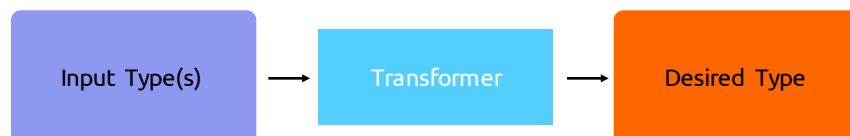




## Module 5: Transforming Data

### Transforming data using transformers

- Anypoint Studio provides a set of transformers to handle the most common data transformation scenarios
- Up to Mule 3.6, this was the main way to transform messages



## We've already used some transformers

- DOM to XML (Delta)
  - Object to String (American DB)
  - File to String (CSV)
  - Object to JSON (Salesforce)
- 
- Variable
  - Property
  - Session

3 | All contents Copyright © 2015, MuleSoft Inc.



## About the DataMapper transformer



- Introduced in Mule 3.3 (2012)
- Provided a way for users to graphically manipulate data and work with many different data formats
- Customers loved the direction, but wanted much more in terms of capabilities, performance, and ease of use
- DataWeave is the new solution for the future
- DataMapper will not be supported from Mule 4.0 on
  - There will be a migration tool later this year

4 | All contents Copyright © 2015, MuleSoft Inc.



## Transforming data using DataWeave (new in 3.7!)

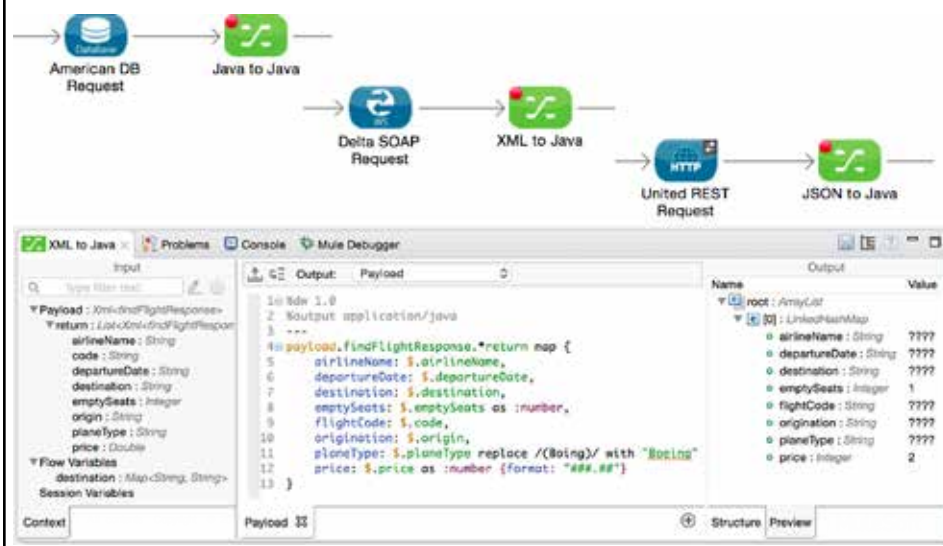


- DataWeave is a full-featured and fully native framework for querying and transforming data on Anypoint Platform
- Powered by the DataWeave data transformation language
  - A JSON-like language that's built just for data transformation use cases
- Powered by the core Mule runtime
  - Provides 5x performance vs previous approaches
- Fully integrated with Anypoint Studio and DataSense
  - Graphical interface with payload-aware development

5 | All contents Copyright © 2015, MuleSoft Inc.



## Goal



## Objectives

- In this module, you will learn:
  - About the different types of transformers
  - To use the DataWeave Transform Message component
  - To write DataWeave expressions for basic and complex XML, JSON, and Java transformations
  - To use DataWeave with data sources that have associated metadata
  - To add custom metadata to data sources

7 | All contents Copyright © 2015, MuleSoft Inc.

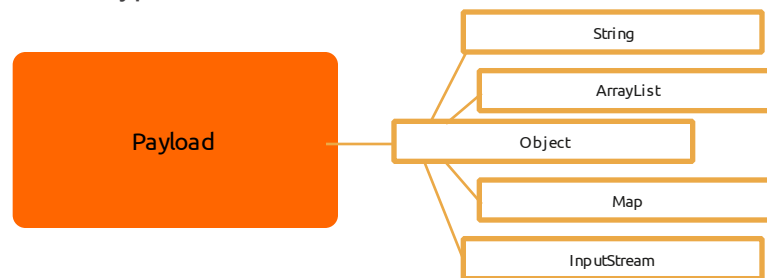


## Introducing transformers

8 | All contents Copyright © 2014, MuleSoft Inc.

## Transformers

- Transformers prepare a message to be processed through a flow by enhancing or altering the message header or message payload
- Remember payloads are Java objects and can be any Java type



9 | All contents Copyright © 2015, MuleSoft Inc.



## Transformer categories

- Java object transformers
  - Transform a Java object into another Java object or some other data type (like HTTP request) or vice versa
- Message and variable transformers
  - Do not modify the message directly, but make special info available as a message makes its way through a Mule app
- Content transformers
  - Modify messages by adding to, deleting from, or converting a message payload (or a message header)
- Script transformers
  - Use Groovy, JavaScript, Python, or Ruby to perform the transformation

10 | All contents Copyright © 2015, MuleSoft Inc.



