Module 12: Transforming Data with DataWeave

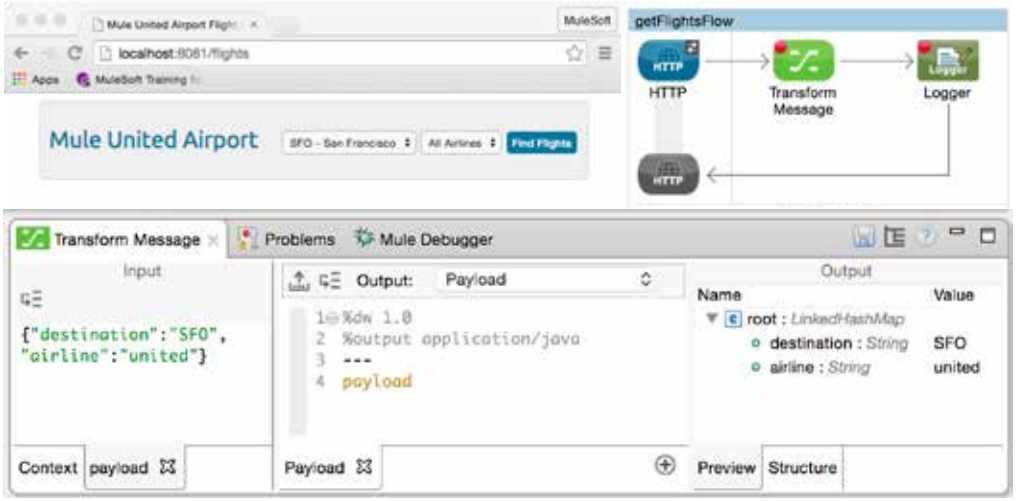
In this module, you will learn:

- To use the DataWeave Transform Message component.
- To write DataWeave expressions for basic and complex XML, JSON, and Java transformations.
- To use DataWeave with data sources that have associated metadata.
- To add custom metadata to data sources.
- To use DataWeave with CSV files.

Walkthrough 12-1: Write your first DataWeave transformation

In this walkthrough, you will work with the data posted from the HTML form. You will:

- Use the DataWeave Transform Message component.
- Add sample data and use live preview.
- Transform form data from JSON to a Java object.



Note: To complete the walkthroughs in this module, you need Anypoint Studio 5.2.1 or later and Mule runtime 3.7.1 or later.

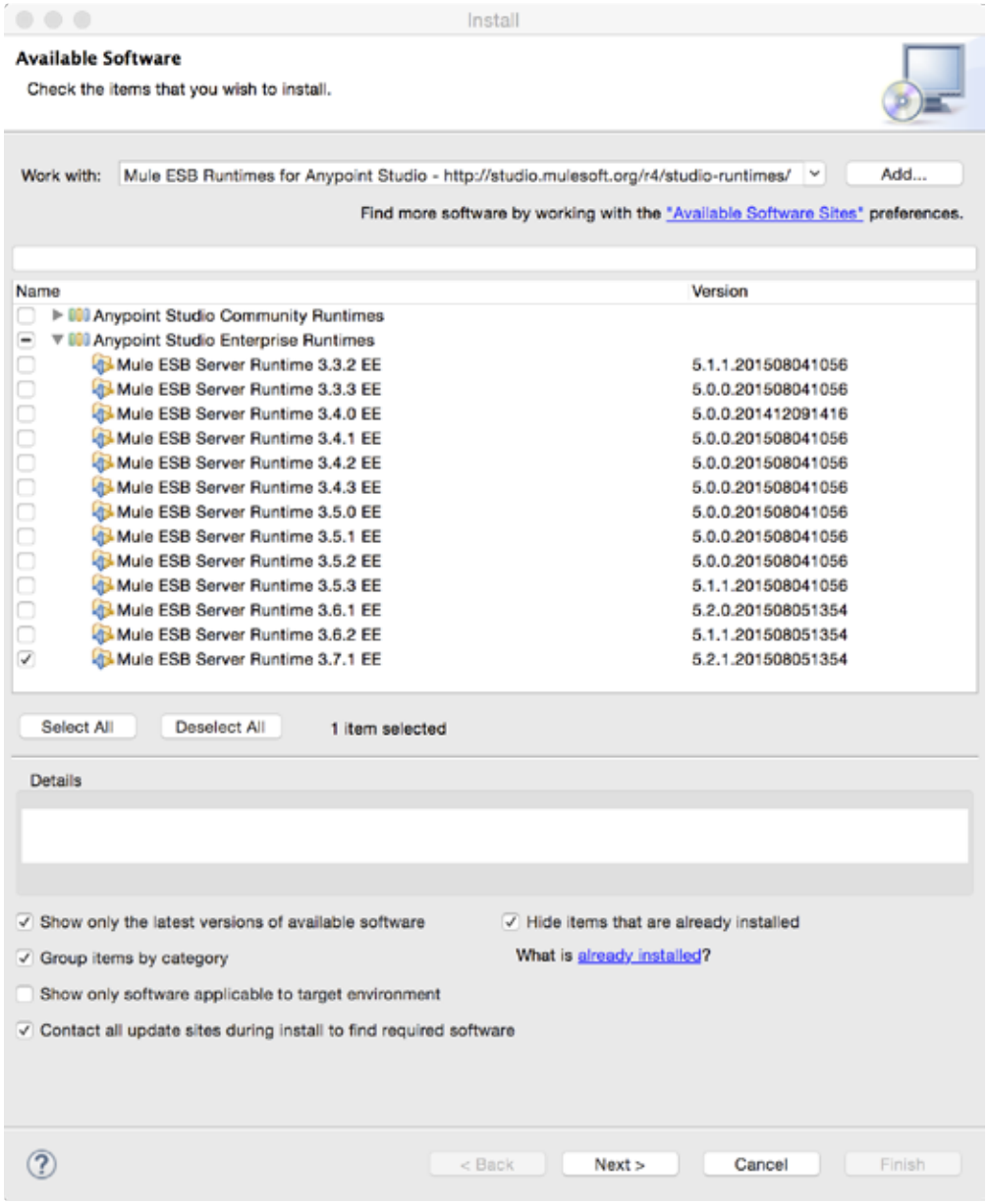
Update Anypoint Studio

1. Return to Anypoint Studio.
2. From the main menu, select Help > Check for Updates.
3. If you get an Available Updates dialog box, click Select All and click Next.
    - On the Update Details page, click Next.
    - On the Review Licenses page, select the I accept the terms radio button and click Finish.
    - Wait for the updates to be installed.
    - In the Software Updates dialog box, click yes to restart Anypoint Studio.
4. If you get an Information dialog box with a no updates found message, click OK.

Install the latest Mule Runtime

5. From the main menu, select Help > Install New Software.
6. In the work with dropdown menu in the Install dialog box, select Mule ESB Runtimes for Anypoint Studio.

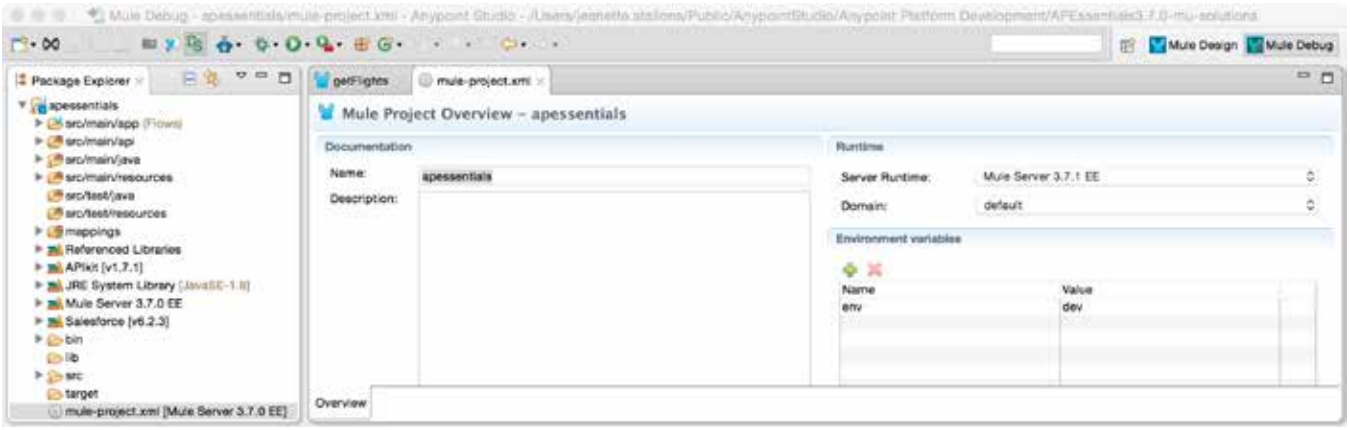
7. Click the arrow to expand Anypoint Studio Enterprise Runtimes.
8. If you see a 3.7.X runtime later than the one you are using, select it and click Next.
9. If you see a 3.7.X runtime later than the one you are using, click Cancel and skip the next several steps.



10. On the Install Details page, click Next.
11. On the Review Licenses page, select the I accept the terms radio button and click Finish.
12. Wait for the software to be installed.
13. In the Software Updates dialog box, click yes to restart Anypoint Studio.

Change the Mule runtime used by a project

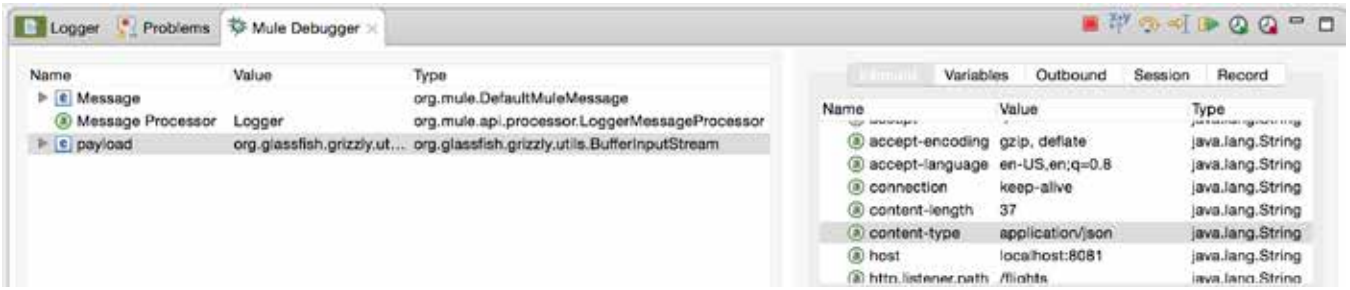
14. In the Package Explorer, locate and open mule-project.xml.
15. Set the server runtime to the the latest 3.7.X runtime.



16. In the Mule ESB version switch dialog box, click yes.

Look at the data posted from the form

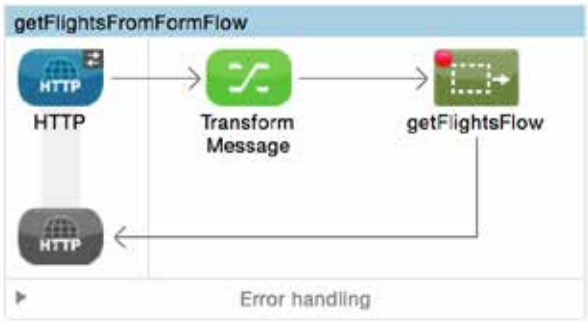
17. Return to getFlights.xml.
18. Locate getFlightsFromFormFlow.
19. Delete the JSON to Object transformer.
20. Add a breakpoint to the Flow Reference component.
21. Save the file and debug the application.
22. Make a request to http://localhost:8081/flights and click the Find Flights button.
23. Look at the Mule Debugger view; you should see the return data is a BufferInputStream.
24. Look at the inbound properties; you should see the content-type is set to application/json.



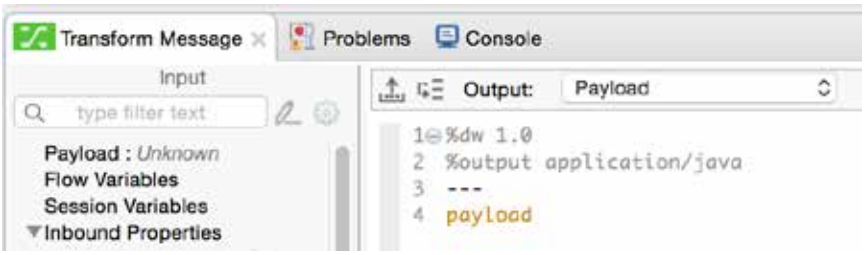
25. Stop the Mule runtime.

Add a DataWeave Transform Message component

26. Add a DataWeave Transform Message component before the Flow Reference.

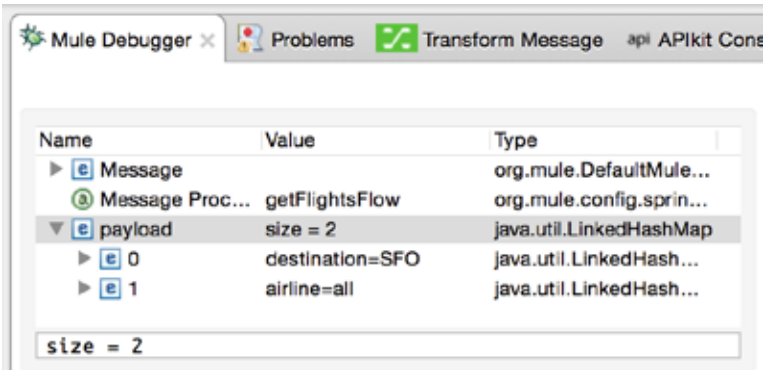


27. In the Transform Message Properties view, look at the Input, Transform, and Output sections.
28. In the Transform section, locate the drop-down menu at the top of the view that sets the output type; it should be set to Payload.
29. Beneath it, locate the directive that sets the output to application/java.
30. Delete the existing DataWeave expression (the curly braces) and set it to payload.



Debug the application

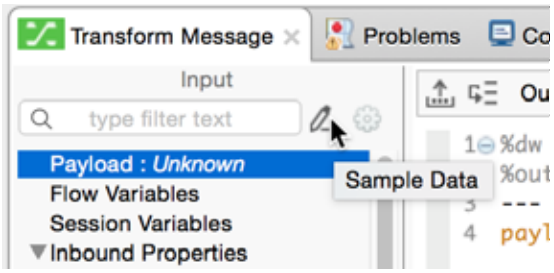
31. Add a breakpoint to the Transform Message component.
32. Save the file to redeploy the application in debug mode.
33. Make a request to http://localhost:8081/flights and submit the form again.
34. Step to the Flow Reference in getFlightsFromFormFlow; you should see the form data is now a LinkedHashMap.



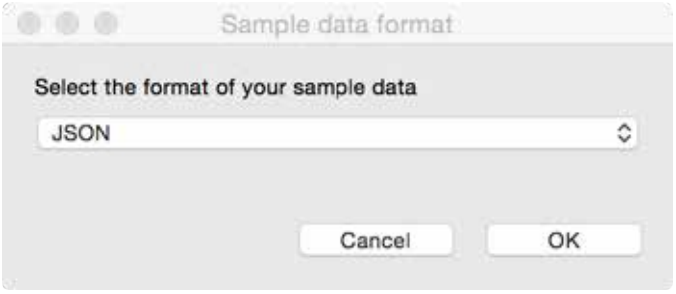
35. Click the Resume button.

Add sample data and preview

36. In the Input section of the Transform Message Properties view, click Payload: Unknown.
37. Click the Sample Data button.



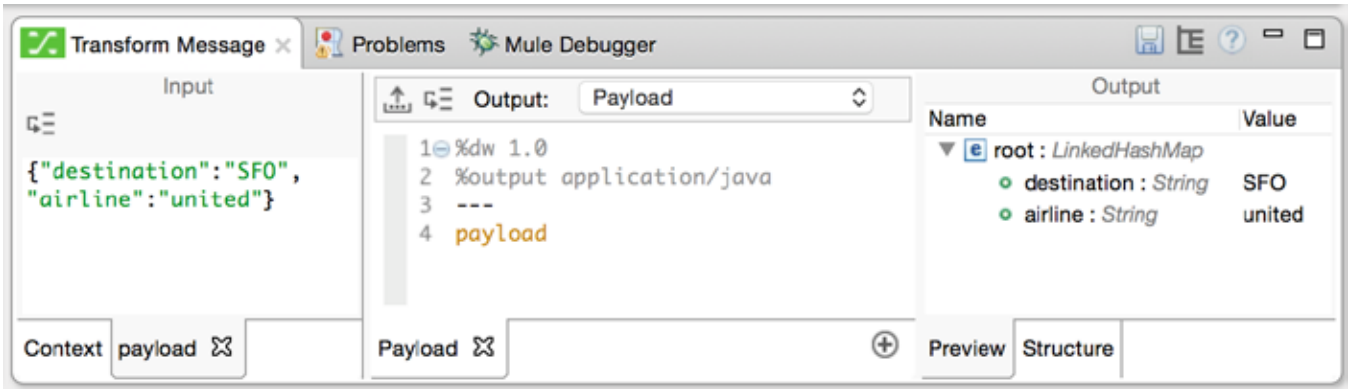
38. In the Sample data format dialog box, select JSON and click OK.



39. In the new payload window that is created in the Input section, replace the sample JSON data with {"destination":"SFO","airline":"united"}.

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

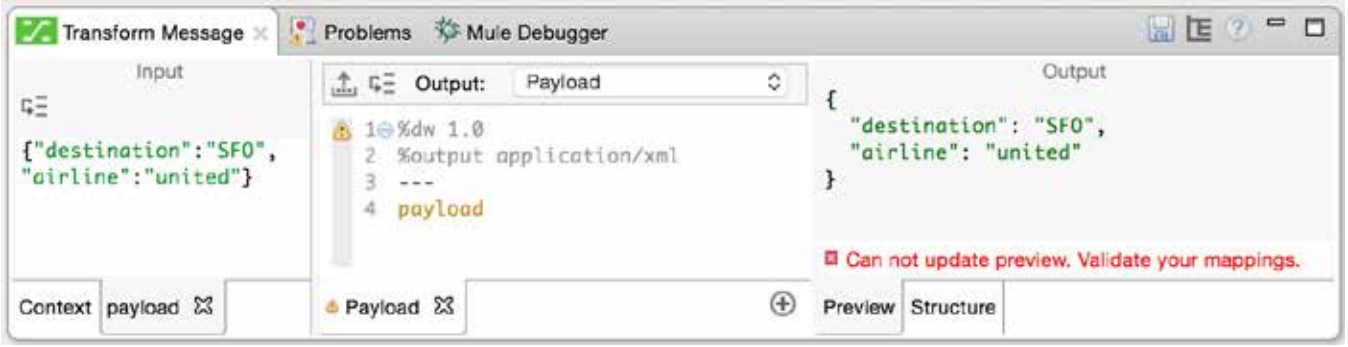
40. Look at the Preview tab in the Output section; you should see sample output of type Java.



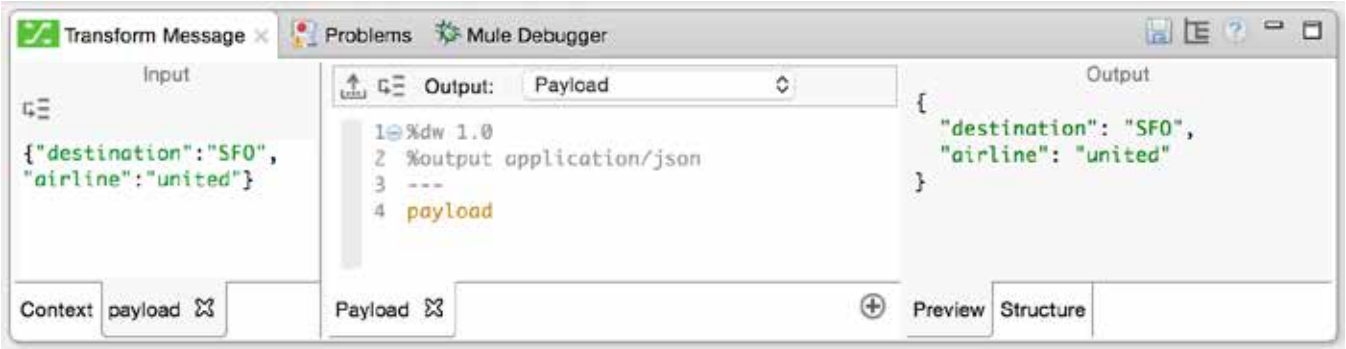
Change the output type

41. In the Transform section, change the output type from application/java to application/xml.

42. Look at the Preview tab; you should get an error.

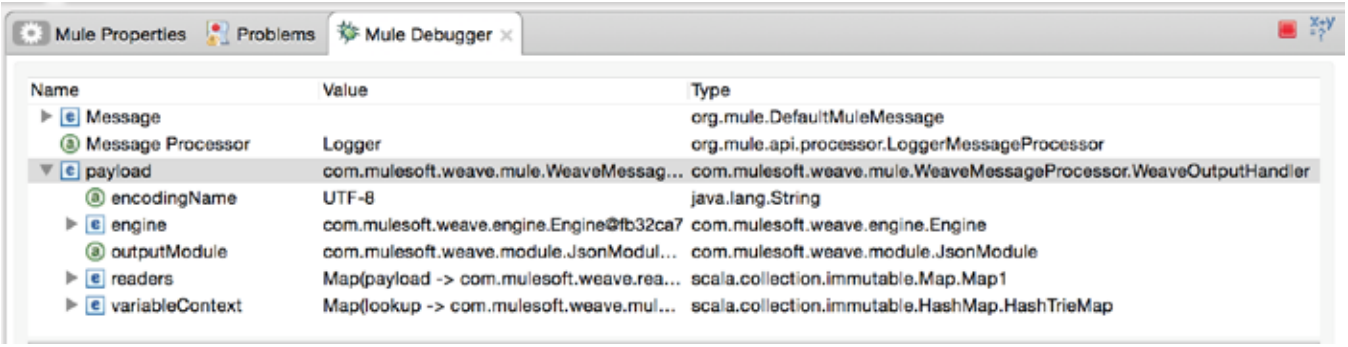


43. Change the output type to application/json; in the Preview tab, you should see sample JSON output.



Debug the application

44. Save the file to redeploy the application in debug mode.
45. Make a request to http://localhost:8081/flights and submit the form again.
46. Step to the Flow Reference in getFlightsFromFormFlow; you should see the form data is now of type WeaveOutputHandler.

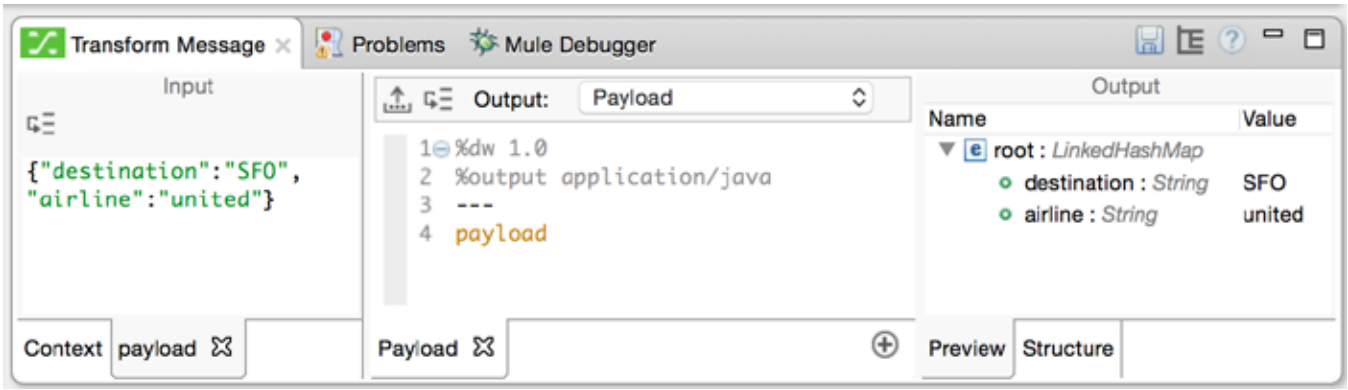


47. Stop the Mule runtime.

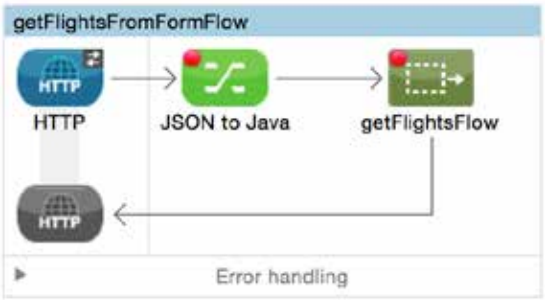
Note: If you click Resume, you will get an error in the application because the other flows are no longer getting the correct type of data.

Change the output type

48. In the Transform section, change the output type from application/json back to application/java.



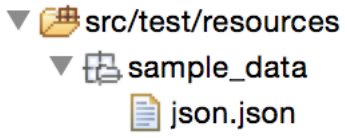
49. Click the name of the Transform Message component in the canvas and change its name to JSON to Java.



