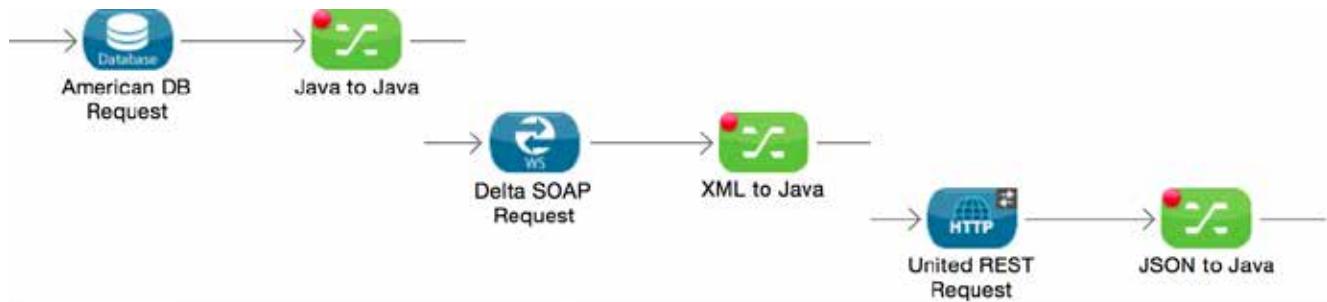


# Module 12: Transforming Data with DataWeave



Screenshot of the Mule Studio DataWeave editor for the "XML to Java" transformation:

**Input:**

```
<?xml version="1.0"?>
<findFlightResponse>
  <return>
```

**Payload:**

```
1@%dw 1.0
2 %output application/java
3 ---
4@payload.findFlightResponse.*return map {
5   airlineName: $.airlineName,
6   departureDate: $.departureDate,
7   destination: $.destination,
8   emptySeats: $.emptySeats as :number,
9   flightCode: $.code,
10  origination: $.origin,
11  planeType: $.planeType replace /({Boeing})/ with "Boeing"
12  price: $.price as :number {format: "###.##"}
13 }
```

**Output:**

Name	Value
root : ArrayList	
[0] : LinkedHashMap	
◦ airlineName : String	????
◦ departureDate : String	????
◦ destination : String	????
◦ emptySeats : Integer	1
◦ flightCode : String	????
◦ origination : String	????
◦ planeType : String	????
◦ price : Integer	2

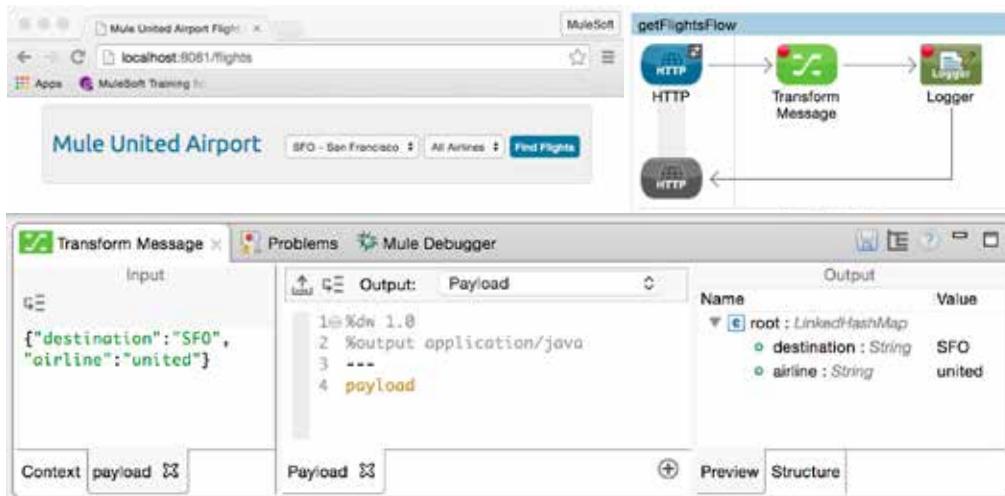
In this module, you will learn:

- 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.*

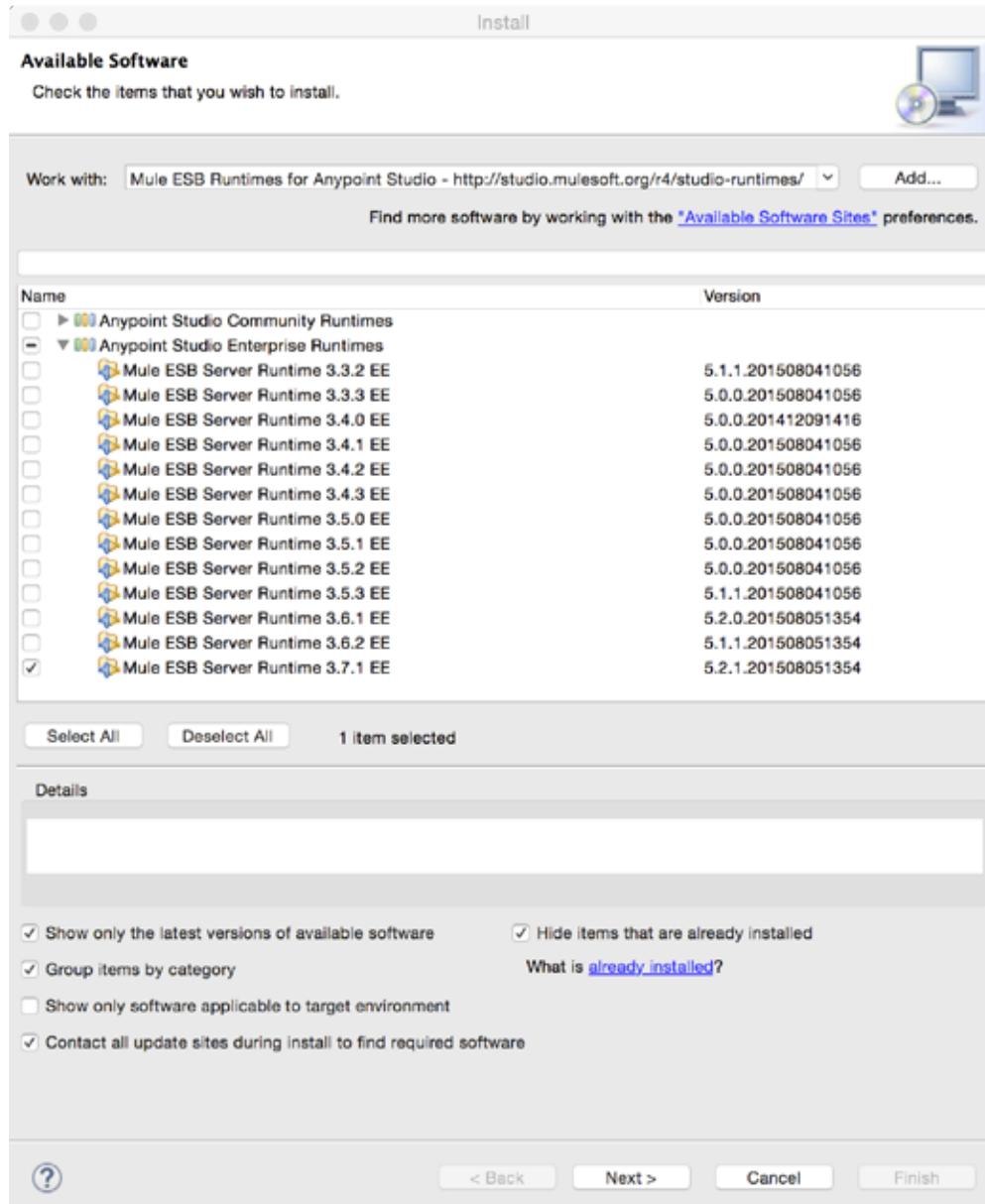
### Update Anypoint Studio

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.

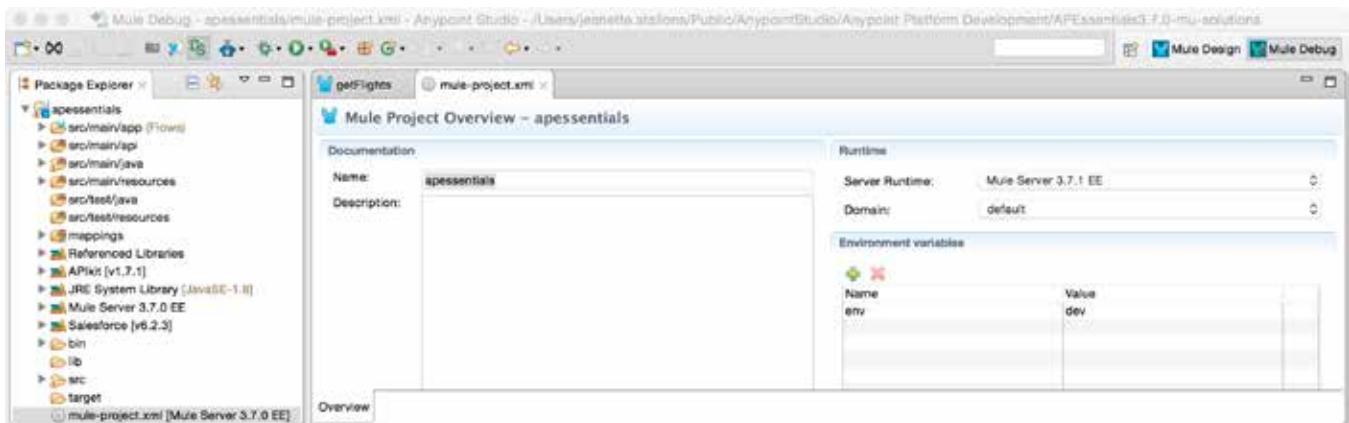
7. Click the arrow to expand Anypoint Studio Enterprise Runtimes.
8. If you see a 3.7.X runtime later than the one you are using, select it and click Next.
9. If you see a 3.7.X runtime later than the one you are using, click Cancel and skip the next several steps.



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

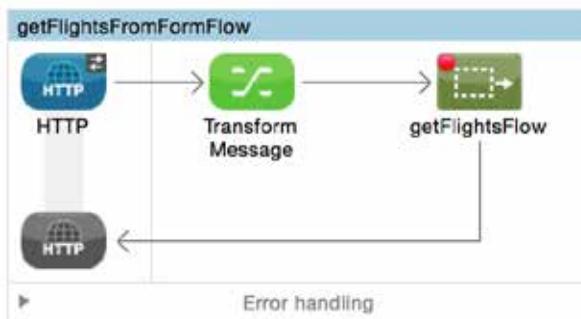
17. Return to getFlights.xml.
18. Locate getFlightsFromFormFlow.
19. Delete the JSON to Object transformer.
20. Add a breakpoint to the Flow Reference component.
21. Save the file and debug the application.
22. Make a request to <http://localhost:8081/flights> and click the Find Flights button.
23. Look 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.

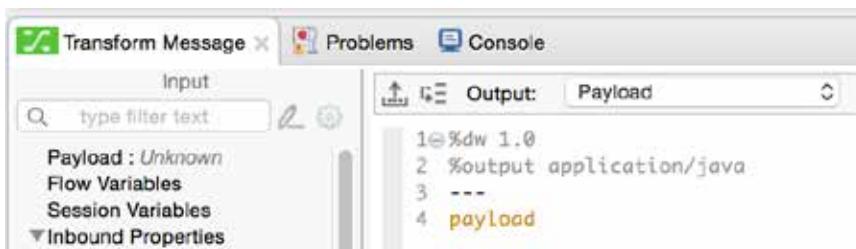


27. In the Transform Message Properties view, look at the Input, Transform, and Output sections.

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.



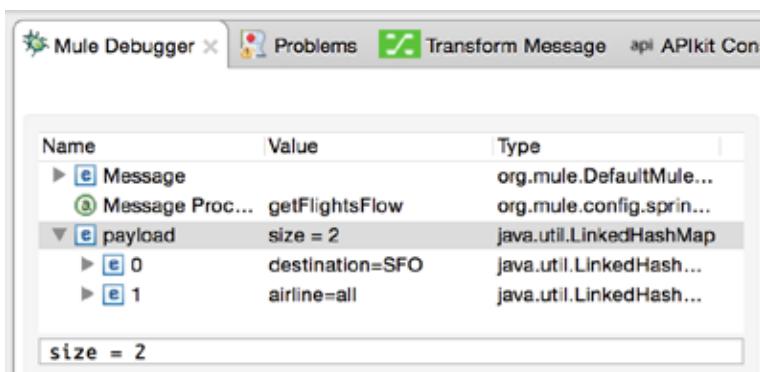
## Debug the application

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.

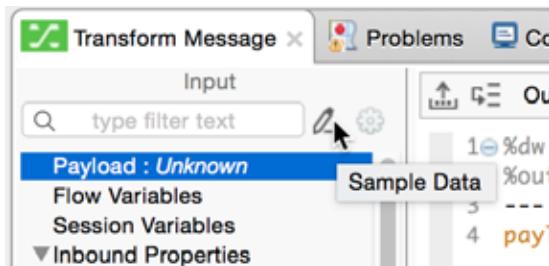


35. Click the Resume button.

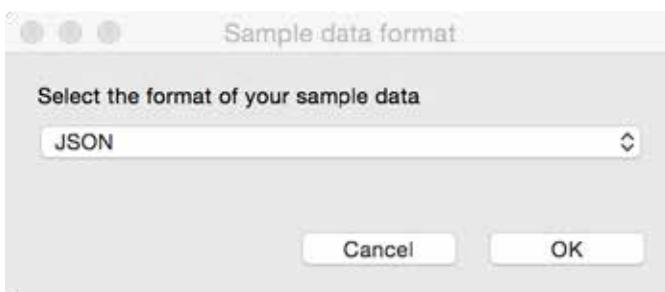
## Add sample data and preview

36. In the Input section of the Transform Message Properties view, click Payload: Unknown.

37. Click the Sample Data button.



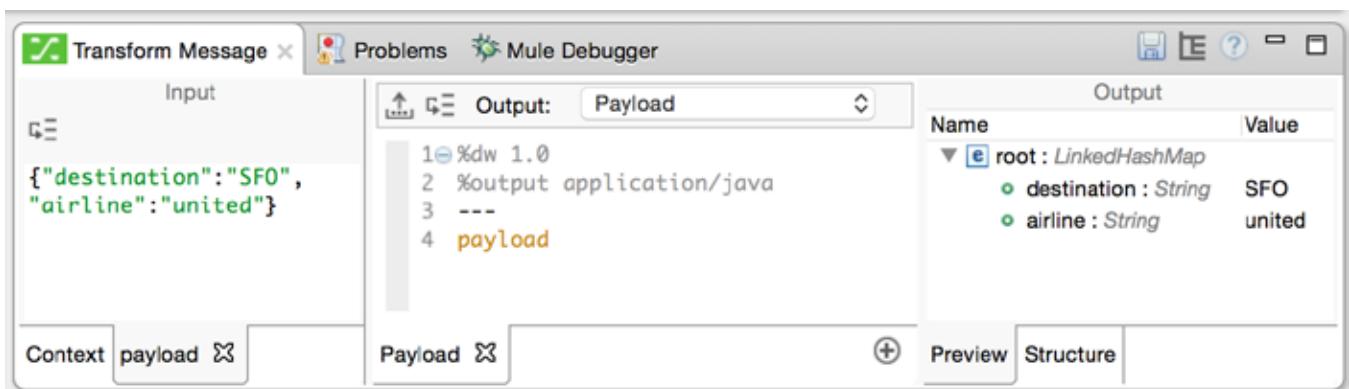
38. In the Sample data format dialog box, select JSON and click OK.



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.

Transform Message

Input:

```
{"destination": "SFO", "airline": "united"}
```

Output: Payload

```
{ "destination": "SFO", "airline": "united" }
```

Context payload

Payload

Preview Structure

Can not update preview. Validate your mappings.

