MuleSoft®

Module 11: Writing DataWeave Transformations

Goal

MuleSoft

- 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
    - Get familiar with the language to write more complicated transformations that are not possible with the drag-and-drop GUI

2

Goal

MuleSoft

Output Payload ▾ Preview

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

3

At the end of this module, you should be able to

MuleSoft

- 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

4

Introducing the DataWeave data transformation language

DataWeave data transformation language

MuleSoft

- 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

- More elegant and re-usable than custom code
  - Data transformations can be stored in external DWL files and used across applications

All contents © MuleSoft Inc.

6

Example DataWeave transformation expression

MuleSoft

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

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

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

8

DataWeave expressions

MuleSoft

- 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**

9

The output directive

MuleSoft

- 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

10

Setting input data MIME types

MuleSoft

- 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
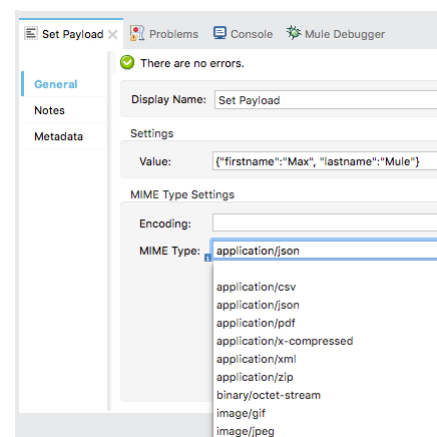
Where is the code?

MuleSoft

- By default, it is placed inline

```xml
<flow name="json2xml">
    <http:listener config-ref="HTTP_Listener_Configurati
    <set-payload value="{&quot;firstname&quot;:&quot;Max
    <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>
```

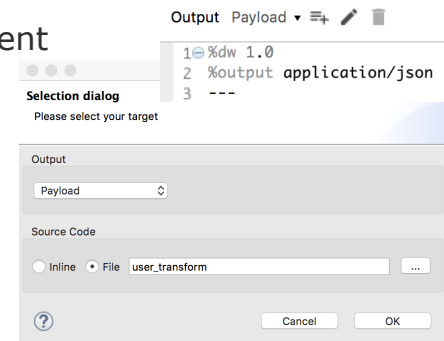- Any added sample data is stored in src/test/resources

  📁 src/test/java
  ▼ 📁 src/test/resources
  　　▼ 🗂 sample_data
  　　　　📄 json.json

12

Reusing transformations

MuleSoft

- To place it 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

Output Payload ▾ ☰+ ✏ 🗑

```
1 %dw 1.0
2 %output application/json
3 ---
```

Selection dialog
Please select your target

Output
| Payload ⌄ |

Source Code

○ Inline ⦿ File user_transform [...]

