



# Module 10: Writing DataWeave Transformations

## Goal



- You have been using DataWeave to transform data throughout class – but (mostly) only using the graphical drag-and-drop editor
- In this module, you learn to write DataWeave transformations from scratch and get familiar with the language to write more complicated transformations that are not possible with the drag-and-drop GUI

## Goal



Output Payload ▾

```
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: $.emptySeats as :number,
14  //totalSeats: getNumSeats($.planeType)
15  totalSeats: lookup("getTotalSeatsFlow",{type: $.planeType})
16 }
```

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

All contents © MuleSoft Inc.

3

## Objectives



- Write DataWeave transformations 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

All contents © MuleSoft Inc.

4

# Introducing DataWeave transformations



## DataWeave data transformation language



- A universal, simple, JSON-like language for transforming and querying data
- Easy to write, easy to maintain, and capable of supporting simple to complex mappings for any data type
  - Supports XML, JSON, Java, CSV, EDI, fixed width, flat file, and COBOL copybook out-of-the-box
  - Extensible for new formats via an API
- More elegant and re-usable than custom code
  - Data transformations can be stored in external DWL files and used across applications

## Example DataWeave transformation expression



The **header** contains directives – high level info about the transformation

Input	Transform	Output
<pre>{   "firstname": "Max",   "lastname": "Mule" }</pre>	<pre>%dw 1.0 %output application/xml --- {   user: {     fname: payload.firstname,     lname: payload.lastname   } }</pre>	<pre>&lt;?xml version="1.0" encoding="UTF-8"?&gt; &lt;user&gt;   &lt;fname&gt;Max&lt;/fname&gt;   &lt;lname&gt;Mule&lt;/lname&gt; &lt;/user&gt;</pre>

The **body** contains a DataWeave expression that generates the output structure

All contents © MuleSoft Inc.

8

## DataWeave expressions



- The DataWeave expression is a data model for the output
  - It is not dependent upon the types of the input and output, just their structures
  - It's against this model that the transform executes
- The data model of the produced output can consist of three different types of data
  - **Objects**: Represented as collection of key value pairs
  - **Arrays**: Represented as a sequence of comma separated values
  - **Simple literals**

All contents © MuleSoft Inc.

9

## The output directive



- Sets the output type of the transformation

- Specified using content/type
  - application/json, text/json
  - application/xml, text/xml
  - application/java, text/java
  - application/csv, text/csv
  - application/dw

```
%dw 1.0
%output application/xml
---
{
  a: payload
}
```

- The structure of the output is defined in the DataWeave body

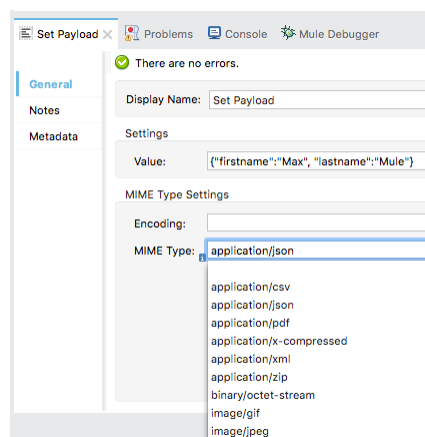
All contents © MuleSoft Inc.

10

## Setting input data MIME types



- When you run your application, you may get an error unless the MIME type for the input data has been set
  - It may be set automatically if the data is posted to the flow
    - See inbound properties content-type
  - Or you can define the input metadata type as you have been doing throughout class
  - Or, you can set the mimeType property of a preceding message processor



All contents © MuleSoft Inc.