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

Transform Message

Input:

```
{"destination": "SFO", "airline": "united"}
```

Output: application/json

```
{ "destination": "SFO", "airline": "united" }
```

Context payload

Payload

Preview Structure

## 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.

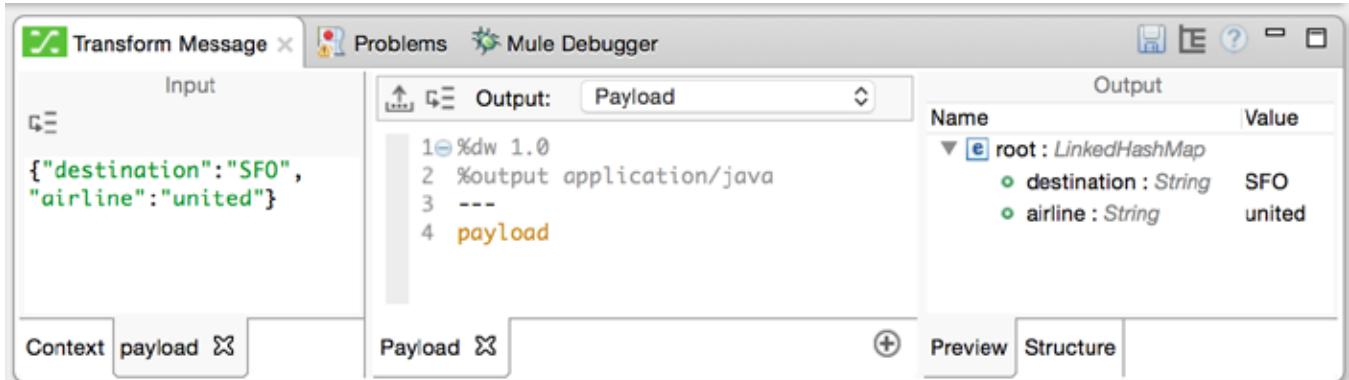
Name	Value	Type
Message	org.mule.DefaultMuleMessage	
Message Processor	Logger	org.mule.api.processor.LoggerMessageProcessor
payload	com.mulesoft.weave.mule.WeaveMessageProcessor\$WeaveOutputHandler	com.mulesoft.weave.mule.WeaveMessageProcessor\$WeaveOutputHandler
encodingName	UTF-8	java.lang.String
engine	com.mulesoft.weave.engine.Engine@fb32ca7	com.mulesoft.weave.engine.Engine
outputModule	com.mulesoft.weave.module.JsonModule	com.mulesoft.weave.module.JsonModule
readers	Map(payload -> com.mulesoft.weave.readers.JsonReader)	scala.collection.immutable.Map\$Map1
variableContext	Map(lookup -> com.mulesoft.weave.modules.lookup.LookupVariable)	scala.collection.immutable.HashMap\$HashTrieMap

47. Stop the Mule runtime.

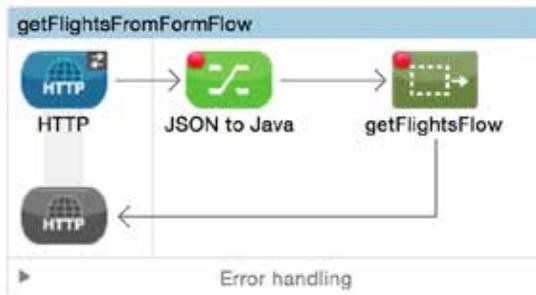
*Note: If you click Resume, you will get an error in the application because the other flows are no longer getting the correct type of data.*

## Change the output type

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



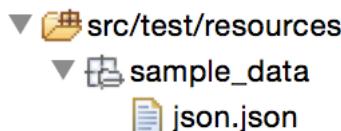
49. Click the name of the Transform Message component in the canvas and change its name to JSON to Java.



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.

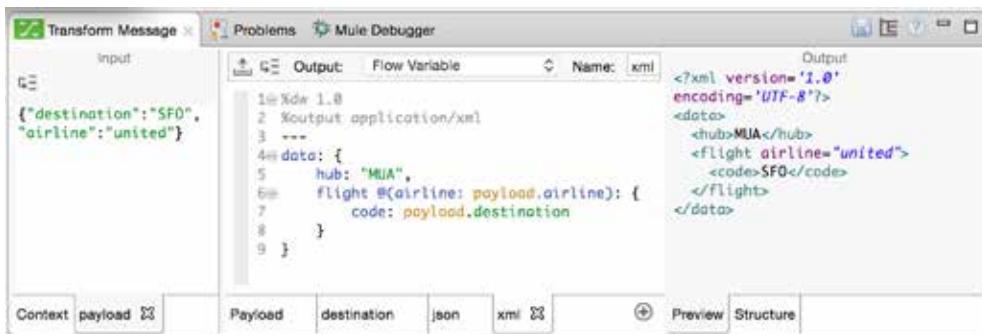
53. Locate the DataWeave code.

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.

The screenshot shows the Mule Studio interface with a 'Transform Message' component selected. The 'Input' tab displays the JSON payload: {"destination": "SFO", "airline": "united"}. The 'Output' tab shows the resulting DataWeave expression and its output: { "destination": "SFO", "airline": "united" }. The 'Preview' tab at the bottom right is visible.

```
%dw 1.0
%output application/json
---
payload
```

```
{
  "destination": "SFO",
  "airline": "united"
}
```

9. Change the DataWeave expression to data: payload.

The screenshot shows the Mule Studio interface with a 'Transform Message' component selected. The 'Input' tab displays the JSON payload: {"destination": "SFO", "airline": "united"}. The 'Output' tab shows the modified DataWeave expression and its output: { "data": { "destination": "SFO", "airline": "united" } }. The 'Preview' tab at the bottom right is visible.

```
%dw 1.0
%output application/json
---
data: payload
```

```
{
  "data": {
    "destination": "SFO",
    "airline": "united"
  }
}
```

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

The screenshot shows the Mule Studio interface with a 'Transform Message' component selected. The 'Input' tab displays the JSON payload: {"destination": "SFO", "airline": "united"}. The 'Output' tab shows the DataWeave expression and its output: { "data": {} }. The 'Preview' tab at the bottom right is visible.

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

```
{
  "data": {}
}
```

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

The screenshot shows the Mule Studio interface with a 'Transform Message' component selected. The 'Input' tab displays the JSON payload: {"destination": "SFO", "airline": "united"}. The 'Output' tab shows the final DataWeave expression and its output: { "data": { "hub": "MUA" } }. The 'Preview' tab at the bottom right is visible.

```
%dw 1.0
%output application/json
---
data: {
  hub: "MUA"
}
```

```
{
  "data": {
    "hub": "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.

The screenshot shows the 'Transform Message' component in the Mule Studio interface. The 'Input' tab displays a JSON object: {"destination": "SFO", "airline": "united"}. The 'Output' tab shows the resulting transformed JSON:

```
%dw 1.0
output application/json
---
data: {
    hub: "MUA",
    code: payload.destination,
    airline: payload.airline
}
```

The 'Payload' tab shows the transformed JSON output:

```
{"data": {"hub": "MUA", "code": "SFO", "airline": "united"}}
```

Below the tabs are buttons for Context, payload, Payload, destination, json, Preview, and Structure.

## 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.

```
%dw 1.0
%output application/xml
---
payload
```

23. Change the DataWeave expression to data: payload.

```
<?xml version='1.0' encoding='UTF-8'?>
<data>
  <destination>SFO</destination>
  <airline>united</airline>
</data>
```

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.

```
<?xml version='1.0' encoding='UTF-8'?>
<data>
  <hub>MUA</hub>
  <flight>
    <code>SFO</code>
    <airline>united</airline>
  </flight>
</data>
```



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

The screenshot shows the Mule Studio interface with the 'Transform Message' component selected. The 'Input' tab displays a JSON payload: {"destination": "SFO", "airline": "united"}. The 'Output' tab shows the resulting XML output:

```
<?xml version='1.0' encoding='UTF-8'?>
<data>
  <hub>MUA</hub>
  <flight airline="united">
    <code>SFO</code>
  </flight>
</data>
```

The 'xml' tab is selected in the bottom navigation bar. Below the tabs, there are buttons for Context, payload, Payload, destination, json, xml, Preview, and Structure.

## 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.

The screenshot shows the Mule Studio interface with the 'Mule Debugger' component selected. The 'Variables' tab is active, displaying two flow variables:

Name	Type
json	java.lang.String
xml	java.lang.String

The 'xml' variable contains the following XML content:

```
<?xml version='1.0' encoding='UTF-8'?>
<data>
  <hub>MUA</hub>
  <flight airline="all">
    <code>SFO</code>
  </flight>
</data>
```

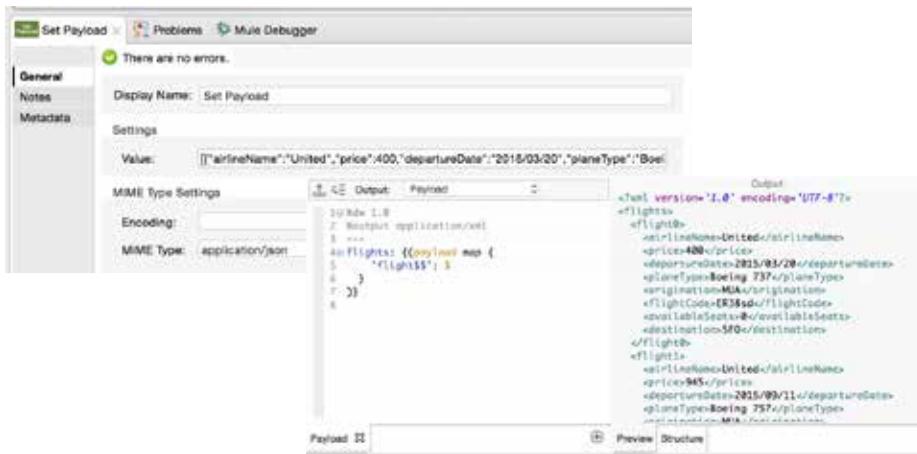
31. Stop the Mule runtime.

*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.



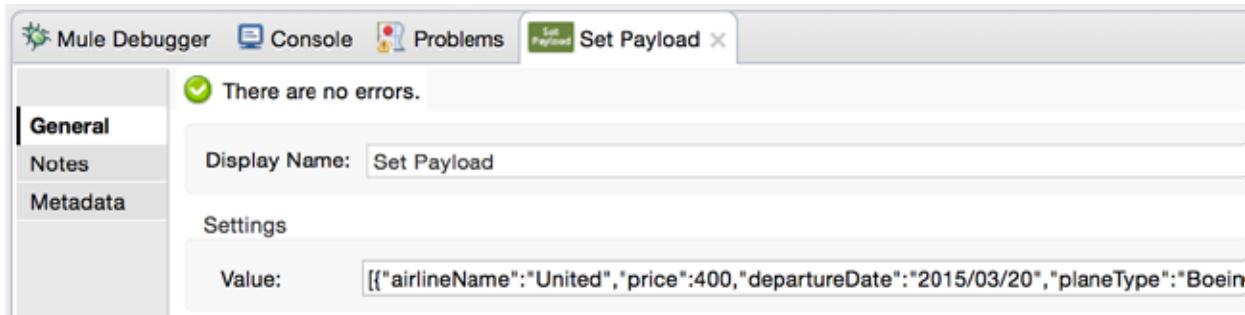
### Create a new flow

1. Return to ap essentials.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

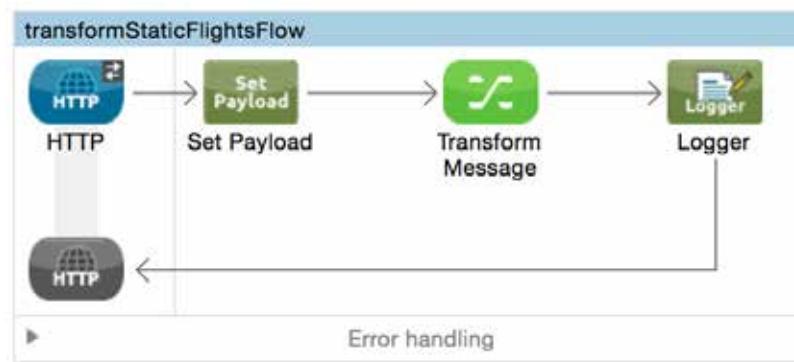
5. Add a Set Payload transformer to the flow.
6. Return to the course snippets.txt file and copy the value for the Example flights JSON.

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.

Name	Type	Value
root	ArrayList	[{"airlineName": "United", "price": 400, "departureDate": "2015/03/20", "planeType": "Boeing 737", "origination": "MUA", "flightCode": "ER38sd", "availableSeats": 0, "destination": "SFO"}, {"airlineName": "United", "price": 945, "departureDate": "2015/09/11", "planeType": "Boeing 757", "origination": "MUA", "flightCode": "ER39rk", "availableSeats": 54, "destination": "SFO"}, {"airlineName": "United", "price": 954, "departureDate": "2015/02/12", "planeType": "Boeing 777", "origination": "MUA", "flightCode": "ER39rj", "availableSeats": 23, "destination": "SFO"}]
[0]	LinkedHashMap	airlineName : S United price : Integer 400 departureDate : 2015/03/20 planeType : Str Boeing 737 origination : Str MUA flightCode : Str ER38sd availableSeats : 0 destination : Str SFO
[1]	LinkedHashMap	airlineName : S United price : Integer 945 departureDate : 2015/09/11 planeType : Str Boeing 757 origination : Str MUA flightCode : Str ER39rk availableSeats : 54 destination : Str SFO
[2]	LinkedHashMap	airlineName : S United price : Integer 954 departureDate : 2015/02/12 planeType : Str Boeing 777 origination : Str MUA flightCode : Str ER39rj availableSeats : 23 destination : Str SFO

## 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.

Name	Value	Type
Message		org.mule.DefaultMuleMessage
Message Processor	Transform Message	com.mulesoft.weave.mule.Weave...
payload	[{"airlineName": "United", "price": 400, "departureDate": "2015/03/20", "planeType": "Boeing 737", "origination": "MUA", "flightCode": "ER38sd", "availableSeats": 0, "destination": "SFO"}, {"airlineName": "United", "price": 945, "departureDate": "2015/09/11", "planeType": "Boeing 757", "origination": "MUA", "flightCode": "ER39rk", "availableSeats": 54, "destination": "SFO"}, {"airlineName": "United", "price": 954, "departureDate": "2015/02/12", "planeType": "Boeing 777", "origination": "MUA", "flightCode": "ER39rj", "availableSeats": 23, "destination": "SFO"}]	java.lang.String

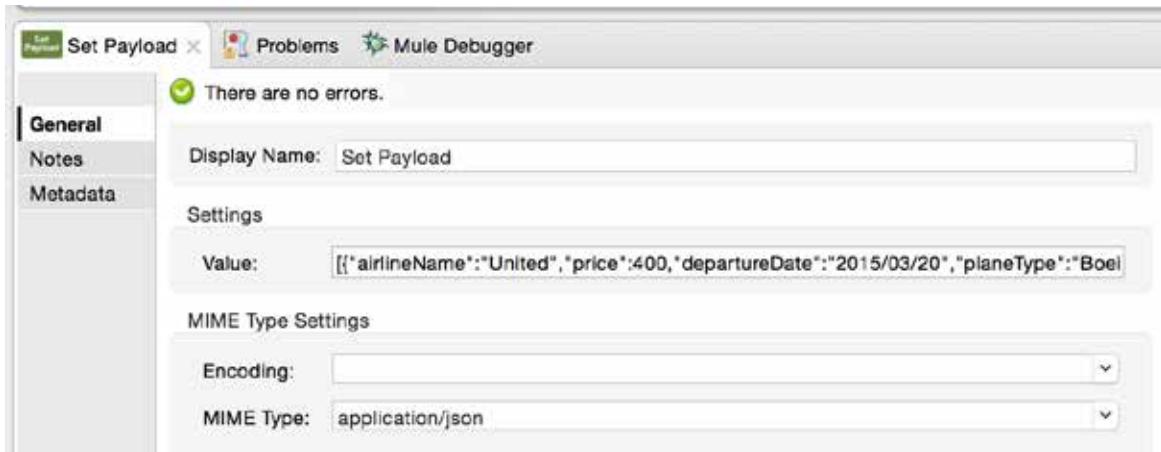
20. Step to the Logger; you should see the payload is still of type String.

Name	Value	Type
Message		org.mule.DefaultMuleMessage
Message Processor	Logger	org.mule.api.processor.LoggerMe...
payload	[{"airlineName": "United", "price": 400, "departureDate": "2015/03/20", "planeType": "Boeing 737", "origination": "MUA", "flightCode": "ER38sd", "availableSeats": 0, "destination": "SFO"}, {"airlineName": "United", "price": 945, "departureDate": "2015/09/11", "planeType": "Boeing 757", "origination": "MUA", "flightCode": "ER39rk", "availableSeats": 54, "destination": "SFO"}, {"airlineName": "United", "price": 954, "departureDate": "2015/02/12", "planeType": "Boeing 777", "origination": "MUA", "flightCode": "ER39rj", "availableSeats": 23, "destination": "SFO"}]	java.lang.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.

Name	Value	Type
Message		org.mule.DefaultMuleMessage
Message Processor	Logger	org.mule.api.processor.LoggerMe...
payload	[{"airlineName": "United", "price": 400, ...}, {"airlineName": "American", "price": 945, ...}, {"airlineName": "Delta", "price": 954, ...}, {"airlineName": "Southwest", "price": 220, ...}, {"airlineName": "Alaska", "price": 400, ...}, {"airlineName": "Frontier", "price": 220, ...}, {"airlineName": "Spirit", "price": 220, ...}, {"airlineName": "Delta", "price": 220, ...}]	java.util.ArrayList
0	{airlineName=United, price=400, ...}	java.util.LinkedHashMap
1	{airlineName=United, price=945, ...}	java.util.LinkedHashMap
2	{airlineName=United, price=954, ...}	java.util.LinkedHashMap
0	airlineName=United	java.util.LinkedHashMap\$Entry
1	price=954	java.util.LinkedHashMap\$Entry
2	departureDate=2015/02/12	java.util.LinkedHashMap\$Entry
3	planeType=Boeing 777	java.util.LinkedHashMap\$Entry
4	origin=MUA	java.util.LinkedHashMap\$Entry
5	code=ER39rj	java.util.LinkedHashMap\$Entry
6	emptySeats=23	java.util.LinkedHashMap\$Entry
7	destination=SFO	java.util.LinkedHashMap\$Entry

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.

The screenshot shows the Mule Studio interface with a 'Transform Message' component selected. The DataWeave editor on the left contains the following code:

```
%dw 1.0
%output application/java
---
[{"airlineName": "United", "price": 400, "departureDate": "2015/03/20", "planeType": "Boeing 737", "origination": "MUA", "flightCode": "ER38sd", "availableSeats": 0, "destination": "SFO"},
```

The Preview tab on the right shows the output structure:

Name	Value
root : ArrayList	
[0] : Integer	400

## 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.

The screenshot shows the Mule Studio interface with a 'Transform Message' component selected. The DataWeave editor on the left contains the following code:

```
%dw 1.0
%output application/java
---
4 payload.price
```

The Preview tab on the right shows the output structure:

Name	Value
root : ArrayList	
[0] : Integer	400
[1] : Integer	945
[2] : Integer	954

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.

The screenshot shows the Mule Studio interface with a 'Transform Message' component selected. The DataWeave editor on the left contains the following code:

```
%dw 1.0
%output application/java
---
4 [payload.*price, payload.availableSeats]
```

The Preview tab on the right shows the output structure:

Name	Value
root : ArrayList	
[0] : ArrayList	<ul style="list-style-type: none"> <li>[0] : Integer 400</li> <li>[1] : Integer 945</li> <li>[2] : Integer 954</li> </ul>
[1] : ArrayList	<ul style="list-style-type: none"> <li>[0] : Integer 0</li> <li>[1] : Integer 54</li> <li>[2] : Integer 23</li> </ul>

## 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.

The screenshot shows the DataWeave editor and its preview pane. The DataWeave code is:

```
1@%dw 1.0
2 %output application/java
3 ---
4 payload map $
```

The preview pane shows the resulting output as a table:

Name	Value
root : ArrayList	
[0] : LinkedHashMap	
[1] : LinkedHashMap	
[2] : LinkedHashMap	
airlineName : String	United
price : Integer	954
departureDate : String	2015/02/12
planeType : String	Boeing 777
origination : String	MUA
flightCode : String	ER39rj
availableSeats : Integer	23
destination : String	SFO

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

The screenshot shows the DataWeave editor and its preview pane. The DataWeave code is:

```
1@%dw 1.0
2 %output application/java
3 ---
4 payload map {
5   flight: $$
6 }
```

The preview pane shows the resulting output as a table:

Name	Value
root : ArrayList	
[0] : LinkedHashMap	0
[1] : LinkedHashMap	1
[2] : LinkedHashMap	2

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.

The screenshot shows the DataWeave editor and its preview pane. The DataWeave code is:

```
1@%dw 1.0
2 %output application/java
3 ---
4 payload map {
5   flight: $$,
6   dest: $.destination
7 }
```

The preview pane shows the resulting output as a table:

Name	Value
root : ArrayList	
[0] : LinkedHashMap	0
dest : String	SFO
[1] : LinkedHashMap	1
dest : String	SFO
[2] : LinkedHashMap	2
dest : String	SFO

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

The screenshot shows the Mule Studio DataWeave editor. The payload is defined as:

```
1@%dw 1.0
2 %output application/java
3 ---
4@payload map {
5   flight: $$,
6   ($$): $ 
7 }
```

The output pane shows the resulting structure:

Name	Value
root : ArrayList	
[0] : LinkedHashMap	
[1] : LinkedHashMap	
[2] : LinkedHashMap	2
2 : LinkedHashMap	
airlineName : String	United
price : Integer	954
departureDate : String	2015/02/12
planeType : String	Boeing 777
origination : String	MUA
flightCode : String	ER39rj
availableSeats : Integer	23
destination : String	SFO

Below the editor are tabs for Payload, Preview, and Structure.

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

The screenshot shows the Mule Studio DataWeave editor. The payload is defined as:

```
1@%dw 1.0
2 %output application/java
3 ---
4@payload map {
5   'flight$$': $ 
6 }
```

The output pane shows the resulting structure:

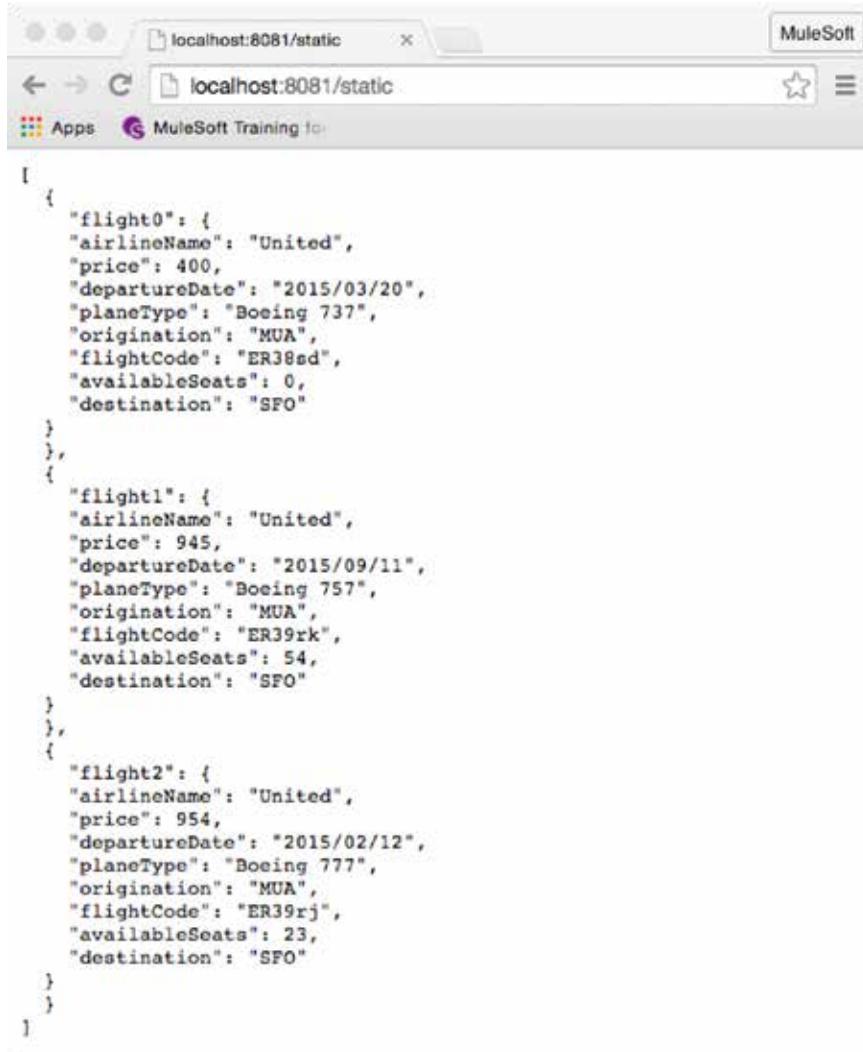
Name	Value
root : ArrayList	
[0] : LinkedHashMap	
[1] : LinkedHashMap	
[2] : LinkedHashMap	
flight2 : LinkedHashMap	
airlineName : String	United
price : Integer	954
departureDate : String	2015/02/12
planeType : String	Boeing 777
origination : String	MUA
flightCode : String	ER39rj
availableSeats : Integer	23
destination : String	SFO

Below the editor are tabs for Payload, Preview, and Structure.

## 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.



The screenshot shows a web browser window with the address bar containing "localhost:8081/static". The page content displays a JSON array of flight information:

```
[{"flight0": {"airlineName": "United", "price": 400, "departureDate": "2015/03/20", "planeType": "Boeing 737", "origination": "MUA", "flightCode": "ER38sd", "availableSeats": 0, "destination": "SFO"}, {"flight1": {"airlineName": "United", "price": 945, "departureDate": "2015/09/11", "planeType": "Boeing 757", "origination": "MUA", "flightCode": "ER39rk", "availableSeats": 54, "destination": "SFO"}, {"flight2": {"airlineName": "United", "price": 954, "departureDate": "2015/02/12", "planeType": "Boeing 777", "origination": "MUA", "flightCode": "ER39rj", "availableSeats": 23, "destination": "SFO"}]
```

## 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$$':$  
})}
```

```

1@%dw 1.0
2 %output application/xml
3 ---
4 flights: {${payload map {
5   'flight$': $}
6 }
7 }
8

```

Output

```

<?xml version='1.0' encoding='UTF-8'?>
<flights>
  <flight0>
    <airlineName>United</airlineName>
    <price>400</price>
    <departureDate>2015/03/20</departureDate>
    <planeType>Boeing 737</planeType>
    <origination>MUA</origination>
    <flightCode>ER38sd</flightCode>
    <availableSeats>0</availableSeats>
    <destination>SFO</destination>
  </flight0>
  <flight1>
    <airlineName>United</airlineName>
    <price>945</price>
    <departureDate>2015/09/11</departureDate>
    <planeType>Boeing 757</planeType>
    <origination>MUA</origination>
  </flight1>

```

Payload X + Preview Structure

45. Save the file and run the application.

46. Make a request to <http://localhost:8081/static>; you should see the XML returned.

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```

<flights>
  <flight0>
    <airlineName>United</airlineName>
    <price>400</price>
    <departureDate>2015/03/20</departureDate>
    <planeType>Boeing 737</planeType>
    <origination>MUA</origination>
    <flightCode>ER38sd</flightCode>
    <availableSeats>0</availableSeats>
    <destination>SFO</destination>
  </flight0>
  <flight1>
    <airlineName>United</airlineName>
    <price>945</price>
    <departureDate>2015/09/11</departureDate>
    <planeType>Boeing 757</planeType>
    <origination>MUA</origination>
    <flightCode>ER39rk</flightCode>
    <availableSeats>54</availableSeats>
    <destination>SFO</destination>
  </flight1>

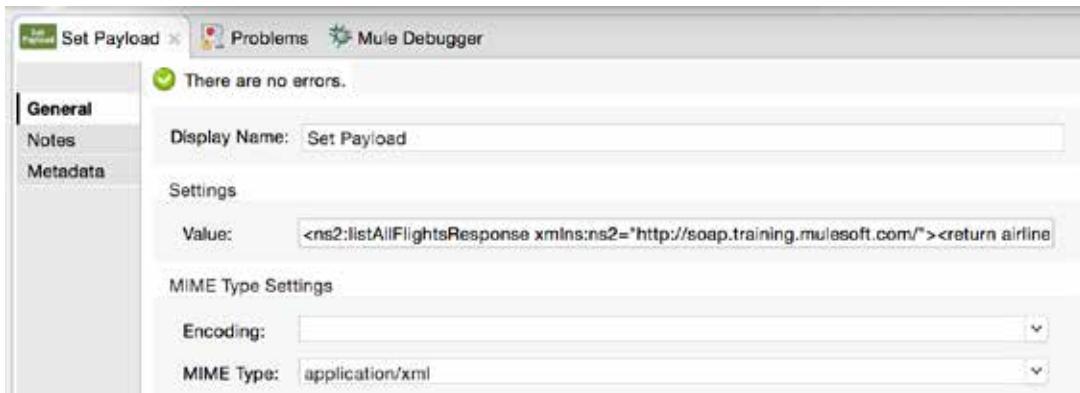
```

## 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.

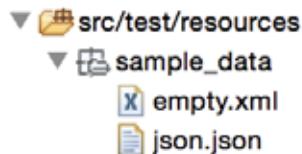
49. Change the MIME type to application/xml.



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.

The screenshot shows the 'Transform Message' properties view in Mule Studio. The 'Input' tab shows XML code: <?xml version='1.0' encoding='UTF-8'?><ns2:listAllFlightsResponse xmlns:ns2="http://soap.training.mulesoft.com/"><return airlineName="United"><code>A1B2C3</code><departureDate>2015-10-20</departureDate><destination>SFO</destination><emptySeats>40</emptySeats><origin>MUA</origin><planeType>Boing 737</planeType><price>400.0</price></return><return airlineName="Delta"><code>A1B2C4</code><departureDate>2015-10-21</departureDate><destination>LAX</destination></return></listAllFlightsResponse>. The 'Output' tab shows a Groovy script: 1@ %dw 1.0 2 %output application/xml 3 --- 4@ flights: {\${payload map { 5 'flight\$': \$ 6 }} 7 }}. The 'Preview' tab shows the resulting XML output: <?xml version='1.0' encoding='UTF-8'?><flights> <flight0> <return airlineName="United"> <code>A1B2C3</code> <departureDate>2015-10-20</departureDate> <destination>SFO</destination> <emptySeats>40</emptySeats> <origin>MUA</origin> <planeType>Boing 737</planeType> <price>400.0</price> </return> <return airlineName="Delta"> <code>A1B2C4</code> <departureDate>2015-10-21</departureDate> <destination>LAX</destination> </return> </flights>.

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

```

Transform Message
Input
<?xml version='1.0' encoding='UTF-8'?>
<ns2:listAllFlightsResponse
xmlns:ns2="http://
soap.training.mulesoft.com/"><return
airlineName="United"><code>A1B2C3</
code><departureDate>2015-10-20</
departureDate><destination>SFO</
destination><emptySeats>40</
emptySeats><origin>MUA</
origin><planeType>Boing 737</
planeType><price>400.0</price></
return><return
airlineName="Delta"><code>A1B2C4</
code><departureDate>2015-10-21</
departureDate>
Output
<?xml version='1.0'
encoding='UTF-8'?>
<flights>
<flight0>
<code>A1B2C3</code>
<departureDate>2015-10-20</
departureDate>
<destination>SFO</destination>
<emptySeats>40</emptySeats>
<origin>MUA</origin>
<planeType>Boing 737</
planeType>
<price>400.0</price>
</flight0>
<flight1>
<code>A1B2C4</code>
<departureDate>2015-10-21</
departureDate>

```

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

```

Transform Message
Input
<?xml version='1.0'
encoding='UTF-8'?>
<ns2:listAllFlightsResponse
xmlns:ns2="http://
soap.training.mulesoft.com/"><r
eturn
airlineName="United"><code>A1B2
C3</
code><departureDate>2015-10-20</
departureDate><destination>SFO</
destination><emptySeats>40</
emptySeats><origin>MUA</
origin><planeType>Boing 737</
planeType>
Output
<?xml version='1.0'
encoding='UTF-8'?>
<flights>
<flight0>A1B2C3</flight0>
<flight1>2015-10-20</flight1>
<flight2>SFO</flight2>
<flight3>40</flight3>
<flight4>MUA</flight4>
<flight5>Boing 737</flight5>
<flight6>400.0</flight6>
</flights>

```

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

```

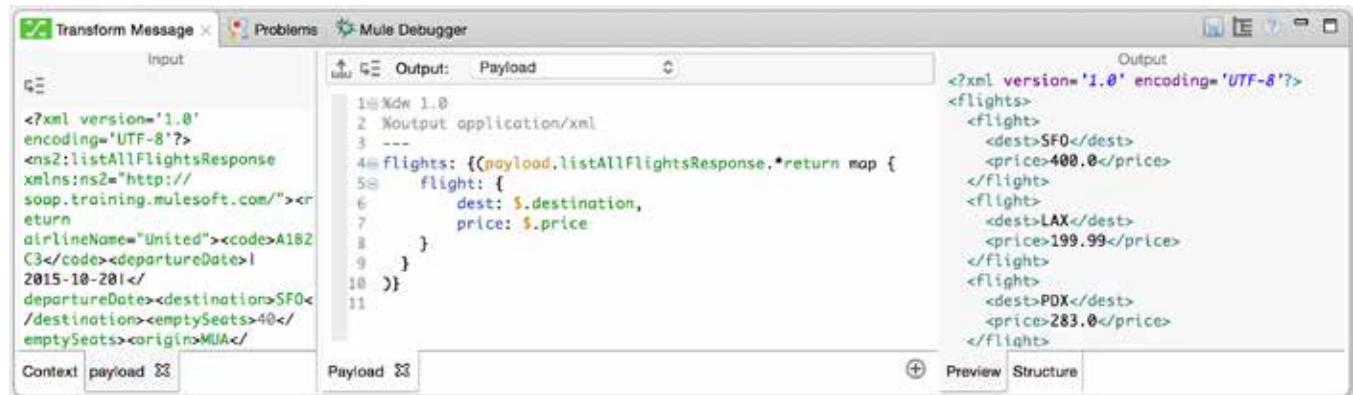
Transform Message
Input
<?xml version='1.0'
encoding='UTF-8'?>
<ns2:listAllFlightsResponse
xmlns:ns2="http://
soap.training.mulesoft.com/"><r
eturn
airlineName="United"><code>A1
B2C3</
code><departureDate>2015-10-2
0</
departureDate><destination>S
F0</
destination><emptySeats>40</
emptySeats><origin>MUA</
origin>
Output
<?xml version='1.0' encoding='UTF-8'?>
>
<flights>
<flight0>
<code>A1B2C3</code>
<departureDate>2015-10-20</
departureDate>
<destination>SFO</destination>
<emptySeats>40</emptySeats>
<origin>MUA</origin>
<planeType>Boing 737</planeType>
<price>400.0</price>
</flight0>
<flight1>
<code>A1B2C4</code>
<departureDate>2015-10-21</
departureDate>

```

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.