50. Save the file.

Examine the code

51. In the Package Explorer, locate the sample data in src/test/resources.

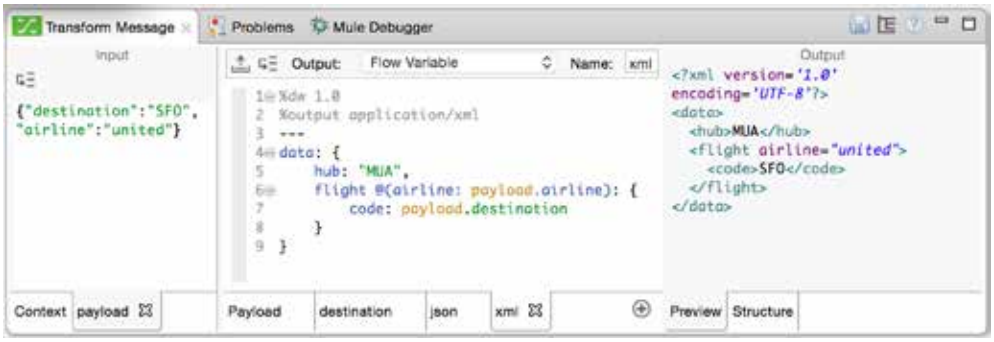


52. Switch to the Configuration XML view.
53. Locate the DataWeave code.
54. Switch back to the Message Flow view.

Walkthrough 12-2: Transform basic Java, JSON, and XML data structures

In this walkthrough, you will continue to work with the data posted from the HTML form. You will:

- Write transformations to store data in multiple outputs as different types of data in different variables.
- Create a second transformation to output data as JSON in a flow variable.
- Create a third transformation to output data as XML as a flow variable.

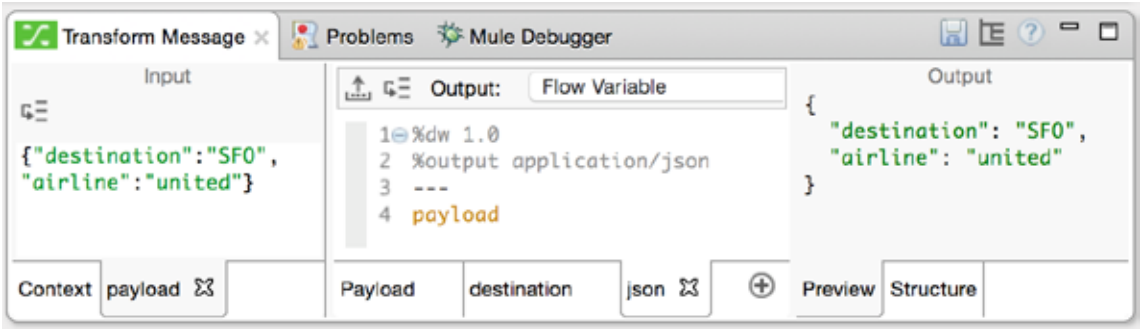


Create a second transformation to output data as JSON

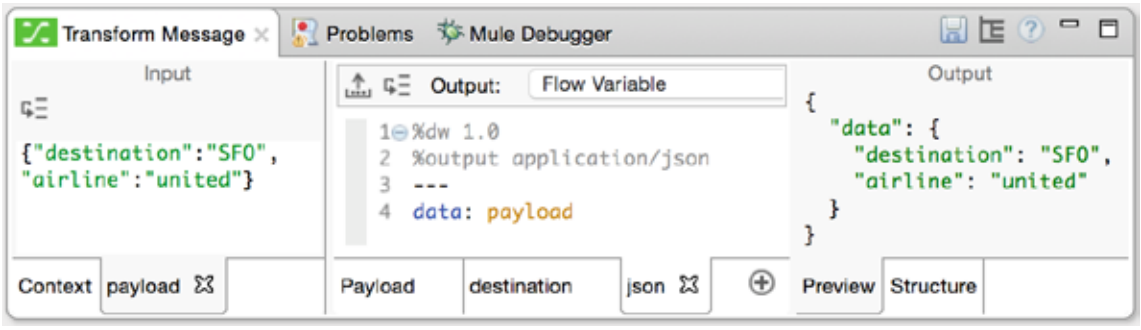
1. Return to getFlights.xml.
2. Return to the Properties view for the JSON to Java component in getFlightsFromFormFlow.
3. In the Transform section, click the Add New Transformation button in the lower-right corner.

4. In the new myVar tabbed window that appears in the Transform section, make sure the Output is set to Flow Variable.
5. Set the name to json.
6. Change the output type to application/json.
7. Set the DataWeave expression to payload.

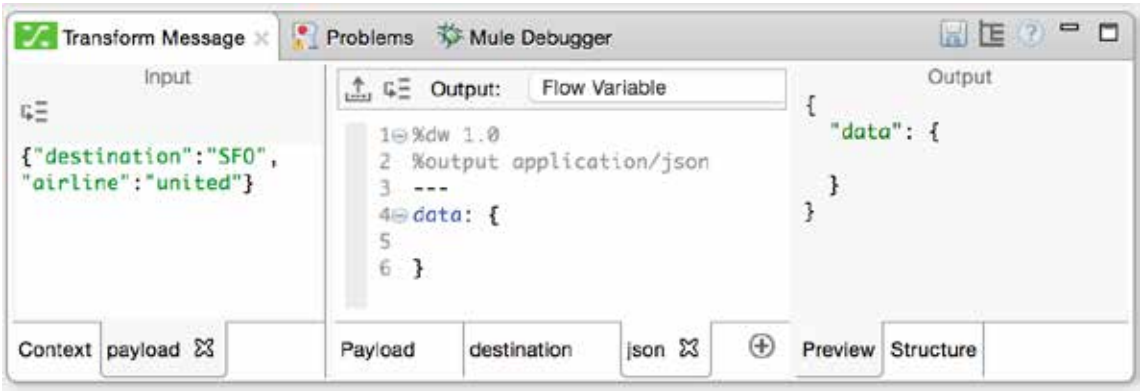
8. Look at the preview in the Output section.



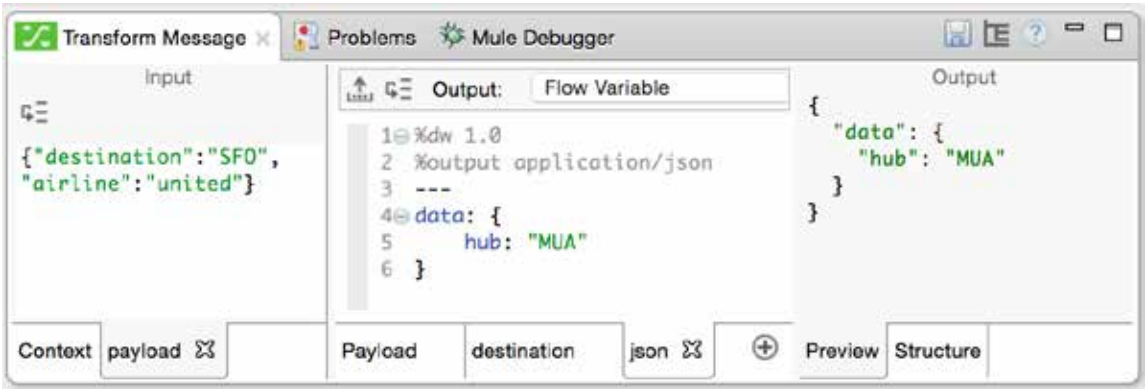
9. Change the DataWeave expression to data: payload.



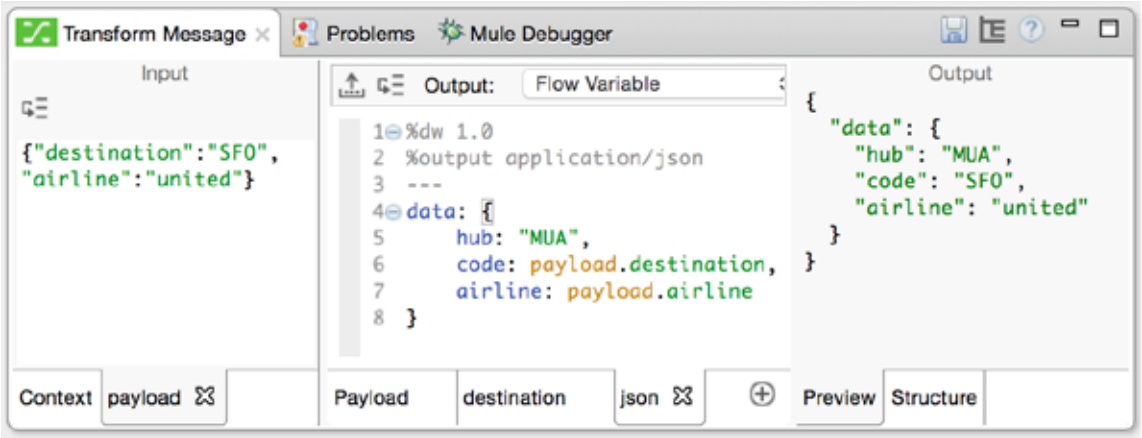
10. Change the DataWeave expression to data: {}.



11. Add a field called hub and set it to “MUA”.

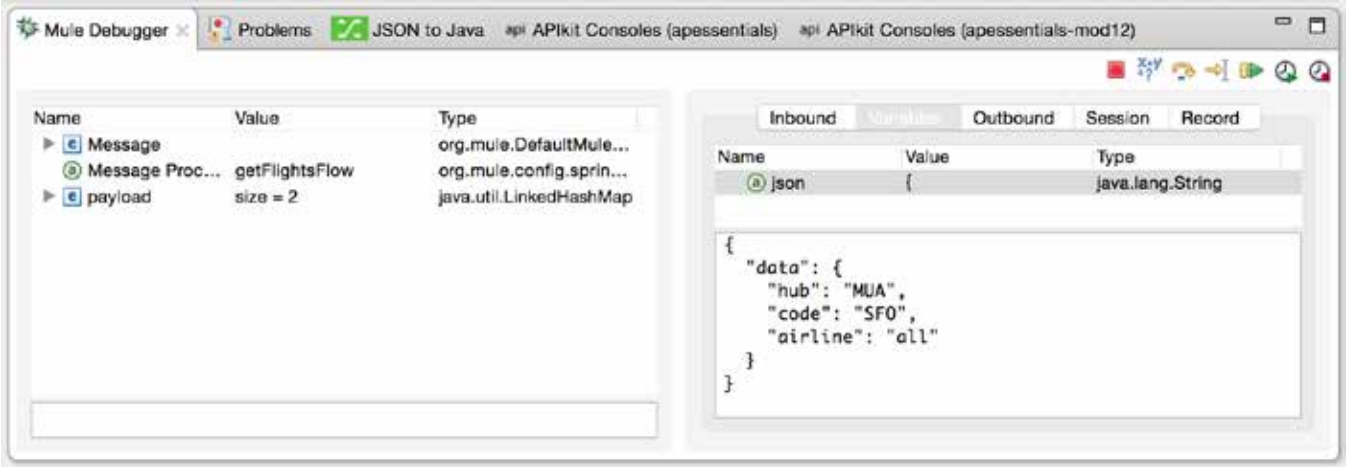


12. Add a field called code and set it to the destination property of the payload; be sure to separate the fields with a comma.
13. Add a field called airline and set it to the airline property of the payload.



Debug the application

14. Save the file and debug the application.
15. Make a request to http://localhost:8081/flights, select LAX and Delta, and submit the form again.
16. Step to the Flow Reference and click the Variables tab in the Mule Debugger; you should now see your flow variable in addition to the transformed payload.

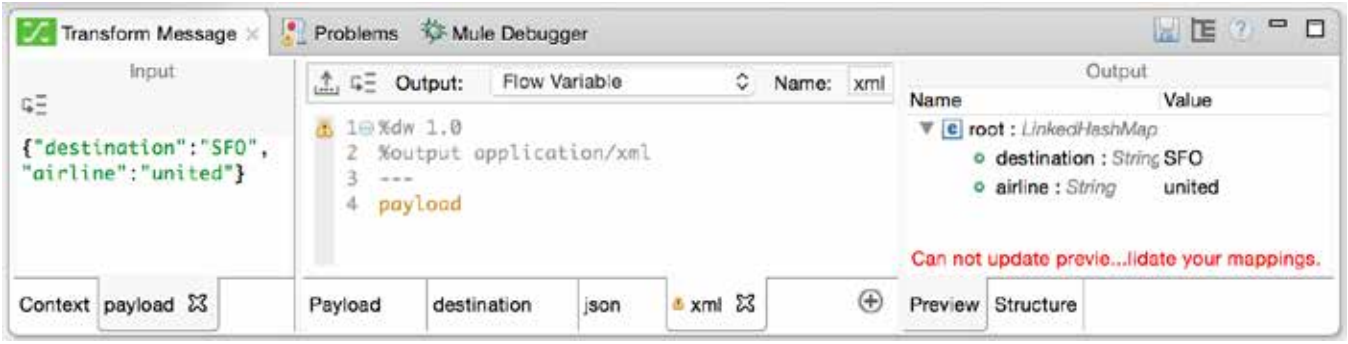


17. Click the Resume button.

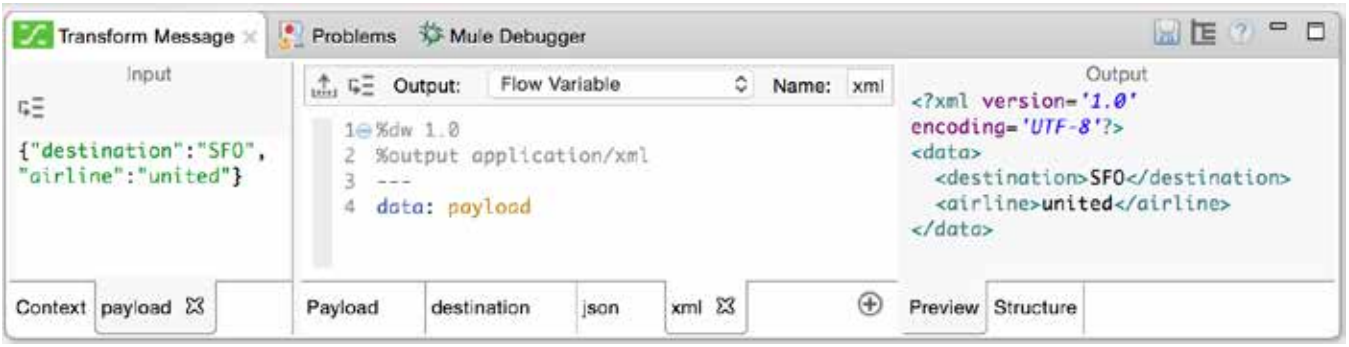
Create a third transformation to output data as XML

18. Click the Add New Transformation button.
19. In the new window, leave the Output set to Flow Variable.
20. Set the name to xml.

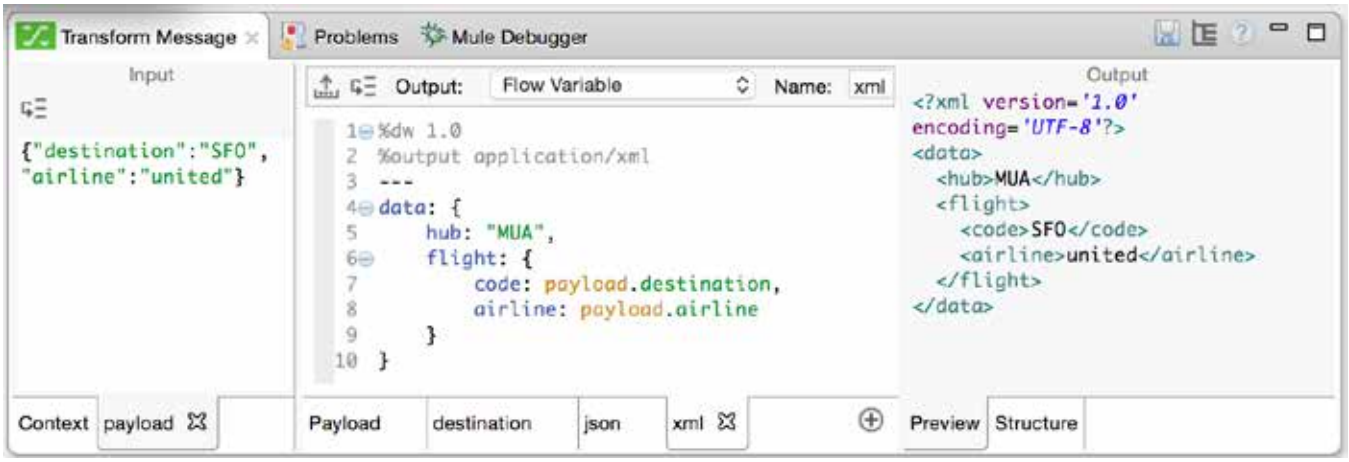
21. Set the DataWeave expression to payload.
22. Change the output type to application/xml; you should get an error message in the preview tab.



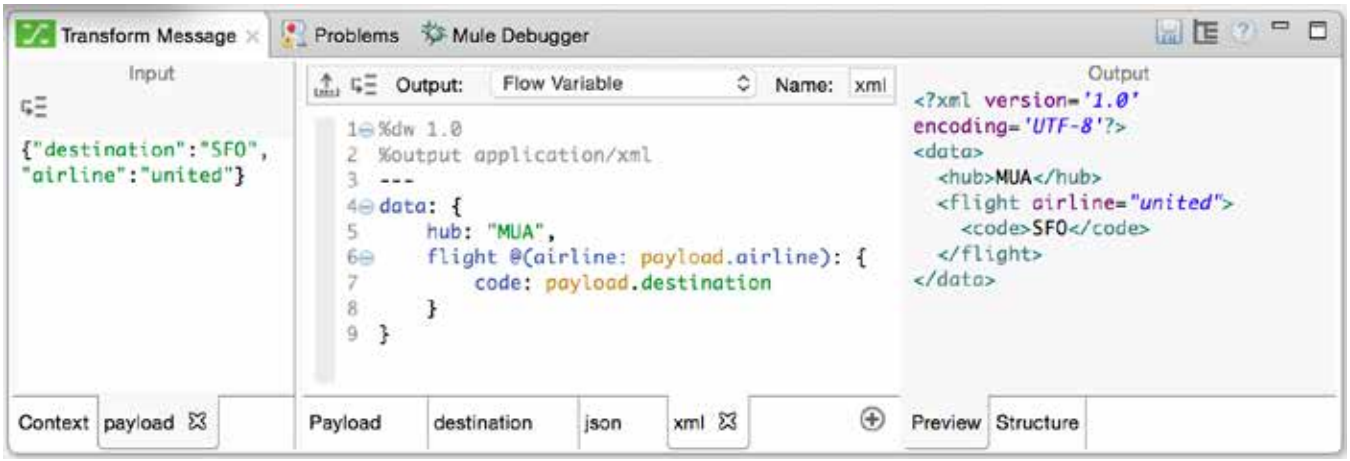
23. Change the DataWeave expression to data: payload.



24. Click the json tab and copy the DataWeave expression.
25. Click the xml tab and replace the DataWeave expression with the one you just copied.
26. Modify the expression so the code and airline properties are child elements of a new element called flight.

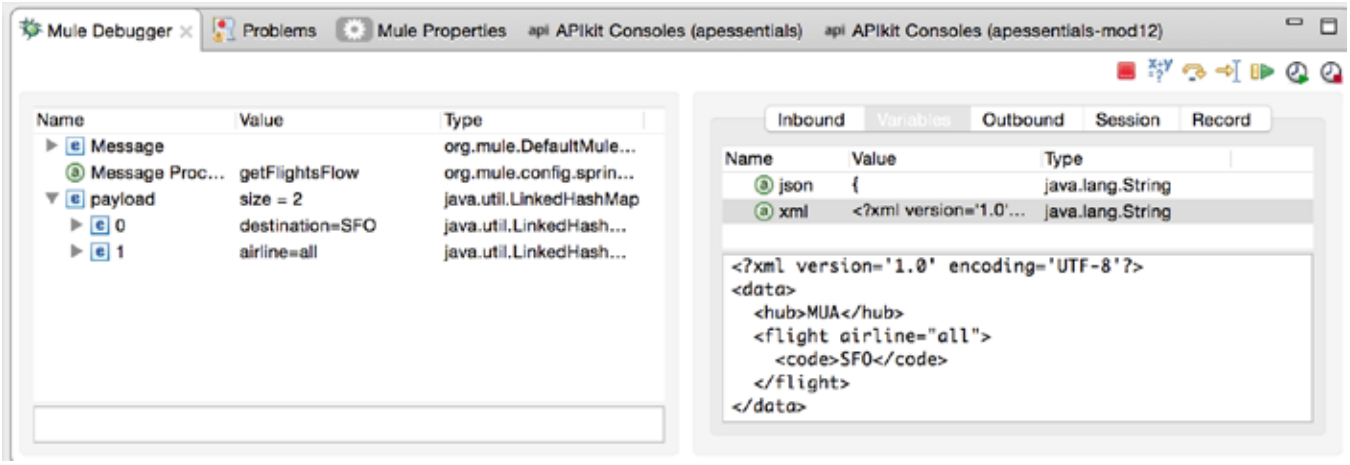


27. Modify the expression so the airline is an attribute of the flight element.



Debug the application

28. Save the file to redeploy the application in debug mode.
29. Make a request to http://localhost:8081/flights and submit the form again.
30. Step to the Flow Reference and click the Variables tab; you should now see two flow variables.



31. Stop the Mule runtime.

Note: The json and xml flow variables were created for learning purposes. If you want, you can delete them from the JSON to Java component.

Walkthrough 12-3: Transform complex data structures

In this walkthrough, you will work with static flight data not retrieved using a connector. You will:

- Create a new flow that receives GET requests at http://localhost:8081/static.
- Transform a JSON array of objects to Java, JSON, and XML.
- Explicitly set the MIME type of the data to be transformed.
- Transform XML with repeated elements to XML and JSON.

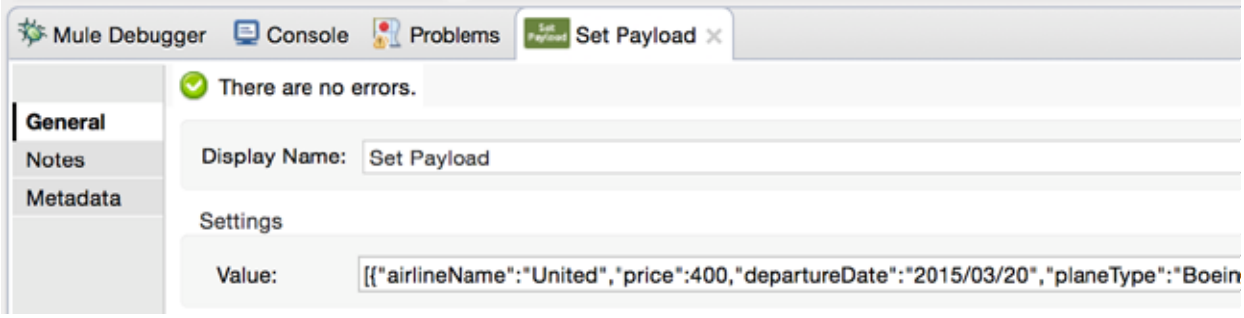
Create a new flow

1. Return to apessentials.xml.
2. Drag out another HTTP connector to create a new flow and name it transformStaticFlightsFlow
3. In the HTTP Properties view, set the connector configuration to the existing HTTP_Listener_Configuration.
4. Set the path to /static and set the allowed methods to GET.

Add static JSON payload data

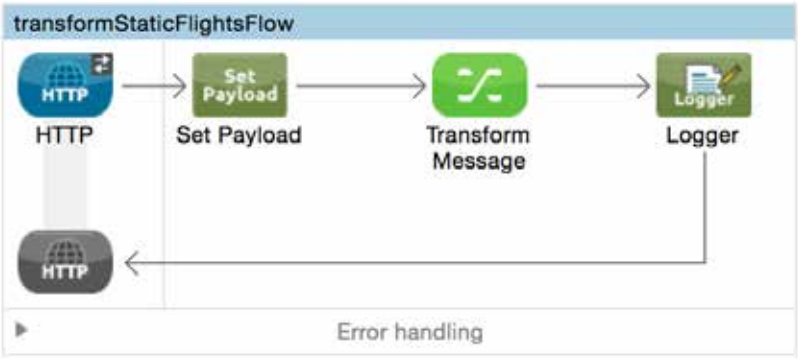
5. Add a Set Payload transformer to the flow.
6. Return to the course snippets.txt file and copy the value for the Example flights JSON.

