



Module 7: Structuring Mule Applications



Goal



▼ apdev-examples
 ▼ src/main/mule (Flows)
 accounts.xml
 apdev-examples.xml
 global.xml
 src/main/java
 src/main/resources
 application-types.xml
 config.yaml
 log4j2.xml
 api



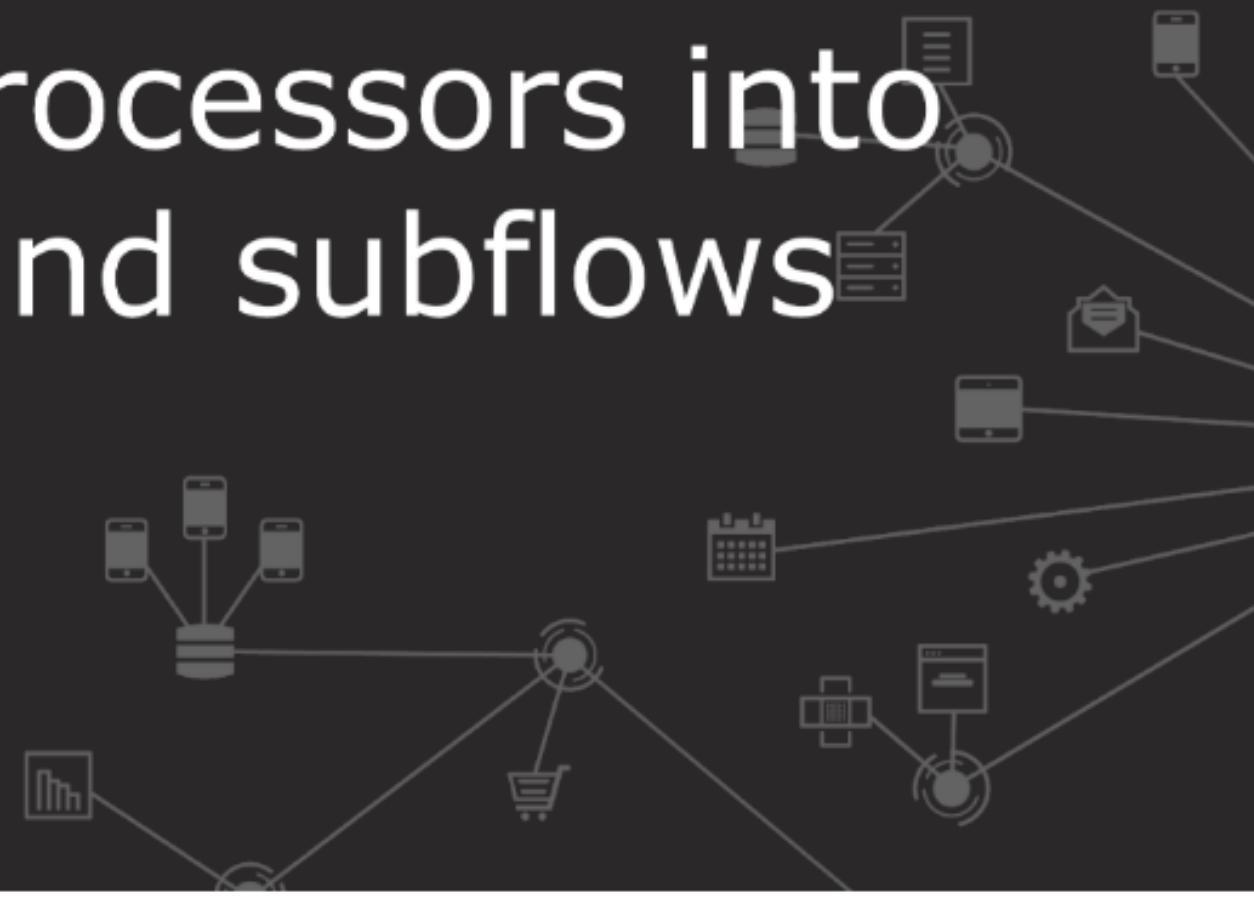
▼ apdev-flights-ws
 ▼ src/main/mule (Flows)
 global.xml
 implementation.xml
 interface.xml
 src/main/java
 com.mulesoft.training
 Flight.java
 src/main/resources
 application-types.xml
 config.yaml
 log4j2.xml
 api
 mua-flights-api.raml
 api.datatypes
 api.examples
 examples
 schemas
 src/test/java
 src/test/munit
 src/test/resources
 APIKit [v1.1.1]
 HTTP [v1.1.0]
 JRE System Library [Java SE 8]
 src
 target
 {} mule-artifact.json
 pom.xml [Mule Server 4.1.1 EE]

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



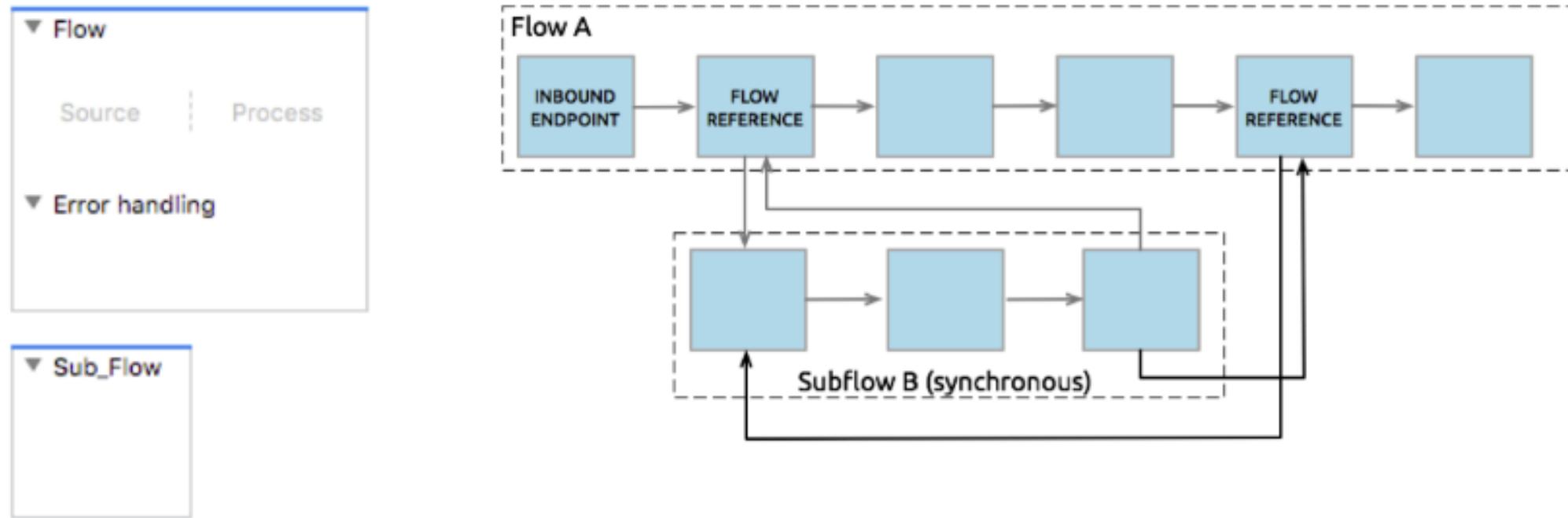
- Create applications composed of multiple flows and subflows
- Pass messages between flows using asynchronous queues
- Encapsulate global elements in separate configuration files
- Specify application properties in a separate properties file and use them in the application
- Describe the purpose of each file and folder in a Mule project
- Define and manage application metadata

Encapsulating processors into separate flows and subflows



- Makes the graphical view more intuitive
 - You don't want long flows that go off the screen
- Makes XML code easier to read
- Enables code reuse
- Provides separation between an interface and implementation
 - We already saw this
- Makes them easier to test

- **Flows** can have their own error handling strategy, **subflows** cannot
- Flows without event sources are sometimes called **private** flows
- Subflows are executed exactly as if the processors were still in the calling flow



- Several methods
 - Add a new scope: Flow or Sub Flow
 - Drag any event processor to the canvas – creates a flow
 - Right-click processor(s) in canvas and select Extract to
- Use Flow Reference component to pass events to other flows or subflows
- Variables persist through all flows unless the event crosses a transport boundary
 - We saw this in the last module

Scopes

-  Async
-  Cache
-  Flow
-  For Each
-  Sub Flow
-  Try
-  Until Successful

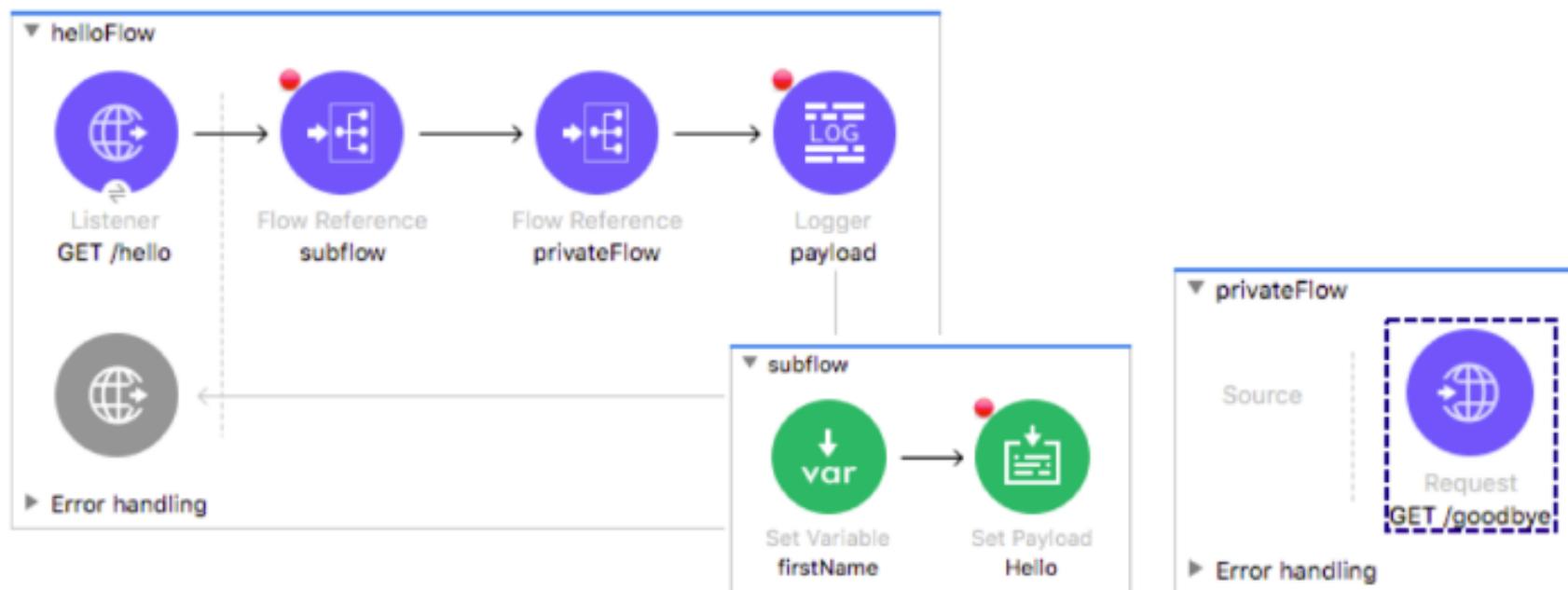
Components

-  Custom Business Event
-  Flow Reference
-  Logger
-  Parse Template
-  Transform Message

Walkthrough 7-1: Create and reference subflows and private flows



- Extract processors into separate subflows and private flows
- Use the Flow Reference component to reference other flows
- Explore event data persistence through subflows and private flows



Walkthrough 7-1: Create and reference subflows and private flows

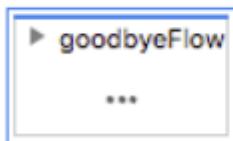
In this walkthrough, you continue to work with apdev-examples.xml. You will:

- Extract processors into separate subflows and private flows.
- Use the Flow Reference component to reference other flows.
- Explore event data persistence through subflows and private flows.



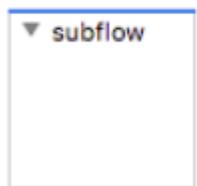
Collapse a flow

1. Return to `apdev-examples.xml` in Anypoint Studio.
2. Click the arrow to the left of the `goodbyeFlow` to collapse it.

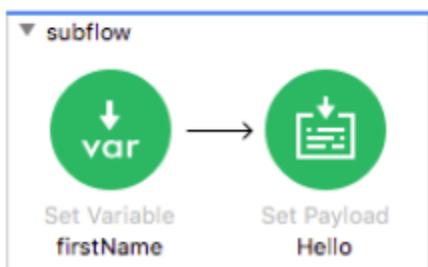
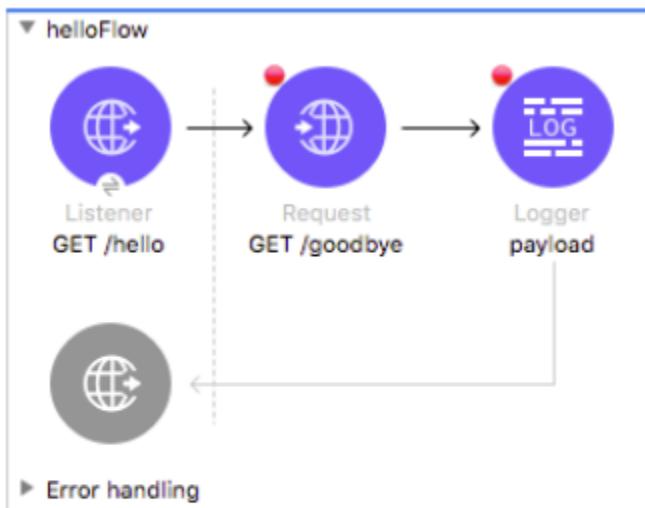


Create a subflow

3. In the Mule Palette, select Core.
4. Drag a Sub Flow scope from the Mule Palette and drop it between the existing flows in the canvas.
5. Change the name of the flow to subflow.

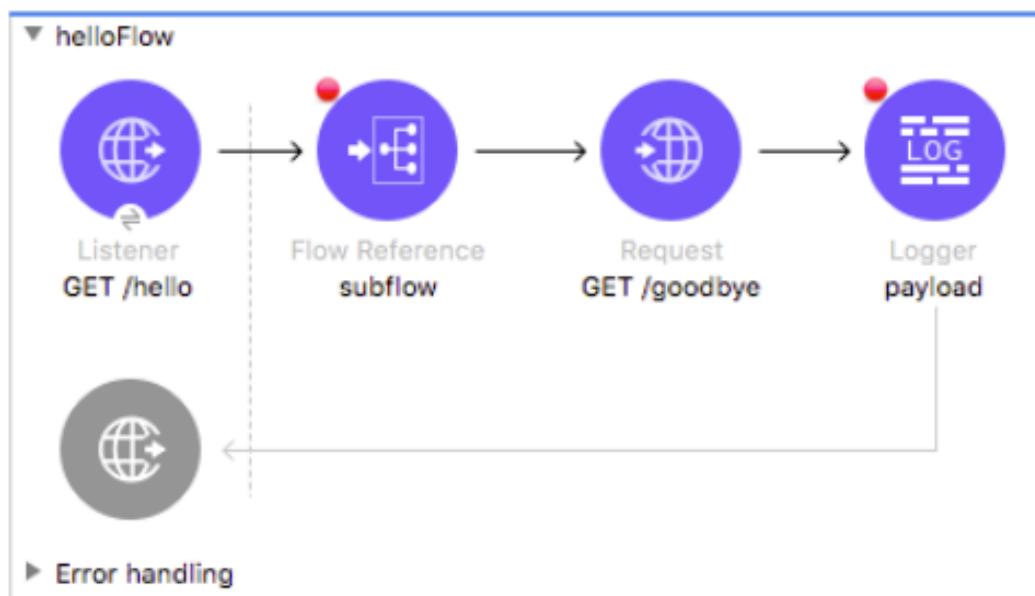


6. Select the Set Variable and Set Payload transformers in helloFlow and drag them into the subflow.



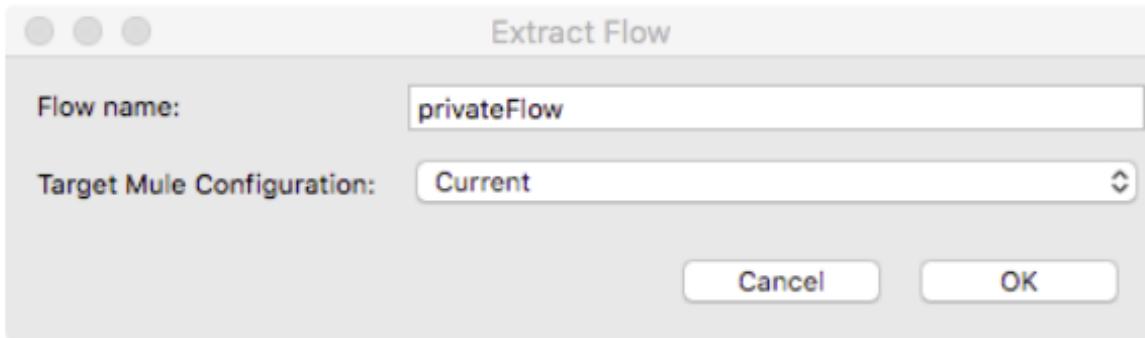
Reference a subflow

7. Drag a Flow Reference component from the Mule Palette and drop it into helloFlow between the GET /hello HTTP Listener and the GET /goodbye HTTP Request.
8. In the Flow Reference properties view, set the flow name and display name to subflow.
9. Add a breakpoint to the subflow Flow Reference.
10. Remove the breakpoint from the GET /goodbye HTTP Request.

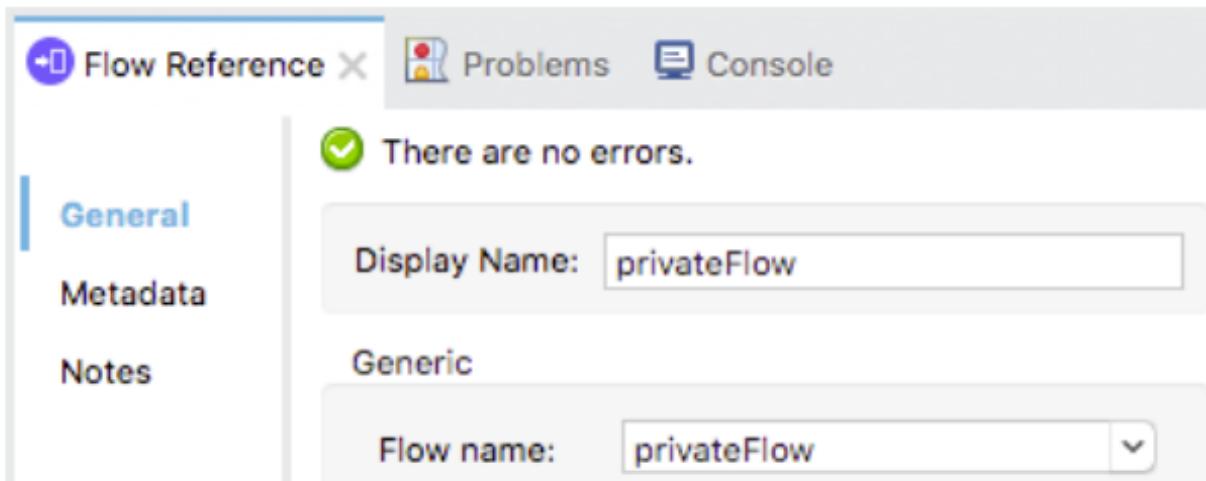


Extract processors into another flow

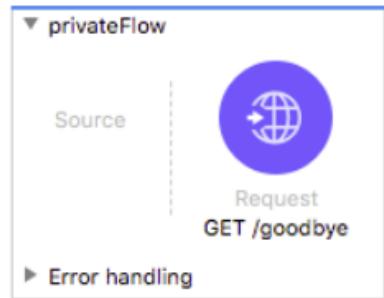
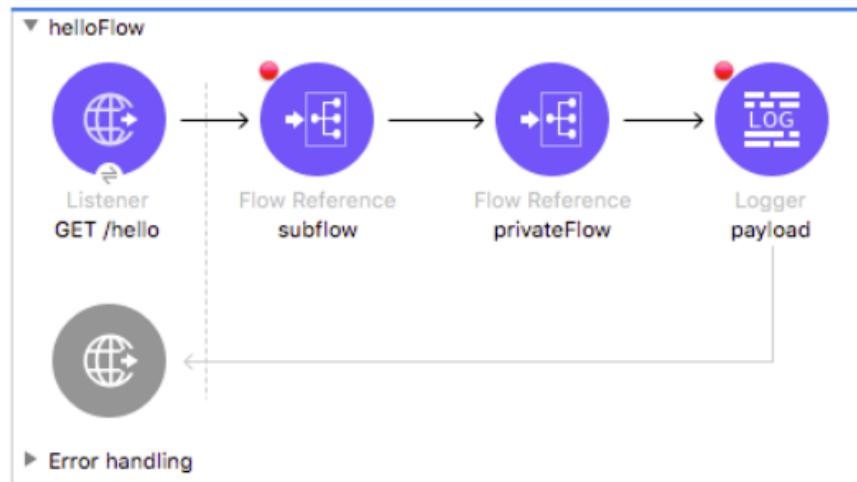
11. Right-click the GET /goodbye HTTP Request and select Extract to > Flow.
12. In the Extract Flow dialog box, set the flow name to privateFlow.



13. Leave the target Mule configuration set to current and click OK.
14. Look at the new Flow Reference properties view; set the display name to privateFlow.

