



MuleSoft®

PART 2: Building Applications with Anypoint Studio

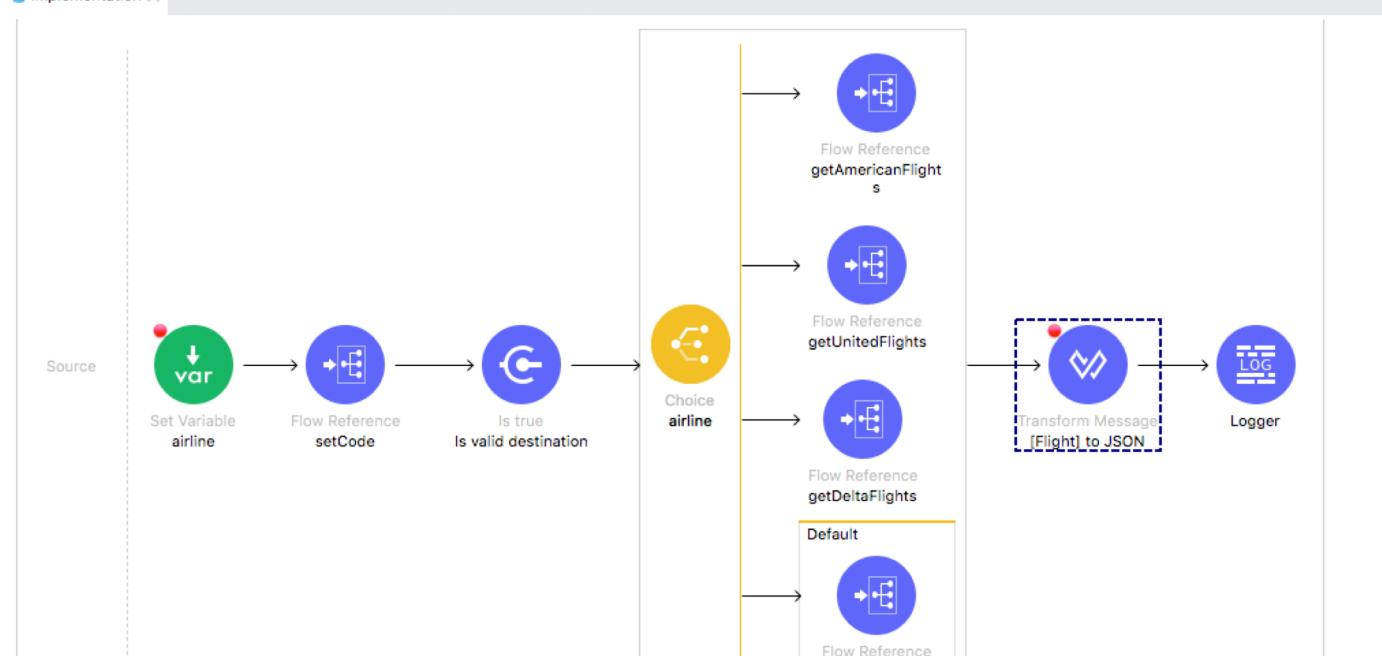


Goal

Mule Debugger X

| Name | Value | Type |
|---|--|---|
| (DataType) | CollectionDataType(type=java.util.ArrayList, itemType=SimpleDataType{type=java.lang.String, name=String, value=UTF-8}) | org.mule.runtime.core.internal.metadata.DefaultCollectionDataType |
| Encoding | UTF-8 | java.lang.String |
| Message | org.mule.runtime.core.internal.message.DefaultMessageBuilder\$MessageImpl | java.util.ArrayList |
| Payload (mimeType="application/java; charset=UTF-8", encoding="UTF-8") size = 9 | com.mulesoft.training.Flight@4d4622c7 com.mulesoft.training.Flight@536cbaa3 com.mulesoft.training.Flight@22eb4a4d com.mulesoft.training.Flight@7c6b872d | com.mulesoft.training.Flight |
| 0 | com.mulesoft.training.Flight@4d4622c7 | com.mulesoft.training.Flight |
| 1 | com.mulesoft.training.Flight@536cbaa3 | com.mulesoft.training.Flight |
| 2 | com.mulesoft.training.Flight@22eb4a4d | com.mulesoft.training.Flight |
| 3 | com.mulesoft.training.Flight@7c6b872d | com.mulesoft.training.Flight |

implementation X



The diagram illustrates a Mule ESB flow. It starts with a 'Source' (green circle with a downward arrow) followed by a 'Set Variable' node ('var', green circle with a downward arrow). This is followed by a 'Flow Reference' node ('setCode', blue circle with a right-pointing arrow) and a 'Decision' node ('Is true', blue circle with a right-pointing arrow). The 'Decision' node leads to a 'Choice' node ('airline', yellow circle with a right-pointing arrow). From the 'Choice' node, four paths emerge: 'Flow Reference getAmericanFlight's', 'Flow Reference getUnitedFlights', 'Flow Reference getDeltaFlights', and a 'Default' path. The 'Flow Reference getUnitedFlights' path leads to a 'Transform Message' node ('[Flight] to JSON', blue circle with a right-pointing arrow), which then connects to a 'Logger' node ('LOG', blue circle with a right-pointing arrow). The 'Mule Palettes' panel on the right lists various components like Logger (Core), Listener (HTTP), Request (HTTP), Transform Message (Core), Set Payload (Core), Flow Reference (Core), and Choice (Core).

Message Flow Global Elements Configuration XML

Console X Problems api APIkit Consoles (apdev-flights-ws)

```
apdev-flights-ws [Mule Applications] Mule Server 4.1.1 EE
*****
default * DEPLOYED *
*****
----- + APPLICATION + ----- * - + DOMAIN + - - * - - + STATUS + - - *
***** pdev-flights-ws * default * DEPLOYED * *****
```

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



- Debug Mule applications
- Read and write event payloads, attributes, and variables using the DataWeave Expression Language
- Structure Mule applications using flows, subflows, asynchronous queues, properties files, and configuration files
- Call RESTful and SOAP web services
- Route and validate events and handle messaging errors
- Write DataWeave scripts for transformations



