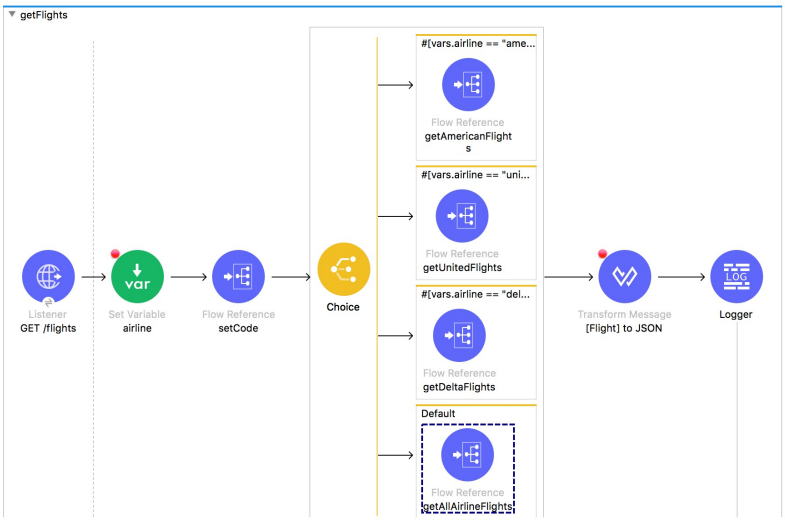
# PART 2: Building Applications with Anypoint Studio

