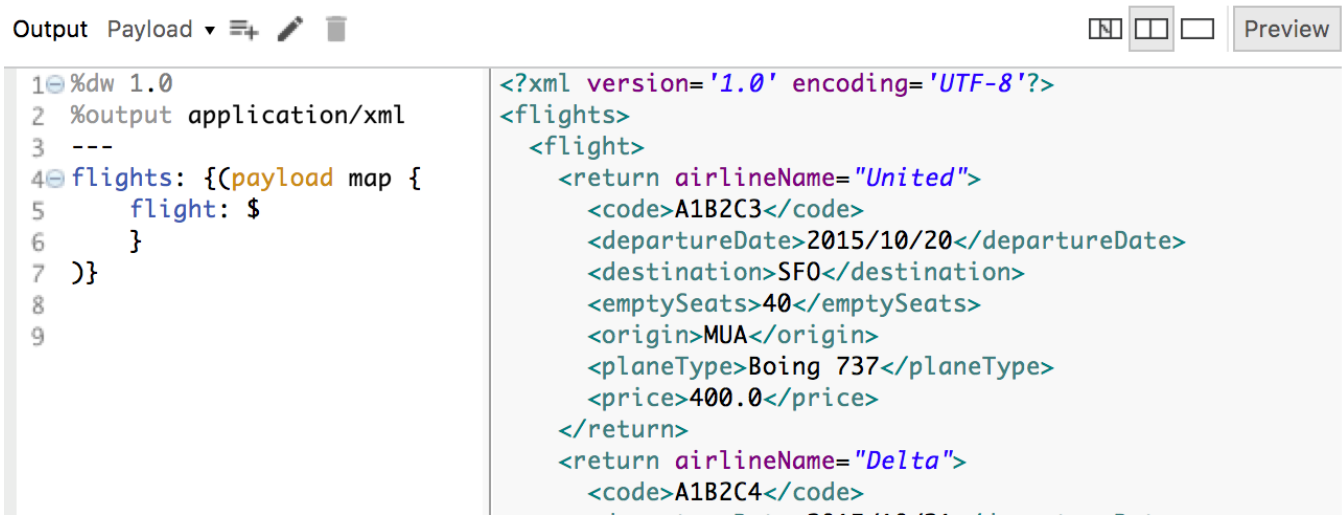
17. In the Sample data types requirements dialog box, click OK.
18. Look at the XML sample data in the input section.



Examine the sample output for the transformation

19. Review the transformation expression.

20. Look at the preview section; you should see all the flights are children of the flight element.

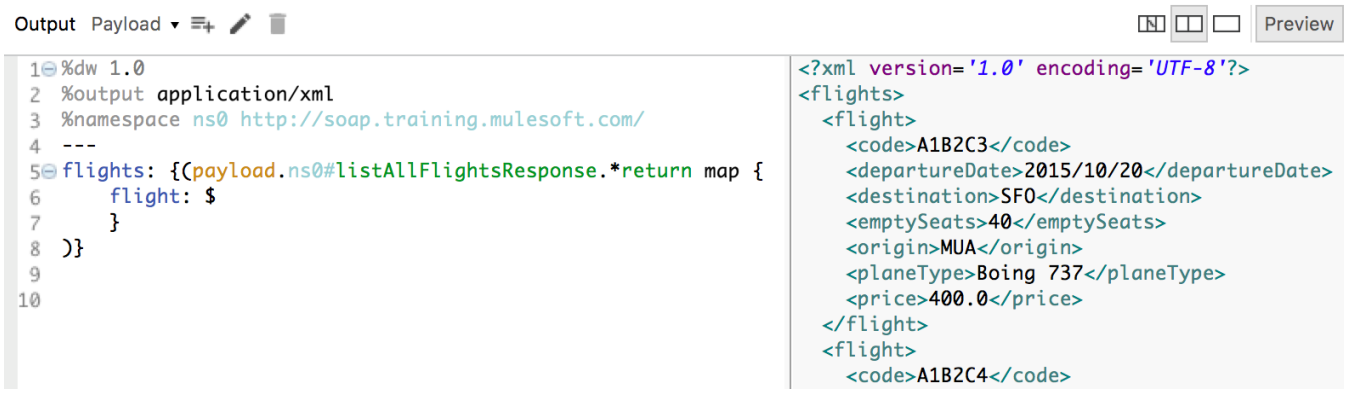


Change the return structure of the XML

21. Change the DataWeave expression to loop over the listAllFlightsResponse element of the payload; you should see the flights are now correctly transformed.

22. Change the DataWeave expression to loop over the payload.listAllFlightsResponse.return element.

23. Change the DataWeave expression use * to loop over the XML repeated return elements.



24. Change the DataWeave expression to return flight elements with dest and price elements that map to the corresponding destination and price input values.

Note: If you want to test the application with Postman, be sure to change the content-type header to application/XML and replace the request body with XML from the flights-example.xml file.

Change the transformation to return Java

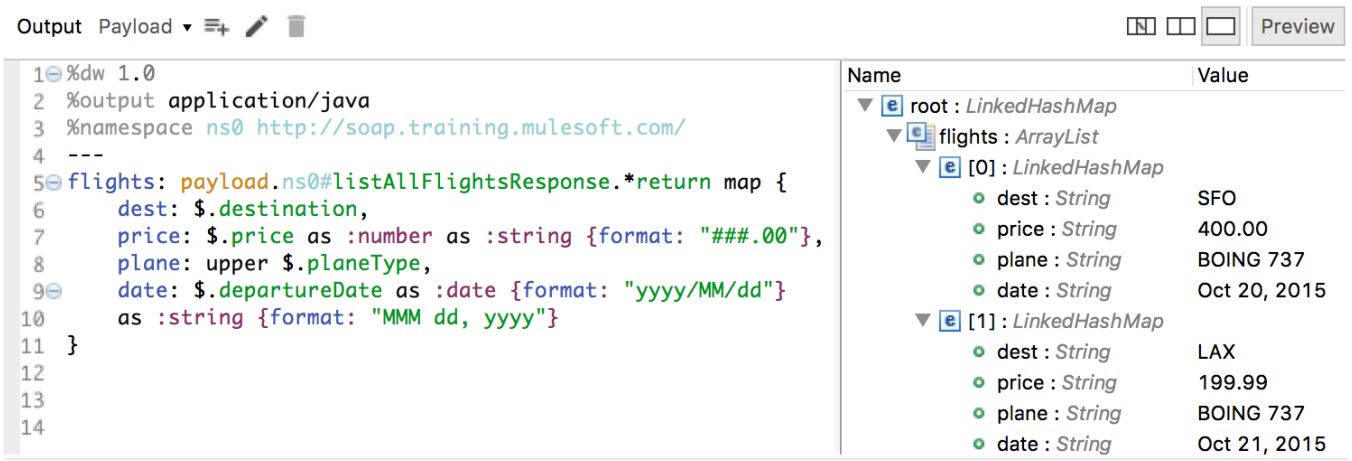
25. Change the output type from application/xml to application/java.
26. Look at the preview; you should see flights is a LinkedHasMap with one flight property.

27. In the DataWeave expression, remove the {(}) around the map expression.
28. Change the DataWeave expression to remove the flight property.
29. Look at the preview; you should see flights is an ArryaList with five LinkedHashMaps.

