

# Module 10: Writing DataWeave Transformations

The screenshot shows the Mule Studio interface with a DataWeave transformation and its output payload.

**DataWeave Transformation:**

```
1@ %dw 1.0
2 %output application/java
3 %namespace ns0 http://soap.training.mulesoft.com/
4 %type currency = :string {format: "###.##"}
5 %type flight = :object {class: "com.mulesoft.training.Flight"}
6 ---
7@ flights: payload.ns0:listAllFlightsResponse.*return map {
8   destination: $.destination,
9   price: $.price as :number as :currency,
10  planeType: upper ($.planeType replace /(Boing)/ with "Boeing"),
11  departureDate: $.departureDate as :date {format: "yyyy/MM/dd"}
12  as :string {format: "MMM dd, yyyy"},
13  availableSeats: $.emptySeats as :number,
14  //totalSeats: getNumSeats($.planeType)
15  totalSeats: lookup("getTotalSeatsFlow",{type: $.planeType})
16 }
```

**Output Payload:**

Name	Value
root	LinkedHashMap
flights	ArrayList
[0]	LinkedHashMap
destination	SFO
price	400.00
planeType	BOEING 737
departureDate	Oct 20, 2015
availableSeats	40
Unknown	
[1]	LinkedHashMap
destination	LAX
price	199.99
planeType	BOEING 737

## Objectives:

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

## Walkthrough 10-1: Write your first DataWeave transformation

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

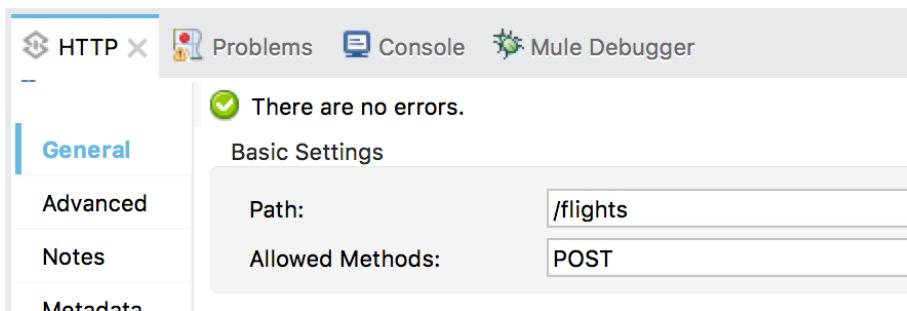
- Create a new flow that receives POST requests.
- Write a DataWeave expression to transform the data to Java, XML, or JSON.
- Add sample data and use live preview.
- Save the transformation in an external file.

The screenshot shows the Mule Studio interface. On the left, the code editor displays `implementation.xml` with several code snippets highlighted in blue. On the right, the resources panel shows files like `flights-DEV.properties`, `json_flight_playground.dwl` (which is selected), and `log4j2.xml`. Below the resources panel, the contents of `json_flight_playground.dwl` are shown in a code editor:

```
1 *%dw 1.0
2 %output application/xml
3 ---
4 payload
```

### Create a new flow

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

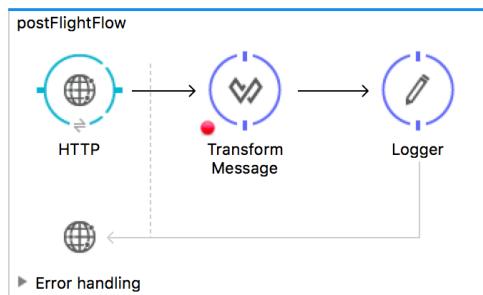


6. Add a Logger to the flow.

### Add a DataWeave Transform Message component

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

8. Add a breakpoint to it.



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

Output Payload ▾ Preview

```

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

## Post data to the flow

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

```

{
  "airline": "Delta",
  "flightCode": "ER0945",
  "fromAirportCode": "PDX",
  "toAirportCode": "CLE",
  "departureDate": "June 1, 2016",
  "emptySeats": 24,
  "totalSeats": 350,
  "price": 450,
  "planeType": "Boeing 747"
}
  
```

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

Name	Value	Type
» <b>e</b> DataType	SimpleDataType{type=or...}	org.mule.transformer.types.SimpleDataType
» <b>a</b> Exception	null	
» <b>e</b> Message		org.mule.DefaultMuleMessage
» <b>a</b> Message Processor	Transform Message	com.mulesoft.weave.mule.WeaveMessag...
» <b>e</b> Payload (mimeType="application/json",...)	org.glassfish.grizzly.util...	org.glassfish.grizzly.utils.BufferInputStream

20. Look at the inbound properties; you should see the content-type is set to application/json.

21. Step to the Logger; you should see the payload now has a mime-type of application/java and is a LinkedHashMap.

Name	Value	Type
» <b>a</b> ivessage		org.mule.DefaultMuleMessage
» <b>a</b> Message Processor	Logger	org.mule.api.processor.LoggerMes...
» <b>e</b> Payload (mimeType="application/java",...)	size = 9 airline=Delta flightCode=ER0945 fromAirportCode=PDX toAirportCode=CLE departureDate=June 1, 2016 emptySeats=24 totalSeats=350 price=450 planeType=Boeing 747	java.util.LinkedHashMap
» <b>e</b> 0	airline=Delta	java.util.LinkedHashMap\$Entry
» <b>e</b> 1	flightCode=ER0945	java.util.LinkedHashMap\$Entry
» <b>e</b> 2	fromAirportCode=PDX	java.util.LinkedHashMap\$Entry
» <b>e</b> 3	toAirportCode=CLE	java.util.LinkedHashMap\$Entry
» <b>e</b> 4	departureDate=June 1, 2016	java.util.LinkedHashMap\$Entry
» <b>e</b> 5	emptySeats=24	java.util.LinkedHashMap\$Entry
» <b>e</b> 6	totalSeats=350	java.util.LinkedHashMap\$Entry
» <b>e</b> 7	price=450	java.util.LinkedHashMap\$Entry
» <b>e</b> 8	planeType=Boeing 747	java.util.LinkedHashMap\$Entry

22. Click the Resume button.

## Change output data type to JSON

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

24. Debug the project.

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

26. In the Mule Debugger, step to the Logger; you should see the payload has a mime-type of application/json and is a Weave ByteArraySeekableStream.

Name	Value	Type
» <b>e</b> DataType	SimpleDataType{type=or...}	org.mule.transformer.types.SimpleDataType
» <b>a</b> Exception	null	
» <b>e</b> Message		org.mule.DefaultMuleMessage
» <b>a</b> Message Processor	Transform Message	com.mulesoft.weave.mule.WeaveMessageProc...
» <b>e</b> Payload (mimeType="application/json",...)	org.glassfish.grizzly.util...	org.glassfish.grizzly.utils.BufferInputStream

27. Step to the end of the application.

## Change output data type to XML

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

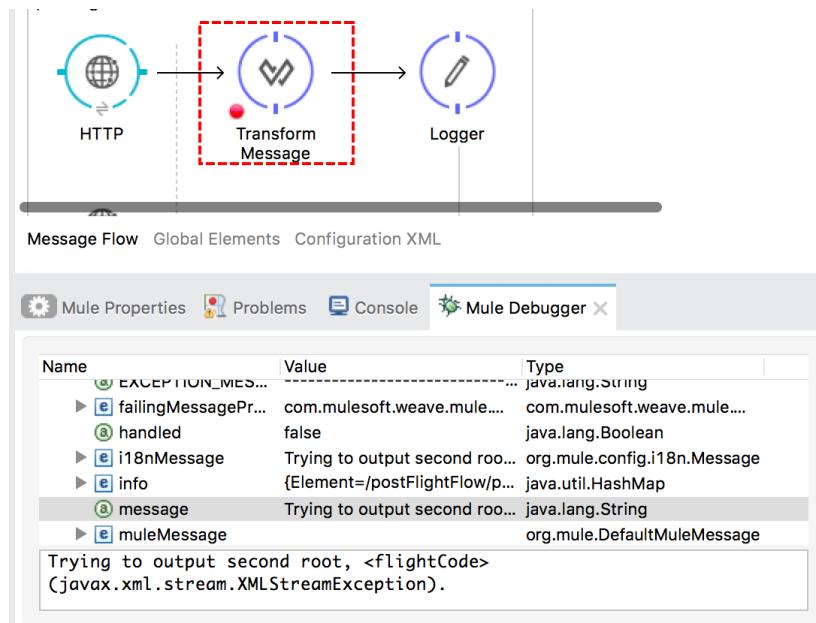
29. Debug the project.

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

31. In the Mule Debugger, click Next processor; you should get an exception.

32. Drill-down into the exceptionThrown and locate the message; you should see there is a

DataWeave error when trying to output the second root.



33. Click Resume.

34. Stop the project.

## Add input metadata and sample data

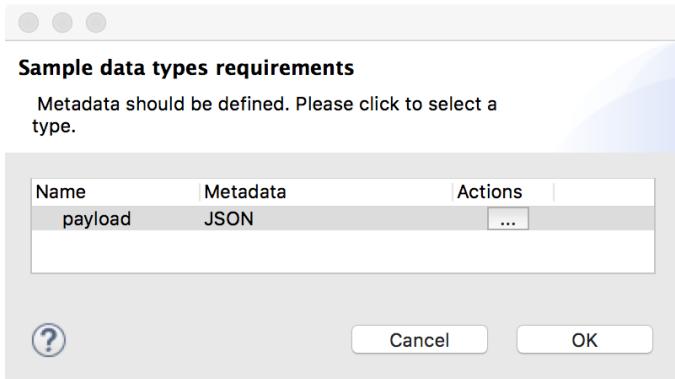
35. In the Transform Message properties view, change the output type from application/xml back to application/json.

36. Click the Preview button.

37. Click the Create required sample data to execute preview link.

38. In the Sample data types requirements dialog box, click the word Unknown under Metadata.

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



40. Click the button under Actions.

41. In the Select metadata type dialog box, click the Add button.

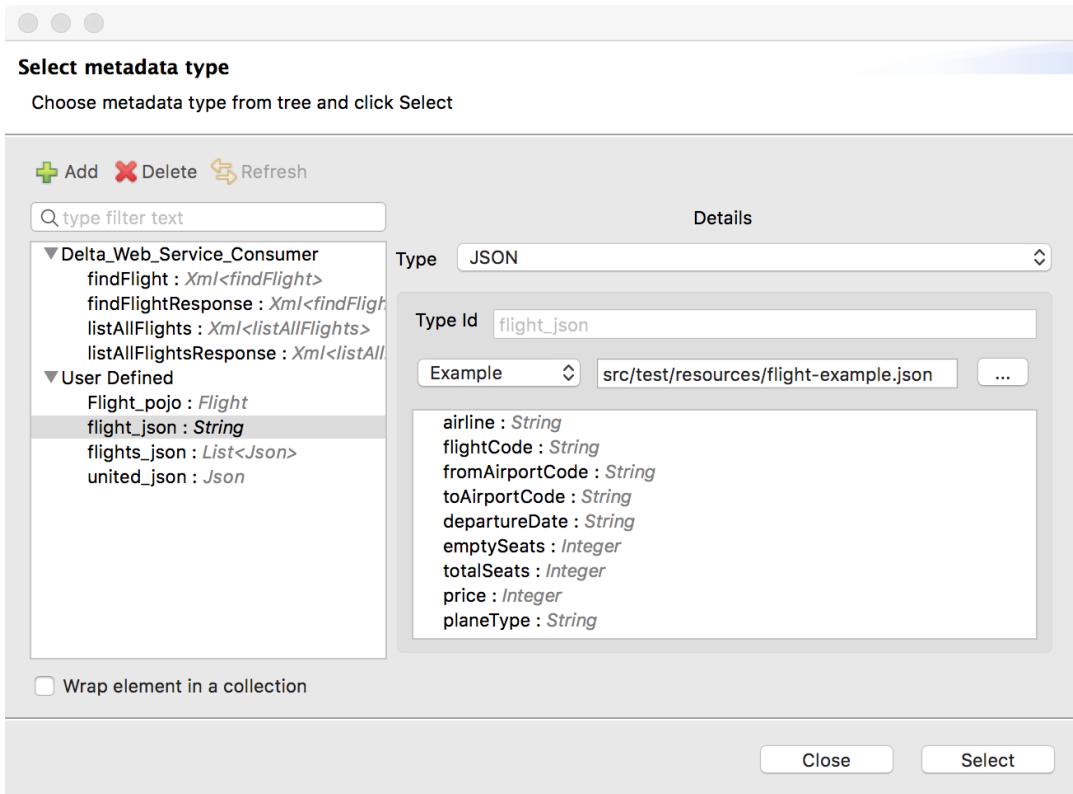
42. In the Create new type dialog box, set the type id to flight\_json and click Create type.

43. In the Select metadata dialog box, set the type to JSON.

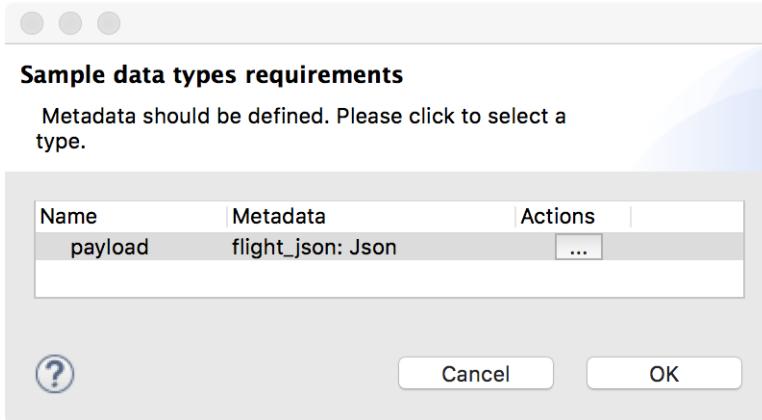
44. Select Example and browse to and select the flight-example.json file in src/main/resources.

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

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



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



## Preview sample data and sample output

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