## Parse Template transformer



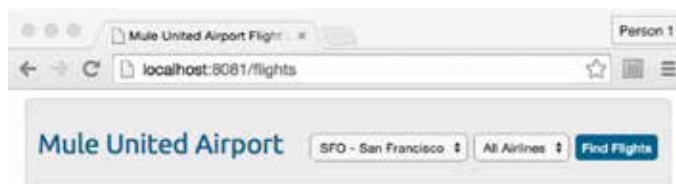
- A content transformer
- Loads the content of an external file into a Mule flow
- Commonly used to return customized HTML responses
  - The file can have MEL expressions in it that are evaluated

11 | All contents Copyright © 2015, MuleSoft Inc.



## Walkthrough 5-1: Load external content into a message

- Create a flow that receives GET requests at <http://localhost:8081/flights>
- Use the Parse Template transformer to load the content of an HTML file into a flow
- Examine the HTML code and determine what data it sends where



12 | All contents Copyright © 2015, MuleSoft Inc.



## Java object transformers

- Transform a Java object into another Java object or some other data type or vice versa
- Some can simply be dropped in (like the ones we used so far)
- Others require configuration using 3<sup>rd</sup> party libraries
  - JAXB
  - Jackson
  - org.w3c.dom

13 | All contents Copyright © 2015, MuleSoft Inc.



## Complex transformations

- Up to Mule 3.6, if a transformer did not exist for your specific needs you would
  - Chain transformers
  - Create a custom Java transformer
  - Use a Script transformer to write the transformation in Groovy, JavaScript, Python, or Ruby

14 | All contents Copyright © 2015, MuleSoft Inc.



## Using DataWeave for all transformations



- For Mule 3.7, you can use the new DataWeave framework for all your transformations
  - From simple to complex
  - No longer need to use most other transformers unless you want to use specific Java frameworks
    - Like JAXB, Jackson, org.w3c.dom
    - To integrate with existing code bases or leverage existing skill sets

15 | All contents Copyright © 2015, MuleSoft Inc.



## Introducing DataWeave

16 | All contents Copyright © 2014, MuleSoft Inc.



## Introducing DataWeave (again)



- DataWeave is a full-featured and fully native framework for querying and transforming data on Anypoint Platform
- Powered by the DataWeave data transformation language
  - A JSON-like language that's built just for data transformation use cases
- Powered by the core Mule runtime
  - Provides 5x performance vs previous approaches
- Fully integrated with Anypoint Studio and DataSense
  - Graphical interface with payload-aware development

17 | All contents Copyright © 2015, MuleSoft Inc.



## 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 out of the box
  - Extensible for new formats via an API
  - Excel support coming later this year
- More elegant and re-usable than custom code
  - Data transformations can be stored in external DWL files and used across applications

18 | All contents Copyright © 2015, MuleSoft Inc.



## DataWeave data transformation use cases



- DataWeave was purposefully built to make it easy to write simple to complex transformations
  - Simple 1-to-1 mappings
  - Transforming hierarchical data models
  - De-duplication of data
  - Filtering data
  - Grouping and partitioning data
  - Joining data across multiple data sources
  - Streaming inbound and outbound data

19 | All contents Copyright © 2015, MuleSoft Inc.



## DataWeave under the hood



- Underneath, DataWeave includes a connectivity layer and engine that is fundamentally different from other transformation technologies
- It contains a data access layer that indexes content and accesses the binary directly, without costly conversions
  - Enables larger than memory payloads
  - Random access to input documents
  - Very high performance

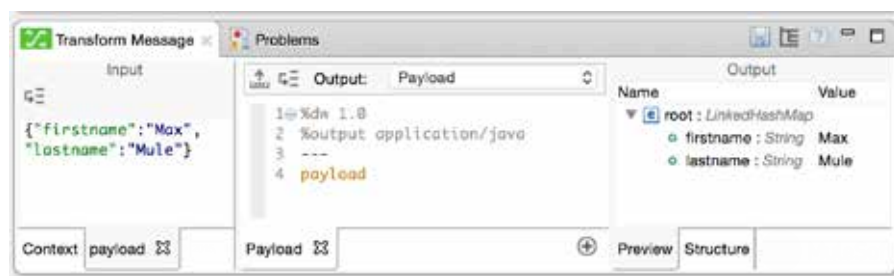
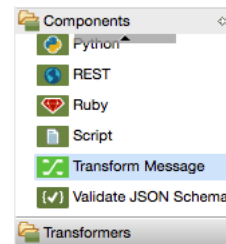
20 | All contents Copyright © 2015, MuleSoft Inc.



## DataWeave integration with Anypoint Studio



- The DataWeave Transform Message component provides an Anypoint Studio interface
- The Properties view has Input, Transform, and Output sections



## DataWeave integration with DataSense



- DataWeave is fully integrated with DataSense allowing payload-aware development
- Metadata from connectors, schemas and sample documents can be used to more easily build transformations
  - In the Input section of the Properties view
    - Any metadata of incoming message displayed
  - In the Transform section
    - Auto-completion of code
    - Auto-scaffolding of transforms
  - In the Output section
    - Any metadata for outbound message displayed
    - Live transformation previews

# Writing DataWeave expressions

23 | All contents Copyright © 2014, MuleSoft Inc.