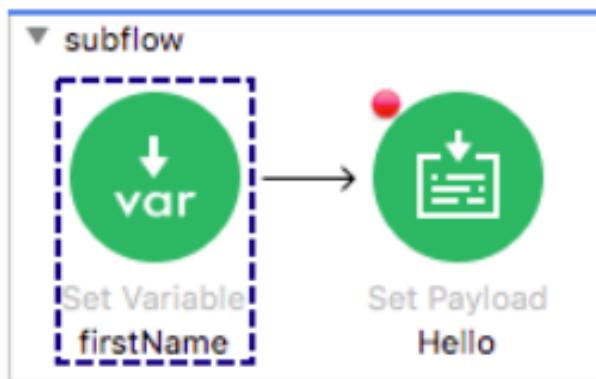


15. Drag privateFlow above subflow in the canvas.



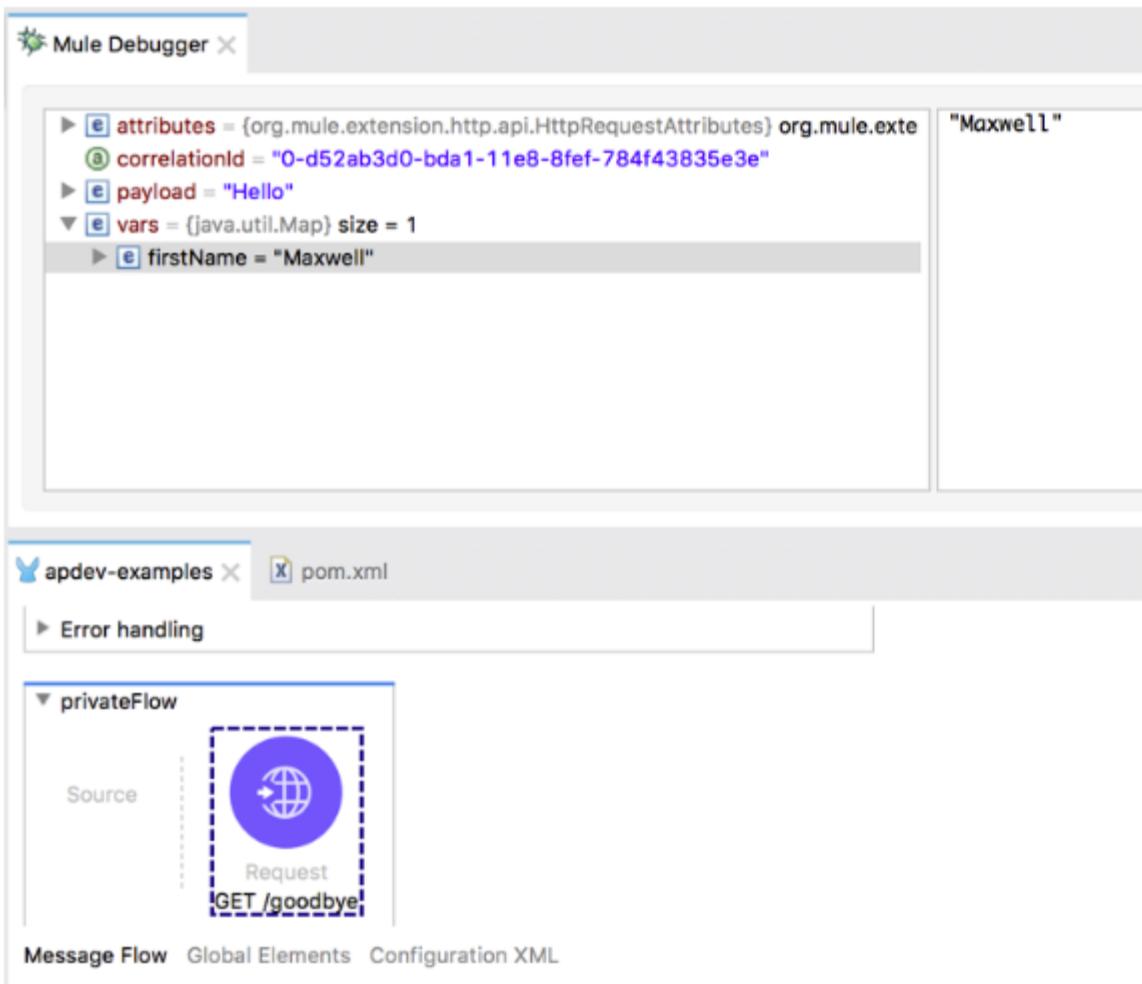
Debug the application

16. Save the file to redeploy the application in debug mode.
17. In Advanced REST Client, send the same request to <http://localhost:8081/hello?fname=Maxwell>.
18. In the Mule Debugger, step through the application, watching as you step into and out of the flows and subflows.



19. Step through the rest of the application.
20. In Advanced REST Client, send the same request again.

21. In the Mule Debugger, step through the application again, this time watching the values of the attributes, payload, and variables in each of the flows.



22. Step through the rest of the application.

23. Switch to the Mule Design perspective.

Passing messages between flows using asynchronous queues



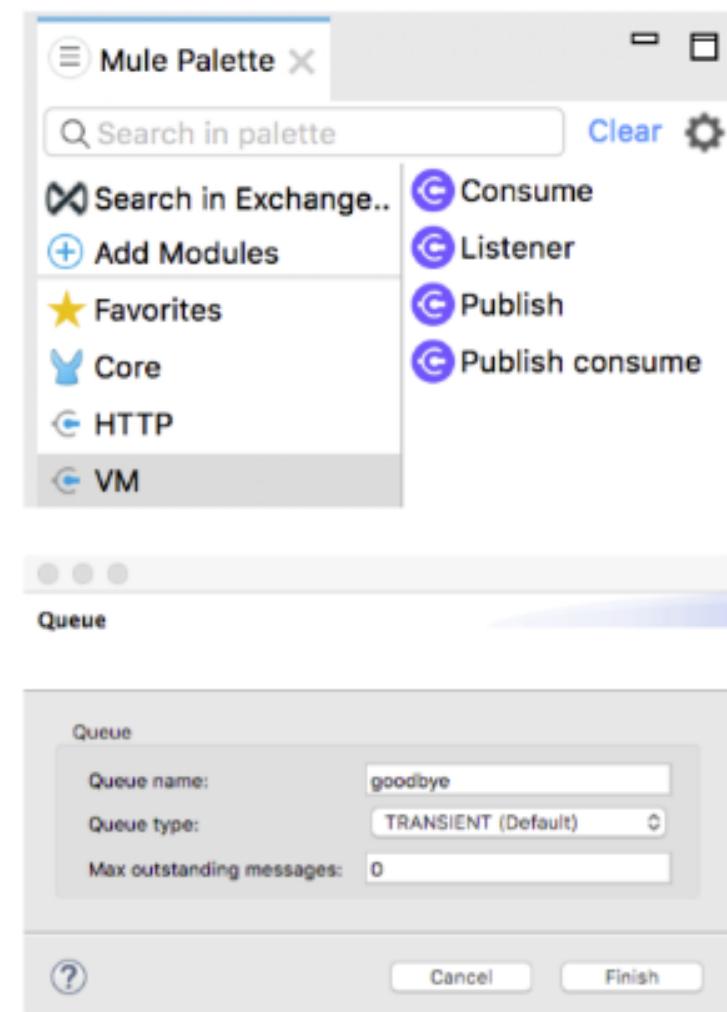
Passing messages between flows using asynchronous queues



- When using Flow Reference, events are passed synchronously between flows
- You may want to pass events asynchronously between flows to
 - Achieve higher levels of parallelism in specific stages of processing
 - Allow for more-specific tuning of areas within a flow's architecture
 - Distribute work across a cluster
 - Communicate with another application running in the same Mule domain
 - Domains will be explained later this module
 - Implement simple queueing that does not justify a full JMS broker
 - JMS is covered in Module 12
- This can be accomplished using the **VM connector**

Using the VM connector

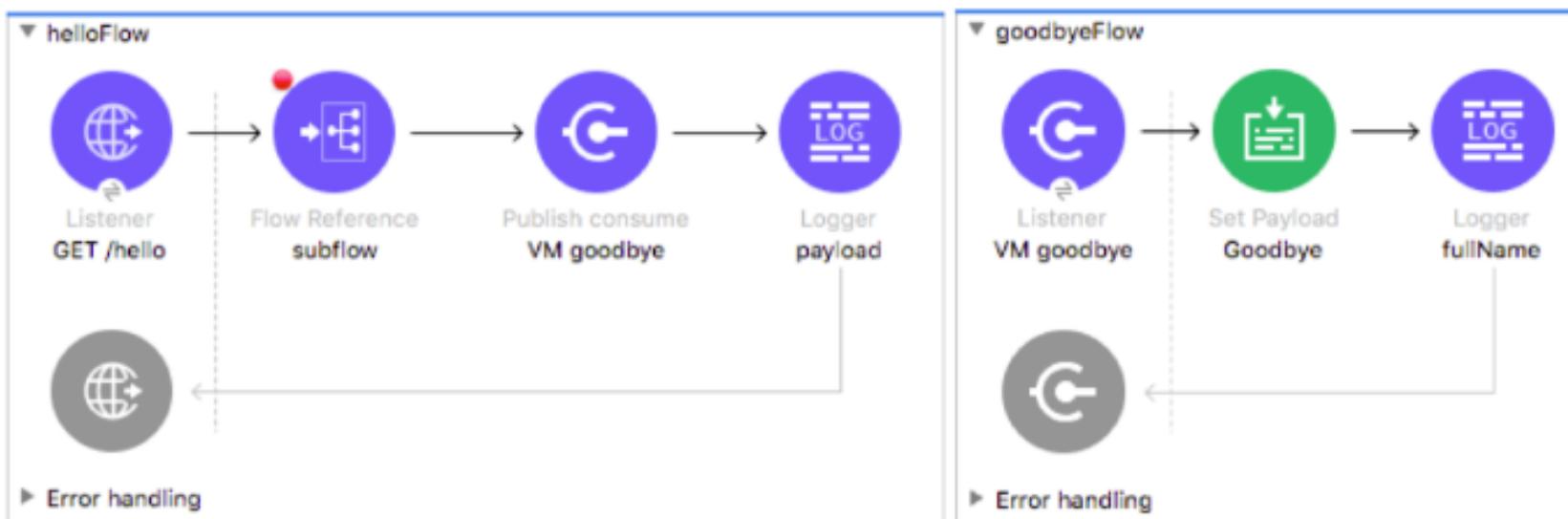
- Use the connector for intra and inter application communication through asynchronous queues
- Add the VM module to the project
- Configure a global element configuration
 - Specify a queue name and type
 - Queues can be transient or persistent
 - By default, the connector uses in-memory queues
 - **Transient** queues are faster, but are lost in the case of a system crash
 - **Persistent** queues are slower but reliable
- Use operations to publish and/or consume messages



Walkthrough 7-2: Pass messages between flows using the VM connector



- Pass messages between flows using the VM connector
- Explore variable persistence with VM communication
- Publish content to a VM queue and then wait for a response
- Publish content to a VM queue without waiting for a response



Walkthrough 7-2: Pass messages between flows using the VM connector

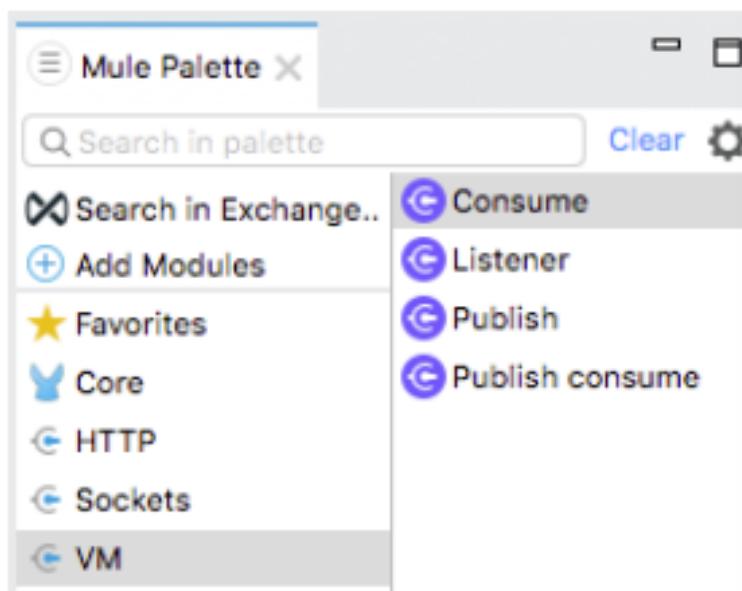
In this walkthrough, you pass messages between flows using asynchronous queues. You will:

- Pass messages between flows using the VM connector.
- Explore variable persistence with VM communication.
- Publish content to a VM queue and then wait for a response.
- Publish content to a VM queue without waiting for a response.

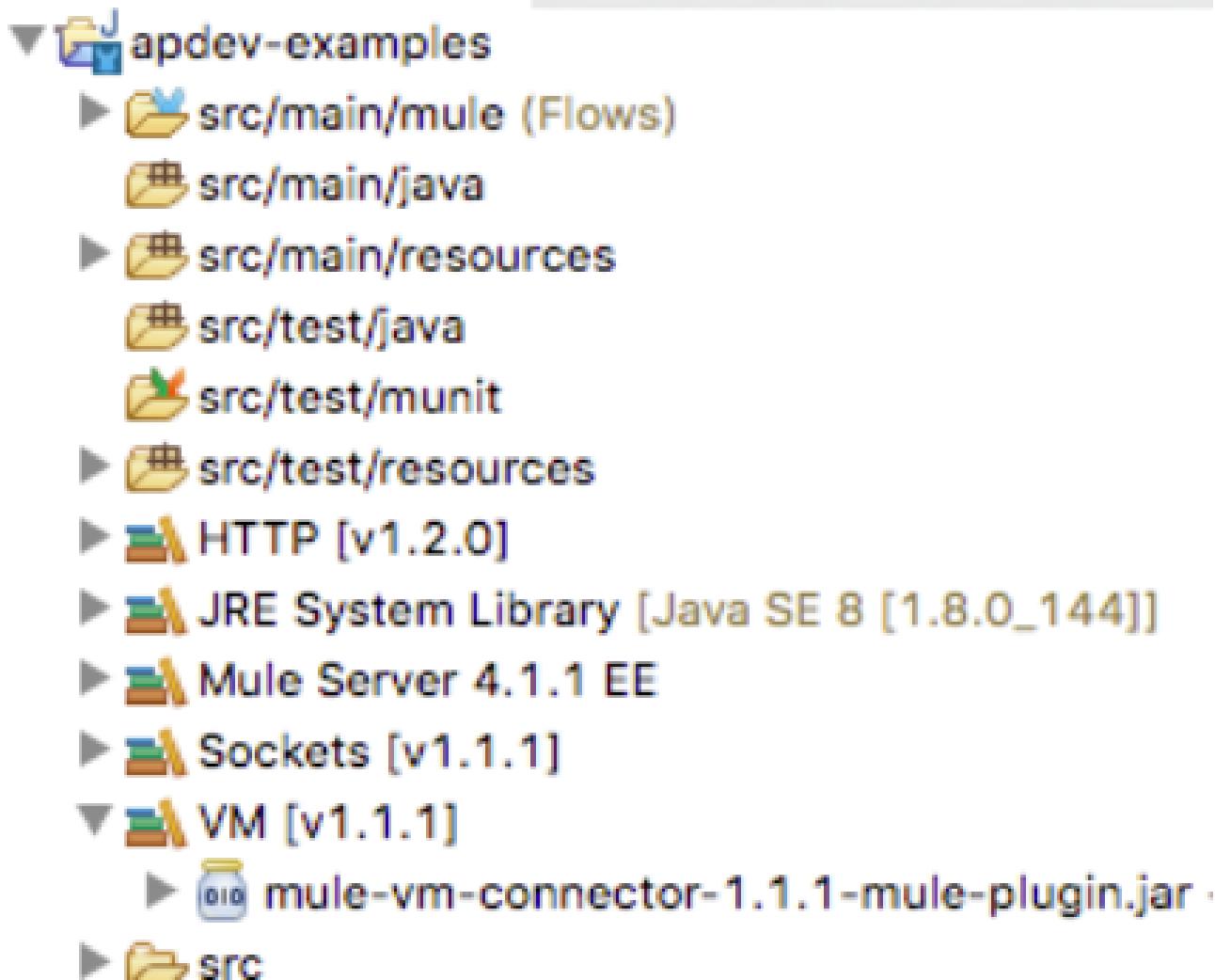


Add the VM module to the project

1. Return to apdev-examples.xml.
2. In the Mule Palette, select Add Modules.
3. Select the VM connector in the right side of the Mule Palette and drag and drop it into the left side.

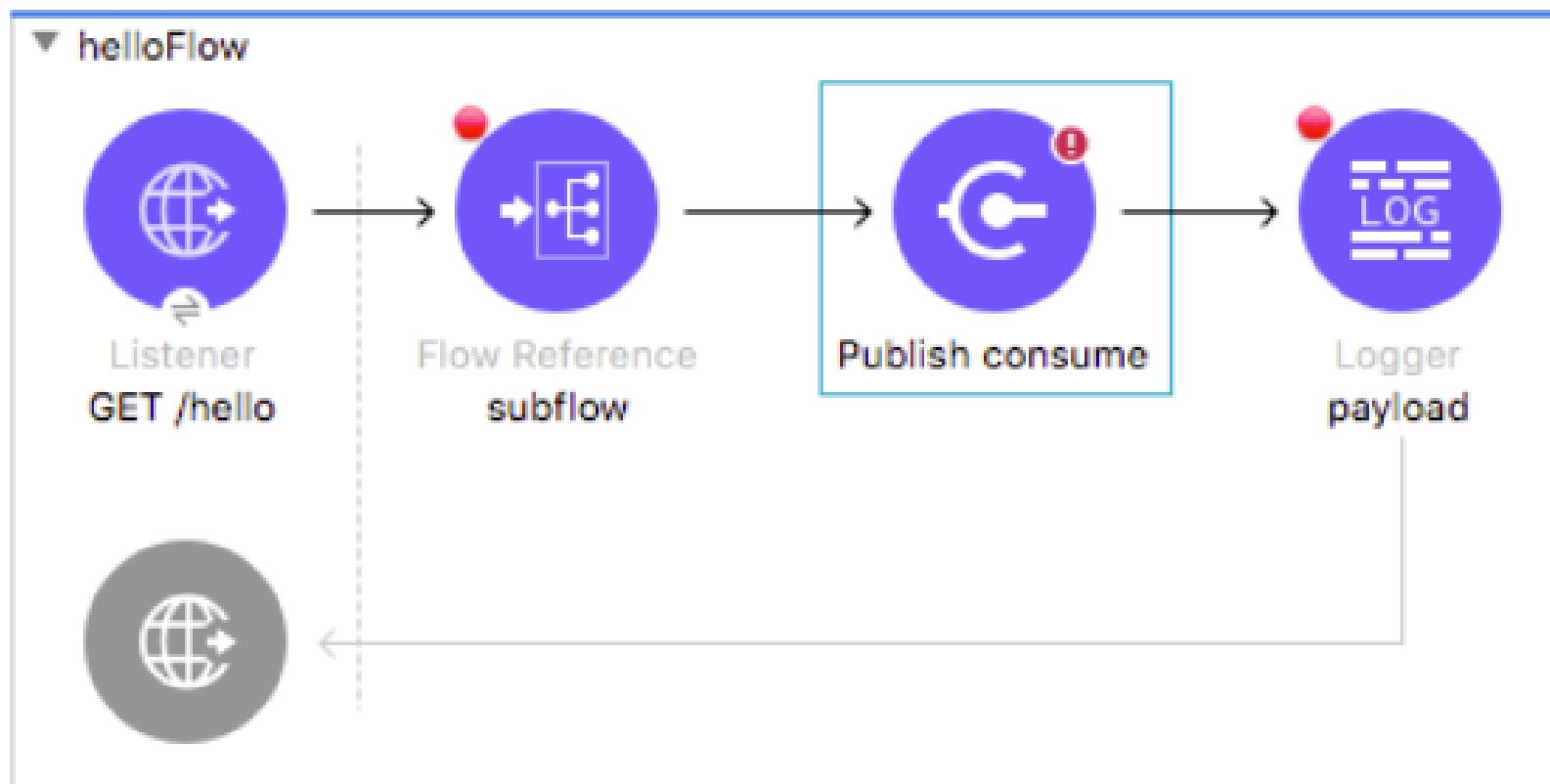


4. In the Project Explorer, locate the JAR file for the VM connector.
-

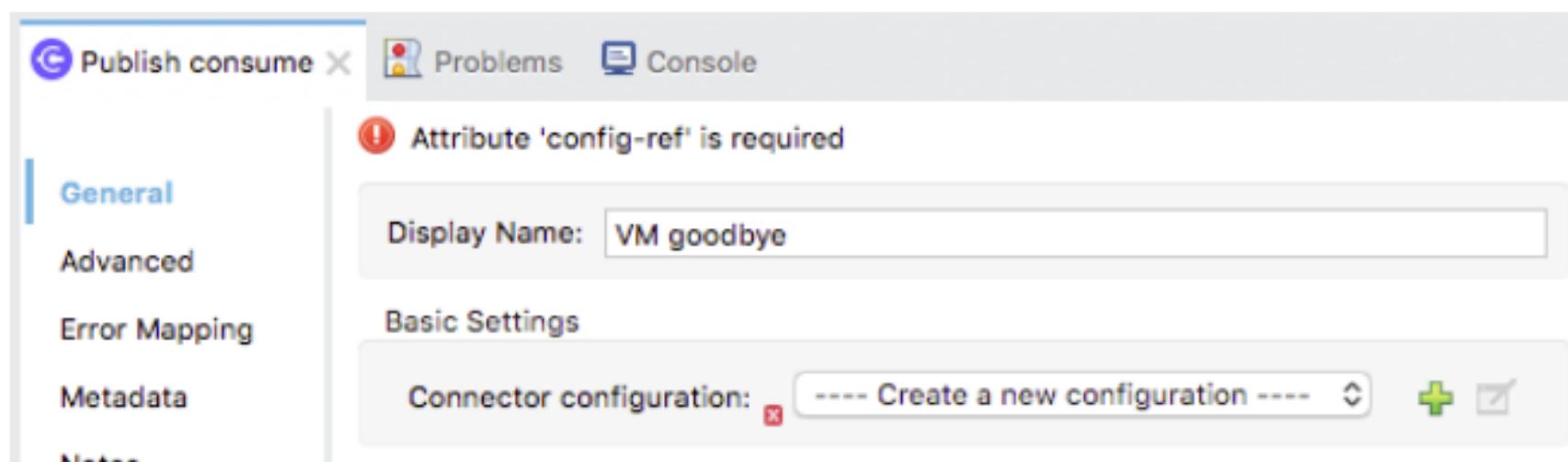


Add a VM Publish Consume operation

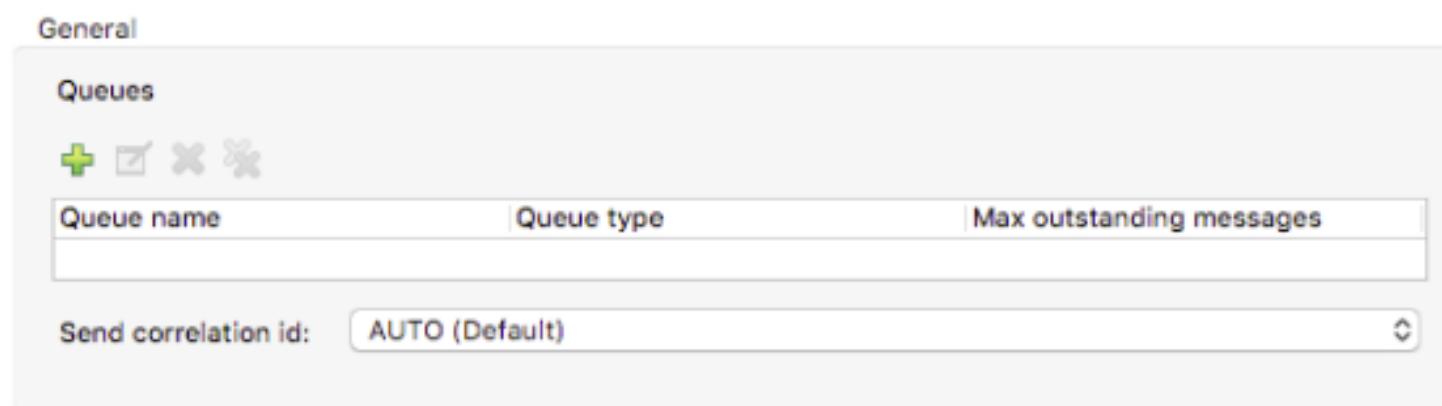
5. In helloFlow, delete the privateFlow Flow Reference.
6. Select VM in the Mule Palette.
7. Select the Publish consume operation and drag and drop it before the Logger in helloFlow.