```

flight-example.json
{
  "airline": "United",
  "flightCode": "ER38sd",
  "fromAirportCode": "LAX",
  "toAirportCode": "SFO",
  "departureDate": "May 21, 2016",
  "emptySeats": 0,
  "totalSeats": 200,
  "price": 199,
  "planeType": "Boeing 737"
}

Context payload

```

Output Payload application/json

```

1 %dw 1.0
2 %output application/json
3 ---
4 payload

```

```

{
  "airline": "United",
  "flightCode": "ER38sd",
  "fromAirportCode": "LAX",
  "toAirportCode": "SFO",
  "departureDate": "May 21, 2016",
  "emptySeats": 0,
  "totalSeats": 200,
  "price": 199,
  "planeType": "Boeing 737"
}

```

## Change the output type

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

The screenshot shows the Mule Studio interface with the following sections:

- flight-example.json**: The JSON input file content is shown, defining a flight with various properties like airline, flightCode, and price.
- Output**: The output configuration is set to `application/java`.
- Preview**: The preview pane displays the resulting LinkedHashMap object with its properties and values.

Name	Value
root	LinkedHashMap
airline	United
flightCode	ER38sd
fromAirportCode	LAX
toAirportCode	SFO
departureDate	May 21, 2016
emptySeats	0
totalSeats	200
price	199
planeType	Boeing 737

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

The screenshot shows the Mule Studio interface with the following sections:

- Output**: The output configuration is set to `application/xml`, indicated by a warning icon.
- The XML transformation code is displayed:

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

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

The screenshot shows the "List of errors" dialog box with the following details:

- List of errors**: A header with three dots.
- Select an error to see details below**: A placeholder text.
- Table**: A table with two columns: **Name** and **Target**. One row is highlighted with a yellow warning icon, showing the error message: `javax.xml.stream.XMLStreamException: Trying to output seco... Payload`.
- Message**: Below the table, the full error message is shown: `javax.xml.stream.XMLStreamException: Trying to output second root, <flightCode>`.
- OK**: A button at the bottom right of the dialog.

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

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

## Save the transformation in an external file

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

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

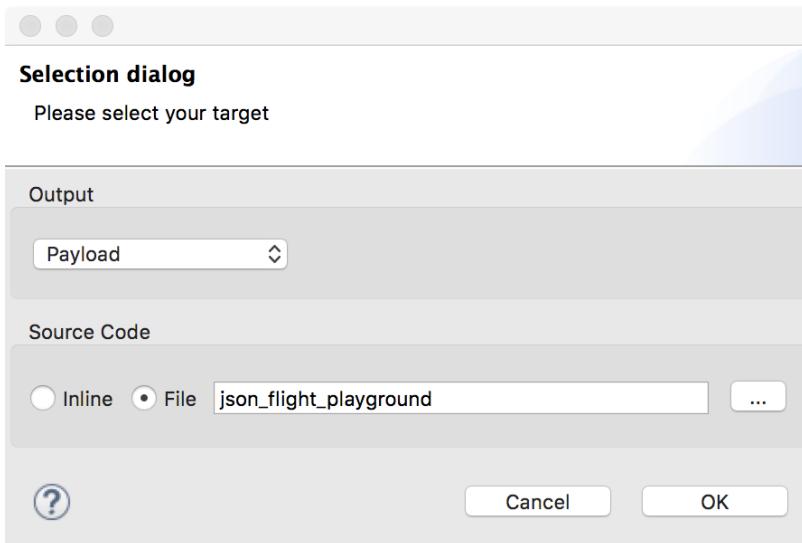
61. Switch back to the Message Flow view.
62. In the Transform Message properties view, click the Edit current target button (the pencil) above the code.

Output Payload ▾  

---

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

63. In the Selection dialog dialog box, change the source code selection form inline to file.
64. Set the file name to json\_flight\_playground.



65. Click OK.

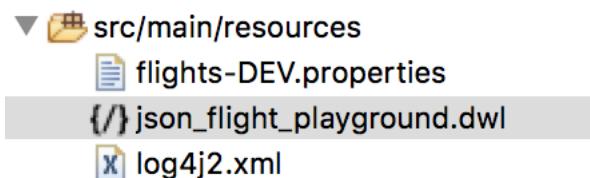
## Locate and review the code

66. Switch to Configuration XML view.

67. Scroll down and look at the new code for the transformation in postFlightFlow.

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

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



69. Open json\_flight\_playground.dwl.

70. Review and then close the file.

```
implementation json_flight_playground.dwl
1%dw 1.0
2%output application/java
3---
4payload
```

71. Return to Message Flow view in implementation.xml.

72. Save the file.

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

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

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

The screenshot shows the 'Output' tab of a transformation message. The code is as follows:

```
1 %dw 1.0
2 %output application/xml
3 ---
4 data: {
5   hub: "MUA",
6   flight @{airline: payload.airline}: {
7     code: payload.toAirportCode }
8 }
```

The preview pane shows the resulting XML output:

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

### Write expressions to transform JSON to various Java structures

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

The screenshot shows the 'Output' tab of a transformation message. The code is as follows:

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

A context menu is open over the 'payload' variable, showing options like 'airline', 'departureDate', 'emptySeats', etc. A preview table is shown to the right:

Name	Value
root : LinkedHashMap	
airline : String	United
flightCode : String	ER38sd
departureDate : string	LAX
emptySeats : number	SFO
flightCode : string	May 21, 20
fromAirportCode : string	0
planeType : string	200
price : number	199
toAirportCode : string	Boeing 737
totalSeats : number	

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

The screenshot shows the 'Output' tab of a transformation message. The code is as follows:

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

The preview table shows the result:

Name	Value
root : Integer	199

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

Output Payload ▾ Preview

<pre> 1 %dw 1.0 2 %output application/java 3 --- 4 data: payload.price       </pre>	<table border="1"> <thead> <tr> <th>Name</th> <th>Value</th> </tr> </thead> <tbody> <tr> <td>root : LinkedHashMap</td> <td></td> </tr> <tr> <td>  data : Integer</td> <td>199</td> </tr> </tbody> </table>	Name	Value	root : LinkedHashMap		data : Integer	199
Name	Value						
root : LinkedHashMap							
data : Integer	199						

5. Change the DataWeave expression to data: payload.

Output Payload ▾ Preview

<pre> 1 %dw 1.0 2 %output application/java 3 --- 4 data: payload       </pre>	<table border="1"> <thead> <tr> <th>Name</th> <th>Value</th> </tr> </thead> <tbody> <tr> <td>root : LinkedHashMap</td> <td></td> </tr> <tr> <td>  data : LinkedHashMap</td> <td></td> </tr> <tr> <td>    airline : String</td> <td>United</td> </tr> <tr> <td>    flightCode : String</td> <td>ER38sd</td> </tr> <tr> <td>    fromAirportCode : String</td> <td>LAX</td> </tr> <tr> <td>    toAirportCode : String</td> <td>SFO</td> </tr> <tr> <td>    departureDate : String</td> <td>May 21, 2016</td> </tr> <tr> <td>    emptySeats : Integer</td> <td>0</td> </tr> <tr> <td>    totalSeats : Integer</td> <td>200</td> </tr> <tr> <td>    price : Integer</td> <td>199</td> </tr> <tr> <td>    planeType : String</td> <td>Boeing 737</td> </tr> </tbody> </table>	Name	Value	root : LinkedHashMap		data : LinkedHashMap		airline : String	United	flightCode : String	ER38sd	fromAirportCode : String	LAX	toAirportCode : String	SFO	departureDate : String	May 21, 2016	emptySeats : Integer	0	totalSeats : Integer	200	price : Integer	199	planeType : String	Boeing 737
Name	Value																								
root : LinkedHashMap																									
data : LinkedHashMap																									
airline : String	United																								
flightCode : String	ER38sd																								
fromAirportCode : String	LAX																								
toAirportCode : String	SFO																								
departureDate : String	May 21, 2016																								
emptySeats : Integer	0																								
totalSeats : Integer	200																								
price : Integer	199																								
planeType : String	Boeing 737																								

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

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

Output Payload ▾ Preview

<pre> 1 %dw 1.0 2 %output application/java 3 --- 4 data: { 5   hub: "MUA" 6 }       </pre>	<table border="1"> <thead> <tr> <th>Name</th> <th>Value</th> </tr> </thead> <tbody> <tr> <td>root : LinkedHashMap</td> <td></td> </tr> <tr> <td>  data : LinkedHashMap</td> <td></td> </tr> <tr> <td>    hub : String</td> <td>MUA</td> </tr> </tbody> </table>	Name	Value	root : LinkedHashMap		data : LinkedHashMap		hub : String	MUA
Name	Value								
root : LinkedHashMap									
data : LinkedHashMap									
hub : String	MUA								

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

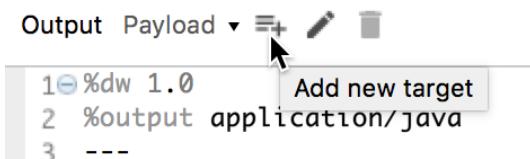
9. Add a field called airline and set it to the airline property of the payload.

Output Payload ▾ Preview

<pre> 1 %dw 1.0 2 %output application/java 3 --- 4 data: { 5   hub: "MUA", 6   code: payload.toAirportCode, 7   airline: payload.airline 8 }       </pre>	<table border="1"> <thead> <tr> <th>Name</th> <th>Value</th> </tr> </thead> <tbody> <tr> <td>root : LinkedHashMap</td> <td></td> </tr> <tr> <td>  data : LinkedHashMap</td> <td></td> </tr> <tr> <td>    hub : String</td> <td>MUA</td> </tr> <tr> <td>    code : String</td> <td>SFO</td> </tr> <tr> <td>    airline : String</td> <td>United</td> </tr> </tbody> </table>	Name	Value	root : LinkedHashMap		data : LinkedHashMap		hub : String	MUA	code : String	SFO	airline : String	United
Name	Value												
root : LinkedHashMap													
data : LinkedHashMap													
hub : String	MUA												
code : String	SFO												
airline : String	United												

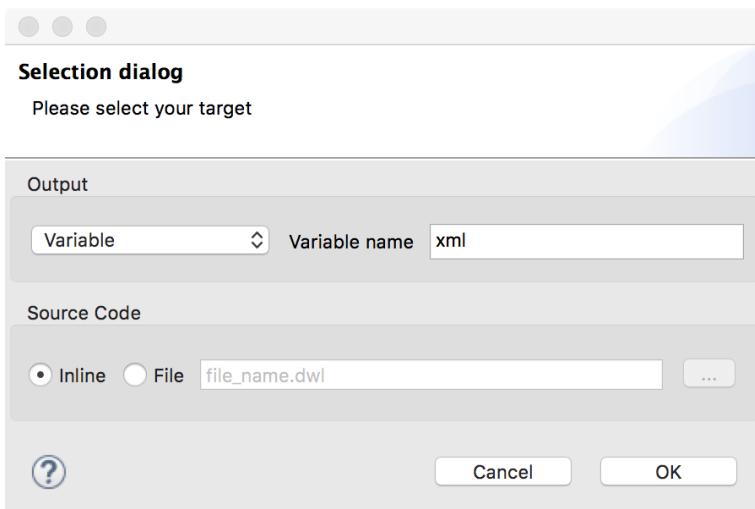
## Create a second transformation with the same component

10. Click the Add new target button.



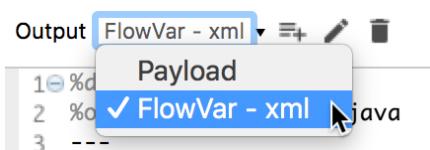
11. In the Selection dialog dialog box, leave the output set to variable.

12. Set the variable name to xml.



13. Click OK.

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



## Write an expression to output data as XML

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

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

Output FlowVar - xml ▾ Preview

