MuleSoft®

Module 2: Use Case: XML Flights to JSON

Review: Use Case – XML flights to JSON

As a developer I want to receive XML flight records and generate a JSON data structure with the corresponding flights so that I communicate them with a third party system.

The JSON data structure contains:

- the flights from the XML inputs
- total seating capacity per flight that is a function of the plane type
- prices adjusted across a number of currencies
- important KPIs

Goal

Input

```
<?xml version="1.0" encoding="UTF-8"?>
<ns2:findFlightResponse
  xmlns:ns2="http://soap.training.mulesoft.com/">
   <return>
      <airlineName>Delta</airlineName>
      <code>A1B2C3</code>
      <departureDate>2018/03/20</departureDate>
      <destination>SFO</destination>
      <emptySeats>40</emptySeats>
      <origin>MUA</origin>
      <planeType>Boing 737</planeType>
      <price>400.0</price>
   </return>
   ….
</ns2:findFlightResonse
```

Expression

```
%dw 2.0
output application/dw
…
---
{
kpis: {
areAllDeltaFlights: payload..*return Arrays::every (e) -> e.airlineName == "Delta",
anyFullFlights: payload..*return Arrays::some $.emptySeats as Number == 0,
noOfFullFlights: payload..*return default [] Arrays::countBy $.emptySeats ~= 0,
sumOfEmptySeats: payload..*return default [] Arrays::sumBy $.emptySeats
},
data: payload..*return map {
($ - "planeType"),
totalSeats: getTotalSeatsL($.planeType),
planeType: $.planeType replace /Boing/ with "Boeing",
priceEUR: adjustFor($.price, "EUR"),
priceGBP: $.price adjustFor "GBP",
priceCAD: $.price dw::modules::Currency::adjustFor "CAD",
priceAUD: $.price Currency::adjustFor "AUD",
priceMXN: $.price Curr::adjustFor "MXN",
priceINR: $.price adj4 "INR"
}
}
```

Output

```
{
  kpis: {
    areAllDeltaFlights: true,
    anyFullFlights: true,
    noOfFullFlights: 1,
    sumOfEmptySeats: 375
  },
  data: [
    {
      airlineName: "Delta",
      code: "A1B2C3",
      departureDate: "2018/03/20",
      destination: "SFO",
      emptySeats: "40",
      origin: "MUA",
      price: "400.0",
      totalSeats: 155,
      planeType: "Boeing 737",
      priceEUR: 1000,
      priceGBP: 1000,
      priceCAD: 520.00,
      priceAUD: 600.00,
      priceMXN: 10000.0,
      priceINR: 28800.0
    },
  …
  ]
}
```

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

- Organize DataWeave code into variables and functions
- Enhance existing objects with extra fields
- Reuse transformations
- Create and use DataWeave modules

Organize DataWeave code into variables and functions

Creating and using variables

- The `var` keyword declares variables
  - Similar to JavaScript variables
  - A variable can be a constant literal

Input	Transform	Output
	%dw 2.0	{
	output application/json	"city": "San Francisco",
		"salesTax": 8.50
		}
	var theSalesTax = 8.50	
	var theCity = "San Francisco"	

	{	
	city: theCity,	
	salesTax: theSalesTax	
	}	

Defining constant variables for various data types

- String
  - Double or single quoted, such as "Hello", 'hello'

```
var theString = "Hello " ++ 'World'
```

- Boolean

```
var debug = true
```

- Number (both Decimal and Integer)
  - Formats can also be applied

```
var theNumber = 100 + 100.2
```

- Date
  - ISO-8601 enclosed in "|"