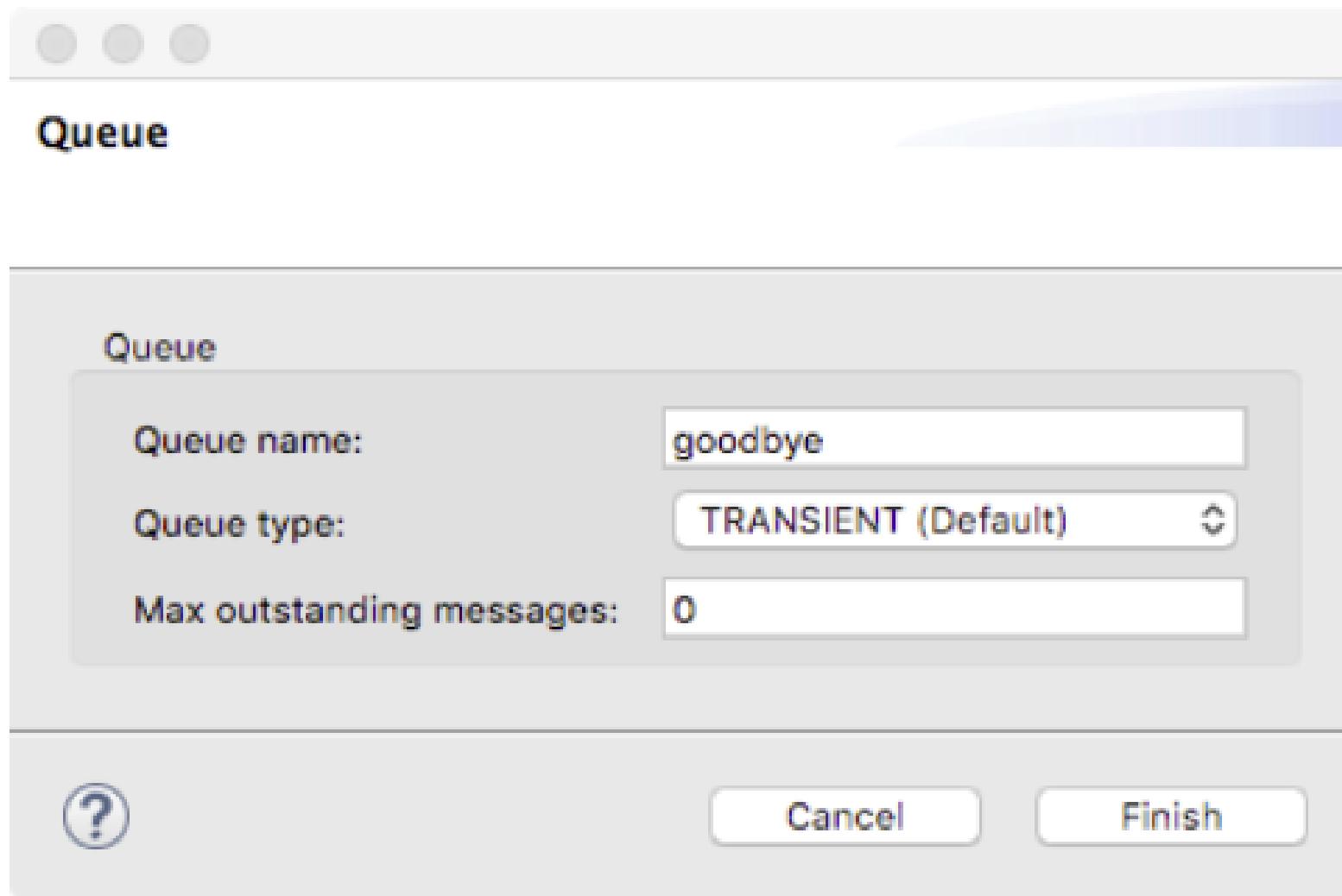
8. In the Publish consume properties view, change the display name to VM goodbye.



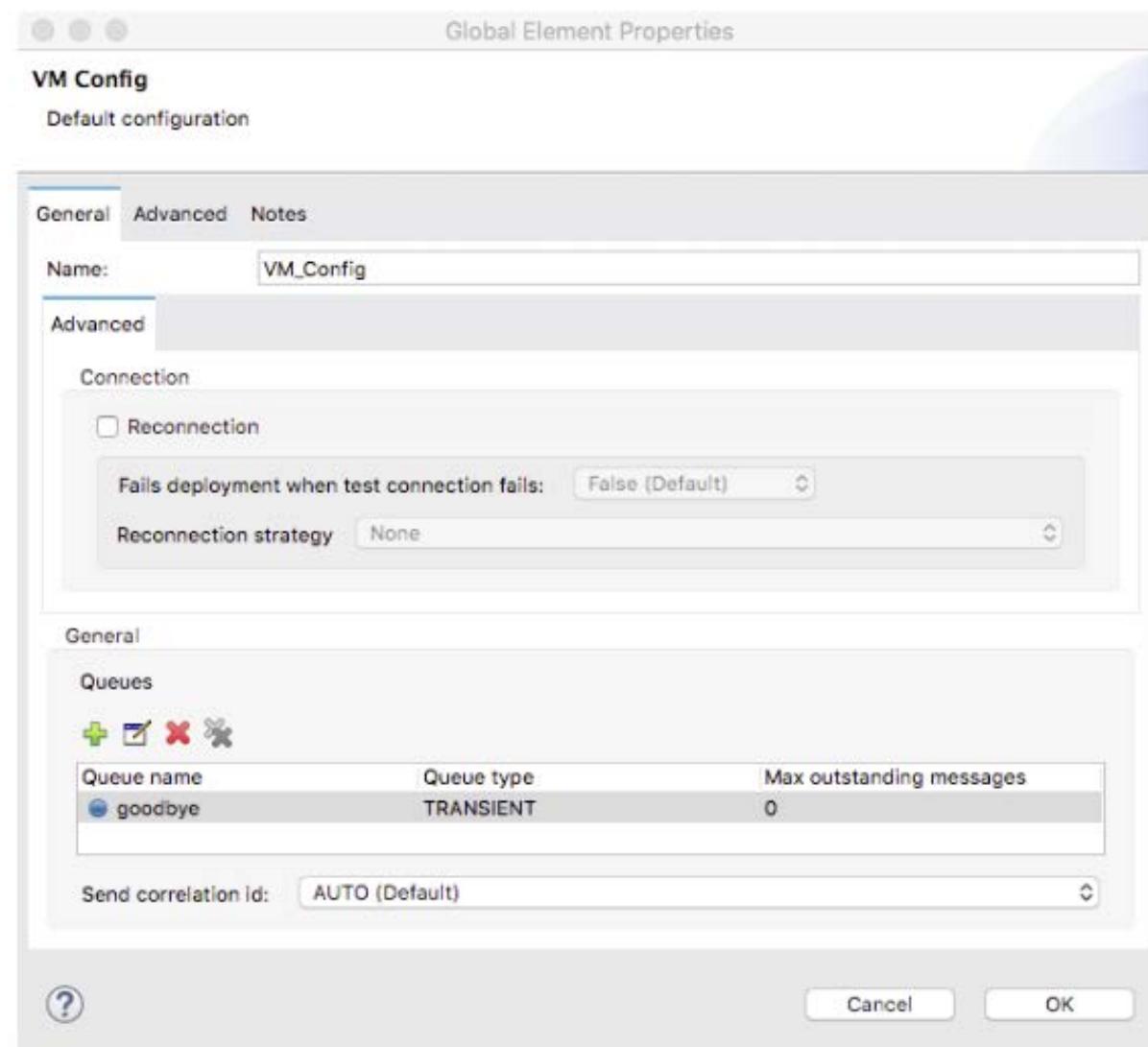
9. Click the Add button next to connector configuration.
10. In the Global Element Properties dialog box, click the Add Queue button.



11. In the Queue dialog box, set the queue name to goodbye and click Finish.



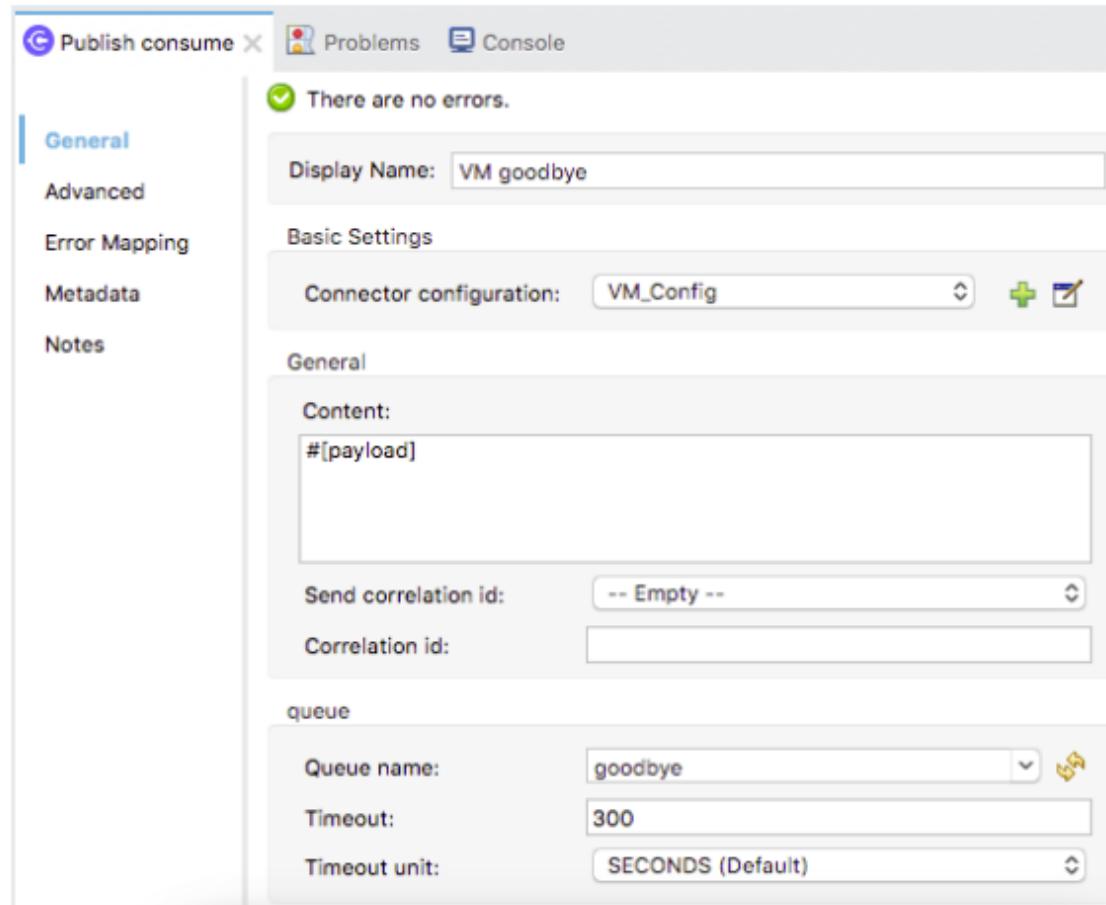
12. In the Global Element Properties dialog box, click OK.



13. In the VM goodbye Publish consume properties view, set the queue name to goodbye.

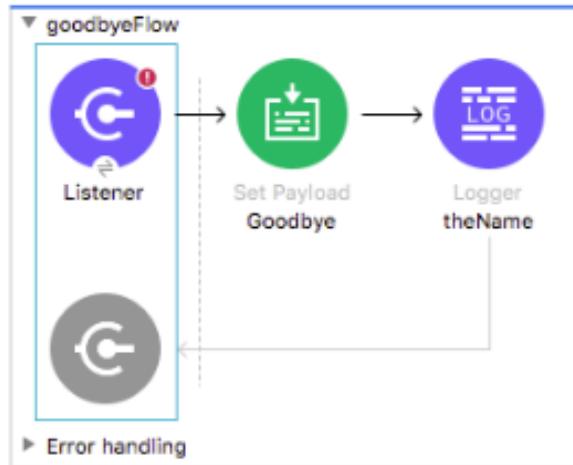
Note: If you do not see the queue name in the drop-down menu, click the refresh icon for it to be populated and then set it.

14. Set the timeout to 300 seconds for debugging purposes.



Add a VM Listener

15. Expand `goodbyeFlow` and delete its HTTP Listener.
16. Locate the Listener operation for the VM connector in the right side of the Mule Palette and drag and drop it in the source section of `goodbyeFlow`.



17. In the VM Listener properties view, change the display name to VM goodbye.
18. Set the connector configuration to the existing VM_Config

19. Set the queue name to goodbye.

The screenshot shows the Apache Camel Listener configuration interface. The left sidebar has tabs: General (selected), Redelivery, Advanced, Metadata, and Notes. The main area has tabs: Listener (selected), Problems, and Console. A message says "There are no errors." Under "Basic Settings", "Display Name" is "VM goodbye". Under "General", "Connector configuration" is "VM_Config" with a dropdown, a plus sign icon, and a checkmark icon. Under "queue", "Queue name" is "goodbye" with a dropdown and a refresh icon. "Timeout" is "5" and "Timeout unit" is "SECONDS (Default)".

Listener X Problems Console

General Redelivery Advanced Metadata Notes

There are no errors.

Display Name: VM goodbye

Basic Settings

Connector configuration: VM_Config

General

Number of consumers: 4

queue

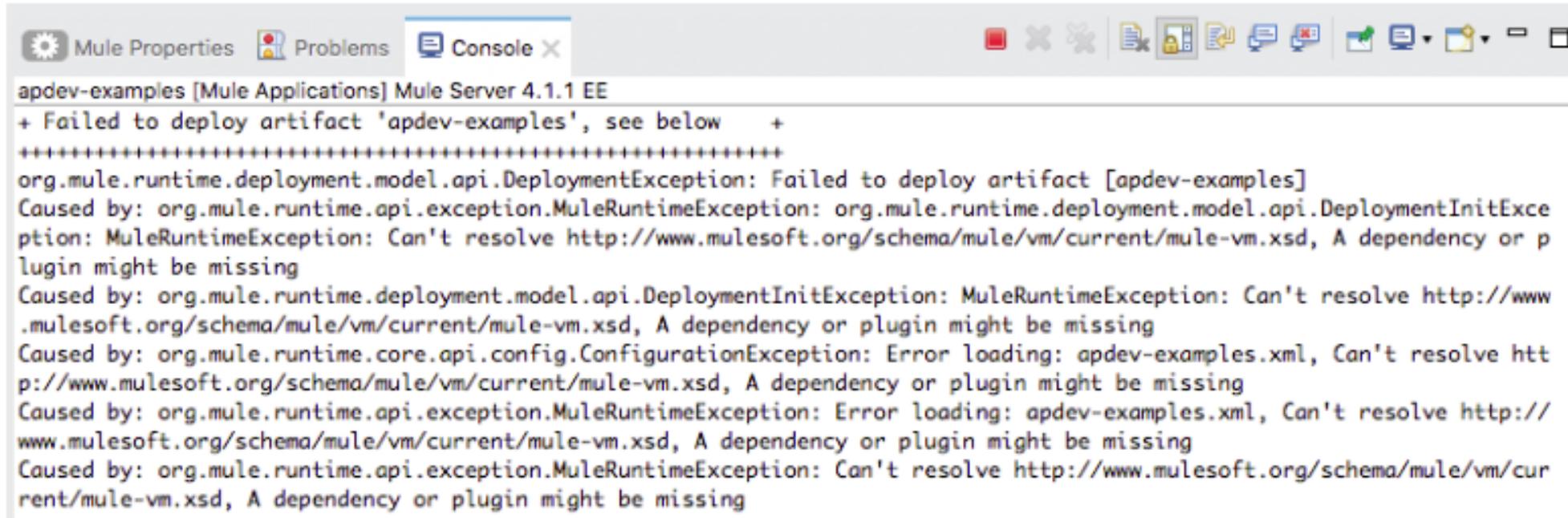
Queue name: goodbye

Timeout: 5

Timeout unit: SECONDS (Default)

Debug the application

20. Save the file to redeploy the project in debug mode; you should get an error that a dependency is missing.

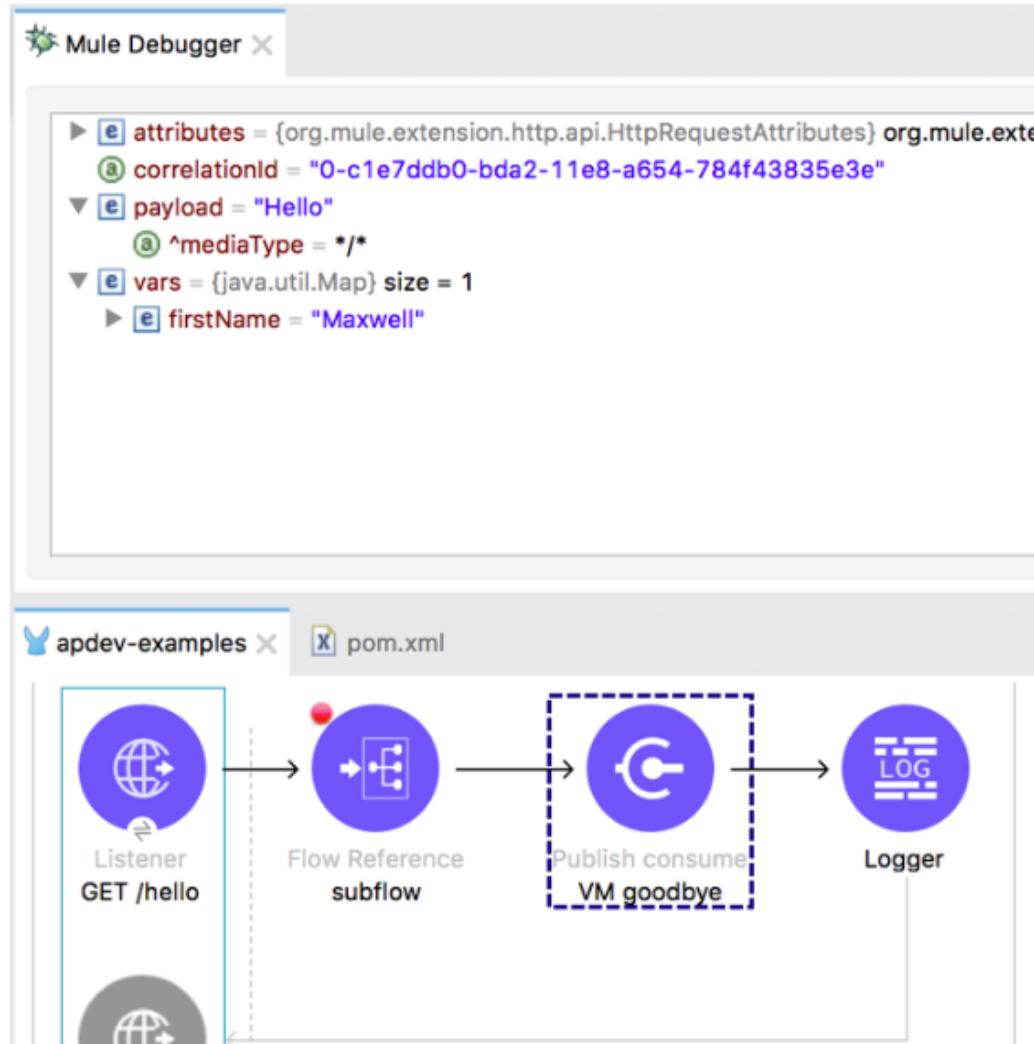


The screenshot shows the Mule Studio interface with the 'Console' tab selected. The output window displays deployment errors for a project named 'apdev-examples'. The errors indicate that the application failed to deploy due to missing dependencies, specifically the schema file 'mule-vm.xsd' from the MuleSoft website. The stack trace shows multiple levels of exception nesting, all pointing to the same schema resolution issue.

```
apdev-examples [Mule Applications] Mule Server 4.1.1 EE
+ Failed to deploy artifact 'apdev-examples', see below +
=====
org.mule.runtime.deployment.model.api.DeploymentException: Failed to deploy artifact [apdev-examples]
Caused by: org.mule.runtime.api.exception.MuleRuntimeException: org.mule.runtime.deployment.model.api.DeploymentInitException: MuleRuntimeException: Can't resolve http://www.mulesoft.org/schema/mule/vm/current/mule-vm.xsd, A dependency or plugin might be missing
Caused by: org.mule.runtime.deployment.model.api.DeploymentInitException: MuleRuntimeException: Can't resolve http://www.mulesoft.org/schema/mule/vm/current/mule-vm.xsd, A dependency or plugin might be missing
Caused by: org.mule.runtime.core.api.config.ConfigurationException: Error loading: apdev-examples.xml, Can't resolve http://www.mulesoft.org/schema/mule/vm/current/mule-vm.xsd, A dependency or plugin might be missing
Caused by: org.mule.runtime.api.exception.MuleRuntimeException: Error loading: apdev-examples.xml, Can't resolve http://www.mulesoft.org/schema/mule/vm/current/mule-vm.xsd, A dependency or plugin might be missing
Caused by: org.mule.runtime.api.exception.MuleRuntimeException: Can't resolve http://www.mulesoft.org/schema/mule/vm/current/mule-vm.xsd, A dependency or plugin might be missing
```

21. Stop the project.
22. Debug the project; the application should successfully deploy.
23. In Advanced REST Client, send the same request.
24. In the Mule Debugger, step through the application to the VM Publish consume.