```
1@%dw 1.0
2 %output application/xml
3 ---
4 data: payload
```

<?xml version='1.0' encoding='UTF-8'?>
<data>
 <airline>United</airline>
 <flightCode>ER38sd</flightCode>
 <fromAirportCode>LAX</fromAirportCode>
 <toAirportCode>SF0</toAirportCode>
 <departureDate>May 21, 2016</departureDate>
 <emptySeats>0</emptySeats>
 <totalSeats>200</totalSeats>
 <price>199</price>
 <planeType>Boeing 737</planeType>
</data>

21. In the output section menu bar, click the drop-down arrow next to FlowVar – xml and select Payload.

22. Copy the DataWeave expression you just wrote.

23. Switch back to FlowVar – xml and replace the DataWeave expression with the one you just copied.

Output FlowVar - xml ▾ Preview

```
1@%dw 1.0
2 %output application/xml
3 ---
4@data: {
5   hub: "MUA",
6   code: payload.toAirportCode,
7   airline: payload.airline
8 }
```

<?xml version='1.0' encoding='UTF-8'?>
<data>
 <hub>MUA</hub>
 <code>SF0</code>
 <airline>United</airline>
</data>

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

Output FlowVar - xml ▾ Preview

```
1@%dw 1.0
2 %output application/xml
3 ---
4@data: {
5   hub: "MUA",
6@  flight: {
7    code: payload.toAirportCode,
8    airline: payload.airline
9  }
10 }
```

<?xml version='1.0' encoding='UTF-8'?>
<data>
 <hub>MUA</hub>
 <flight>
 <code>SF0</code>
 <airline>United</airline>
 </flight>
</data>

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

The screenshot shows the Mule Studio interface. On the left, the XML configuration is displayed:

```
1 @%dw 1.0
2 %output application/xml
3 ---
4 @data: {
5     hub: "MUA",
6     flight @!(airline: payload.airline): {
7         code: payload.toAirportCode
8     }
}
```

On the right, the XML preview shows the resulting output:

```
<?xml version='1.0' encoding='UTF-8'?>
<data>
    <hub>MUA</hub>
    <flight airline="United">
        <code>SF0</code>
    </flight>
</data>
```

## Debug the application

26. Debug the project.

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

28. In the Mule Debugger, step to the Logger and click the Variables tab; you should now see a flow variable.

The screenshot shows the Mule Debugger interface with the Variables tab selected. The left panel displays the flow variables:

Name	Value	Type
Message	org.mule.DefaultMule...	org.mule.api.processo...
Message Proce... Logger	org.mule.api.processo...	org.mule.api.processo...
Payload (mime... size = 1	{hub=MUA, code=CLE...	java.util.LinkedHashMap
@key	data	java.lang.String
value	{hub=MUA, code=CLE...	java.util.LinkedHashMap
0	hub=MUA	java.util.LinkedHashMap
1	code=CLE	java.util.LinkedHashMap
2	airline=Delta	java.util.LinkedHashMap
size = 1		

The right panel shows the XML message being processed:

Inbound	Variables	Outbound	Session	Record
xml (mimeTyp... <?xml version='1.0'... java.lang.String				
<?xml version='1.0' encoding='UTF-8'?>				
<data>				
<hub>MUA</hub>				
<flight airline="Delta">				
<code>CLE</code>				
</flight>				
</data>				

29. Stop the project.

## Walkthrough 10-3: Transform complex data structures with arrays

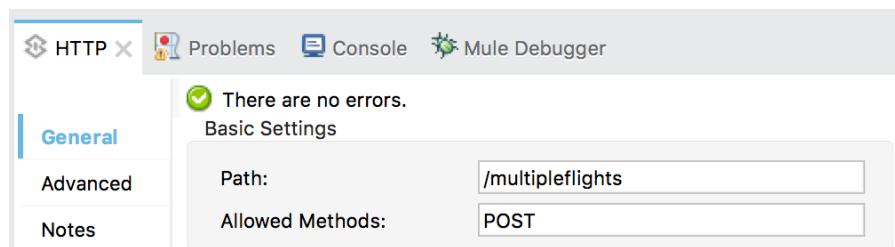
In this walkthrough, you work with JSON data for multiple flights posted to a flow. You will:

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

Name	Value
root : ArrayList	
[0] : LinkedHashMap	
flight : Integer	0
flightO : LinkedHashMap	
airline : String	United
flightCode : String	ER38sd
fromAirportCode : String	LAX
toAirportCode : String	SFO
departureDate : String	May 21, 2016
emptySeats : Integer	0
totalSeats : Integer	200
price : Integer	199
planeType : String	Boeing 737
[1] : LinkedHashMap	
flight : Integer	1

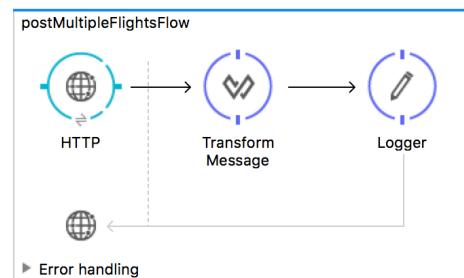
### Create a new flow

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



### Transform the payload to Java

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



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

## Add input metadata and sample data

- In the preview section, click the Create required sample data to execute preview link.
- In the Sample data types requirements dialog box, set the metadata type to JSON.
- Click the Actions button.
- In the Select metadata dialog box, select flights\_json and click Select.
- In the Sample data types requirements dialog box, click OK.
- Look at the preview section; the sample output should be an ArrayList of LinkedHashMaps.

The screenshot shows the Mule Studio interface with the 'Transform Message' tab selected. In the preview section, there is a JSON editor containing the following flight data:

```
{
  "airline": "United",
  "flightCode": "ER38sd",
  "fromAirportCode": "LAX",
  "toAirportCode": "SFO",
  "departureDate": "May 21, 2016",
  "emptySeats": 0,
  "totalSeats": 200,
  "price": 199,
  "planeType": "Boeing 737"
}, {
}
```

Below the JSON editor, there are tabs for 'Context' and 'payload'. To the right of the preview section is a 'Define metadata' panel and a 'Output' panel. The 'Output' panel shows the DataWeave code:

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

Finally, on the far right is a detailed 'Preview' table showing the structure of the ArrayList of LinkedHashMaps:

Name	Value
root : ArrayList	
[0] : LinkedHashMap	
airline : String	United
flightCode : String	ER38sd
fromAirportCode : String	LAX
toAirportCode : String	SFO
departureDate : String	May 21, 2016
emptySeats : Integer	0
totalSeats : Integer	200
price : Integer	199
planeType : String	Boeing 737
[1] : LinkedHashMap	
airline : String	Delta

## Return values for the first object in the collection

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

The screenshot shows the Mule Studio interface with the 'Transform Message' tab selected. In the preview section, the DataWeave code is:

```
1 %dw 1.0
2 %output application/java
3 ---
4 payload[0].*price
```

To the right is a 'Preview' table showing the result:

Name	Value
root : ArrayList	
[0] : Integer	199

## Return collections of values

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

```
payload.*price
```

Output	Payload	Preview
1 %dw 1.0 2 %output application/java 3 --- 4 payload.*price	Name   Value root : ArrayList   [0] : Integer   199 [1] : Integer   450	

19. Change the DataWeave expression to payload.price; you should get the same result.

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

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

Output	Payload	Preview
1 %dw 1.0 2 %output application/java 3 --- 4 [payload.price, payload.emptySeats] 5	Name   Value root : ArrayList   [0] : ArrayList   [0] : Integer   199 [1] : Integer   450 [1] : ArrayList   [0] : Integer   0 [1] : Integer   24	

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

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

Output	Payload	Preview
1 %dw 1.0 2 %output application/java 3 --- 4 payload map \$ 5	Name   Value root : ArrayList   [0] : LinkedHashMap   airline : String   Delta flightCode : String   ER0945 fromAirportCode : String   PDX toAirportCode : String   CLE departureDate : String   June 1, 2016 emptySeats : Integer   24 totalSeats : Integer   350 price : Integer   450 planeType : String   Boeing 747	

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

Output Payload ▾

Name	Value
root : ArrayList	
[0] : LinkedHashMap	
flight : Integer	0
[1] : LinkedHashMap	
flight : Integer	1