```
Input
<?xml version='1.0' encoding='UTF-8'?>
<ns2:listAllFlightsResponse xmlns:ns2="http://soap.training.mulesoft.com/"><return
airlineName="United"><code>A1B2C3</code><departureDate>2015-10-28</departureDate><destination>SFO</destination><emptySeats>40</emptySeats><origin>MIA</origin>
```

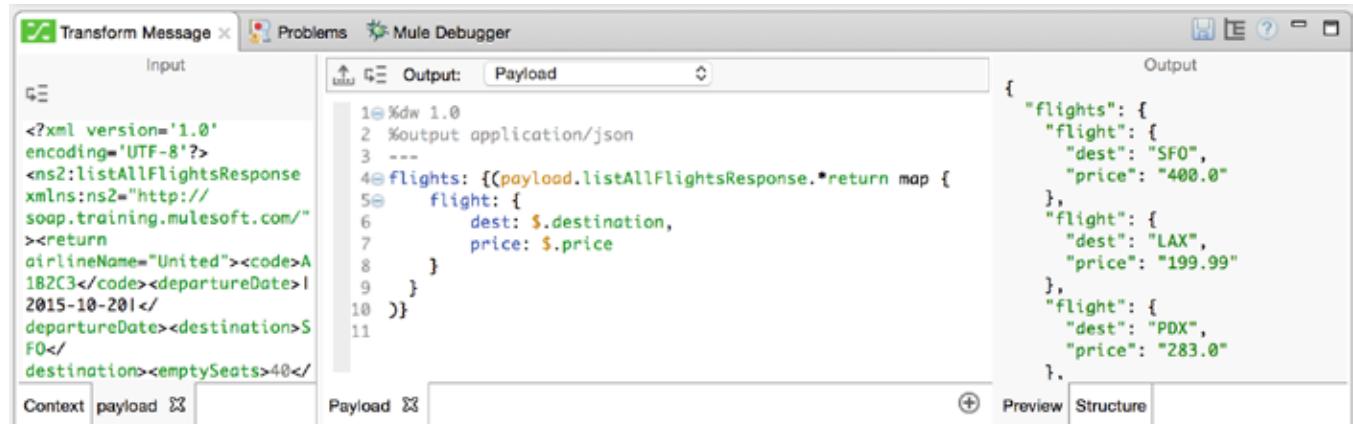
```
Output
<?xml version='1.0' encoding='UTF-8'?>
<flights>
<flight>
<dest>SFO</dest>
<price>400.0</price>
</flight>
<flight>
<dest>LAX</dest>
<price>199.99</price>
</flight>
<flight>
<dest>PDX</dest>
<price>283.0</price>
</flight>
```

```
Payload
```

```
Preview Structure
```

## Change the output structure to JSON

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



```
Input
<?xml version='1.0' encoding='UTF-8'?>
<ns2:listAllFlightsResponse xmlns:ns2="http://soap.training.mulesoft.com/"><return
airlineName="United"><code>A1B2C3</code><departureDate>2015-10-28</departureDate><destination>SFO</destination><emptySeats>40</emptySeats>
```

```
Output
{
  "flights": [
    "flight": {
      "dest": "SFO",
      "price": "400.0"
    },
    "flight": {
      "dest": "LAX",
      "price": "199.99"
    },
    "flight": {
      "dest": "PDX",
      "price": "283.0"
    }
  ]
}
```

```
Payload
```

```
Preview Structure
```

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.

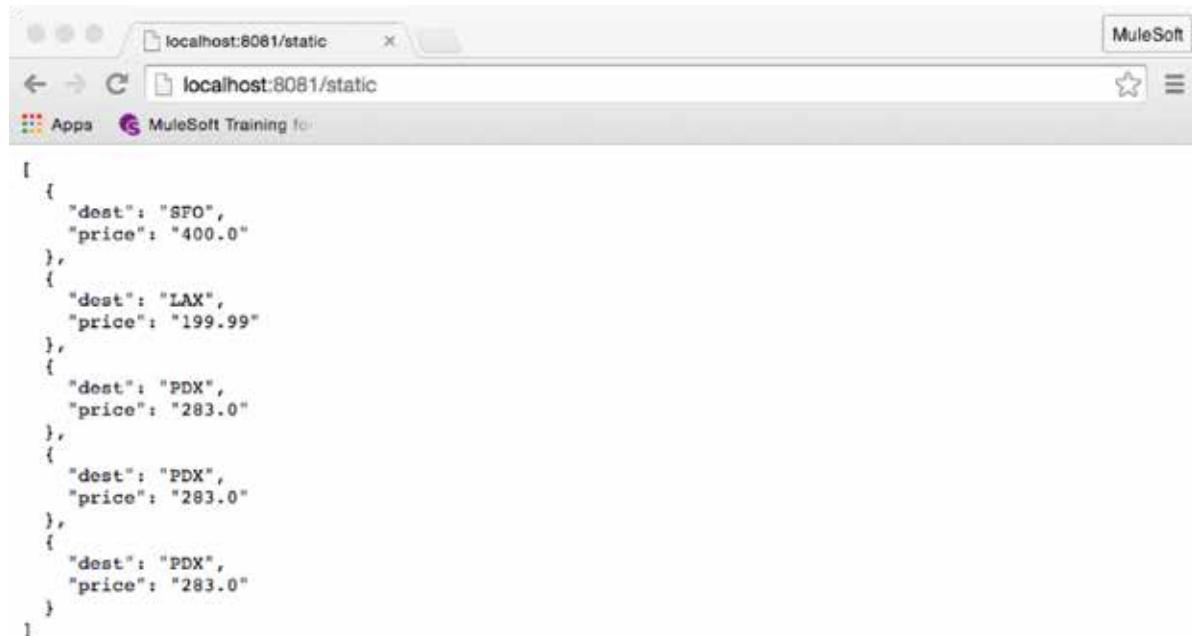
```
flights: payload.listAllFlightsResponse.*return map {
    flight: {
        dest: $.destination,
        price: $.price
    }
}
```

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.



```
[{"dest": "SFO", "price": "400.0"}, {"dest": "LAX", "price": "199.99"}, {"dest": "PDX", "price": "283.0"}, {"dest": "PDX", "price": "283.0"}, {"dest": "PDX", "price": "283.0"}]
```



## 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.

```
dw 1.0
output application/json
---
payload.listAllFlightsResponse.*return map {
    dest: $.destination,
    price: $.price,
    plane: upper $.planeType
}
```

Name	Type	Value
root	ArrayList	
[0]	Flight	<ul style="list-style-type: none"><li>airlineName : String</li><li>availableSeats : Integer 10</li><li>departureDate : String Oct 21, 2015</li><li>destination : String LAX</li><li>flightCode : String</li><li>origination : String</li><li>planeType : String BOING 737</li><li>price : Double 199.99</li></ul>
[1]	Flight	<ul style="list-style-type: none"><li>airlineName : String</li><li>availableSeats : Integer 23</li><li>departureDate : String Oct 20, 2015</li></ul>

### Format a string

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

plane: upper \$.planeType

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

```
dw 1.0
output application/json
---
payload.listAllFlightsResponse.*return map {
    dest: $.destination,
    price: $.price,
    plane: upper $.planeType
}
```

Index	Flight	Destination	Price	Plane Type
0	United	SFO	"400.0"	BOING 737
1	Delta	LAX	"199.99"	BOING 737

## 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,
```

The screenshot shows the Mule Studio interface with the 'Transform Message' component open. The 'Input' tab displays an XML message structure. The 'Output' tab shows the DataWeave code and the resulting JSON output. The DataWeave code is:

```
1@%dw 1.0
2 %output application/json
3 ---
4@payload.listAllFlightsResponse.*return map {
5   dest: $.destination,
6   price: $.price as :number,
7   plane: upper $.planeType
8 }
9
```

The 'Output' tab shows the generated JSON:

```
[{"dest": "SFO", "price": 400.0, "plane": "BOING 737"}, {"dest": "LAX", "price": 199.99, "plane": "BOING 737"}]
```

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.

The screenshot shows the Mule Studio interface with the 'Transform Message' component open. The 'Input' tab displays an XML message structure. The 'Output' tab shows the DataWeave code and the resulting Java output. The DataWeave code is identical to the previous step:

```
1@%dw 1.0
2 %output application/java
3 ---
4@payload.listAllFlightsResponse.*return map {
5   dest: $.destination,
6   price: $.price as :number,
7   plane: upper $.planeType,
8   departureDate: $.departureDate
9 }
10
```

The 'Output' tab shows the generated Java code in the 'Preview' tab:

```
root : ArrayList
  [0] : LinkedHashMap
    dest : String          SFO
    price : Integer        400
    plane : String         BOING 737
    departureDate : String 2015/10/20
  [1] : LinkedHashMap
    dest : String          LAX
    price : Double         199.99
```

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.

Transform Message

Input

```
<?xml version='1.0' encoding='UTF-8'?>
<ns2:listAllFlightsResponse xmlns:ns2="http://
soap.training.mulesoft.com/"><return
airlineName="United"><code>A1B2C3</
code><departureDate>2015/10/20</
departureDate><destination>SFO</
destination><emptySeats>40</
emptySeats><origin>MUA</
origin><planeType>Boing 737</planeType><price>400.0</price></
return><return
airlineName="Delta"><code>A1B2C4</
code><departureDate>2015/10/21</departureDate><destination>LAX</
destination><emptySeats>10</emptySeats><origin>MUA</
origin><planeType>Boing 737</planeType><price>199.99</price></
return>
```

Output

Name	Value
root : ArrayList	
[0] : LinkedHashMap	
dest : String	SFO
price : Integer	400
plane : String	BOING 737
departureDate : String	2015/10/20
[1] : LinkedHashMap	
dest : String	LAX

Preview

Can not update preview. Validate your mappings.

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

Transform Message

Input

```
<?xml version='1.0' encoding='UTF-8'?>
<ns2:listAllFlightsResponse xmlns:ns2="http://
soap.training.mulesoft.com/"><return
airlineName="United"><code>A1B2C3</
code><departureDate>2015/10/20</departureDate><destination>SFO</
destination><emptySeats>40</emptySeats><origin>MUA</
origin><planeType>Boing 737</planeType><price>400.0</price></
return><return
airlineName="Delta"><code>A1B2C4</
code><departureDate>2015/10/21</departureDate><destination>LAX</
destination><emptySeats>10</emptySeats><origin>MUA</
origin><planeType>Boing 737</planeType><price>199.99</price></
return>
```

Payload

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.

Transform Message

Input

```
<?xml
version='1.0'
encoding='UTF-8'
?>
<ns2:listAllFlightsResponse xmlns:ns2="http://
soap.training.mulesoft.com/"><return
airlineName="United"><code>A1B2C3</
code><departureDate>2015/10/20</departureDate><destination>SFO</
destination><emptySeats>40</emptySeats><origin>MUA</
origin><planeType>Boing 737</planeType><price>400.0</price></
return><return
airlineName="Delta"><code>A1B2C4</
code><departureDate>2015/10/21</departureDate><destination>LAX</
destination><emptySeats>10</emptySeats><origin>MUA</
origin><planeType>Boing 737</planeType><price>199.99</price></
return>
```

Output

Name	Value
root : ArrayList	
[0] : LinkedHashMap	
dest : String	SFO
price : Integer	400
plane : String	BOING 737
departureDate : Date	Tue Oct 20 12:00:00 PDT
[1] : LinkedHashMap	
dest : String	LAX
price : Double	199.99

Preview

## 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.

The screenshot shows the Mule Studio interface with a 'Transform Message' component selected. The 'Input' tab displays an XML message structure. The 'Output' tab shows the transformed payload. The 'Preview' tab on the right displays the resulting data structure, which includes a list of flights with their destination, price, plane type, and now-formatted departure date ('2015/10/20').

Name	Value
root	ArrayList
[0]	LinkedHashMap
dest	SFO
price	400.0
plane	BOING 737
departureDate	2015/10/20
[1]	LinkedHashMap
dest	LAX
price	199.99

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.

The screenshot shows the Mule Studio interface with a 'Transform Message' component selected. The 'Input' tab displays an XML message structure. The 'Output' tab shows the transformed payload. The 'Preview' tab on the right displays the resulting data structure, which includes a list of flights with their destination, price, plane type, and now-formatted departure date ('Oct 20, 2015').

Name	Value
root	ArrayList
[0]	LinkedHashMap
dest	SFO
price	400.0
plane	BOING 737
departureDate	Oct 20, 2015
[1]	LinkedHashMap
dest	LAX
price	199.99

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.

```
[{"dest": "SFO", "price": "400.0", "plane": "BOING 737", "departureDate": "Oct 20, 2015"}, {"dest": "LAX", "price": "199.99", "plane": "BOING 737", "departureDate": "Oct 21, 2015"}, {"dest": "PDX", "price": "283.0", "plane": "Airbus A320", "departureDate": "Oct 22, 2015"}]
```

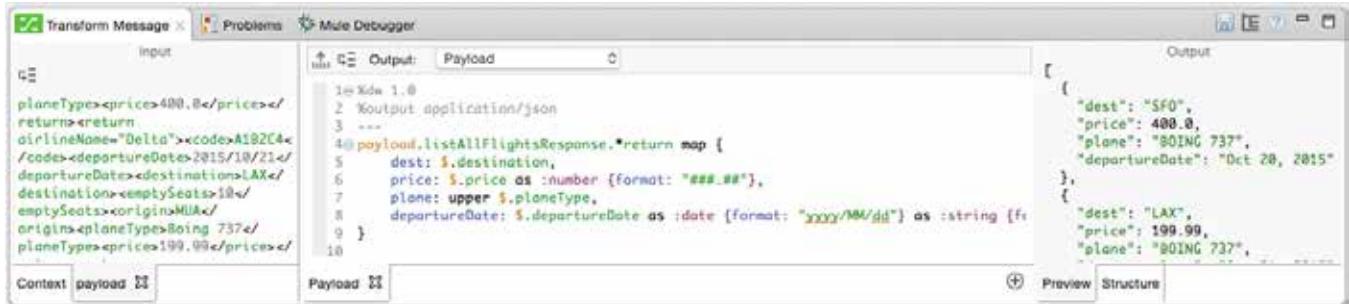


## Format a number

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.



```
1@%dw 1.0
2 %output application/json
3 ===
4@payload.listAllFlightsResponse.*return map {
5   dest: $.destination,
6   price: $.price as :number {format: "###.##"}, // This line formats the price
7   plane: upper $.planeType,
8   departureDate: $.departureDate as :date {format: "yyyy/MM/dd"} as :string [f
9 }
```

The Output tab shows the resulting JSON:

```
[{"dest": "SFO", "price": 400.0, "plane": "BOING 737", "departureDate": "Oct 28, 2015"}, {"dest": "LAX", "price": 199.99, "plane": "BOING 737", "departureDate": "Oct 28, 2015"}]
```

*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.

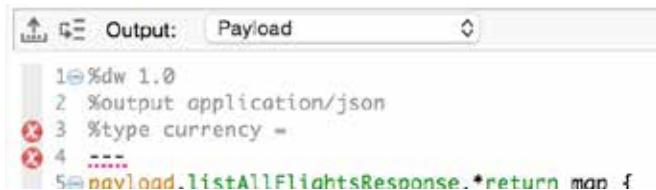
```
price: $.price as :number as :string {format: "#,###.00"},
```

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.



```
1@%dw 1.0
2 %output application/json
3 %type currency =
4 ===
5@payload.listAllFlightsResponse.*return map {
```

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.