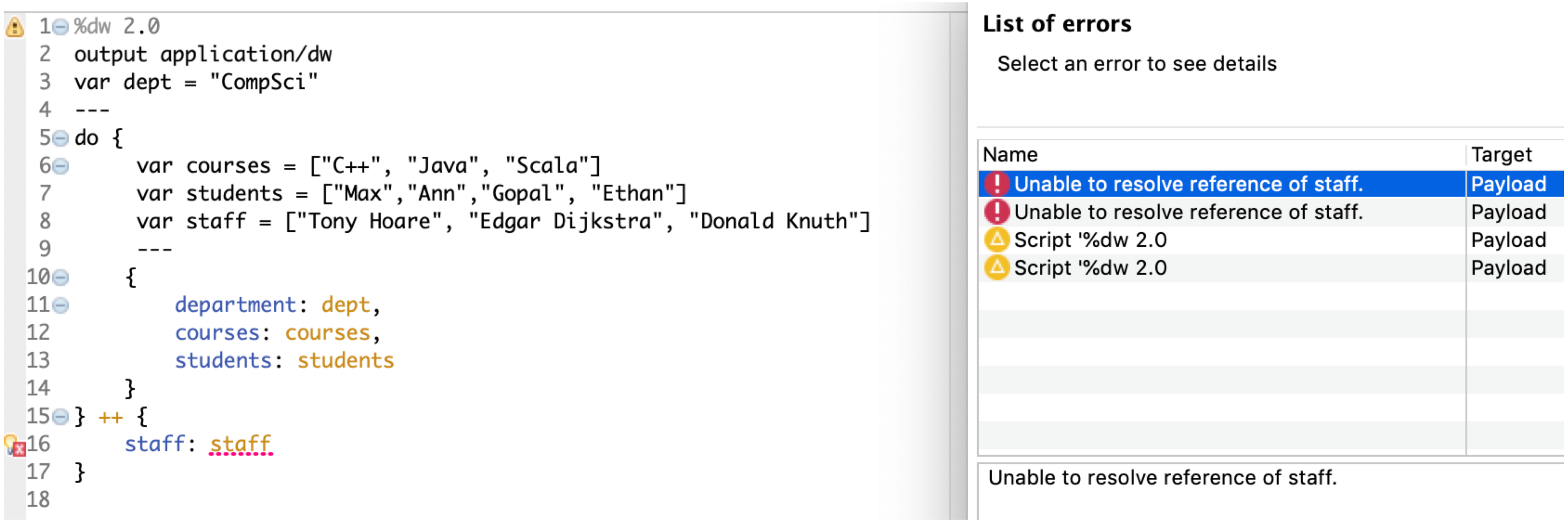
```
var theDate = |2003-10-01T23:57:59Z|
```

- Regex

```
var theRegex = /[a-zA-z0-9]+{10,}/
```

Variable scopes

- Variables declared in the header are global to the body
- Variables declared inside the `do {}` construct are localized



```
%dw 2.0
output application/dw
var dept = "CompSci"
---
do {
      var courses = ["C++", "Java", "Scala"]
      var students = ["Max","Ann","Gopal", "Ethan"]
      var staff = ["Tony Hoare", "Edgar Dijkstra", "Donald Knuth"]
      ---
    {
        department: dept,
        courses: courses,
        students: students
    }
} ++ {
    staff: staff
 }
```

List of errors

Select an error to see details

Name	Target
Unable to resolve reference of staff.	Payload
Unable to resolve reference of staff.	Payload
Script '%dw 2.0	Payload
Script '%dw 2.0	Payload

Unable to resolve reference of staff.

Defining and reusing functions

- Functions are Lambda expressions and vice-versa
- Functions in DataWeave are first-class citizens
  - Can be inputs to lambda expression
  - Can be returned from lambda expressions
  - Can be assigned to variables
- Two ways to declare functions
  - `fun id(e) = e`
  - `var id = (e) -> e`
- The **`fun`** directive is syntactic sugar to **`var`**
  - You MUST use `fun` for overloaded functions

Walkthrough 2-1: Organize DataWeave code with variables and functions

As a developer I want to receive XML flight records and generate a JSON data structure with the corresponding flights so that I communicate them with a third party system.

The JSON data structure contains:

- the flights from the XML inputs
  - using a selector
- total seating capacity per flight that is a function of the plane type
  - declaring variables and functions
- prices adjusted across a number of currencies
  - using a map containing exchange rates and functions

Reuse DataWeave Transformations

Where is the code?