7. In the Set Payload Properties view, paste the copied value into the value field.



Transform the payload to Java

8. Add a DataWeave Transform Message component and a Logger to the flow.

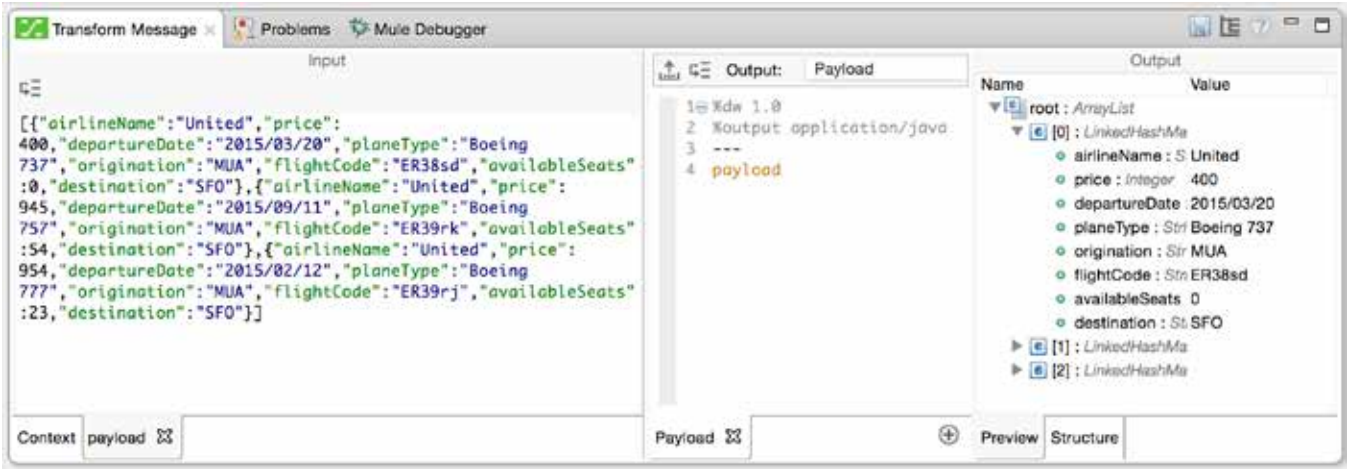


9. In the Transform Message Properties view, set the DataWeave expression to payload.
10. Look at the Preview tab in the Output section; you should not see any preview of the data.

Add sample data

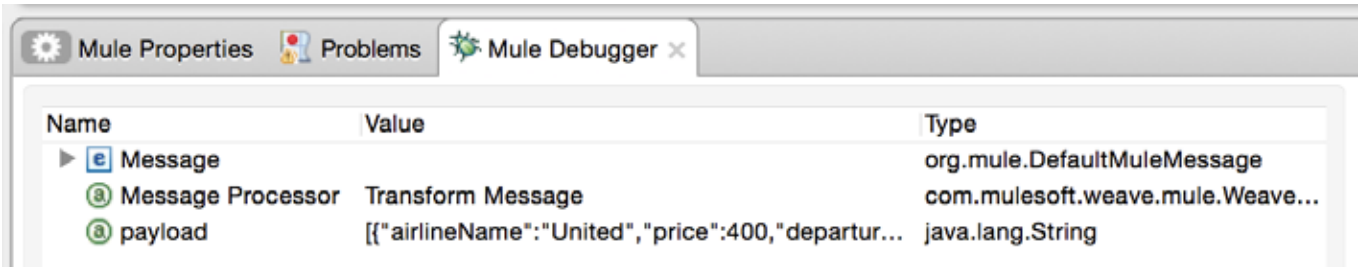
11. In the Input section, click Payload: Unknown.
12. Click the Sample Data button.
13. In the Sample data format dialog box, select JSON and click OK.
14. In the new payload tab, replace the sample JSON with the JSON you used in the Set Payload transformer.

15. Look at the Preview tab in the Output section; you should now see a preview of the data and there should be three LinkedHashMap objects.

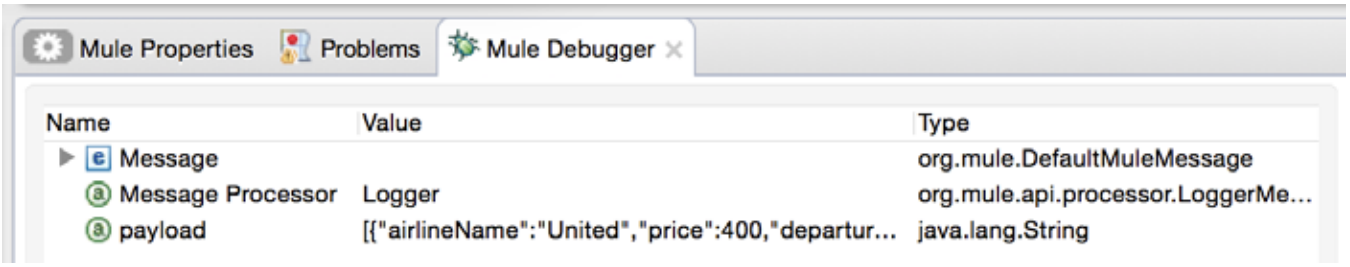


Debug the application

16. Add a breakpoint to the Transform Message component.
17. Save the file and debug the application.
18. Make a request to http://localhost:8081/static.
19. Step to the Transform Message component; you should see the payload is of type String.



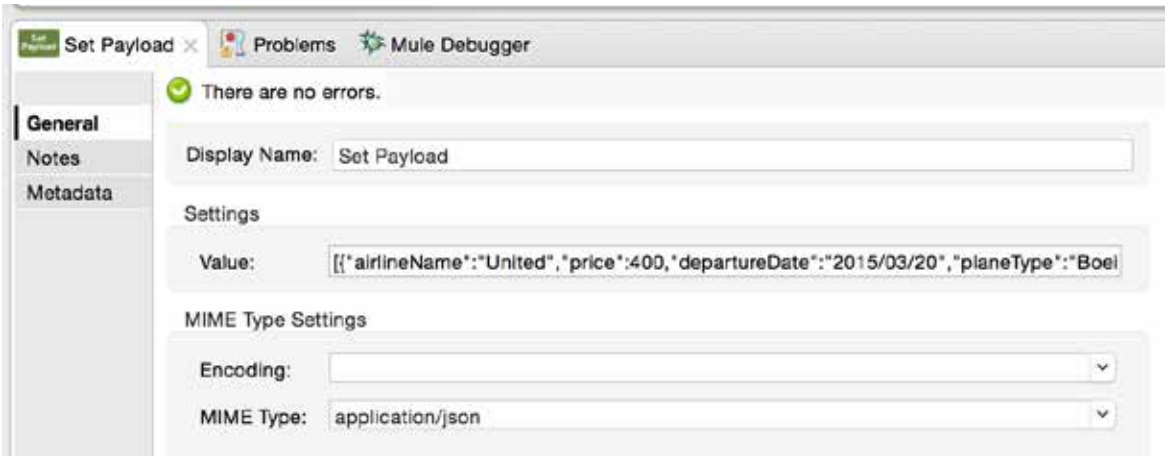
20. Step to the Logger; you should see the payload is still of type String.



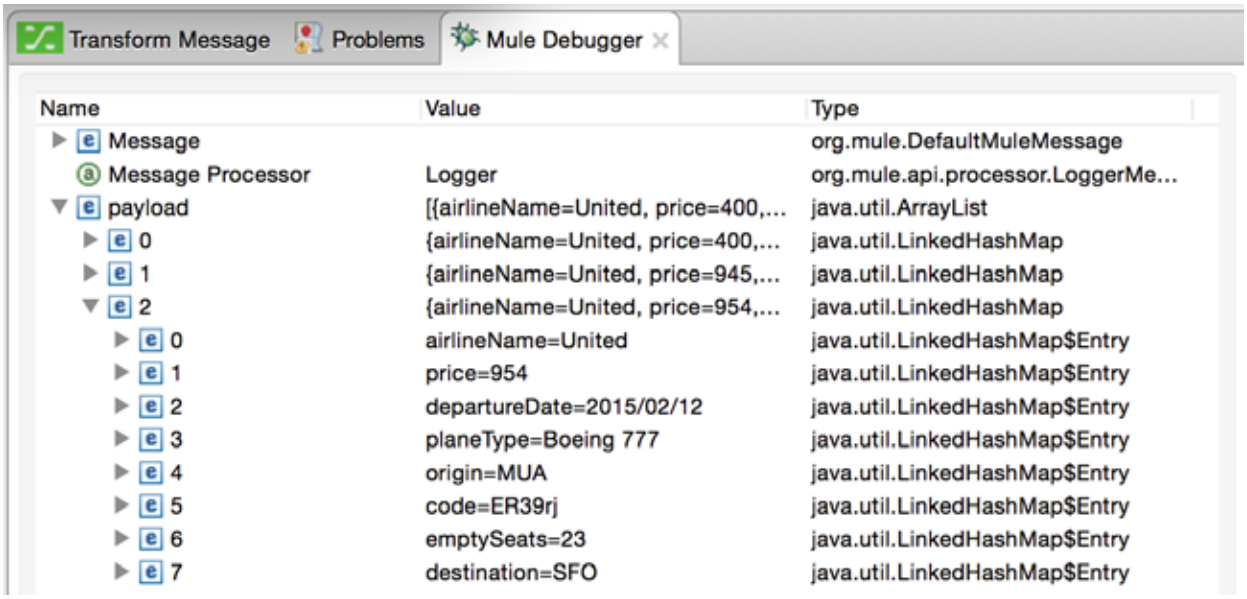
21. Click the Resume button.

Set the payload MIME type

22. In the Set Payload Properties view, set the MIME type to application/json.



23. Apply the changes and wait for the application to redeploy.
24. Make a request to http://localhost:8081/static.
25. Step to the Logger; the payload should now be of type ArrayList.

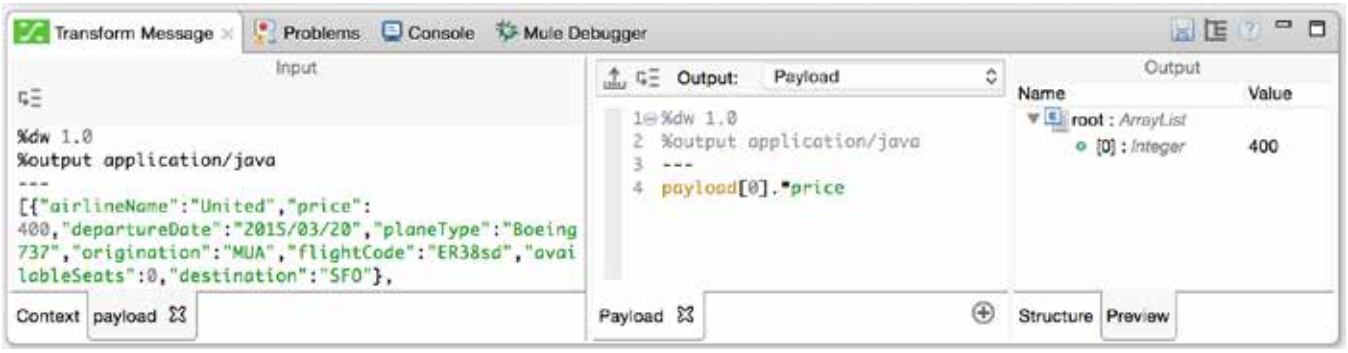


26. Click the Resume button; you should get a file download or garbled data depending upon if you are using a browser or some other tool to make your request.

Return values for the first object in the collection

27. Change the DataWeave expression to payload[0]; in the Preview tab, you should see one LinkedHashMap object.

28. Change the DataWeave expression to payload[0].price; in the Preview tab, you should get 400, the first price value.
29. Change the DataWeave expression to payload[0].*price; in the Preview tab, you should see an ArrayList with one integer value of 400 – the first price value.



Return collections of values

30. Change the DataWeave expression to return a collection of all the prices.

```
payload.*price or payload.price
```

31. Look at the Preview tab; you should see an ArrayList with three values.

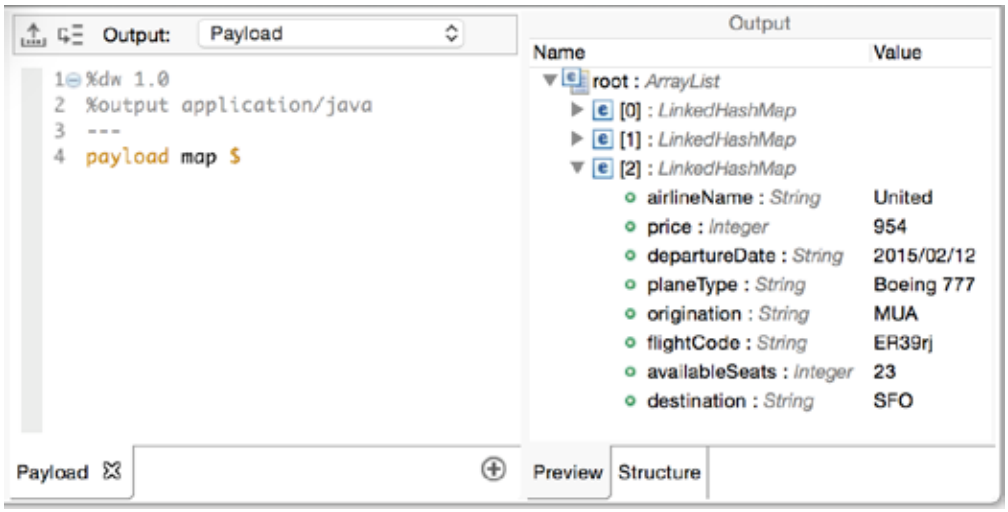
32. Change the DataWeave expression to return a collection of prices and available seats.

```
[payload.price, payload.availableSeats]
```

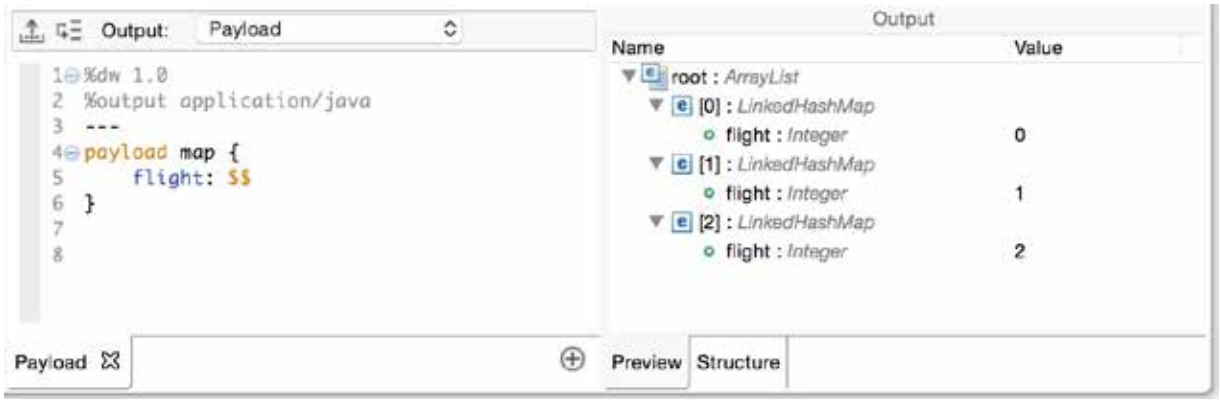
33. Look at the Preview tab; you should see an ArrayList with two ArrayLists.

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

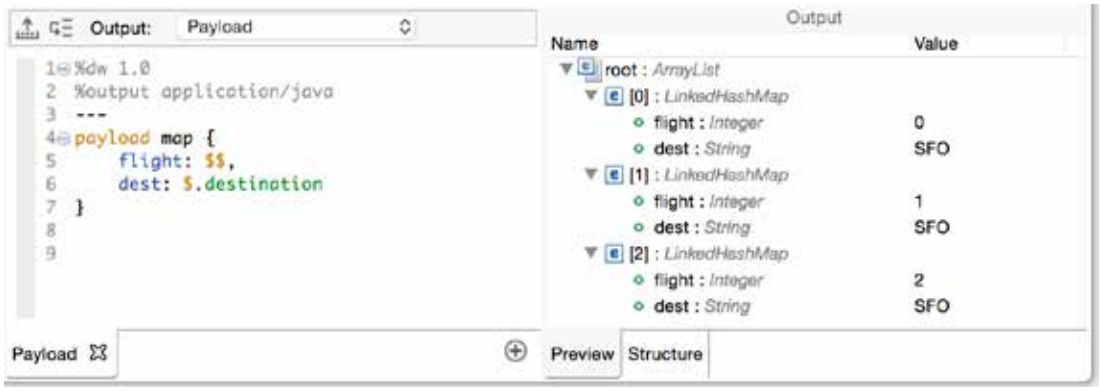
34. Change the DataWeave expression to payload map $, in the Preview tab, you should see three Java objects again.



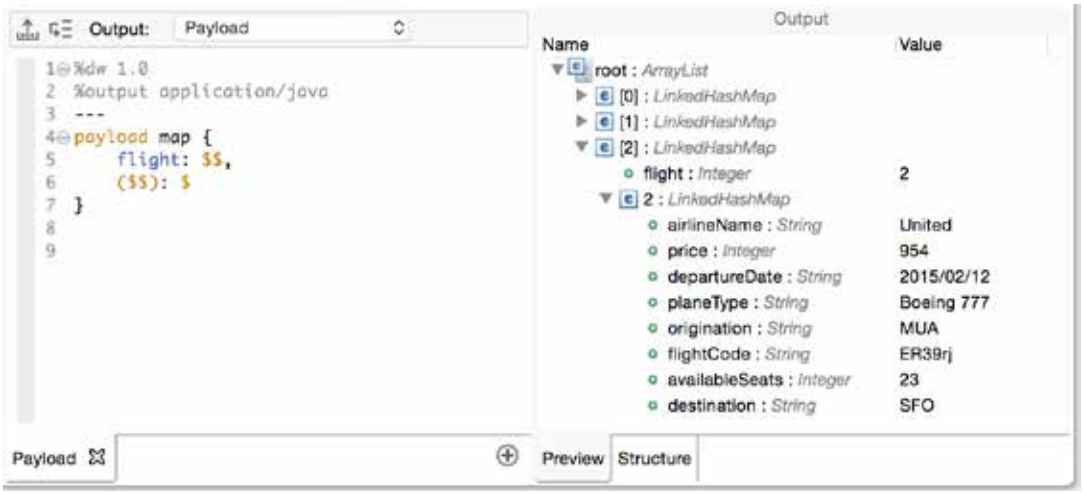
35. Change the DataWeave expression to set a property called flight for each object that is equal to its index value in the collection.



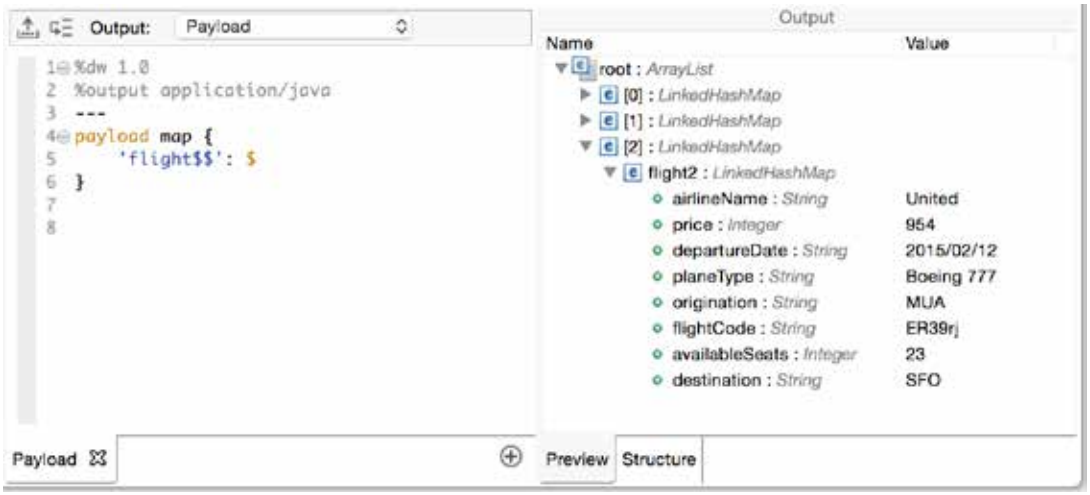
36. Change the DataWeave expression to create a property called dest for each object that is equal to the value of the destination field in the payload collection.



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



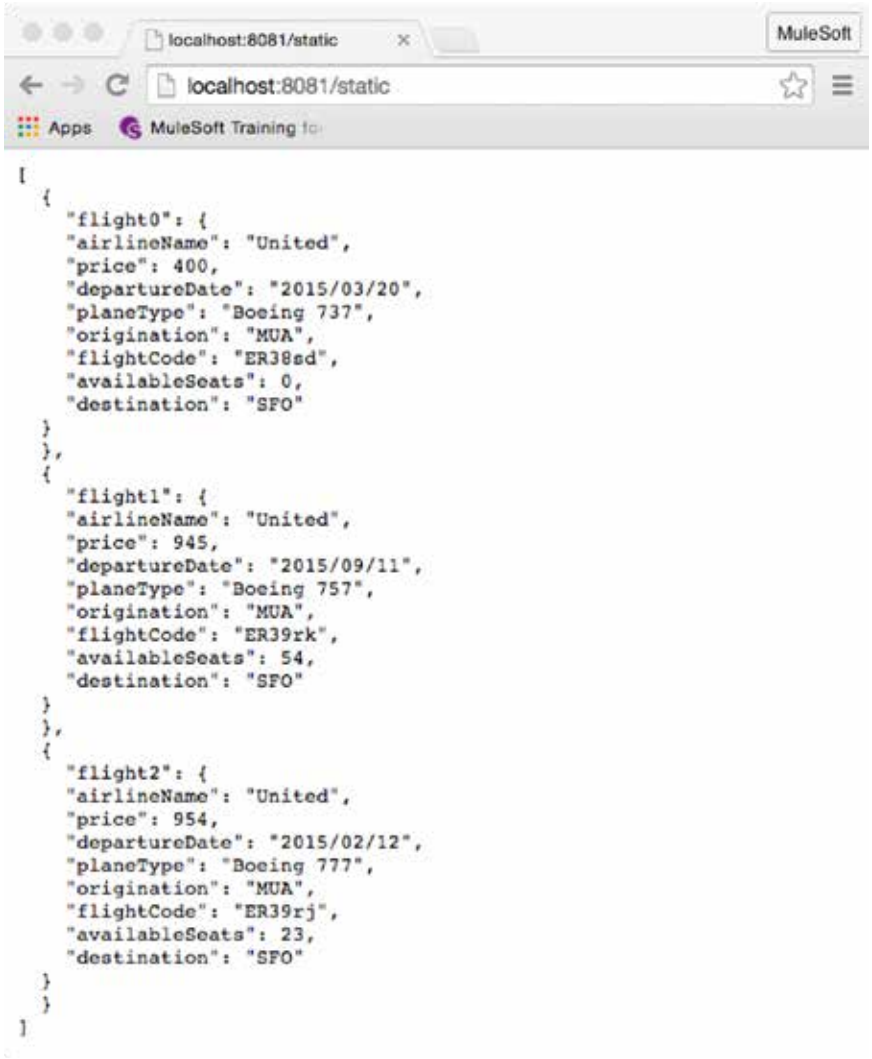
38. Change the DataWeave expression so the name of each object is flight+index and you get flight0, flight1, and flight2.



Change the output to JSON and test the application

39. Change the DataWeave expression output type from application/java to application/json.
40. Save the file and run the application.

41. Make a request to http://localhost:8081/static; you should see the correct JSON returned.

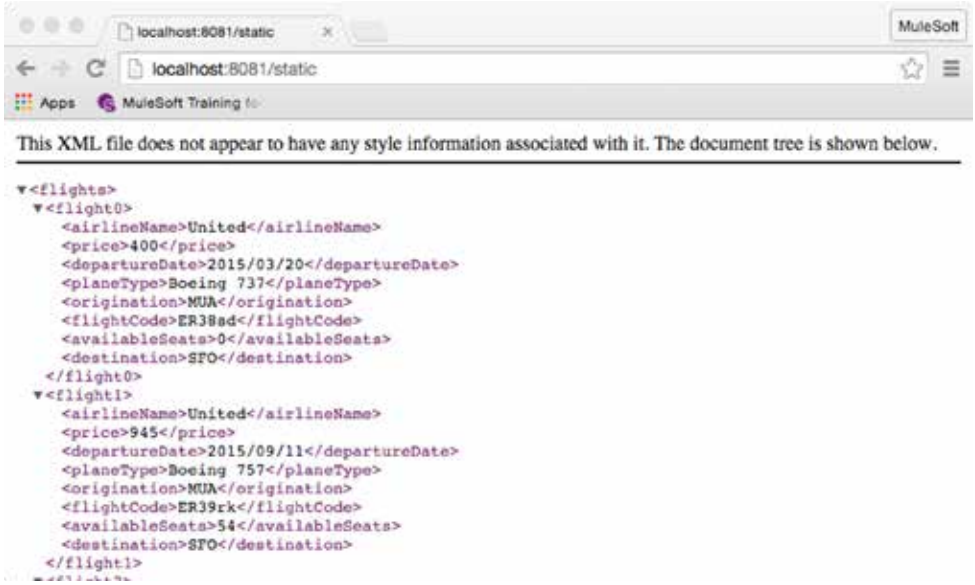


Change the expression to output valid XML and test the application

42. Change the DataWeave expression output type from application/json to application/xml; you should get an error in the Preview tab.
43. Change the DataWeave expression to create a root node called flights; you should still get an error in the Preview tab.
44. Change the expression to map each item in the input array to an XML element.