MuleSoft®

Module 6: Accessing and Modifying Mule Events



Goal

Mule Debugger

| Name | Value | Type |
|-----------------------------|--|--|
| ► e Attributes | org.mule.extension.http.api.HttpRequestAttributes | org.mule.extension.http.api.HttpRequestAttributes |
| @ Component Path | fundamentalsFlow/processors/2 | org.mule.runtime.dsl.api.component.config.DefaultComponentLocation |
| ► e DataType | SimpleDataType{type=java.lang.String, mimeType='/*/*'} | org.mule.runtime.core.internal.metadata.SimpleDataType |
| @ Message | | org.mule.runtime.core.internal.message.DefaultMessageBuilder\$... |
| @ Payload (mimeType="/*/*") | Hello | java.lang.String |
| ► e Variables | size = 1 | java.util.HashMap |
| ► e 0 | code=SFO | java.util.HashMap\$Node |

size = 1

fundamentals

fundamentalsFlow

```
graph LR; Listener[HTTP Listener] --> SetPayload[Set Payload]; SetPayload --> SetVariable[Set Variable]; SetVariable --> Request["Request<br>HTTP Request"]; Request --> Logger[Logger];
```

Watches

| Name | Value | Type |
|------|-------|------|
| | | |

All contents © Mule

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

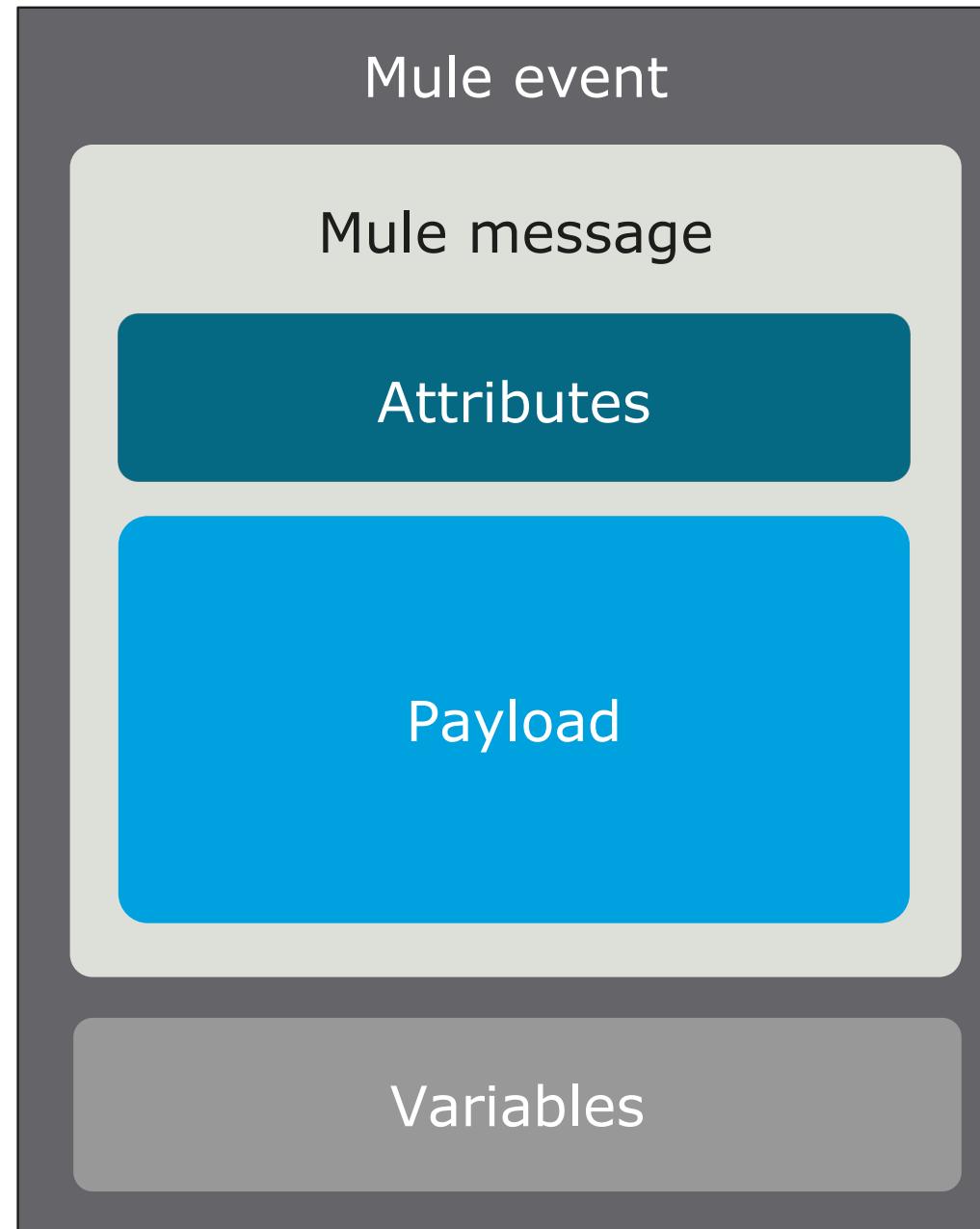


- Log event data
- Debug Mule applications
- Read and write event properties
- Write expressions with the DataWeave expression language
- Create variables