```

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

```

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

Output Payload ▾

Name	Value
root : ArrayList	
[0] : LinkedHashMap	
flight : Integer	0
destination : String	SFO
[1] : LinkedHashMap	
flight : Integer	1
destination : String	CLE

```

1@%dw 1.0
2 %output application/java
3 ---
4@payload map {
5   flight: $$,
6   destination: $.toAirportCode
7 }
8

```

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

Output Payload ▾

Name	Value
root : ArrayList	
[0] : LinkedHashMap	
flight : Integer	0
0 : LinkedHashMap	
airline : String	United
flightCode : String	ER38sd
fromAirportCode : String	LAX
toAirportCode : String	SFO
departureDate : String	May 21, 2016
emptySeats : Integer	0
totalSeats : Integer	200
price : Integer	199
planeType : String	Boeing 737
[1] : LinkedHashMap	
flight : Integer	1

```

1@%dw 1.0
2 %output application/java
3 ---
4@payload map {
5   flight: $$,
6   ($$): $
7 }
8

```

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

The screenshot shows the DataWeave preview interface. On the left, the DataWeave code is:

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

On the right, the preview pane shows the resulting structure:

Name	Value
root : ArrayList	
[0] : LinkedHashMap	
flight : Integer	0
flight0 : LinkedHashMap	
airline : String	United
flightCode : String	ER38sd
fromAirportCode : String	LAX
toAirportCode : String	SFO
departureDate : String	May 21, 2016
emptySeats : Integer	0
totalSeats : Integer	200
price : Integer	199
planeType : String	Boeing 737
[1] : LinkedHashMap	
flight : Integer	1

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

## Change the expression to output XML

26. Change the DataWeave expression output type from application/java to application/xml; you should get an issue displayed.

27. Change the DataWeave expression to create a root node called flights; you should still get an issue.

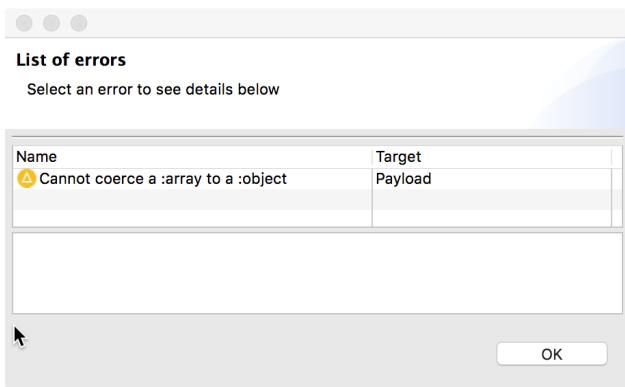
The screenshot shows the DataWeave preview interface. On the left, the DataWeave code is identical to the previous example, but the output type is set to application/xml:

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

An orange warning icon indicates "1 issue found". On the right, the preview pane shows the resulting structure:

Name	Value
root : ArrayList	
[0] : LinkedHashMap	
flight : Integer	0
flight0 : LinkedHashMap	
airline : String	United
flightCode : String	ER38sd

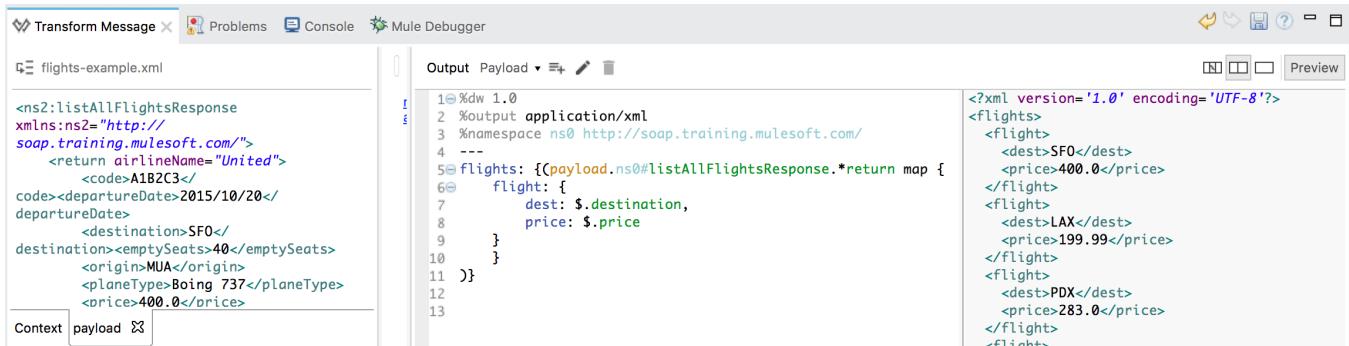
28. Look at the error message.



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

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

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



```
flights-example.xml
<ns2:listAllFlightsResponse
    xmlns:ns2="http://soap.training.mulesoft.com/">
    <return>
        <airlineName>United</airlineName>
        <code>A1B2C3</code>
        <departureDate>2015/10/20</departureDate>
        <destination>SFO</destination>
        <emptySeats>40</emptySeats>
        <origin>MUA</origin>
        <planeType>Boeing 737</planeType>
        <price>400.0</price>
    </return>
</ns2:listAllFlightsResponse>
```

```
Output Payload
1 %dw 1.0
2 %output application/xml
3 %namespace ns0 http://soap.training.mulesoft.com/
4 ---
5 flights: {$(payload.ns0:listAllFlightsResponse.*return map {
6     flight: {
7         dest: $.destination,
8         price: $.price
9     }
10 })
11 }
12
13
```

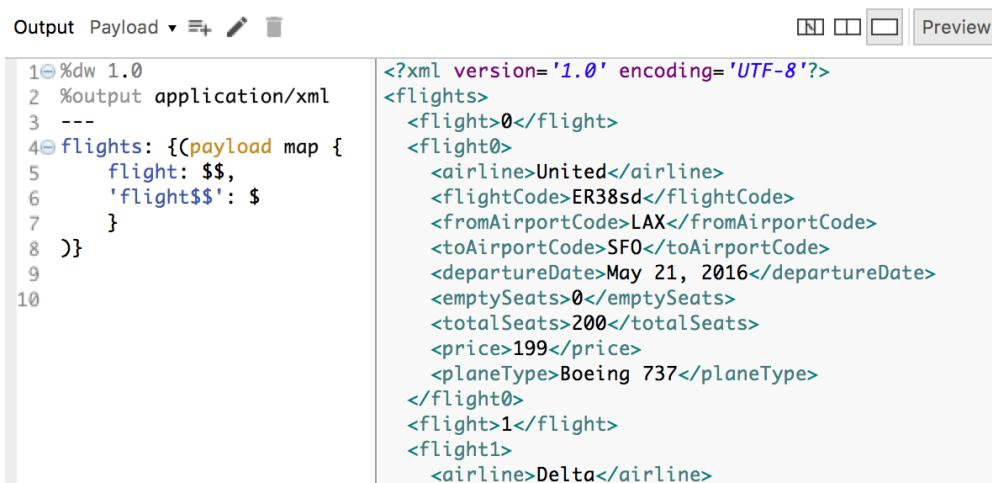
```
<?xml version='1.0' encoding='UTF-8'?>
<flights>
    <flight>
        <dest>SFO</dest>
        <price>400.0</price>
    </flight>
    <flight>
        <dest>LAX</dest>
        <price>199.99</price>
    </flight>
    <flight>
        <dest>PDX</dest>
        <price>283.0</price>
    </flight>
</flights>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

### Transform the JSON array of objects to XML

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

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

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



```
Output Payload
1 %dw 1.0
2 %output application/xml
3 ---
4 flights: {$(payload map {
5     flight: $$,
6     'flight$$': $
7 })
8 }
9
10
```

```
<?xml version='1.0' encoding='UTF-8'?>
<flights>
    <flight>
        <flight0>
            <airline>United</airline>
            <flightCode>ER38sd</flightCode>
            <fromAirportCode>LAX</fromAirportCode>
            <toAirportCode>SFO</toAirportCode>
            <departureDate>May 21, 2016</departureDate>
            <emptySeats>0</emptySeats>
            <totalSeats>200</totalSeats>
            <price>199</price>
            <planeType>Boeing 737</planeType>
        </flight0>
        <flight1>
            <airline>Delta</airline>
```

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

Output Payload Preview

```

1@ %dw 1.0
2 %output application/xml
3 ---
4@ flights: ${payload map {
5   flight: $}
6 }
7 }
8
9

```

<?xml version='1.0' encoding='UTF-8'?>
<flights>
 <flight>
 <airline>United</airline>
 <flightCode>ER38sd</flightCode>
 <fromAirportCode>LAX</fromAirportCode>
 <toAirportCode>SFO</toAirportCode>
 <departureDate>May 21, 2016</departureDate>
 <emptySeats>0</emptySeats>
 <totalSeats>200</totalSeats>
 <price>199</price>
 <planeType>Boeing 737</planeType>
 </flight>
 <flight>
 <airline>Delta</airline>
 </flight>

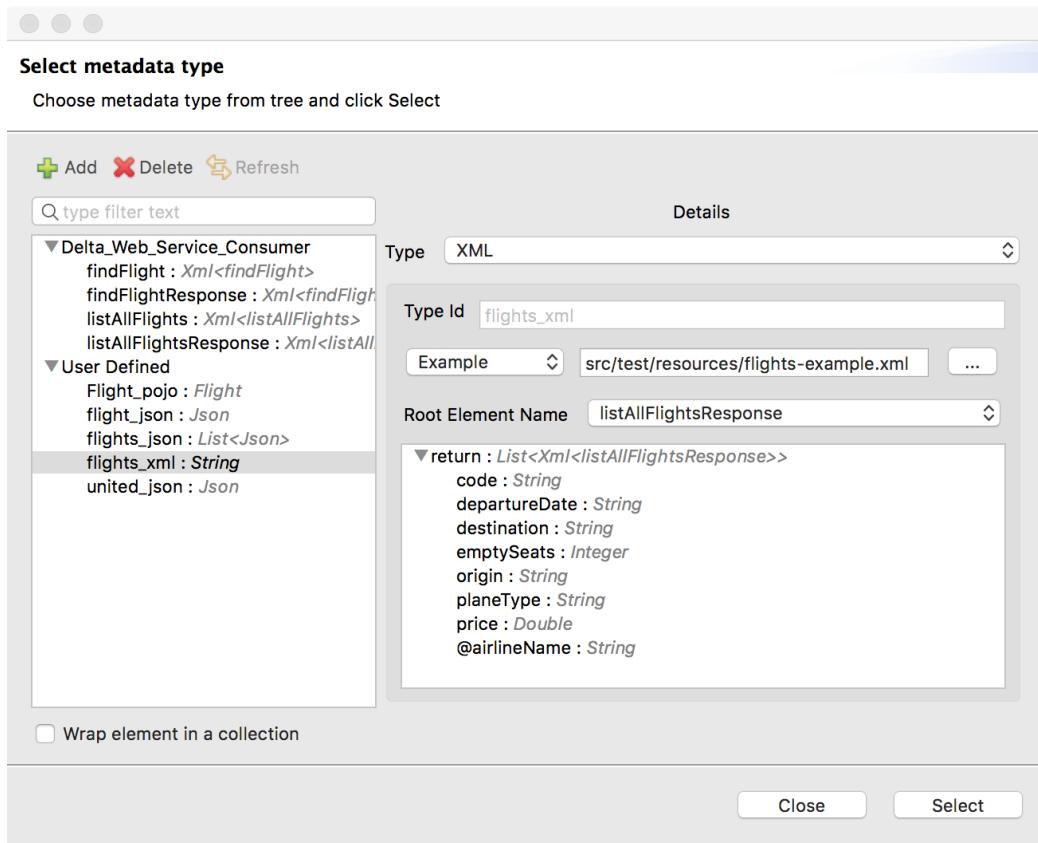
## Change input metadata and sample data to XML