## Where is the code?



- By default, it is placed inline

```
<flow name="json2xml">
  <http:listener config-ref="HTTP_Listener_Configurati
  <set-payload value="{&quot;firstname&quot;:&quot;Max&quot;;&quot;lastname&quot;:&quot;Max&quot;}">
  <dw:transform-message doc:name="Transform Message">
    <dw:input-payload doc:sample="json.json"/>
    <dw:set-payload><![CDATA[%dw 1.0
%output application/xml
---
{
  user: {
    lname: payload.lastname,
    fname: payload.firstname
  }
}]]></dw:set-payload>
  </dw:transform-message>
  <logger level="INFO" doc:name="Logger"/>
</flow>
```

- Any added sample data is stored in src/test/resources

```
src/test/java
└─ src/test/resources
   └─ sample_data
      └─ json.json
```

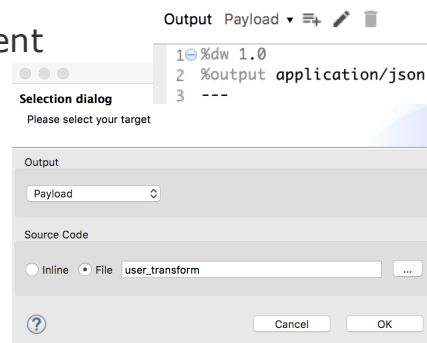
All contents © MuleSoft Inc.

12

## Reusing transformations



- To place it in an external file, click the Edit current target button and set source code to file
  - Transform is saved in a DWL file
  - DWL files are stored in src/main/resources



- To reference an existing DWL, specify it in code

```
<flow name="json2xml">
  <http:listener config-ref="HTTP_Listener_Configuration" path="/json" allowedMe
  <set-payload value="{&quot;firstname&quot;:&quot;Max&quot;;&quot;lastname&quot;:&quot;Max&quot;}">
  <dw:transform-message doc:name="Transform Message">
    <dw:input-payload doc:sample="json.json"/>
    <dw:set-payload resource="classpath:user_transform.dwl"></dw:set-payload>
  </dw:transform-message>
  <logger level="INFO" doc:name="Logger"/>
</flow>
```

```
src/main/java
└─ src/main/resources
   └─ user_transform.dwl
src/test/java
└─ src/test/resources
   └─ sample_data
      └─ json.json
```

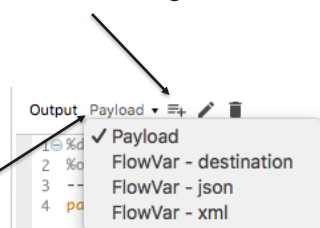
13

## Creating multiple transformations



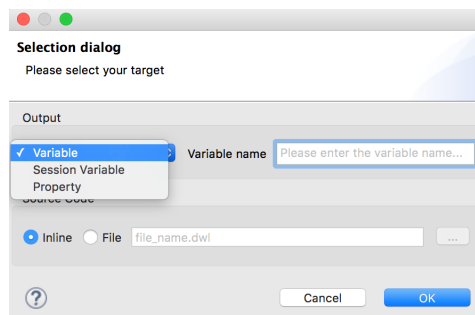
- You can also create multiple transformations with one Transform Message component

(1) Use Add new target button



(3) Switch between multiple transformations

(2) Set where to store results: Variable/property type and name



All contents © MuleSoft Inc.

14

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



- 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



# Transforming basic data structures

## Writing expressions for JSON or Java input and output



- The data model can consist of three different types of data: objects, arrays, simple literals

Input	Transform	JSON output
<pre>{   "firstname": "Max",   "lastname": "Mule" }</pre>	fname: payload.firstname	<pre>{"fname": "Max"}</pre>
	{fname: payload.firstname}	<pre>{"fname": "Max"}</pre>
	<pre>user: {   fname: payload.firstname,   lname: payload.lastname,   num: 1 }</pre>	<pre>{"user": {   "fname": "Max",   "lname": "Mule",   "num": "1" }}</pre>
	<pre>[   {fname: payload.firstname,     num: 1},   {lname: payload.lastname,     num: 2} ]</pre>	<pre>[   {"fname": "Max", "num": 1},   {"lname": "Mule", "num": 2} ]</pre>

All contents © MuleSoft Inc.