1

## Goal

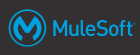







2

## 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



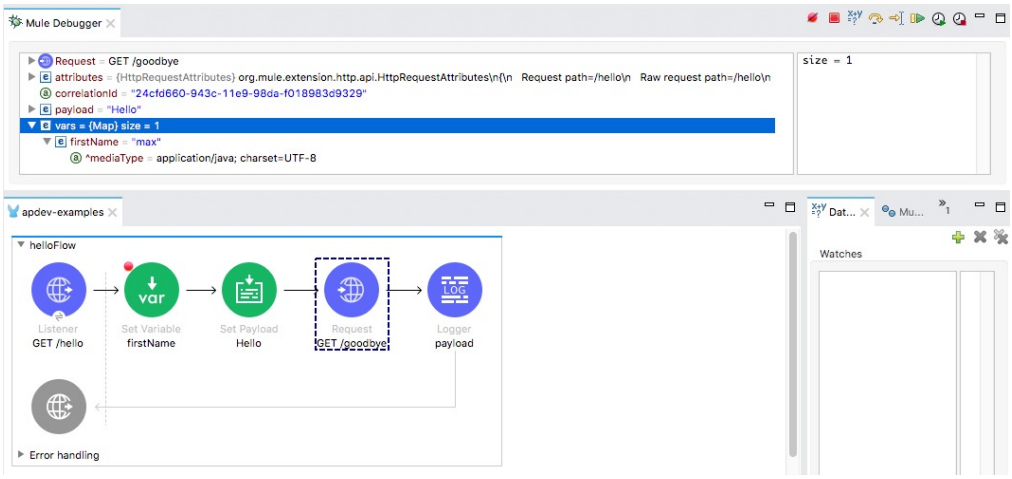


3

# Module 6: Accessing and modifying Mule events

4

## Goal










## 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

7

## Reviewing the structure of Mule 4 events

MuleSoft

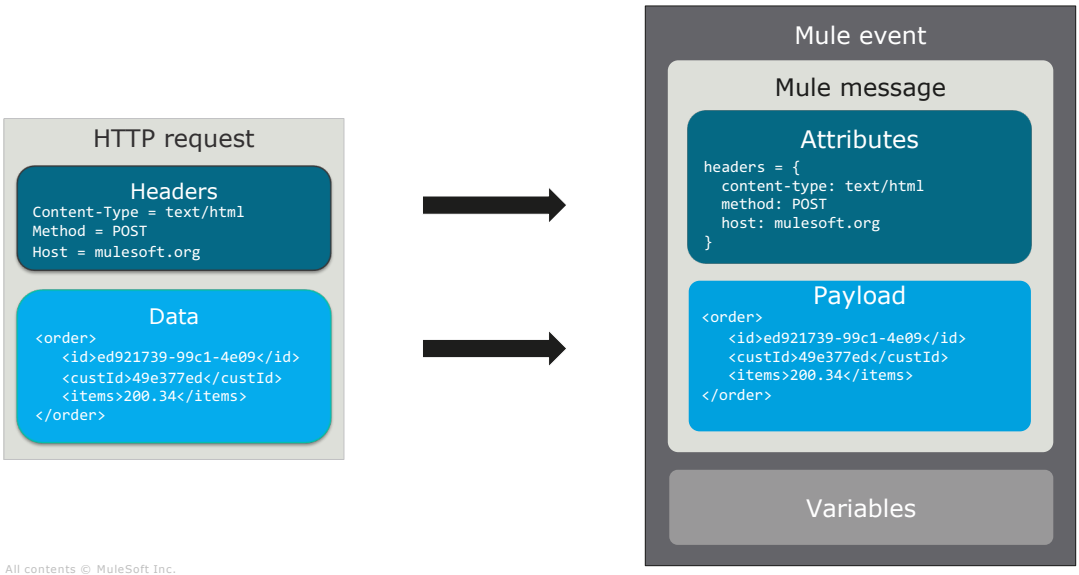

- The data that passes through flows in the app
- Metadata contained in the message header
- The core info of the message - the data the app processes
- Metadata for the Mule event - can be defined and referenced in the app processing the event





8

## Event data populated from an HTTP request







9

## Viewing event data

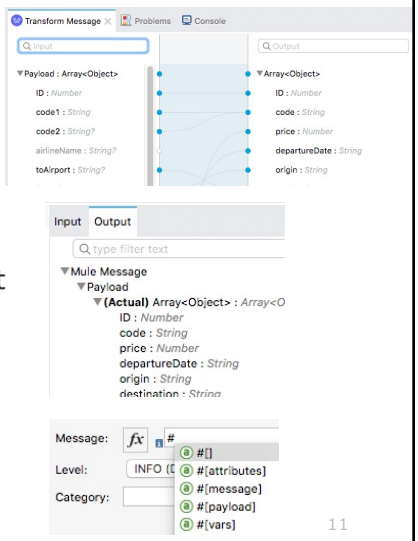
- At **design time** in Anypoint Studio
  - What did we see so far in the course and where?
- At **run time**
  - What did we see so far in the course and where?




10

## 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



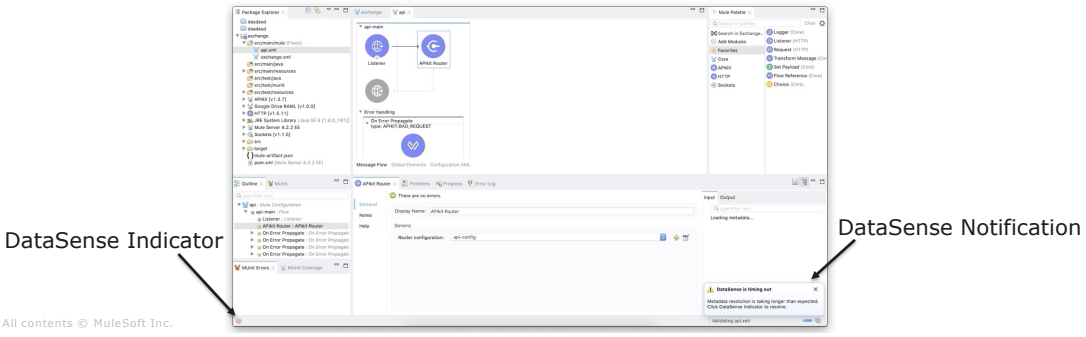





11

## DataSense Notification and Indicator

- If DataSense is timing out, a notification will appear
- A red dot DataSense indicator will also appear in the lower-left corner
- Clicking on this indicator allows for checking on the DataSense process and restarting it if necessary