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



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

16. Click Select.



17. In the Sample data types requirements dialog box, click OK.

18. Look at the XML sample data in the input section.

The screenshot shows the Mule Studio interface with the 'Transform Message' tab selected. The top bar includes tabs for 'Problems', 'Console', and 'Mule Debugger'. Below the tabs, the file 'flights-example.xml' is open. The XML content is as follows:

```
<ns2:listAllFlightsResponse xmlns:ns2="http://soap.training.mulesoft.com/">
    <return airlineName="United">
        <code>A1B2C3</code><departureDate>2015/10/20</departureDate>
        <destination>SF0</destination><emptySeats>40</emptySeats>
        <origin>MUA</origin>
        <planeType>Boing 737</planeType>
        <prices>400.0</prices>
    </return>
    <return airlineName="Delta">
        <code>A1B2C4</code>
        <departureDate>2015/10/21</departureDate><destination>LAX</destination><emptySeats>10</emptySeats>
    </return>

```

At the bottom, there are buttons for 'Context' and 'payload'.

## Examine the sample output for the transformation

19. Review the transformation expression.

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

Output Payload ▾  

Preview

```
1 %dw 1.0
2 %output application/xml
3 ---
4 flights: {payload map {
5   flight: $}
6 }
7 )
8
9
```

```
<?xml version='1.0' encoding='UTF-8'?>
<flights>
  <flight>
    <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>
```

## Change the return structure of the XML

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

Output Payload ▾  

Preview

```
1 %dw 1.0
2 %output application/xml
3 %namespace ns0 http://soap.training.mulesoft.com/
4 ---
5 flights: {payload.ns0#listAllFlightsResponse map {
6   flight: $}
7 }
8 )
9
10
```

```
<?xml version='1.0' encoding='UTF-8'?>
<flights>
  <flight>
    <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>
  </flight>
  <flight>
    <code>A1B2C4</code>
    <departureDate>2015/10/21</departureDate>
```

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

Output Payload ▾  

Preview

```
1 %dw 1.0
2 %output application/xml
3 %namespace ns0 http://soap.training.mulesoft.com/
4 ---
5 flights: {payload.ns0#listAllFlightsResponse.return map {
6   flight: $}
7 }
8 )
9
10
```

```
<?xml version='1.0' encoding='UTF-8'?>
<flights>
  <flight>A1B2C3</flight>
  <flight>2015/10/20</flight>
  <flight>SFO</flight>
  <flight>40</flight>
  <flight>MUA</flight>
  <flight>Boing 737</flight>
  <flight>400.0</flight>
</flights>
```

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

The screenshot shows the Mule Studio DataWeave editor. On the left, the DataWeave code is displayed:

```
1 %dw 1.0
2 %output application/xml
3 %namespace ns0 http://soap.training.mulesoft.com/
4 ---
5 flights: {${payload.ns0:listAllFlightsResponse.*}return map {
6   flight: $
7 }
8 }
9
10
```

On the right, the preview pane shows the resulting XML output:

```
<?xml version='1.0' encoding='UTF-8'?>
<flights>
  <flight>
    <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>
  </flight>
  <flight>
    <code>A1B2C4</code>
```

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

The screenshot shows the Mule Studio DataWeave editor. On the left, the DataWeave code is displayed:

```
1 %dw 1.0
2 %output application/xml
3 %namespace ns0 http://soap.training.mulesoft.com/
4 ---
5 flights: {${payload.ns0:listAllFlightsResponse.*}return map {
6   flight: {
7     dest: $.destination,
8     price: $.price
9   }
10 }
11 }
```

On the right, the preview pane shows the resulting XML 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>
```

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

## Change the transformation to return Java

25. Change the output type from application/xml to application/java.

26. Look at the preview; you should see flights is a LinkedHashMap with one flight property.

The screenshot shows the Mule Studio DataWeave editor. On the left, the DataWeave code is displayed:

```
1 %dw 1.0
2 %output application/java
3 %namespace ns0 http://soap.training.mulesoft.com/
4 ---
5 flights: {${payload.ns0:listAllFlightsResponse.*}return map {
6   flight: {
7     dest: $.destination,
8     price: $.price
9   }
10 }
11 }
```

On the right, the preview pane shows the resulting Java object structure:

```
Name
  root : LinkedHashMap
    flights : LinkedHashMap
      flight : LinkedHashMap
        dest : String
        price : String
```

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

Output Payload

Preview

Name	Value
root : LinkedHashMap	
flights : ArrayList	
[0] : LinkedHashMap	SFO 400.0
[1] : LinkedHashMap	LAX

```

1@%dw 1.0
2 %output application/java
3 %namespace ns0 http://soap.training.mulesoft.com/
4 ---
5@ flights: payload.ns0:listAllFlightsResponse.*return map {
6     dest: $.destination,
7     price: $.price|
8 }
```

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

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

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

The screenshot shows the Anypoint Studio interface. On the left, there is a code editor window titled "Output Payload" containing DataWeave code. The code defines a variable "flights" as a map where each item has a destination, price, plane type, and departure date. It uses various DataWeave functions like `map`, `as`, `upper`, and `format` to structure the data. On the right, there is a data grid titled "Name | Value" showing the resulting data structure. The root is a LinkedHashMap named "root". It contains an "flights" ArrayList with two items. Item [0] is a LinkedHashMap with keys "dest", "price", "plane", and "date", and values "SFO", "400.00", "BOING 737", and "Oct 20, 2015" respectively. Item [1] is another LinkedHashMap with the same structure and values for "LAX", "199.99", "BOING 737", and "Oct 21, 2015".

Name	Value
root : LinkedHashMap	
flights : ArrayList	
[0] : LinkedHashMap	<ul style="list-style-type: none"><li>dest : String SFO</li><li>price : String 400.00</li><li>plane : String BOING 737</li><li>date : String Oct 20, 2015</li></ul>
[1] : LinkedHashMap	<ul style="list-style-type: none"><li>dest : String LAX</li><li>price : String 199.99</li><li>plane : String BOING 737</li><li>date : String Oct 21, 2015</li></ul>

## Browse DataWeave reference documentation

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

The screenshot shows the "Welcome to Anypoint Studio 6.0.0!" dialog. It highlights the "Unified! Single Runtime, APIKit Bundled within Studio and Updated New Project Wizard" section. It explains that Mule 3.8 has become a unified single runtime for both integration and APIs, and that APIKit is now bundled within the studio. It also mentions the updated New Project Wizard. Below this, there is a screenshot of the "New Mule Project" dialog and a "Browse API Manager for APIs" window. The "New Mule Project" dialog shows a "Project Name" field set to "API Project". The "Browse API Manager for APIs" window shows a list of APIs, with "Mulesoft" selected. At the bottom, there is a link to "Read the [detailed release notes](#)".

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

Search the docs

Nav Edit on GitHub ★★★★★

Mule User Guide Version 3.8 (Latest) ▾

Map

In this topic:

- Map
- Map Object
- Pluck
- Filter
- Remove

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

Search the docs

Nav Edit on GitHub ★★★★★

Mule User Guide Version 3.8 (Latest) ▾

Coerce to date

`(:string', 'type')('(:number', 'type') => 'date'`

Date types can be coerced from string or number.

Any format pattern accepted by [DateTimeFormatter](#) is allowed.

Transform

```
1 %dw 1.0
2 %output application/json
3 ---
4 {
5   a: 1436287232 as :datetime,
6   b: "2015-10-07 16:40:32.000" as :localdatetime {format: "yyyy-MM-dd HH:mm:s"
7 }
```

DATAWEAVE

Output

```
1 {
2   "a": "2015-07-07T16:40:32Z",
3   "b": "2015-10-07 16:40:32.000"
4 }
```

JSON

In this topic:

- Remove by Matching Key and Value
- AND
- OR
- IS
- Concat
- Contains
- AS (Type Coercion)
- Type Of
- Flatten
- Size Of
- Array Push

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

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

The screenshot shows a web browser displaying the Mule User Guide at <https://docs.mulesoft.com/mule-user-guide/v/3.8/dataweave-operators#as-type-coercion>. The page title is "AS (Type Coercion)". A sidebar on the right lists various DataWeave operators: Contains, AS (Type Coercion), Type Of, Flatten, Size Of, Array Push, Remove from Array, Remove Matching from Array, and Average of Array. The main content area contains two callout boxes: one about DataWeave's default conversion behavior and another about checking the type coercion table.

AS (Type Coercion)

Coerce the given value to the specified type.

DataWeave by default attempts to convert the type of a value before failing, so using this operator to convert is sometimes not required but still recommended.

Check the [type coercion table](#) to see what conversions between what types are allowed in DataWeave.

10. Browse the type coercion table.

The screenshot shows a web browser displaying the Mule User Guide at <https://docs.mulesoft.com/mule-user-guide/v/3.8/dataweave-types#type-coercion-table>. The page title is "Type Coercion Table". A sidebar on the right lists topics: Array, Object, String, Number, Boolean, and Dates. The main content area contains a callout box explaining the automatic coercion behavior and a table showing source-target pairs and their properties.

Type Coercion Table

In DataWeave, types can be coerced from one type to other using the [AS Operator](#). This table shows the possible combinations and the properties from the schema that are used in the transformation.

When you provide an [operator](#) with properties that don't match the expected types, DataWeave automatically attempts to coerce the provided property to the required type.

Source	Target	Property
:object	:array	(1)
:range	:array	
:number	:binary	

## Format a string

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

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

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

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

Name	Value
root : LinkedHashMap	
flights : ArrayList	
[0] : LinkedHashMap	
dest : String	SFO
price : String	400.0
plane : String	BOING 737
[1] : LinkedHashMap	
dest : String	LAX

## Coerce a string to a number

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

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

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

Name	Value
root : LinkedHashMap	
flights : ArrayList	
[0] : LinkedHashMap	
dest : String	SFO
price : Integer	400
plane : String	BOING 737
[1] : LinkedHashMap	
dest : String	LAX
price : Double	199.99
plane : String	BOING 737

## Coerce a string to a specific type of number object

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

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

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

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