Walkthrough 10-5: Coerce and format strings, numbers, and dates

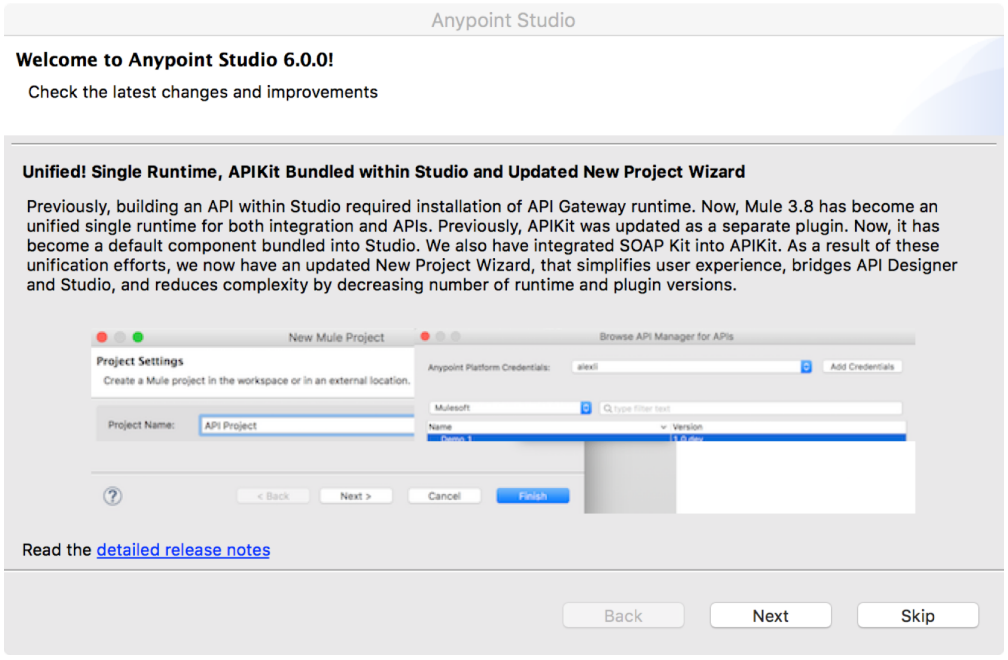
In this walkthrough, you continue to work with the XML flights data posted to the flow. You will:

- Explore the DataWeave documentation.
- Coerce data types.
- Format strings, numbers, and dates.

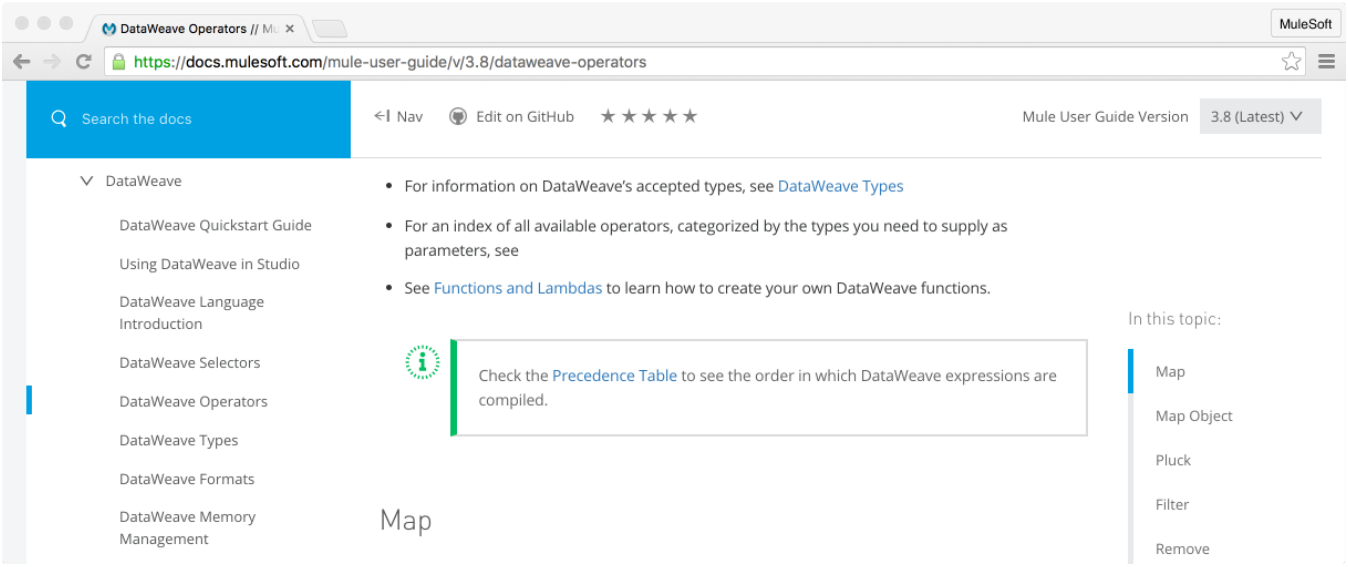


Browse DataWeave reference documentation

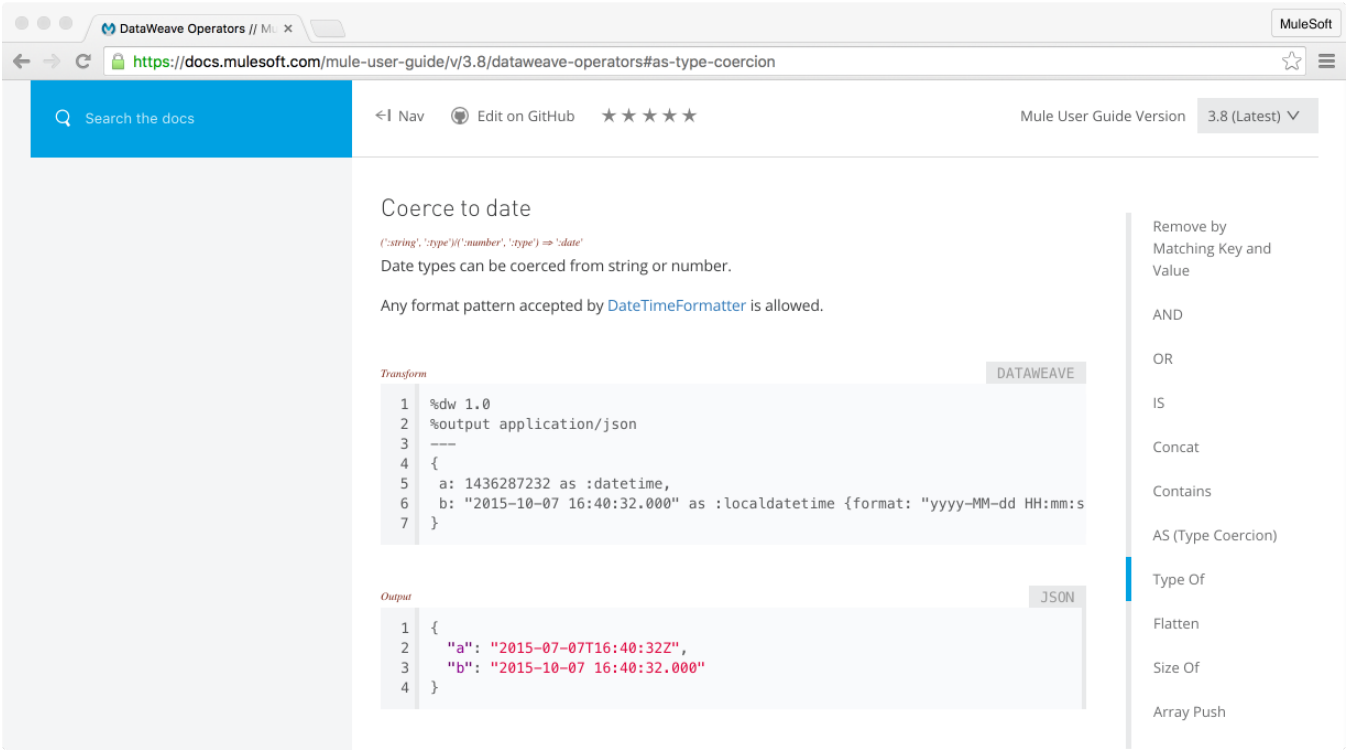
1. In the main menu bar in Anypoint Studio, select Help > What's new in Anypoint Studio?
2. In the Anypoint Studio dialog box, click the detailed release notes link; a new web browser window should open.



3. In the left-side navigation, navigate to Mule User Guide > Developing > Transformers > DataWeave > DataWeave Operators.