Viewing information about Mule 4 events

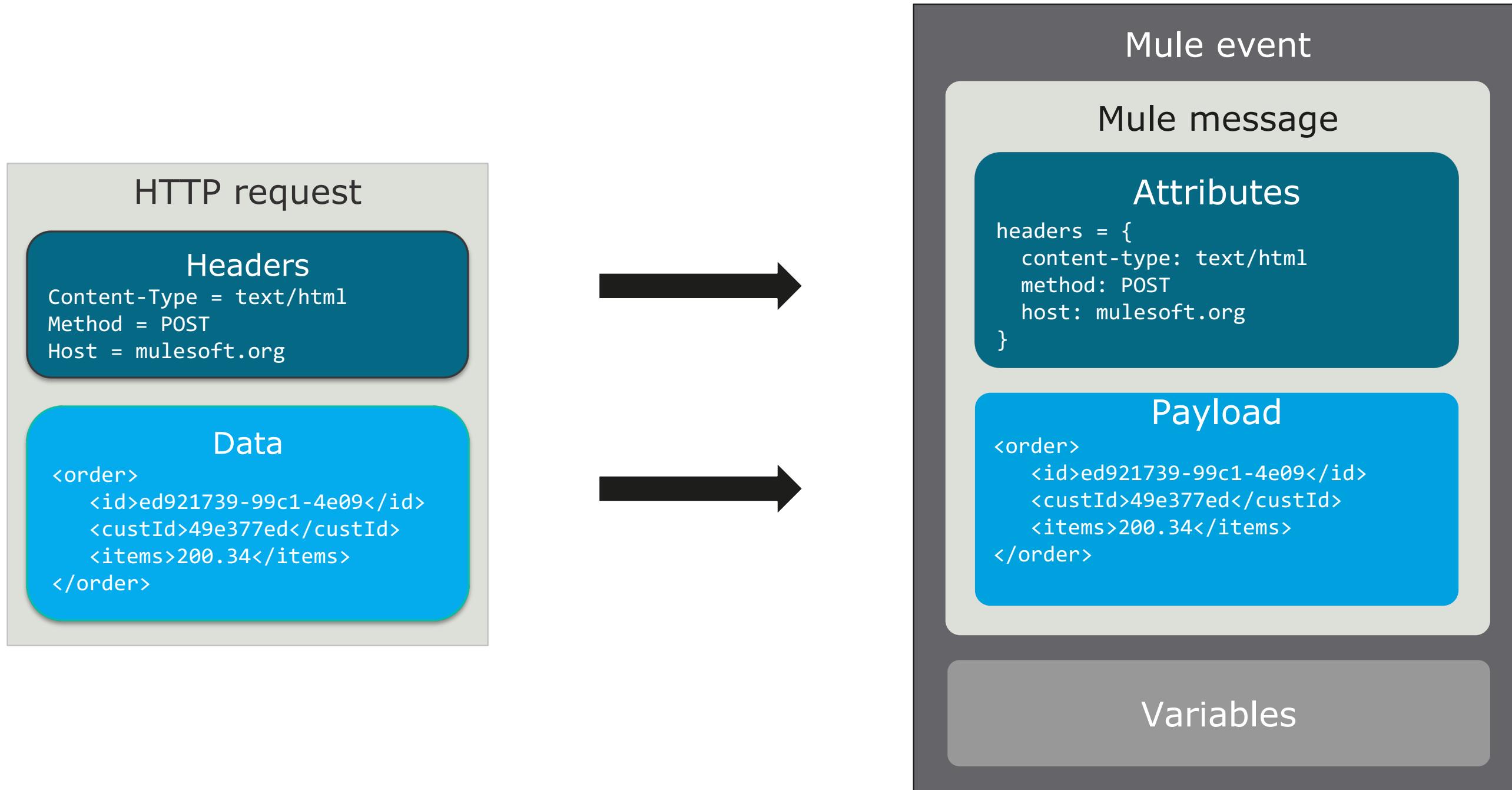


Reviewing the structure of Mule 4 events



- The data that passes through flows in the app
- Metadata contained in the message header
- The core info of the message - the data the app processes
- Metadata for the Mule event - can be defined and referenced in the app processing the event

Event data populated from an HTTP request



- At **design time** in Anypoint Studio
 - What did we see so far in the course and where?
- At **run time**
 - What did we see so far in the course and where?

Viewing event data at design time

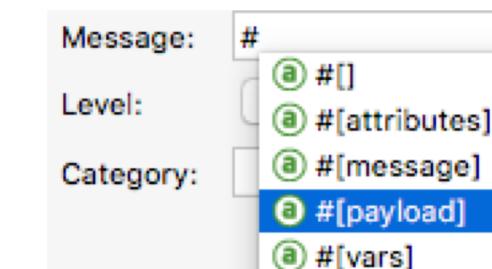
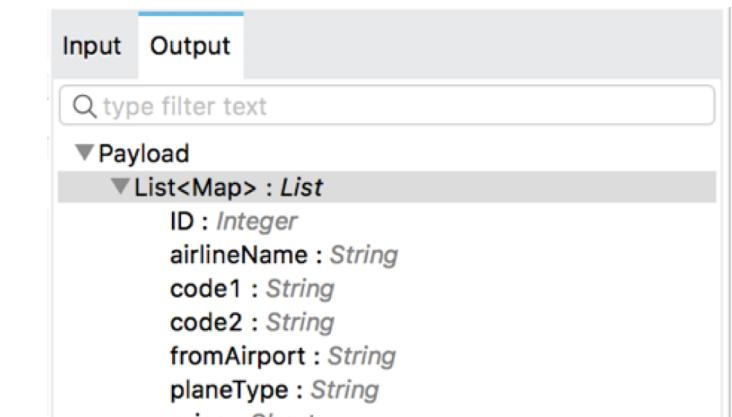
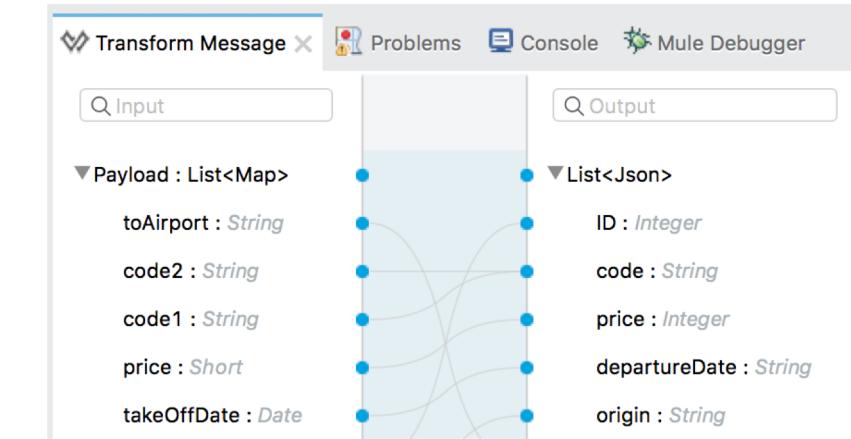


- At **design time** in Anypoint Studio using DataSense

- In the Transform Message component
- In the DataSense Explorer
- When writing expressions using auto-completion

- **DataSense** is the ability to proactively discover metadata from internal and external resources

- Keeps you from having to manually discover info about the data
- Facilitates transformations by providing DataWeave expected input or output



Viewing event data at run time



- **At run time**

- In the client when making a request
- For deployed applications, in the log files
- In the Anypoint Studio console by using a Logger
- In Anypoint Studio using the Visual Debugger
 - Most comprehensive way

200 Success 1759.59 ms DETAILS ^

GET http://localhost:8081/hello?fname=max&lname=mule

Response headers 4 Request headers 0 Redirects 0 Timings

name: Max
content-type: application/octet-stream; charset=UTF-8
content-length: 7
date: Thu, 19 Apr 2018 20:46:15 GMT

Goodbye

Mule Debugger

| Name | Value |
|---------------|---------------------------------|
| method | GET |
| queryParams | MultiMap{[fullName=[Max Mule]]} |
| 0 | fullName=Max Mule |
| key | fullName |
| value | Max Mule |
| queryString | fullName=Max Mule |
| relativePath | /goodbye |
| remoteAddress | /127.0.0.1:52451 |
| requestPath | /goodbye |
| requestUri | /goodbye?fullName=Max Mule |

View event data by logging it



- Add a **Logger** component to a flow and view its output
- Logged values are displayed
 - For an application run from Anypoint Studio with embedded runtime, in the **Console view**
 - For applications deployed to CloudHub or customer-hosted runtimes, in the **log files**