25. Look at the payload, attributes, and variables.



26. Step into goodbyeFlow.

27. Look at the payload, attributes, and variables (or lack thereof).

The top window is titled "Mule Debugger" and shows the following message details:

- attributes = {org.mule.extensions.vm.api.VMMessageAttributes} org.mule.exten
- correlationId = "0-c1e7ddb0-bda2-11e8-a654-784f43835e3e"
- payload = "Hello"
 - ^mediaType = */*; charset=UTF-8
 - vars = {java.util.Map} size = 0

The bottom window is titled "apdev-examples" and shows a flow named "goodbyeFlow". The flow consists of three components connected sequentially:

- A purple "Listener" component.
- A green "Set Payload" component with a dashed border around it. Inside the component, the payload is set to "Goodbye".
- A purple "Logger" component with the log message "LOG" and the variable "fullName".

28. Step through the flow until the event returns to helloFlow.

29. Look at the payload, attributes, and variables.

Mule Debugger

```
▶ e attributes = {org.mule.extensions.vm.api.VMMessageAttributes} org.mule.exten  
③ correlationId = "0-c1e7ddb0-bda2-11e8-a654-784f43835e3e"  
▼ e payload = "GOODBYE Maxine"  
③ ^mediaType = application/java; charset=UTF-8  
▼ e vars = {java.util.Map} size = 1  
▶ e firstName = "Maxwell"
```

apdev-examples

```
graph LR; Listener[Listener<br/>GET /hello] --> Subflow((Flow Reference subflow)); Subflow --> Publish[Publish consume<br/>VM goodbye]; Publish --> Logger[Logger]
```

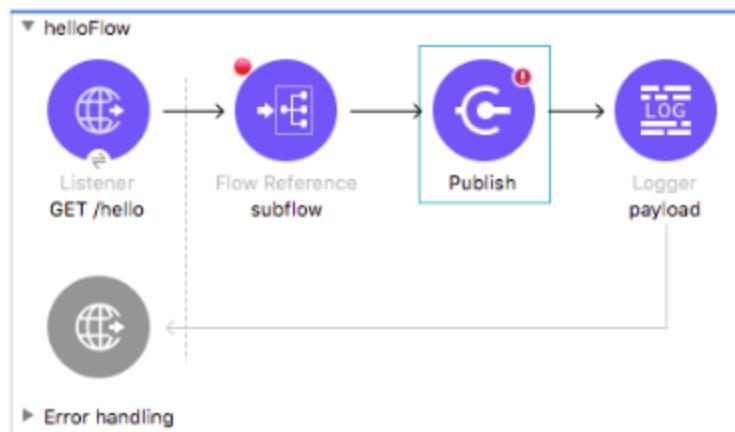
The diagram illustrates a Mule ESB application named 'apdev-examples'. It consists of four main components connected sequentially: 1) A 'Listener' component with the label 'GET /hello'. 2) A 'Flow Reference subflow' component, indicated by a dashed arrow pointing to it. 3) A 'Publish consume VM goodbye' component. 4) A 'Logger' component, which is highlighted with a dashed blue border. The flow starts at the Listener, goes to the subflow, then to the publish/consume component, and finally to the logger.

30. Step through the rest of the application.

31. Switch perspectives.

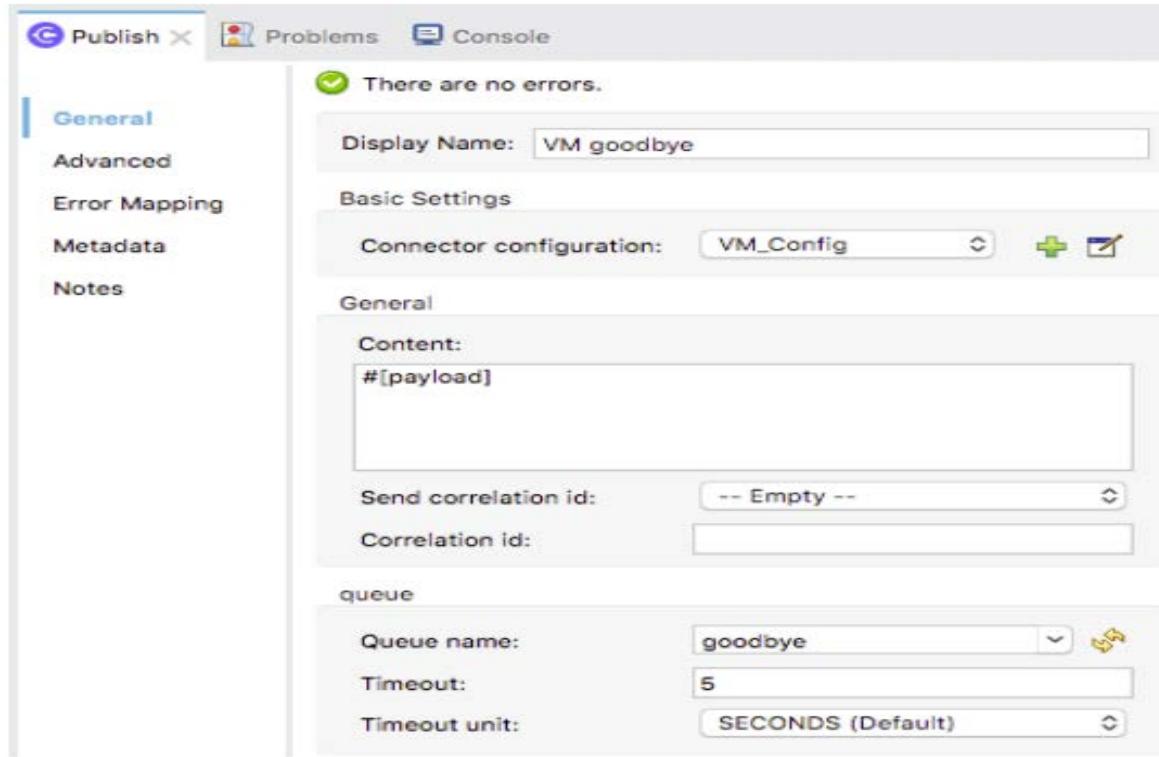
Change the VM Publish Consume operation to Publish

32. Delete the VM Publish consume operation in helloFlow.
33. Drag a VM Publish operation from the Mule Palette and drop it before the Logger.



34. In the Publish properties view, set the display name to VM goodbye.
35. Set the connector configuration to the existing VM_Config.

36. Set the queue name to goodbye.



Debug the application

37. Save the file to redeploy the project in debug mode.
38. In Advanced REST Client, send the same request.
39. In the Mule Debugger, step through the application to the VM Publish operation.
40. Step again; you should step to the Logger in helloFlow.

41. Look at the value of the payload.



42. Step again; you should see execution stopped in goodbyeFlow.

43. Step to the end of the application.

44. Return to Advanced REST Client; you should see a response of Hello – not GOODBYE.

200 Success 160792.30 ms



Hello

45. Return to Anypoint Studio and switch perspectives.

46. Stop the project.

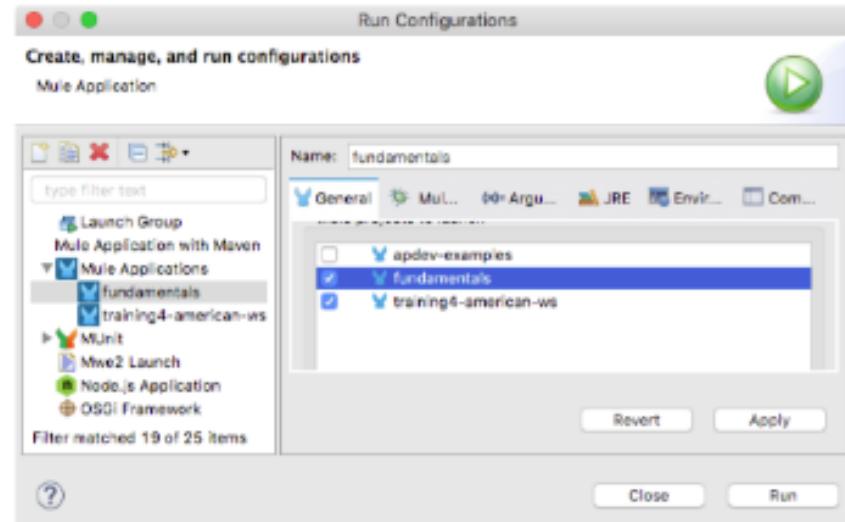
Organizing Mule application files



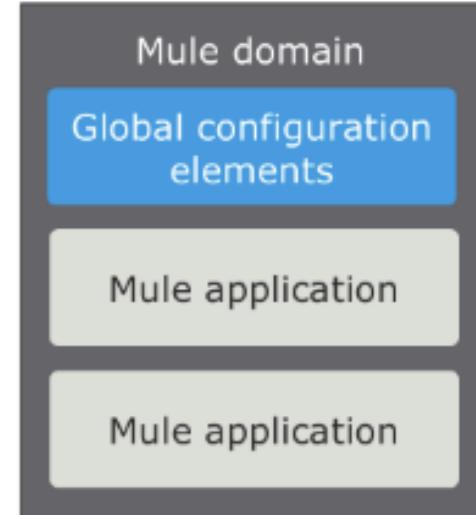
- Just as we separated flows into multiple flows, we also want to separate configuration files into multiple configuration files
- Monolithic files are difficult to read and maintain
- Separating an application into multiple configuration files makes code
 - Easier to read
 - Easier to work with
 - Easier to test
 - More maintainable

- If you reference global elements in one file that are defined in various, unrelated files
 - It can be confusing
 - It makes it hard to find them
- A good solution is to put most global elements in one config file
 - All the rest of the files reference them
 - If a global element is specific to and only used in one file, it can make sense to keep it in that file

- You are also not going to want all your flows in one application/project
- Separate functionality into multiple applications to
 - Allow managing and monitoring of them as separate entities
 - Use different, incompatible JAR files
- Run more than one application at a time in Anypoint Studio by creating a run configuration

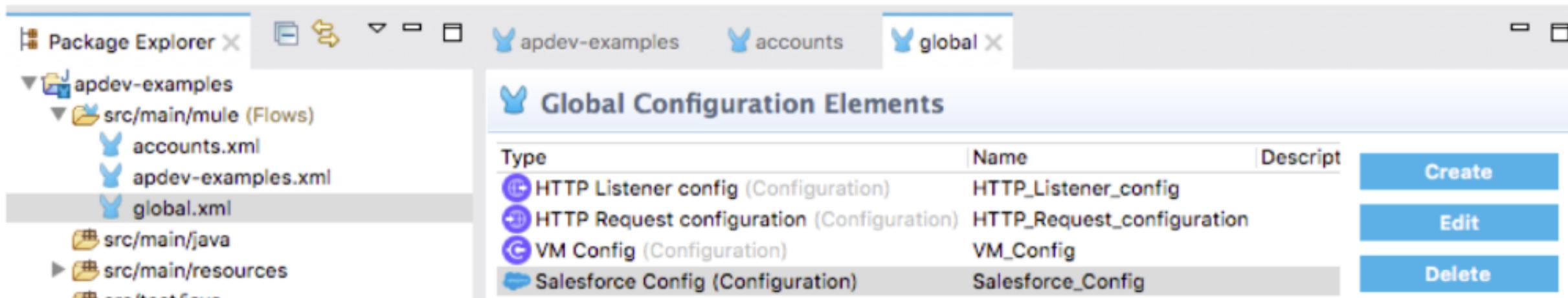


- A **domain project** can be used to share global configuration elements between applications, which lets you
 - Ensure consistency between applications upon any changes, as the configuration is only set in one place
 - Expose multiple services within the domain on the same port
 - Share the connection to persistent storage (Module 12)
 - Call flows in other applications using the VM connector
- Only available for customer-hosted Mule runtimes, not on CloudHub
- The general process
 - Create a Mule Domain Project and associate Mule applications with a domain
 - Add global element configurations to the domain project



Walkthrough 7-3: Encapsulate global elements in a separate configuration file

- Create a new configuration file with an endpoint that uses an existing global element
- Create a configuration file `global.xml` for just global elements
- Move the existing global elements to `global.xml`
- Create a new global element in `global.xml` and configure a new connector to use it



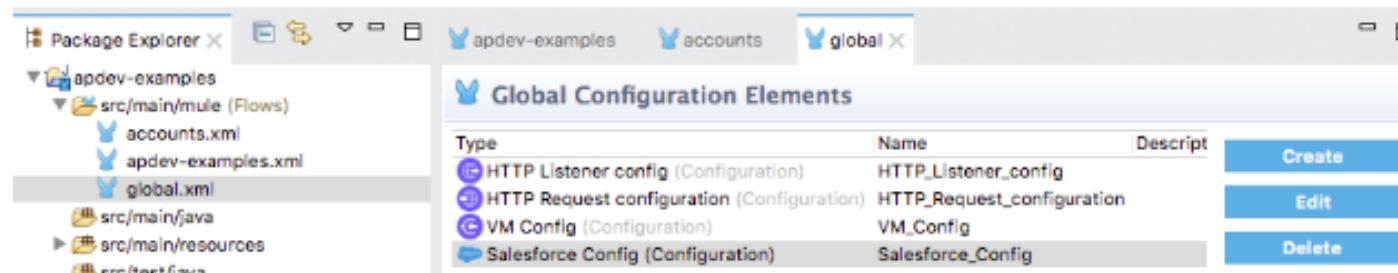
The screenshot shows the Mule Studio interface. On the left, the Package Explorer view displays the project structure under the `apdev-examples` folder. It includes `src/main/mule` (Flows) containing `accounts.xml`, `apdev-examples.xml`, and `global.xml`. Other sections like `src/main/java`, `src/main/resources`, and `src/test/java` are also visible. On the right, the Global Configuration Elements view is open, showing a table of existing global configurations:

Type	Name	Description
HTTP Listener config (Configuration)	HTTP_Listener_config	Create
HTTP Request configuration (Configuration)	HTTP_Request_configuration	Edit
VM Config (Configuration)	VM_Config	Delete
Salesforce Config (Configuration)	Salesforce_Config	Delete

Walkthrough 7-3: Encapsulate global elements in a separate configuration file

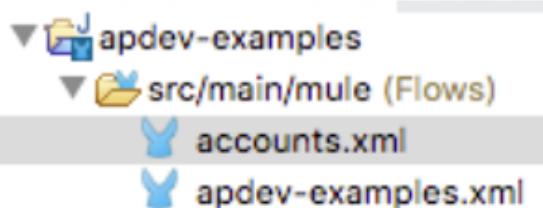
In this walkthrough, you refactor your apdev-examples project. You will:

- Create a new configuration file with an endpoint that uses an existing global element.
- Create a configuration file global.xml for just global elements.
- Move the existing global elements to global.xml.
- Create a new global element in global.xml and configure a new connector to use it.

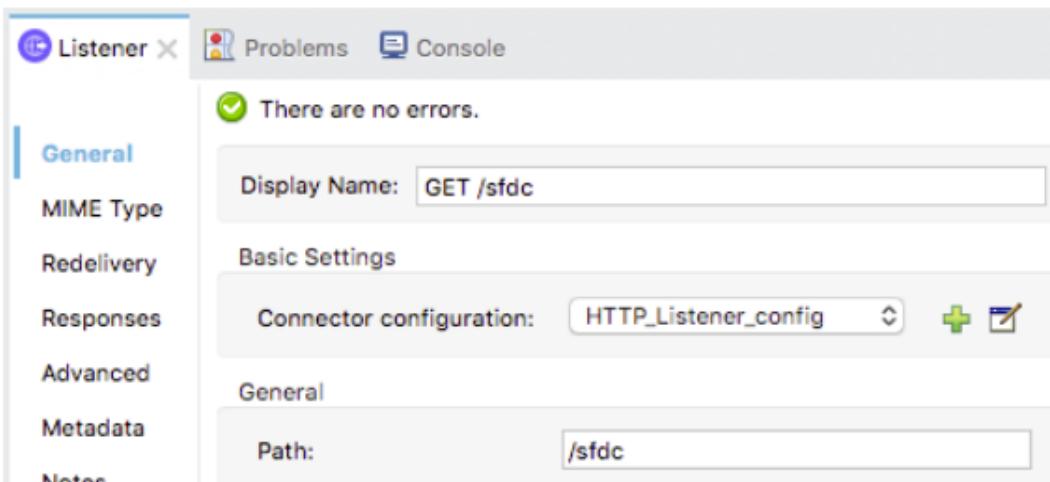


Create a new configuration file

1. Return to the apdev-examples project.
2. In the Package Explorer, right-click the project and select New > Mule Configuration File.
3. In the dialog box, set the name to accounts and click Finish.



4. Drag an HTTP Listener to the canvas from the Mule Palette.
5. In the Listener properties view, set the display name to GET /sfdc.
6. Set the connector configuration to the existing HTTP_Listener_config.
7. Set the path to /sfdc and the allowed methods to GET.



8. Add a Transform Message component to the flow.
9. In the Transform Message properties view, change the output type to application/json and set the expression to payload.

```
1 ⊕ %dw 2.0
2   output application/json
3   ---
4   payload
```

10. Change the name of the flow to getSFDAccounts.

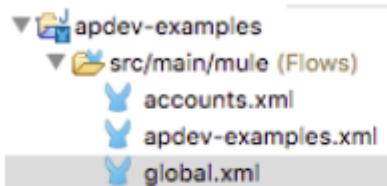


11. Switch to the Global Elements view; you should not see any global elements.

The screenshot shows the 'Global Configuration Elements' view in Mule Studio. At the top, there are tabs for 'apdev-examples' and '*accounts'. The '*accounts' tab is selected, showing a list of configuration elements. The list has columns for 'Type', 'Name', and 'Description'. Below the list are three buttons: 'Create' (blue), 'Edit' (grey), and 'Delete' (grey). The title bar also includes standard window controls for minimize and maximize.

Create a global configuration file

12. Create a new Mule configuration file called global.xml.



13. In global.xml, switch to the Configuration XML view.

14. Place some empty lines between the start and end mule tags.

The screenshot shows the content of the 'global.xml' file in Mule Studio. The code is as follows:

```
1 <?xml version="1.0" encoding="UTF-8"?>
2
3<mule xmlns="http://www.mulesoft.org/schema/mule/core" xmlns:doc="http://www.mule
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.mulesoft.org/schema/mule/core http://www.mules
6
7
8 </mule>
```

Move the existing global elements to the new global configuration file

15. Return to apdev-examples.xml.
16. Switch to the Global Elements view and see there are three configurations.

The screenshot shows the 'Global Configuration Elements' view in the Mule Studio interface. The title bar has tabs for 'apdev-examples', '*accounts', and '*global'. The '*global' tab is selected. The main area displays a table with three rows:

Type	Name	Description
HTTP Listener config (Configuration)	HTTP_Listener_config	
HTTP Request configuration (Configuration)	HTTP_Request_configuration	
VM Config (Configuration)	VM_Config	

On the right side of the table, there are three buttons: 'Create' (blue), 'Edit' (grey), and 'Delete' (grey).

17. Switch to the Configuration XML view.
18. Select and cut the three configuration elements defined before the flows.

The screenshot shows the Configuration XML view in the Mule Studio interface. The title bar has tabs for 'apdev-examples', '*accounts', and '*global'. The '*global' tab is selected. The main area displays the XML code:

```
<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns:vm="http://www.mulesoft.org/schema/mule/vm" xmlns:http="http://www.mulesoft.org/schema/mule/http"
      xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:doc="http://www.mulesoft.org/schema/mule/documentation" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.mulesoft.org/schema/mule http://www.mulesoft.org/schema/mule/3.6/mule.xsd
                        http://www.mulesoft.org/schema/mule/vm http://www.mulesoft.org/schema/mule/vm.xsd
                        http://www.mulesoft.org/schema/mule/http http://www.mulesoft.org/schema/mule/http.xsd">
    <http:listener-config name="HTTP_Listener_config" doc:name="HTTP Listener Configuration">
        <http:listener-connection host="0.0.0.0" port="8081" />
    </http:listener-config>
    <http:request-config name="HTTP_Request_configuration" doc:name="HTTP Request Configuration">
        <http:request-connection host="localhost" port="8081" />
    </http:request-config>
    <vm:config name="VM_Config" doc:name="VM Config" doc:id="c634cc4e-d75a-43d0-8f0d-41aa-a664-664c7c0b9073">
        <vm:queues>
            <vm:queue queueName="goodbye" />
        </vm:queues>
    </vm:config>
    <flow name="helloFlow" doc:id="056f3975-8f0d-41aa-a664-664c7c0b9073" >
        <http:listener doc:name="HTTP / hello" doc:id="45ae12nd-4f22-dd5h-9e">
```

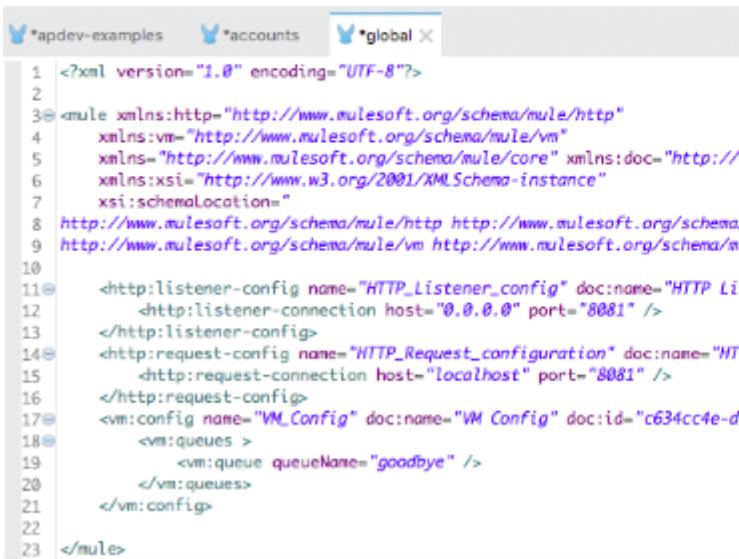
The XML code includes three configuration elements: 'HTTP_Listener_config', 'HTTP_Request_configuration', and 'VM_Config'. These three elements are highlighted with a blue selection box.