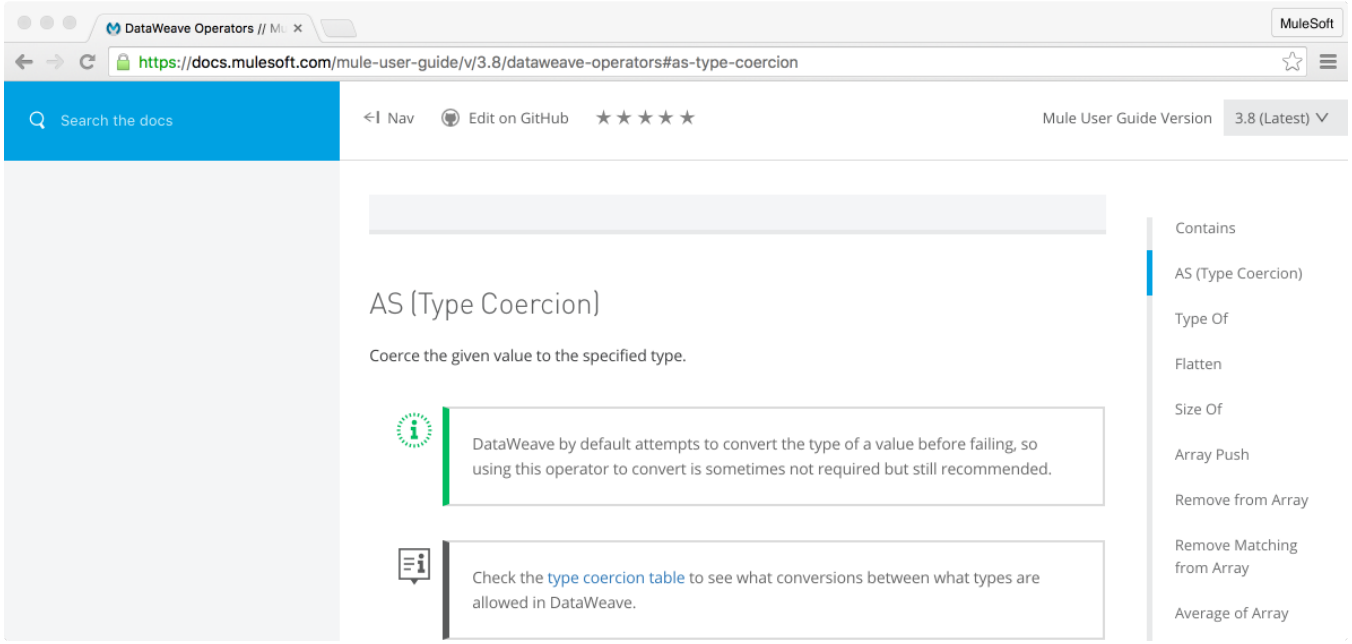


4. Locate the In this topic navigation on the right-side of the page.
5. Scroll the right-side navigation.
6. Click the AS (Type Coercion) link.
7. Scroll the page and browse the examples and documentation.

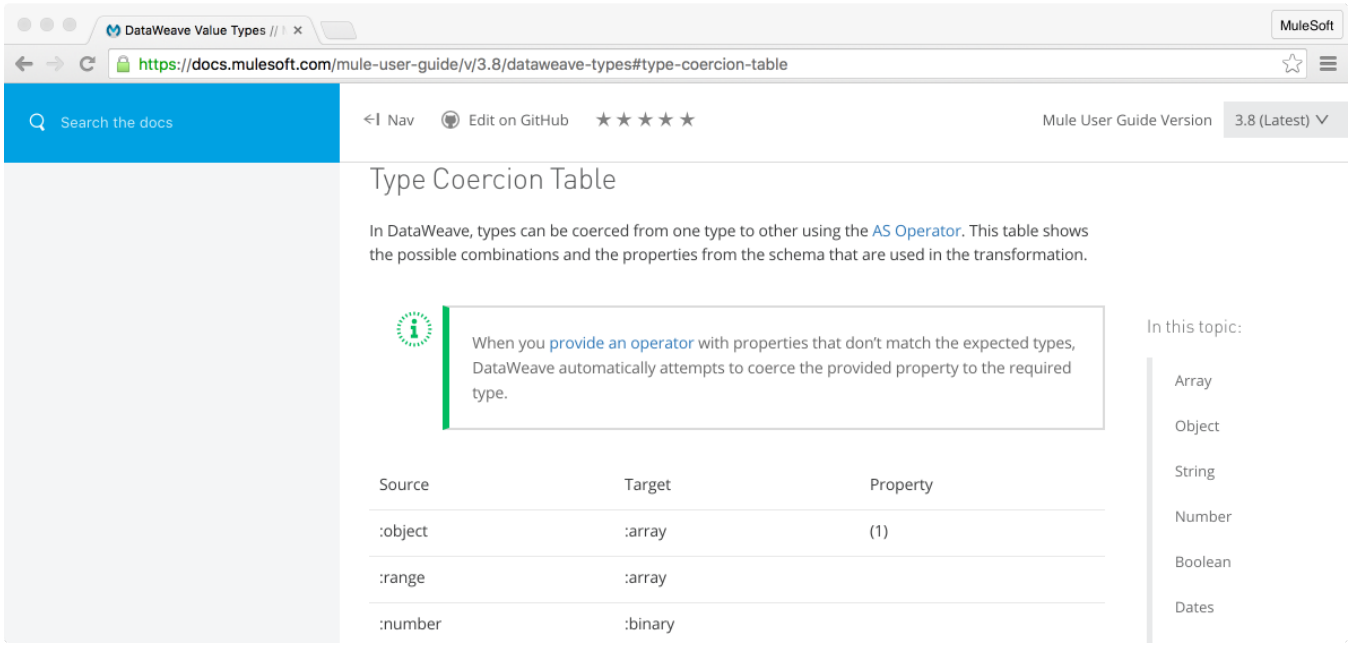


8. In the right-side navigation, click the AS (Type Coercion) link again.

9. In the page, locate and click the type coercion table link.



10. Browse the type coercion table.



Format a string

11. Return to Anypoint Studio.
12. In the Anypoint Studio dialog box, click the Skip button.

13. Navigate to the Transform Message properties view for the transformation in postMultipleFlightsFlow.
14. Change the DataWeave expression to return objects that also have a property called plane equal to the value of the input planeType values.
15. Use the upper operator to return the value in uppercase.

```
plane: upper $.planeType
```

16. Look at the preview; you should see the uppercase plane fields.

Coerce a string to a number

17. In the preview, look at the data type of the prices; you should see that they are strings.
18. Change the DataWeave expression to use the as operator to return the prices as numbers.

```
price: $.price as :number,
```

19. Look at the preview; you should see the prices are now either Integer or Double objects.

Coerce a string to a specific type of number object

20. Change the DataWeave expression to use the class metadata key to coerce the prices to java.lang.Double objects.

```
price: $.price as :number {class:"java.lang.Double"},
```

21. Look at the preview; you should see the prices are now all Double objects.

Format a number

22. Use the format schema property in the DataWeave expression to format the prices to zero decimal places.
23. Remove the coercion to a java.lang.Double.

```
price: $.price as :number as :string {format: "###"},
```

24. Look at the preview; the prices should now be formatted.

25. Change the DataWeave expression to format the prices to two decimal places.

```
price: $.price as :number as :string {format: "###.##"},
```

26. Look at the preview; the prices should be formatted differently.

27. Change the DataWeave expression to format the prices to two minimal decimal places.

```
price: $.price as :number as :string {format: "###.00"},
```

28. Look at the preview; all the prices should be formatted to two decimal places.

Note: If you are not a Java programmer, you may want to look at the Java documentation for the DecimalFormatter class to review documentation on patterns.
https://docs.oracle.com/javase/8/docs/api/java/text/DecimalFormat.html