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

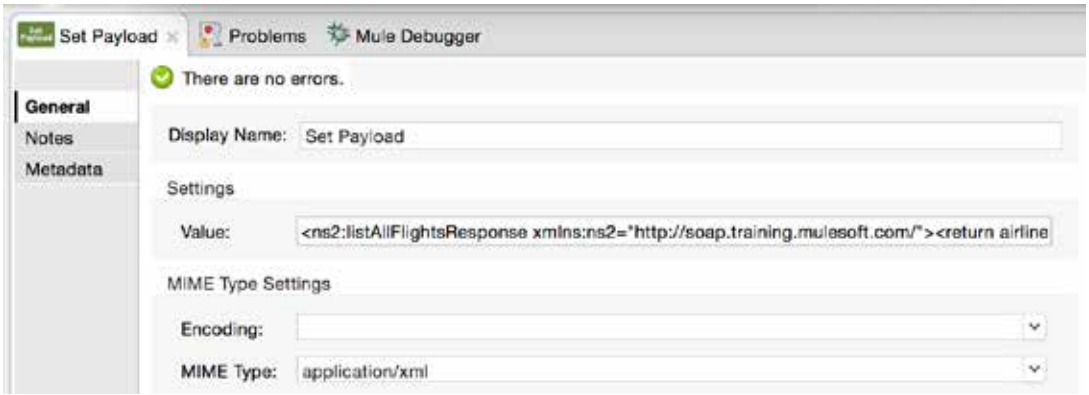
45. Save the file and run the application.
46. Make a request to http://localhost:8081/static; you should see the XML returned.



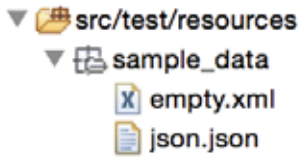
Add static XML payload data

47. Return to the course snippets.txt file and copy the value for the Example flights XML.
48. In the Set Payload Properties view, paste the copied value into the value field.

49. Change the MIME type to application/xml.



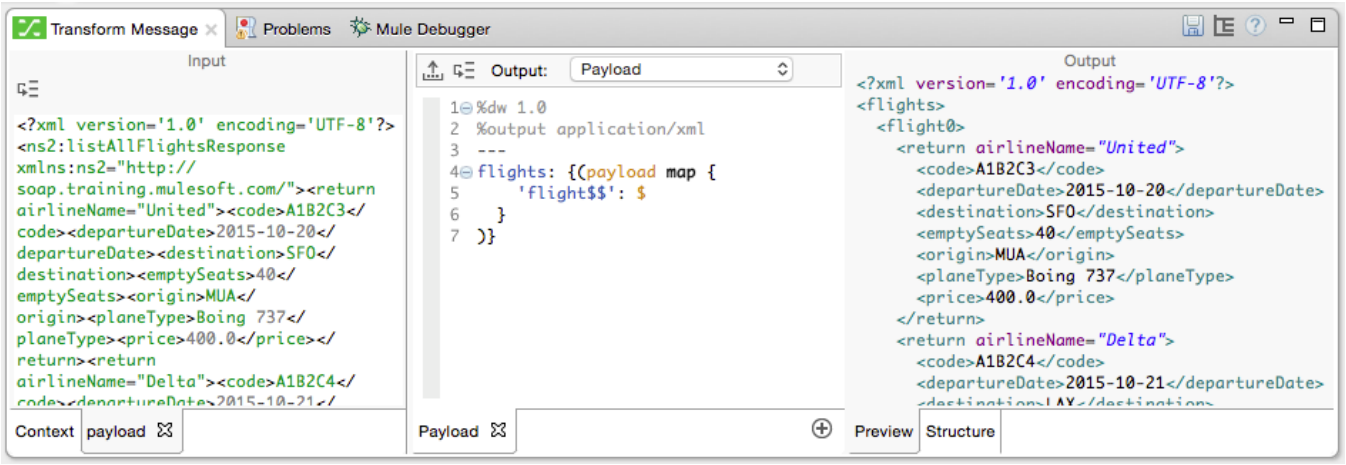
50. In the Input section of the Transform Message Properties view, click the x on the payload tab to delete the src/test/resources/sample_data/json_1.json file.
51. Click Payload: Unknown and then the Sample Data button.
52. In the Sample data format dialog box, select XML and click OK; a new sample data file empty.xml should be created.



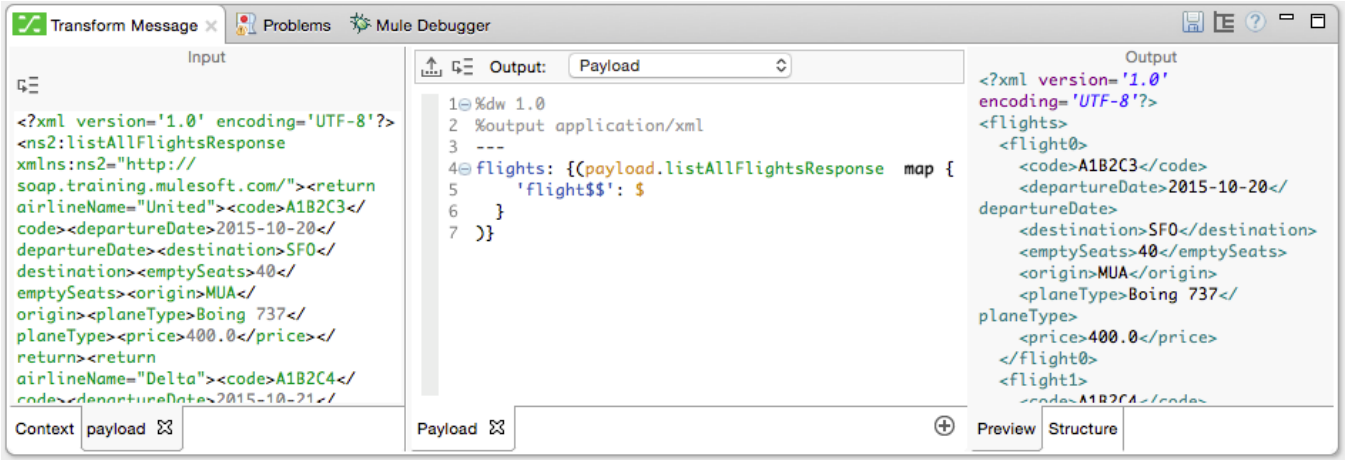
53. In the Transform Message Properties view, replace the sample payload data with the Example flights XML you copied.

Change the return structure of the XML

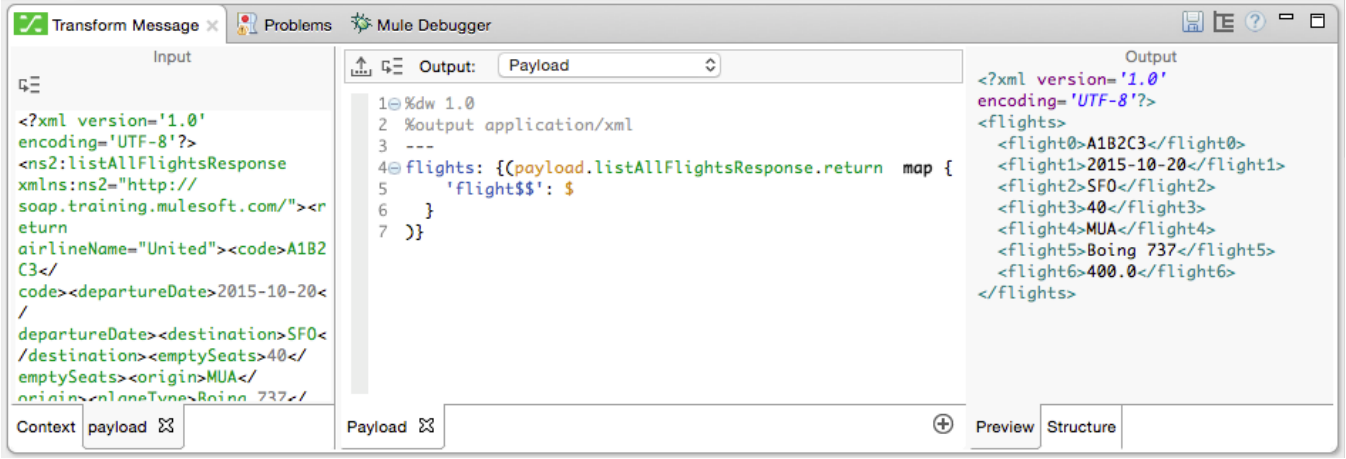
54. Look at the Preview tab; you should see all the flights are children of the flight0 element.



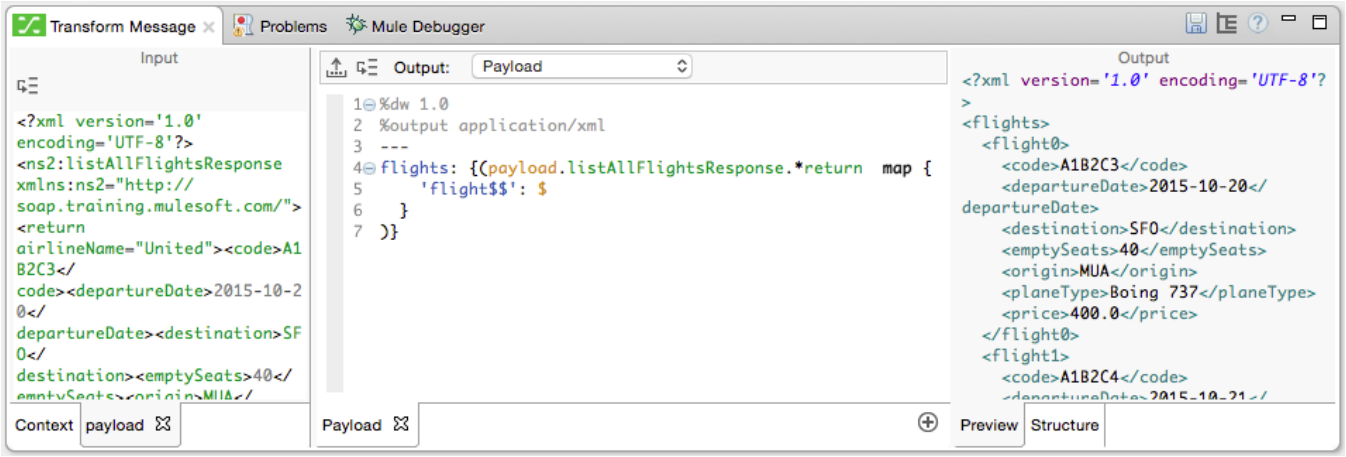
55. Change the DataWeave expression to loop over the payload.listAllFlightsResponse element; you should see the flights are now correctly transformed.



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



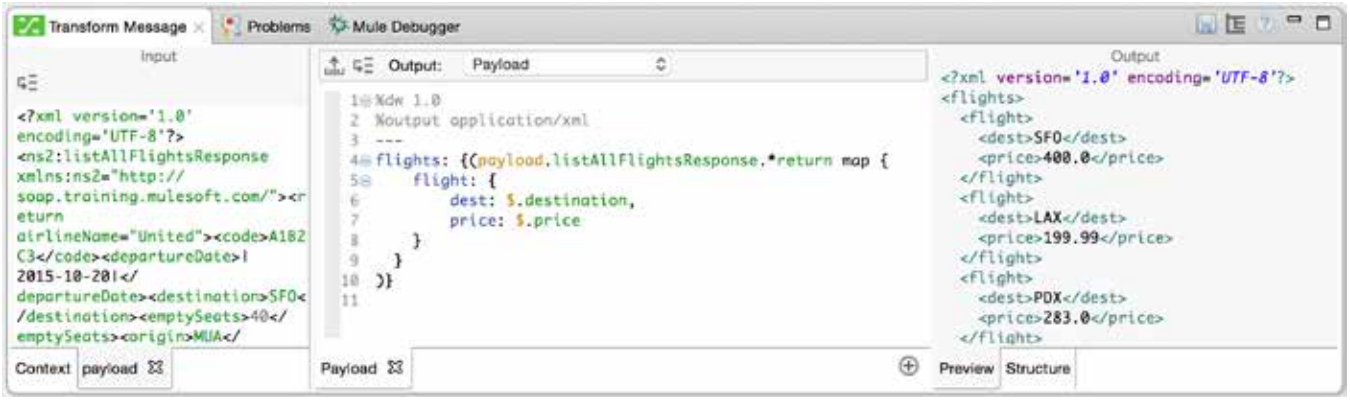
57. Change the DataWeave expression use * to loop over the XML repeated return elements.



58. Change the DataWeave expression to return XML with repeated flight elements.

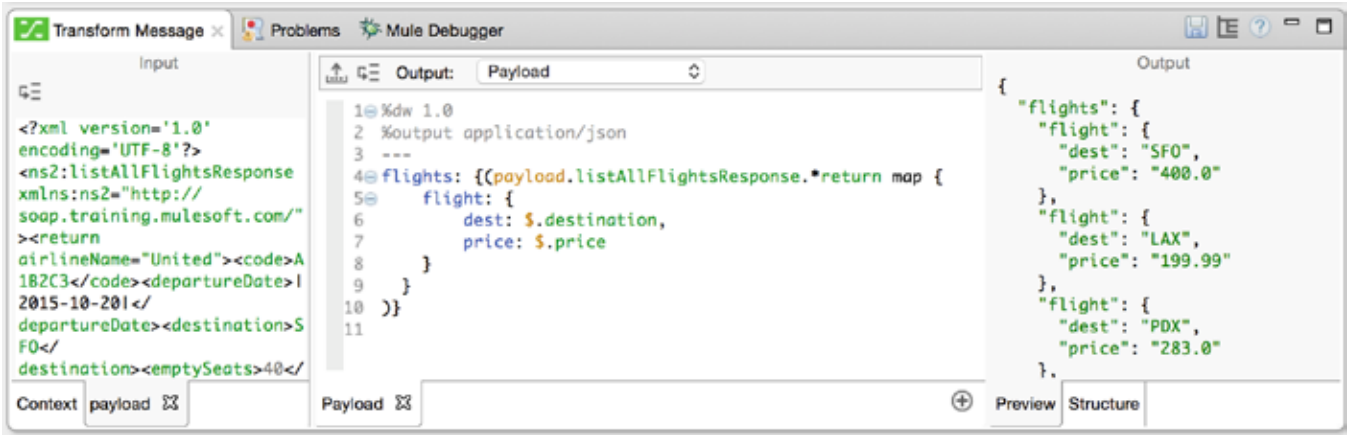
```
flights: {(payload.listAllFlightsResponse.*return map {
     flight: $
  }
)}
```

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



Change the output structure to JSON

60. Change the transform output type from XML to JSON.



Note: This output JSON has repeated object keys and is considered by most parsers to be invalid JSON.

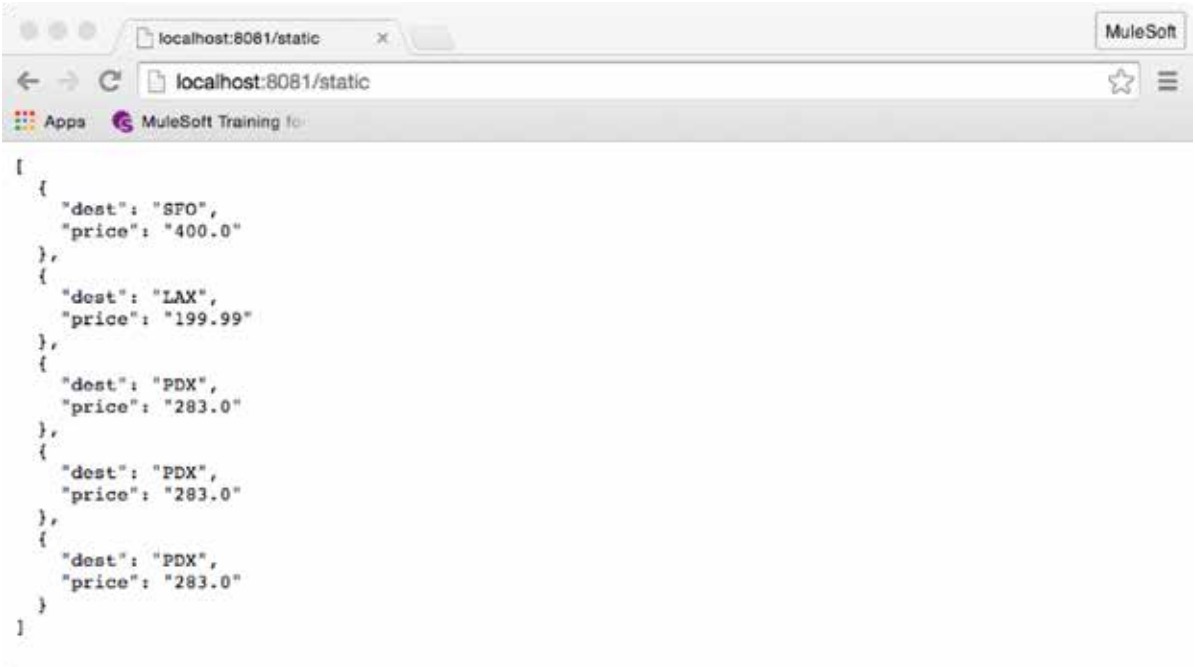
61. Remove the {(and)} around the payload map expression.

```
flights: payload.listAllFlightsResponse.*return map {
      flight: {
            dest: $.destination,
            price: $.price
      }
}
```

62. Change the DataWeave expression to return an array of objects without the flights and flight properties.

```
payload.listAllFlightsResponse.*return map {
      dest: $.destination,
      price: $.price
}
```

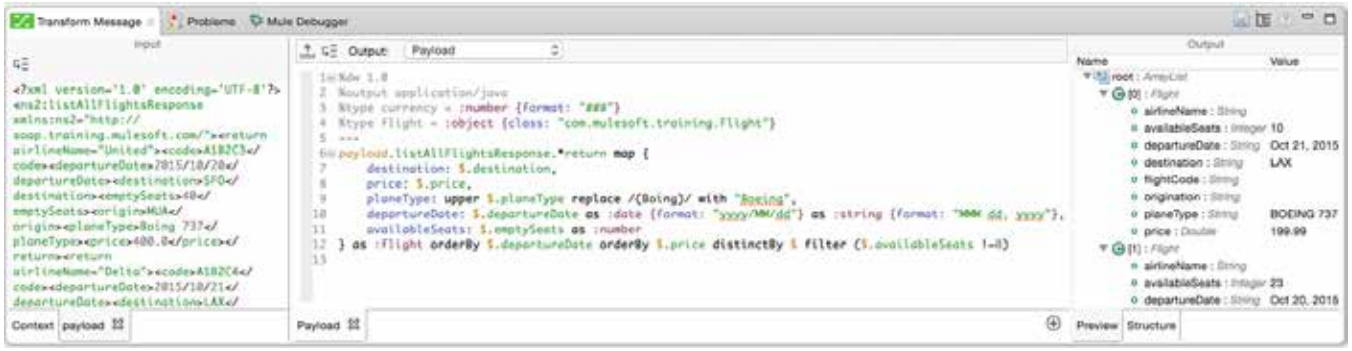
63. Save the file and run the application.
64. Make a request to http://localhost:8081/static and look at the JSON structure returned.



Walkthrough 12-4: Use DataWeave operators

In this walkthrough, you will continue to work with the static flight data from the last exercise. You will:

- Format strings, dates, and numbers.
- Convert data types.
- Replace data values using pattern matching.
- Order data, filter data, and remove duplicate data.
- Define and use custom data types.
- Transform objects to POJOs.

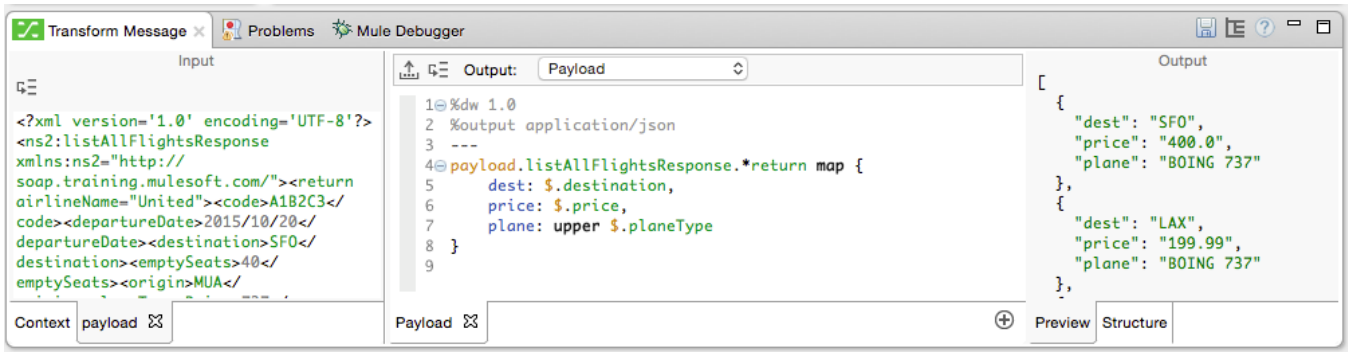


Format a string

1. Return to transformStaticFlightsFlow in getFlights.xml.
2. Change the DataWeave expression to return objects with an additional property called plane equal to the value of the input planeType values.
3. Use the upper operator to return the value in uppercase.

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

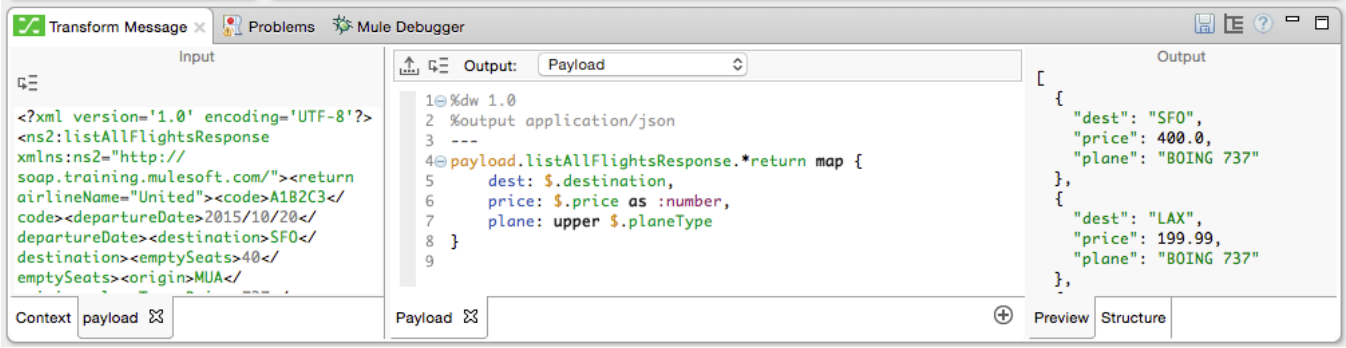
4. Look at the Preview tab; you should see the uppercase plane fields.
5. Look at the data type of the prices; you should see that they are strings (surrounded by quotation marks).



Convert data types

6. Change the DataWeave expression to use the as operator to return the prices as numbers (not surrounded by quotation marks) instead of strings.

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



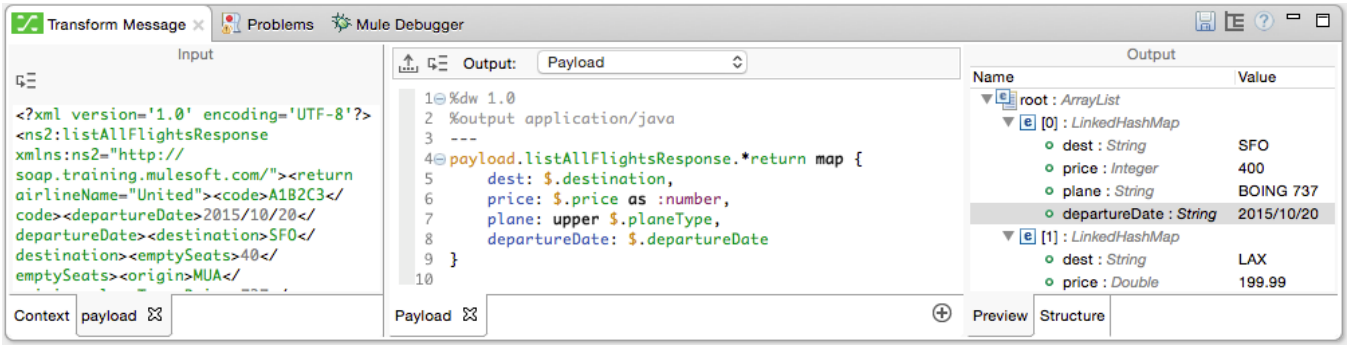
7. Add a departureDate field to the return object.

```
payload.listAllFlightsResponse.*return map {
      dest: $.destination,
      price: $.price as :number,
      plane: upper $.planeType,
      departureDate: $.departureDate
}
```

8. Change the transform output type from json to java.

```
%output application/java
```

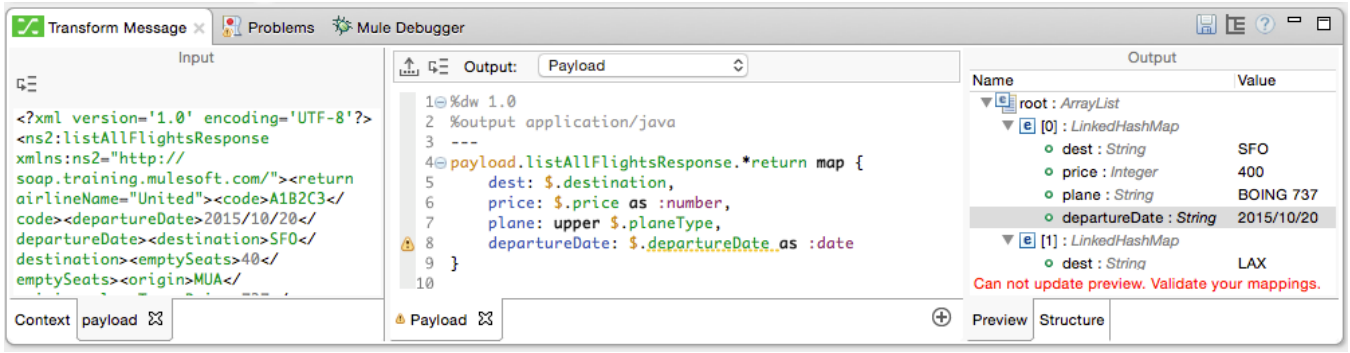
9. Look at the Preview tab; you should see the departureDate property is a String and price is an Integer or Double.