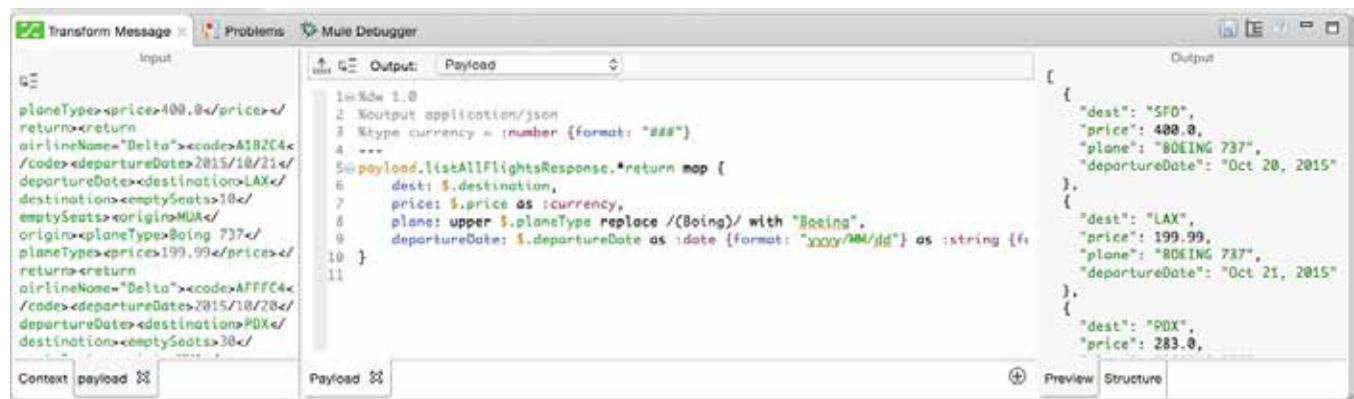
*Note: There is a bug in Mule runtime 3.7.1 and the numbers are not formatted correctly.*

## 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.



```
Input
1<?xml version="1.0"
2<?xml-stylesheet type="text/xsl" href="FlightList.xsl"
3<flights>
4<flight>
5<id>1</id>
6<origin>SFO</origin>
7<destination>LAX</destination>
8<date>2015/10/21</date>
9<time>10:00</time>
10<price>400.0</price>
11</flight>
12<flight>
13<id>2</id>
14<origin>LAX</origin>
15<destination>SFO</destination>
16<date>2015/10/21</date>
17<time>12:00</time>
18<price>400.0</price>
19</flight>
20<flight>
21<id>3</id>
22<origin>SFO</origin>
23<destination>PDX</destination>
24<date>2015/10/21</date>
25<time>14:00</time>
26<price>283.0</price>
27</flight>
28</flights>
29</?xml>

Output
[{"dest": "SFO", "price": 400.0, "plane": "BOEING 737", "departureDate": "Oct 21, 2015"}, {"dest": "LAX", "price": 400.0, "plane": "BOEING 737", "departureDate": "Oct 21, 2015"}, {"dest": "PDX", "price": 283.0, "plane": "Airbus A320"}]
```

## 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.

```
payload.listAllFlightsResponse.*return map {
    dest: $destination,
    price: $.price as :currency,
    plane: upper $.planeType replace /(Boing)/ with "Boeing",
    departureDate: $.departureDate as :date {format: "yyyy/MM/dd"} as :string
} orderBy $.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.

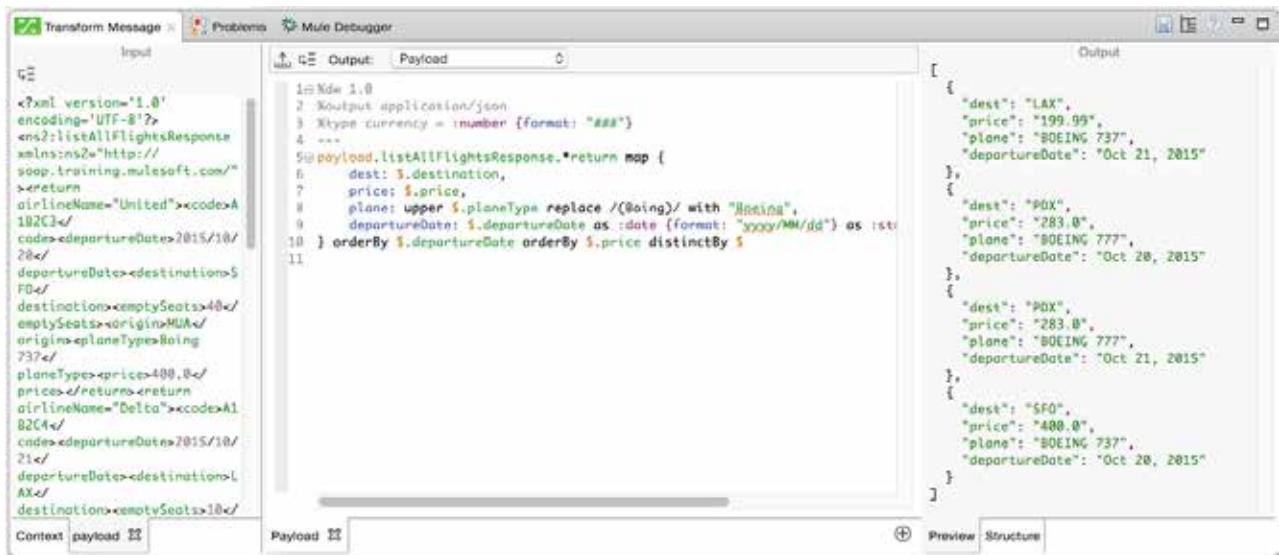
```
payload.listAllFlightsResponse.*return map {
    dest: $destination,
    price: $.price as :currency,
    plane: upper $.planeType replace /(Boing)/ with "Boeing",
    departureDate: $.departureDate as :date {format: "yyyy/MM/dd"} as :string
} orderBy $.price orderBy $.departureDate
```

## 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.



```
1<!-- Node 1.0
2<!-- Output application/json
3<-- type currency = :number {format: "###,##0"}
4-->
5@payload.listAllFlightsResponse.*return map {
6    dest: $.destination,
7    price: $.price,
8    plane: upper $.planeType replace /(@oing)/ with "Coming",
9    departureDate: $.departureDate as :date {format: "yyyy/MM/dd"} as ist
10 } orderBy $.departureDate orderBy $.price distinctBy $
```

The screenshot shows the Mule Studio interface with the 'Transform Message' component selected. The 'Input' tab displays an XML snippet representing flight requests. The 'Output' tab contains a MEL expression that maps the input to a JSON structure. The 'Preview' tab shows the resulting JSON output, which includes only four flight records, as instructed.

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: "###,##0"} instead of price: \$.price as :currency. This is a bug in Mule runtime 3.7.1.

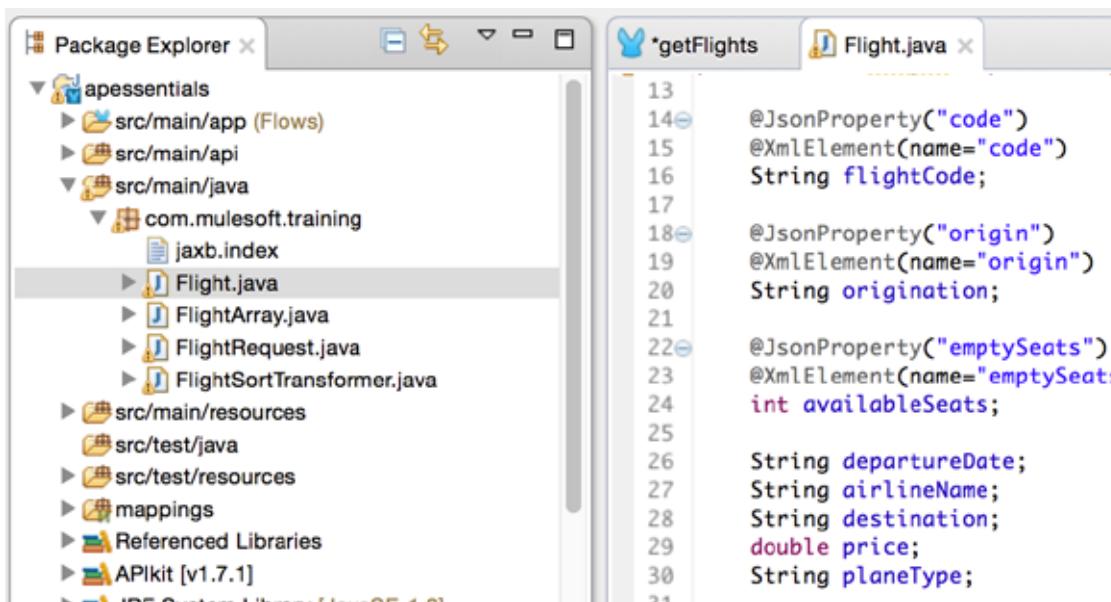
44. Add an availableSeats field that is equal to the emptySeats field and coerce it to a number.

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.



```
13
14 @JsonProperty("code")
15 @XmlElement(name="code")
16 String flightCode;
17
18 @JsonProperty("origin")
19 @XmlElement(name="origin")
20 String origination;
21
22 @JsonProperty("emptySeats")
23 @XmlElement(name="emptySeats")
24 int availableSeats;
25
26 String departureDate;
27 String airlineName;
28 String destination;
29 double price;
30 String planeType;
```

The screenshot shows the Eclipse IDE interface with the 'Package Explorer' and 'Flight.java' editor tabs. The 'Package Explorer' shows the project structure with 'src/main/java/com.mulesoft.training' expanded, revealing 'Flight.java'. The 'Flight.java' editor shows the Java code for the Flight class, which has properties for flightCode, origination, availableSeats, departureDate, airlineName, destination, price, and planeType.

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.

The screenshot shows the Mule Studio interface with the 'Transform Message' editor open. The 'Input' tab displays an XML payload with flight information. The 'Output' tab shows the DataWeave code being executed. The 'Preview' tab on the right displays the resulting data as three LinkedHashMap objects under the 'root' key.

Name	Value
root	ArrayList
[0]	LinkedHashMap
dest	LAX
price	199.99
plane	BOEING 737
departureDate	Oct 21, 2015
availableSeats	10
[1]	LinkedHashMap
dest	PDX
price	285.0
plane	BOEING 777
departureDate	Oct 20, 2015
availableSeats	0
[2]	LinkedHashMap

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.

The screenshot shows the Mule Studio interface with the 'Transform Message' component selected. The 'Input' tab displays an XML payload with flight data. The 'Output' tab shows the transformed payload in JSON format, which is then mapped to a 'Flight' object. The 'Payload' tab shows the resulting list of flights. The 'Preview' tab on the right shows the detailed structure of the 'Flight' objects, including fields like airlineName, availableSeats, departureDate, destination, flightCode, origination, planeType, and price. The 'Structure' tab is also visible.

```
1<!> Ndw 1.0
2<!> Output application/json
3<!> Xtype currency = :number {format: "###"}
4<!> Xtype flight = :object {class: "com.mulesoft.training.Flight"}
5<!>
6<!> payload.listAllFlightsResponse.*return map {
7<!>     destination: $.destination,
8<!>     price: $.price,
9<!>     planeType: upper $.planeType replace /(Boing)/ with "Boeing",
10<!>    departureDate: $.departureDate as :date {format: "yyyy/MM/dd"} as :string {
11<!>        availableSeats: $.emptySeats as :number
12<!>    } as :flight orderBy $.departureDate orderBy $.price distinctBy $
13<!>
```

Name	Type
root	ArrayList
[0]	Flight
airlineName	String
availableSeats	Integer
departureDate	Date
destination	String
flightCode	String
origination	String
planeType	String
price	Double
[1]	Flight
airlineName	String
availableSeats	Integer
departureDate	Date

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.

The screenshot shows the Mule Properties view with the 'payload' object expanded. It contains five elements, each representing a 'Flight' object. The first four elements have their properties filled with values (e.g., airlineName, availableSeats, departureDate, destination), while the fifth element has all properties set to null. This demonstrates that the transformation did not correctly handle the null values from the input XML.

Name	Value	Type
payload	size = 5	java.util.ArrayList
0	com.mulesoft.training.Flight@92...	com.mulesoft.training.Flight
1	com.mulesoft.training.Flight@7ff...	com.mulesoft.training.Flight
2	com.mulesoft.training.Flight@59...	com.mulesoft.training.Flight
3	com.mulesoft.training.Flight@6a...	com.mulesoft.training.Flight
4	com.mulesoft.training.Flight@7d...	com.mulesoft.training.Flight
airlineName	null	java.lang.String
availableSeats	40	java.lang.Integer
departureDate	Oct 20, 2015	java.lang.String
destination	SFO	java.lang.String
flightCode	null	java.lang.String
origination	null	java.lang.String
planeType	BOEING 737	java.lang.String
price	400.0	java.lang.Double

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.

The screenshot shows the Mule Studio interface with the 'Transform Message' tab selected. On the left, the 'Input' section displays an XML schema for a flight response. The 'Payload' section contains the following DataWeave script:

```
1@Xdw 1.0  
2 .setOutput application/java  
3 	xtype currency = :number {format: "###"}  
4 	xtype flight = :object {class: "com.mulesoft.training.Flight"}  
5  ---  
6 @payload.listAllFlightsResponse.*return map {  
7     destination: $.destination,  
8     price: $.price,  
9     planeType: upper $.planeType replace /(Boing)/ with "Boeing",  
10    departureDate: $.departureDate as :date {format: "yyyy/MM/dd"} as :string {format: "MM dd, yy"},  
11    availableSeats: $.emptySeats as :number  
12 } as :flight orderBy $.departureDate orderBy $.price distinctBy $.filter ($.availableSeats !=0)  
13
```

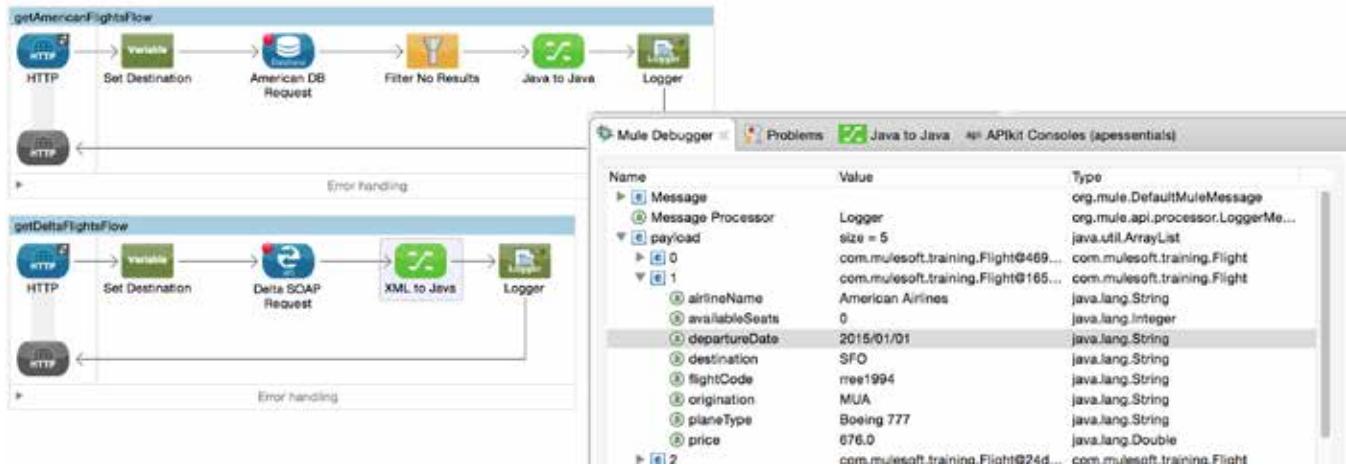
The 'Output' pane on the right shows the resulting payload structure:

Name	Type	Value
root	ArrayList	
[0]	Flight	<ul style="list-style-type: none"><li>airlineName : String</li><li>availableSeats : 10</li><li>departureDate : Oct 21, 2015</li><li>destination : String LAX</li><li>flightCode : String</li><li>origination : String</li><li>planeType : String BOEING 737</li><li>price : Double 199.99</li></ul>
[1]	Flight	<ul style="list-style-type: none"><li>airlineName : String</li><li>availableSeats : 23</li><li>departureDate : Oct 20, 2015</li></ul>

## 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.

- 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.

The screenshot shows the 'Java to Java' tab of the Mule Debugger. The 'Input' tab is selected. The payload is shown as a List<Map> with the following properties:

- airlineName : String
- code1 : String
- code2 : String
- fromAirport : String
- planeType : String
- price : Short
- seatsAvailable : String
- takeOffDate : Date
- toAirport : String

Below the payload, there are sections for 'Flow Variables' (destination : Map<String, String>) and 'Session Variables'. At the bottom, there are tabs for 'Context' and 'payload', with 'payload' being the active tab.

*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.*

- In the Transform section, leave the output set to Payload and of type application/java.
- Delete the existing DataWeave expression (the curly braces) and set it to payload.
- 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.

The screenshot shows the 'Output' tab of the Mule Debugger. The structure is defined as follows:

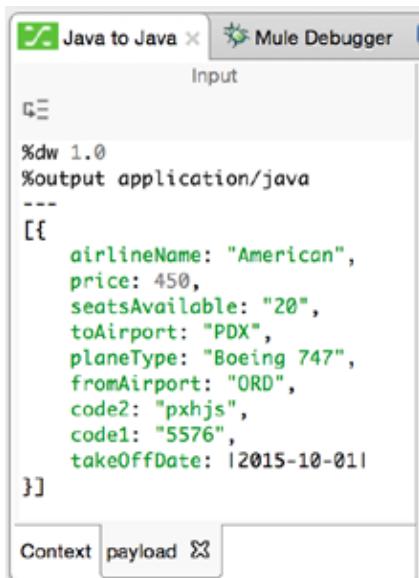
- root : ArrayList
- [0] : LinkedHashMap
  - airlineName : S ???? (String)
  - price : Integer 1 (Integer)
  - seatsAvailable : ???? (String)
  - toAirport : Strin ????? (String)
  - planeType : Stri ????? (String)
  - fromAirport : St ????? (String)
  - code2 : String ???? (String)
  - code1 : String ???? (String)
  - takeOffDate : D Wed Oct 01 12:00:00 (Date)

At the bottom, there are tabs for 'Structure' and 'Preview', with 'Structure' being the active tab.



## Add sample data and preview

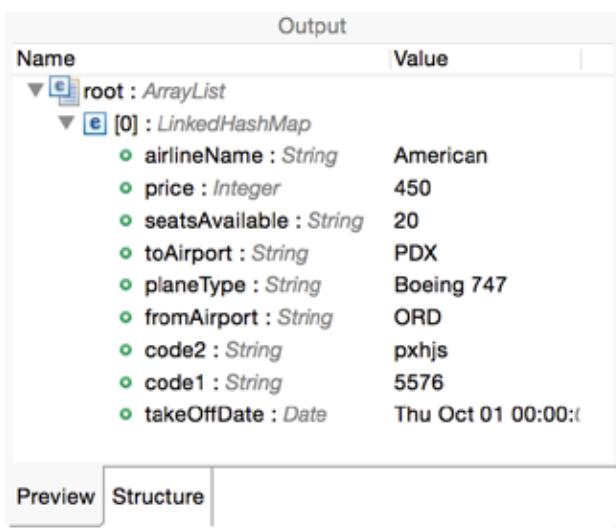
9. In the Input section of the Transform Message Properties view, click Payload: List<Map>.
10. Click the Sample Data button.
11. In the new payload tab, replace the “????” with sample values.