Coerce a string to a date

29. Add a date field to the return object.

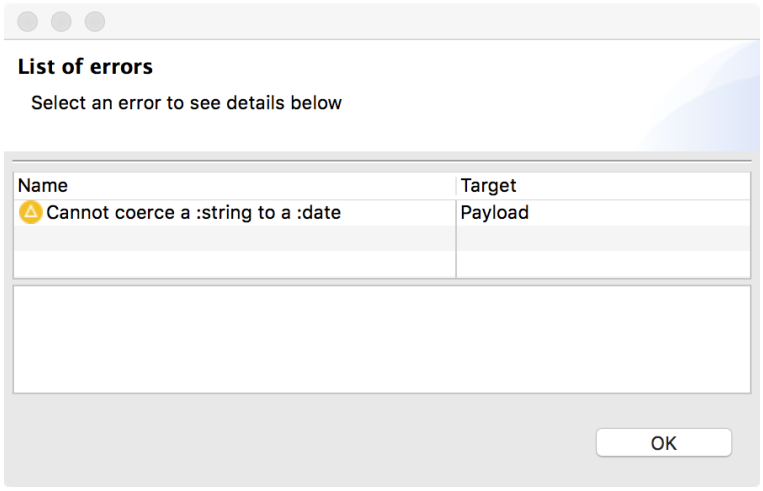
```
date: $.departureDate
```

30. Look at the preview; you should see the date property is a String.

31. Change the DataWeave expression to use the as operator to convert departureDate to a date object; you should get an exception.

```
date: $.departureDate as :date
```

32. Look at the exception.

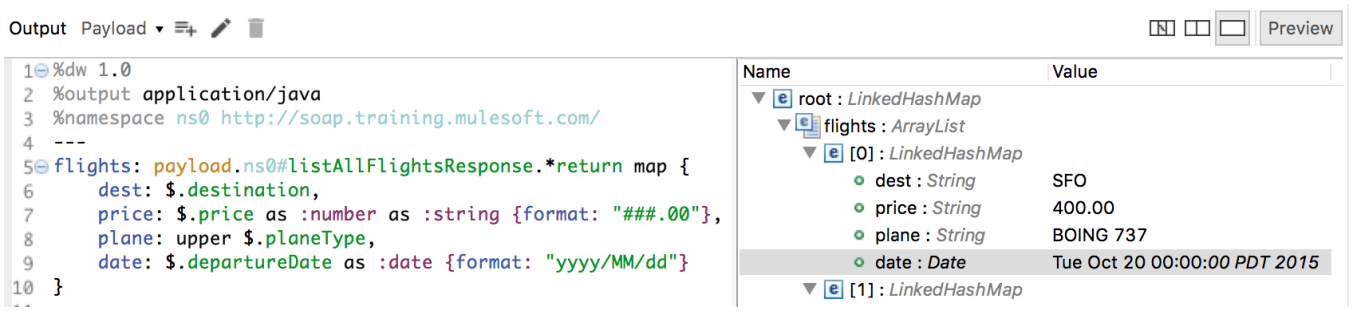


33. Look at the format of the dates in the preview.
34. Change the DataWeave expression to use the format schema property to specify the pattern of the input date strings.

```
date: $.departureDate as :date {format: "yyyy/MM/dd"}
```

Note: If you are not a Java programmer, you may want to look at the Java documentation for the DateTimeFormatter class to review documentation on pattern letters.
https://docs.oracle.com/javase/8/docs/api/java/time/format/DateTimeFormatter.html

35. Look at the preview; you should see date is now a Date object.



Format a date

36. Use the as operator to convert the dates to strings, which can then be formatted.

```
date: $.departureDate as :date {format: "yyyy/MM/dd"} as :string
```

37. Look at the preview section; the date is again a String – but with a different format.

38. Use the format schema property with any pattern letters and characters to format the date strings.

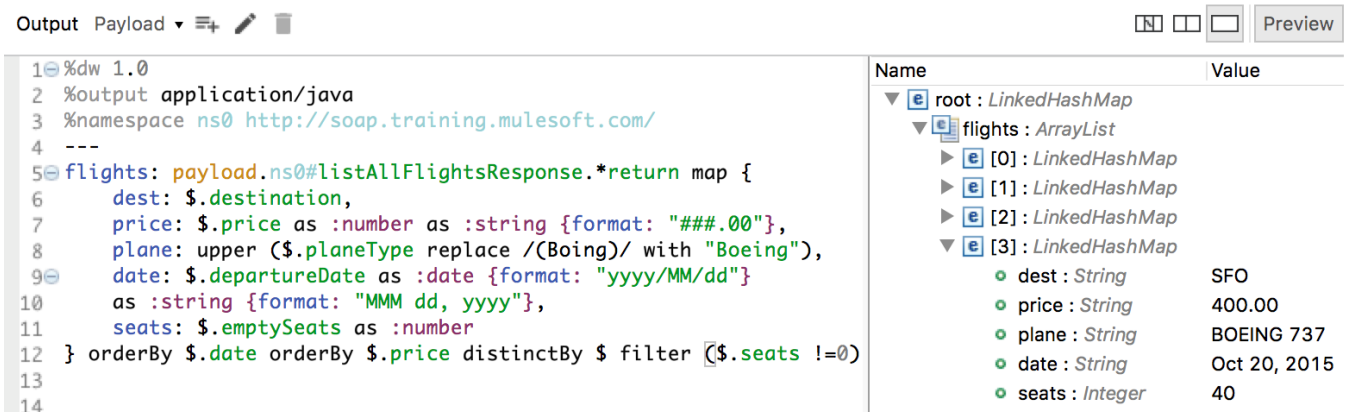
```
date: $.departureDate as :date {format: "yyyy/MM/dd"} as :string
{format: "MMM dd, yyyy"}
```

39. Look at the preview; the dates should now be formatted according to the pattern.

Walkthrough 10-6: Use DataWeave operators

In this walkthrough, you continue to work with the flights JSON posted to the flow. You will:

- Replace data values using pattern matching.
- Order data, filter data, and remove duplicate data.



Replace data values

1. Return to the Transform Message properties view for the transformation in postMultipleFlightsFlow.
2. Use the replace operator in the DataWeave expression to replace the string Boing with Boeing.

```
plane: upper $.planeType replace /(BOING)/ with "BOEING",
```

3. Look at the preview section; Boeing should not be spelled correctly.
4. In the DataWeave expression, place parentheses around the replace operator expression.
5. Look at the preview section; Boeing should now be spelled correctly.

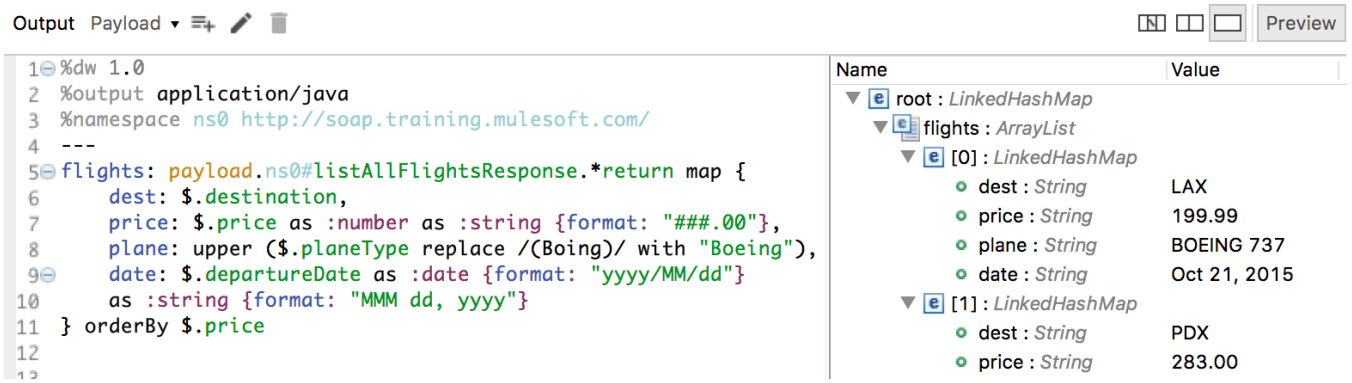
Order data

6. In the preview section, look at the flight prices; the flights should not be ordered by price.

7. Change the DataWeave expression to use the orderBy operator to order the objects by price.

```
payload.listAllFlightsResponse.*return map {
      …
} orderBy $.price
```