Note: If you delete the global elements from the Global Elements view instead, the config-ref values are also removed from the connector operations and you need to re-add them.

19. Return to the Message Flow view.

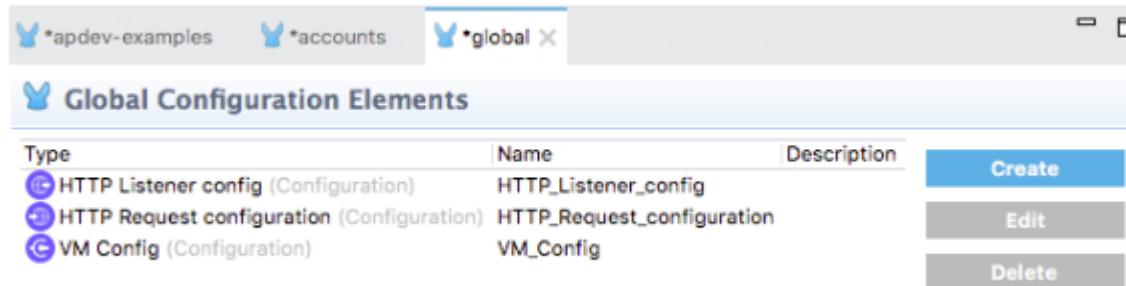
20. Return to global.xml.

21. Paste the global elements you cut to the clipboard between the start and end mule tags.



```
1 <?xml version="1.0" encoding="UTF-8"?>
2
3@<mule xmlns:http="http://www.mulesoft.org/schema/mule/http"
4   xmlns:vm="http://www.mulesoft.org/schema/mule/vm"
5   xmlns="http://www.mulesoft.org/schema/mule/core" xmlns:doc="http://
6   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
7   xsi:schemaLocation="
8     http://www.mulesoft.org/schema/mule/http http://www.mulesoft.org/schema/
9     http://www.mulesoft.org/schema/mule/vm http://www.mulesoft.org/schema/m
10
11@  <http:listener-config name="HTTP_Listener_config" doc:name="HTTP Li
12    <http:listener-connection host="0.0.0.0" port="8081" />
13  </http:listener-config>
14@  <http:request-config name="HTTP_Request_configuration" doc:name="HT
15    <http:request-connection host="localhost" port="8081" />
16  </http:request-config>
17@  <vm:config name="VM_Config" doc:name="VM Config" doc:id="c634cc4e-d
18@    <vm:queues >
19      <vm:queue queueName="goodbye" />
20    </vm:queues>
21  </vm:config>
22
23</mule>
```

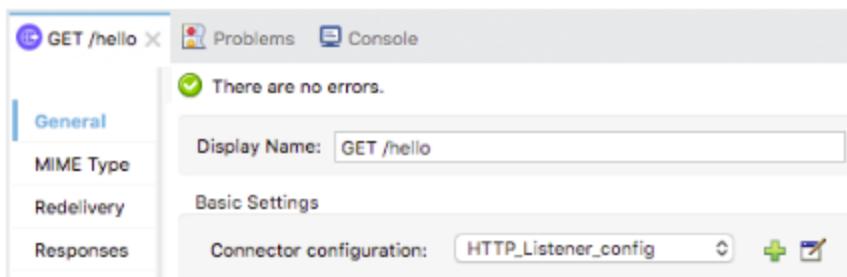
22. Switch to the Global Elements view; you should see the three configurations.



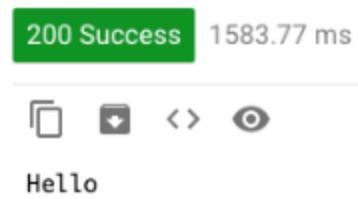
Type	Name	Description	Create
HTTP Listener config (Configuration)	HTTP_Listener_config		Edit
HTTP Request configuration (Configuration)	HTTP_Request_configuration		Delete
VM Config (Configuration)	VM_Config		

Test the application

23. Return to apdev-examples.xml.
24. Select the GET /hello HTTP Listener in helloFlow; the connector configuration should still be set to HTTP_Listener_config, which is now defined in global.xml.

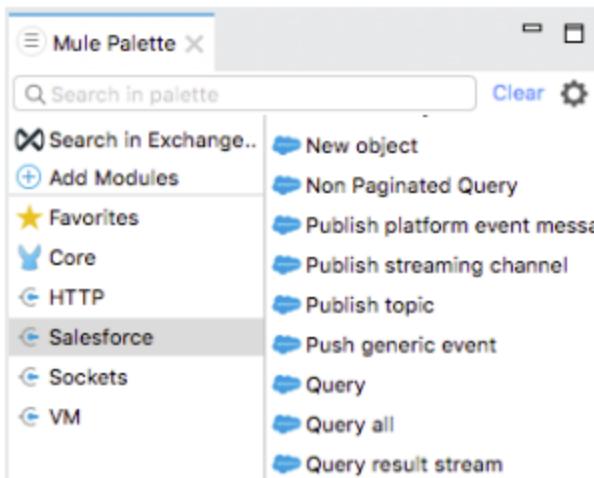


25. Run the project.
26. In Advanced REST Client, send the same request; you should still get a response of Hello.



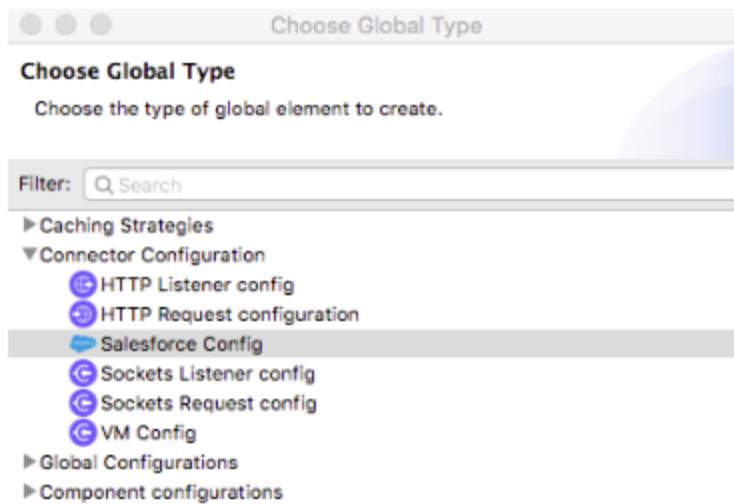
Create a new global element for the Salesforce component in global.xml

27. Return to apdev-examples.xml.
28. In the Mule Palette, select Add Modules.
29. Select the Salesforce connector in the right side of the Mule Palette and drag and drop it into the left side.

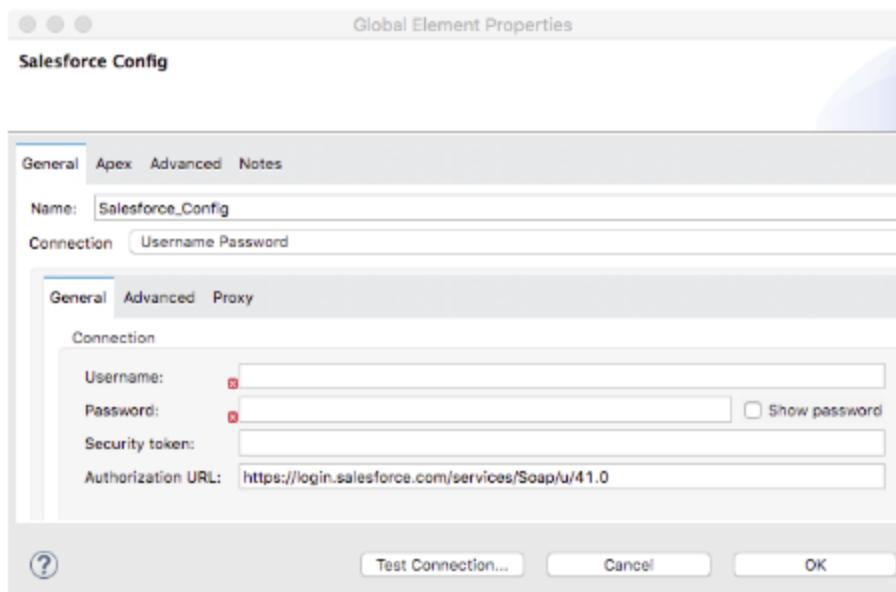


30. Locate the new Salesforce JAR in the project.
31. Return to global.xml.
32. Click Create.

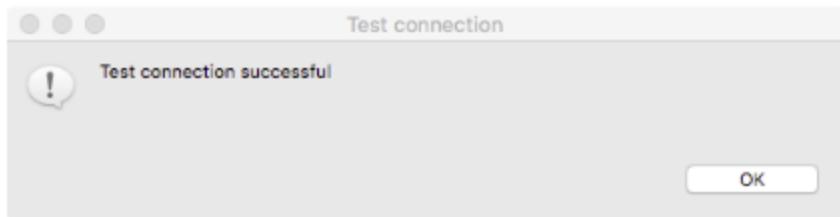
33. In the Choose Global Type dialog box, select Connector Configuration > Salesforce Config and click OK.



34. In the Global Element Properties dialog box enter your Salesforce username, password, and security token.



35. Click Test Connection; your connection should be successful.



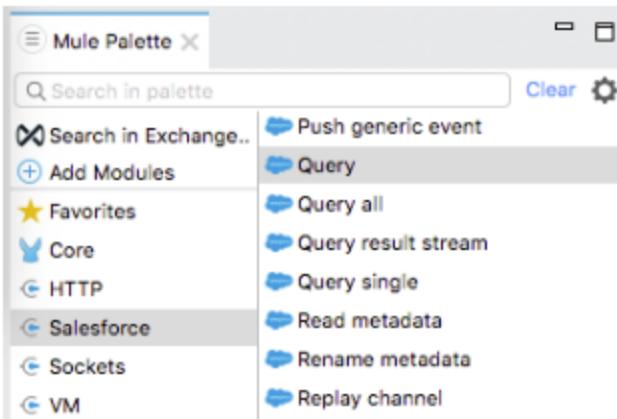
36. In the Test connection dialog box, click OK.

37. In the Global Element Properties dialog box, click OK; you should now see the new configuration in global.

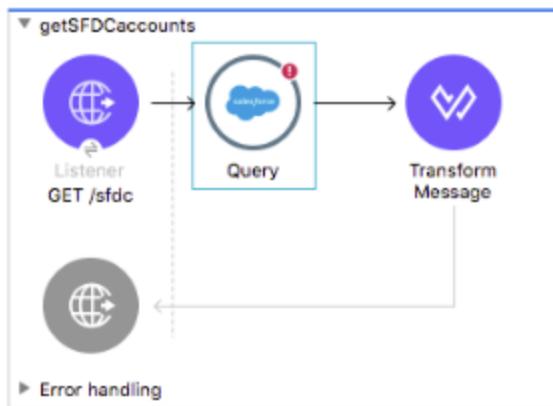
A screenshot of the 'Global Configuration Elements' dialog box. The top navigation bar has tabs for 'apdev-examples', 'accounts', and '*global', with '*global' being the active tab. Below the tabs is a section titled 'Global Configuration Elements'. A table lists four configuration types: 'HTTP Listener config (Configuration)', 'HTTP Request configuration (Configuration)', 'VM Config (Configuration)', and 'Salesforce Config (Configuration)'. Each row has columns for 'Type', 'Name', 'Description', and three buttons: 'Create', 'Edit', and 'Delete'. The 'Edit' button for the first row is highlighted in blue.

Add a new Salesforce operation that uses the global configuration

38. Return to the Message Flow view in accounts.xml.
39. In the Mule Palette, select Salesforce.

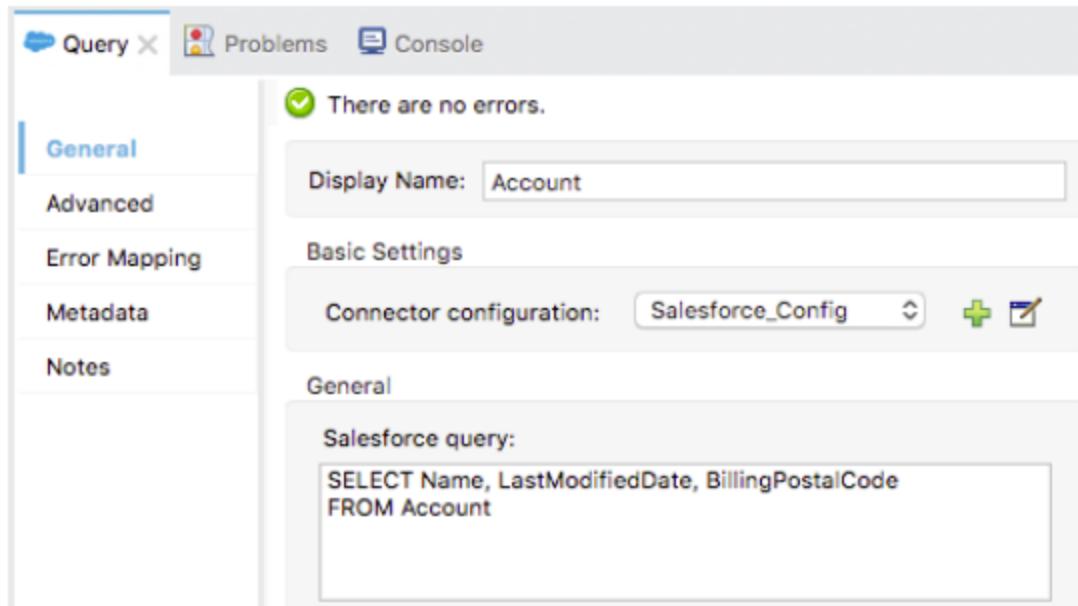


40. Locate the Query operation in the right side of the Mule Palette and drag and drop it before the Logger in getSFDCCaccounts.



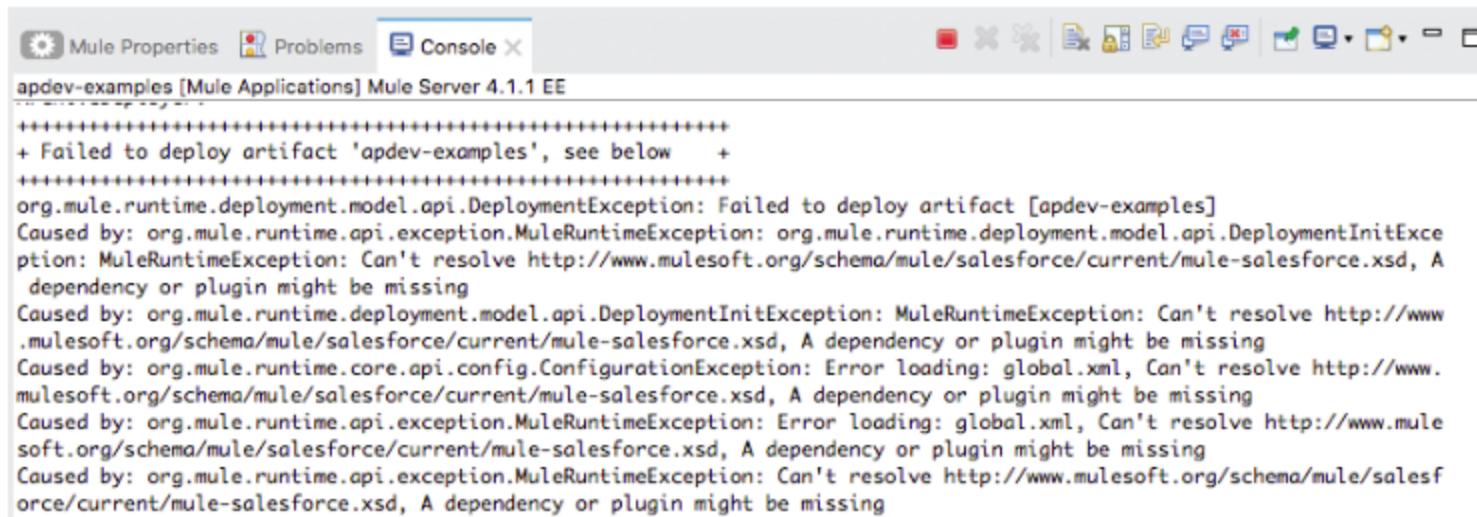
41. In the Query properties view, set the display name to Account.
42. Set the connector configuration to the existing Salesforce_Config.

43. Return to the course snippets.txt file and copy the Salesforce query.
44. Return to Anypoint Studio and paste the query in the Salesforce query section of the Query properties view.



Test the application

45. Save all the files; you should get a missing dependency error in the console.



The screenshot shows the Mule Studio interface with the 'Console' tab selected. The title bar indicates the project is 'apdev-examples [Mule Applications]' and the server is 'Mule Server 4.1.1 EE'. The console output displays a deployment error:

```
+ Failed to deploy artifact 'apdev-examples', see below +  
+-----+  
org.mule.runtime.deployment.model.api.DeploymentException: Failed to deploy artifact [apdev-examples]  
Caused by: org.mule.runtime.api.exception.MuleRuntimeException: org.mule.runtime.deployment.model.api.DeploymentInitException: MuleRuntimeException: Can't resolve http://www.mulesoft.org/schema/mule/salesforce/current/mule-salesforce.xsd, A dependency or plugin might be missing  
Caused by: org.mule.runtime.deployment.model.api.DeploymentInitException: MuleRuntimeException: Can't resolve http://www.mulesoft.org/schema/mule/salesforce/current/mule-salesforce.xsd, A dependency or plugin might be missing  
Caused by: org.mule.runtime.core.api.config.ConfigurationException: Error loading: global.xml, Can't resolve http://www.mulesoft.org/schema/mule/salesforce/current/mule-salesforce.xsd, A dependency or plugin might be missing  
Caused by: org.mule.runtime.api.exception.MuleRuntimeException: Error loading: global.xml, Can't resolve http://www.mulesoft.org/schema/mule/salesforce/current/mule-salesforce.xsd, A dependency or plugin might be missing  
Caused by: org.mule.runtime.api.exception.MuleRuntimeException: Can't resolve http://www.mulesoft.org/schema/mule/salesforce/current/mule-salesforce.xsd, A dependency or plugin might be missing
```

46. Stop the project.

47. Run the project.

48. In Advanced REST Client, send a request to <http://localhost:8081/sfdc>; you should get a list of the accounts in your Salesforce account.

Method Request URL
GET <http://localhost:8081/sfdc>

SEND ⋮

Parameters ⋮

200 OK 1757.84 ms DETAILS ⋮

[
{
 "LastModifiedDate": "2015-07-12T21:00:51.000Z",
 "BillingPostalCode": null,
 "Id": null,
 "type": "Account",
 "Name": "GenePoint"
},
{
 "LastModifiedDate": "2015-07-12T21:00:51.000Z",
 "BillingPostalCode": null
}]

Organizing and parameterizing application properties



- Provide an easier way to manage connector properties, credentials, and other configurations
- Replace static values
- Are defined in a configuration file
 - Either in a .yaml file or a .properties file
- Are implemented using property placeholders
- Can be encrypted
- Can be overridden by system properties when deploying to different environments

Defining application properties

- Create a YAML properties file in the src/main/resources folder

`config.yaml`

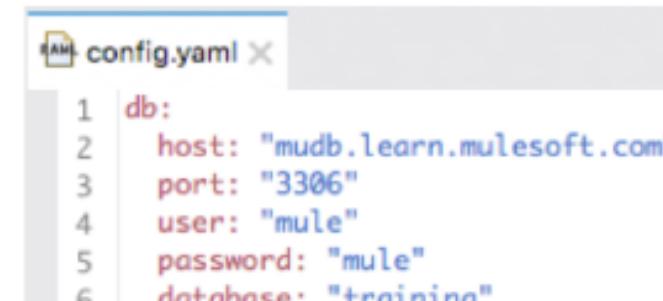
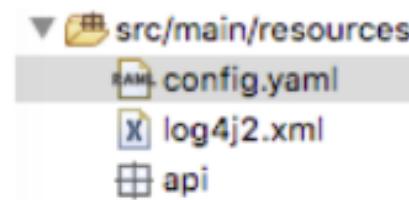
- Define properties in the hierarchical YAML file

`db:`

`port: "3306"`

`user: "mule"`

- Create a Configuration properties global element



```
1 db:  
2   host: "mudb.learn.mulesoft.com"  
3   port: "3306"  
4   user: "mule"  
5   password: "mule"  
6   database: "training"
```