```
1 %dw 1.0
2 %output application/java
3 %namespace ns0 http://soap.training.mulesoft.com/
4 ---
5 flights: payload.ns0:listAllFlightsResponse.*return map {
6     dest: $.destination,
7     price: $.price as :number {class:"java.lang.Double"}, // Coercion rule
8     plane: upper $.planeType
9 }
10
11
12
```

On the right is a preview table showing the resulting data structure:

Name	Value
root	LinkedHashMap
flights	ArrayList
[0]	LinkedHashMap
dest	SFO
price	400.0
plane	BOING 737
[1]	LinkedHashMap
dest	LAX
price	199.99

## Format a number

22. Use the format schema property in the DataWeave expression to format the prices to zero decimal places.

23. Remove the coercion to a java.lang.Double.

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

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

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

```
1 %dw 1.0
2 %output application/java
3 %namespace ns0 http://soap.training.mulesoft.com/
4 ---
5 flights: payload.ns0:listAllFlightsResponse.*return map {
6     dest: $.destination,
7     price: $.price as :number as :string {format: "###"}, // Format schema
8     plane: upper $.planeType
9 }
10
11
```

On the right is a preview table showing the resulting data structure:

Name	Value
root	LinkedHashMap
flights	ArrayList
[0]	LinkedHashMap
dest	SFO
price	400
plane	BOING 737
[1]	LinkedHashMap
dest	LAX
price	200

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

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

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

Output Payload ▾  

Name	Value
root : LinkedHashMap	
flights : ArrayList	
[0] : LinkedHashMap	
dest : String	SFO
price : String	400
plane : String	BOING 737
[1] : LinkedHashMap	
dest : String	LAX
price : String	199.99

```
1@ %dw 1.0
2 %output application/java
3 %namespace ns0 http://soap.training.mulesoft.com/
4 ---
5@ flights: payload.ns0#listAllFlightsResponse.*return map {
6   dest: $.destination,
7   price: $.price as :number as :string {format: "###.##"},
8   plane: upper $.planeType
9 }
10
11
```

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

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

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

Output Payload ▾  

Name	Value
root : LinkedHashMap	
flights : ArrayList	
[0] : LinkedHashMap	
dest : String	SFO
price : String	400.00
plane : String	BOING 737
[1] : LinkedHashMap	
dest : String	LAX
price : String	199.99

```
1@ %dw 1.0
2 %output application/java
3 %namespace ns0 http://soap.training.mulesoft.com/
4 ---
5@ flights: payload.ns0#listAllFlightsResponse.*return map {
6   dest: $.destination,
7   price: $.price as :number as :string {format: "##.00"},
8   plane: upper $.planeType
9 }
10
11
12
```

*Note: If you are not a Java programmer, you may want to look at the Java documentation for the DecimalFormat class to review documentation on patterns.*

<https://docs.oracle.com/javase/8/docs/api/java/text/DecimalFormat.html>

## Coerce a string to a date

29. Add a date field to the return object.

```
date: $.departureDate
```

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

The screenshot shows the Mule Studio interface with the DataWeave editor and preview pane. The DataWeave code is:

```
1@ %dw 1.0
2 %output application/java
3 %namespace ns0 http://soap.training.mulesoft.com/
4 ---
5@ flights: payload.ns0:listAllFlightsResponse.*return map {
6   dest: $.destination,
7   price: $.price as :number as :string {format: "###.##"}, 
8   plane: upper $.planeType,
9   date: $.departureDate
10 }
```

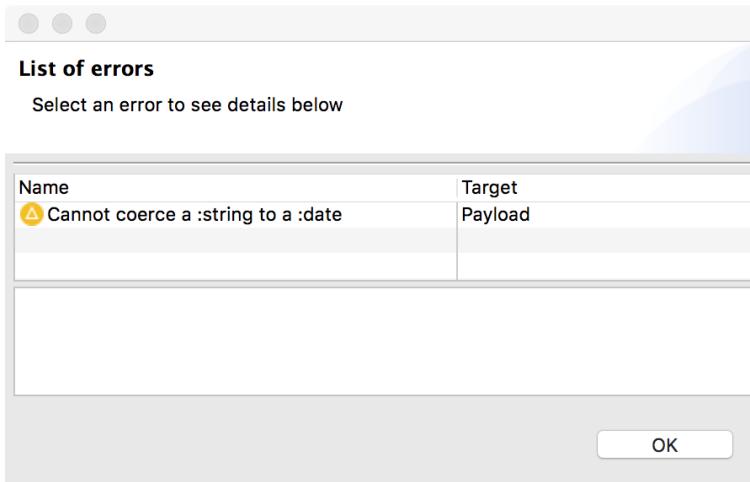
The preview pane shows the resulting data structure:

Name	Value
root	LinkedHashMap
flights	ArrayList
[0]	LinkedHashMap
dest	SFO
price	400.00
plane	BOING 737
date	2015/10/20
[1]	LinkedHashMap

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

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

32. Look at the exception.



33. Look at the format of the dates in the preview.

34. Change the DataWeave expression to use the format schema property to specify the pattern of the input date strings.

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

*Note: If you are not a Java programmer, you may want to look at the Java documentation for the DateFormat class to review documentation on pattern letters.*

<https://docs.oracle.com/javase/8/docs/api/java/text/format/DateFormat.html>

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

Output Payload ▾  

Name	Value
root : LinkedHashMap	
flights : ArrayList	
[0] : LinkedHashMap	
dest : String	SFO
price : String	400.00
plane : String	BOING 737
date : Date	Tue Oct 20 00:00:00 PDT 2015
[1] : LinkedHashMap	

```
1@ %dw 1.0
2 %output application/java
3 %namespace ns0 http://soap.training.mulesoft.com/
4 ---
5@ flights: payload.ns0#listAllFlightsResponse.*return map {
6   dest: $.destination,
7   price: $.price as :number as :string {format: "###.00"}, 
8   plane: upper $.planeType,
9   date: $.departureDate as :date {format: "yyyy/MM/dd"} 
10 }
```

## Format a date

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

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

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

Output Payload ▾  

Name	Value
root : LinkedHashMap	
flights : ArrayList	
[0] : LinkedHashMap	
dest : String	SFO
price : String	400.00
plane : String	BOING 737
date : String	2015-10-20

```
1@ %dw 1.0
2 %output application/java
3 %namespace ns0 http://soap.training.mulesoft.com/
4 ---
5@ flights: payload.ns0#listAllFlightsResponse.*return map {
6   dest: $.destination,
7   price: $.price as :number as :string {format: "###.00"}, 
8   plane: upper $.planeType,
9   date: $.departureDate as :date {format: "yyyy/MM/dd"} as :string
10 }
```

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

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

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

Output Payload ▾  

Name	Value
root : LinkedHashMap	
flights : ArrayList	
[0] : LinkedHashMap	
dest : String	SFO
price : String	400.00
plane : String	BOING 737
date : String	Oct 20, 2015
[1] : LinkedHashMap	
dest : String	LAX
price : String	199.99
plane : String	BOING 737
date : String	Oct 21, 2015

```
1@ %dw 1.0
2 %output application/java
3 %namespace ns0 http://soap.training.mulesoft.com/
4 ---
5@ flights: payload.ns0#listAllFlightsResponse.*return map {
6   dest: $.destination,
7   price: $.price as :number as :string {format: "###.00"}, 
8   plane: upper $.planeType,
9@   date: $.departureDate as :date {format: "yyyy/MM/dd"} 
10  as :string {format: "MMM dd, yyyy"} 
11 }
12
13
14
```

## Walkthrough 10-6: Use DataWeave operators

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

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

The screenshot shows the Mule Studio interface with the DataWeave editor open. The payload is a list of flights. The DataWeave code is as follows:

```
1 %dw 1.0
2 %output application/java
3 %namespace ns0 http://soap.training.mulesoft.com/
4 ---
5 flights: payload.ns0:listAllFlightsResponse.*return map {
6     dest: $.destination,
7     price: $.price as :number as :string {format: "###.00"},
8     plane: upper ($.planeType replace /(Boing)/ with "Boeing"),
9     date: $.departureDate as :date {format: "yyyy/MM/dd"}
10    as :string {format: "MMM dd, yyyy"},
11    seats: $.emptySeats as :number
12 } orderBy $.date orderBy $.price distinctBy $ filter ($.seats !=0)
13
14
```

The preview section shows the resulting flight data:

Name	Value
root	LinkedHashMap
flights	ArrayList
[0]	LinkedHashMap
[1]	LinkedHashMap
[2]	LinkedHashMap
[3]	LinkedHashMap
dest	SFO
price	400.00
plane	BOEING 737
date	Oct 20, 2015
seats	40

### Replace data values

- Return to the Transform Message properties view for the transformation in postMultipleFlightsFlow.
- Use the replace operator in the DataWeave expression to replace the string Boing with Boeing.  
plane: upper \$.planeType replace /(BOING)/ with "BOEING",
- Look at the preview section; Boeing should not be spelled correctly.
- In the DataWeave expression, place parentheses around the replace operator expression.
- Look at the preview section; Boeing should now be spelled correctly.

The screenshot shows the Mule Studio interface with the DataWeave editor open. The payload is a list of flights. The DataWeave code is as follows:

```
1 %dw 1.0
2 %output application/java
3 %namespace ns0 http://soap.training.mulesoft.com/
4 ---
5 flights: payload.ns0:listAllFlightsResponse.*return map {
6     dest: $.destination,
7     price: $.price as :number as :string {format: "###.00"},
8     plane: upper ($.planeType replace /(Boing)/ with "Boeing"),
9     date: $.departureDate as :date {format: "yyyy/MM/dd"}
10    as :string {format: "MMM dd, yyyy"}
11 }
```

The preview section shows the resulting flight data, with the plane type now correctly spelled "Boeing":

Name	Value
root	LinkedHashMap
flights	ArrayList
[0]	LinkedHashMap
dest	SFO
price	400.00
plane	BOEING 737
date	Oct 20, 2015
[1]	LinkedHashMap
dest	LAX

### Order data

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

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

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

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

The screenshot shows the MuleSoft Anypoint Studio interface with the DataWeave editor open. The code is as follows:

```
1@ %dw 1.0
2 %output application/java
3 %namespace ns0 http://soap.training.mulesoft.com/
4 ---
5@ flights: payload.ns0:listAllFlightsResponse.*return map {
6     dest: $.destination,
7     price: $.price as :number as :string {format: "###.00"}, 
8     plane: upper ($.planeType replace /(Boing)/ with "Boeing"),
9@     date: $.departureDate as :date {format: "yyyy/MM/dd"}
10    as :string {format: "MMM dd, yyyy"}
11 } orderBy $.price
12
13
```

To the right is a preview table showing the resulting flight data:

Name	Value
root	LinkedHashMap
flights	ArrayList
[0]	LinkedHashMap
dest	LAX
price	199.99
plane	BOEING 737
date	Oct 21, 2015
[1]	LinkedHashMap
dest	PDX
price	283.00

9. Look at the PDX flights; they should not be ordered by date.

10. Change the DataWeave expression to first sort by date and then by price.

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

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

The screenshot shows the MuleSoft Anypoint Studio interface with the DataWeave editor open. The code is identical to the previous one:

```
1@ %dw 1.0
2 %output application/java
3 %namespace ns0 http://soap.training.mulesoft.com/
4 ---
5@ flights: payload.ns0:listAllFlightsResponse.*return map {
6     dest: $.destination,
7     price: $.price as :number as :string {format: "###.00"}, 
8     plane: upper ($.planeType replace /(Boing)/ with "Boeing"),
9@     date: $.departureDate as :date {format: "yyyy/MM/dd"}
10    as :string {format: "MMM dd, yyyy"}
11 } orderBy $.date orderBy $.price
12
13
```

To the right is a preview table showing the resulting flight data:

Name	Value
date	Oct 20, 2015
[2]	LinkedHashMap
dest	PDX
price	283.00
plane	BOEING 777
date	Oct 20, 2015
[3]	LinkedHashMap
dest	PDX
price	283.00
plane	BOEING 777
date	Oct 21, 2015

## Remove duplicate data

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

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

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

The screenshot shows the MuleSoft Anypoint Studio interface. On the left, there is a code editor window titled "Output Payload" containing a DataWeave script. The script defines a variable "flights" which is a map of flight details. It includes fields like destination, price, plane type, date, and a formatted date string. The script then orders the flights by date and price, and uses the "distinctBy" operator to remove any duplicates. Lines 1 through 14 are shown in the code editor.

On the right, there is a "Preview" window showing a table with four flight records. Each record is a LinkedHashMap with keys "dest", "price", "plane", and "date". The first flight is to PDX on Oct 20, 2015, with a Boeing 777 and a price of 283.00. The second flight is to PDX on Oct 21, 2015, with a Boeing 777 and a price of 283.00. The third flight is to BOEING 777 on Oct 20, 2015, with a price of 283.00. The fourth flight is to BOEING 777 on Oct 21, 2015, with a price of 283.00.

Name	Value
root	LinkedHashMap
flights	ArrayList
[0]	LinkedHashMap
[1]	LinkedHashMap
[2]	LinkedHashMap
[3]	LinkedHashMap

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

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

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

The screenshot shows the MuleSoft Anypoint Studio interface. The code editor window now includes a new "seats" field in the "flights" map, which is coerced from the "emptySeats" field to a number. The preview window shows the same four flights as before, but now each flight has a "seats" field added to the map, with a value of 23.

Name	Value
root	LinkedHashMap
flights	ArrayList
[0]	LinkedHashMap
[1]	LinkedHashMap
[2]	LinkedHashMap
[3]	LinkedHashMap
[4]	LinkedHashMap

## Filter data

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

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

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

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

The screenshot shows the MuleSoft Anypoint Studio interface. On the left, there is a code editor window titled "Output Payload" containing the following DataWeave code:

```
1@ %dw 1.0  
2 %output application/java  
3 %namespace ns0 http://soap.training.mulesoft.com/  
4 ---  
5@ flights: payload.ns0:listAllFlightsResponse.*return map {  
6     dest: $.destination,  
7     price: $.price as :number as :string {format: "###.00"},  
8     plane: upper ($.planeType replace /(Boing)/ with "Boeing"),  
9     date: $.departureDate as :date {format: "yyyy/MM/dd"}  
10    as :string {format: "MMM dd, yyyy"},  
11    seats: $.emptySeats as :number  
12 } orderBy $.date orderBy $.price distinctBy $ filter ($.seats !=0)  
13  
14
```

On the right, there is a preview table with the following data:

Name	Value
root	LinkedHashMap
flights	ArrayList
[0]	LinkedHashMap
[1]	LinkedHashMap
[2]	LinkedHashMap
[3]	LinkedHashMap
dest	SFO
price	400.00
plane	BOEING 737
date	Oct 20, 2015
seats	40

## Walkthrough 10-7: Define and use custom data types

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

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

The screenshot shows the Mule Studio interface with the DataWeave preview window open. On the left, the DataWeave script is displayed:

```
1 %dw 1.0
2 %output application/java
3 %namespace ns0 http://soap.training.mulesoft.com/
4 %type currency = :string {format: "###.00"}
5 %type flight = :object {class: "com.mulesoft.training.Flight"}
6 ---
7 @ flights: payload.ns0#listAllFlightsResponse.*return map {
8     destination: $.destination,
9     price: $.price as :number as :currency,
10    planeType: upper ($.planeType replace /(Boing)/ with "Boeing"),
11    departureDate: $.departureDate as :date {format: "yyyy/MM/dd"}
12    as :string {format: "MMM dd, yyyy"},
13    availableSeats: $.emptySeats as :number
14 } as :flight
15
16
```

On the right, the resulting JSON structure is shown in a tree view:

Name	Value
root	LinkedHashMap
flights	ArrayList
[0]	Flight
airlineName	String
availableSeats	Integer 40
departureDate	Date Oct 20, 2015
destination	String SFO
flightCode	String
origination	String
planeType	String BOEING 737
price	Double 400.0
[1]	Flight
[2]	Flight
[3]	Flight
[4]	Flight

### Define a custom data type

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

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

```
1 %dw 1.0
2 %output application/java
3 %namespace ns0 http://soap.training.mulesoft.com/
4 %type currency = :string {format: "###.00"}
5 ---
```

### Use a custom data type

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

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

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

The screenshot shows the Mule Studio interface with a DataWeave script in the left pane and a preview table in the right pane.

```

1 %dw 1.0
2 %output application/java
3 %namespace ns0 http://soap.training.mulesoft.com/
4 %type currency = :string {format: "###.00"}
5 ---
6 flights: payload.ns0:listAllFlightsResponse.*return map {
7     dest: $.destination,
8     price: $.price as :number as :currency,
9     plane: upper ($.planeType replace /(Boing)/ with "Boeing"),
10    date: $.departureDate as :date {format: "yyyy/MM/dd"}
11    as :string {format: "MMM dd, yyyy"}, 
12    seats: $.emptySeats as :number
13 } orderBy $.date orderBy $.price distinctBy $ filter ($.seats !=0)
  
```

Name	Value
root	LinkedHashMap
flights	ArrayList
[0]	LinkedHashMap
dest	LAX
price	199.99
plane	BOEING 737
date	Oct 21, 2015
seats	10
[1]	LinkedHashMap
dest	PDX
price	283.00

## Transform objects to POJOs

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

```

1 package com.mulesoft.training;
2
3 import java.util.Comparator;
4
5 public class Flight implements java.io.Serializable,
6
7     String flightCode;
8     String origination;
9     int availableSeats;
10    String departureDate;
11    String airlineName;
12    String destination;
13    double price;
14    String planeType;
15
16    public Flight() {
17
18    }
  
```

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

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

```

1 %dw 1.0
2 %output application/java
3 %namespace ns0 http://soap.training.mulesoft.com/
4 %type currency = :string {format: "###.00"}
5 %type flight = :object {class: "com.mulesoft.training.Flight"}
6 ---
```

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

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

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

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

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

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

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

```
payload.listAllFlightsResponse.*return map {  
    ...  
} orderBy $.departureDate orderBy $.price distinctBy $ filter  
($.availableSeats !=0) as :flight  
  
1@%dw 1.0  
2 %output application/java  
3 %namespace ns0 http://soap.training.mulesoft.com/  
4 %type currency = :string {format: "###.00"}  
5 %type flight = :object {class: "com.mulesoft.training.Flight"}  
6 ---  
7@ flights: payload.ns0:listAllFlightsResponse.*return map {  
8     destination: $.destination,  
9     price: $.price as :number as :currency,  
10    planeType: upper ($.planeType replace /(Boing)/ with "Boeing"),  
11@    departureDate: $.departureDate as :date {format: "yyyy/MM/dd"}  
12    as :string {format: "MMM dd, yyyy"},  
13    availableSeats: $.emptySeats as :number  
14@ } orderBy $.departureDate orderBy $.price distinctBy $  
15 filter ($.availableSeats !=0) as :flight
```

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

16. If you get an exception instead; delete the operators after the map.

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

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

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

Output Payload ▾  

```
1@ %dw 1.0
2 %output application/java
3 %namespace ns0 http://soap.training.mulesoft.com/
4 %type currency = :string {format: "###.##"}
5 %type flight = :object {class: "com.mulesoft.training.Flight"}
6 ---
7@ flights: payload.ns0:listAllFlightsResponse.*return map {
8     destination: $.destination,
9     price: $.price as :number as :currency,
10    planeType: upper ($.planeType replace /(Boing)/ with "Boeing"),
11@    departureDate: $.departureDate as :date {format: "yyyy/MM/dd"},
12    as :string {format: "MMM dd, yyyy"}, 
13    availableSeats: $.emptySeats as :number
14 } as :flight
15
16
```

Name	Value
root	LinkedHashMap
flights	ArrayList
[0]	Flight
airlineName	String
availableSeats	Integer 40
departureDate	Date Oct 20, 2015
destination	String SFO
flightCode	String
origination	String
planeType	String BOEING 737
price	Double 400.0
[1]	Flight
[2]	Flight
[3]	Flight
[4]	Flight

## Walkthrough 10-8: Call MEL functions and other flows

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

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

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

The screenshot shows the Anypoint Studio interface. On the left, the 'global' configuration file is open, showing XML code for defining a global function named 'getNumSeats'. On the right, the 'Expression' editor is open, displaying the MEL expression used to call this function. The 'Source' tab shows the MEL code, and the 'Expression' tab shows the expanded Java code generated by the expression language.

```
<ws:consumer-config name="Delta_Web_Service_Consumer" service="TicketServiceService">
<configuration defaultExceptionStrategy-ref="implementationChoice_Exception_Strategy">
    <http:config useTransportForUris="false"/>
    <expression-language></expression-language>
</configuration>
<payload-type-filter expectedType="java.util.ArrayList" name="Filter_Not_ArrayList">
```

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

### Define a global MEL function in global.xml

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

```
<ws:consumer-config name="Delta_Web_Service_Consumer" service="TicketServiceService">
<configuration defaultExceptionStrategy-ref="implementationChoice_Exception_Strategy">
    <http:config useTransportForUris="false"/>
    <expression-language></expression-language>
</configuration>
<payload-type-filter expectedType="java.util.ArrayList" name="Filter_Not_ArrayList">
```

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

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

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

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

```
def getNumSeats(type){
}

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

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

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

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

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

10. Save the file.

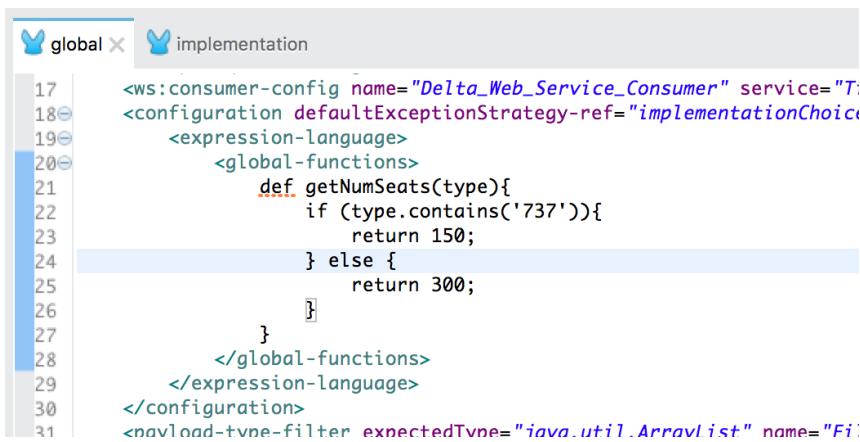
11. Run the project; the application should fail to deploy.