17



## Writing expressions for XML output



- XML can only have one top-level value and that value must be an object with one property
  - This will throw an exception

### List of errors

Select an error to see details below

```
1 @dw 1.0
2 %output application/xml
3 ---
4 {
5     fname: payload.firstname,
6     lname: payload.lastname
7 }
```

Name	Target
java.xml.stream.XMLStreamException: Trying to output second root, <lname> Payload	Payload

- This will work

```
1 @dw 1.0
2 %output application/xml
3 ---
4 {
5     user: {
6         fname: payload.firstname,
7         lname: payload.lastname
8     }
9 }
```

```
<?xml version='1.0' encoding='UTF-8'?>
<user>
  <fname>Max</fname>
  <lname>Mule</lname>
</user>
```

All contents © MuleSoft Inc.

18

## Specifying attributes for XML output



- Use @(attName: attValue) to create an attribute

```
1 @dw 1.0
2 %output application/xml
3 ---
4 {
5     user @(fname: payload.firstname,
6           lname: payload.lastname): {
7         lname: payload.lastname
8     }
9 }
```

```
<?xml version='1.0' encoding='UTF-8'?>
<user fname="Max" lname="Mule">
  <lname>Mule</lname>
</user>
```

All contents © MuleSoft Inc.

19

## Writing expressions for XML input



- By default, only XML elements and not attributes are created as JSON fields or Java object properties
- Use @ to reference attributes

Input	Transform	JSON Output
<pre>&lt;user firstname="Max"&gt;   &lt;lastname&gt;Mule&lt;/lastname&gt; &lt;/user&gt;</pre>	payload	<pre>{ "user": {   "lastname": "Mule" } }</pre>
	payload.user	<pre>{"lastname": "Mule" }</pre>
	<pre>{   fname: payload.user.@firstname,   lname: payload.user.lastname }</pre>	<pre>{"fname": "Max",   "lname": "Mule" }</pre>

All contents © MuleSoft Inc.

20

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



- 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

Output FlowVar - xml ▾ ✎ 🗑

Preview

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

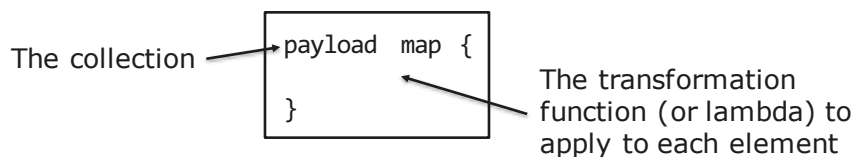
```
<?xml version='1.0' encoding='UTF-8'?>
<data>
  <hub>MUA</hub>
  <flight airline="United">
    <code>SFO</code>
  </flight>
</data>
```

# Transforming complex data structures with arrays

## Working with collections



- Use the **map** operator to apply a transformation to each element in a collection
  - A collection can be JSON or Java arrays or XML repeated elements



- The map operator
  - Returns an array of elements
  - Can be applied to each element in an array or each value in an object

## The transformation function



- Inside the transformation function
  - \$\$ refers to the index (or key)
  - \$ refers to the value

Input	Transform	Output
<pre>[   {"firstname":"Max",    "lastname":"Mule"},   {"firstname":"Molly",    "lastname":"Mule"} ]</pre>	<pre>%dw 1.0 %output application/json --- payload map {   num: \$\$,   fname: \$.firstname,   lname: \$.lastname }</pre>	<pre>[   {"num": 0,    "fname": "Max",    "lname": "Mule"},   {"num": 1,    "fname": "Molly",    "lname": "Mule"} ]</pre>
	<pre>%dw 1.0 %output application/json --- users: payload map {   user: {     fname: \$.firstname,     lname: \$.lastname   } }</pre>	<pre>{   "users": [     {       "user": {         "fname": "Max",         "lname": "Mule"       }     },     {       "user": {         "fname": "Molly",         "lname": "Mule"       }     }   ] }</pre>

All contents © MuleSoft Inc.