Logger

Walkthrough 6-1: View event data



- Create a new Mule project with an HTTP Listener and set the payload
- View event data in the DataSense Explorer
- Use a Logger to view event data in the Anypoint Studio console

The screenshot illustrates the Mule project setup and runtime environment:

- Flow Diagram (helloFlow):** Shows an **Listener** (GET /hello) connected to a **Set Payload** component (Hello) which then connects to a **Logger**.
- DataSense Explorer:** Displays the **Output** tab with the following tree structure:
 - Mule Message
 - Payload (String : String)
 - Attributes
 - HttpRequestAttributes : Object
 - listenerPath : String
 - relativePath : String
 - version : String
 - schema : String
 - method : String
 - requestUri : String
 - queryString : String
 - localAddress : String
- Console Tab:** Shows the application logs for **apdev-examples [Mule Applications]** running on **Mule Server 4.1.1 EE**. The log output is:

```
Request path=/hello
Method=GET
Listener path=/hello
Local Address=localhost/127.0.0.1:8081
Query String=fname=max&lname=mule
Relative Path=/hello
Remote Address=/127.0.0.1:49405
Request Uri=/hello?fname=max&lname=mule
Scheme=http
Version=HTTP/1.1
Headers=
  host=localhost:8081
```

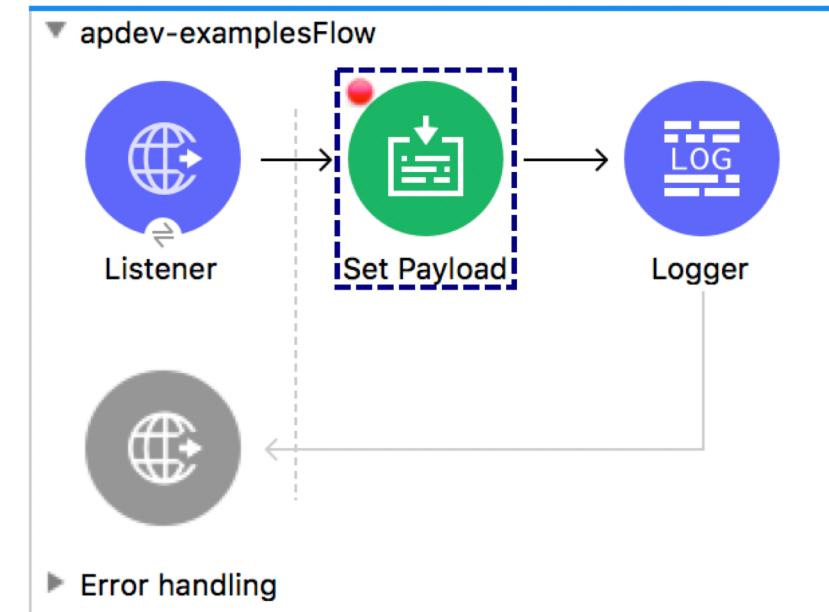
Debugging Mule applications



Debugging applications with the Mule Debugger



- Can add breakpoints to processors and step through the application
 - Watch event properties and values
 - Watch and evaluate DataWeave expressions
- By default, Debugger listens for incoming TCP connections on localhost port 6666
 - Can change this in a project's run configuration



Walkthrough 6-2: Debug a Mule application



- Locate the port used by the Mule Debugger
- Add a breakpoint, debug an application, and step through the code
- Use the Mule Debugger to view event properties
- Pass query parameters to a request and locate them in the Debugger
- Increase the request timeout for Advanced REST Client

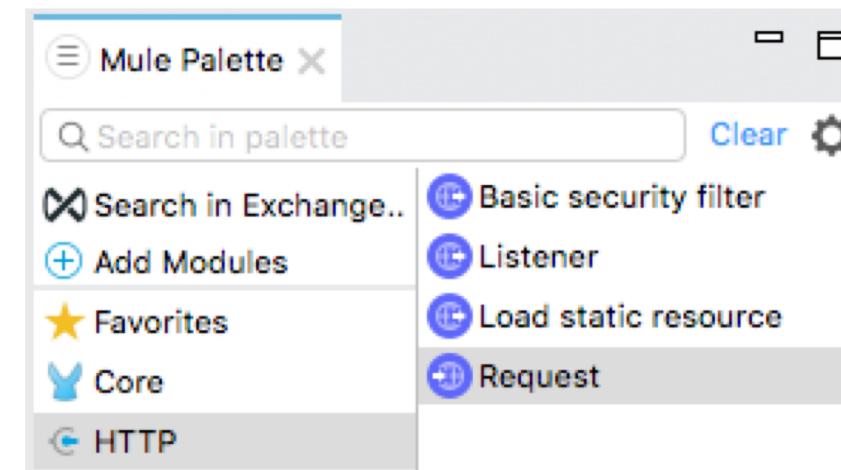
A screenshot of the Mule Studio interface. At the top, there's a toolbar with various icons. Below it is a 'Mule Debugger' window showing a table of event properties. The table has columns for Name, Value, and Type. Some entries include 'Attributes' (clientCertificate, DOUBLE_TAB, headers, listenerPath, localAddress, method, queryParams, queryString, relativePath), 'headers' (host=localhost:8081, content-type=application/json), 'method' (GET), and 'queryParams' (fname=max&lname=mule). The 'DataWeave Expression Watcher' window is also visible. At the bottom, the 'HelloFlow' editor shows a flow starting with a 'Listener' (GET /hello) that sends a message to a 'Set Payload' component (payload: Hello), which then goes to a 'Logger'. There's also an 'Error handling' section.