Global Element Properties

Configuration properties

General Notes

Settings

File: config.yaml

Using application properties

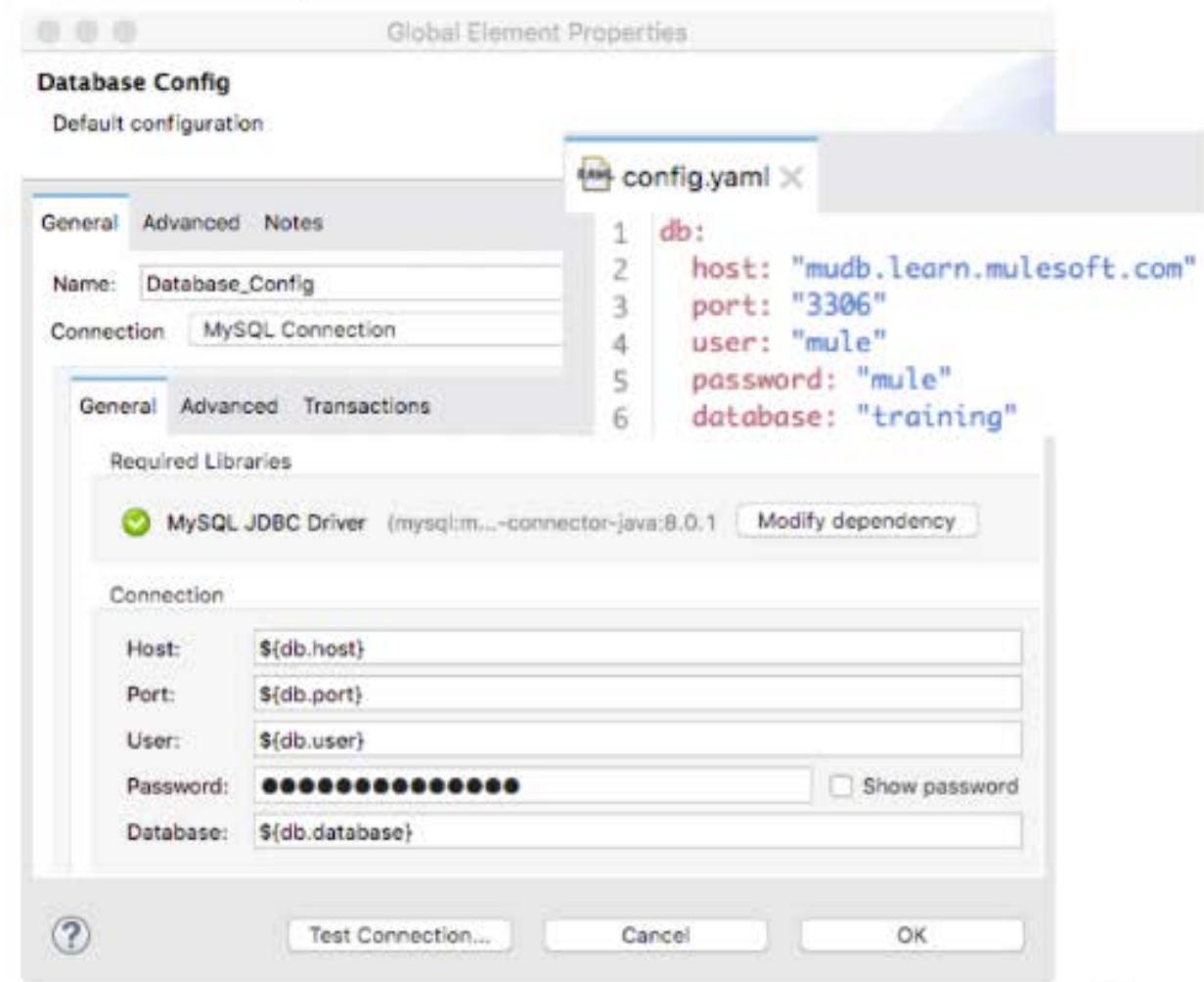


- In global element configurations and event processors

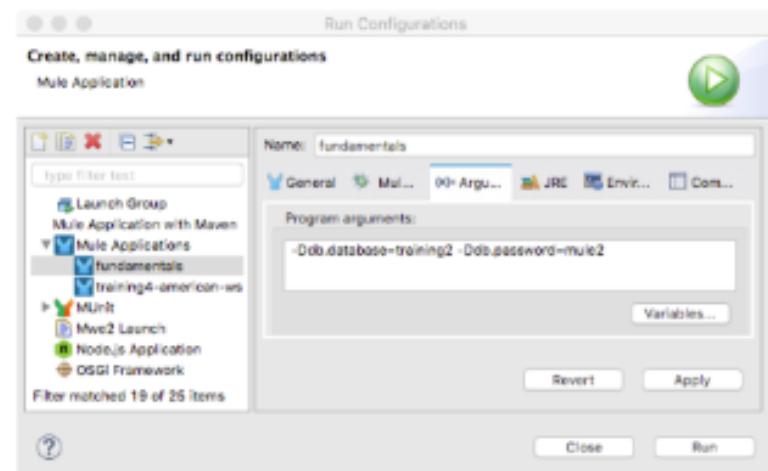
`#{db.port}`

- In DataWeave expressions

`{port: p('db.port')}`



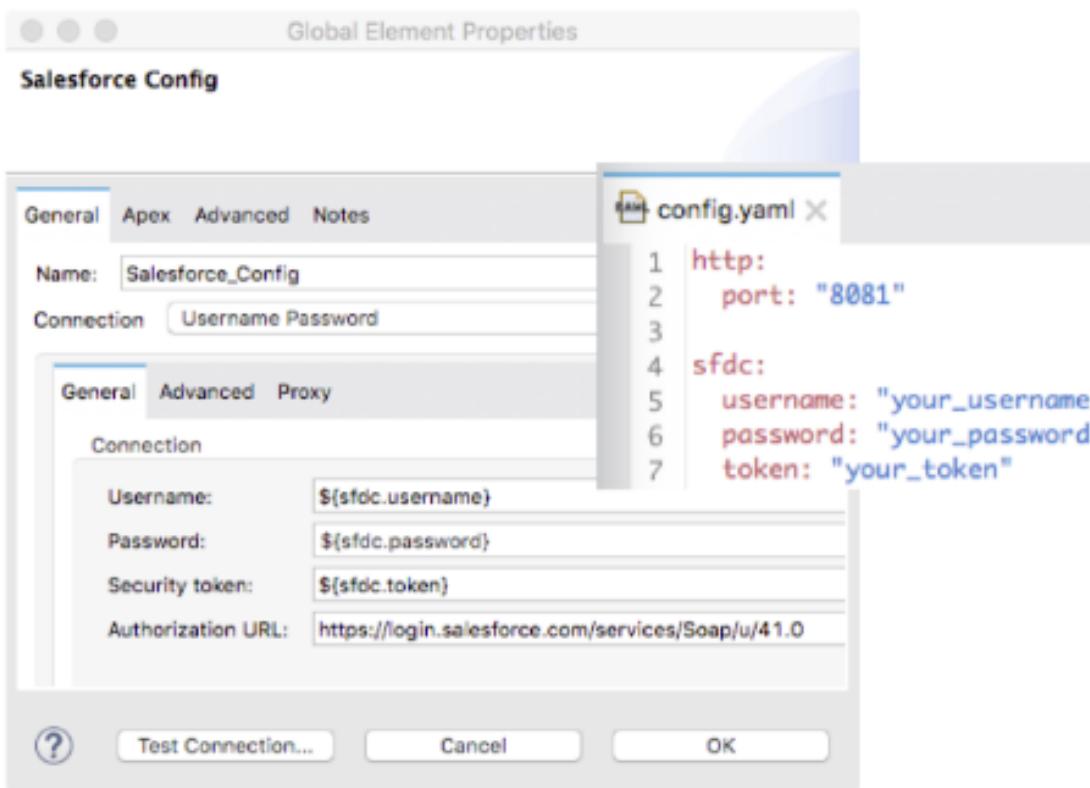
- Use **system properties** to override property values when deploying an application to a different environment (like dev, qa, production),
- Set system properties (JVM parameters) from
 - Anypoint Studio
in Run > Run Configurations > Arguments
 - The command line for a standalone Mule instance
`mule -M-Ddb.database=training2 -M-Ddb.password=mule2`



Walkthrough 7-4: Use property placeholders in connectors



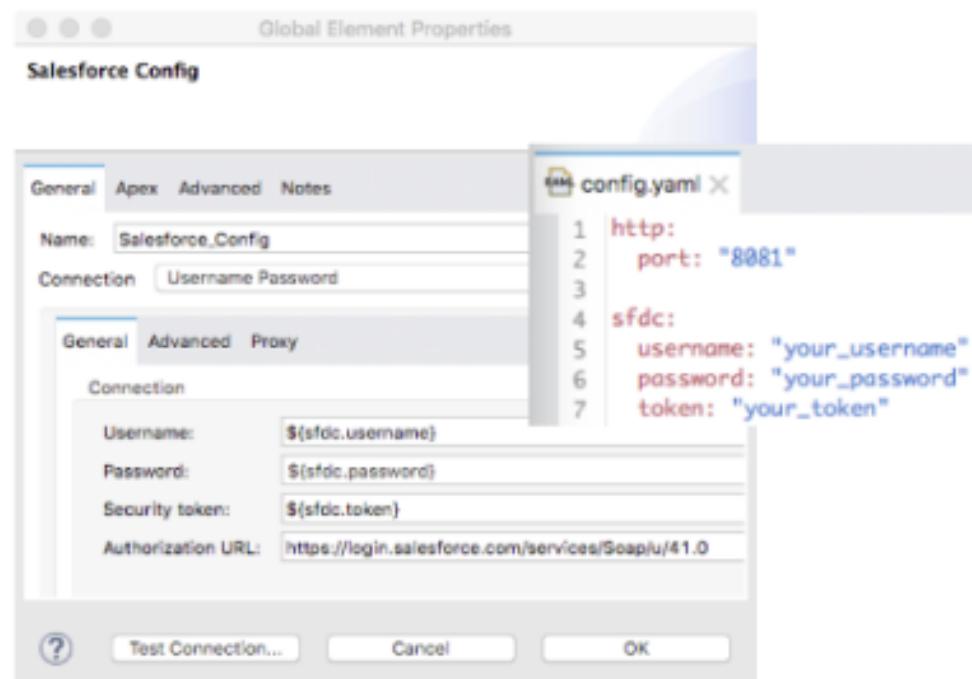
- Create a YAML properties file for an application
- Configure an application to use a properties file
- Define and use HTTP and Salesforce connector properties



Walkthrough 7-4: Use property placeholders in connectors

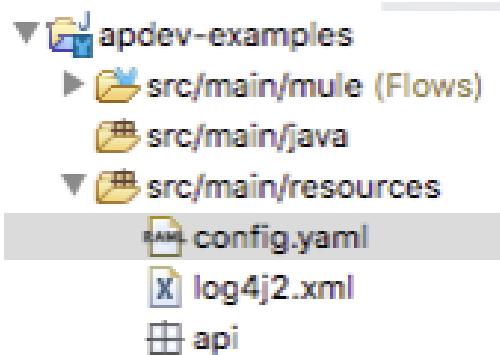
In this walkthrough, you introduce properties into your API implementation. You will:

- Create a YAML properties file for an application.
- Configure an application to use a properties file.
- Define and use HTTP and Salesforce connector properties.



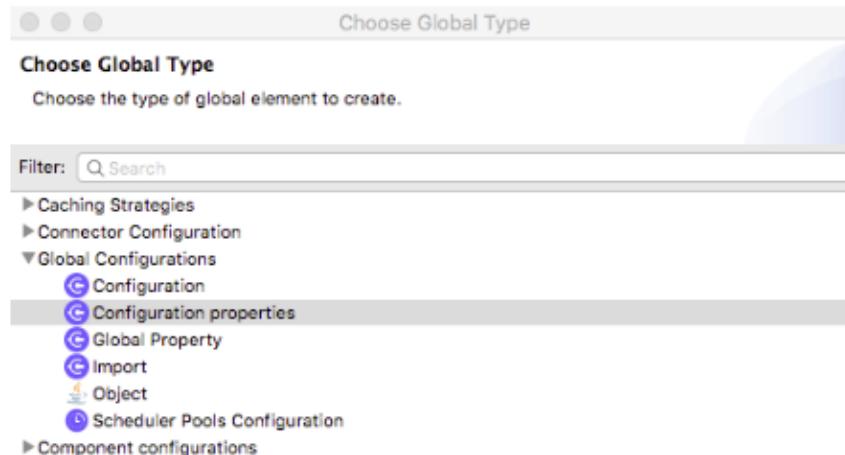
Create a properties file

1. Return to the apdev-examples project.
2. Right-click the src/main/resources folder in the Package Explorer and select New > File.
3. Set the file name to config.yaml and click Finish.

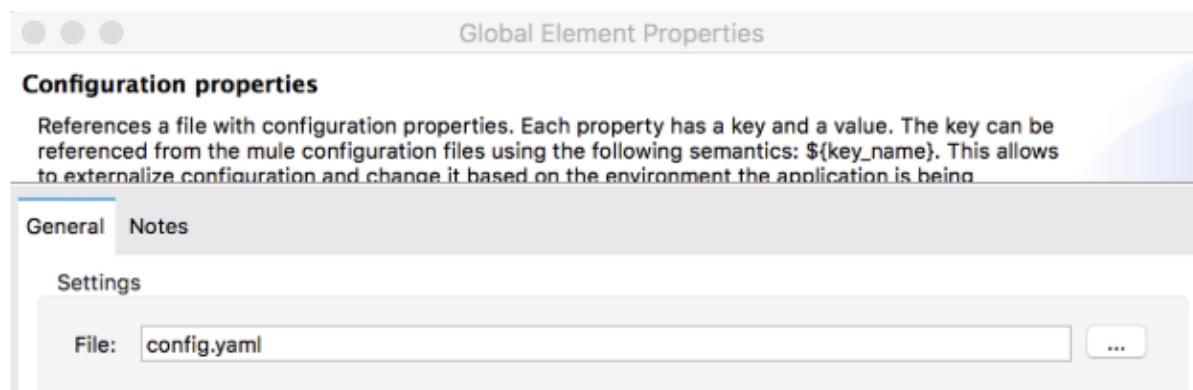


Create a Configuration properties global element

4. Return to global.xml.
5. In the Global Elements view, click Create.
6. In the Choose Global Type dialog box, select Global Configurations > Configuration properties and click OK.

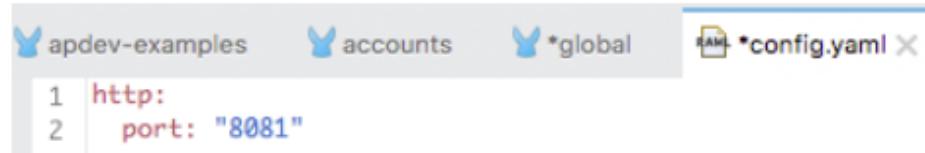


7. In the Global Element Properties dialog box, set the file to config.yaml and click OK.



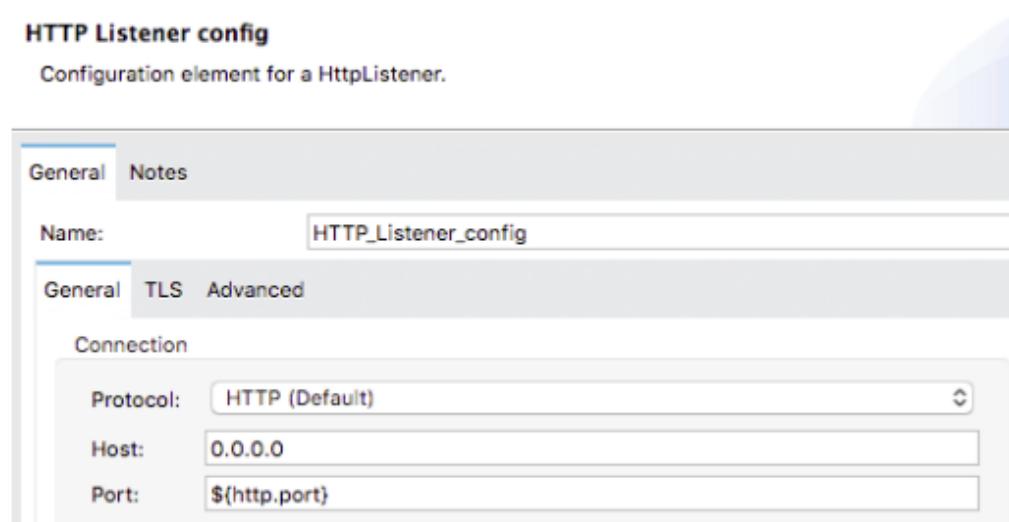
Parameterize the HTTP Listener port

8. Return to config.yaml.
9. Define a property called http.port and set it to 8081.



```
apdev-examples accounts *global config.yaml
1 http:
2   port: "8081"
```

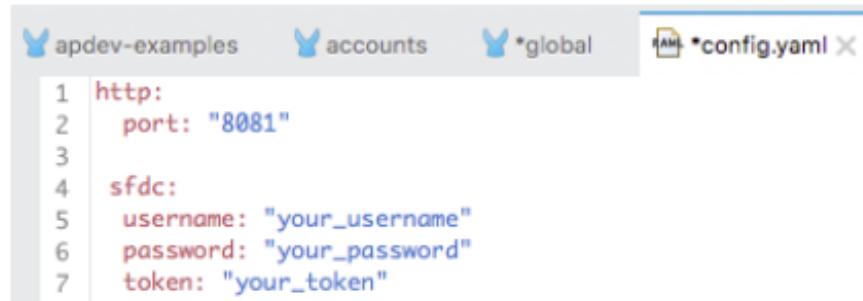
10. Save the file.
11. Return to global.xml.
12. Double-click the HTTP Listener config global element.
13. Change the port from 8081 to the application property, \${http.port}.



14. Click OK.

Parameterize the Salesforce credentials

15. Double-click the Salesforce Config global element.
16. Copy the token value and click OK.
17. Return to config.yaml .
18. Define a property called sfdc.username and set it to your Salesforce username.
19. Add properties for password and token and set them to your values.



```
apdev-examples accounts *global *config.yaml X
1 http:
2   port: "8081"
3
4 sfdc:
5   username: "your_username"
6   password: "your_password"
7   token: "your_token"
```

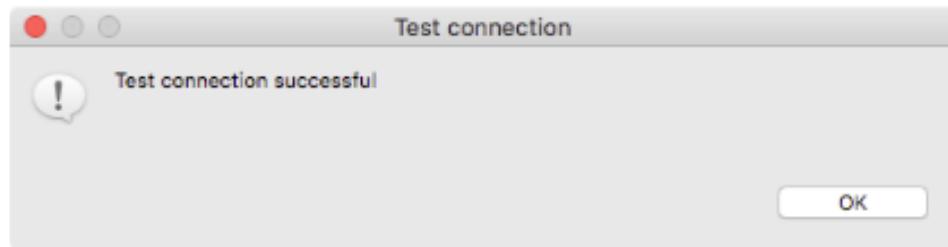
20. Save the file.
21. Return to global.xml.
22. Double-click the Salesforce Config global element.
23. Change the values to use the application properties.



Connection

Username:	<input type="text" value="\${sfdc.username}"/>
Password:	<input type="text" value="\${sfdc.password}"/> <input checked="" type="checkbox"/> Show password
Security token:	<input type="text" value="\${sfdc.token}"/>
Authorization URL:	<input type="text" value="https://login.salesforce.com/services/Soap/u/41.0"/>

24. Click Test Connection and make sure it succeeds.



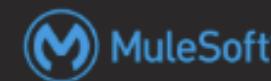
Note: If your connection fails, click OK and then go back and make sure you do not have any syntax errors in config.yaml and that you saved the file.

25. Click OK.
26. In the Global Element Properties dialog box, click OK.

Organizing Mule project files



Examining the folder structure for a Mule project



- The names of folders indicate what they should contain
- src/test folders should contain files only needed at development time
 - Like schema and example files for metadata types, sample data for transformations
 - They are not included in the application JAR when it is packaged

```
apdev-flights-ws
├── src/main/mule (Flows)
└── src/main/java
    └── com.mulesoft.training
        └── Flight.java
└── src/main/resources
    ├── application-types.xml
    ├── config.yaml
    └── json_flight_playground.dwl
    ├── log4j2.xml
    ├── api
    ├── api.datatypes
    ├── api.examples
    ├── examples
    └── schemas
    └── src/test/java
    └── src/test/munit
```

```
src/test/resources
├── flight-example.json
├── flights-example.json
└── flights-example.xml
├── log4j2-test.xml
└── united-flights-example.json
    └── sample_data
        ├── American Flights API [v1.0.2]
        ├── APIKit [v1.1.1]
        ├── HTTP [v1.1.0]
        ├── JRE System Library [Java SE 8 [1.8.0_144]]
        ├── Mule Server 4.1.1 EE
        ├── Validation [v1.2.1]
        └── Web Service Consumer [v1.1.1]
└── src
    └── target
        └── mule-artifact.json
    └── pom.xml [Mule Server 4.1.1 EE]
```



```
apdev-flights-ws
├── api
└── com
└── META-INF
└── repository
    ├── config.yaml
    ├── global.xml
    ├── implementation.xml
    ├── interface.xml
    └── log4j2.xml
```

- **Maven** is a tool for building and managing any Java-based project that provides
 - A standard way to build projects
 - A clear definition of what a project consists of
 - An easy way to publish project information
 - A way to share JARs across several projects
- Maven manages a project's build, reporting, and documentation from a central piece of information – the **project object model (POM)**
- A Maven build produces one or more **artifacts**, like a compiled JAR
 - Each artifact has a group ID (usually a reversed domain name, like com.example.foo), an artifact ID (just a name), and a version string



Apache Maven Project

<http://maven.apache.org/>

The POM (Project Object Model)



- Is an XML file that contains info about the project and configuration details used by Maven to build the project including
 - Project info like its version, description, developers, and more
 - Project dependencies
 - The plugins or goals that can be executed

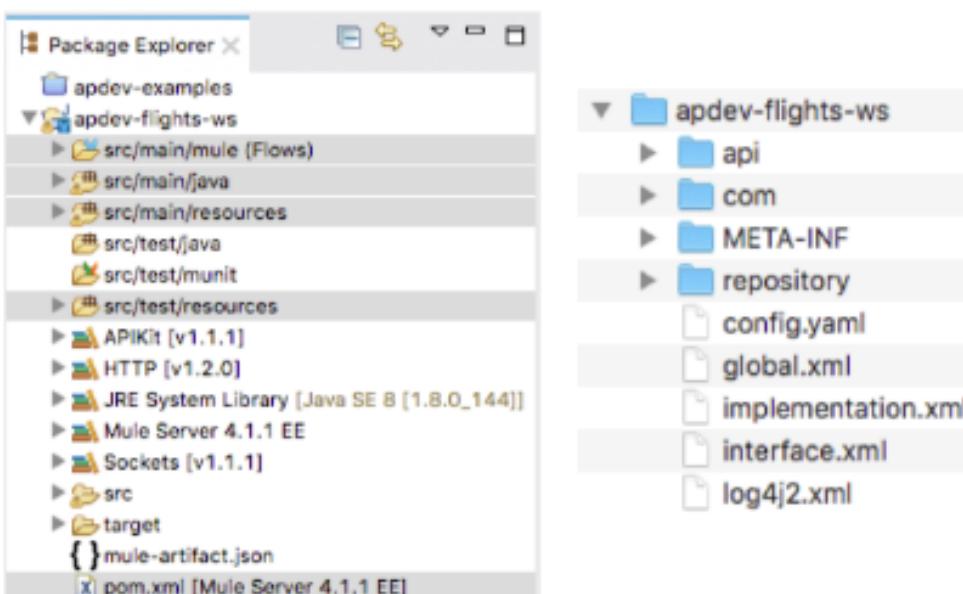
A screenshot of a code editor window titled "pom.xml". The XML code is color-coded for syntax highlighting. The code defines a Maven project with group ID "com.mulesoft", artifact ID "apdev-flights-ws", and version "1.0.0-SNAPSHOT". It specifies "mule-application" as the packaging and "apdev-flights-ws" as the name. The XML includes sections for properties, build, and dependencies, listing several dependencies including "org.mule.modules/mule-ppikit-module" and "org.mule.connectors/mule-http-connector".