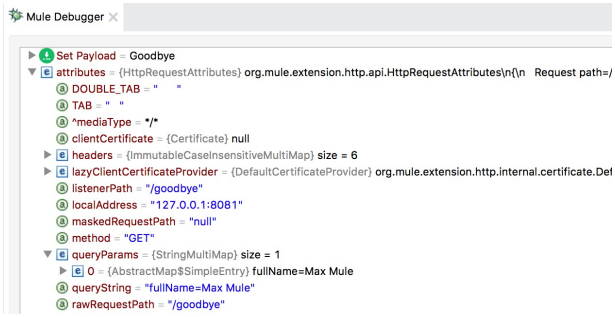





12

## 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







## View event data by logging it

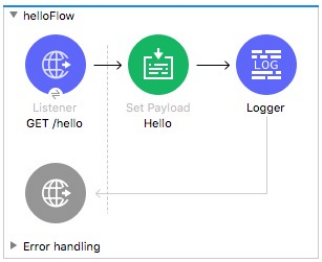
- Add a **Logger** component to a flow and view its output

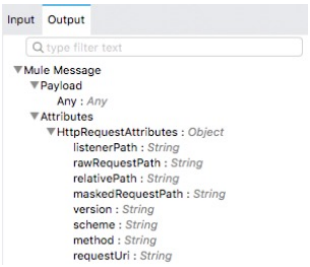
Logger

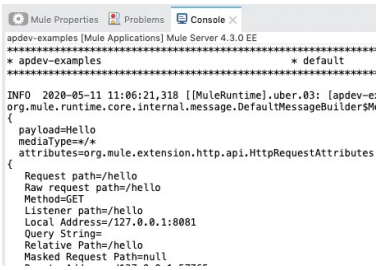
- Logged values are displayed
  - For an application run from Anypoint Studio with embedded runtime, in the **Console view**
  - For applications deployed to CloudHub or customer-hosted runtimes, in the **log files**

## Walkthrough 6-1: View event data

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







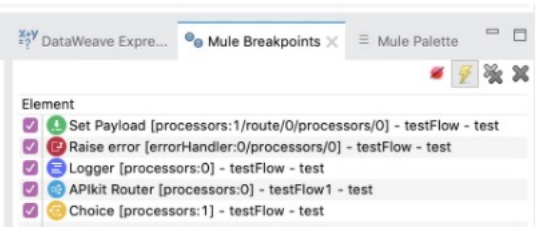

15

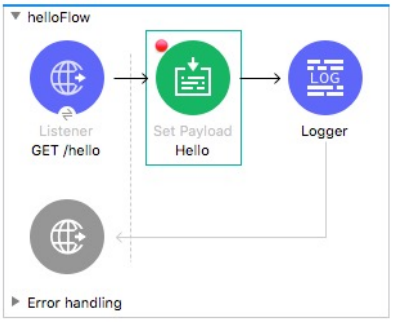
# Debugging Mule applications

16

## 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
  - Can be managed from the breakpoints list



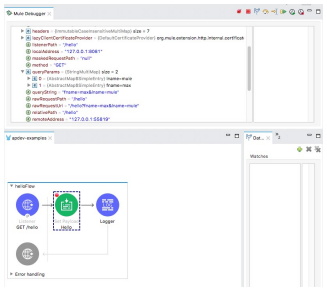

- By default, Debugger listens for incoming TCP connections on localhost port 6666
  - Can change this in a project's run configuration




17

## 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






18

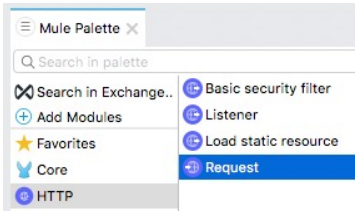
# Tracking event data as it moves in and out of Mule applications

19

## Changes to the event object

MuleSoft

- 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?