8. Look at the preview; the flights should now be ordered by price.



9. Look at the PDX flights; they should not be ordered by date.
10. Change the DataWeave expression to first sort by date and then by price.

```
payload.listAllFlightsResponse.*return map {
      …
} orderBy $.date orderBy $.price
```

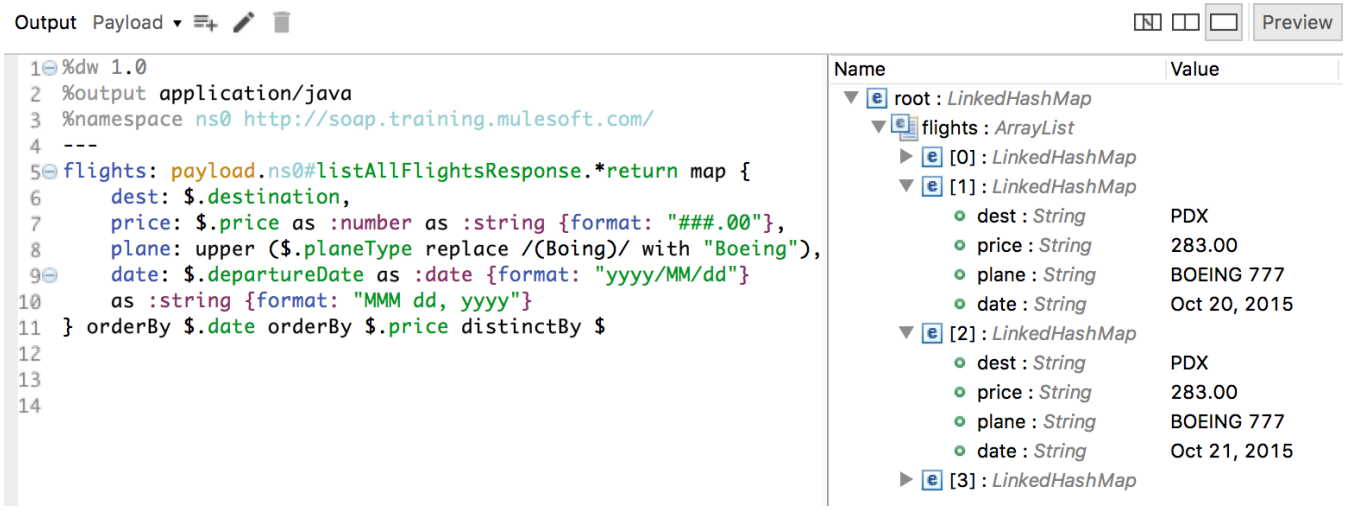
11. Look at the preview; the flights should now be ordered by price and flights of the same price should be sorted by date.

Remove duplicate data

12. Use the distinctBy operator in the DataWeave expression to remove any duplicate objects.

```
payload.listAllFlightsResponse.*return map {
      …
} orderBy $.date orderBy $.price distinctBy $
```

13. Look at the preview; you should now get only four flights instead of five.



14. Add an seats field that is equal to the emptySeats field and coerce it to a number.

```
seats: $.emptySeats as :number
```

15. Look at the preview section; you should get five flights again.

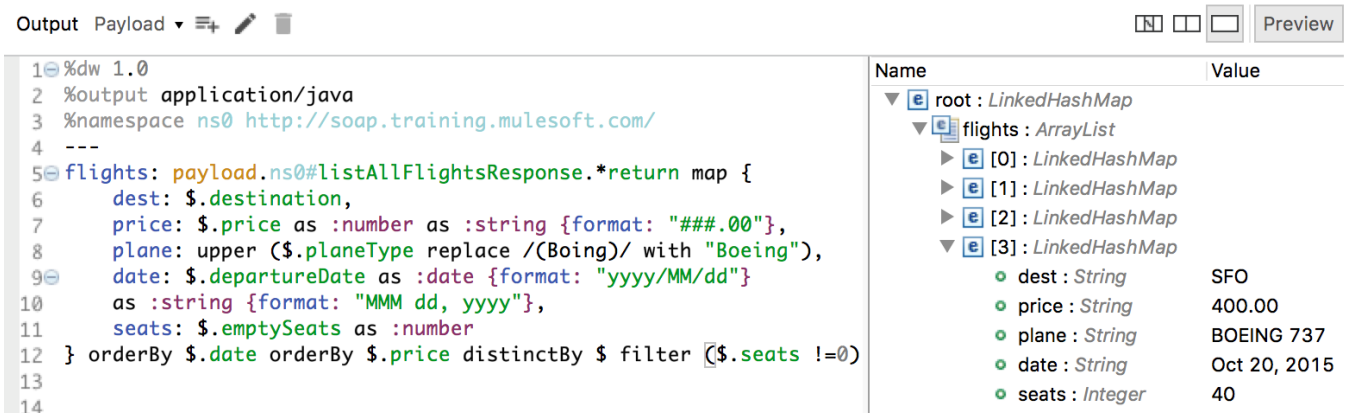
Filter data

16. In the preview section, look at the values of the seats properties; ou should see some are equal to zero.

17. Add a filter to the DataWeave expression that removes any objects that have availableSeats equal to 0.

```
payload.listAllFlightsResponse.*return map {
      …
} orderBy $.departureDate orderBy $.price distinctBy $
     filter ($.seats !=0)
```

18. Look at the preview; you should no longer get the flight that had no available seats.



Walkthrough 10-7: Define and use custom data types

In this walkthrough, you continue to work with the flight JSON posted to the flow. You will:

- Define and use custom data types.
- Transform objects to POJOs.

Define a custom data type

1. Return to the Transform Message properties view for the transformation in postMultipleFlightsFlow.
2. Select and cut the string formatting expression for the prices.
3. In the header section of the transform section, define a custom data type called currency.
4. Set it equal to the value you copied.
5. Remove the as operator.

```
%type currency = :string {format: "###.00"}
```

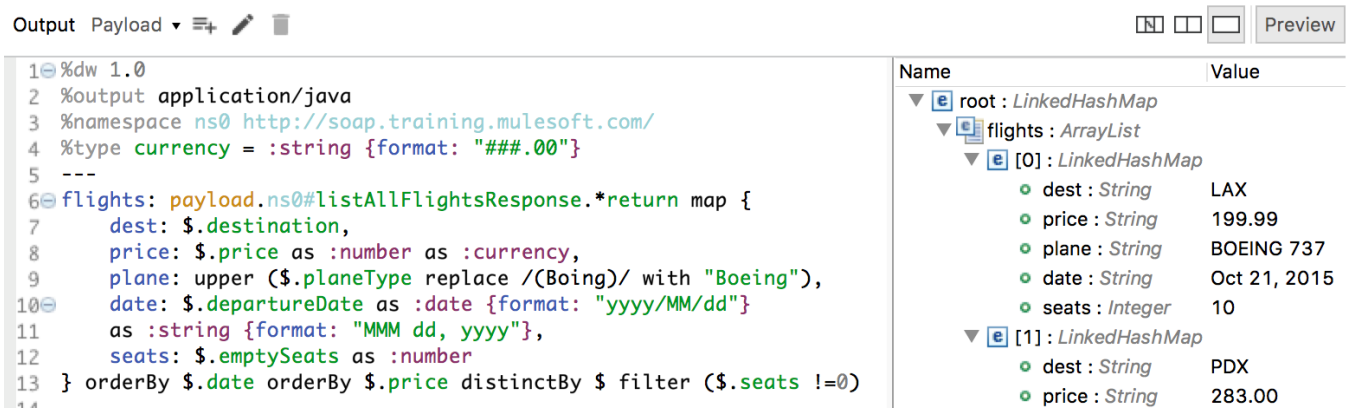
```
%dw 1.0
%output application/java
%namespace ns0 http://soap.training.mulesoft.com/
%type currency = :string {format: "###.00"}
---
```