## Example DataWeave 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

24 | All contents Copyright © 2015, MuleSoft Inc.



## 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**

25 | All contents Copyright © 2015, MuleSoft Inc.



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

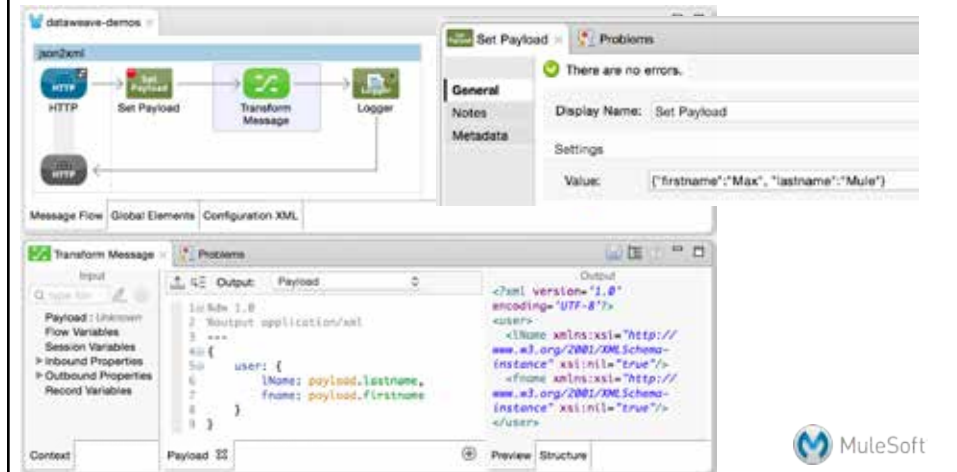
```
%dw 1.0
%output application/xml
---
{
  a: payload
}
```

26 | All contents Copyright © 2015, MuleSoft Inc.

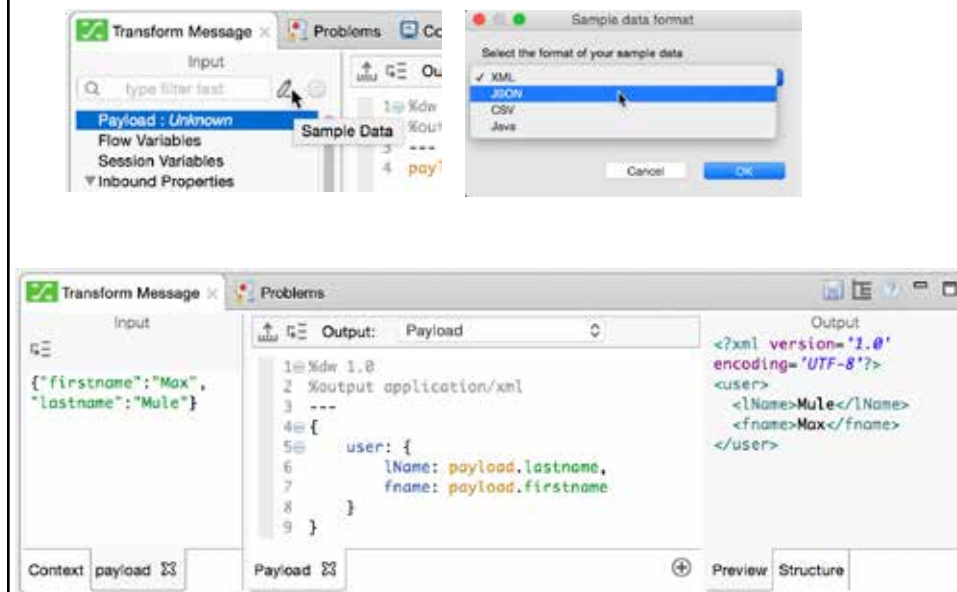


## Previewing transformations in Anypoint Studio

- As you write a DataWeave expression, a live preview of output will be shown
  - You will not see actual values unless you add sample data



## Adding sample data to get live transformation previews



## 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
  - Otherwise, you can set it
    - The transformers have a mimeType property



29 | All contents Copyright © 2015, MuleSoft Inc.



## Where is the code?

- By default, it is placed inline

```
<flow name="json2xml">
  <http:listener config-ref="HTTP-Listener_Configuration"
  <set-payload value="&quot;first&name=&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:transform-message>
  <logger level="INFO" doc:name="Logger"/>
</flow>
```

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



## Walkthrough 5-2: Write your first DataWeave transformation

- Create a flow that receives POST requests at <http://localhost:8081/flights>
- Use the DataWeave Transform Message component
- Add sample data and use live preview
- Transform the form data from JSON to a Java object

The screenshot shows the MuleSoft IDE interface. At the top, a flow named 'getFlightInfo' is visible, consisting of an HTTP listener, a Transform Message component, and a Logger. Below this, the 'Transform Message' component is configured with the following input payload:

```
["destination":"SFO",
"airline":"united"]
```

The 'Output' tab shows the transformed payload as a Java object:

```
1= flow 1,0
2= output application/java
3= ---
4= payload
```

The 'Preview' tab shows the output structure:

Name	Value
root : LinkedHashMap	
destination : String	SFO
airline : String	united

The MuleSoft logo is visible in the bottom right corner.