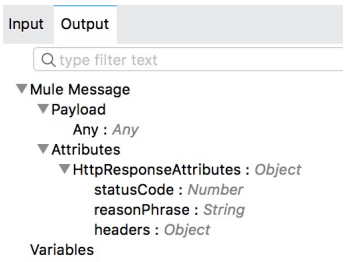

20

20

## 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.*






21

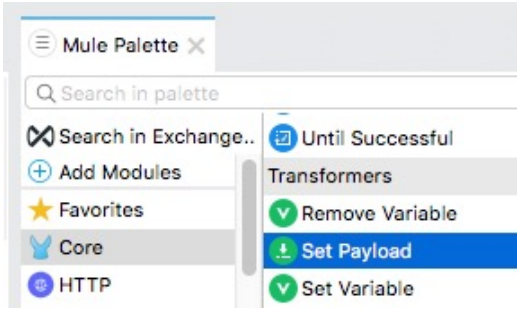
# Setting request and response data

22

## Setting the event payload

MuleSoft

- Use the Set Payload transformer to set the payload



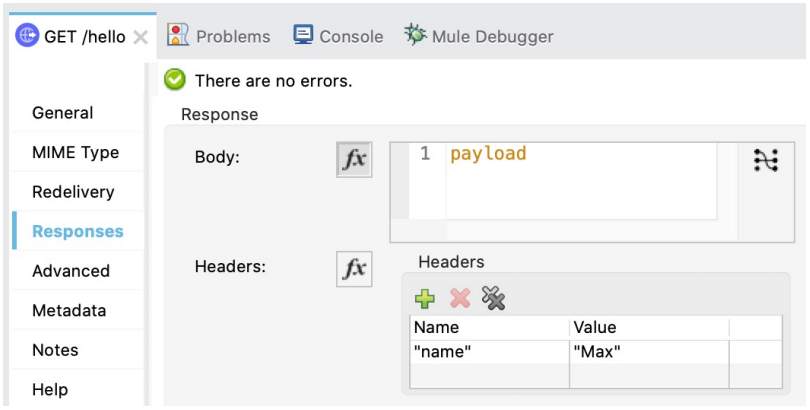




## Setting data returned from HTTP listeners

MuleSoft

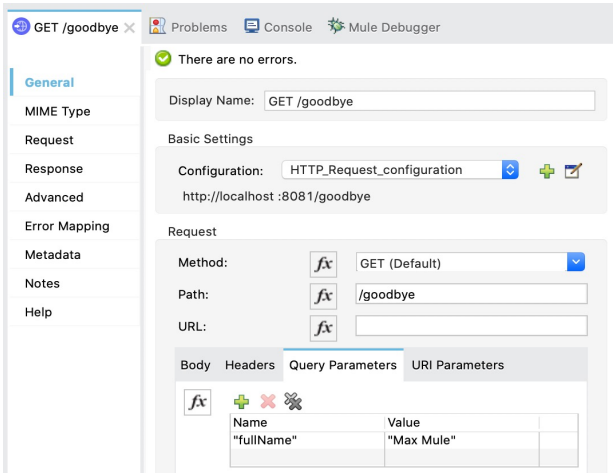
- Set in the HTTP Listener properties view > Responses