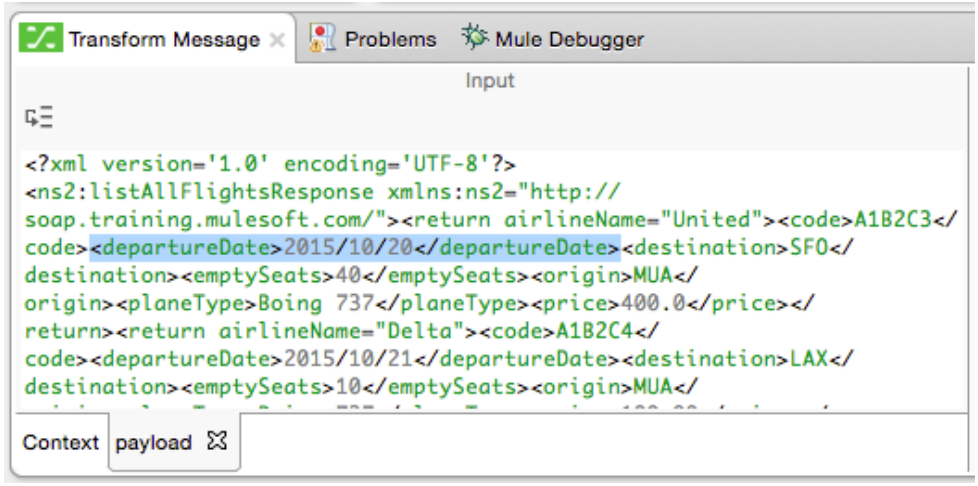
10. Change the DataWeave expression to use the as operator to convert departureDate to a date object.

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

11. Look at the Preview tab; you should get an error message.



12. Look at the format of the dates in the sample data in the payload tab in the Input section.

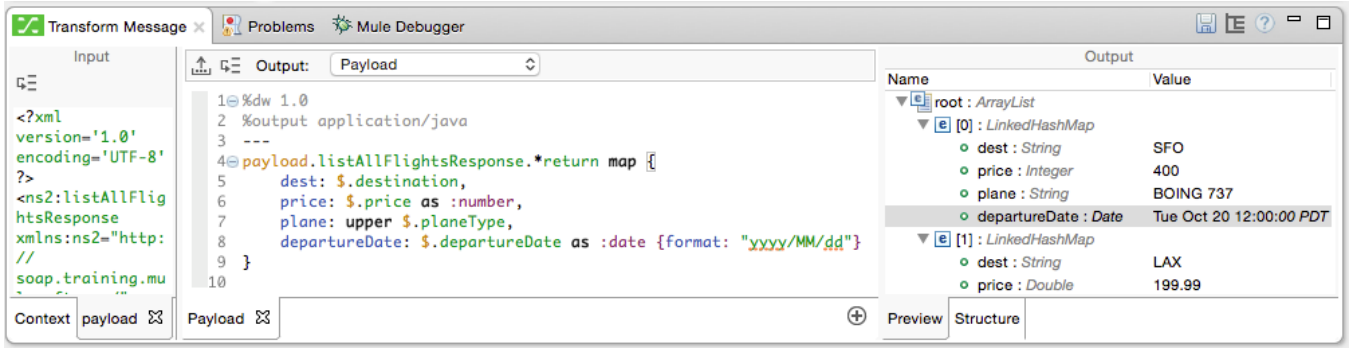


13. Use the format operator to specify the pattern of the input date strings.

```
departureDate: $.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. See: https://docs.oracle.com/javase/8/docs/api/java/time/format/DateTimeFormatter.html.

14. Look at the Preview tab; you should see departureDate is now a Date object.

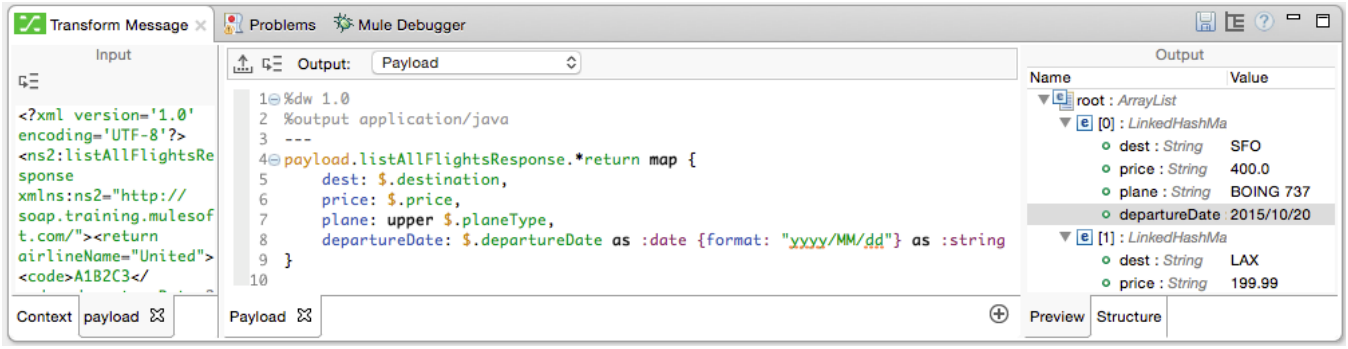


Format a date

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

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

16. Look at the Preview tab; the departureDate is again a String.



17. Use the format operator with any pattern letters and characters to format the date strings.

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

18. Look at the Preview tab; the departureDate properties should now be formatted.

19. Change the transform output from java to json.

```
%output application/json
```

20. Save the file and run the application.
21. Make a request to http://localhost:8081/static; you should see the formatted dates.

Format a number

22. Use the format operator in the DataWeave expression to format the prices with two decimal places.

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

23. Look at the Preview tab; the prices should be formatted.

Note: There is a bug in Anypoint Studio 5.2.1 (whose live preview uses Mule runtime 3.7.1) and Mule runtime 3.7.1 and the numbers are not formatted correctly.

24. Save the file and run the application.
25. Make a request to http://localhost:8081/static; you should see the formatted prices (but may not).
26. Change the DataWeave expression to format the prices to zero decimal places.

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

27. Look at the Preview tab; you should (but may not) see the second price is now 200 instead of 199.99.
28. To get trailing zeros, convert the numbers to strings and format the strings.

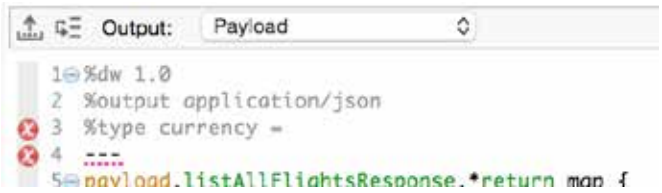
```
price: $.price as :number as :string {format: "#,###.00"},
```

29. Use the as operator again to convert the formatted strings back to numbers.

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

Define a custom data type

30. In the header section of the Transform section, define a custom data type called currency.



31. Set it to a number formatted to have no digits after the decimal point.

```
%type currency = :number {format: "###"}
```

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

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

33. Look at the Preview tab; the prices should still be formatted.

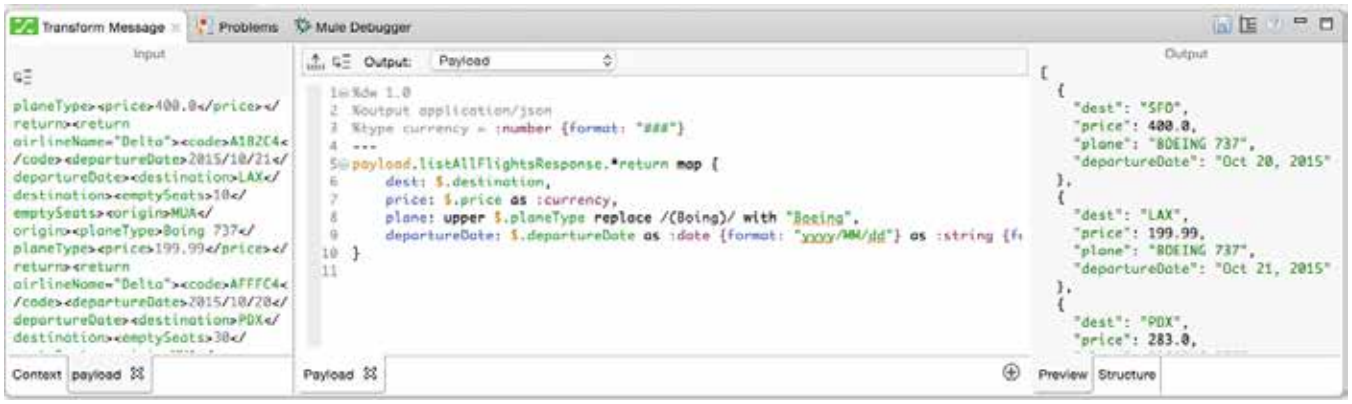
Note: There is a bug in Mule runtime 3.7.1 and the numbers are not formatted correctly.

Replace data values

34. Use the replace operator in the DataWeave expression to replace the string Boing with Boeing.

```
plane: upper $.planeType replace /(Boing)/ with "Boeing",
```

35. Look at the Preview tab; Boeing should be spelled correctly.

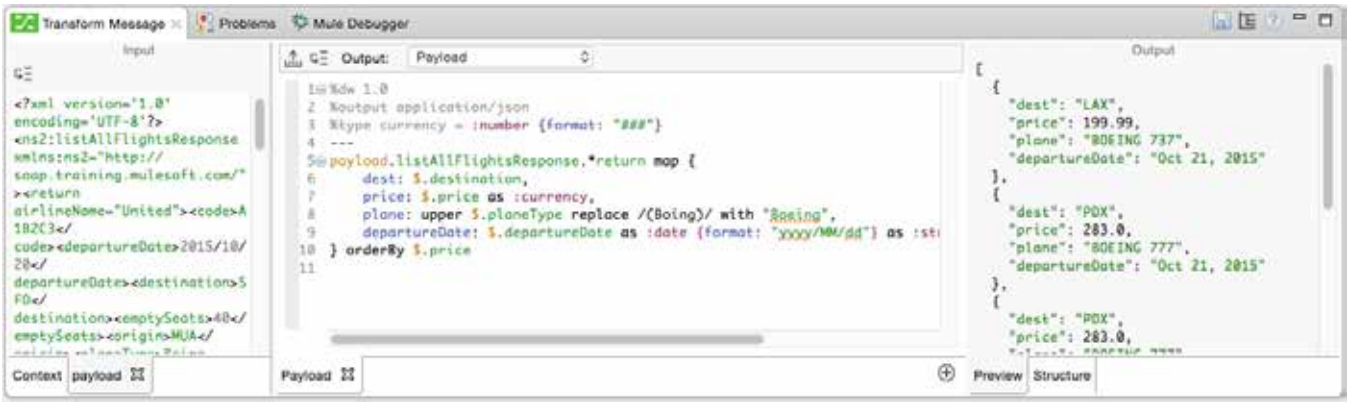


Order data

36. In the Preview tab, look at the flight prices; the flights should not be ordered by price.
37. Change the DataWeave expression to use the orderBy operator to order the objects by price.

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

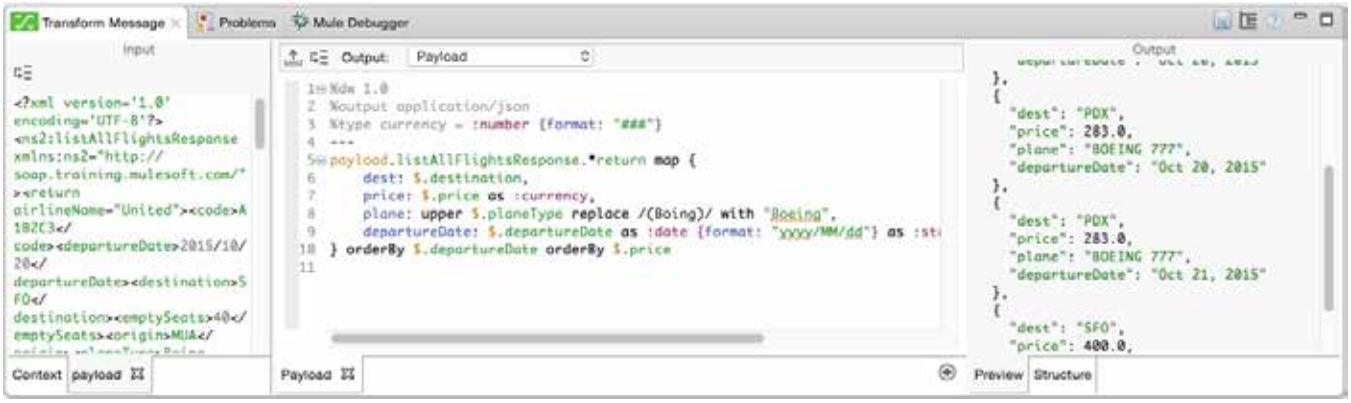
38. Look at the Preview tab or run the application; the flights should now be ordered by price.



39. Look at the PDX flights; they should not be ordered by departureDate.
40. Change the DataWeave expression to first sort by departureDate and then by price.

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

41. Look at the Preview tab or run the application; the flights should now be ordered by price and flights of the same price should be sorted by departureDate.

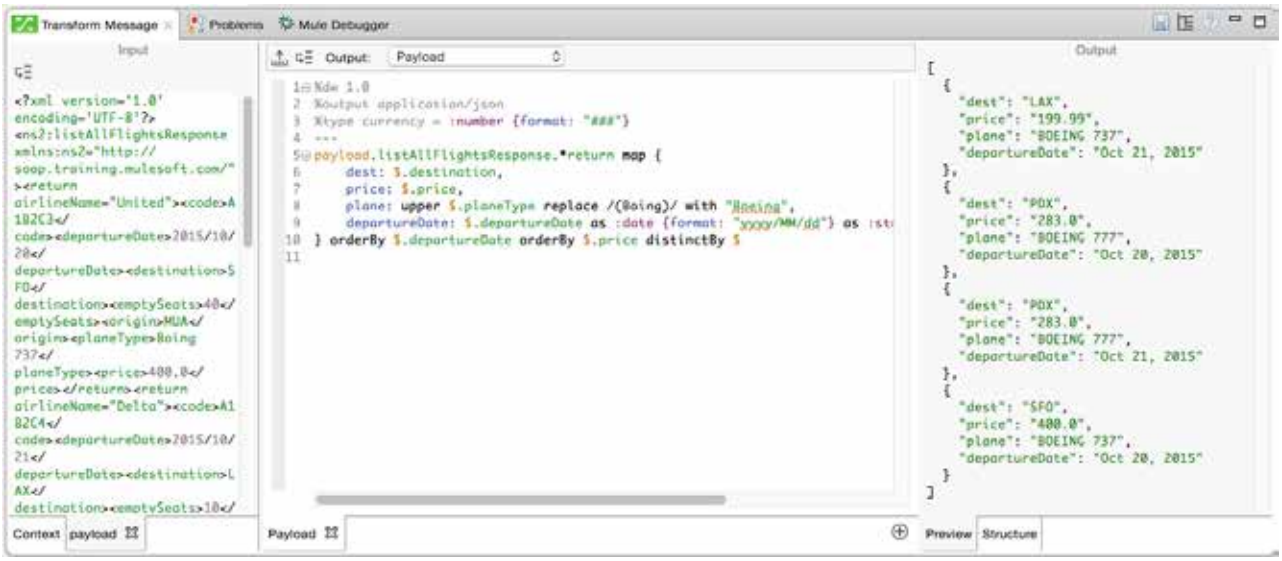


Remove duplicate data

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

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

43. Look at the Preview tab; you should now see only four flights instead of five.



Note: If you still get five flights, remove the currency type coercion from the price field; use price: $.price or price: $.price as :number {format: "###"} instead of price: $.price as :currency. This is a bug in Mule runtime 3.7.1.

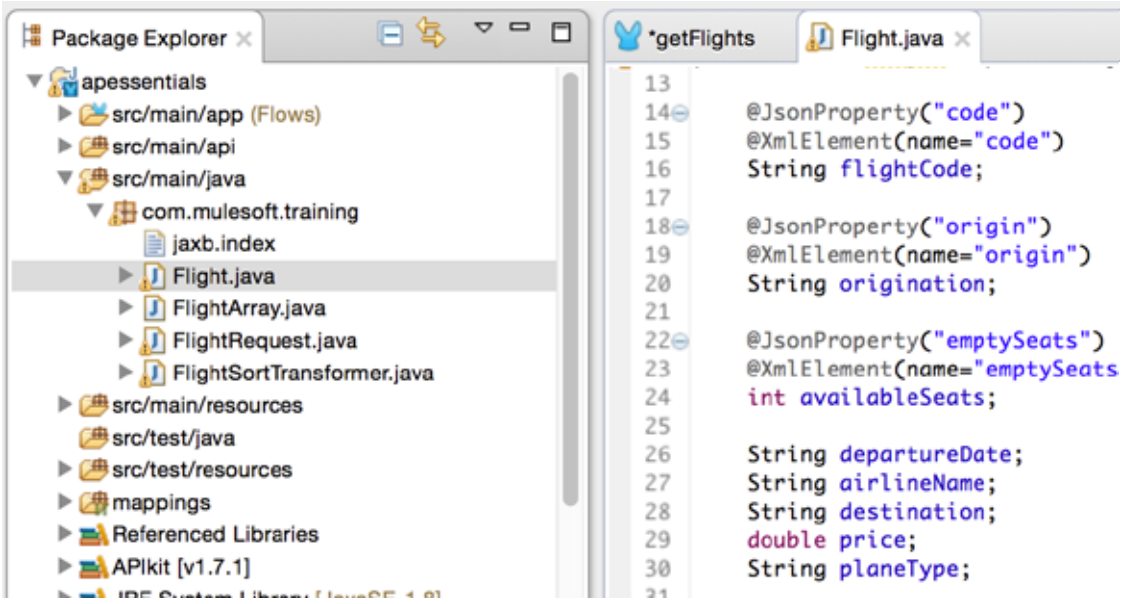
44. Add an availableSeats field that is equal to the emptySeats field and coerce it to a number.

```
availableSeats: $.emptySeats as :number
```

45. Look at the Preview tab; you should see five flights again.

Transform objects to POJOs

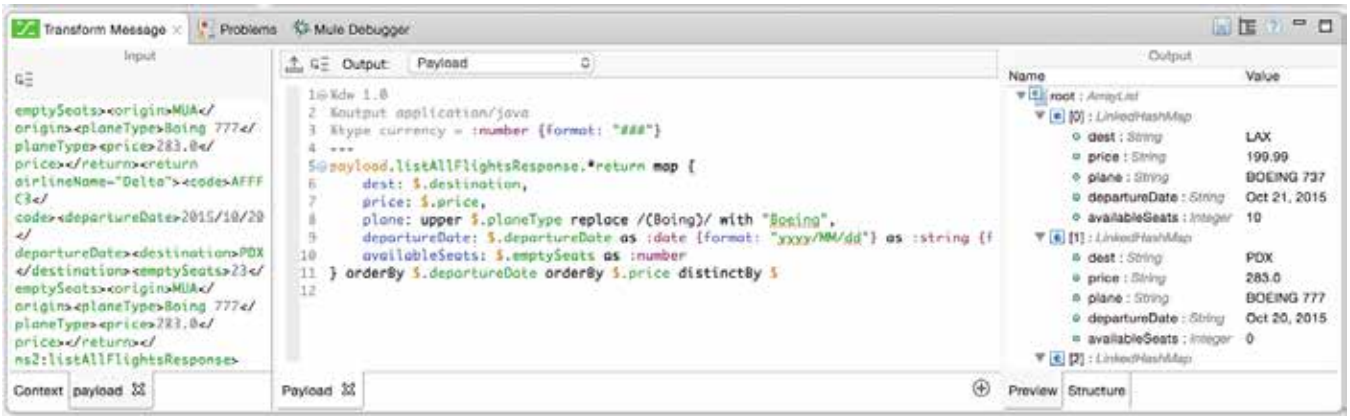
46. Open the Flight.java class in src/main/java and look at the names of the properties.



47. Return to transformStaticFlightsFlow in getFlights.xml.
48. Change the transformation output from json to java.

```
%output application/java
```

49. Look at the Preview tab and see that LinkedHashMap objects are created.



50. In the header section of the Transform section, define a custom data type called flight that is a com.mulesoft.training.Flight Java object.

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

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

```
payload.listAllFlightsResponse.*return map {
        …
} as :flight orderBy $.departureDate orderBy $.price distinctBy $
```

52. In the Transform section, look at the warning you get: Key 'dest' not found.

```
3  %type currency = :number {format: "###"}
4  %type flight = :object {class: "com.mulesoft.training.Flight"}
5  ---
6  payload.listAllFlightsResponse.*return map {
7      dest: $.destination,
8      price: $.price,
9      plane: upper $.planeType replace /(Boing)/ with "Boeing",
10     departureDate: $.departureDate as :date {format: "yyyy/MM/dd"} as :stri
11     availableSeats: $.emptySeats as :number
12 } as :flight orderBy $.departureDate orderBy $.price distinctBy $
13
```

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

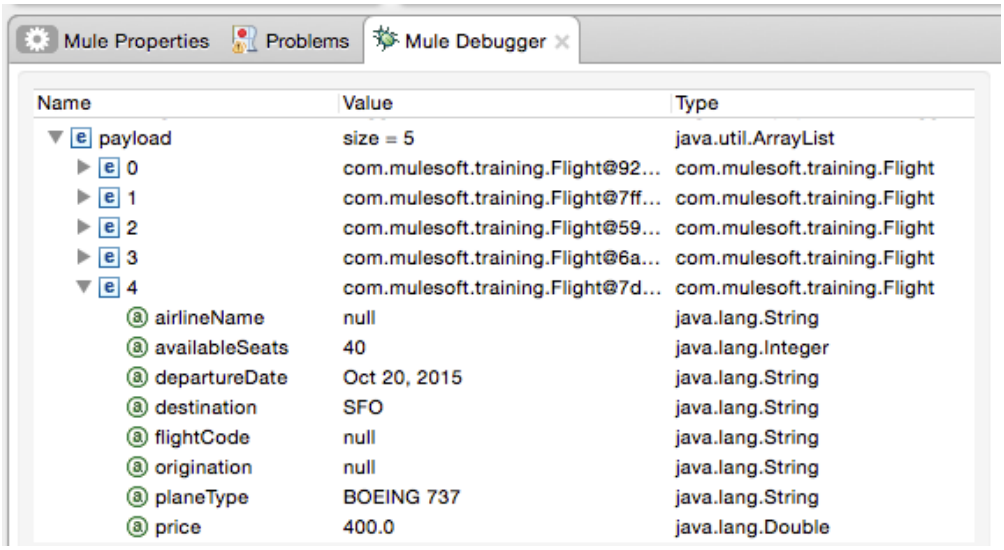
```
destination: $.destination,
```

54. Change the name of the plane key to planeType.