## Reusing transformations

- To place it in an external file, click the Data Source button
  - Transform is saved in a DWL file
  - DWL files are stored in src/main/resources

The screenshot shows the 'Data Source' button in the component palette, which is used to save the current transformation as a DataWeave Language (DWL) file.

The 'Script Data Source' dialog box is shown, allowing the user to specify whether the script should be stored in a separate file. The 'On File' option is selected, and the file path is set to 'src/main/resources/user\_transform.dwl'.

- 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>
```

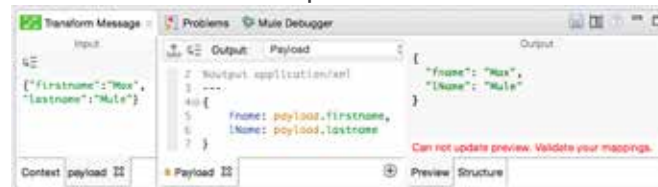
©Soft



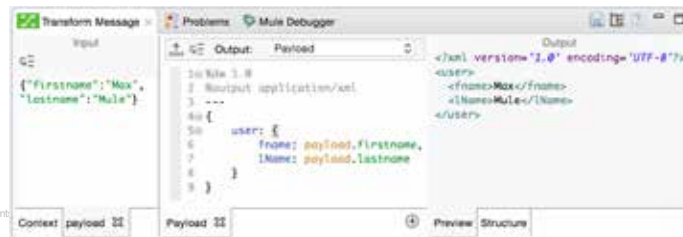
## Writing expressions for XML output



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



- This will work



33 | All contents

MuleSoft

## Specifying attributes for XML output



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



34 | All contents Copyright © 2015, MuleSoft Inc.



## 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
<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" }

- Note: Be sure to update to Anypoint Studio 5.2.1 or later
  - Live preview for XML sample input does not work in 5.2.0

35 | All contents Copyright © 2015, MuleSoft Inc.



## Referencing message variables



- So far, we referenced payload
- You can also reference
  - flowVars
  - inboundProperties
  - outboundProperties
- This is not MEL!
  - Do not preface these values by “message.” or use #[]