- DataWeave code is embedded inline in the XML by default

```
  <ee:transform doc:name="Transform Message" doc:id="a51354d3-19ca-41de-aa1b-
c35e844db780" >

    <ee:message >

      <ee:set-payload ><![CDATA[%dw 2.0

        output application/java

        ---

        {

        }]]></ee:set-payload>

    </ee:message>

  </ee:transform>
```

How to reuse the code?

- Decouple the code from the XML
  - Store the DW code in a separate file
  - The location of the file must be under your classpath, usually src/main/resources, or a subfolder thereof
  - Use the pencil (Edit current target) button to store the code in a separate file
- Reuse the file by editing the XML

```
<ee:transform doc:name="Transform Message"
    doc:id="51587c84-8932-4268-a141-5afc56440444">
  <ee:message>
    <ee:set-payload
      resource="dw/transforms/mod2/flights.dwl" />
  </ee:message>
</ee:transform>
```

Walkthrough 2-2: Organize DataWeave code with variables and functions

- Decouple the DW code from the XML and store it in a file
- Reuse the DW code

MuleSoft®

Create and use DataWeave modules

Creating a DataWeave module

- Create a file under your classpath or a subfolder thereof
- Add `%dw 2.0` as the first line of your file
- Only declarations can be placed under a module

```
%dw 2.0
var xes = {
  USD: 1.0,
  EUR: 0.8,
  GBP: 0.9,
}
var adjustFor = (p,c) -> p * xes[c]
```

Using a DataWeave module

- Fully-qualified name of the module’s declaration

```
$.price dw::modules::Currency::adjustFor "CAD”
```

- Import module

```
import dw::modules::Currency

---

$.price Currency::adjustFor "AUD"
```

- Import module with an alias

```
import dw::modules::Currency as Curr

---

$.price Curr::adjustFor ”MXN"
```

Using a DataWeave module, continued

- Import all declarations

```
import * from dw::modules::Currency
---
$.price adjustFor "INR"
```

- Import selectively declarations with an alias

```
import adjustFor as adj4 from dw::modules::Currency
---
$.price adj4 "INR"
```

Walkthrough 2-3: Create and use DataWeave modules

- Create a module
- Use the module

MuleSoft®

Built-in modules

Built-in modules

- A number of modules are created and packaged with DataWeave
- The topics range from
  - Arrays
  - Objects
  - Strings
  - Trees
- Documentation is extensive and accessible through the DataWeave Reference

The Arrays module

- Contains functions that operate over arrays
  - `every`
    - Iterates over an array and applies a function to each element in the array that returns a Boolean value. If all elements return true, `every` returns true
  - `Some`
    - Iterates over an array and applies a function to each element in the array that returns a Boolean value. If at least one element returns true, `some` returns true
  - `countBy`
    - Iterates over an array and applies a function to each element in the array that returns a Boolean value. Each element that returns true increments a counter by one, `countBy` returns value of the counter
  - `sumBy`
    - Iterates over an array and applies a function to each element in the array that returns a Number value. `sumBy` returns the summation of all the values returned by the function

Walkthrough 2-4: Optional: Calculate KPIs using the Arrays Module.

- Assert all flights are Delta operated
  - Using the `every` function
- Assert the existence of full flights
  - Using the `some` function
- Calculate the number of full flights
  - Using the `countBy` function
- Sum the total number of empty seats across all flights
  - Using the `sumBy` function

MuleSoft®

Summary

Summary

- In DW you can organize your code with variables, functions, and modules
- Variables and other declarations can have ”global” and local scope
- Functions can be declared in two ways:
  - Using the `fun` syntax, which is syntactic sugar to
  - Declaring a variable and assigning to it a lambda-expression
- Full transformations can be reused by being stored in separate files first
- Modules can be created containing only declarations.
- Finally, there are ready made modules provided by DataWeave.