```
planeType: $.planeType,
```

55. Look at the Preview tab; the objects should now be of type Flight.

56. Save the file and debug the application.
57. Make a request to http://localhost:8081/static.
58. Step to the Logger and drill-down into the payload; the objects should be of type Flight and several of the fields should have null values because no values were assigned to them.



59. Stop the debugger.

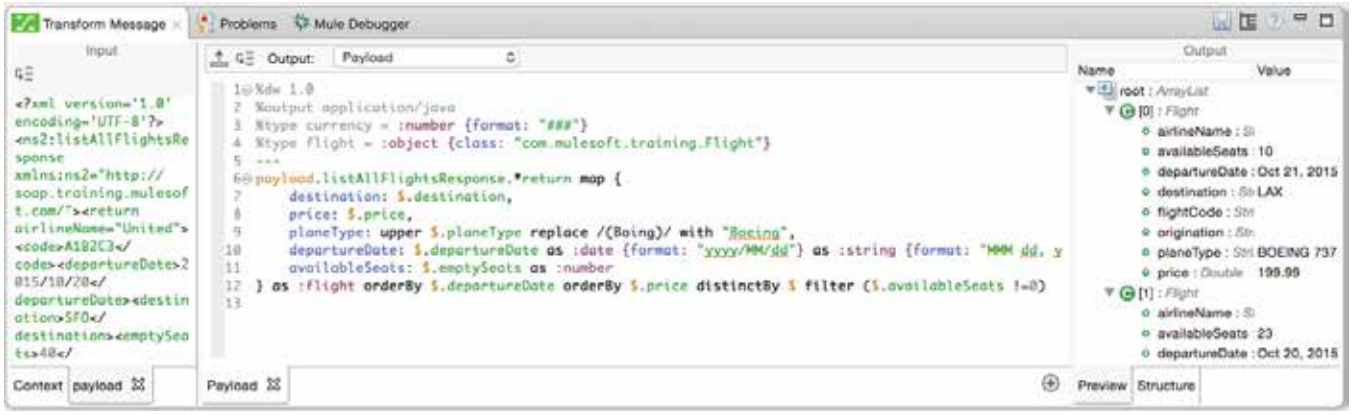
Filter data

60. Return to transformStaticFlightsFlow.
61. In the Preview tab, look at the values of the availableSeats properties.

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

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

63. Look at the Preview tab; you should no longer see the flight that had no seats.



Walkthrough 12-5: Transform data sources that have associated metadata

In this walkthrough, you will work with the American and Delta flight data. You will:

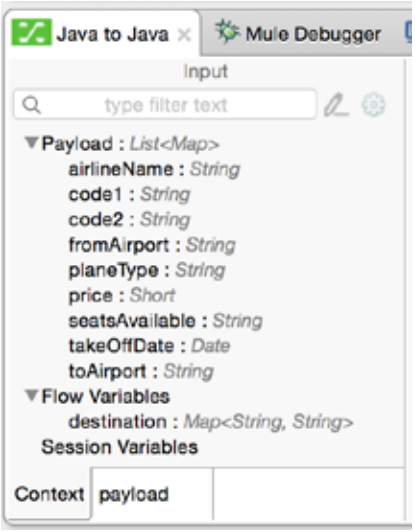
- Use DataWeave to transform the American flight data from a collection of Java objects to to POJOs with a different data structure.
- Use DataWeave to transform the Delta flight data from XML to a collection of POJOs.

Add a DataWeave Transform Message component

1. Locate getAmericanFlightsFlow in getFlights.xml.
2. Delete the DataMapper transformer.
3. Replace it with a Transform Message component.

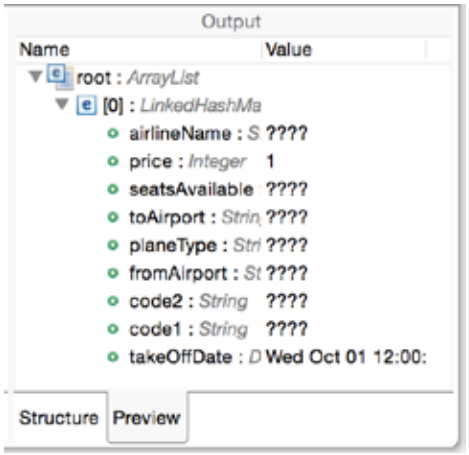
4. Change the name of the Transform Message component to Java to Java.

5. Look at the Input section of the Transform Message Properties view; the payload should automatically be set to a List<Map> with correct properties because the Database connector configuration is DataSense enabled by default.



Note: If the payload is not automatically set to a List<Map>, you probably need to refresh the connector metadata. In the American DB Request Properties view, click the Output tab in the Metadata section and see if the payload is shown to be of type List<Map>. If it is not, click the Refresh Metadata link beneath the metadata. If this does not work, select Anypoint > Preferences / Window > Preferences and navigate to Anypoint Studio > DataSense and make sure Automatic DataSense metadata retrieval is selected and click OK. If that does not work, try restarting Anypoint Studio.

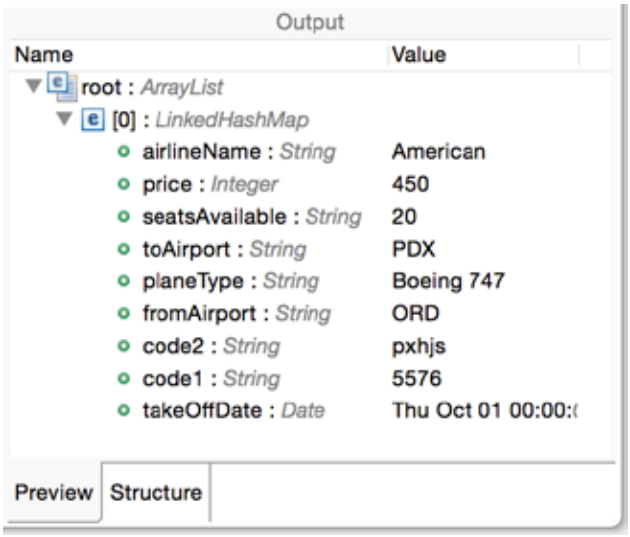
6. In the Transform section, leave the output set to Payload and of type application/java.
7. Delete the existing DataWeave expression (the curly braces) and set it to payload.
8. Look at the Preview tab in the Output section; you should see the output will be the same data type as the input but it does not have any sample values.



Add sample data and preview

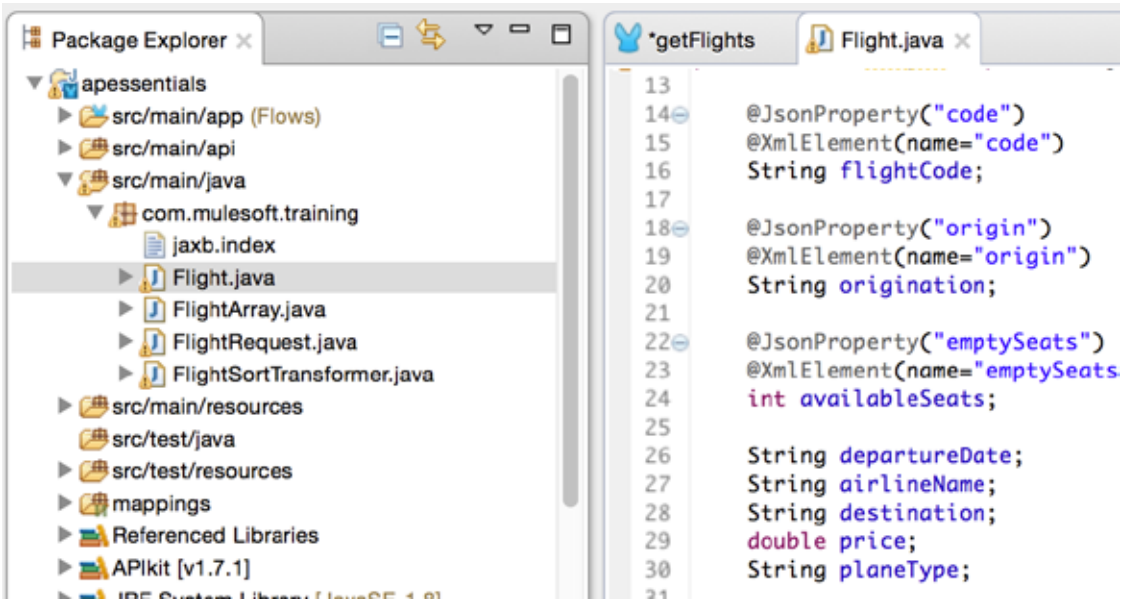
9. In the Input section of the Transform Message Properties view, click Payload: List<Map>.
10. Click the Sample Data button.
11. In the new payload tab, replace the “????” with sample values.

12. Look at the Preview tab in the Output section; you should see sample values.



Look at the desired output structure

13. Return to the Flight.java class in src/main/java and look at the names of the properties.



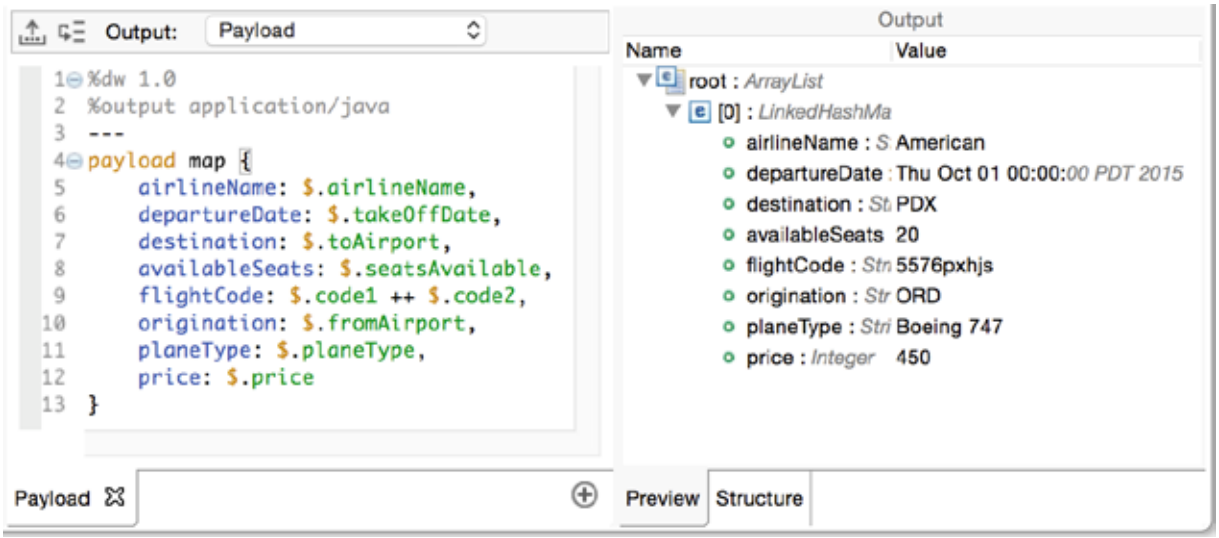
Change the output structure

14. Return to getAmericanFlightsFlow and change the DataWeave expression to payload map { }.
15. Inside the {}, define a new property called airlineName and map it to the payload airlineName property; be sure to use auto-completion!

16. Add additional properties with the following mappings:

    - departureDate < takeOffDate
    - destination < toAirport
    - availableSeats < seatsAvailable
    - flightCode < code1 concatenated with code2
    - origination < fromAirport
    - planeType < planeType
    - price < price

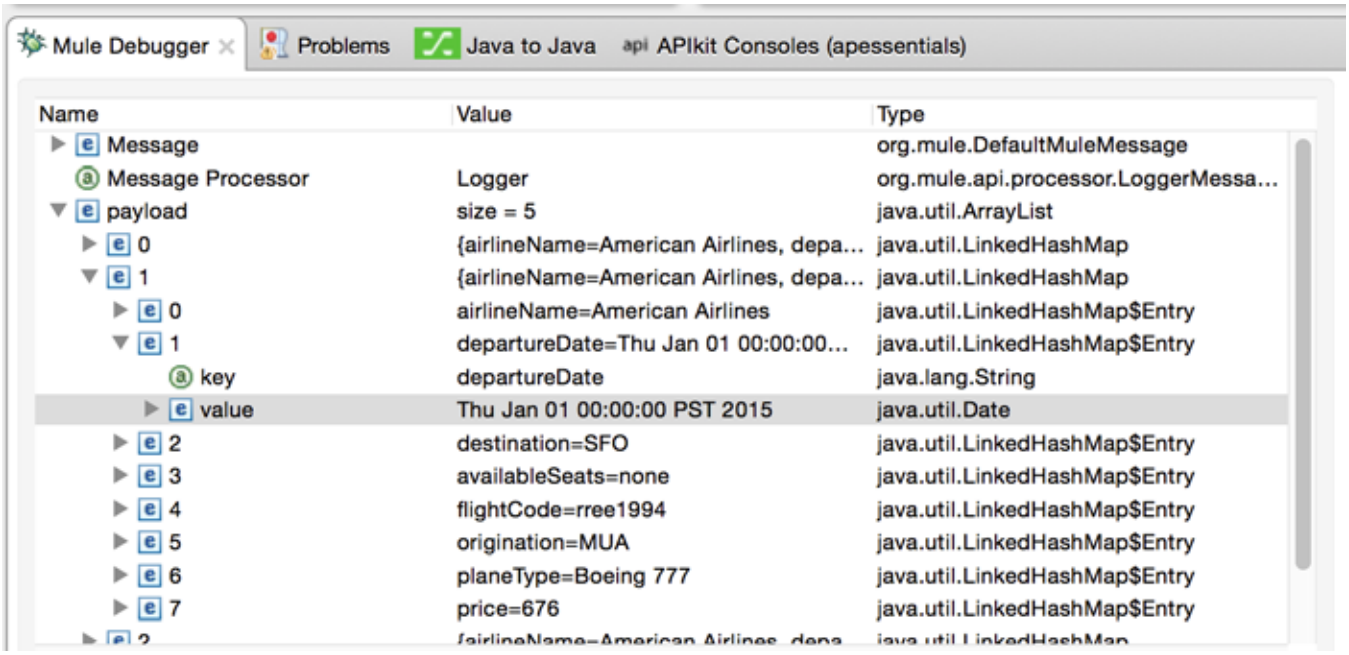
    Note: You can also copy this code from the course snippets.txt file.



Debug the application

17. Save the file and debug the application.
18. Make a request to http://localhost:8081/american.
19. Watch the payload value and step through to the Logger.
20. Drill-down into the data for the first and second flights in the payload and look at the data types and values.
21. Look at the values for availableSeats and see that for two of the flights it is equal to "none".

22. Look at the values for departureDate and see that they are of type Date.



23. Click Resume.

Format the data

24. In the DataWeave expression, format the price field as a number with the pattern “###.##”.

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

Note: You can also define and use a custom data type.

25. In the DataWeave expression, add logic to set the availableSeats field to a number unless it is equal to “none” in which case set it to 0.

```
availableSeats: $.seatsAvailable as :number unless
$.seatsAvailable=="none" otherwise 0,
```

26. In the DataWeave expression, format the departureDate field as a string with the pattern yyyy/MM/dd; this is the pattern used by the Delta and United web services.

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

27. In the header section of the Transform section, define a custom data type called flight that is a com.mulesoft.training.Flight Java object.

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

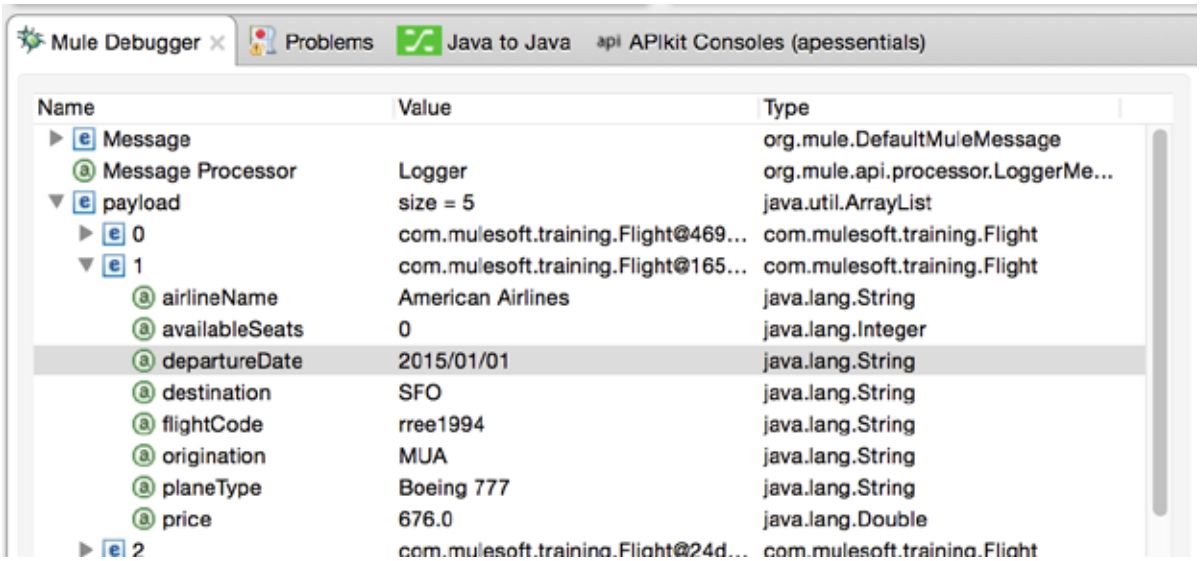
28. In the DataWeave expression, set the map objects to be of type flight.

```
payload. map {
      …
} as :flight
```

29. Look at the Preview tab; the objects should now be of type Flight and the departureDate should be formatted.

Debug the application

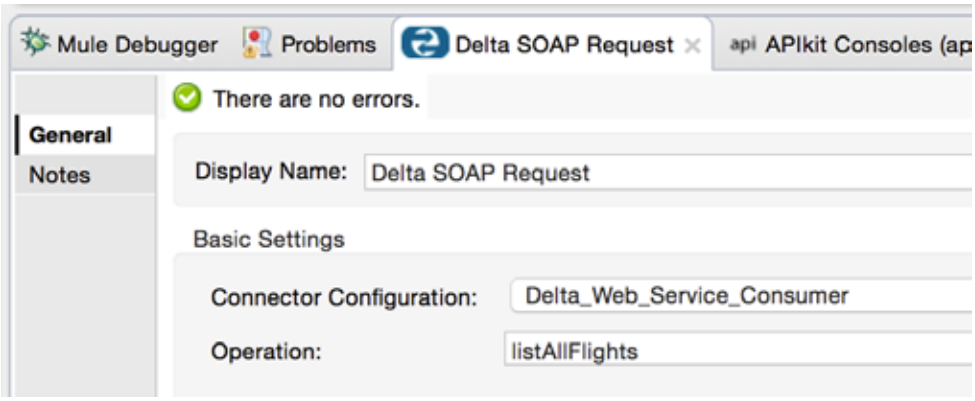
30. Save the file to redeploy the application.
31. Make a request to http://localhost:8081/american.
32. Watch the payload value and step through to the Logger.
33. Drill-down into the payload; you should see the flights are now of type Flight.
34. Drill-down into the data for the first and second flights in the payload and look at the data types and values and make sure availableSeats, price, and departureDate are set correctly.



35. Stop the Mule Debugger.

Change the Delta flow to get all flights

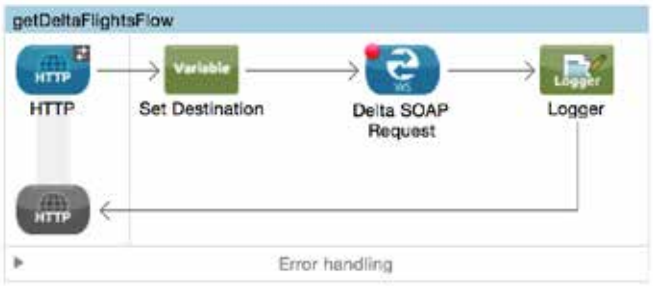
36. Locate getDeltaFlightsFlow.
37. Delete the String to XML<findFlight> DataMapper.
38. In the Delta SOAP Request Properties view, change the operation to listAllFlights.



Note: In the next walkthrough, you will use the Transform Message component and DataWeave to pass arguments to a SOAP web service.

Remove the existing transformation logic

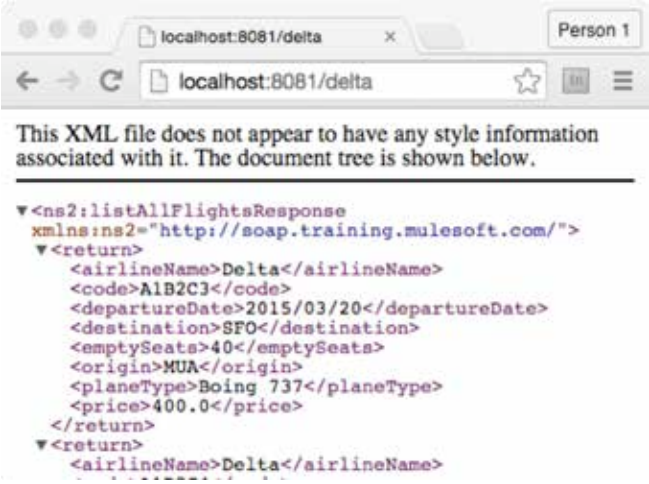
39. Delete the XML to JAXB Object transformer.
40. Delete the Collection of Flights transformer.
41. Delete the filter.



Review the Delta flight data

42. Run the application and make a request to http://localhost:8081/delta.

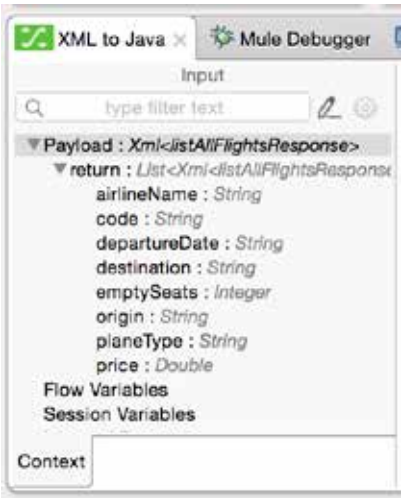
43. Look at the names of the XML elements and the value of the planeType element.



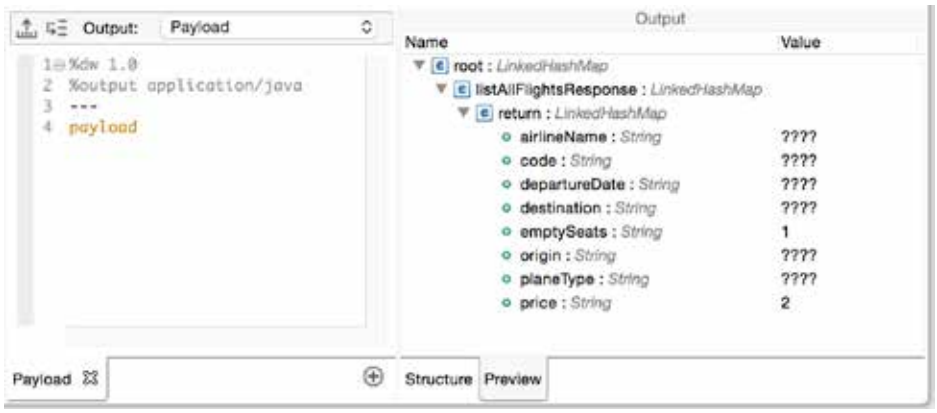
Transform the XML to Java using DataWeave

44. Add a Transform Message component between the Delta endpoint and the Logger.
45. Change the name of the component to XML to Java.

46. Look at the Input section of the XML to Java Properties view; the payload should automatically be set to a Xml<listAllFlightsResponse> with correct properties because the Web Service Consumer connector configuration is DataSense enabled by default.

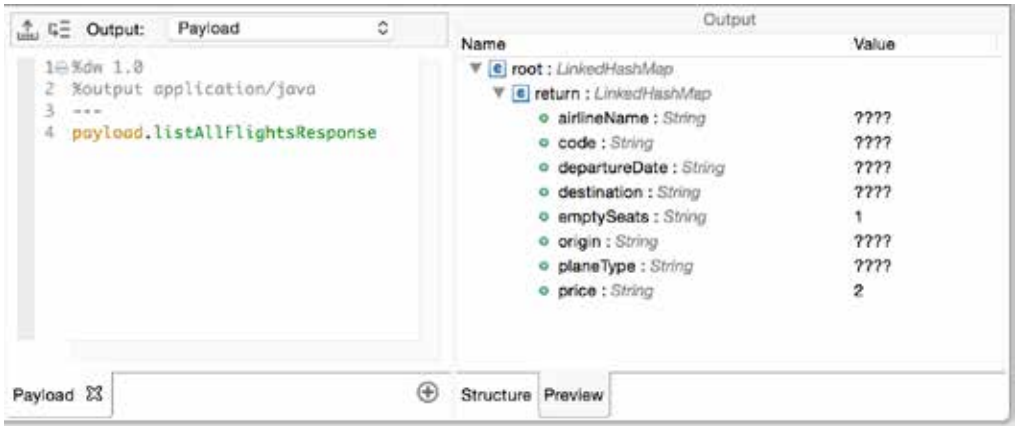


47. In the Transform section, leave the output set to Payload and of type application/java.
48. Delete the existing DataWeave expression (the curly braces) and set it to payload.
49. In the Preview tab, look at the resulting output structure.

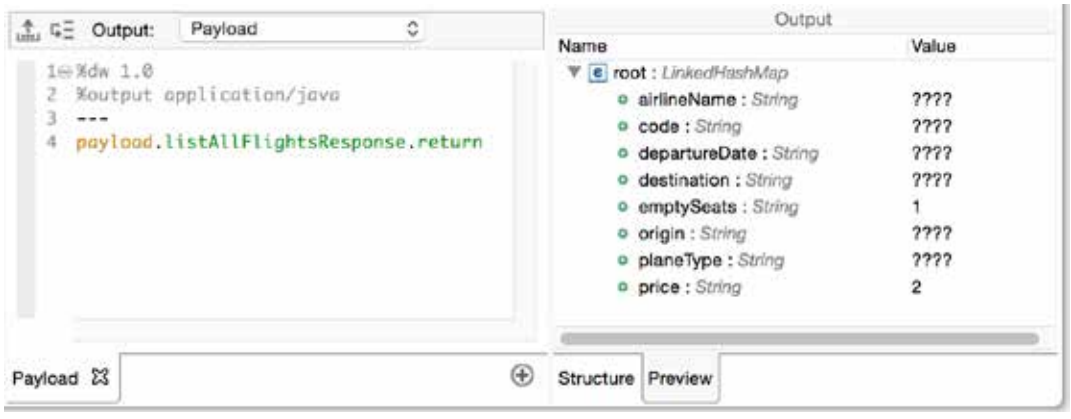


Change the output structure

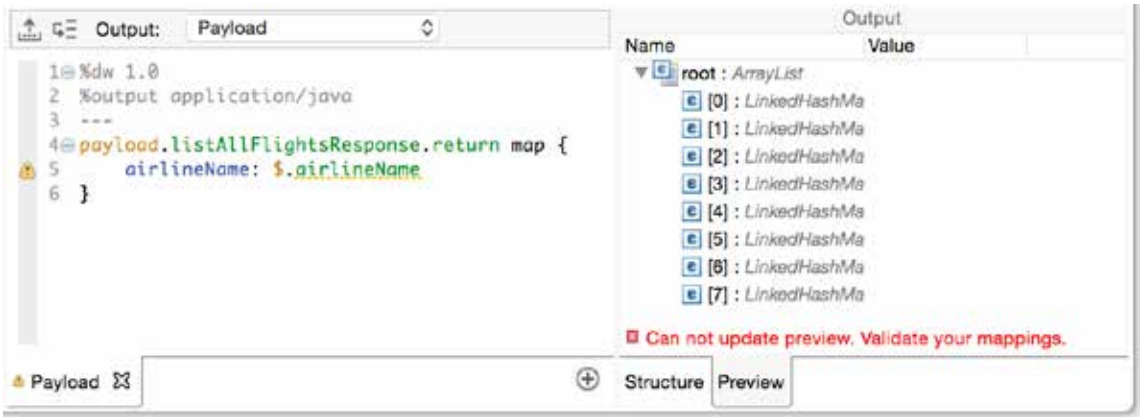
50. Change the DataWeave expression to payload.listAllFlightsResponse and look at the preview.



51. Change the DataWeave expression to payload.listAllFlightsResponse.return and look at the preview.



52. Modify the DataWeave expression to use the map operator to populate a new airline property with the value of the airlineName property in each object; you should get a message that the preview cannot be updated.



53. Change the DataWeave expression to payload.listAllFlightsResponse.*return.

54. Add additional properties with the following mappings:

- departureDate < departureDate
- destination < destination
- availableSeats < emptySeats as a number
- flightCode < code
- origination < origin
- planeType < planeType with the string Boing replaced with Boeing
- price < price and format it as a number with the pattern “###.##”

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

Output: Payload