```
%dw 1.0
%output application/xml
---
{
  a: flowVars.userName
}
```

36 | All contents Copyright © 2015, MuleSoft Inc.



## Creating multiple transformations

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

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

Use Add New Transformation button

Get multiple transformations

37 | All contents Copyright © 2015, MuleSoft Inc.

MuleSoft

## Walkthrough 5-3: Transform basic Java, JSON, and XML data structures

- Write transformations to store data in multiple outputs as different types of data
- Create a 2<sup>nd</sup> transformation to store destination in a flow variable
- Create a 3<sup>rd</sup> transformation to output data as JSON
- Create a 4<sup>th</sup> transformation to output data as XML

Transform Message

Problems Mule Debugger

Output: Flow Variable Name: xml

1: Xdw 1.0

2: Xoutput application/xml

3: ---

4: data: {

5: hub: "MIA",

6: flight @(airline: payload.airline): {

7: code: payload.destination

8: }

9: }

Output

<?xml version="1.0" encoding="UTF-8"?>

<data>

<hub>MIA</hub>

<flight airline="united">

<code>SFO</code>

</flight>

</data>

Context payload

Payload destination json xml

Preview Structure

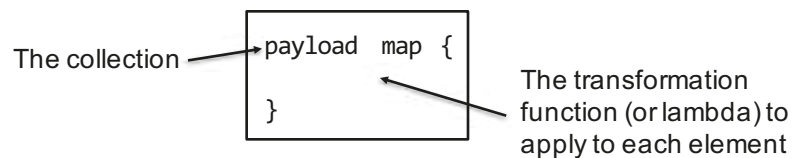
# Transforming complex data structures with DataWeave

39 | All contents Copyright © 2014, MuleSoft Inc.

## 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

40 | All contents Copyright © 2015, MuleSoft Inc.



## The transformation function

- 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" }} ] }

41 | All contents Copyright © 2015, MuleSoft Inc.



## 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
[{"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" } } ]

42 | All contents Copyright © 2015, MuleSoft Inc.



## 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
[{"firstname":"Max", "lastname":"Mule"}, {"firstname":"Molly", "lastname":"Mule"}]	<pre>%dw 1.0 %output application/xml --- users: payload map {   fname: \$.firstname,   lname: \$.lastname }</pre>	Cannot coerce an array to an object (starting with 3.7.1)
	<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>

43 | All contents Copyright © 2015, MuleSoft Inc.



## Writing expressions for XML output (cont)

Input	Transform	Output
[{"firstname":"Max", "last Name":"Mule"}, {"firstname":"Molly", "lastName":"Mule"}]	<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;?xml version='1.0' encoding='UTF-8'?&gt; &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>

44 | All contents Copyright © 2015, MuleSoft Inc.

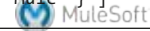


## Writing expressions for XML input

- 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;Mule&lt;/lastname&gt;   &lt;/user&gt; &lt;/users&gt;</pre>	payload	{ "users": { "user": { "lastname": "Mule" } } }
	payload.users	{ "user": { "lastname": "Mule" } }
	payload.users.user	{ "lastName": "Mule" }
	payload.users.*user	[ { "lastName": "Mule" }, { "lastName": "Mule" } ]
	payload.users.*user map { fname: \$.@firstname, lname: \$.lastName }	[ { "fname": "Max", "lname": "Mule" }, { "fname": "Molly", "lname": "Mule" } ]

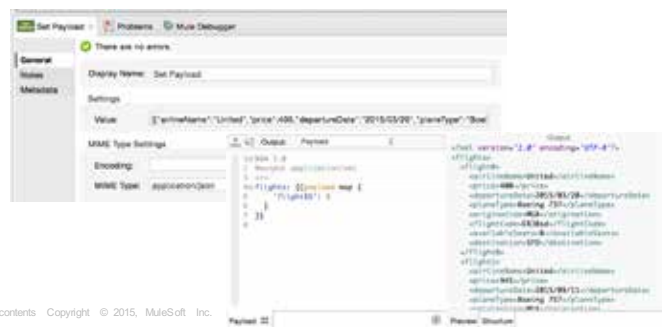
45 | All contents Copyright © 2015, MuleSoft Inc.



## Walkthrough 5-4: Transform complex data structures

- Create a new flow that receives GET requests at <http://localhost:8081/static>
- Transform a JSON array of objects to Java, JSON, and XML
- Explicitly set the MIME type of the data to be transformed
- Transform XML with repeated elements to XML and JSON

*Note: You will work with CSV data in the Processing Records module*



46 | All contents Copyright © 2015, MuleSoft Inc.



# Using DataWeave operators

47 | All contents Copyright © 2014, MuleSoft Inc.

## DataWeave reference documentation

- <https://developer.mulesoft.com/docs/dataweave>

From an incoming Mule Message

4. Operators

- 4.1. Map
- 4.2. Map Object
- 4.3. Pluck
- 4.4. Filter
- 4.5. Remove
- 4.6. Default
- 4.7. When / Otherwise
- 4.8. Unless / Otherwise
- 4.9. AND
- 4.10. OR
- 4.11. Concat
- 4.12. AS (Type Correlation)
- 4.13. Flatten
- 4.14. Size Of
- 4.15. Push
- 4.16. Reduce
- 4.17. Join By
- 4.18. Split By
- 4.19. Order By
- 4.20. Group By
- 4.21. Distinct By

## Weave Documentation

MuleSoft Inc.

[IMPORTANT] You're viewing documentation written using our new documentation platform. The following content is official MuleSoft documentation. If you have any feedback, please contact [documentation@mulesoft.com](mailto:documentation@mulesoft.com).

### 1. Introduction

The DataWeave Language is a powerful template engine that allows you to transform data to and from any kind of format (XML, CSV, JSON, Pojos, Maps, etc).

#### 1.1. Getting started:

In order to show the power of DataWeave, here are a few examples to get started.

48 | All cor



## Formatting operators



Input	Transform	Output
	%dw 1.0 %output application/xml ---	
{ "name": "max_mule" }	n: upper payload.firstname	<n>MAX_MULE</n>
	n: lower payload.firstname	<n>Max_mule</n>
	n: camelize payload.name	<n>maxMule</n>
	n: capitalize payload.name	<n>Max Mule</n>
	n: dasherize payload.name	<n>max-mule</n>
	n: pluralize payload.name	<n>max-mules</n>
	n: upper (dasherize payload.name)	<n>MAX-MULE</n>
{ "name": "max mules" }	n: singularize payload.name	<n>max mule</n>
	n: underscore payload.name	<n>max_mules</n>
{ "place": 2 }	n: ordinalize payload.place	<n>2nd</n>

49 | All contents Copyright © 2015, MuleSoft Inc.



## Using the as operator for type coercion

```
price: payload.price as :number
```

- Defined types include
  - :string
  - :number
  - :boolean
  - :object
  - :array
  - :date, :time, :timezone, :datetime, :localdatetime, :period
  - :regex

50 | All contents Copyright © 2015, MuleSoft Inc.



## Specifying custom data types

- Specify inline

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

- Assign a custom name with the type directive
  - Name has to be all lowercase letters
    - No special characters, uppercase letters, or numbers

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

## Using format patterns

- Use metadata **format** key to format numbers and dates

- Inline

```
tax: (tax * 100) as :number {format: "##.##" } ++ "%"
someDate as :datetime {format: "yyyyMMddHHmm"}
```

For pattern letters, see Java DateTimeFormatter class API  
<https://docs.oracle.com/javase/8/docs/api/java/time/format/DateTimeFormatter.html>

- With custom type