Tracking event data as it moves in and out of Mule applications



Changes to the event object

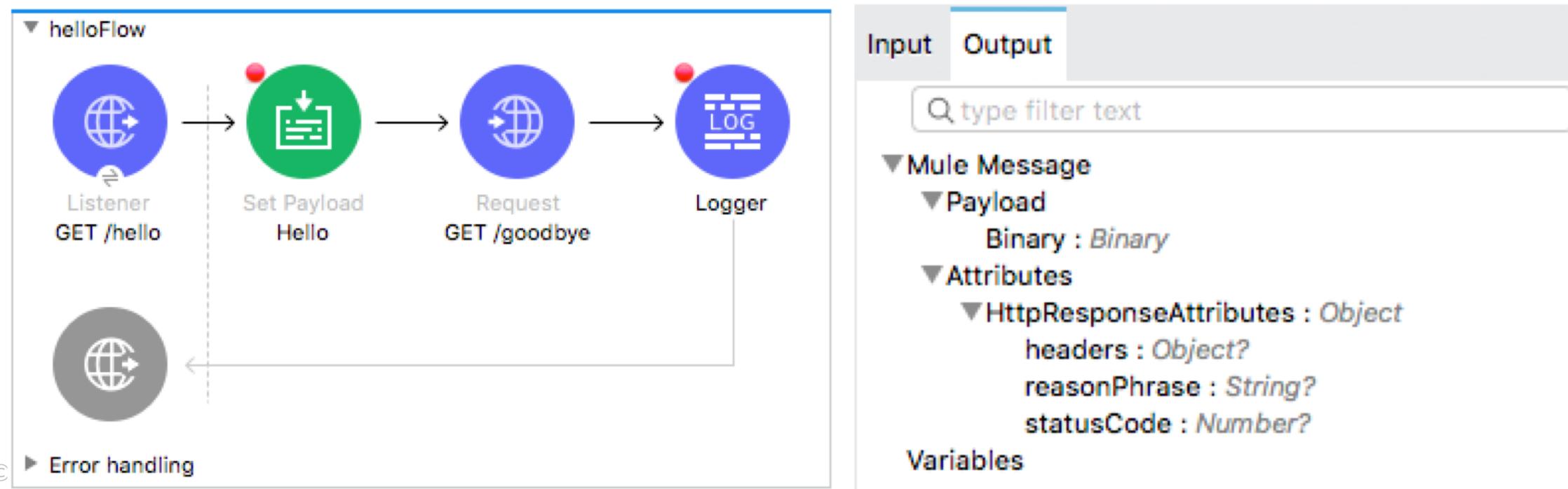
- We examined changes to an event as it moved through a flow and we set the payload
- What happens to the event object when calls are made to an external resource from a flow?
 - For example, you call a web service, and get return data?



Walkthrough 6-3: Track event data as it moves in and out of a Mule application

- Create a second flow with an HTTP Listener
- Make an HTTP request from the first flow to the new HTTP Listener
- View the event data as it moves through both flows

*Note: You are making an HTTP request from one flow to another in this exercise **only** so you can watch the value of event data as it moves in and out of a Mule application. You will learn how to pass events between flows within and between Mule applications in the next module.*



Setting request and response data

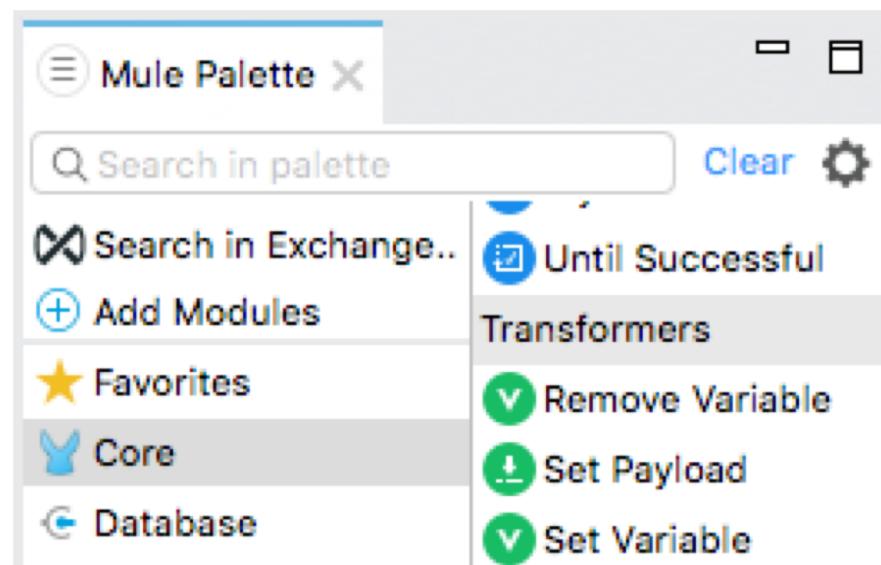


Setting the message payload

- Use the Set Payload transformer to set the payload



Set Payload



Setting data returned from HTTP listeners



- Set in the HTTP Listener properties view > Responses

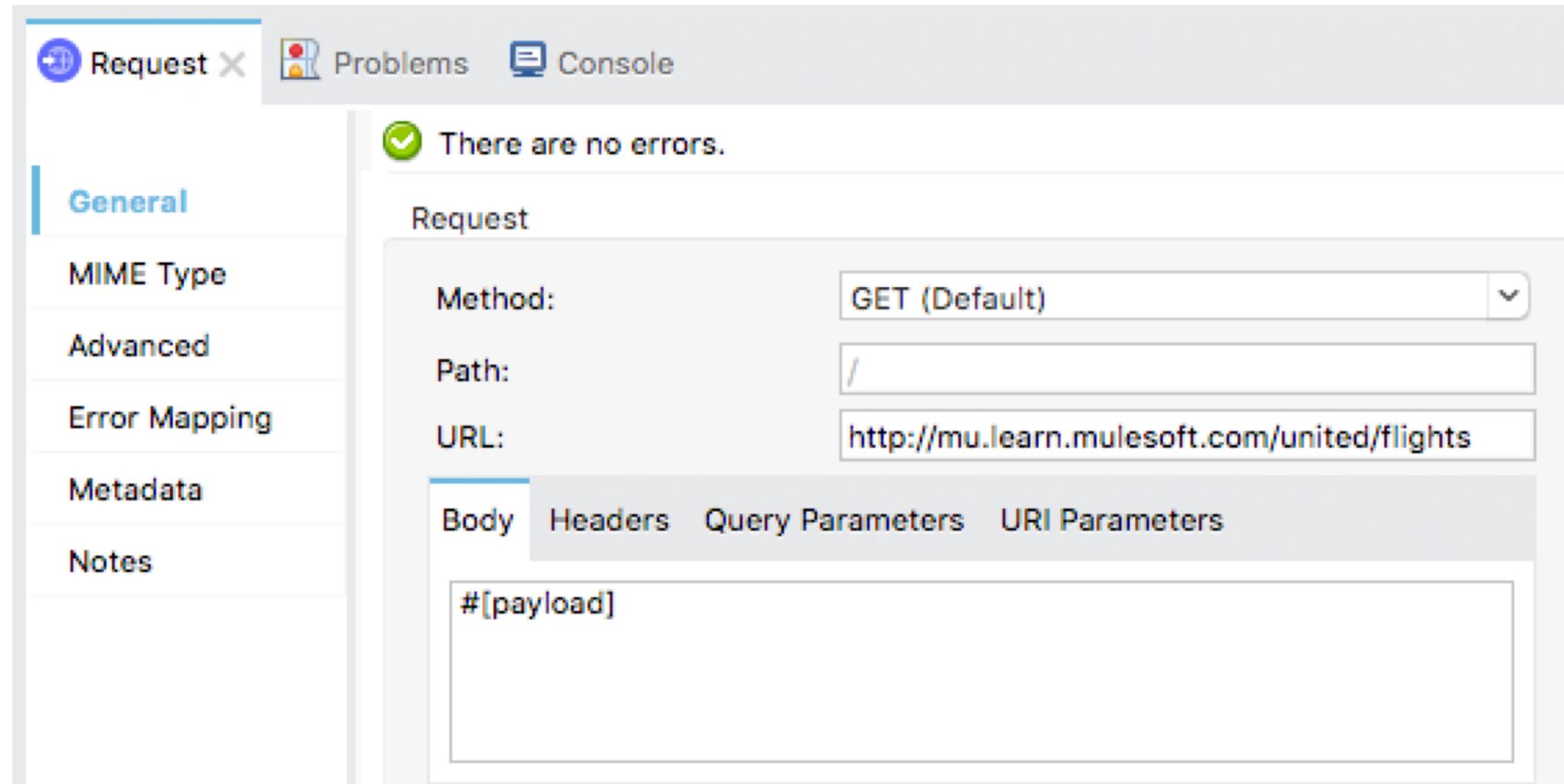
The screenshot shows the 'HTTP Listener' properties view in MuleSoft Anypoint Studio. The left sidebar lists tabs: General, MIME Type, Redelivery, **Responses**, Advanced, Metadata, and Notes. The 'Responses' tab is selected. The main area displays the 'Response' configuration:

- Body:** `#[payload]`
- Headers:** A table with columns 'Name' and 'Value'. It contains one row with 'name' in the Name column and 'max' in the Value column. There are also icons for adding (+), removing (X), and editing (pencil).
- Status code:** 200
- Reason phrase:** Success

A status message at the top says, "There are no errors." with a green checkmark icon.

Setting data sent to HTTP requests

- Set in the HTTP Request properties view > General



Walkthrough 6-4: Set request and response data

- View the default setting for a response body
- Set a response header
- View the default setting for a request body
- Set a request query parameter

The screenshot shows two API configurations side-by-side in the MuleSoft Anypoint Studio interface.

Left API (GET /hello):

- General:** Response Body: #[payload], Headers: name: Max.
- MIME Type:** Not explicitly set.
- Redelivery:** Not explicitly set.
- Responses:** Selected tab. Response Headers: name: Max.
- Advanced:** Not explicitly set.
- Metadata:** Not explicitly set.
- Notes:** Not explicitly set.

Right API (GET /goodbye):

- General:** Request Method: GET (Default), Path: /goodbye.
- MIME Type:** Not explicitly set.
- Advanced:** Not explicitly set.
- Error Mapping:** Not explicitly set.
- Metadata:** Not explicitly set.
- Notes:** Not explicitly set.

Request Headers: Name: "fullName", Value: "Max Mule".

Using DataWeave expressions to read and write event data



The DataWeave expression language



- A Mule-specific expression and transformation language
- Can be used to access and evaluate the data in the payload, attributes, and variables of a Mule event
- Accessible and usable from all event processors and global elements
 - Is used to modify the way the processors act upon the event such as routing
- Case-sensitive
- Easy to use with auto-complete everywhere

Types of DataWeave expressions



- **Standalone scripts**

- Were generated using the Transform Message graphical editor in Module 4
- We will write these from scratch in Module 11

- **Inline expressions**

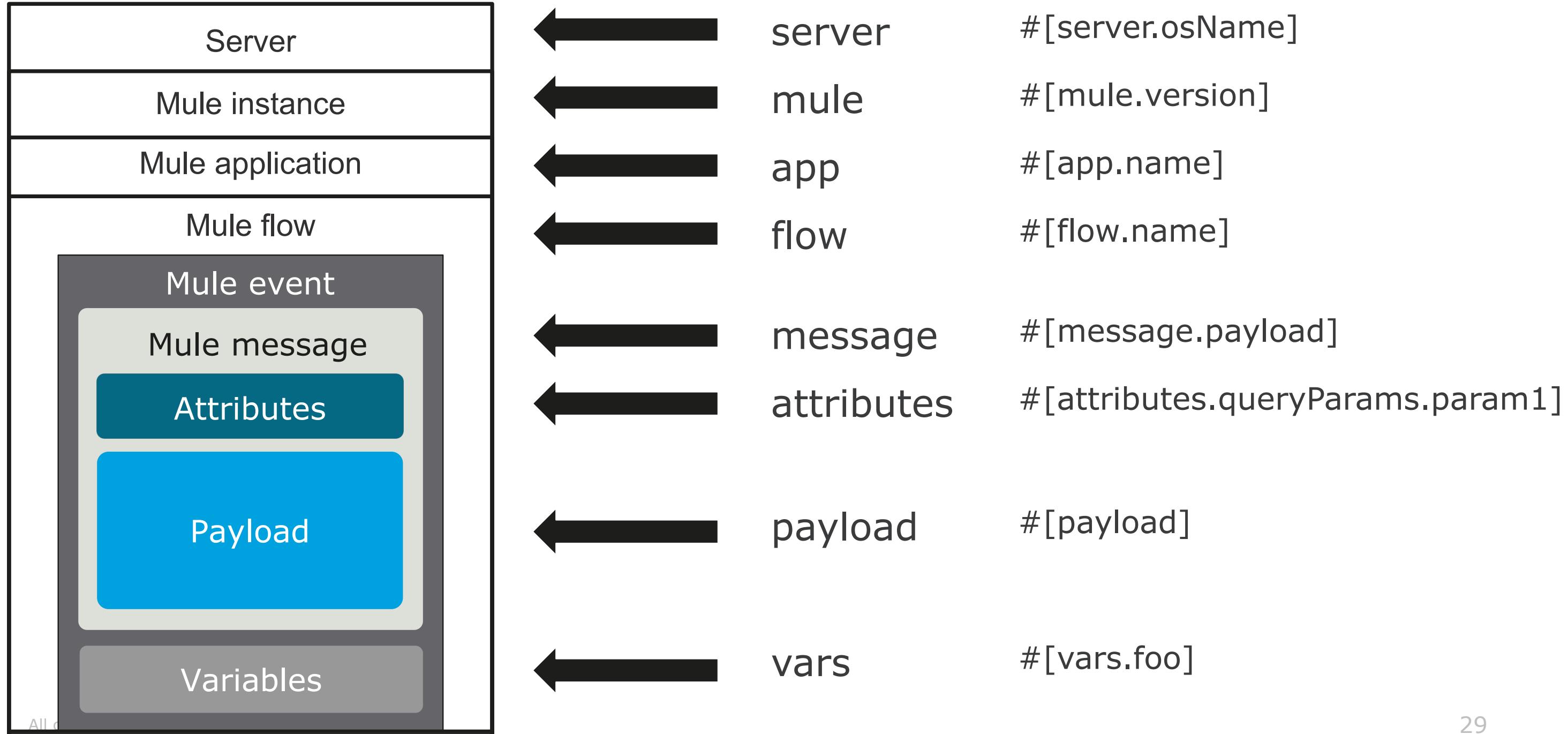
- Are used to dynamically set the value of properties in an event processor or global configuration element
- Are enclosed in #[]