```
%dw 1.0
%output application/java
---
[{
    airlineName: "American",
    price: 450,
    seatsAvailable: "20",
    toAirport: "PDX",
    planeType: "Boeing 747",
    fromAirport: "ORD",
    code2: "pxhjs",
    code1: "5576",
    takeOffDate: "2015-10-01"
}]

```

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



Output	
Name	Value
root : ArrayList	
[0] : LinkedHashMap	
airlineName : String	American
price : Integer	450
seatsAvailable : String	20
toAirport : String	PDX
planeType : String	Boeing 747
fromAirport : String	ORD
code2 : String	pxhjs
code1 : String	5576
takeOffDate : Date	Thu Oct 01 00:00:00

## 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.

The screenshot shows the Eclipse IDE interface. On the left, the Package Explorer view displays the project structure under 'apsessentials'. It includes 'src/main/app (Flows)', 'src/main/api', and 'src/main/java'. Within 'src/main/java', there is a package 'com.mulesoft.training' containing 'jaxb.index', 'Flight.java', 'FlightArray.java', 'FlightRequest.java', and 'FlightSortTransformer.java'. Below these are 'src/main/resources', 'src/test/java', 'src/test/resources', 'mappings', 'Referenced Libraries', and 'APIkit [v1.7.1]'. The 'Flight.java' file is currently selected in the editor, showing its Java code:

```
13 @JsonProperty("code")
14 @XmlElement(name="code")
15 String flightCode;
16
17 @JsonProperty("origin")
18 @XmlElement(name="origin")
19 String origination;
20
21 @JsonProperty("emptySeats")
22 @XmlElement(name="emptySeats")
23 int availableSeats;
24
25 String departureDate;
26 String airlineName;
27 String destination;
28 double price;
29 String planeType;
```

## 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!

The screenshot shows the Mule Studio 'Transform Message' tool. The 'Input' tab contains the following DataWeave code:

```
%dw 1.0
%output application/java
---
[{
    airlineName: "American",
    price: 450,
    seatsAvailable: "20",
    toAirport: "PDX",
    planeType: "Boeing 747",
    fromAirport: "ORD",
    code2: "pxhjs",
    code1: "5576",
    takeOffDate: 2015-10-01
}]
```

The 'Output' tab shows the resulting payload map structure:

```
1 %dw 1.0
2 %output application/java
3 ---
4 payload map {
5     airlineName: $.airlineName
6 }
```

The 'Payload' tab shows the resulting JSON output:

```
[{"airlineName": "American"}]
```

The 'Structure' tab shows the output schema:

Name	Type	Value
root	ArrayList	
[0]	LinkedHashMap	airlineName : String American



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.*

The screenshot shows the Mule Studio interface with a payload map configuration and its resulting output.

**Payload Map Configuration:**

```
1 %dw 1.0
2 %output application/java
3 ---
4 payload map {
5     airlineName: $.airlineName,
6     departureDate: $.takeOffDate,
7     destination: $.toAirport,
8     availableSeats: $.seatsAvailable,
9     flightCode: $.code1 ++ $.code2,
10    origination: $.fromAirport,
11    planeType: $.planeType,
12    price: $.price
13 }
```

**Output Structure:**

Name	Type	Value
root	ArrayList	
[0]	LinkedHashMap	
airlineName	String	S American
departureDate	Date	Thu Oct 01 00:00:00 PDT 2015
destination	String	St PDX
availableSeats	Number	20
flightCode	String	Str 5576pxhjs
origination	String	ORD
planeType	String	Boeing 747
price	Integer	450

## 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.

The screenshot shows the Mule Debugger interface with the 'Java to Java' tab selected. The message payload is expanded to show its contents. The 'departureDate' entry is highlighted, showing its value as 'Thu Jan 01 00:00:00 PST 2015' and its type as 'java.util.Date'.

Name	Value	Type
► e Message		org.mule.DefaultMuleMessage
⑧ Message Processor	Logger	org.mule.api.processor.LoggerMessa...
▼ e payload	size = 5	java.util.ArrayList
► e 0	{airlineName=American Airlines, depa...	java.util.LinkedHashMap
▼ e 1	{airlineName=American Airlines, depa...	java.util.LinkedHashMap
► e 0	airlineName=American Airlines	java.util.LinkedHashMap\$Entry
▼ e 1	departureDate=Thu Jan 01 00:00:00...	java.util.LinkedHashMap\$Entry
⑧ key	departureDate	java.lang.String
► e value	Thu Jan 01 00:00:00 PST 2015	java.util.Date
► e 2	destination=SFO	java.util.LinkedHashMap\$Entry
► e 3	availableSeats=none	java.util.LinkedHashMap\$Entry
► e 4	flightCode=rree1994	java.util.LinkedHashMap\$Entry
► e 5	origination=MUA	java.util.LinkedHashMap\$Entry
► e 6	planeType=Boeing 777	java.util.LinkedHashMap\$Entry
► e 7	price=676	java.util.LinkedHashMap\$Entry
► e 2	airlineName=American Airlines, depa...	java.util.LinkedHashMap

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.

The screenshot shows the Mule Studio interface. On the left is the DataWeave editor with the following code:

```
1@ Xdm 1.0  
2 %output application/java  
3 %type flight = :object {class: "com.mulesoft.training.Flight"}  
4 ---  
5 payload map {  
6     airlineName: $.airlineName,  
7     departureDate: $.takeOffDate as :string {format: "yyyy/MM/dd"},  
8     destination: $.toAirport,  
9     availableSeats: $.seatsAvailable as :number unless $.seatsAvailable=="none" otherwise 0,  
10    flightCode: $.code1 ++ $.code2,  
11    origination: $.fromAirport,  
12    planeType: $.planeType,  
13    price: $.price as :number {format: "###.##"}  
14 } as :flight
```

On the right is the Preview tab, which displays the resulting data structure:

Name	Value
root : ArrayList	
[0] : Flight	
airlineName	American
availableSeats	20
departureDate	2015/10/01
destination	PDX
flightCode	5576pxhjs
origination	ORD
planeType	Boeing 747
price	450.0

## 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.

The screenshot shows the Mule Debugger tool. The payload structure is displayed in a tree view:

Name	Value	Type
Message		org.mule.DefaultMuleMessage
Message Processor	Logger	org.mule.api.processor.LoggerMe...
payload	size = 5	java.util.ArrayList
0	com.mulesoft.training.Flight@469...	com.mulesoft.training.Flight
1	com.mulesoft.training.Flight@165...	com.mulesoft.training.Flight
airlineName	American Airlines	java.lang.String
availableSeats	0	java.lang.Integer
departureDate	2015/01/01	java.lang.String
destination	SFO	java.lang.String
flightCode	rree1994	java.lang.String
origination	MUA	java.lang.String
planeType	Boeing 777	java.lang.String
price	676.0	java.lang.Double



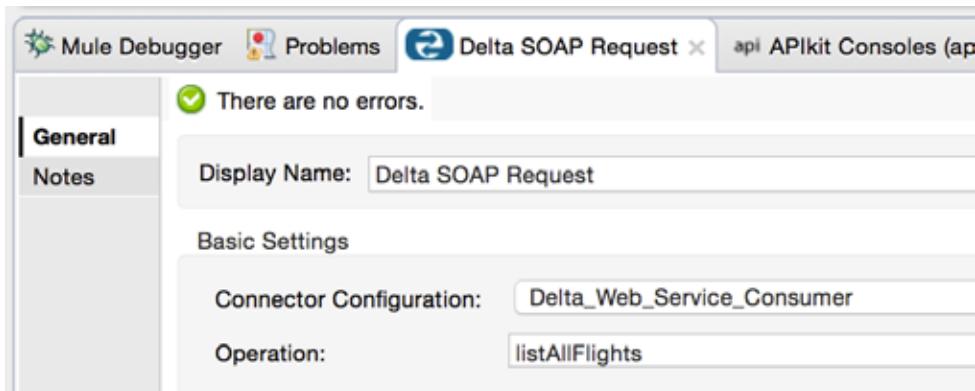
35. Stop the Mule Debugger.

## 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`.



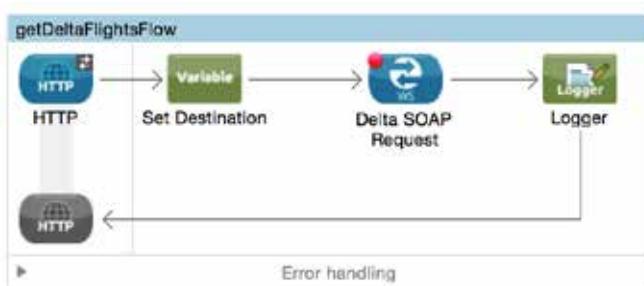
*Note: In the next walkthrough, you will use the Transform Message component and DataWeave to pass arguments to a SOAP web service.*

## Remove the existing transformation logic

39. Delete the XML to JAXB Object transformer.

40. Delete the Collection of Flights transformer.

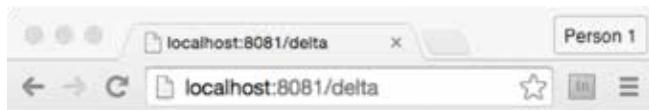
41. Delete the filter.



## 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.



This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
▼<ns2:listAllFlightsResponse
  xmlns:ns2="http://soap.training.mulesoft.com/">
  ▼<return>
    <airlineName>Delta</airlineName>
    <code>A1B2C3</code>
    <departureDate>2015/03/20</departureDate>
    <destination>SFO</destination>
    <emptySeats>40</emptySeats>
    <origin>MUA</origin>
    <planeType>Boing 737</planeType>
    <price>400.0</price>
  ▲</return>
  ▼<return>
    <airlineName>Delta</airlineName>
```

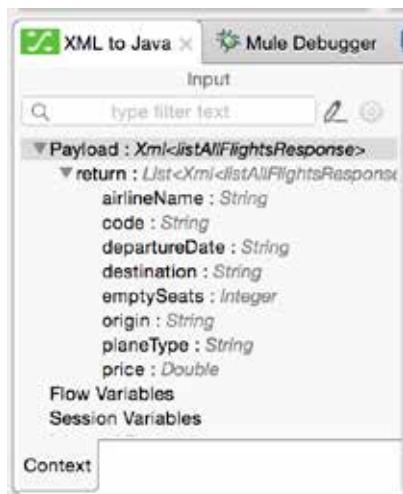
## Transform the XML to Java using DataWeave

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.

The screenshot shows the DataWeave preview window with the following details:

- Output:** Payload
- Payload:** %dw 1.0
 

```
1 %dw 1.0
2 %output application/java
3 ---
4 payload
```
- Structure:** Preview
- Output Structure:**

Name	Type	Value
root	LinkedHashMap	
listAllFlightsResponse	LinkedHashMap	
return	LinkedHashMap	
airlineName	String	????
code	String	????
departureDate	String	????
destination	String	????
emptySeats	String	1
origin	String	????
planeType	String	????
price	String	2

## Change the output structure

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

The screenshot shows the DataWeave preview window with the following details:

- Output:** Payload
- Payload:** %dw 1.0
 

```
1 %dw 1.0
2 %output application/java
3 ---
4 payload.listAllFlightsResponse
```
- Structure:** Preview
- Output Structure:**

Name	Type	Value
root	LinkedHashMap	
return	LinkedHashMap	
airlineName	String	????
code	String	????
departureDate	String	????
destination	String	????
emptySeats	String	1
origin	String	????
planeType	String	????
price	String	2

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

The screenshot shows the DataWeave preview window with the following details:

- Output:** Payload
- Payload:** %dw 1.0
 

```
1 %dw 1.0
2 %output application/java
3 ---
4 payload.listAllFlightsResponse.return
```
- Structure:** Preview
- Output Structure:**

Name	Type	Value
root	LinkedHashMap	
airlineName	String	????
code	String	????
departureDate	String	????
destination	String	????
emptySeats	String	1
origin	String	????
planeType	String	????
price	String	2



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.

```
%dw 1.0
%output application/java
---
payload.listAllFlightsResponse.return map {
    airlineName: $.airlineName
}
```

Name	Value
root : ArrayList	
[0] : LinkedHashMap	
[1] : LinkedHashMap	
[2] : LinkedHashMap	
[3] : LinkedHashMap	
[4] : LinkedHashMap	
[5] : LinkedHashMap	
[6] : LinkedHashMap	
[7] : LinkedHashMap	

Can not update preview. Validate your mappings.

Payload Structure Preview

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

```
%dw 1.0
%output application/java
---
payload.listAllFlightsResponse.*return map {
    airlineName: $.airlineName
}
```

Name	Value
root : ArrayList	
[0] : LinkedHashMap	airlineName : S ????

Payload Structure Preview

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

```

1 %dw 1.0
2 %output application/java
3 ---
4 payload.listAllFlightsResponse.*return map {
5   airlineName: $.airlineName,
6   departureDate: $.departureDate,
7   destination: $.destination,
8   availableSeats: $.emptySeats as :number,
9   flightCode: $.code,
10  origination: $.origin,
11  planeType: $.planeType replace /(Boing)/ with "Boeing",
12  price: $.price as :number {format: "###.##"}
13 }

```

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

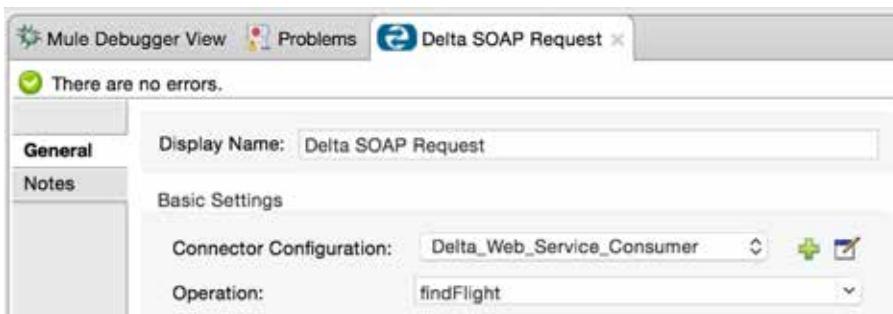
In this walkthrough, you will continue to work with the Delta flight data. You will:

- 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.

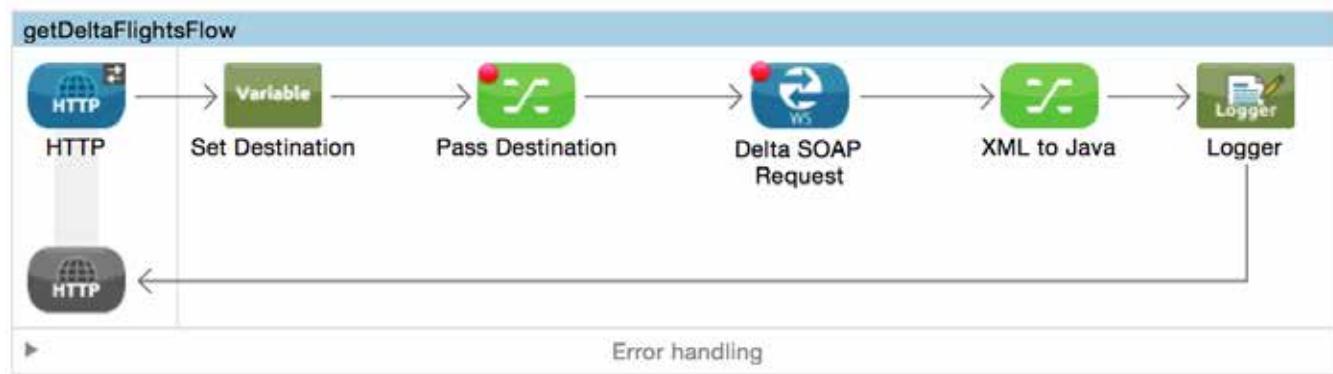
7. Look at the console; you should see a SoapFaultException because you are calling an operation that requires parameters but you have not passed it any.