## Setting data sent to HTTP requests

MuleSoft

- Set in the HTTP Request properties view > General





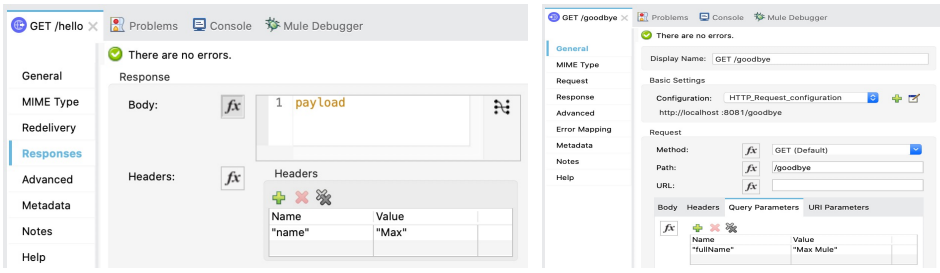





## Walkthrough 6-4: Set request and response data

MuleSoft

- 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



# Using DataWeave expressions to read and write event data

27

## The DataWeave expression language

MuleSoft

- 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



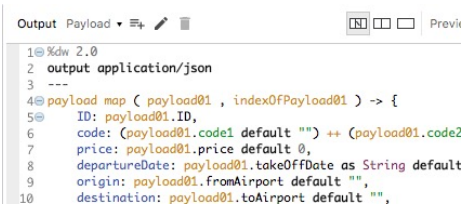


28

## Types of DataWeave expressions

MuleSoft

- **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 #[]
    - Many places in the product offer an expression mode button

#[]

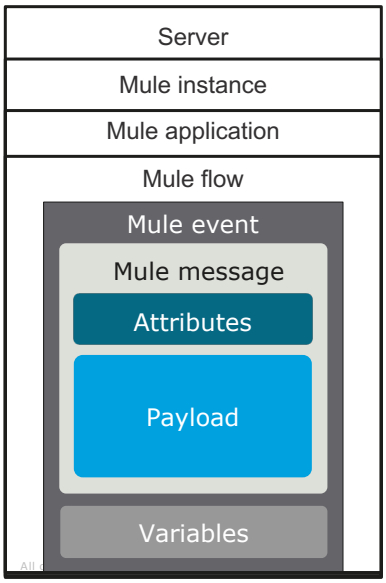




29

## Referencing Mule objects in DataWeave expressions

MuleSoft

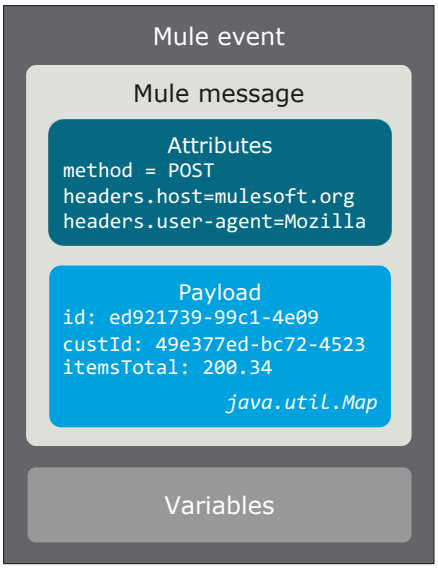



| | |
|---|---|
| server | #[server.osName] |
| mule | #[mule.version] |
| app | #[app.name] |
| flow | #[flow.name] |
| message | #[message.payload] |
| attributes | #[attributes.queryParams.param1] |
| payload | #[payload] |
| vars | #[vars.foo] |



30

## Accessing event attributes




| | |
|---|---|
| `#[message.attributes.method]` | POST |
| `#[attributes.method]` | POST |
| `#[attributes.headers.host]` | mulesoft.org |
| `#[attributes.headers['user-agent']]` | Mozilla |

| | |
|---|---|
| `#[message.payload.id]` | ed921739-99c1-4e09 |
| `#[payload.id]` | ed921739-99c1-4e09 |
| `#[payload['id']]` | ed921739-99c1-4e09 |
| `#[payload.itemsTotal]` | 200.34 |

31

31

## Using selectors in DataWeave expressions

| Description | Selector | Example |
|---|---|---|
| Value of a key:value pair | Single Value selector | `#[payload.name]`<br>`#[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]` |