(?)　　　　　　Cancel　　OK

- To reference an existing DWL, specify it in code

```xml
<flow name="json2xml">
    <http:listener config-ref="HTTP_Listener_Configuration" path="/json" allowedMe
    <set-payload value="{&quot;firstname&quot;:&quot;Max&quot;, &quot;lastname&quo
    <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
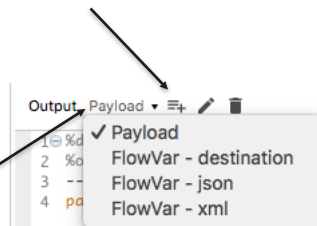▼ 📁 src/test/resources
　　▼ 🗂 sample_data
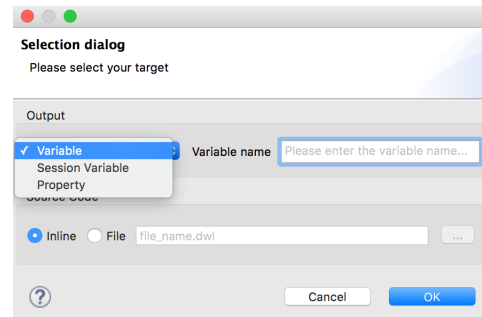　　　　📄 json.json

13

Creating multiple transformations

MuleSoft

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

(1) Use Add new target button

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

(3) Switch between multiple transformations

All contents © MuleSoft Inc.

14

Walkthrough 11-1: Write your first DataWeave transformation

MuleSoft

- 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

MuleSoft

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

Input	Transform	JSON output

{
  "firstname":"Max",
  "lastname":"Mule"
}
	fname: payload.firstname	{"fname": "Max"}

	{fname: payload.firstname}	{"fname": "Max"}

	user: {
	 fname: payload.firstname,
	 lname: payload.lastname,
	 num: 1
	}
		{"user": {
		  "fname": "Max",
		  "lname": "Mule",
		  "num": 1
		}}

	[{fname: payload.firstname, num: 1}, {lname: payload.lastname, num: 2}]	[{"fname": "Max","num": 1}, {"lname": "Mule","num": 2}]

17

Writing expressions for XML output

MuleSoft

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

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

List of errors
Select an error to see details below

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

  - This will work

All contents © MuleSoft Inc. 18

Specifying attributes for XML output

MuleSoft

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

All contents © MuleSoft Inc. 19

Writing expressions for XML input

MuleSoft

- 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

`<user firstname="Max">`
`  <lastname>Mule</lastname>`
`</user>`
	`payload`	`{ "user": {`
		`  "lastname": "Mule"`
		`  }`
		`}`

	`payload.user`	`{"lastname": "Mule" }`

	`{`
	`  fname: payload.user.@firstname,`
	`  lname: payload.user.lastname`
	`}`
		`{"fname": "Max",`
		`"lname": "Mule" }`

20

Walkthrough 11-2: Transform basic Java, JSON, and XML data structures

MuleSoft

- 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 ▾ ☰+ ✏ 🗑   [N] [□□] [□] 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>
```
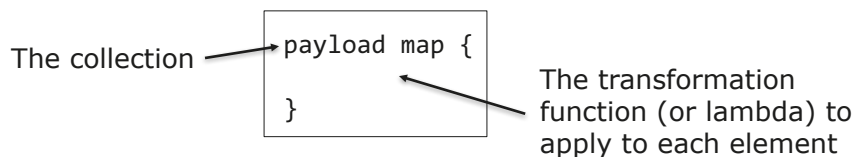
10

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 collection →
```
payload map {

}
```
The transformation function (or lambda) to apply to each element

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

23

The transformation function

MuleSoft

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

Input	Transform	Output

`[`
`  {"firstname":"Max",`
`   "lastname":"Mule"},`
`  {"firstname":"Molly",`
`   "lastname":"Mule"}`
`]`
	`%dw 1.0`
	`%output application/json`
	`---`
	`payload map {`
	`        num: $$,`
	`        fname: $.firstname,`
	`        lname: $.lastname`
	`}`
		`[`
		`  {"num": 0,`
		`     "fname": "Max",`
		`     "lname": "Mule"},`
		`  {"num": 1,`
		`     "fname": "Molly",`
		`     "lname": "Mule"}`
		`]`

	`%dw 1.0`
	`%output application/json`
	`---`
	`users:  payload map {`
	`        user: {`
	`            fname: $.firstname,`
	`            lname: $.lastname`
	`        }`
	`}`
		`{`
		`  "users": [`
		`    {"user": {`
		`        "fname": "Max",`
		`        "lname": "Mule"`
		`    }},`
		`    {"user": {`
		`        "fname": "Molly",`
		`        "lname": "Mule"`
		`    }}`
		`  ]`
		`}`

All contents © MuleSoft Inc.

24

Using the index as a key in a transformation function

MuleSoft

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

Input	Transform	Output

`[`
`  {"firstname":"Max",`
`   "lastname":"Mule"},`
`  {"firstname":"Molly",`
`   "lastname":"Mule"}`
`]`
	`%dw 1.0`
	`%output application/json`
	`---`
	`payload map {`
	`   num: $$,`
	`   ($$): $`
	`}`
		`[`
		`  {`
		`    "num": 0,`
		`    "0": {`
		`      "firstname": "Max",`
		`      "lastname": "Mule"`
		`    }`
		`  },`
		`  {`
		`    "num": 1,`
		`    "1": {`
		`      "firstname": "Molly",`
		`      "lastname": "Mule"`
		`    }`
		`  }`
		`]`

	`payload map {`
	`   num: $$,`
	`   '$$': $`
	`}`	

	`payload map {`
	`   'num$$': $`
	`}`
		`[`
		`  {`
		`    "num0": {`
		`      "firstname": "Max",`
		`      "lastname": "Mule"`
		`    }`
		`  },`
		`  {`
		`    "num1": {`
		`      "firstname": "Molly",`
		`      "lastname": "Mule"`
		`    }`
		`  }`
		`]`

All contents © MuleSoft Inc.