24

## Using the index as a key in a transformation function



- To set the index as a new key, surround it with () or ''

Input	Transform	Output
<pre>[   {"firstname":"Max",    "lastname":"Mule"},   {"firstname":"Molly",    "lastname":"Mule"} ]</pre>	<pre>%dw 1.0 %output application/json --- payload map {   num: \$\$,   (\$\$): \$ }</pre>	<pre>[   {     "num": 0,     "0": {       "firstname": "Max",       "lastname": "Mule"     }   },   {     "num": 1,     "1": {       "firstname": "Molly",       "lastname": "Mule"     }   } ]</pre>
	<pre>payload map {   num: \$\$,   '\$\$': \$ }</pre>	
	<pre>payload map {   'num\$\$': \$ }</pre>	<pre>[   {     "num0": {       "firstname": "Max",       "lastname": "Mule"     }   },   {     "num1": {       "firstname": "Molly",       "lastname": "Mule"     }   } ]</pre>

All contents © MuleSoft Inc.

25

## Walkthrough 10-3: Transform complex data structures



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

Output Payload ▾ Preview

```

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

```

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

26

## Transforming complex XML data structures

## Writing expressions for XML output



- When mapping array elements (JSON or JAVA) to XML, wrap the map operation in `{{ ... }}`
  - `{}` are defining the object
  - `()` are transforming each element in the array as a key/value pair

Input	Transform	Output
<pre>[   {"firstname": "Max",    "lastname": "Mule"},   {"firstname": "Molly",    "lastname": "Mule"} ]</pre>	<pre>%dw 1.0 %output application/xml --- users: payload map {   fname: \$.firstname,   lname: \$.lastname }</pre>	Cannot coerce an array to an object
	<pre>users: {(payload map {   fname: \$.firstname,   lname: \$.lastname })}</pre>	<pre>&lt;users&gt;   &lt;fname&gt;Max&lt;/fname&gt;   &lt;lname&gt;Mule&lt;/lname&gt;   &lt;fname&gt;Molly&lt;/fname&gt;   &lt;lname&gt;Mule&lt;/lname&gt; &lt;/users&gt;</pre>

All contents © MuleSoft, Inc.

28

## Writing expressions for XML output (cont)



Input	Transform	Output
<pre>[   {"firstname": "Max",    "lastname": "Mule"},   {"firstname": "Molly",    "lastname": "Mule"} ]</pre>	<pre>users: {(payload map {   fname: \$.firstname,   lname: \$.lastname })}</pre>	<pre>&lt;users&gt;   &lt;fname&gt;Max&lt;/fname&gt;   &lt;lname&gt;Mule&lt;/lname&gt;   &lt;fname&gt;Molly&lt;/fname&gt;   &lt;lname&gt;Mule&lt;/lname&gt; &lt;/users&gt;</pre>
	<pre>users: {( payload map {   user: {     fname: \$.firstname,     lname: \$.lastname   } })}</pre>	<pre>&lt;users&gt;   &lt;user&gt;     &lt;fname&gt;Max&lt;/fname&gt;     &lt;lname&gt;Mule&lt;/lname&gt;   &lt;/user&gt;   &lt;user&gt;     &lt;fname&gt;Molly&lt;/fname&gt;     &lt;lname&gt;Mule&lt;/lname&gt;   &lt;/user&gt; &lt;/users&gt;</pre>

All contents © MuleSoft, Inc.