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.mulesoft</groupId>
    <artifactId>apdev-flights-ws</artifactId>
    <version>1.0.0-SNAPSHOT</version>
    <packaging>mule-application</packaging>
    <name>apdev-flights-ws</name>
    <properties></properties>
    <build></build>
    <dependencies>
        <dependency>
            <groupId>org.mule.modules</groupId>
            <artifactId>mule-ppikit-module</artifactId>
            <version>1.1.1</version>
            <classifier>mule-plugin</classifier>
        </dependency>
        <dependency>
            <groupId>74922056-9245-48e0-99df-a8141d1d3e9f</groupId>
            <artifactId>american4-flights-ppi</artifactId>
            <version>1.0.2</version>
            <classifier>mule-plugin</classifier>
        </dependency>
        <dependency>
            <groupId>org.mule.connectors</groupId>
            <artifactId>mule-http-connector</artifactId>
```

Walkthrough 7-5: Create a well-organized Mule project



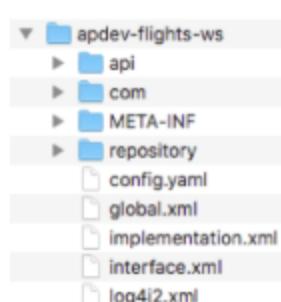
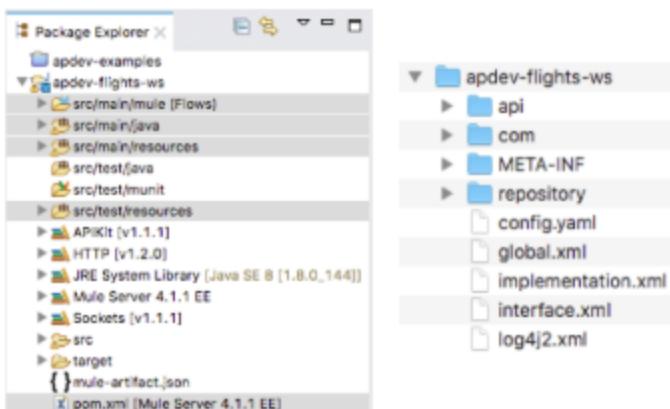
- Create a project with a new RAML file that is added to Anypoint Platform
- Review the project's configuration and properties files
- Create an application properties file and a global configuration file
- Add Java files and test resource files to the project
- Create and examine the contents of a deployable archive for the project



Walkthrough 7-5: Create a well-organized Mule project

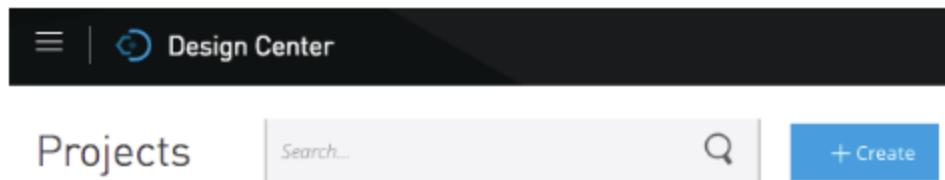
In this walkthrough, you create a new project for the Mule United Airlines (MUA) flights application that you will build during the course and then review and organize its files and folders. You will:

- Create a project based on a new API in Anypoint Platform Design Center.
- Review the project's configuration and properties files.
- Create an application properties file and a global configuration file for the project.
- Add Java files and test resource files to the project.
- Create and examine the contents of a deployable archive for the project.

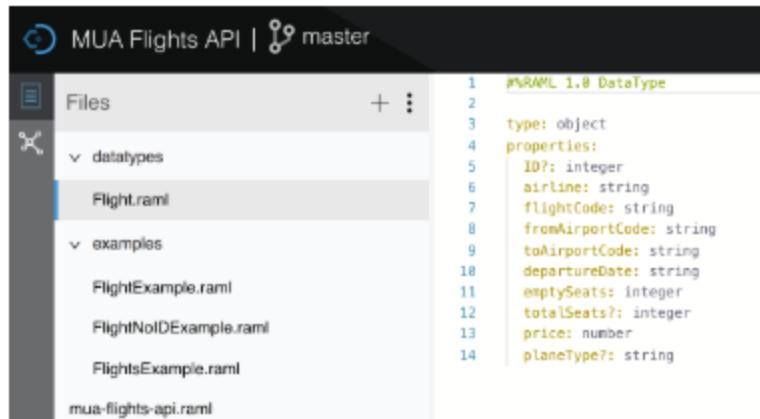


Create a new API in Design Center

1. Return to Anypoint Platform in a web browser.
2. In Design Center, create a new API specification project called MUA Flights API.

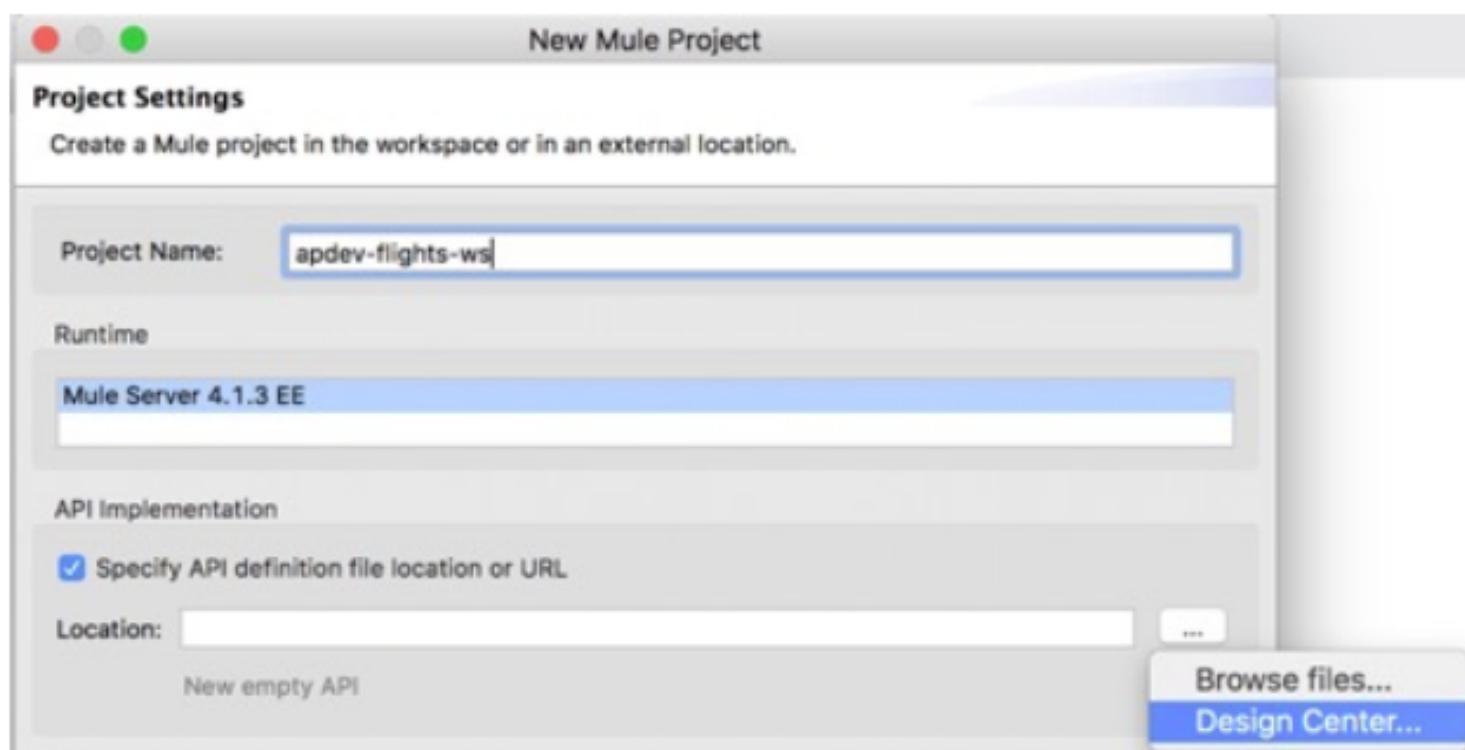


3. In API designer, click the options menu in the file browser and select Import.
4. In the Import dialog box, browse to your student files and select the MUA Flights API.zip located in the resources folder.
5. Click Import.
6. In the Replace dialog box, click Replace file.
7. Explore the API; be sure to look at what resources are defined, the name of the query parameter, and the structure of the Flight data type.

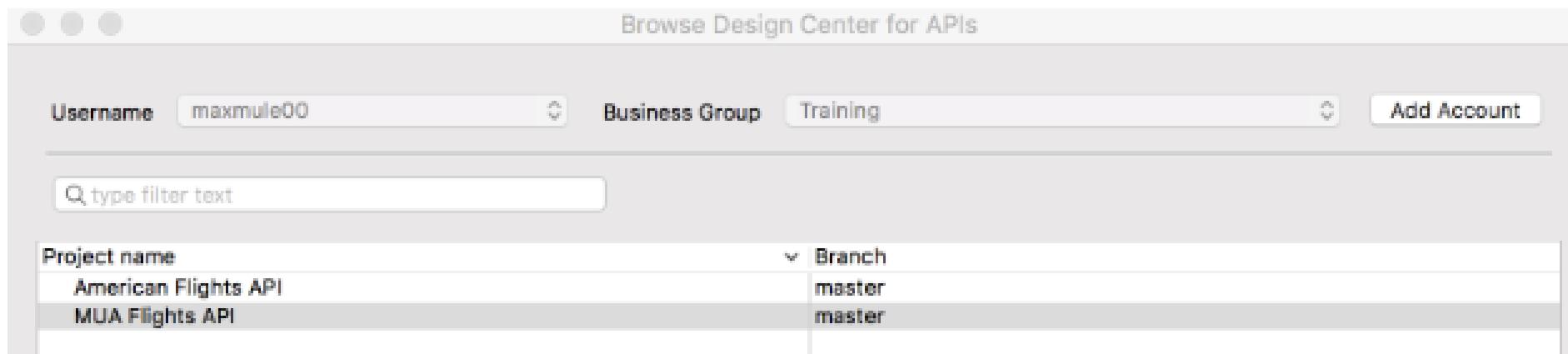


Create a new project to implement this API in Anypoint Studio

8. Return to Anypoint Studio.
9. Right-click in the Package Explorer and select New > Mule Project.
10. In the New Mule Project dialog box, set the project name to apdev-flights-ws.
11. Check Specify API definition file location or URL.
12. Click the browse button next to location and select Design Center.



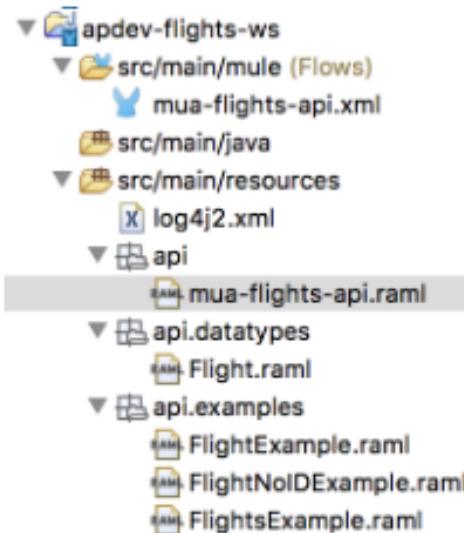
13. In the Browse API Design Center for APIs dialog box, select MUA Flights API and click OK.



14. In the New Mule Project dialog box, click Finish.

Locate the new RAML files in Anypoint Studio

15. Return to the apdev-flights-ws project in Anypoint Studio.
16. In the Package Explorer, expand the folders in src/main/resources folder; you should see the MUA Flights API files.



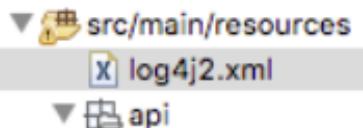
17. Open `mua-flights-api.raml` and review the file.
18. Leave the file open.

Review project configuration files

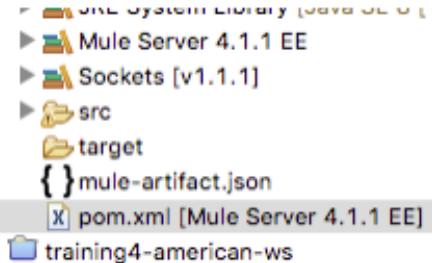
19. Look at the mua-flights-api.xml file that was created; it should have a get:flights flow and a post:flights flow.
20. Rename mua-flights-api.xml to interface.xml.
21. Create a new Mule configuration file called implementation.xml.



22. Locate log4j2.xml in src/main/resources and open it.



23. Review its contents and then close the file.
24. Locate pom.xml in the project and open it.



25. Review its contents.
26. Return to the apdev-examples project and open its pom.xml file.

27. Compare the two pom.xml files.

The screenshot shows a code editor with two tabs open, both titled "pom.xml". The left tab contains XML code for dependencies and repositories, including a dependency on "org.mule.modules" and a repository for "anypoint-exchange". The right tab contains similar XML code, with a dependency on "org.mule.connectors" and a repository for "maven.anypoint.mulesoft.com". The code is color-coded for syntax highlighting.

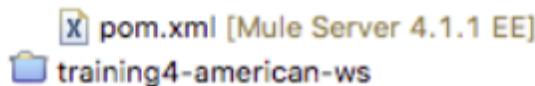
```
<dependency>
<groupId>org.mule.modules</groupId>
<artifactId>mule-apikit-module</artifactId>
<version>1.1.1</version>
<classifier>mule-plugin</classifier>
</dependency>
</dependencies>
<repositories>
<repository>
<id>anypoint-exchange</id>
<name>Anypoint Exchange</name>
<url>https://maven.anypoint.mulesoft.com</url>
</repository>

```

```
<dependency>
<groupId>org.mule.connectors</groupId>
<artifactId>mule-xml-connector</artifactId>
<version>1.1.1</version>
<classifier>mule-plugin</classifier>
</dependency>
<dependency>
<groupId>com.mulesoft.connectors</groupId>
<artifactId>mule-salesforce-connector</artifactId>
<version>9.1.0</version>
<classifier>mule-plugin</classifier>
</dependency>
</dependencies>
```

28. Close both pom.xml files.

29. In the Package Explorer, right-click apdev-examples and select Close Project.



Create a properties file for application properties

30. In the apdev-flights-ws project, right-click src/main/resources and select New > File.

31. In the New File dialog box, set the file name to config.yaml and click Finish.

32. In config.yaml, define a property called http.port equal to "8081".

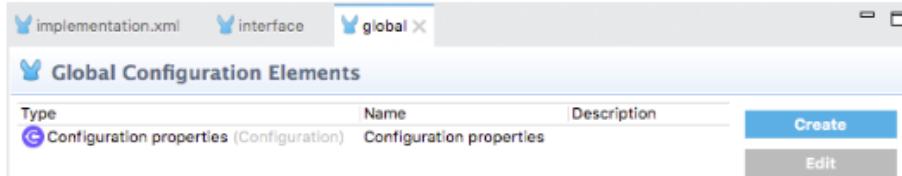
The screenshot shows a code editor with a tab titled "config.yaml". The file contains a single YAML key-value pair: "http: port: \"8081\"".