25

12

Walkthrough 11-3: Transform complex data structures

MuleSoft

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

Transforming complex XML data structures

Writing expressions for XML output

MuleSoft

- 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

[
 {"firstname":"Max",
  "lastname":"Mule"},
 {"firstname":"Molly",
  "lastname":"Mule"}
]
	```%dw 1.0```
	```%output application/xml```
	```---```
	```users: payload map {```
	``` fname: $.firstname,```
	``` lname: $.lastname```
	```}```
		Cannot coerce an array to an object

	```users: {(payload map {```
	``` fname: $.firstname,```
	``` lname: $.lastname```
	```})}```
		```<users>```
		``` <fname>Max</fname>```
		``` <lname>Mule</lname>```
		``` <fname>Molly</fname>```
		``` <lname>Mule</lname>```
		```</users>```

All contents © MuleSoft Inc.

28

Writing expressions for XML output (cont)

MuleSoft

Input	Transform	Output

[
 {"firstname":"Max",
  "lastname":"Mule"},
{"firstname":"Molly",
  "lastname":"Mule"}
]
	```users: {(payload map {```
	``` fname: $.firstname,```
	``` lname: $.lastname```
	```})}```
		```<users>```
		``` <fname>Max</fname>```
		``` <lname>Mule</lname>```
		``` <fname>Molly</fname>```
		``` <lname>Mule</lname>```
		```</users>```

	```users: {(payload map {```
	``` user: {```
	``` fname: $.firstname,```
	``` lname: $.lastname```
	``` }```
	``` }```
	```)}```
		```<users>```
		``` <user>```
		``` <fname>Max</fname>```
		``` <lname>Mule</lname>```
		``` </user>```
		``` <user>```
		``` <fname>Molly</fname>```
		``` <lname>Mule</lname>```
		``` </user>```
		```</users>```

All contents © MuleSoft Inc.

29

14

Writing expressions for XML input

MuleSoft

- Use * to reference repeated elements

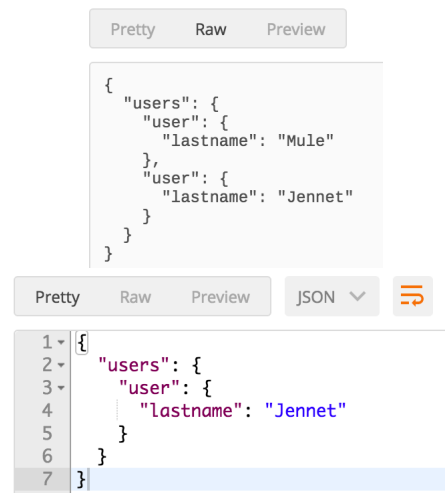| Input	Transform	JSON Output

`<users>`
`<user firstname="Max">`
`<lastname>Mule</lastname>`
`</user>`
`<user firstname="Molly">`
`<lastname>Jennet</lastname>`
`</user>`
`</users>`
	`payload`	`{"users":`
		`{"user": {"lastname": "Mule"},`
		`"user": {"lastname": "Jennet"}`
		`}`
		`}`

	`payload.users`	`{{"user": {"lastname": "Mule"},`
		`"user": {"lastname": "Jennet"}`
		`}`
		`}`

	`payload.users.user`	`{"lastname": "Mule" }`

	`payload.users.*user`	`[`
		`{"lastname": "Mule" },`
		`{"lastname": "Jennet" }`
		`]`

	`payload.users.*user map {`
	`    fname: $.@firstname,`
	`    lname: $.lastname`
	`}`
		`[`
		`{"fname": "Max", "lname": "Mule" },`
		`{"fname": "Molly", "lname": "Jennet" }`
		`]`

All contents © MuleSoft Inc.

30

Beware of tools that "prettify" responses

MuleSoft

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

Input	Transform

`<users>`
`<user firstname="Max">`
`<lastname>Mule</lastname>`
`</user>`
`<user firstname="Molly">`
`<lastname>Jennet</lastname>`
`</user>`
`</users>`
	`payload`

Pretty · **Raw** · Preview

```
{
  "users": {
    "user": {
      "lastname": "Mule"
    },
    "user": {
      "lastname": "Jennet"
    }
  }
}
```

Pretty · Raw · Preview   JSON ∨

```
1 ▾ {
2 ▾   "users": {
3 ▾     "user": {
4         "lastname": "Jennet"
5       }
6     }
7 }
```

All contents © MuleSoft Inc.