Note: You will learn additional selectors in Module 11

32

32

## Using operators in DataWeave expressions

MuleSoft

| 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/mule-runtime/4.3/dataweave-flow-control-precedence




33

## Using conditional logic statements in DataWeave

MuleSoft

- 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"
```



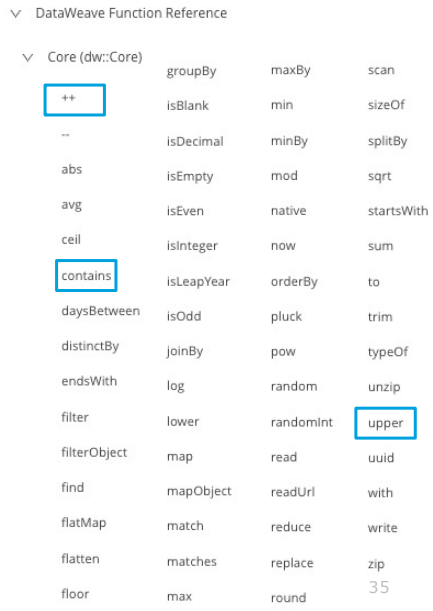

34

## Using DataWeave functions

MuleSoft

- Functions are packaged in modules
- Functions in the Core module are imported automatically into DataWeave scripts
- For function reference, see
  - https://docs.mulesoft.com/mule-runtime/4.3/dataweave
- For functions with 2 parameters, an alternate syntax can be used

```
#[contains(payload, "max")]
#[payload contains "max"]
```





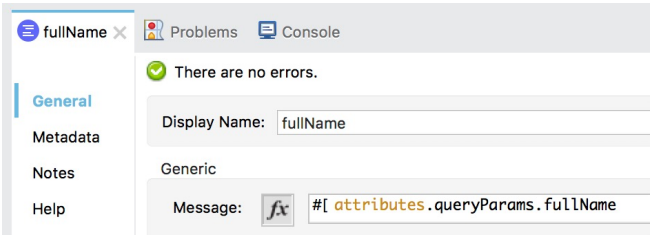

35

35

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

MuleSoft

- 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, the concatenation, as, and default operators






36

# Creating variables

37

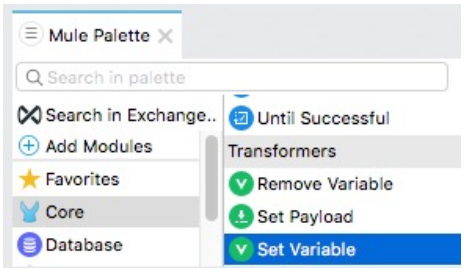
## Creating variables

MuleSoft

- Create variables to store metadata for the Mule event
- Use the Set Variable transformer to create a variable


Set Variable

- In expressions, reference as vars
  - #[vars.foo]





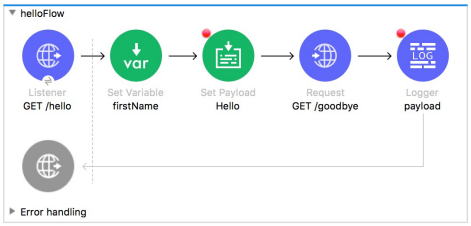

38

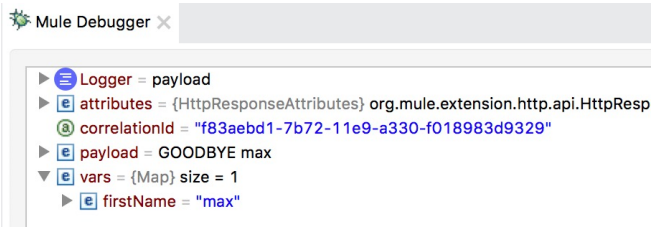
38

## Walkthrough 6-6: Set and get variables

MuleSoft

- 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








39

## Summary

40

## Summary

MuleSoft

- 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