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

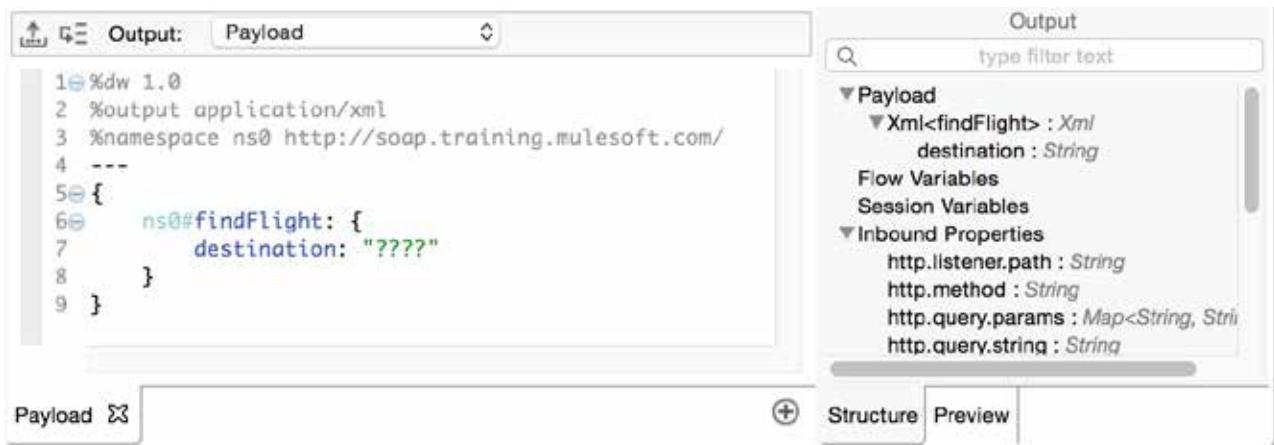
8. Stop the debugger.

## 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.

The screenshot shows the Mule Studio interface with the DataWeave editor open. The code pane contains:

```
1%dw 1.0
2 %output application/xml
3 %namespace ns0 http://soap.training.mulesoft.com/
4 ---
5{
6    ns0#findFlight: {
7        destination: "SFO"
8    }
9}
```

The preview pane shows the resulting XML output:

```
<?xml version='1.0' encoding='UTF-8'?>
<ns0:findFlight xmlns:ns0="http://soap.training.mulesoft.com/">
    <destination>SFO</destination>
</ns0:findFlight>
```

Below the editor are tabs for Payload, Structure, and Preview.

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

The screenshot shows the Mule Studio interface with the DataWeave editor open. The code pane contains:

```
1%dw 1.0
2 %output application/xml
3 %namespace ns0 http://soap.training.mulesoft.com/
4 ---
5{
6    ns0#findFlight: {
7        destination: flowVars.destination
8    }
9}
```

## 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.

The screenshot shows the Mule Studio interface with the Mule Debugger tool open. The table displays the following flight variables:

Name	Value	Type
Message		org.mule.DefaultMuleMessage
Message Processor	Logger	org.mule.api.processor.LoggerMe...
payload	size = 3	java.util.ArrayList
0	com.mulesoft.training.Flight@441573	com.mulesoft.training.Flight
airlineName	Delta	java.lang.String
availableSeats	40	java.lang.Integer
departureDate	2015/03/20	java.lang.String
destination	SFO	java.lang.String
flightCode	A1B2C3	java.lang.String
origination	MUA	java.lang.String
planeType	Boeing 737	java.lang.String
price	400.0	java.lang.Double
1	com.mulesoft.training.Flight@270...	com.mulesoft.training.Flight
2	com.mulesoft.training.Flight@34c...	com.mulesoft.training.Flight

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.

Mule Debugger		
Name	Value	Type
► [e] Message		org.mule.DefaultMuleMessage
⑧ Message Processor	Logger	org.mule.api.processor.LoggerMe...
▼ [e] payload	size = 3	java.util.ArrayList
▼ [e] 0	com.mulesoft.training.Flight@26b...	com.mulesoft.training.Flight
⑧ airlineName	Delta	java.lang.String
⑧ availableSeats	50	java.lang.Integer
⑧ departureDate	2015/06/11	java.lang.String
⑧ destination	CLE	java.lang.String
⑧ flightCode	A1B34S	java.lang.String
⑧ origination	MUA	java.lang.String
⑧ planeType	Boeing 707	java.lang.String
⑧ price	420.0	java.lang.Double
► [e] 1	com.mulesoft.training.Flight@512...	com.mulesoft.training.Flight
► [e] 2	com.mulesoft.training.Flight@7c8...	com.mulesoft.training.Flight

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.

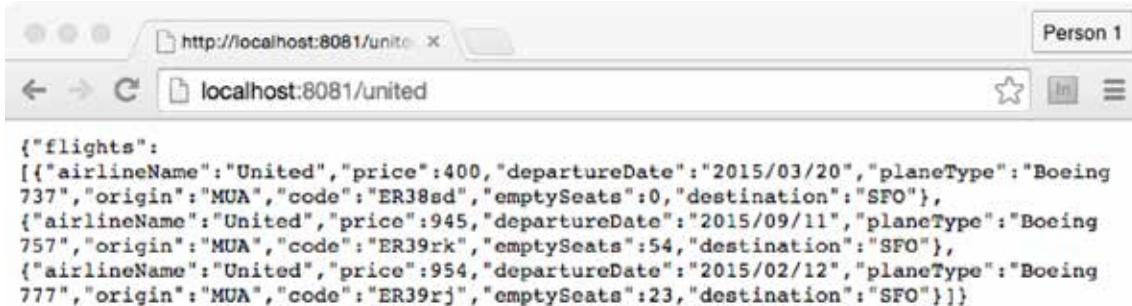


### Remove the existing United transformation logic

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

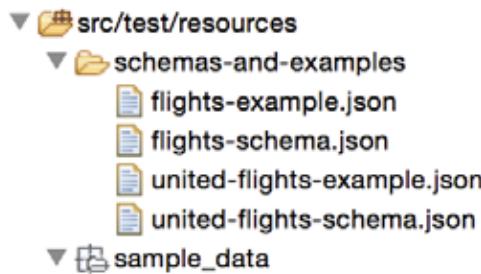
### Review the United flight data

4. Run the application and make a request to <http://localhost: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.

The screenshot shows two code editors side-by-side. The left editor is titled 'united-flights-example.json' and contains the following JSON code:

```
1 {
2   "flights": [
3     {
4       "airlineName": "United",
5       "price": 400,
6       "departureDate": "2015/03/20",
7       "planeType": "Boeing 737",
8       "origin": "MUA",
9       "code": "ER38sd",
10      "emptySeats": 0,
11      "destination": "SFO"
12    },
13    {
14      "airlineName": "United",
15      "price": 345.99,
16      "departureDate": "2015/02/11",
17      "planeType": "Boeing 737",
18      "origin": "MUA",
19      "code": "ER45if",
20      "emptySeats": 52,
21      "destination": "LAX"
22    },
23    {
24      "airlineName": "United",
25      "price": 345.99,
```

The right editor is titled 'united-flights-schema.json' and contains the following JSON schema code:

```
1 {
2   "$schema": "http://json-schema.org/draft-04/s
3   "type": "object",
4   "properties": {
5     "flights": {
6       "description": "An array that contains al
7       "type": "array",
8       "items": [
9         {
10           "type": "object",
11           "properties": {
12             "price": {
13               "description": "The flight price"
14               "type": "integer"
15             },
16             "airlineName": {
17               "description": "The airline name"
18               "type": "string"
19             },
20             "departureDate": {
21               "description": "The flight depart
22               "type": "string"
23             },
24             "planeType": {
25               "description": "The plane type"
```

## Transform the XML to Java using DataWeave

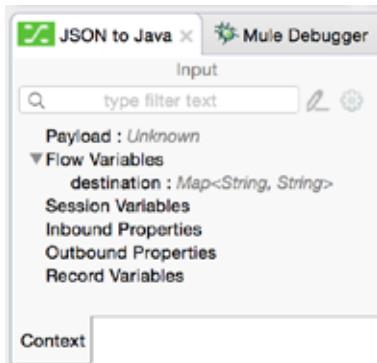
9. Add a Transform Message component between the filter and the Logger.

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



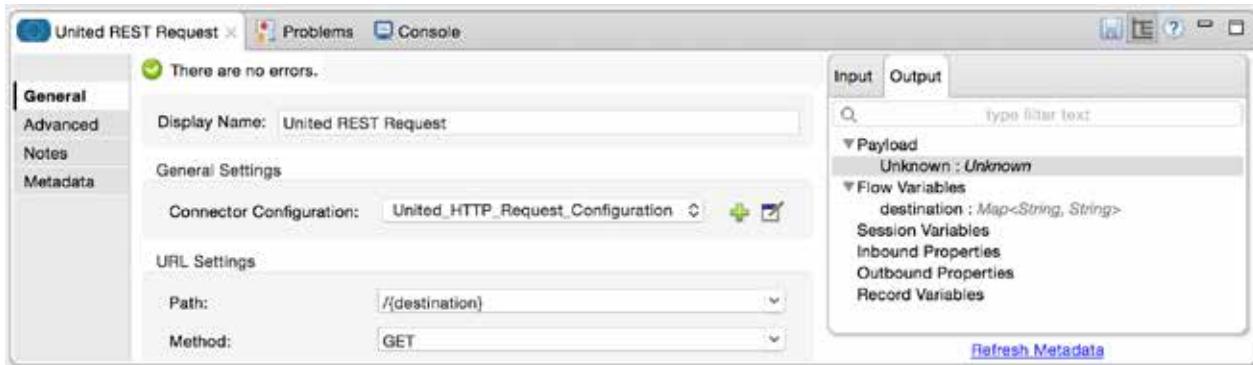
11. In the JSON to Java Properties view, leave the output set to Payload and of type application/java.

12. Look at the Input section of the Transform Message Properties view; the payload is of type unknown.



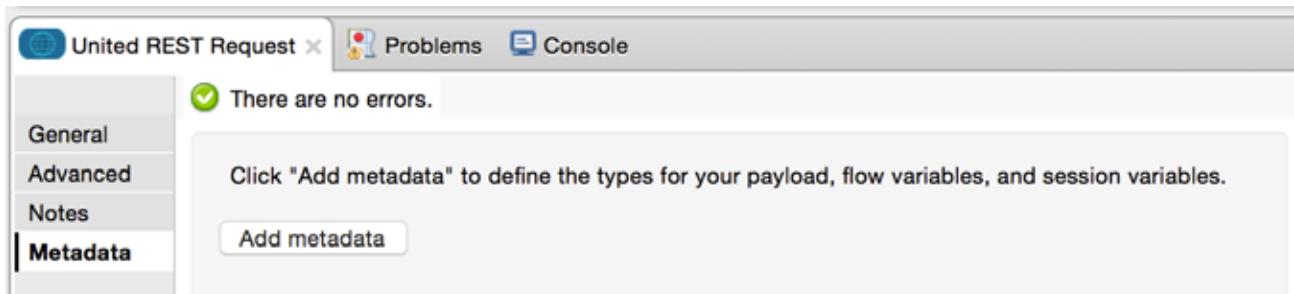
## Add metadata to the HTTP Request endpoint

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.



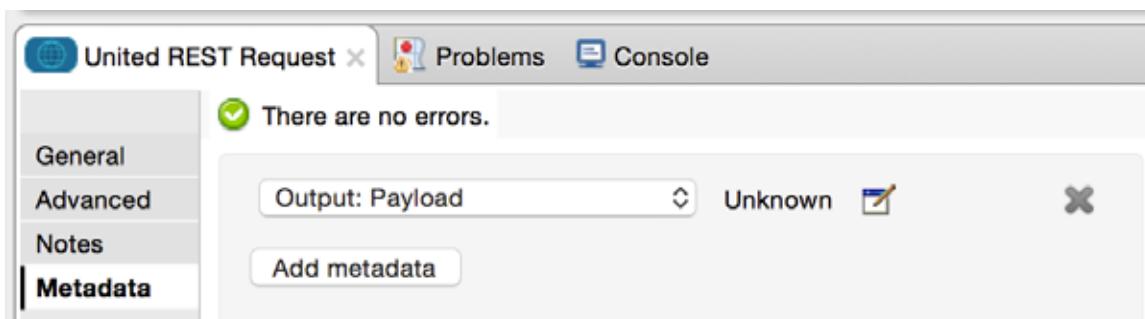
14. Click the Metadata link in the left-side navigation.

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.

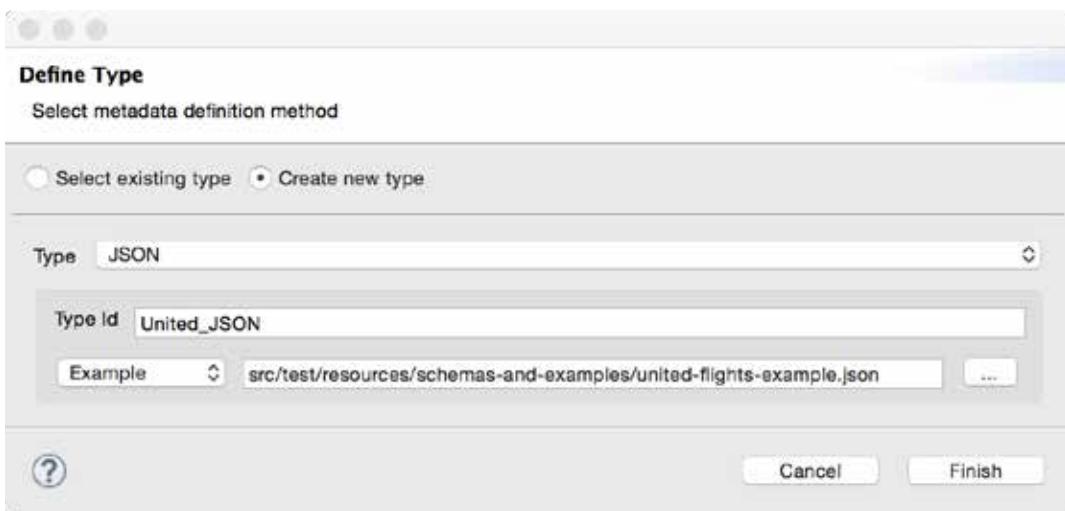
20. Set the type id to United\_JSON.

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.

Input   Output

Q type filter text

Payload

  Json : Json

    flights : List<Json>

      airlineName : String

      code : String

      departureDate : String

      destination : String

      emptySeats : integer

      origin : String

      planeType : String

      price : Integer

  Flow Variables

    destination : Map<String, String>

Refresh Metadata

## 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.

JSON to Java × Mule Debugger

Input

Q type filter text

Payload : Json

  flights : List<Json>

    airlineName : String

    code : String

    departureDate : String

    destination : String

    emptySeats : Integer

    origin : String

    planeType : String

    price : Integer

  Flow Variables

    destination : Map<String, String>

Session Variables

Context

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

Transform Message × Problems Console

Input

Q type filter text

Payload : Unknown

Flow Variables

Session Variables

Inbound Properties

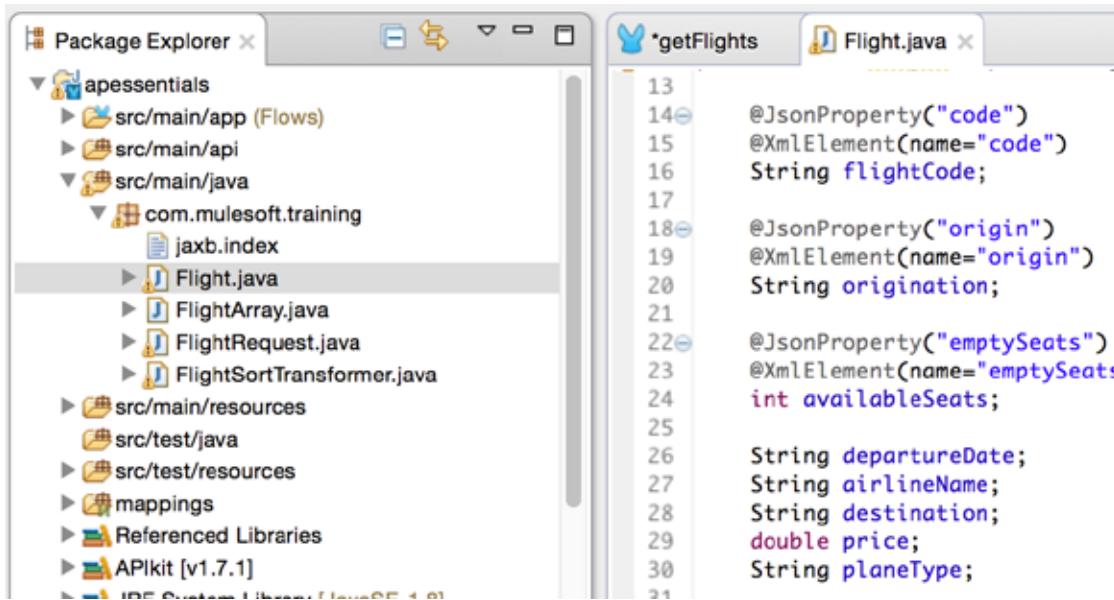
Output: Payload

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

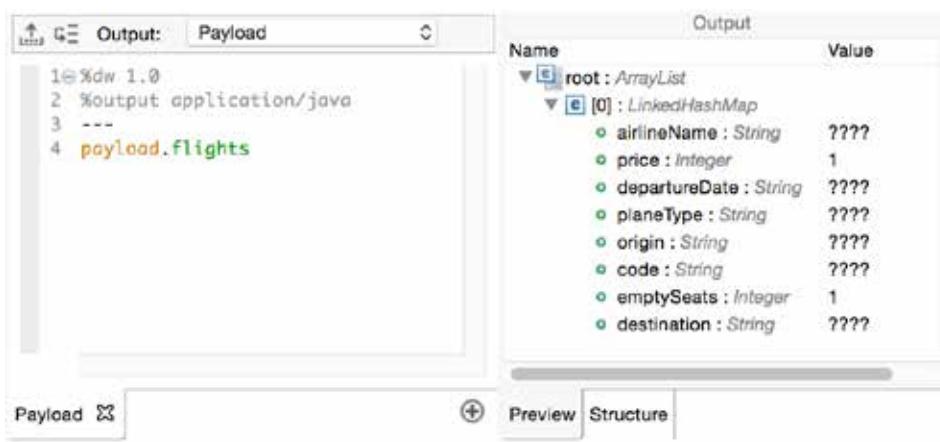


The screenshot shows the Eclipse IDE interface. On the left, the Package Explorer view displays the project structure under 'apessentials'. It includes 'src/main/app (Flows)', 'src/main/api', 'src/main/java' (which contains 'com.mulesoft.training' with files like 'jaxb.index', 'Flight.java', 'FlightArray.java', 'FlightRequest.java', and 'FlightSortTransformer.java'), 'src/main/resources', 'src/test/java', 'src/test/resources', 'mappings', 'Referenced Libraries', and 'APIkit [v1.7.1]'. On the right, the code editor window is titled 'Flight.java' and shows the following Java code:

```
13  
14 @JsonProperty("code")  
15 @XmlElement(name="code")  
16 String flightCode;  
17  
18 @JsonProperty("origin")  
19 @XmlElement(name="origin")  
20 String origination;  
21  
22 @JsonProperty("emptySeats")  
23 @XmlElement(name="emptySeats")  
24 int availableSeats;  
25  
26 String departureDate;  
27 String airlineName;  
28 String destination;  
29 double price;  
30 String planeType;  
31
```

## Change the output structure

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



The screenshot shows the MuleSoft Anypoint Studio interface. On the left, the DataWeave editor window has the following code:

```
1@%dw 1.0  
2 %output application/java  
3 ---  
4 payload.flights
```

On the right, the Output tab displays the resulting structure:

Name	Output	Value
root : ArrayList		
[0] : LinkedHashMap		
airlineName : String	????	
price : Integer	1	
departureDate : String	????	
planeType : String	????	
origin : String	????	
code : String	????	
emptySeats : Integer	1	
destination : String	????	