```
%dw 1.0
%output application/java
---
payload.listAllFlightsResponse.*return map {
    airlineName: $.airlineName,
    departureDate: $.departureDate,
    destination: $.destination,
    availableSeats: $.emptySeats as :number,
    flightCode: $.code,
    origination: $.origin,
    planeType: $.planeType replace /(Boing)/ with "Boeing",
    price: $.price as :number {format: "###.##"}
}
```

55. In the header section of the Transform section, define a custom data type called flight that is a com.mulesoft.training.Flight Java object.

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

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

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

Debug the application

57. Save the file and debug the application.
58. Make a request to http://localhost:8081/delta.
59. Watch the payload value and step through to the Logger.
60. Drill-down into the data and make sure it all mapped correctly.

Mule Debugger | Problems | XML to Java | APIkit Consoles (apessentials)

Name	Value	Type
Message		org.mule.DefaultMuleMessage
Message Processor	Filter No Results	org.mule.routing.MessageFilter
payload	size = 12	java.util.ArrayList
0	com.mulesoft.training.Flight@53b...	com.mulesoft.training.Flight
1	com.mulesoft.training.Flight@62f...	com.mulesoft.training.Flight
airlineName	Delta	java.lang.String
availableSeats	10	java.lang.Integer
departureDate	2015/02/11	java.lang.String
destination	LAX	java.lang.String
flightCode	A1B2C4	java.lang.String
origination	MUA	java.lang.String
planeType	Boeing 737	java.lang.String
price	199.99	java.lang.Double

61. Stop the Mule Debugger.

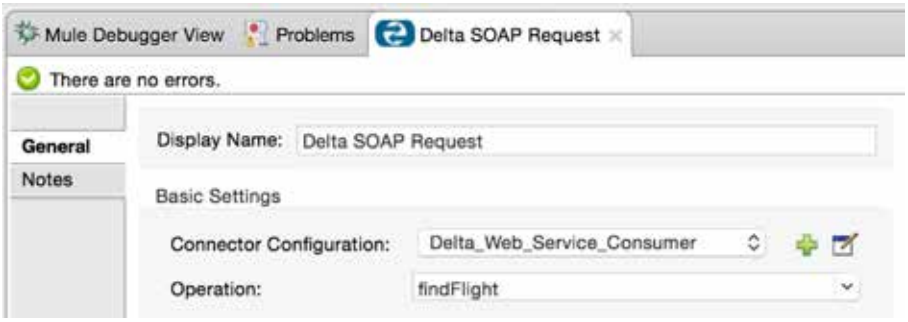
Walkthrough 12-6: Pass arguments to a SOAP web service

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

- Return the flights for a specific destination instead of all the flights.
- Change the web service operation invoked to one that requires a destination as an input argument.
- Use DataWeave to pass an argument to a web service operation.
- Create a variable to set the destination to a dynamic query parameter value.

Call a different web service operation

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



3. In the Transform Message Properties view, change the DataWeave expression to use the descendant operator so that the transformation will work with findFlightResponse or listAllFlightsResponse.

```
payload..*return map {
```

Debug the application

4. Apply the changes and debug the application.
5. Make a request to http://localhost:8081/delta.
6. Return to Anypoint Studio and step through the flow; you should get an error.

7. Look at the console; you should a SoapFaultException because you are calling an operation that requires parameters but you have not passed it any.