```
%type currency = :number {format: "##"}
price: $.price as :currency
```

## Conditional logic operators



- In expressions, you can use `==`, `!=`, or `~=` (equal regardless of type)

Input	Transform	Output
	<code>%dw 1.0</code> <code>%output application/xml</code> <code>---</code>	
<code>{"firstname": "Max", "lastname": "Mule"}</code>	<code>n: payload.nickname default payload.firstname</code>	<code>&lt;n&gt;Max&lt;/n&gt;</code>
<code>{"firstname": "Max", "lastname": "Mule", "nickname": ""}</code>	<code>n: payload.nickname</code> <code>when payload.nickname != ""</code> <code>otherwise payload.firstname</code>	<code>&lt;n&gt;Max&lt;/n&gt;</code>
	<code>n: payload.firstname</code> <code>unless payload.nickname != ""</code> <code>otherwise payload.nickname</code>	<code>&lt;n&gt;Max&lt;/n&gt;</code>
	<code>n: payload.firstname</code> <code>unless payload.nickname != "" or payload.firstname != ""</code> <code>otherwise payload.nickname</code>	<code>&lt;n&gt;&lt;/n&gt;</code>
	<code>n: payload.lastname</code> <code>unless payload.nickname != "" and payload.firstname != ""</code> <code>otherwise payload.nickname</code>	<code>&lt;n&gt;Max&lt;/n&gt;</code>

53 | All contents Copyright © 2015, MuleSoft Inc.



## Math operators



- `+`
- `-`
- `*`
- `/`
- `max`: returns the highest number in an array or object
- `min`: returns the lowest number in an array or object
- `sizeOf`: returns number of elements in an array

54 | All contents Copyright © 2015, MuleSoft Inc.



## Additional operators



- concat  
n: payload.firstname ++ " " ++ payload.lastname
- orderBy
- distinctBy
- groupBy
- replace
- matches
- regex
- More...

55 | All contents Copyright © 2015, MuleSoft Inc.



## Walkthrough 5-5: Use DataWeave operators

- Format strings, dates, and numbers
- Convert data types
- Replace data values using pattern matching
- Order data, filter data, and remove duplicate data
- Define and use custom data types
- Transform objects to POJOs

The screenshot shows the MuleSoft IDE with a DataWeave script in the left pane and its output in the right pane. The script processes a payload containing flight information and formats it into a structured object.