A screenshot of the MuleSoft Anypoint Studio interface, specifically the DataWeave editor. The title bar says "Output Payload". The editor area contains the following DataWeave code:

```
1@%dw 2.0
2 output application/json
3 ---
4@payload map ( payload01 , indexOfPayload01 ) -> {
5@    ID: payload01.ID,
6@    code: payload01.code1 default "",
7@    departureDate: payload01.takeOffDate as String
8 }
```

[]

Referencing Mule objects in DataWeave expressions



Accessing event attributes



Mule event

Mule message

Attributes

```
method = POST  
headers.host=mulesoft.org  
headers.user-agent=Mozilla
```

Payload

```
id: ed921739-99c1-4e09  
custId: 49e377ed-bc72-4523  
itemsTotal: 200.34
```

java.util.Map

Variables

`#[message.attributes.method]`

POST

`# [attributes.method]`

POST

`# [attributes.headers.host]`

mulesoft.org

`# [attributes.header['user-agent']]`

Mozilla

`# [message.payload.id]`

ed921739-99c1-4e09

`# [payload.id]`

ed921739-99c1-4e09

`# [payload['id']]`

ed921739-99c1-4e09

`# [payload.itemsTotal]`

200.34

Using selectors in DataWeave expressions



| Description | Selector | Example |
|--------------------------------------|-----------------------|--|
| Value of a key:value pair | Single Value selector | # [payload.name] # [attributes.queryParams] |
| Value at selected array index | Indexed selector | # [payload[0].name] |
| Array with values of key:value pairs | Multi Value selector | # [payload.*name] |
| Array with values of key:value pairs | Descendants selector | # [payload..zip] |

Using operators in DataWeave expressions



| Description | Operators | Example |
|---------------|------------------|--|
| Arithmetic | + , - , / , * | # [payload . age * 2] |
| Equality | ==, !=, ~ = | # [payload . name == "max"] |
| Relational | >, >=, <, <=, is | # [payload . age > 30] |
| Conditional | and, or, not | # [(payload . name == "max") and (payload . age > 30)] |
| Type coercion | as | # [(payload . age as Number) * 2] |
| Default value | default | # [payload . type default "student"] |

Note: For operator precedence, see

<https://docs.mulesoft.com/mule4-user-guide/v/4.1/dataweave-flow-control-precedence>

Using conditional logic statements in DataWeave



- Use if / else if / else statements

```
if (payload.age < 15)
    group: "child"
else if (payload.age < 25)
    group: "youth"
else if (payload.age < 65)
    group: "adult"
else
    group: "senior"
```

Using DataWeave functions

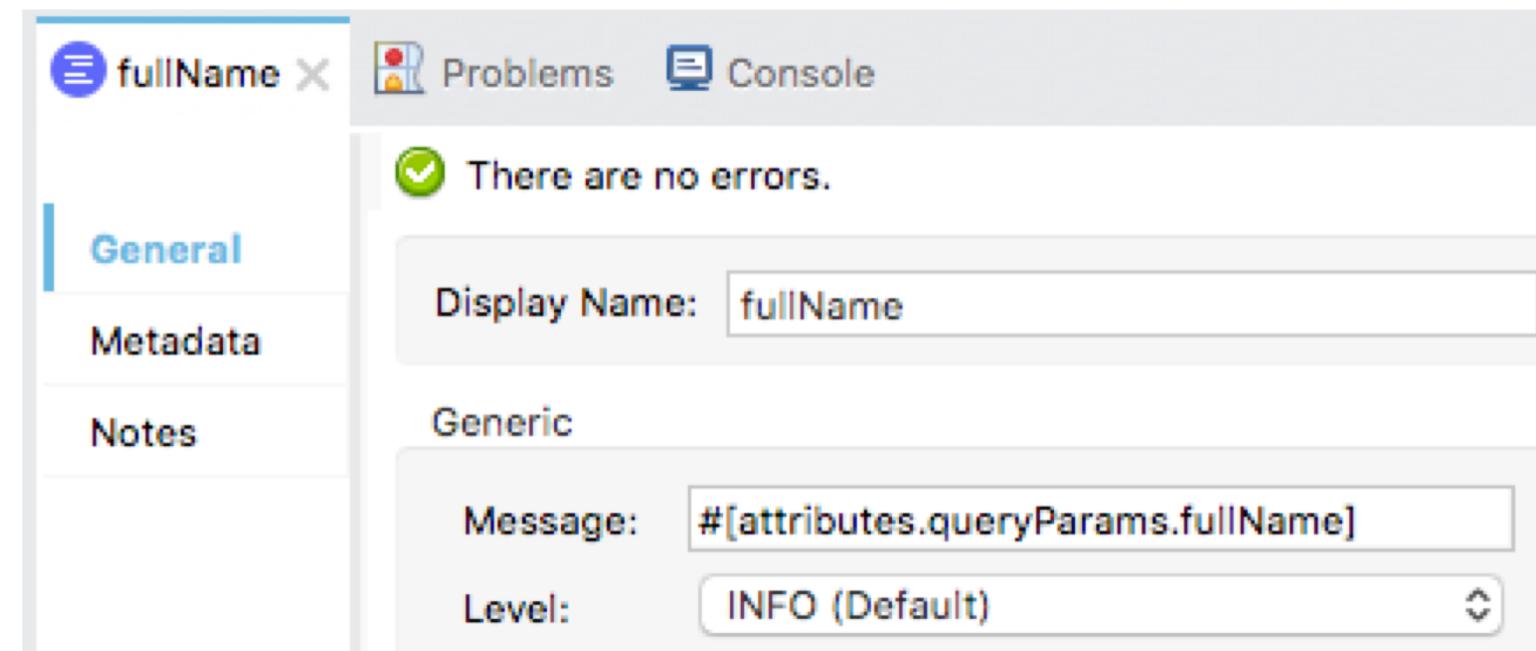


- Functions are packaged in modules
- Functions in the Core module are imported automatically into DataWeave scripts
- For function reference, see
 - <https://docs.mulesoft.com/mule4-user-guide/v/4.1/dataweave>
- For functions with 2 parameters, an alternate syntax can be used
 - #[contains(payload,max)]
 - #[payload contains "max "]

| DataWeave Function Reference | | | | |
|------------------------------|------------|-----------|------------|--|
| Core (dw::Core) | groupBy | maxBy | scan | |
| ++ | isBlank | min | sizeOf | |
| -- | isDecimal | minBy | splitBy | |
| abs | isEmpty | mod | sqrt | |
| avg | isEven | native | startsWith | |
| ceil | isInteger | now | sum | |
| contains | isLeapYear | orderBy | to | |
| daysBetween | isOdd | pluck | trim | |
| distinctBy | joinBy | pow | typeOf | |
| endsWith | log | random | unzip | |
| filter | lower | randomInt | upper | |
| filterObject | map | read | uuid | |
| find | mapObject | readUrl | with | |
| flatMap | match | reduce | write | |
| flatten | matches | replace | zip | |
| floor | max | round | 34 | |