29

## Writing expressions for XML input



- Use \* to reference repeated elements

Input	Transform	JSON Output
<pre>&lt;users&gt;   &lt;user firstname="Max"&gt;     &lt;lastname&gt;Mule&lt;/lastname&gt;   &lt;/user&gt;   &lt;user firstname="Molly"&gt;     &lt;lastname&gt;Jennet&lt;/lastname&gt;   &lt;/user&gt; &lt;/users&gt;</pre>	payload	<pre>{   "users": {     "user": { "lastname": "Mule" },     "user": { "lastname": "Jennet" }   } }</pre>
	payload.users	<pre>{   "user": { "lastname": "Mule" },   "user": { "lastname": "Jennet" } }</pre>
	payload.users.user	<pre>{ "lastname": "Mule" }</pre>
	payload.users.*user	<pre>[   { "lastname": "Mule" },   { "lastname": "Jennet" } ]</pre>
	payload.users.*user map { fname: \$.@firstname, lname: \$.lastname }	<pre>[   { "fname": "Max", "lname": "Mule" },   { "fname": "Molly", "lname": "Jennet" } ]</pre>

All contents © MuleSoft Inc.

30

## Beware of tools that “prettify” responses



- In some tools (like Postman), make sure you look at the raw response and not the pretty response

Input	Transform
<pre>&lt;users&gt;   &lt;user firstname="Max"&gt;     &lt;lastname&gt;Mule&lt;/lastname&gt;   &lt;/user&gt;   &lt;user firstname="Molly"&gt;     &lt;lastname&gt;Jennet&lt;/lastname&gt;   &lt;/user&gt; &lt;/users&gt;</pre>	payload



All contents © MuleSoft Inc.

31

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



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

```

<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>
</ns2:listAllFlightsResponse>
  
```

```

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>
  
```

All contents © MuleSoft Inc.

32

# Formatting and coercing data



## DataWeave documentation



- <https://docs.mulesoft.com/mule-user-guide/v/3.8/dataweave>

## Formatting operators



Input	Transform	Output
	<code>%dw 1.0 %output application/xml ---</code>	
<code>{"name": "max_mule"}</code>	<code>n: upper (dasherize payload.name)</code>	<code>&lt;n&gt;MAX-MULE&lt;/n&gt;</code>

Upper  
Lower  
Camelize  
Capitalize  
Dasherize  
Underscore  
Pluralize  
Singularize  
Trim  
Ordinalize

## Using the as operator for type coercion



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

- Defined types include
  - :string
  - :number
  - :boolean
  - :object
  - :array
  - :date, :time, :timezone, :datetime, :localdatetime, :period
  - :regex, more...
- See what conversions between what types are allowed in DataWeave
  - <https://docs.mulesoft.com/mule-user-guide/v/3.8/dataweave-types#type-coercion-table>

All contents © MuleSoft Inc.

37

## Using format patterns



- Use metadata **format** schema property to format numbers and dates

```
someDate as :datetime {format: "yyyyMMddHHmm"}
```

- For formatting patterns, see
  - <https://docs.oracle.com/javase/8/docs/api/java/time/format/DateTimeFormatter.html>
  - <https://docs.oracle.com/javase/8/docs/api/java/text/DecimalFormat.html>

All contents © MuleSoft Inc.

38

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



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

Output Payload ▾

```

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

```

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

Preview

# Using DataWeave operators

## Math operators



- +
- -
- \*
- /

Basic Math  
Operations

Max

Min

Round

Sqrt

Pow

Ceil

Floor

Abs

Mod

All contents © MuleSoft Inc.

41

## Conditional logic operators



- default
  - when
  - unless
  - otherwise
- In expressions, you can use
    - ==
    - !=
    - ~= (equal regardless of type)

Input	Transform	Output
	%dw 1.0 %output application/xml ---	
{ "firstname": "Max", "lastname": "Mule" }	n: payload.nickname <b>default</b> payload.firstname	<n>Max</n>
{ "firstname": "Max", "lastname": "Mule", "nickname": "" }	n: payload.nickname <b>when</b> payload.nickname != "" <b>otherwise</b> payload.firstname	<n>Max</n>

All contents © MuleSoft Inc.

43

## Additional operators

- flatten
- orderBy
- concat
- distinctBy
- groupBy
- replace
- matches
- regex
- More...

All contents © MuleSoft Inc.