Use a custom data type

6. In the DataWeave expression, set the price to be of type currency.

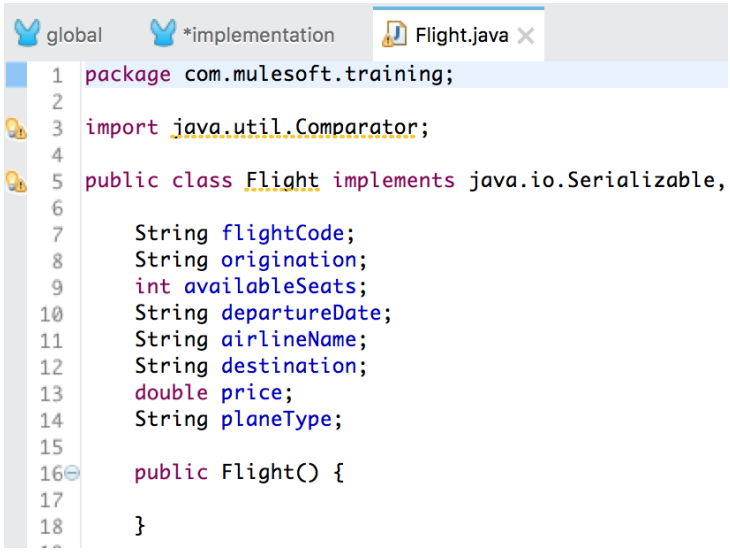
```
price: $.price as :number as :currency,
```

7. Look at the preview; the prices should still be formatted as strings to two decimal places.



Transform objects to POJOs

8. Open the Flight.java class in the project's src/main/java folder and look at the names of the properties.



9. Return to postMultipleFlightsFlow in implementation.xml.
10. In the header section of the transform section, define a custom data type called flight that is of type com.mulesoft.traning.Flight.

```
%type flight = :object {class: "com.mulesoft.training.Flight"}
```

```
%dw 1.0
%output application/java
%namespace ns0 http://soap.training.mulesoft.com/
%type currency = :string {format: "###.00"}
%type flight = :object {class: "com.mulesoft.training.Flight"}
---
```

11. In the DataWeave expression, set the map objects to be of type flight.

```
payload.listAllFlightsResponse.*return map {
      …
} orderBy $.date orderBy $.price distinctBy $ filter ($.seats !=0)
as :flight
```

12. Change the name of the dest key to destination.

```
destination: $.destination,
```

13. Change the other keys to match the names of the Flight class properties:

    - plane to planeType
    - date to departureDate
    - seats to availableSeats

14. Change the operators to use the new property names.

```
 payload.listAllFlightsResponse.*return map {
      …
} orderBy $.departureDate orderBy $.price distinctBy $ filter
($.availableSeats !=0) as :flight
```

```
%dw 1.0
%output application/java
%namespace ns0 http://soap.training.mulesoft.com/
%type currency = :string {format: "###.00"}
%type flight = :object {class: "com.mulesoft.training.Flight"}
---
flights: payload.ns0#listAllFlightsResponse.*return map {
    destination: $.destination,
    price: $.price as :number as :currency,
    planeType: upper ($.planeType replace /(Boing)/ with "Boeing"),
    departureDate: $.departureDate as :date {format: "yyyy/MM/dd"}
    as :string {format: "MMM dd, yyyy"},
    availableSeats: $.emptySeats as :number
} orderBy $.departureDate orderBy $.price distinctBy $
filter ($.availableSeats !=0) as :flight
```

15. Look at the preview; you should now get an ArrayList of Flight objects.
16. If you get an exception instead; delete the operators after the map.

```
payload.listAllFlightsResponse.*return map {
      …
} as :flight
```

Note: There is a bug in Mule 3.8.0 with custom objects.

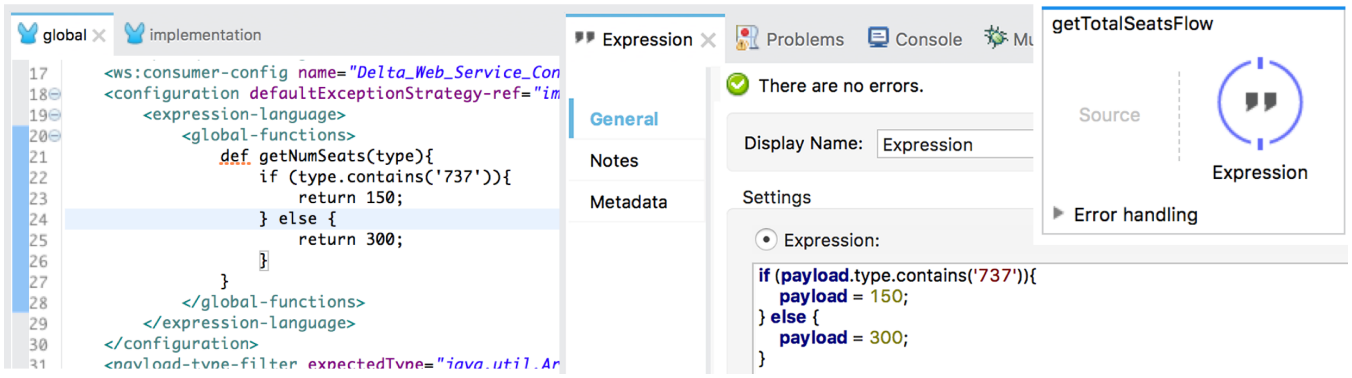
17. Look at the preview; you should now get an ArrayList of Flight objects.

Walkthrough 10-8: Call MEL functions and other flows

In this walkthrough, you continue to work with the DataWeave transformation in getMultipleFlightsFlow. You will:

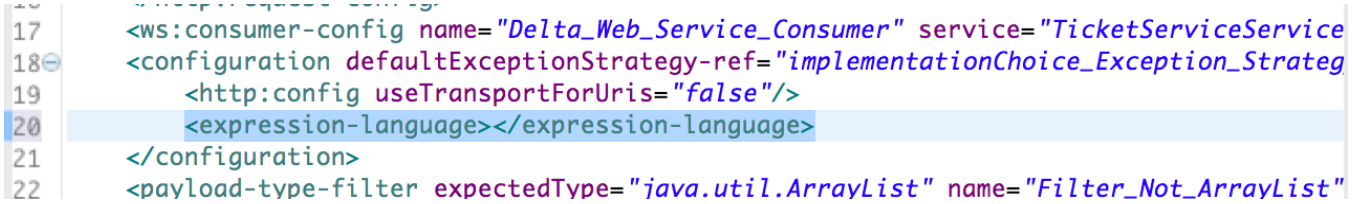
- Define a global MEL function in global.xml.
- Call a global MEL function in a DataWeave expression.
- Call a flow in a DataWeave expression.

```
//totalSeats: getNumSeats($.planeType)
totalSeats: lookup("getTotalSeatsFlow",{type: $.planeType})
```



Define a global MEL function in global.xml

1. Return to global.xml and switch to the Configuration XML view.
2. Locate the configuration element.
3. On a new line between the configuration tags, type <e and the press Ctrl+Space for autocompletion; an expression-language tag set should be added.



4. Place the expression-language tags on different lines and place the cursor between them.
5. Type <g, press Ctrl+Space, and select global-functions; a global-functions tag set should be added.