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.*

```
1 %dw 1.0
2 %output application/java
3 ---
4 @payload.flights map {
5   airlineName: $.airlineName,
6   departureDate: $.departureDate,
7   destination: $.destination,
8   availableSeats: $.emptySeats as :number,
9   flightCode: $.code,
10  origination: $.origin,
11  planeType: $.planeType,
12  price: $.price as :number {format: "###.##"}
13 }
```

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
```

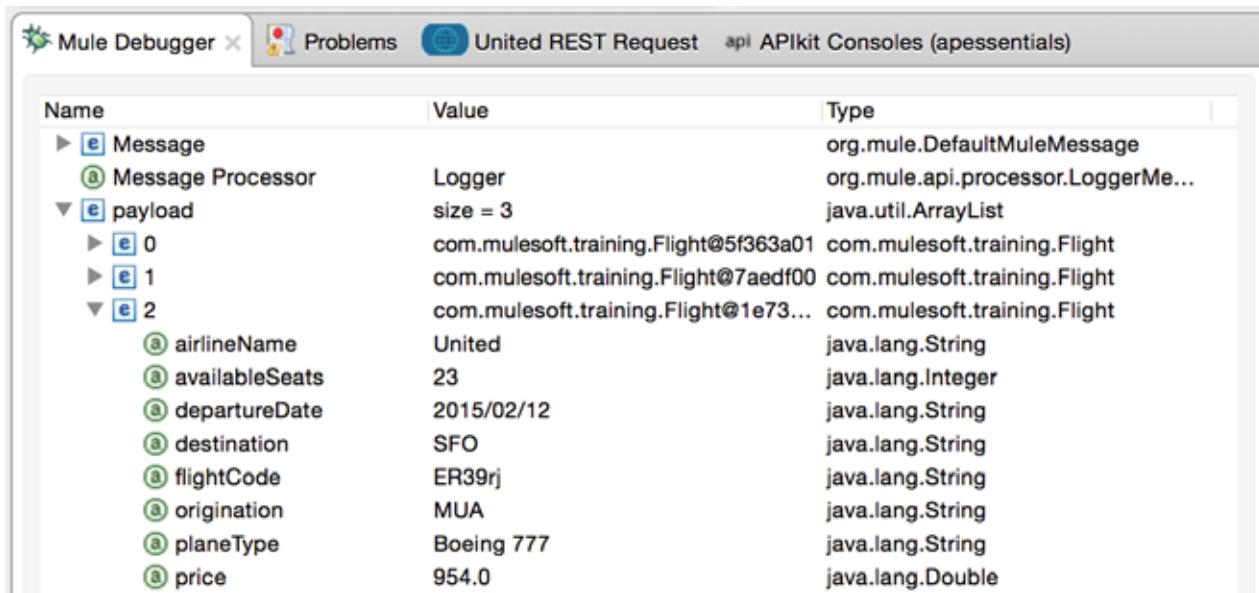
## Debug the application

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.



Name	Value	Type
► <b>e</b> Message		org.mule.DefaultMuleMessage
⑧ Message Processor	Logger	org.mule.api.processor.LoggerMe...
▼ <b>e</b> payload	size = 3	java.util.ArrayList
► <b>e</b> 0	com.mulesoft.training.Flight@5f363a01	com.mulesoft.training.Flight
► <b>e</b> 1	com.mulesoft.training.Flight@7aedf00	com.mulesoft.training.Flight
► <b>e</b> 2	com.mulesoft.training.Flight@1e73...	com.mulesoft.training.Flight
⑧ airlineName	United	java.lang.String
⑧ availableSeats	23	java.lang.Integer
⑧ departureDate	2015/02/12	java.lang.String
⑧ destination	SFO	java.lang.String
⑧ flightCode	ER39rj	java.lang.String
⑧ origination	MUA	java.lang.String
⑧ planeType	Boeing 777	java.lang.String
⑧ price	954.0	java.lang.Double

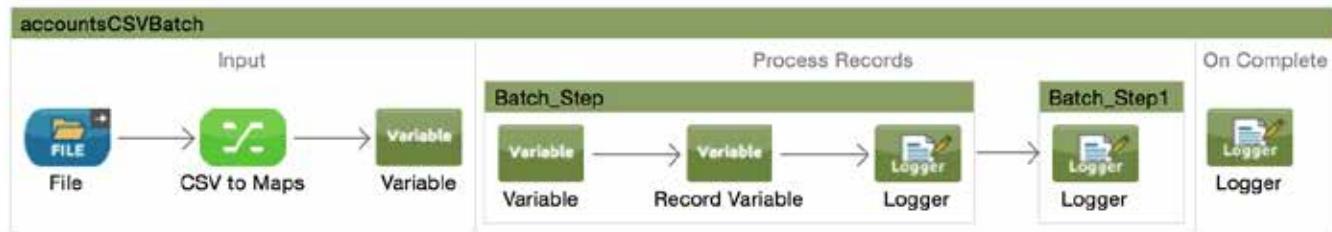
41. Stop the Mule Debugger.

*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 a 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.

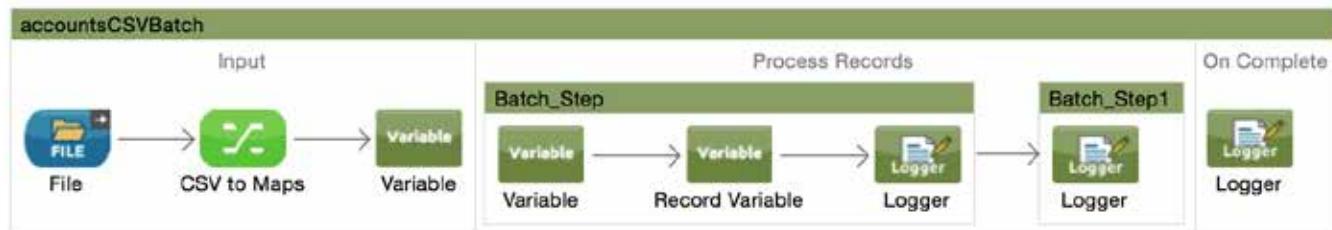


### Remove the existing transformation

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.



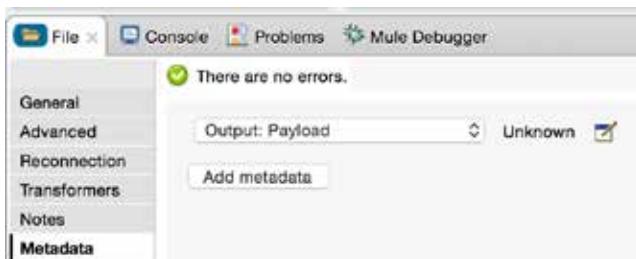
6. In the Input section of the CSV to Maps Properties view, make sure the Payload is specified as a List<Csv> with appropriate fields.

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



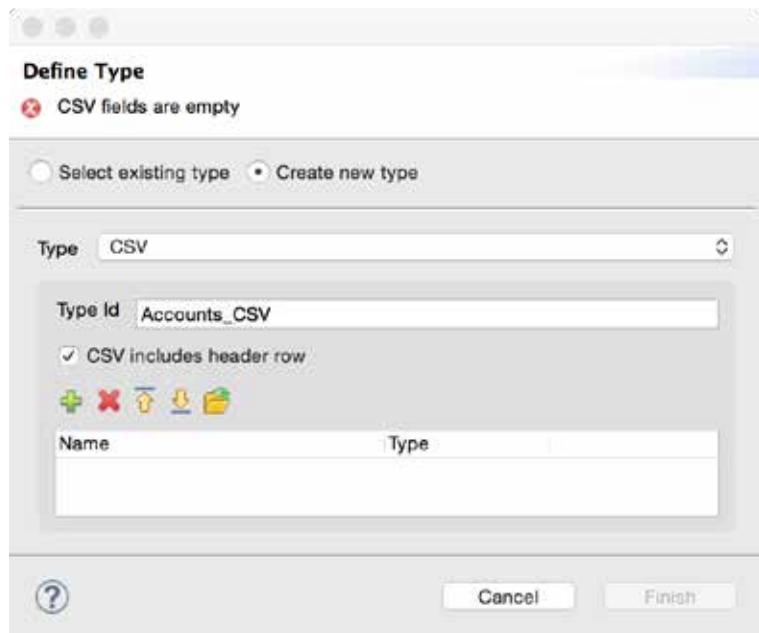
## Add File endpoint metadata

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.

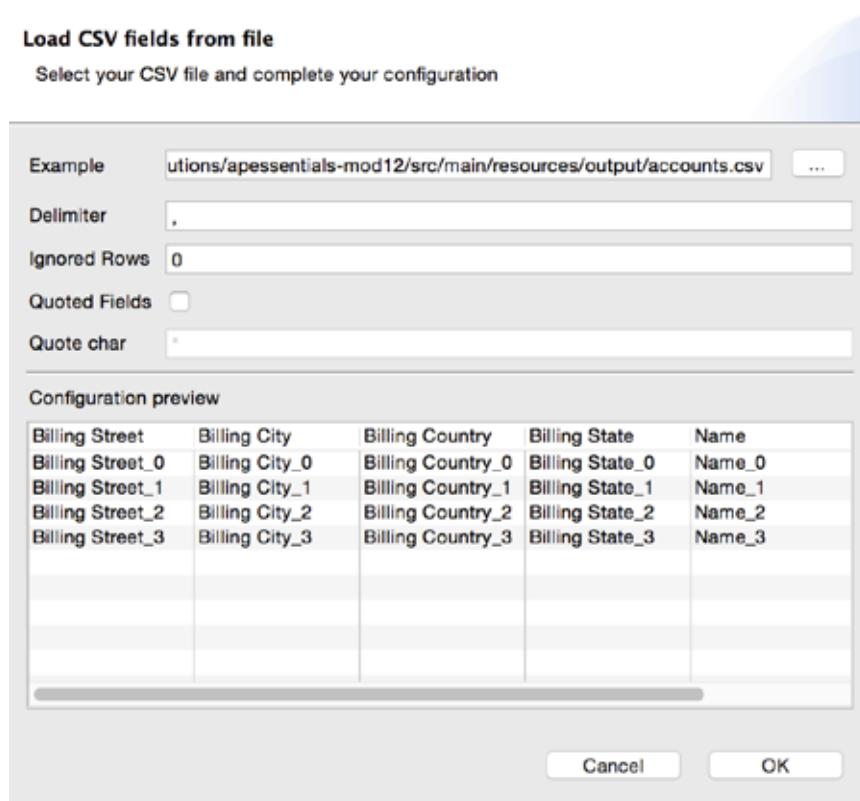


11. Click the Edit button next to the drop-down menu.
12. In the Define Type dialog box, select Create new type.
13. Change the type to CSV.
14. Set the Type Id to Accounts\_CSV.
15. Make sure CSV includes header row is checked.

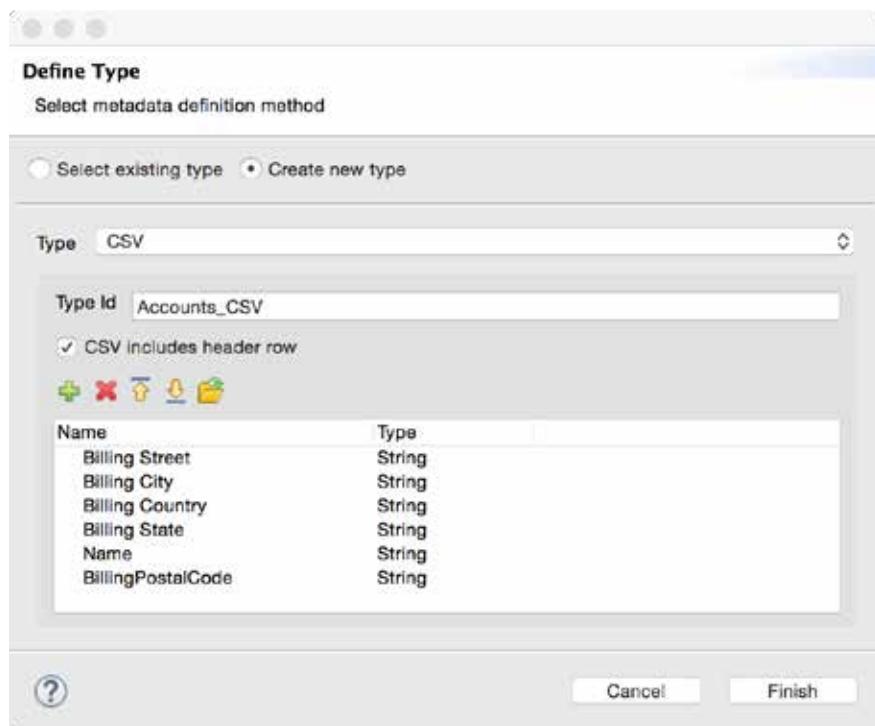
16. Click the Load from example button (the folder icon).



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 be an ArrayList of Maps.

The screenshot shows the Mule Studio interface with the 'CSV to Maps' component selected. The 'Properties' view is open, showing the 'Input' tab with 'Payload : List<Csv>' and the 'Output' tab with a preview of the DataWeave expression '%dw 1.0 %output application/java root : ArrayList [ { } ]'. The 'Preview' tab shows the resulting output as an ArrayList of LinkedHashMaps.

23. Change the DataWeave expression to payload.

24. Look at the preview.

The screenshot shows the Mule Debugger interface with the tab 'CSV to Maps' selected. On the left, there's a tree view under 'Payload' showing fields like Billing City, Billing Country, etc. In the center, the 'Output' section is set to 'Payload' and displays the following text:  
1 %dw 1.0  
2 %output application/java  
3 ---  
4 payload

On the right, the 'Output' table shows the structure of the payload:

Name	Value
root	ArrayList
[0]	LinkedHashMap
Billing Street	String ?????
Billing City	String ?????
Billing Country	String ?????
Billing State	String ?????
Name	String ?????
BillingPostalCode	String ?????

Below the table are tabs for 'Preview' and 'Structure'.

## 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.

The screenshot shows the Mule Properties view with the 'Mule Debugger' tab selected. It lists variables and their values:

Name	Value	Type
Message		org.mule.DefaultMuleMessage
Message Processor	Variable	org.mule.transformer.simple
payload	size = 3	java.util.ArrayList
0	{Billing Street=111 Boulevard Hausmann, Bi...	java.util.LinkedHashMap
1	{Billing Street=400 South St, Billing City=Sa...	java.util.LinkedHashMap
2	{Billing Street=777 North St, Billing City=Sa...	java.util.LinkedHashMap
0	Billing Street=777 North St	java.util.LinkedHashMap\$Entry
1	Billing City=San Francisco	java.util.LinkedHashMap\$Entry
2	Billing Country=USA	java.util.LinkedHashMap\$Entry
3	Billing State=CA	java.util.LinkedHashMap\$Entry
4	Name=Cat Park Industries	java.util.LinkedHashMap\$Entry
5	BillingPostalCode=91156	java.util.LinkedHashMap\$Entry

29. Stop the Mule runtime.