31

15

Walkthrough 11-4: Transform to and from XML with repeated elements

MuleSoft

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

32

Formatting and coercing data

DataWeave documentation

MuleSoft

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

34

Formatting operators

MuleSoft

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

Upper

Lower

Camelize

Capitalize

Dasherize

Underscore

Pluralize

Singularize

Trim

Ordinalize

All contents © MuleSoft Inc.

35

Using the as operator for type coercion

MuleSoft

```
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

37

Using format patterns

MuleSoft

- 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

38

Walkthrough 11-5: Coerce and format strings, numbers, and dates

MuleSoft

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

Output Payload ▾ ≡₊ ✎ 🗑   ⊡ ⊡ ⊡ Preview

```
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
▼ e root : *LinkedHashMap*	
▼ e flights : *ArrayList*	
▼ e [0] : *LinkedHashMap*	
○ dest : *String*	SFO
○ price : *String*	400.00
○ plane : *String*	BOING 737
○ date : *String*	Oct 20, 2015
▼ e [1] : *LinkedHashMap*	
○ dest : *String*	LAX
○ price : *String*	199.99
○ plane : *String*	BOING 737
○ date : *String*	Oct 21, 2015

Using DataWeave operators

Math operators

MuleSoft

- +
- -
- *
- /

| Basic Math Operations |
| Max |
| Min |
| Round |
| Sqrt |
| Pow |
| Ceil |
| Floor |
| Abs |
| Mod |

41

Conditional logic operators

MuleSoft

- 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 **default** payload.firstname	<n>Max</n>
{"firstname":"Max", "lastname":"Mule", "nickname":""}	n: payload.nickname **when** payload.nickname != "" **otherwise** payload.firstname	<n>Max</n>

43

Additional operators

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

| |
| 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 11-6: 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
▼ e root : *LinkedHashMap*	
▼ e flights : *ArrayList*	
▶ e [0] : *LinkedHashMap*	
▶ e [1] : *LinkedHashMap*	
▶ e [2] : *LinkedHashMap*	
▼ e [3] : *LinkedHashMap*	
○ dest : *String*	SFO
○ price : *String*	400.00
○ plane : *String*	BOEING 737
○ date : *String*	Oct 20, 2015
○ seats : *Integer*	40

45

Using custom data types

Specifying custom data types

MuleSoft

- 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
```

47

22

Transforming objects to POJOs

MuleSoft

- 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
```

48

Walkthrough 11-7: Define and use custom data types

MuleSoft

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

```
Output  Payload ▾ ⬚+ ✎ 🗑                                              Ⓝ ☐☐ ☐   Preview
 1⊖ %dw 1.0                                              Name                    Value
 2  %output application/java                             ▼ e root : LinkedHashMap
 3  %namespace ns0 http://soap.training.mulesoft.com/       ▼ ⬚ flights : ArrayList
 4  %type currency = :string {format: "###.00"}                ▼ Ⓖ [0] : Flight
 5  %type flight = :object {class: "com.mulesoft.training.Flight"}       ○ flightCode : String
 6  ---                                                            ○ availableSeats : Integer  10
 7⊖ flights: payload.ns0#listAllFlightsResponse.*return map {       ○ origination : String
 8     destination: $.destination,                                 ○ price : Double           199.99
 9     price: $.price as :number as :currency,                     ○ destination : String     LAX
10     planeType: upper ($.planeType replace /(Boing)/ with "Boeing"),  ○ departureDate : String  Oct 21, 2015
11⊖    departureDate: $.departureDate as :date {format: "yyyy/MM/dd"}   ○ airlineName : String
12     as :string {format: "MMM dd, yyyy"},                        ○ planeType : String       BOEING 737
13     availableSeats: $.emptySeats as :number                ▼ Ⓖ [1] : Flight
14⊖ } as :flight orderBy $.departureDate orderBy $.price distinctBy $       ○ flightCode : String
15  filter ($.availableSeats !=0)
```

49

23

Referencing other data besides the payload

Referencing other data besides the payload

MuleSoft

- 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

51

Referencing message variables and properties

MuleSoft

- 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 #[]

52

Calling global MEL functions

MuleSoft

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

```
<configuration doc:name="Configuration" defaultExceptionStrategy-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())}
```

53

Calling external flows

MuleSoft

- 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

54

Walkthrough 11-8: 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})
```


55

Summary

Summary

MuleSoft

- 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

57

Summary

MuleSoft

- 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

58