6. Place the global-functions tags on different lines and place the cursor between them.

```
18    <configuration defaultExceptionStrategy-ref="implementationChoice_Exception_
19        <http:config useTransportForUris="false"/>
20        <expression-language>
21            <global-functions>
22
23            </global-functions>
24        </expression-language>
25    </configuration>
```

7. Use the def keyword to define a function called getNumSeats with an argument called type.

```
def getNumSeats(type){
}
```

```
18    <configuration defaultExceptionStrategy-ref="implemen
19        <http:config useTransportForUris="false"/>
20        <expression-language>
21            <global-functions>
22                def getNumSeats(type){
23
24                }
25            </global-functions>
26        </expression-language>
27    </configuration>
```

8. Inside the function, add an if/else block that checks to see if type contains the string 737.

```
if (type.contains('737')){
}
```

9. If the expression is true, return 150 and if false, return 300.

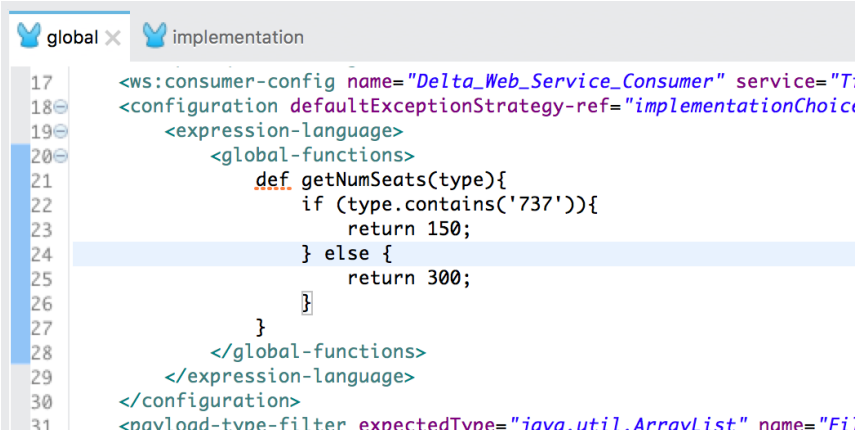
```
if (type.contains('737')){
  return 150;
} else {
  return 300;
}
```

10. Save the file.
11. Run the project; the application should fail to deploy.
12. Locate the exception in the console.

```
ERROR 2016-06-07 19:53:03,383 [main] org.mule.module.launcher.application.DefaultMuleApplication: null
org.xml.sax.SAXParseException: cvc-complex-type.2.4.a: Invalid content was found starting with element
'expression-language'. One of '{"http://www.mulesoft.org/schema/mule/core":abstract-configuration-exten
sion}' is expected.
        at org.apache.xerces.util.ErrorHandlerWrapper.createSAXParseException(Unknown Source) ~[?:?]
        at org.apache.xerces.util.ErrorHandlerWrapper.error(Unknown Source) ~[?:?]
```

13. Return to global.xml and delete the http:config tag inside the configuration tag set.

```
<http:config useTransportForUris="false"/>
```



14. Run the project; the application should deploy.
15. Stop the project.

Call a global MEL function from DataWeave

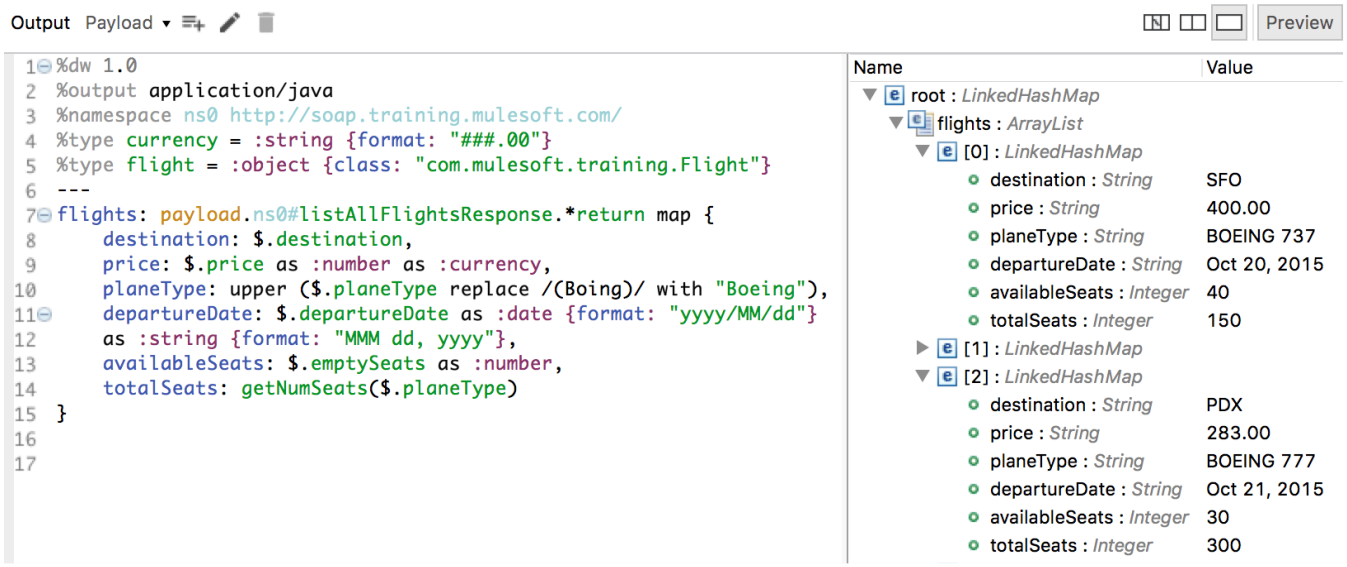
16. Return to postMultipleFlightsFlow in implementation.xml.
17. In the Transform Message properties view, add a field called totalSeats inside the transformation function.
18. Set totalSeats equal to the return value from the getNumSeats() function.
19. Pass to getNumSeats(), the planeType.

```
totalSeats: getNumSeats($.planeType)
```

20. Look at the preview; you should see the Flight objects now have a totalSeats property equal to 150 or 300.
21. If you get an exception, remove the coercion to a Flight object.

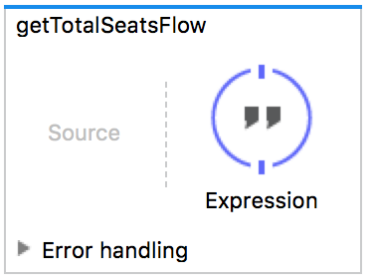
    Note: There is a bug in Mule 3.8.0 with custom objects.

22. Look at the preview; you should see the Flight objects now have a totalSeats property equal to 150 or 300.



Create a lookup flow

23. Return to implementation.xml.
24. Drag an Expression component (not a filter or a transformer!) from the Mule Palette and drop it at the bottom of the canvas to create a new flow.
25. Change the name of the flow to getTotalSeatsFlow.

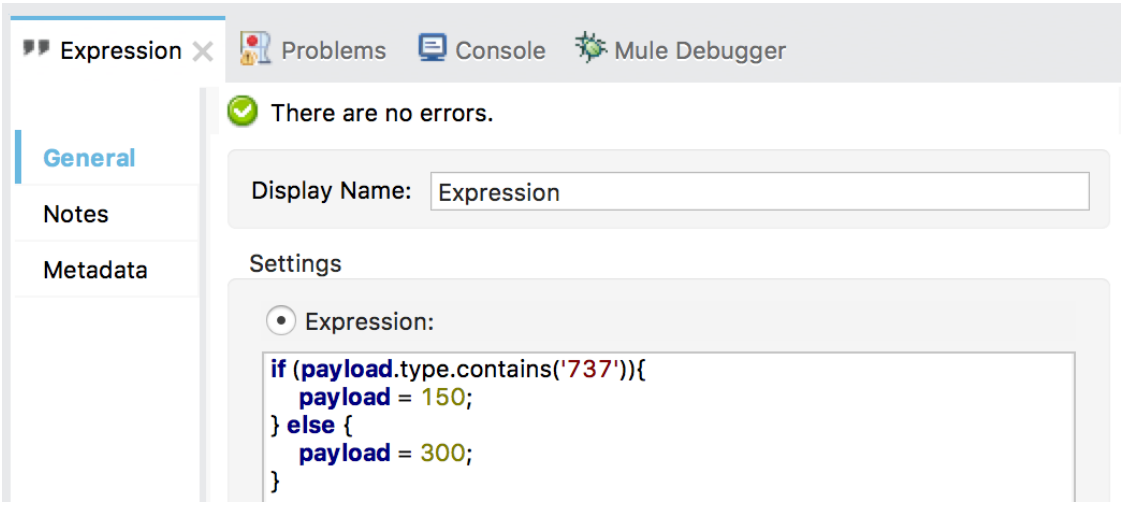


26. Return to global.xml and copy the if/else block.
27. Return to implementation.xml.
28. In the Expression properties view, paste the if/else block you copied into the expression text area.
29. In the expression, change type to payload.type.

```
if (payload.type.contains('737')){
```