Walkthrough 6-5: Get and set event data using DataWeave expressions

- Use expressions to set the payload and a logged value
- Use expressions to set a response header and a request query param
- In expressions, reference values for the event payload and attributes
- Use the DataWeave upper() function and the concatenation, as, and default operators



Creating variables



Creating variables

- Create variables to store metadata for the Mule event

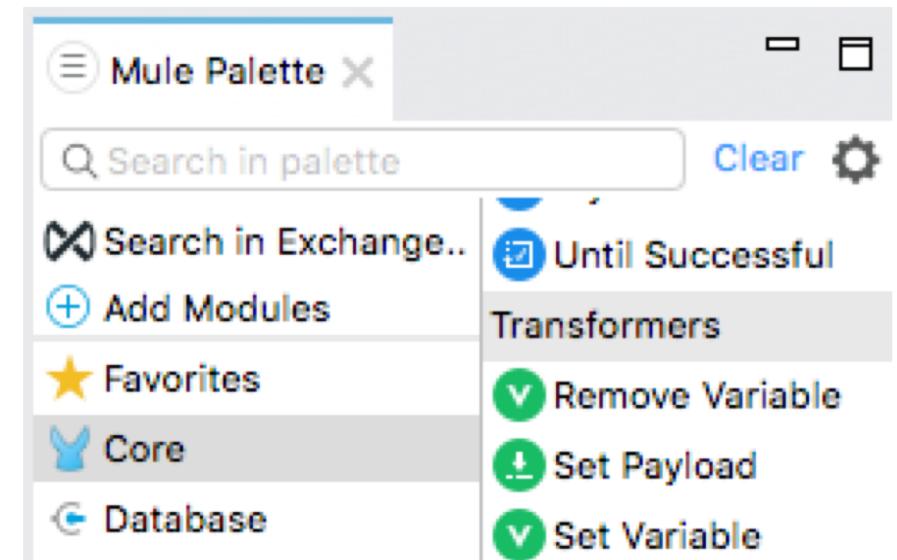
- Use the Set Variable transformer to create a variable

- In expressions, reference as vars

- #[vars.foo]



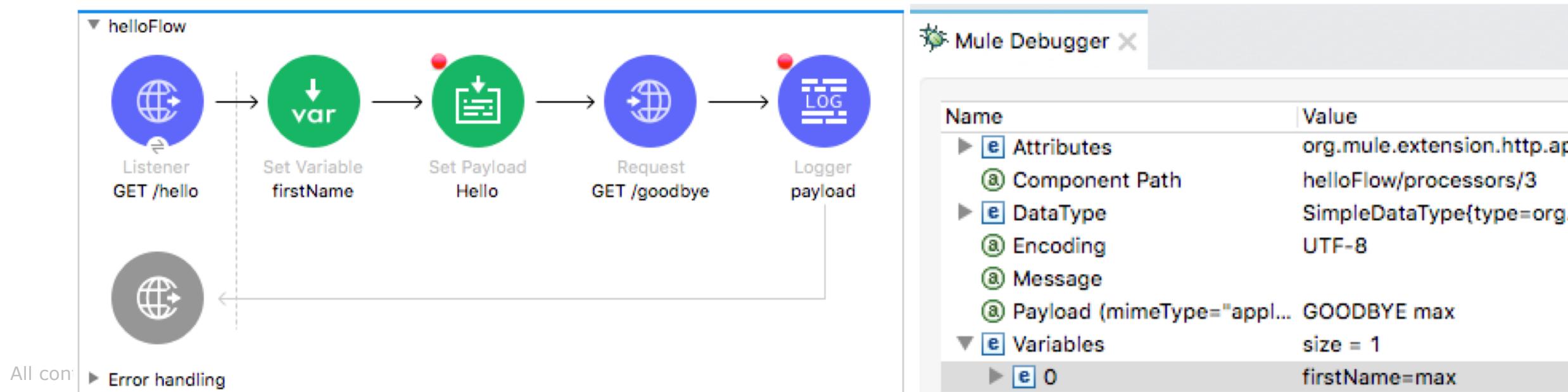
Set Variable



Walkthrough 6-5: Set and get variables



- Use the Set Variable transformer to create a variable
- Reference a variable in a DataWeave expression
- Use a variable to dynamically set a response header
- Use the Mule Debugger to see the value of a variable
- Track variables across a transport boundary



The image shows a Mule flow diagram titled "helloFlow" and a "Mule Debugger" window.

Mule Flow Diagram:

- Starts with a "Listener" component (blue circle with a globe icon) receiving "GET /hello".
- An arrow points to a "Set Variable" transformer (green circle with a downward arrow icon). It has a configuration: "Name" is "firstName" and its value is "var".
- From the "Set Variable" transformer, an arrow points to a "Set Payload" transformer (green circle with a clipboard icon). Its configuration is "Hello".
- From the "Set Payload" transformer, an arrow points to a "Request" component (blue circle with a globe icon) sending "GET /goodbye".
- Finally, an arrow points to a "Logger" component (blue circle with a log icon) with the message "payload".

Mule Debugger Window:

| Name | Value |
|------------------------------|----------------------------|
| Attributes | org.mule.extension.http.ap |
| Component Path | helloFlow/processors/3 |
| DataType | SimpleDataType{type=org.j |
| Encoding | UTF-8 |
| Message | |
| Payload (mimeType="appl...") | GOODBYE max |
| Variables | size = 1 |
| 0 | firstName=max |

Summary



- The best way to view event data is to add breakpoints to a flow and use the **Mule Debugger**
- Use the **Logger** component to display data in the console
- Use the **Set Payload** transformer to set the payload
- Use the properties view to set response data for an HTTP Listener and request data for an HTTP Request operation
- Use the **DataWeave language** to write inline expressions in `#[]`
- Use the **Set Variable** transformer to create variables