```

1 payload.payloadName = "MuleSoft";
2 payload.payloadName = payload.payloadName.toUpperCase();
3 payload.payloadName = payload.payloadName.toLowerCase();
4 payload.payloadName = payload.payloadName.replaceAll(" ", "");
5 payload.payloadName = payload.payloadName.replaceAll(" ", "");
6 payload.payloadName = payload.payloadName.replaceAll(" ", "");
7 payload.payloadName = payload.payloadName.replaceAll(" ", "");
8 payload.payloadName = payload.payloadName.replaceAll(" ", "");
9 payload.payloadName = payload.payloadName.replaceAll(" ", "");
10 payload.payloadName = payload.payloadName.replaceAll(" ", "");
11 payload.payloadName = payload.payloadName.replaceAll(" ", "");
12 payload.payloadName = payload.payloadName.replaceAll(" ", "");
13 payload.payloadName = payload.payloadName.replaceAll(" ", "");
14 payload.payloadName = payload.payloadName.replaceAll(" ", "");
15 payload.payloadName = payload.payloadName.replaceAll(" ", "");
16 payload.payloadName = payload.payloadName.replaceAll(" ", "");
17 payload.payloadName = payload.payloadName.replaceAll(" ", "");
18 payload.payloadName = payload.payloadName.replaceAll(" ", "");
19 payload.payloadName = payload.payloadName.replaceAll(" ", "");
20 payload.payloadName = payload.payloadName.replaceAll(" ", "");
21 payload.payloadName = payload.payloadName.replaceAll(" ", "");
22 payload.payloadName = payload.payloadName.replaceAll(" ", "");
23 payload.payloadName = payload.payloadName.replaceAll(" ", "");
24 payload.payloadName = payload.payloadName.replaceAll(" ", "");
25 payload.payloadName = payload.payloadName.replaceAll(" ", "");
26 payload.payloadName = payload.payloadName.replaceAll(" ", "");
27 payload.payloadName = payload.payloadName.replaceAll(" ", "");
28 payload.payloadName = payload.payloadName.replaceAll(" ", "");
29 payload.payloadName = payload.payloadName.replaceAll(" ", "");
30 payload.payloadName = payload.payloadName.replaceAll(" ", "");
31 payload.payloadName = payload.payloadName.replaceAll(" ", "");
32 payload.payloadName = payload.payloadName.replaceAll(" ", "");
33 payload.payloadName = payload.payloadName.replaceAll(" ", "");
34 payload.payloadName = payload.payloadName.replaceAll(" ", "");
35 payload.payloadName = payload.payloadName.replaceAll(" ", "");
36 payload.payloadName = payload.payloadName.replaceAll(" ", "");
37 payload.payloadName = payload.payloadName.replaceAll(" ", "");
38 payload.payloadName = payload.payloadName.replaceAll(" ", "");
39 payload.payloadName = payload.payloadName.replaceAll(" ", "");
40 payload.payloadName = payload.payloadName.replaceAll(" ", "");
41 payload.payloadName = payload.payloadName.replaceAll(" ", "");
42 payload.payloadName = payload.payloadName.replaceAll(" ", "");
43 payload.payloadName = payload.payloadName.replaceAll(" ", "");
44 payload.payloadName = payload.payloadName.replaceAll(" ", "");
45 payload.payloadName = payload.payloadName.replaceAll(" ", "");
46 payload.payloadName = payload.payloadName.replaceAll(" ", "");
47 payload.payloadName = payload.payloadName.replaceAll(" ", "");
48 payload.payloadName = payload.payloadName.replaceAll(" ", "");
49 payload.payloadName = payload.payloadName.replaceAll(" ", "");
50 payload.payloadName = payload.payloadName.replaceAll(" ", "");
51 payload.payloadName = payload.payloadName.replaceAll(" ", "");
52 payload.payloadName = payload.payloadName.replaceAll(" ", "");
53 payload.payloadName = payload.payloadName.replaceAll(" ", "");
54 payload.payloadName = payload.payloadName.replaceAll(" ", "");
55 payload.payloadName = payload.payloadName.replaceAll(" ", "");
56 payload.payloadName = payload.payloadName.replaceAll(" ", "");
57 payload.payloadName = payload.payloadName.replaceAll(" ", "");
58 payload.payloadName = payload.payloadName.replaceAll(" ", "");
59 payload.payloadName = payload.payloadName.replaceAll(" ", "");
60 payload.payloadName = payload.payloadName.replaceAll(" ", "");
61 payload.payloadName = payload.payloadName.replaceAll(" ", "");
62 payload.payloadName = payload.payloadName.replaceAll(" ", "");
63 payload.payloadName = payload.payloadName.replaceAll(" ", "");
64 payload.payloadName = payload.payloadName.replaceAll(" ", "");
65 payload.payloadName = payload.payloadName.replaceAll(" ", "");
66 payload.payloadName = payload.payloadName.replaceAll(" ", "");
67 payload.payloadName = payload.payloadName.replaceAll(" ", "");
68 payload.payloadName = payload.payloadName.replaceAll(" ", "");
69 payload.payloadName = payload.payloadName.replaceAll(" ", "");
70 payload.payloadName = payload.payloadName.replaceAll(" ", "");
71 payload.payloadName = payload.payloadName.replaceAll(" ", "");
72 payload.payloadName = payload.payloadName.replaceAll(" ", "");
73 payload.payloadName = payload.payloadName.replaceAll(" ", "");
74 payload.payloadName = payload.payloadName.replaceAll(" ", "");
75 payload.payloadName = payload.payloadName.replaceAll(" ", "");
76 payload.payloadName = payload.payloadName.replaceAll(" ", "");
77 payload.payloadName = payload.payloadName.replaceAll(" ", "");
78 payload.payloadName = payload.payloadName.replaceAll(" ", "");
79 payload.payloadName = payload.payloadName.replaceAll(" ", "");
80 payload.payloadName = payload.payloadName.replaceAll(" ", "");
81 payload.payloadName = payload.payloadName.replaceAll(" ", "");
82 payload.payloadName = payload.payloadName.replaceAll(" ", "");
83 payload.payloadName = payload.payloadName.replaceAll(" ", "");
84 payload.payloadName = payload.payloadName.replaceAll(" ", "");
85 payload.payloadName = payload.payloadName.replaceAll(" ", "");
86 payload.payloadName = payload.payloadName.replaceAll(" ", "");
87 payload.payloadName = payload.payloadName.replaceAll(" ", "");
88 payload.payloadName = payload.payloadName.replaceAll(" ", "");
89 payload.payloadName = payload.payloadName.replaceAll(" ", "");
90 payload.payloadName = payload.payloadName.replaceAll(" ", "");
91 payload.payloadName = payload.payloadName.replaceAll(" ", "");
92 payload.payloadName = payload.payloadName.replaceAll(" ", "");