30. Change the two return statements to set the payload with the values instead.

```
if (payload.type.contains('737')){
      payload = 150;
} else {
      payload = 300;
}
```



Call a flow from a DataWeave expression

31. Return to the Transform Message properties view of the component in postMultipleFlightsFlow.
32. Comment out the existing totalSeats assignment by placing // in front of it.

```
//totalSeats: getNumSeats($.planeType)
```

33. Add another property called totalSeats inside the transformation function.
34. Set totalSeats equal to the return value from the DataWeave lookup() function.

```
totalSeats: lookup()
```

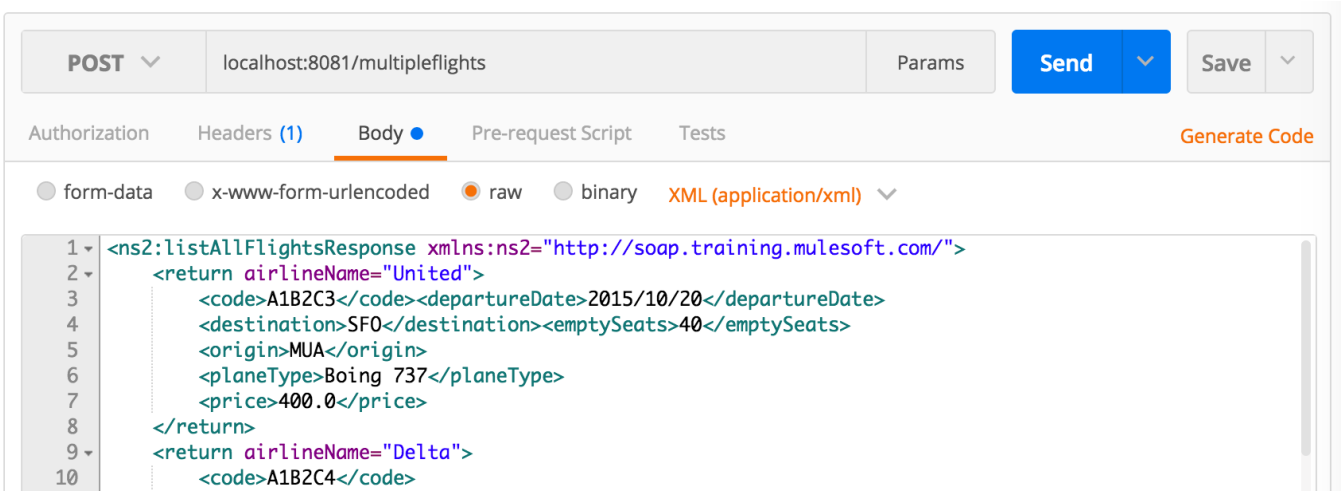
35. Pass to lookup() an argument that is equal to the name of the flow to call: "getTotalSeatsFlow".
36. Pass an object to lookup() as the second argument.
37. Give the object a field called plane (to match what the flow is expecting) and set it equal to the value of the planeType field.

```
totalSeats: lookup("getTotalSeatsFlow",{type: $.planeType})
```

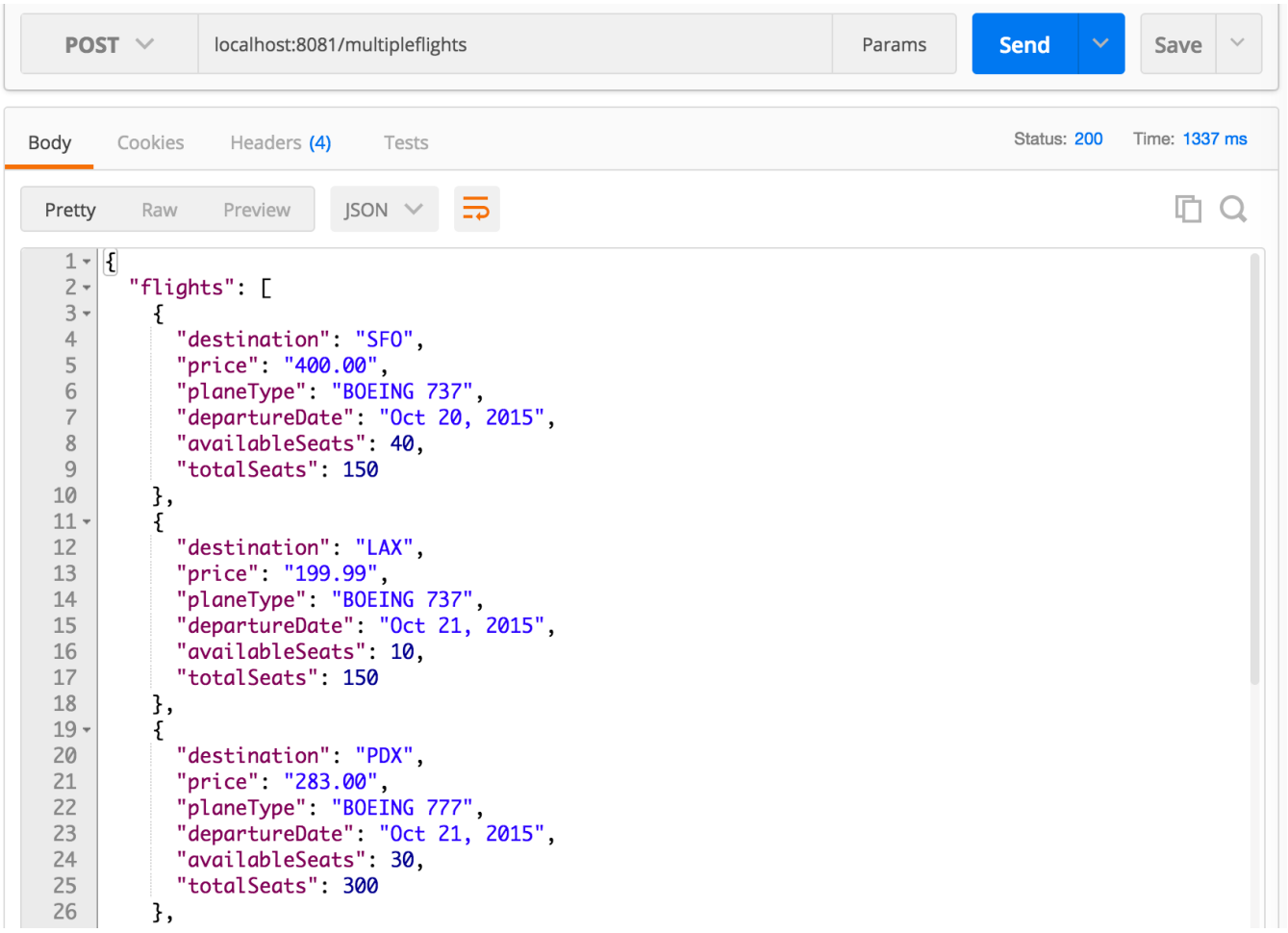
38. Look at the preview; you should not see the totalSeats because its value can only be evaluating by calling the flow at runtime.

Test the application

39. Change the output type to application/json.
40. Run the project.
41. In Postman, make sure the method is set to POST and the request URL is set to localhost:8081/multipleflights.
42. Set a Content-Type header to application/xml.
43. Set the request body to the value contained in the flights-example.xml file.



44. Send the request; you should get JSON flight data returned and each flight should have a totalSeats property equal to 150 or 300.



45. Stop the project.
46. Close the project.