```
ERROR 2015-07-08 19:05:22,005 [[apessentials-mod04].HTTP_Listener_Configuration.worker.01] org.mule.exception.DefaultMessagingExce
ptionStrategy:
********************************************************************************
Message               : No binding operation info while invoking unknown method with params unknown.. Message payload is of type:
NullPayload
Type                  : org.mule.module.ws.consumer.SoapFaultException
Code                  : MULE_ERROR--2
Payload               : {NullPayload}
JavaDoc               : http://www.mulesoft.org/docs/site/current3/apidocs/org/mule/module/ws/consumer/SoapFaultException.html
********************************************************************************
```

8. Stop the debugger.

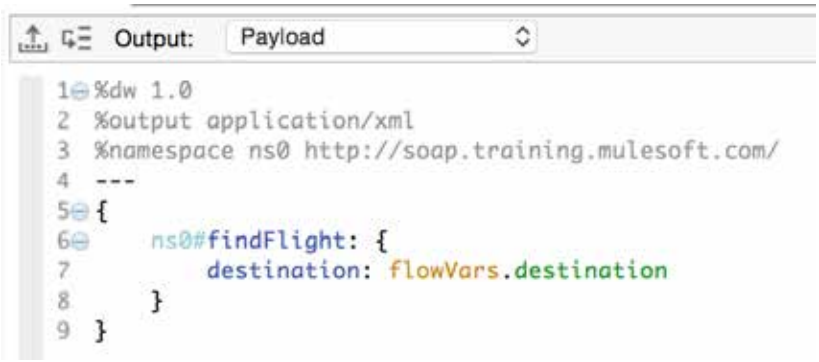
Use DataWeave to pass parameters to the web service

9. Add a Transform Message component to the left of the Delta SOAP Request endpoint.
10. Change its name to Pass Destination.

11. Look at the Pass Destination Properties view.

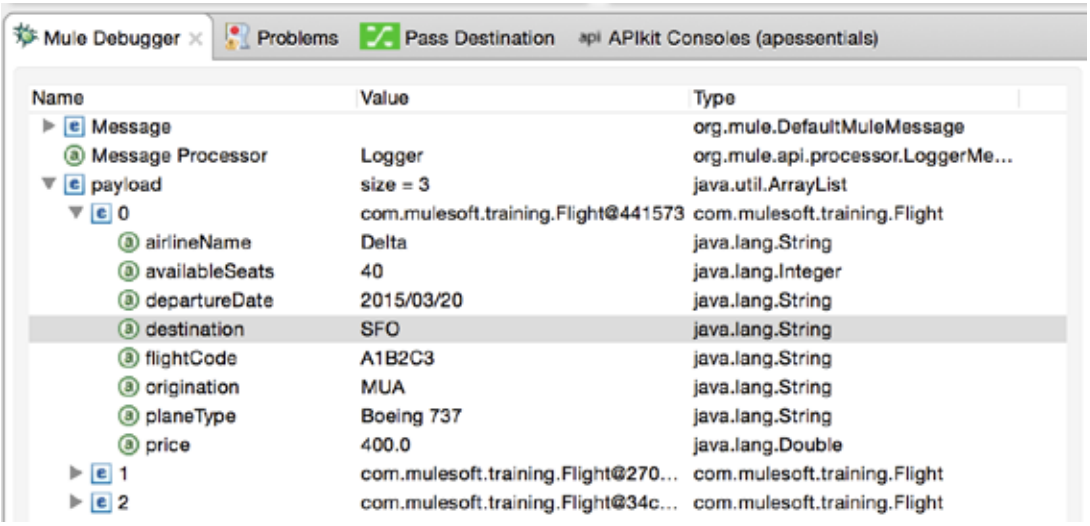
12. In the DataWeave expression, set the destination to SFO and look at the preview.

13. Replace the literal of SFO with a reference to the destination flow variable.



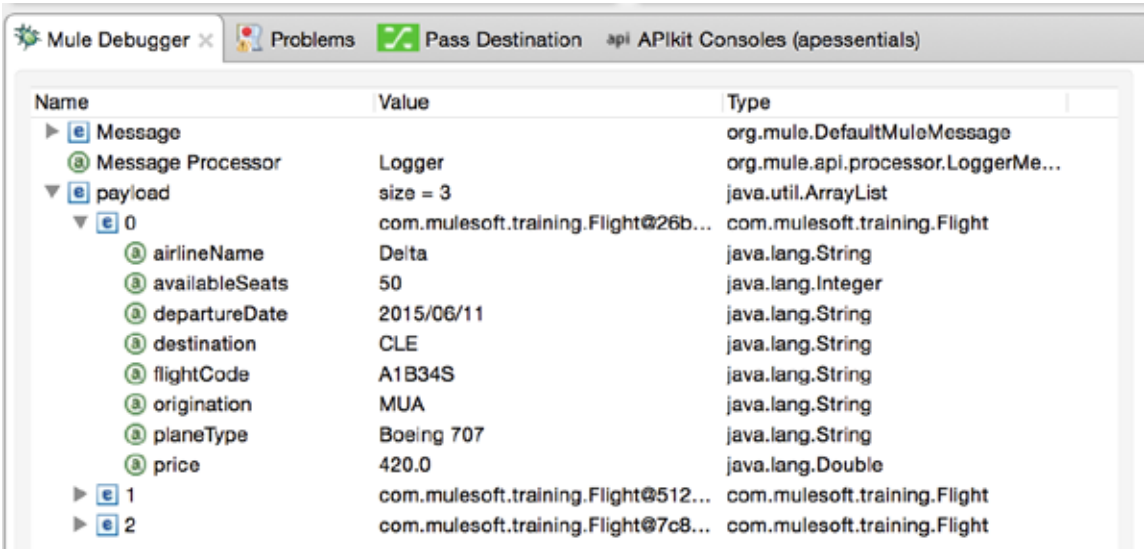
Debug the application

14. Save the file and debug the application.
15. Make a request to http://localhost:8081/delta.
16. In the Mule Debugger, step though to the Logger; you should now see only the SFO flights.



17. Click Resume.

18. Make another request to http://localhost:8081/delta?code=CLE and step through to the Logger; you should now only see flights to CLE.



19. Click Resume.

Walkthrough 12-7: Transform a data source to which you add custom metadata

In this walkthrough, you will work with the United flight data. You will:

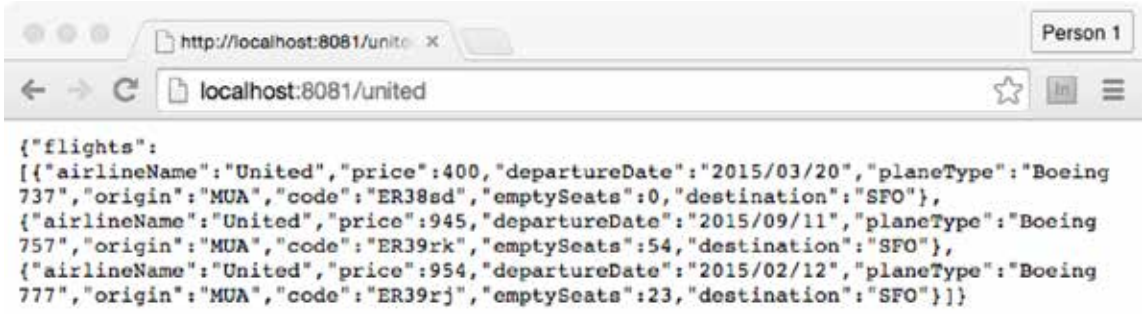
- Add custom metadata to an HTTP Request endpoint.
- Use DataWeave to transform the United flight data from JSON to a collection of POJOs.

Remove the existing United transformation logic

1. Locate getUnitedFlightsFlow.
2. Delete the JSON to Object transformer.
3. Delete the Collection of Flights transformer.

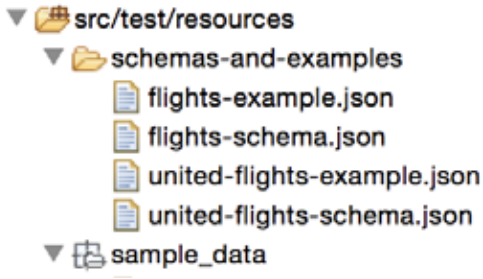
Review the United flight data

4. Run the application and make a request to http://locahost:8081/united.
5. Look at the names of the object keys.

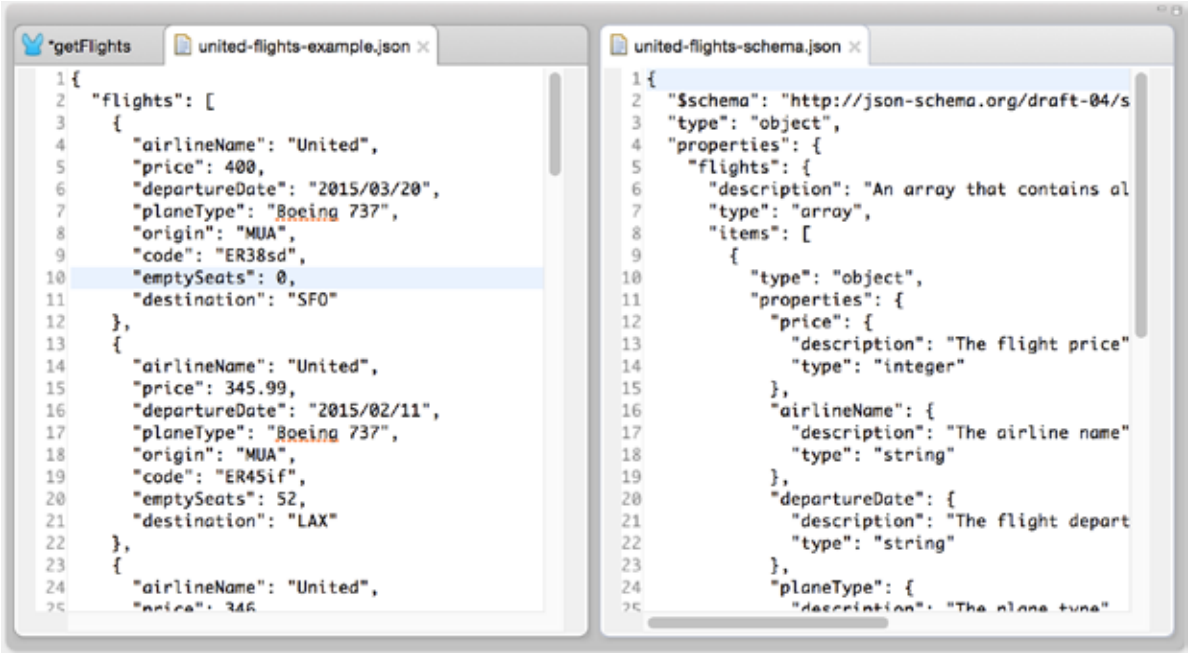


Look at the flights schema and example files

6. On your computer, navigate to the resources folder in the student files folder for the course: MUEssentials3.7_studentFiles_{date}.
7. Copy and paste (or drag) the schemas-and-examples folder into your project's src/test/resources folder.



8. Open the united-flights-schema.json and united-flights-example.json files and examine the code.

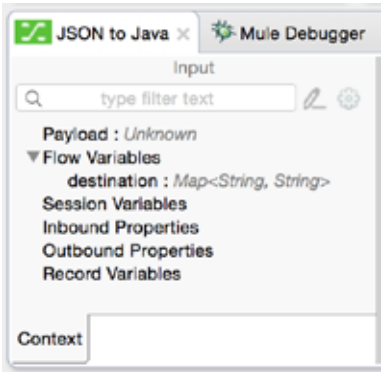


Transform the XML to Java using DataWeave

9. Add a Transform Message component between the filter and the Logger.

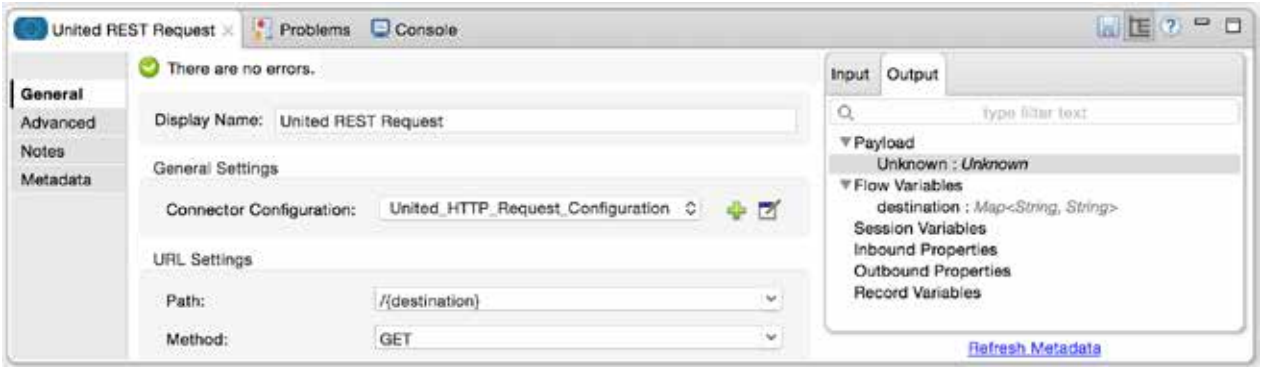
10. Change the name of the component to JSON to Java.

11. In the JSON to Java Properties view, leave the output set to Payload and of type application/java.
12. Look at the Input section of the Transform Message Properties view; the payload is of type unknown.



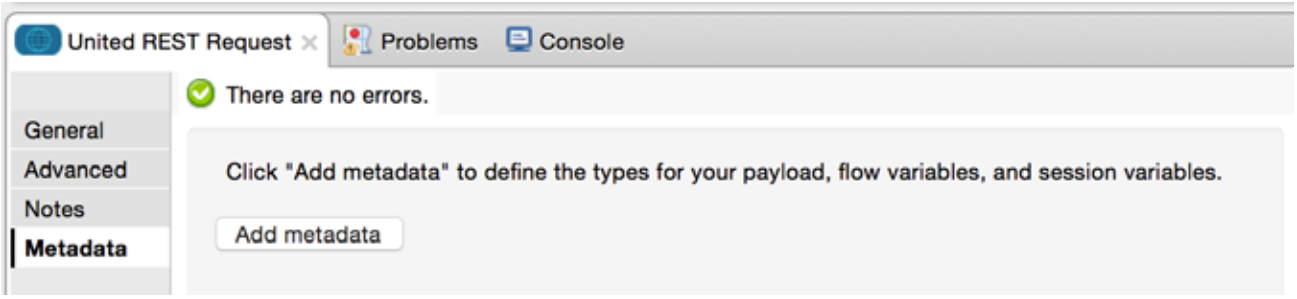
Add metadata to the HTTP Request endpoint

13. In the United REST Request Properties view, click the Output tab in the Metadata section; you should see the payload is of type unknown.

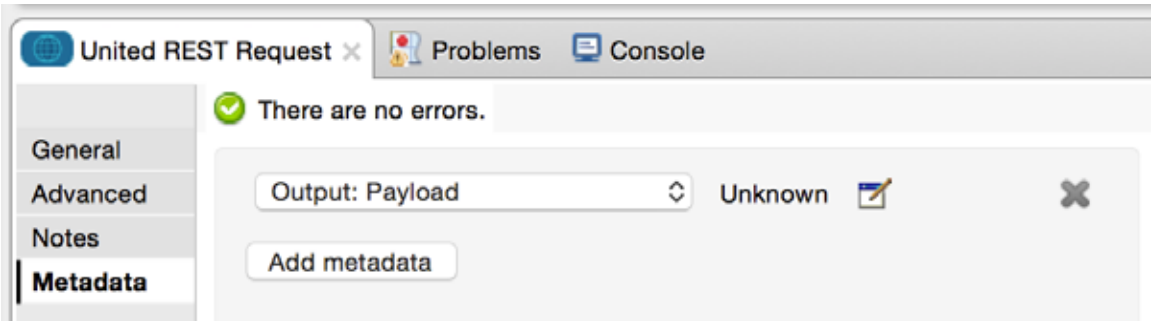


14. Click the Metadata link in the left-side navigation.

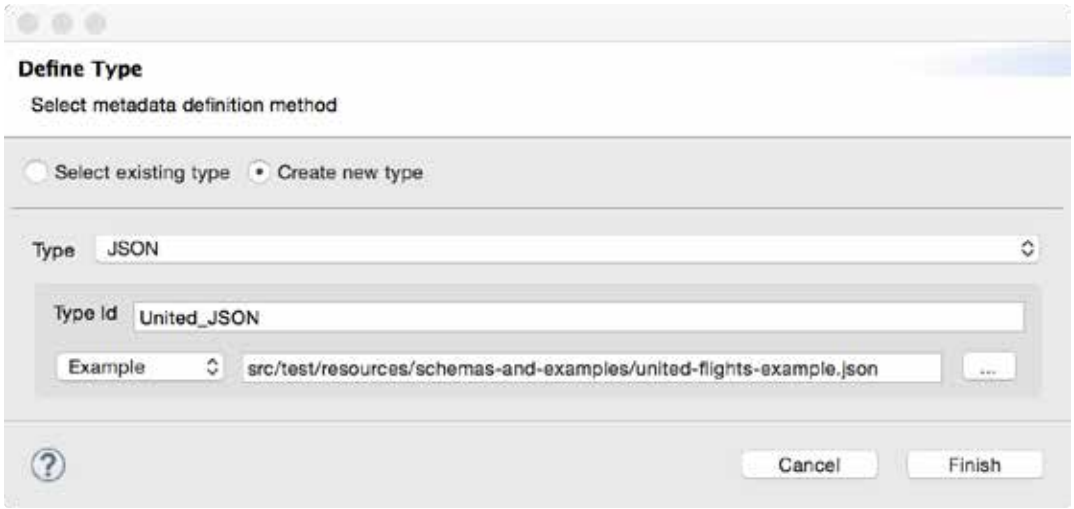
15. Click the Add metadata button.



16. In the drop-down menu that appears, select Output: Payload.
17. Click the Edit button located to the right of the drop-down menu.



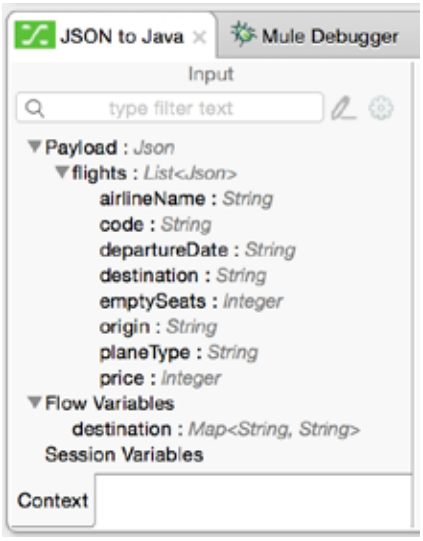
18. In the Define Type dialog box, select Create new type.
19. Set the type to JSON.
20. Set the type id to United_JSON.
21. In the drop-down menu, select Example.
22. Click the browse button.
23. In the Open dialog box browse to src/test/resources/schemas-and-examples/united-flights-example.json and click Open.
24. In the Define Type dialog box, click Finish.



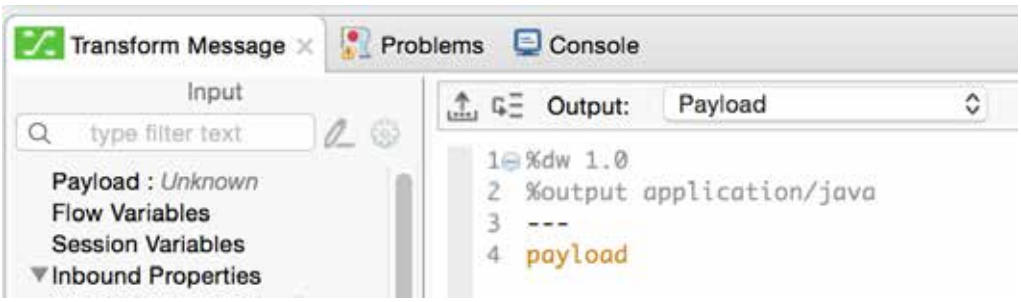
25. Look at the Output tab in the Metadata section again; you should now see the payload is of type JSON with a flights property that is a list of objects with specific fields.

Transform the JSON to a collection of POJOs

26. Return to the JSON to Java component Properties view.
27. Look at the Input section again; the payload should now be of type Json.
28. Expand the flights List; you should see the the field names of each object.



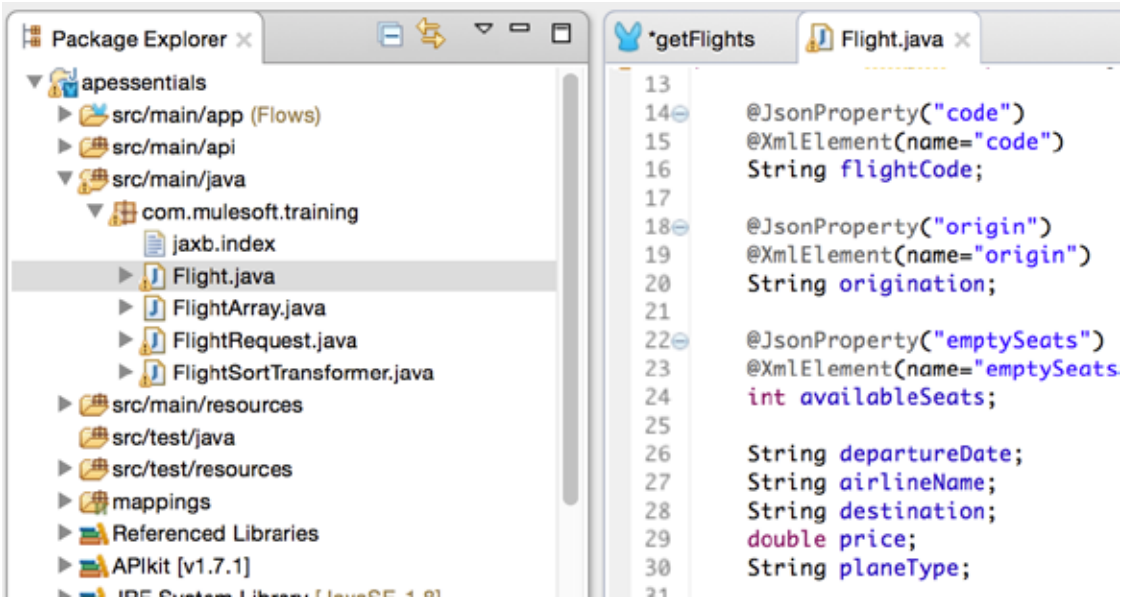
29. Delete the existing DataWeave expression (the curly braces) and set it to payload.



30. Look at the Preview tab and examine the output structure.

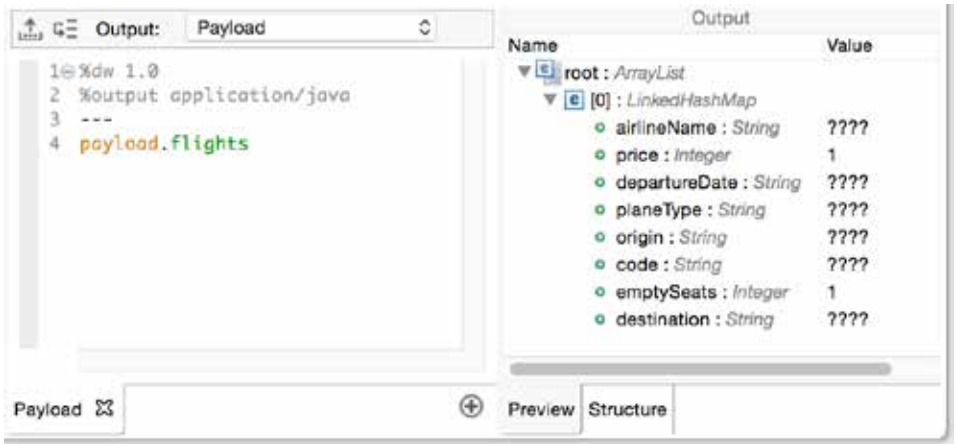
Look at the desired output structure

31. Return to the Flight.java class in src/main/java and look at the names of the properties.



Change the output structure

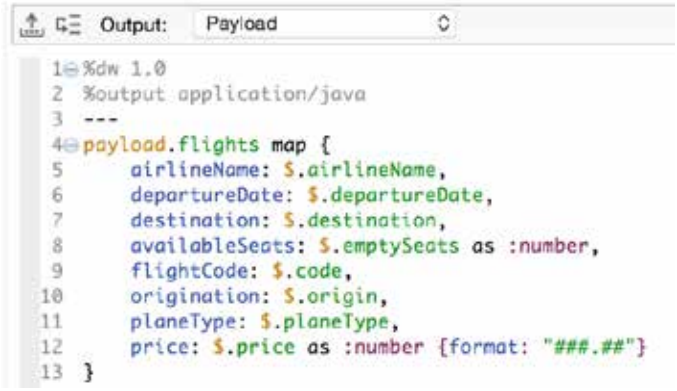
32. Change the DataWeave expression to payload.flights; the preview should now be a List of Map objects.



33. Modify the DataWeave expression to use the map operator.
34. In the transformation function, set the following mappings:

    - airlineName < airlineName
    - departureDate < departureDate
    - destination < destination
    - availableSeats < emptySeats as a number
    - flightCode < code
    - origination < origin
    - planeType < planeType
    - price < price and format it as a number with the pattern “###.##”

    Note: If you want, you can copy these lines of code from the Delta DataWeave expression.



35. In the header section of the Transform section, define a custom data type called flight that is a com.mulesoft.training.Flight Java object.

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

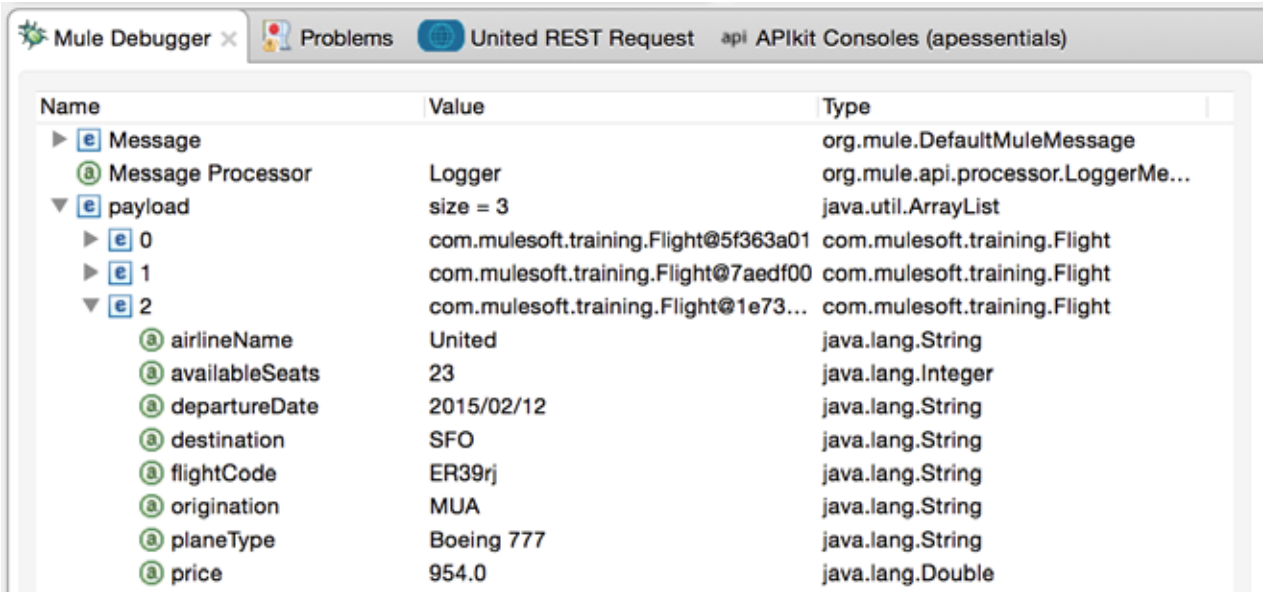
36. In the DataWeave expression, set the map objects to be of type flight.

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

Debug the application

37. Save the file and debug the application.
38. Make a request to http://localhost:8081/united.
39. Watch the payload value and step through to the Logger.

40. Drill-down into the data and make sure it all mapped correctly.



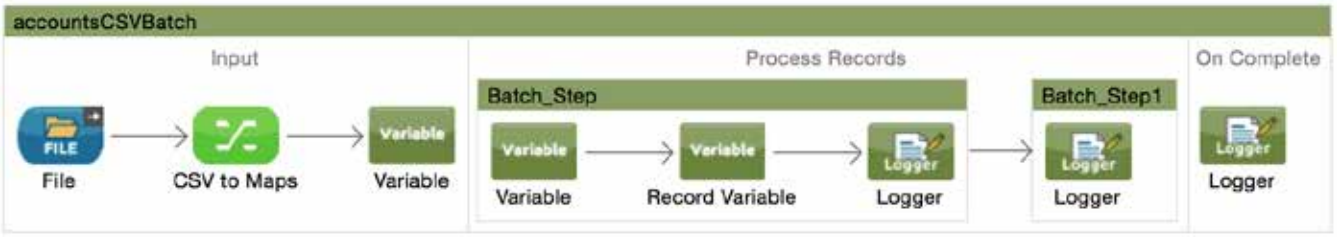
41. Stop the Mule Debugger.

Note: At this point, DataWeave is being used for all of the flight transformations in the MUA application except the last one in getFlightsFlow that sorts the flights by price and returns them as JSON. A good exercise for you as as student would be to replace these two transformers with a single Transform Message component that accomplishes the same transformation. A JSON sample and schema for the output expected by the form are included in the schemas-and-examples folder.

Walkthrough 12-8: Transform a CSV file

In this walkthrough, you will convert a CSV file to a collection of Java objects. You will:

- Add metadata to a File endpoint.
- Read a CSV file and use DataWeave to convert it to a collection of objects.

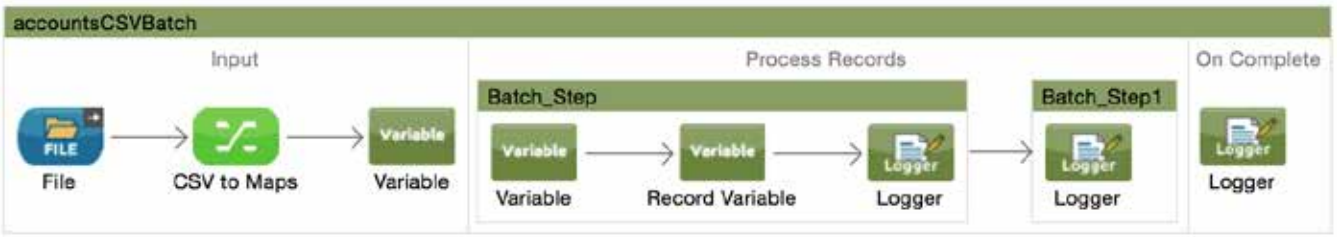


Remove the existing transformation

1. Open accounts.xml and locate accountsCSVBatch.
2. Delete the CSV to Map DataMapper.
3. In the Delete grf file dialog box, click No; the GRF file is also being used by the DataMapper in the getCSVAccountsFlow.

Transform the CSV to Java using DataWeave

4. Add a Transform Message component between the File and the For Each scope.
5. Change the name of the component to CSV to Maps.

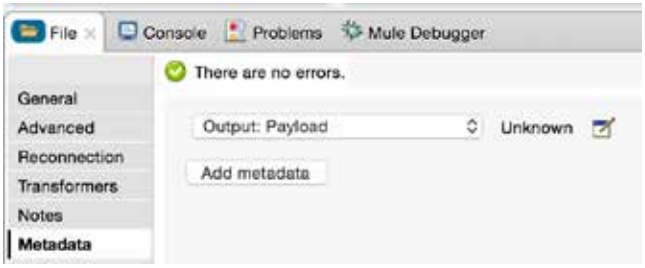


6. In the Input section of the CSV to Maps Properties view, make sure the Payload is specified as a List<Csv> with appropriate fields.

7. Look at the Input section of the CSV to Maps Properties view; you should see the payload is set to Unknown.

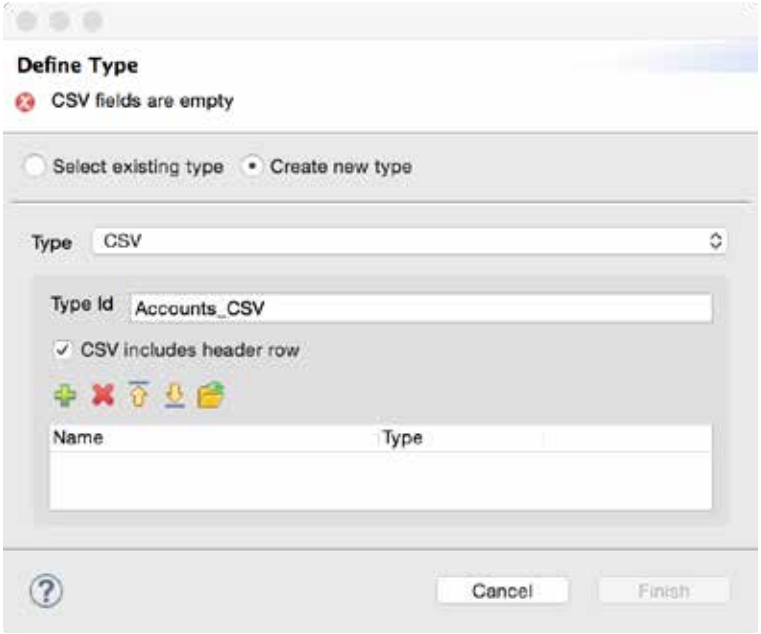
Add File endpoint metadata

8. In the File Properties view, click Metadata in the left-side navigation.
9. Click the Add metadata button.
10. In the drop-down menu that appears, make sure Output: Payload is selected.

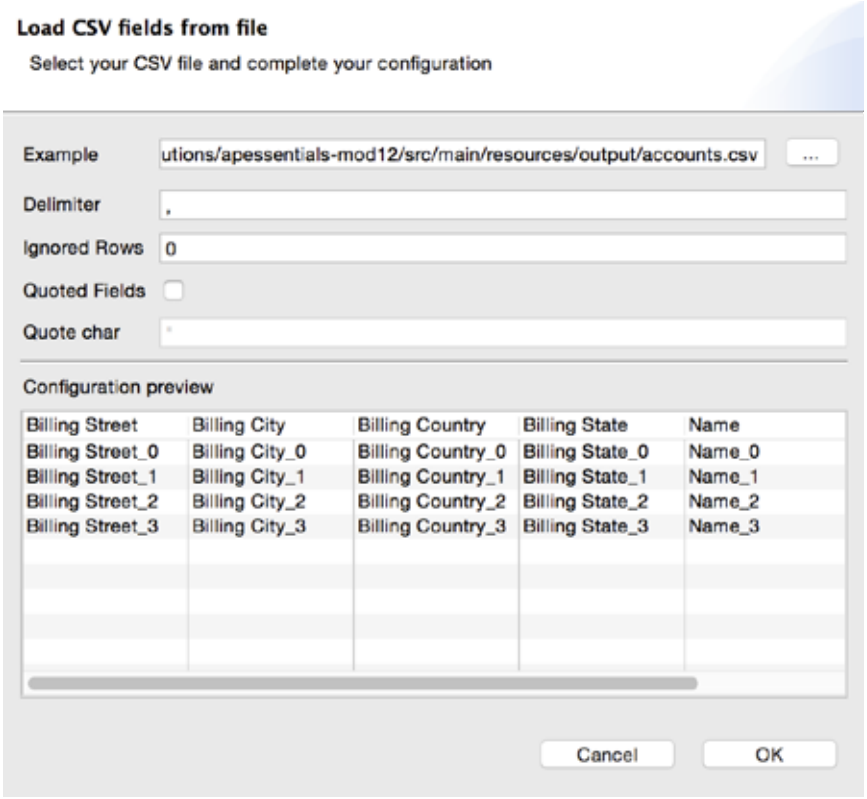


11. Click the Edit button next to the drop-down menu.
12. In the Define Type dialog box, select Create new type.
13. Change the type to CSV.
14. Set the Type Id to Accounts_CSV.
15. Make sure CSV includes header row is checked.

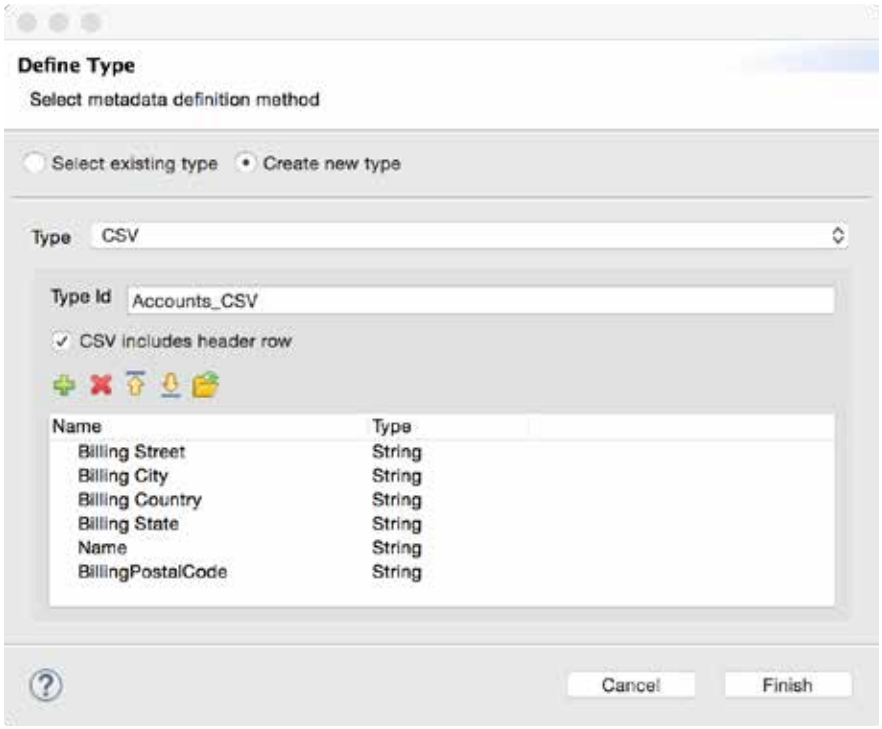
16. Click the Load from example button (the folder icon).



17. In the Load CSV fields from file dialog box, click the Browse button next to Example.
18. In the Select CSV example dialog box, browse to the project's src/main/resources/output folder, select accounts.csv, and click Open.
19. In the Load CSV fields from file dialog box, click OK.

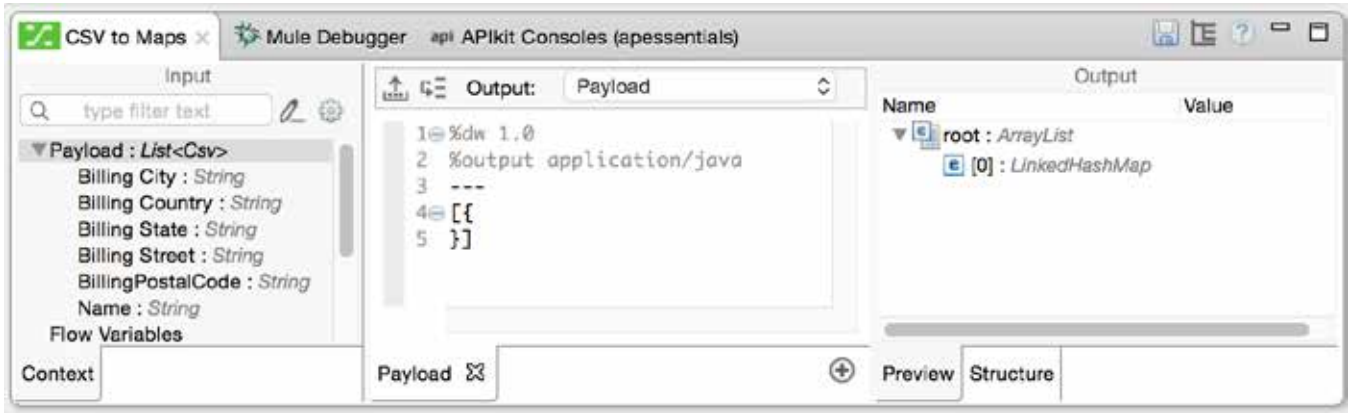


20. In the Define Type dialog box, click Finish.



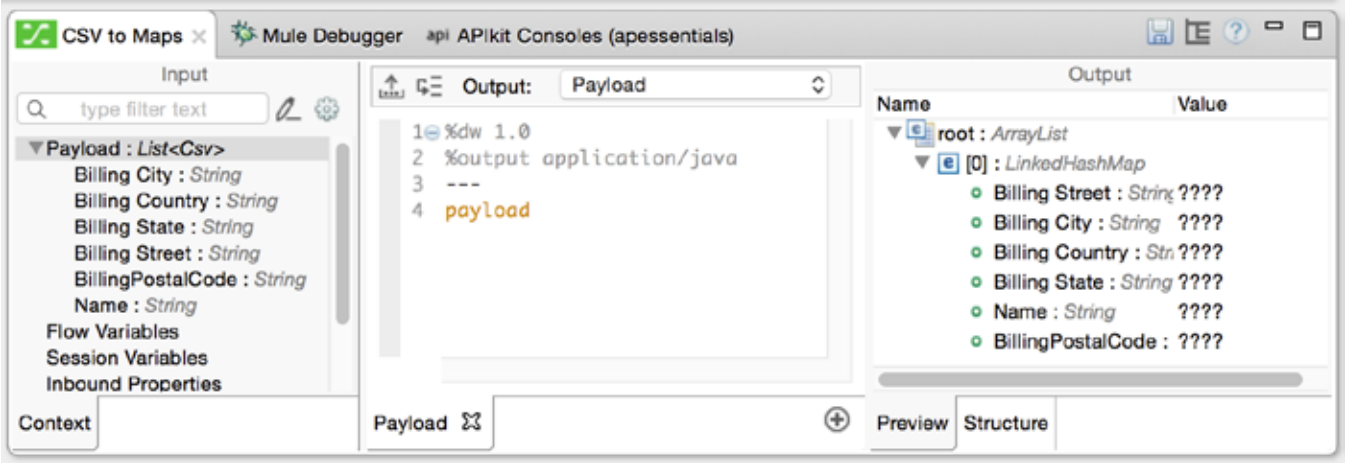
Use DataWeave to convert a CSV file to a collection of objects

21. Return to CSV to Maps Properties view; in the Input section, the payload should now be specified as a List<Csv> with appropriate fields.
22. Look at the Preview tab; the output should an ArrayList of Maps.



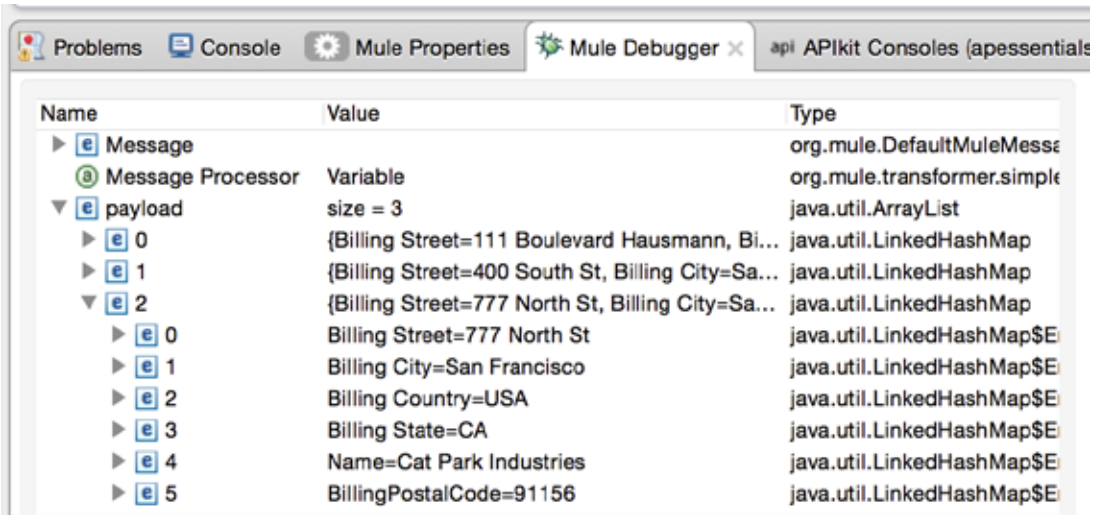
23. Change the DataWeave expression to payload.

24. Look at the preview.



Debug the application

25. Add a breakpoint to the Variable transformer.
26. Save the file and debug the application.
27. Move accounts.csv from src/main/resources/output to src/main/resources/input.
28. Return to the Mule Debugger in Anypoint Studio; you should see the CSV data has been converted to an ArrayList of Map objects.



29. Stop the Mule runtime.