93 payload.payloadName = payload.payloadName.replaceAll(" ", "");
94 payload.payloadName = payload.payloadName.replaceAll(" ", "");
95 payload.payloadName = payload.payloadName.replaceAll(" ", "");
96 payload.payloadName = payload.payloadName.replaceAll(" ", "");
97 payload.payloadName = payload.payloadName.replaceAll(" ", "");
98 payload.payloadName = payload.payloadName.replaceAll(" ", "");
99 payload.payloadName = payload.payloadName.replaceAll(" ", "");
100 payload.payloadName = payload.payloadName.replaceAll(" ", "");

```

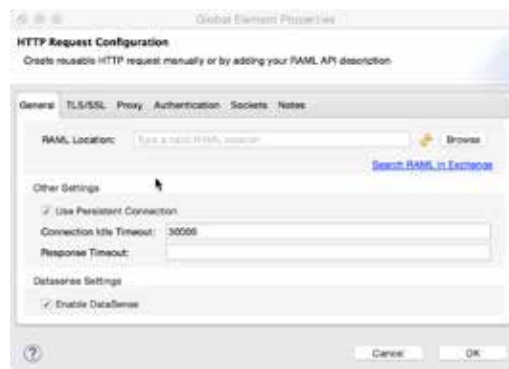
The output pane shows the result of the script, which is a structured object with fields like payloadName, payloadType, and payloadValue.

## Using DataWeave to transform data structures that have associated metadata

57 | All contents Copyright © 2014, MuleSoft Inc.

### DataSense and metadata

- Many connectors can be DataSense enabled



58 | All contents Copyright © 2015, MuleSoft Inc.

## Transformations and metadata

- If message input and/or output has metadata, Anypoint Studio will build an initial scaffolding for the transformation based on it
  - Based on the processors on either side of the transformer
  - The scaffolding is just the starting transformation code automatically written based on metadata
  - You may need to modify this a little or a lot depending upon what the metadata is and what you want to accomplish
- For this reason, it is best to add processors first and then the Transform Message component

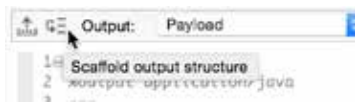
59 | All contents Copyright © 2015, MuleSoft Inc.



## Updating the scaffolding

- If you add new processors upstream or downstream or add metadata to existing ones, you can update the scaffolding

- Refresh metadata and/or recreate scaffolding



- To recreate scaffolding from metadata, click Scaffold output structure button
  - Deletes existing DataWeave code and re-scaffolds output

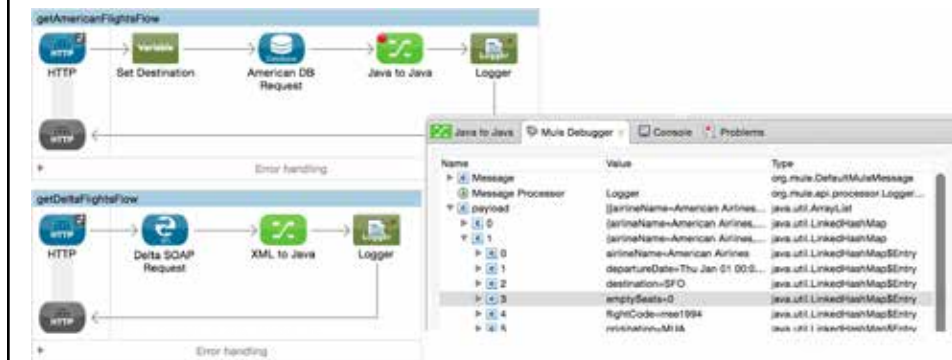
60 | All contents Copyright © 2015, MuleSoft Inc.



## Walkthrough 5-6: Transform data structures that have associated metadata



- Use DataWeave to transform American flight data from a collection of Java objects to one with a different data structure
- Use DataWeave to transform the Delta flight data from XML to a collection of Java objects



Using DataWeave to pass arguments to SOAP web services

## Using DataWeave to specify SOAP web service input arguments



- When you drag-and-drop a DataWeave message transformer before a Web Service Consumer that has input arguments, a scaffold for populating the arguments will be created automatically



## Walkthrough 5-7: Pass arguments to a SOAP web service



- Return the flights for a specific destination instead of all the flights
- Change the web service operation invoked to one that requires a destination as an input argument
- Use DataWeave to pass an argument to a web service operation
- Create a variable to set the destination to a dynamic query parameter value





## Using DataWeave to transform data structures that need custom metadata

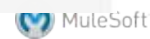
65 | All contents Copyright © 2014, MuleSoft Inc.

### A more metadata aware Anypoint Studio and Mule runtime

- New in 3.7, Mule now tracks the payload type internally so that metadata can be used during transformations
- In Anypoint Studio, you can now provide design time metadata to message processors and declare the type of the payload
  - Provides content-assist capabilities
  - Gives you visibility into your payload everywhere



66 | All contents Copyright © 2015, MuleSoft Inc.



## Adding metadata

- Specify what the metadata is for



- Specify the data type: JSON, XML, CSV, JAVA



67 | All contents Copyright © 2015, MuleSoft Inc.



## Adding metadata

- For XML and JSON
  - Provide schema or example file
- For CSV
- Specify the data type: JSON, XML, CSV, JAVA



68 | All contents Copyright © 2015, MuleSoft Inc.



## Walkthrough 5-8: Transform a data source to which you add custom metadata



- Add custom metadata to an HTTP Request endpoint
- Use DataWeave to transform the United flight data from JSON to a collection of Java objects



## Using DataWeave with CSV data

- You will apply this technique to transform CSV data in a later module, Processing Records
  - Add metadata to File endpoint using example CSV file

## Using dw() to query data



- The DataWeave universal language for data access can not only be used for transformation, but also for querying data throughout your flow
- Using the dw() function, you can quickly query data and use it to log information from payloads, route data, or extract it for message enrichment

71 | All contents Copyright © 2015, MuleSoft Inc.



## Summary

72 | All contents Copyright © 2014, MuleSoft Inc.

## Summary

- In this module, you learned about the different types of transformers and the DataWeave framework
- There are Java object, message and variable, content, and script transformers
- The Parse Template transformer loads the content of an external file (that can have MEL expressions)
- The DataWeave Transform Message component can be used in place of most other transformers
- DataWeave a full-featured and fully native framework for querying and transforming data on Anypoint Platform
- DataWeave is new in Mule 3.7

73 | All contents Copyright © 2015, MuleSoft Inc.



## Summary



- For the DataWeave component, you set the output type and a transformation expression using the DataWeave data transformation language
  - A JSON-like language built just for data transformation use cases
- DataWeave transformations are fast and reusable
- DataWeave is fully integrated with Anypoint Studio and DataSense
  - There is a graphical interface that is aware of associated metadata for input and output structures
  - Easy to use with data sources that have associated metadata

74 | All contents Copyright © 2015, MuleSoft Inc.