12. Locate the exception in the console.

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

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

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



```
17 <ws:consumer-config name="Delta_Web_Service_Consumer" service="Ti
18 <configuration defaultExceptionStrategy-ref="implementationChoice
19 <expression-language>
20 <global-functions>
21     def getNumSeats(type){
22         if (type.contains('737')){
23             return 150;
24         } else {
25             return 300;
26         }
27     }
28 </global-functions>
29 </expression-language>
30 </configuration>
31 <onload-type-filter expectedType="java.util.ArrayList" name="Fil
```

14. Run the project; the application should deploy.

15. Stop the project.

## Call a global MEL function from DataWeave

16. Return to postMultipleFlightsFlow in implementation.xml.

17. In the Transform Message properties view, add a field called totalSeats inside the transformation function.

18. Set totalSeats equal to the return value from the getNumSeats() function.

19. Pass to getNumSeats(), the planeType.

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

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

21. If you get an exception, remove the coercion to a Flight object.

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

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

```

1 %dw 1.0
2 %output application/java
3 %namespace ns0 http://soap.training.mulesoft.com/
4 %type currency = :string {format: "###.##"}
5 %type flight = :object {class: "com.mulesoft.training.Flight"}
6 ---
7 @ flights: payload.ns0:listAllFlightsResponse.*return map {
8     destination: $.destination,
9     price: $.price as :number as currency,
10    planeType: upper ($.planeType replace /(Boing)/ with "Boeing"),
11    departureDate: $.departureDate as :date {format: "yyyy/MM/dd"},
12    as :string {format: "MMM dd, yyyy"},
13    availableSeats: $.emptySeats as :number,
14    totalSeats: getNumSeats($.planeType)
15 }
16
17

```

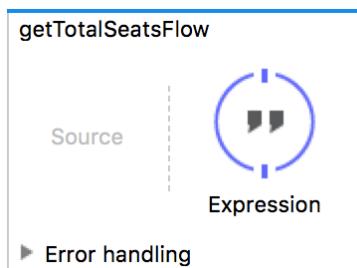
Name	Value
root	LinkedHashMap
flights	ArrayList
[0]	LinkedHashMap
destination	SFO
price	400.00
planeType	BOEING 737
departureDate	Oct 20, 2015
availableSeats	40
totalSeats	150
[1]	LinkedHashMap
destination	PDX
price	283.00
planeType	BOEING 777
departureDate	Oct 21, 2015
availableSeats	30
totalSeats	300

## Create a lookup flow

23. Return to implementation.xml.

24. Drag an Expression component (not a filter or a transformer!) from the Mule Palette and drop it at the bottom of the canvas to create a new flow.

25. Change the name of the flow to getTotalSeatsFlow.



26. Return to global.xml and copy the if/else block.

27. Return to implementation.xml.

28. In the Expression properties view, paste the if/else block you copied into the expression text area.

29. In the expression, change type to payload.type.

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

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

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

The screenshot shows the 'Expression' editor window in Mule Studio. The title bar includes tabs for 'Expression', 'Problems', 'Console', and 'Mule Debugger'. A message at the top says 'There are no errors.' The left sidebar has 'General' selected, along with 'Notes' and 'Metadata'. The main area contains a 'Display Name' field with 'Expression' and a 'Settings' section. Under 'Settings', there is a radio button for 'Expression:' which is selected, and a code editor containing the provided DataWeave code. The code is highlighted with syntax coloring: 'if' is red, 'payload' is blue, and numbers are yellow.

## Call a flow from a DataWeave expression

31. Return to the Transform Message properties view of the component in postMultipleFlightsFlow.

32. Comment out the existing totalSeats assignment by placing // in front of it.

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

33. Add another property called totalSeats inside the transformation function.

34. Set totalSeats equal to the return value from the DataWeave lookup() function.

```
totalSeats: lookup()
```

35. Pass to lookup() an argument that is equal to the name of the flow to call: "getTotalSeatsFlow".

36. Pass an object to lookup() as the second argument.

37. Give the object a field called plane (to match what the flow is expecting) and set it equal to the value of the planeType field.

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

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

The screenshot shows the MuleSoft Anypoint Studio interface. In the top right, there are tabs for 'Output', 'Payload' (with a dropdown menu), and 'Preview'. The 'Payload' tab is active, displaying the following MEL code:

```
1 %dw 1.0
2 %output application/java
3 %namespace ns0 http://soap.training.mulesoft.com/
4 %type currency = :string {format: "###.##"}
5 %type flight = :object {class: "com.mulesoft.training.Flight"}
6 ---
7 @ flights: payload.ns0:listAllFlightsResponse.*return map {
8     destination: $.destination,
9     price: $.price as :number as :currency,
10    planeType: upper ($.planeType replace /(Boing)/ with "Boeing"),
11    departureDate: $.departureDate as :date {format: "yyyy/MM/dd"}
12    as :string {format: "MMM dd, yyyy"},
13    availableSeats: $.emptySeats as :number,
14    //totalSeats: getNumSeats($.planeType)
15    totalSeats: lookup("getTotalSeatsFlow", {type: $.planeType})
16 }
```

The 'Preview' tab is also visible on the right, showing a table with two rows of flight data:

Name	Value
root	LinkedHashMap
flights	ArrayList
[0]	LinkedHashMap
destination	SFO
price	400.00
planeType	BOEING 737
departureDate	Oct 20, 2015
availableSeats	40
Unknown	
[1]	LinkedHashMap
destination	LAX
price	199.99
planeType	BOEING 737

## Test the application

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

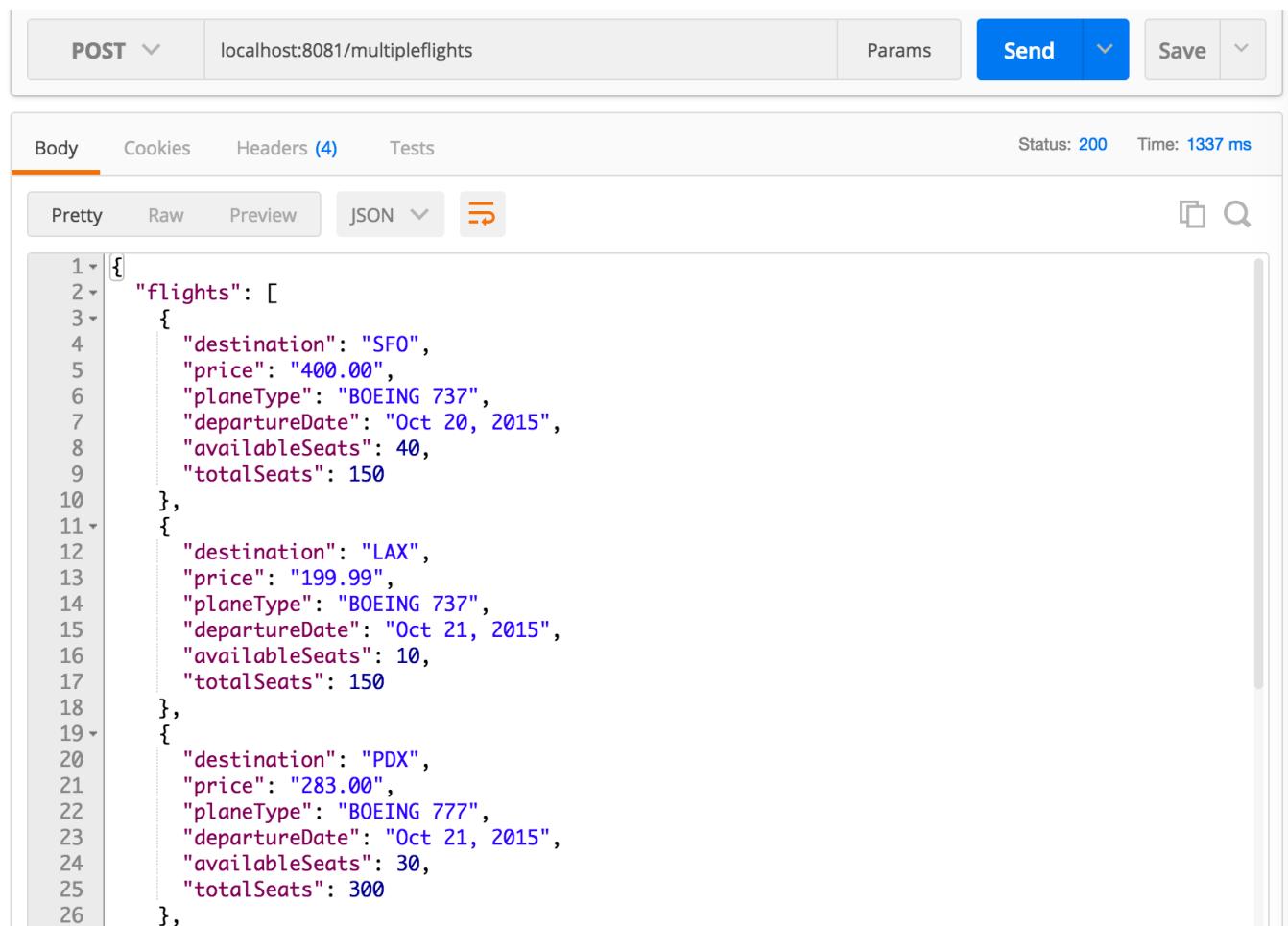
The screenshot shows the Postman application interface. The request details are as follows:

- Method: POST
- URL: localhost:8081/multipleflights
- Headers: (1)
- Body: raw XML (application/xml)
- Content-Type: application/xml

The XML body is defined as:

```
1 <ns2:listAllFlightsResponse xmlns:ns2="http://soap.training.mulesoft.com/">
2   <return airlineName="United">
3     <code>A1B2C3</code><departureDate>2015/10/20</departureDate>
4     <destination>SFO</destination><emptySeats>40</emptySeats>
5     <origin>MUA</origin>
6     <planeType>Boing 737</planeType>
7     <price>400.0</price>
8   </return>
9   <return airlineName="Delta">
10    <code>A1B2C4</code>
```

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



```
1 [ {  
2   "flights": [  
3     {  
4       "destination": "SFO",  
5       "price": "400.00",  
6       "planeType": "BOEING 737",  
7       "departureDate": "Oct 20, 2015",  
8       "availableSeats": 40,  
9       "totalSeats": 150  
10      },  
11      {  
12        "destination": "LAX",  
13        "price": "199.99",  
14        "planeType": "BOEING 737",  
15        "departureDate": "Oct 21, 2015",  
16        "availableSeats": 10,  
17        "totalSeats": 150  
18      },  
19      {  
20        "destination": "PDX",  
21        "price": "283.00",  
22        "planeType": "BOEING 777",  
23        "departureDate": "Oct 21, 2015",  
24        "availableSeats": 30,  
25        "totalSeats": 300  
26      }  
},  
]  
]
```

45. Stop the project.

46. Close the project.