Join By  
Split By  
Order By  
Group By  
Distinct By  
Zip Arrays  
Unzip Array  
Replace  
Matches  
Starts With  
Ends With  
Find  
Match  
Scan  
Similar

## Walkthrough 10-6: Use DataWeave operators



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

Output Payload ▾

```
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
```

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

All contents © MuleSoft Inc.

45

# Using custom data types



## Specifying custom data types



- Use type header directive
  - Name has to be all lowercase letters
    - No special characters, uppercase letters, or numbers

```
%dw 1.0
```

```
%output application/json
```

```
%type ourdateformat = :datetime {format: "yyyyMMddHHmm"}
```

```
---
```

```
someDate: payload.departureDate as :ourdateformat
```

## Transforming objects to POJOs



- Use as :object
- Specify inline

```
customer:payload.user as :object {class: "my.company.User"}
```

- Or define a custom data type to use

```
%type user = :object {class: "my.company.User"}
customer:payload.user as :user
```

All contents © MuleSoft Inc.

48

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



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

Output Payload ▾

```
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
```

Preview

Name	Value
root : LinkedHashMap	
flights : ArrayList	
[0] : Flight	
airlineName : String	
availableSeats : Integer	40
departureDate : String	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	

All contents © MuleSoft Inc.

49

# Referencing other data besides the payload



## Referencing other data besides the payload



- Up to now, we have only referenced the message payload in the DataWeave expression
- You can also
  - Reference message variables and properties
  - Call global MEL functions
  - Call other Mule flows



## Referencing message variables and properties



- In addition to payload, you can also reference
  - flowVars
  - inboundProperties
  - outboundProperties
  - p (System or Spring properties)

```
{n: flowVars.username}
```

- This is not MEL!
  - Do not preface these values by "message." or use #[]

## Calling global MEL functions



- Define global MEL functions in the configuration element you used to specify a default global exception handler

```
<configuration doc:name="Configuration" defaultExceptionHandler-ref="..">
  <expression-language>
    <global-functions>
      def newUser() { return ["name" : "max"] }
      def upperName(user) {
        return user.name.toUpperCase()
      }
    </global-functions>
  </expression-language>
</configuration>
```

- Call the MEL functions from DataWeave

```
{"foo" : newUser(),
 "bar": upperName(newUser())}
```

## Calling external flows



- From a DataWeave transform, you can call a different flow in your Mule application
  - Whatever the flow returns is what the expression returns

```
{a: lookup("mySecondFlow",{b:"Hello"}) }
```

- The first argument is the name of the flow to be called
- The second argument is the payload to send to the flow as a map

All contents © MuleSoft Inc.

54

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

- 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 displays the Mule Studio IDE interface. On the left, the 'global.xml' file is open, showing the implementation of a global MEL function named `getNumSeats`. The function is defined within an `<expression-language>` block and uses a `def` statement to define the function logic. The logic checks if the `type` contains the string '737'. If it does, it returns 150; otherwise, it returns 300. On the right, the 'Expression' editor is open, showing the DataWeave expression `lookup("getTotalSeatsFlow",{type: $.planeType})` being used in a flow. The 'Settings' tab is active, showing the expression and its logic. The 'Problems' tab shows 'There are no errors.'

55

# Summary



## Summary



- DataWeave is a JSON-like language
- The data model for the transformation can consist of three different types of data: objects, arrays, and simple literals
- DataWeave supports XML, JSON, Java, CSV, EDI, fixed width, flat file, and COBOL copybook out-of-the-box
- Transformations can be written inline or stored in external DWL files
- There is a large set of operators that can be used in the expressions
  - upper, replace, as, orderBy, distinctBy, filter, flatten

## Summary



- Use the metadata format schema property to format numbers and dates
- Use the type header directive to specify custom data types
- Transform objects to POJOs using `as:object {class: "com.myPOJO"}`
- Transformations can reference the message payload, flow variables, inbound properties, and outbound properties
- Transformations can call global MEL functions
- Use `lookup()` to get data by calling other flows