```
1 http:
2   port: "8081"
```

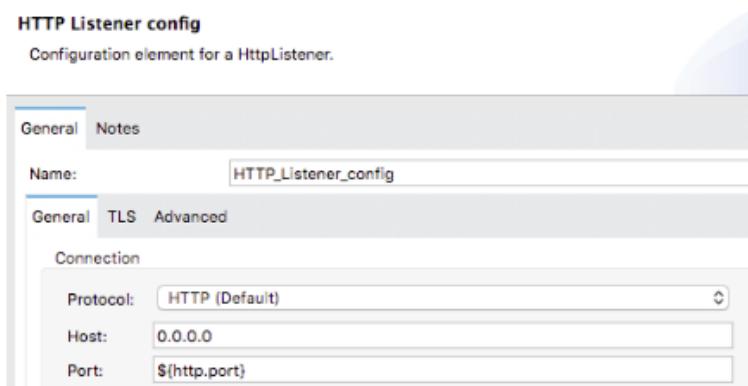
33. Save and close the file.

Create a global configuration file

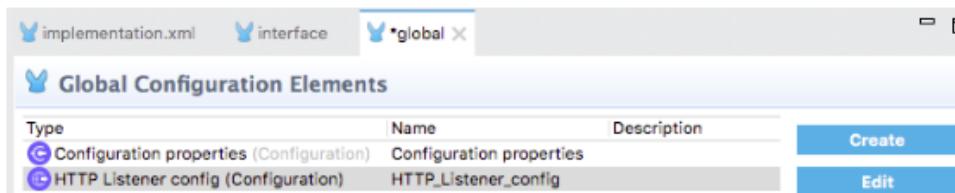
34. In src/main/mule, create a new Mule configuration file called global.
35. In global.xml, switch to the Global Elements view.
36. Create a new Configuration properties element with the file set to config.yaml.



37. Create a new HTTP Listener config element with the host set to 0.0.0.0 and the port set to the http.port property.



38. Confirm you now have two global elements defined in global.xml.



39. Save and close the file.

40. Go to the Global Elements view in interface.xml.

41. Delete the mua-flights-api-httpListener HTTP Listener Configuration.

Type	Name	Description
Router (Configuration)	mua-flights-api-config	

Create

Edit

42. Return to the Message Flow view.

43. Select the HTTP Listener in mua-flights-api-main, and in the Listener properties view set the Connector configuration to `HTTP_Listener_config`.

Display Name: Listener

Basic Settings

Connector configuration: HTTP_Listener_config

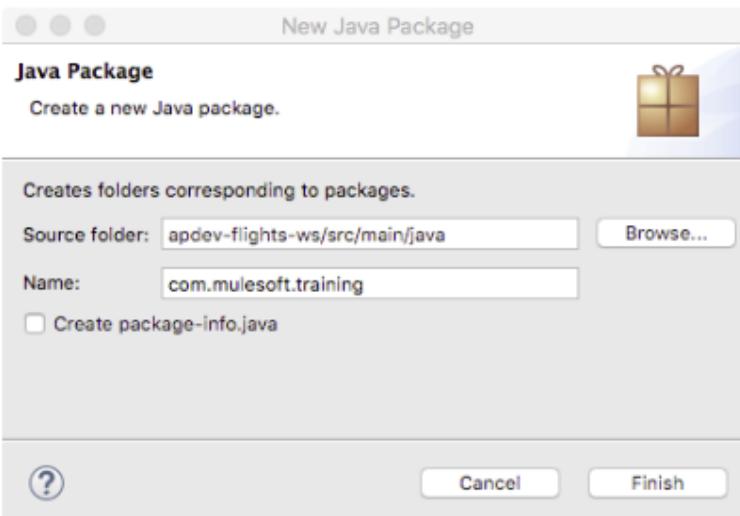
General

Path: /api/*

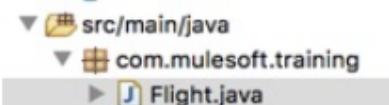
44. Select the HTTP Listener in mua-flights-api-console, and in the Listener properties view set the connector configuration to `HTTP_Listener_config`.

Add Java files to src/main/java

45. In the Package Explorer, right-click the src/main/java folder and select New > Package.
46. In the New Java Package dialog box, set the name to com.mulesoft.training and click Finish.

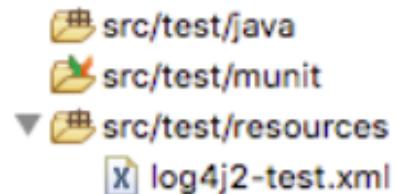


47. In your computer's file browser, locate the Flight.java file in the resources folder.
48. Drag the Flight.java file into the new com.mulesoft.training package in the src/main/java folder in the Anypoint Studio apdev-flights-ws project.
49. In the File Operation dialog box, select Copy files and click OK.
50. Open the Flight.java file.
51. Review the code.



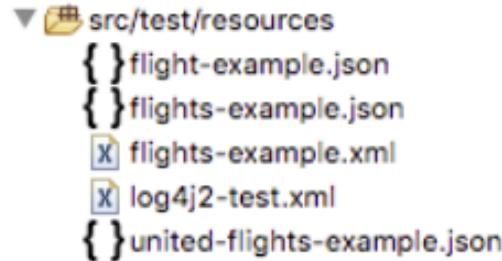
Review src/test folders

52. In the project, locate the three src/test folders.
53. Expand the src/test/resources folder.



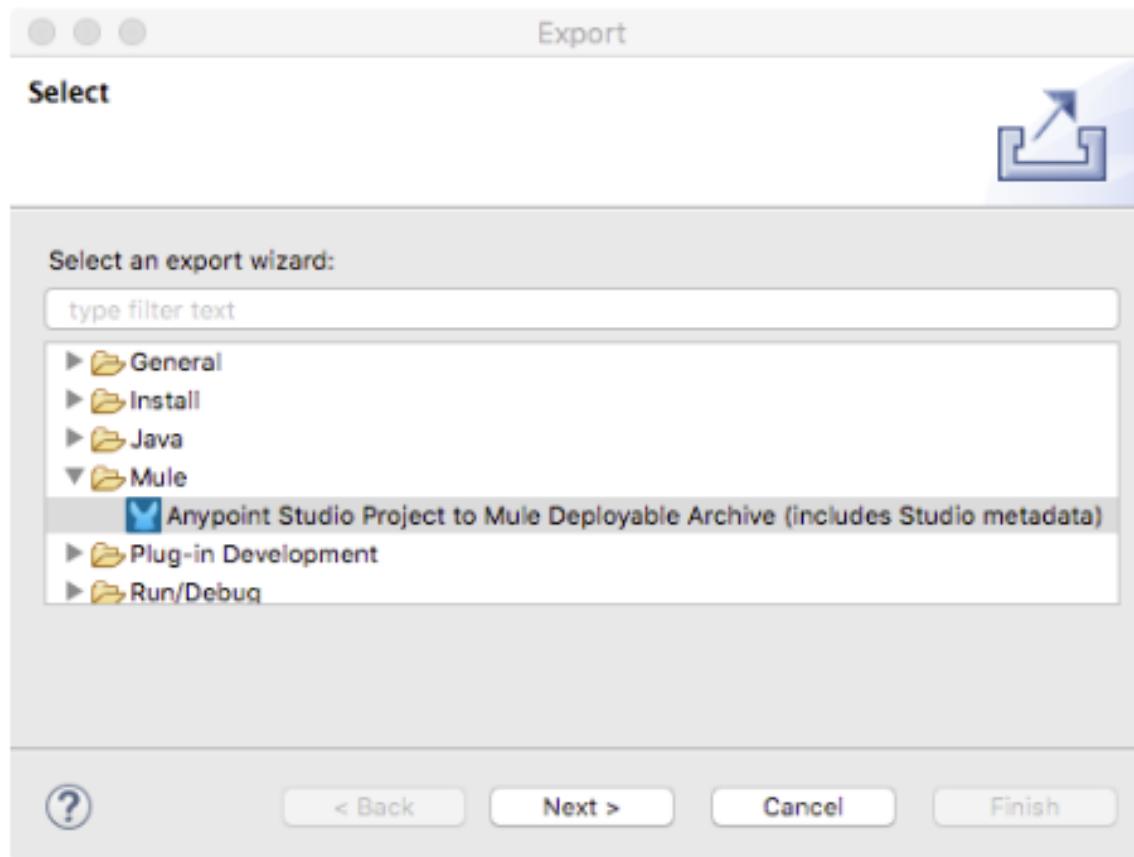
Add test resources

54. In your computer's file browser, expand the examples folder in the student files.
55. Select the three flights and one united-flights files (JSON and XML) and drag them into the src/test/resources folder in the Anypoint Studio apdev-flights-ws project.

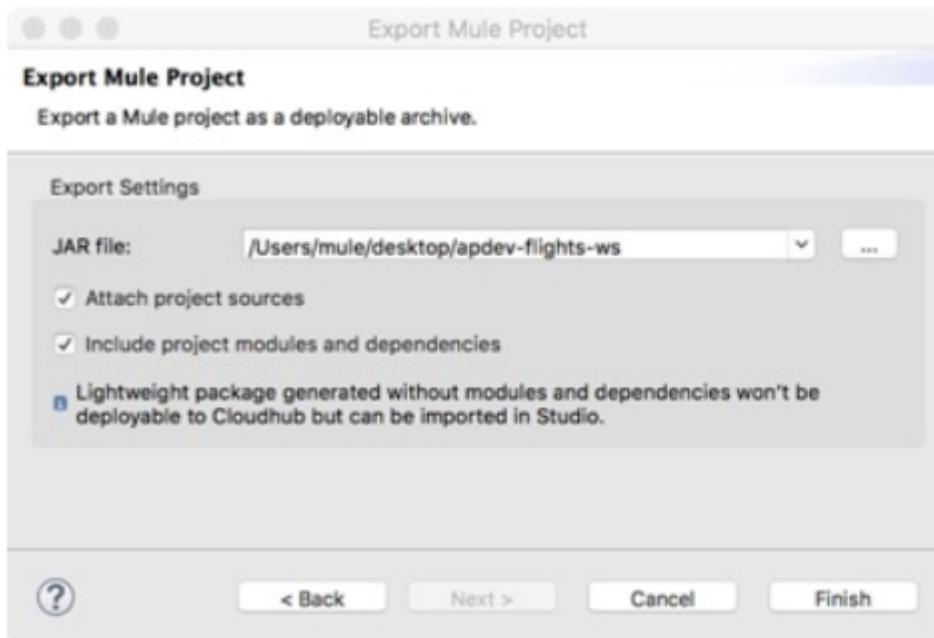


Examine the contents of a deployable archive

56. In Anypoint Studio, right-click the project and select Export.
57. In the Export dialog box, select Mule > Anypoint Studio Project to Mule Deployable Archive and click Next.

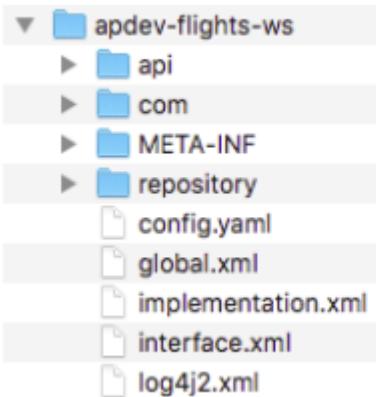


58. In the Export Mule Project dialog box, set the JAR file to a location that you can find and click Finish.



59. In your computer's file browser, locate the JAR file and unzip it.

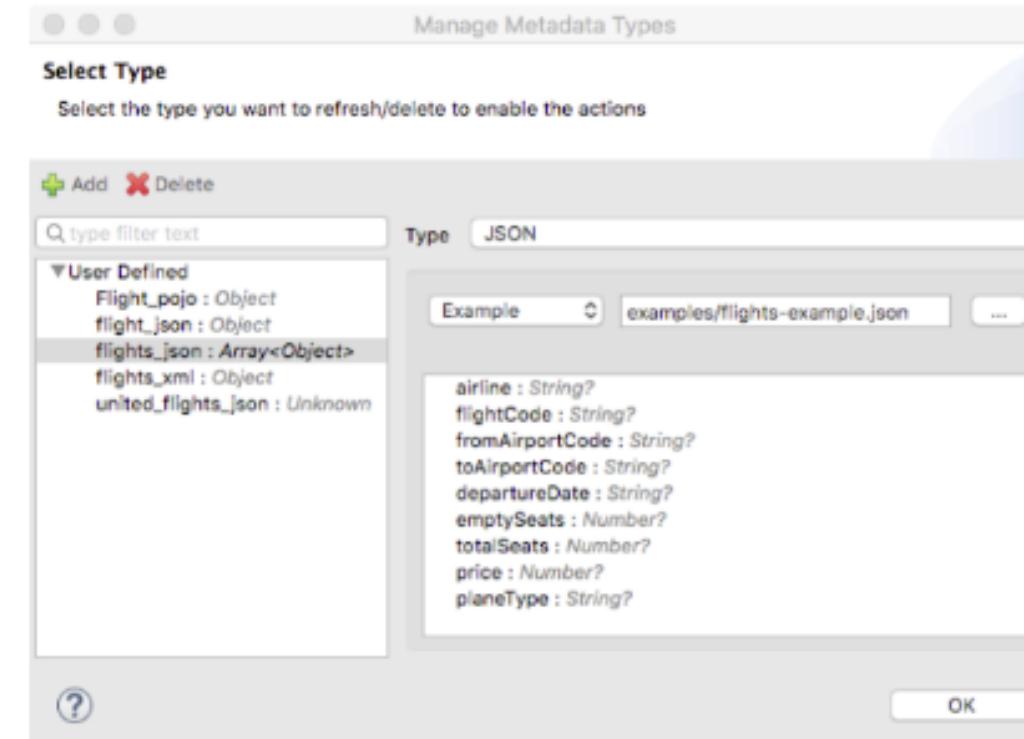
60. Open the resulting apdev-flights-ws folder and examine the contents.



Managing metadata for a project



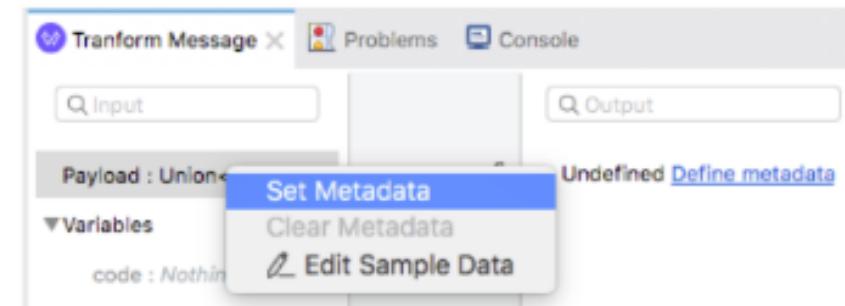
- It is often beneficial to define metadata for an application
 - For the output structures required for transformations
 - You did this in Module 4 when transforming database output to JSON defined in the API
 - For the output of operations that can connect to data sources of different structures
 - Like the HTTP Request connector
 - For the output of connectors that are not DataSense enabled
 - And do not automatically provide metadata about the expected input and output



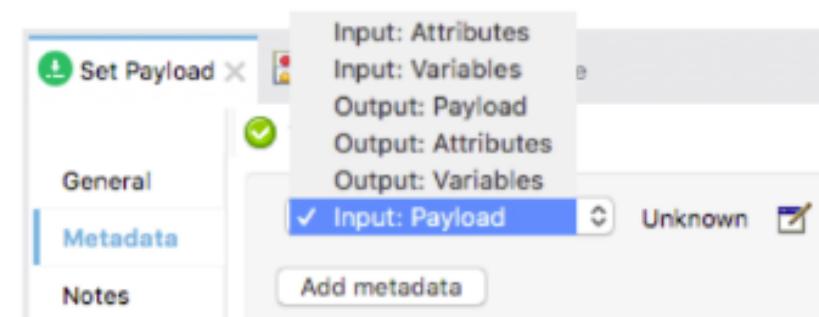
Ways to access the Metadata Editor



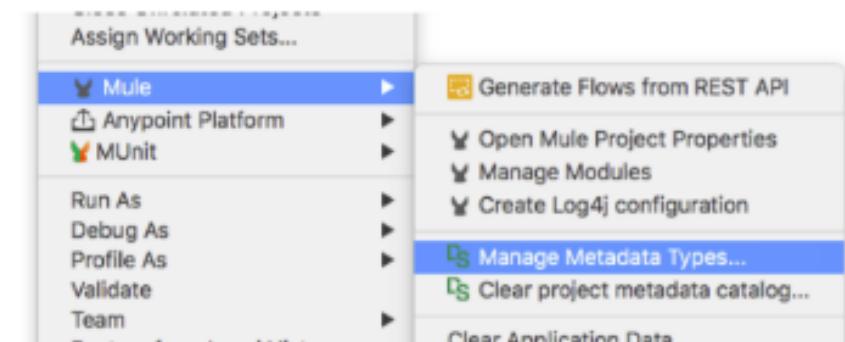
- From the Transform Message component



- From the Metadata tab in the properties view for most event processors

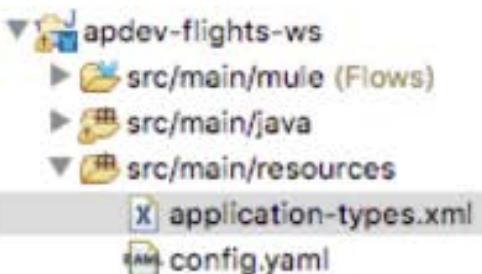


- From a project's menu in the Package Explorer



Where is metadata stored?

- In application-types.xml in src/main/resources



The image shows the content of the application-types.xml file. The code is as follows:

```
<?xml version='1.0' encoding='UTF-8'?>
<types:mule xmlns:types="http://www.mulesoft.org/schema/mule/types">
  <types:catalog>
    <types:type name="united_flights_json" format="json">
      <types:type name="Flight_pojo" format="java">
        <types:type name="flights_json" format="json">
          <types:type name="flights_xml" format="xml">
            <types:type name="flight_json" format="json">
              <types:example format="json" location="examples/flight-example.json"/>
            </types:type>
          </types:type>
        </types:type>
      </types:type>
    </types:type>
  </types:catalog>
  <types:enrichment select="#d1d1a734-0050-43cf-baba-a52bfabba85f">
    <types:enrichment select="#f17bea7e-21f3-42a2-a94e-926a936a6f01">
      <types:enrichment select="#659fe94d-369d-4f7b-b9e0-733914e41624">
        <types:enrichment select="#09e05593-c069-49e2-8562-4aac7e67cf10">
          <types:enrichment select="#6309e615-e6e9-4808-b78b-ec75b0a58778">
            <types:enrichment select="#f963e3e4-5b30-41bf-8c32-ef79bdce4a32">
              <types:enrichment select="#be12bc7d-22f8-49c5-8566-68741d80c449">
                <types:processor-declaration>
                  <types:input-event>
                    <types:message>
                      <types:payload type="flight_json"/>
                    </types:message>
                  </types:input-event>
                </types:processor-declarations>
              </types:enrichment>
            <types:enrichment select="#edcf1fd1-6237-40d1-ad8d-c196a9b00f21">
        </types:enrichment>
      </types:enrichment>
    </types:enrichment>
  </types:mule>
```

Walkthrough 7-6: Manage metadata for a project



- Review existing metadata for the training-american-ws project
- Define new metadata to be used in transformations in the new apdev-flights-ws project
- Manage metadata

Manage Metadata Types

Select Type

Select the type you want to refresh/delete to enable the actions

Add Delete

Q type filter text

Type Object

User Defined

- Flight_pojo : String
- flight_json : Object
- flights_json : Array<Object>
- flights_xml : Object

Class

Data structure

com.mulesoft.training.Flight

airlineName : String?

availableSeats : Number?

departureDate : String?

destination : String?

flightCode : String?

origination : String?

OK

?

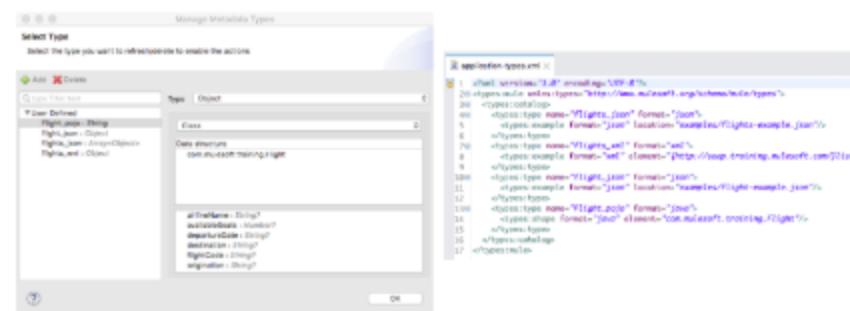
application-types.xml

```
<?xml version='1.0' encoding='UTF-8'?>
<types:mule xmlns:types="http://www.mulesoft.org/schema/mule/types">
  <types:catalog>
    <types:type name="flights_json" format="json">
      <types:example format="json" location="examples/flights-example.json"/>
    </types:type>
    <types:type name="flights_xml" format="xml">
      <types:example format="xml" element="(http://soap.training.mulesoft.com)/list"/>
    </types:type>
    <types:type name="flight_json" format="json">
      <types:example format="json" location="examples/flight-example.json"/>
    </types:type>
    <types:type name="Flight_pojo" format="java">
      <types:shape format="java" element="com.mulesoft.training.Flight"/>
    </types:type>
  </types:catalog>
</types:mule>
```

Walkthrough 7-6: Manage metadata for a project

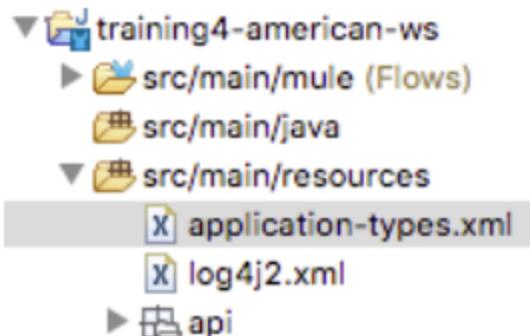
In this walkthrough, you define metadata for the new apdev-flights-ws project. You will:

- Review existing metadata for the training-american-ws project.
 - Define new metadata to be used in transformations in the new apdev-flights-ws project.
 - Manage metadata.



Locate existing metadata in the training-american-ws project

1. Open the training-american-ws project.
2. Locate application-types.xml in src/main/resources.



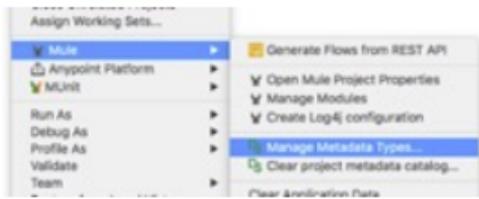
3. Open the file and review the code.

The screenshot shows a code editor window with the title bar 'application-types.xml'. The XML code is displayed in the editor:

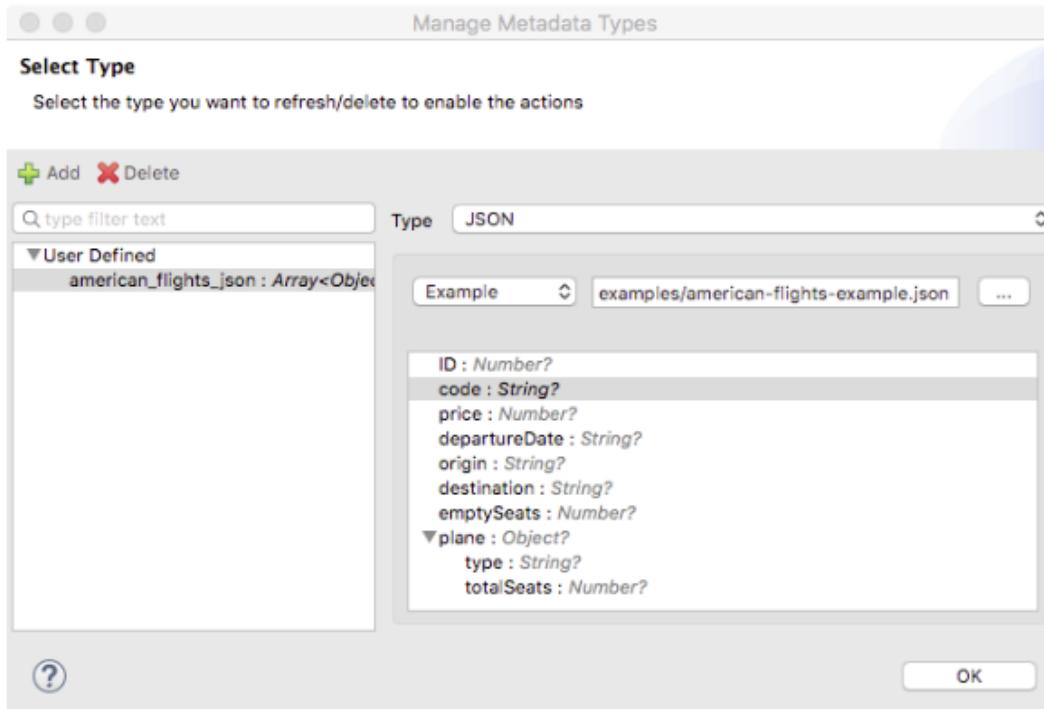
```
1 <?xml version='1.0' encoding='UTF-8'?>
2 <types:mule xmlns:types="http://www.mulesoft.org/schema/mule/types">
3   <types:catalog>
4     <types:type name="american_flights_json" format="json">
5       <types:example format="json" location="examples/american-Flights-example.json"/>
6     </types:type>
7   </types:catalog>
8   <types:enrichment select="#89727fef-578a-4d3e-b1ce-533af28359e7">
9     <types:processor-declaration>
10    <types:output-event>
11      <types:message>
12        <types:payload type="american_flights_json"/>
13      </types:message>
14    </types:output-event>
15  </types:processor-declaration>
16 </types:enrichment>
17 </types:mule>
```

The line containing the XML declaration ('<?xml version='1.0' encoding='UTF-8'?>') has a yellow warning icon next to it. Line 5, which defines the 'american_flights_json' type, is highlighted with a blue selection bar. The entire code block is surrounded by a light gray background, matching the highlighting of the file in the file tree above.

4. Right-click the training-american-ws project in the Project Explorer and review the options under Mule; you should see two for metadata.



5. Select Mule > Manage Metadata types.
6. In the Manage Metadata Types dialog box, select the american-flights_json type.



7. Click OK.
8. Close the project.

Review the API specification and resources for the new apdev-flights-ws project

9. Return to the apdev-flights-ws project.
10. Return to mua-flights-api.raml and review the specified response and request types.

```
responses:  
  200:  
    body:  
      application/json:  
        type: Flight[]  
        example: !include examples/FlightsExample.raml  
  
  post:  
    body:  
      application/json:  
        type: Flight  
        example: !include examples/FlightNoIDExample.raml
```

11. Locate the files you added to the src/test/resources folder.

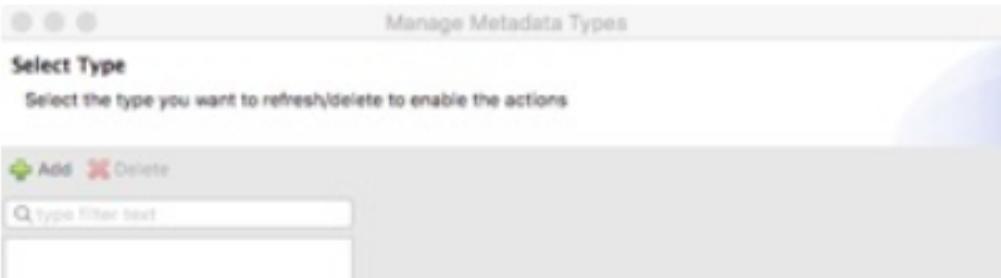
```
▼ src/test/resources  
  { } flight-example.json  
  { } flights-example.json  
  X flights-example.xml  
  X log4j2-test.xml  
  { } united-flights-example.json
```

12. Locate the files you added to the src/main/java folder.

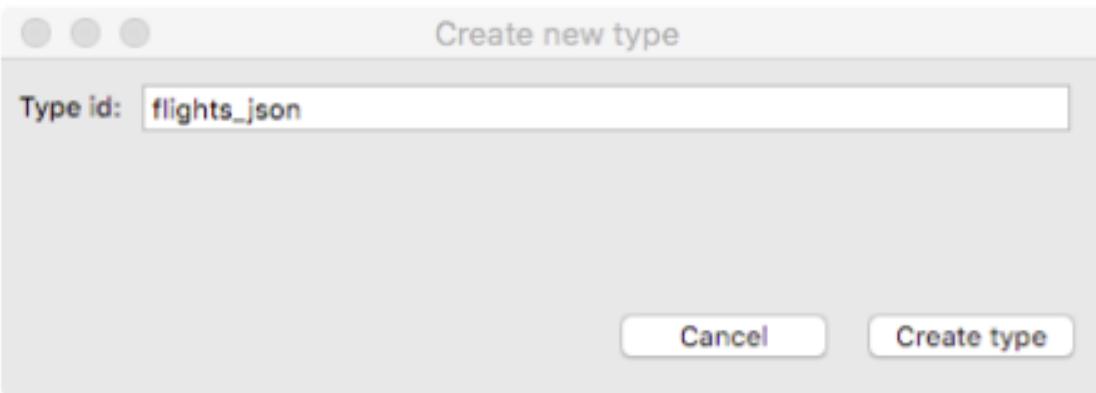
```
▼ src/main/java  
  ▼ com.mulesoft.training  
    ► Flight.java
```

Create flights_json metadata type

13. Right-click the project and select Mule > Manage Metadata Types.
14. In the Select metadata type dialog box, click the Add button.

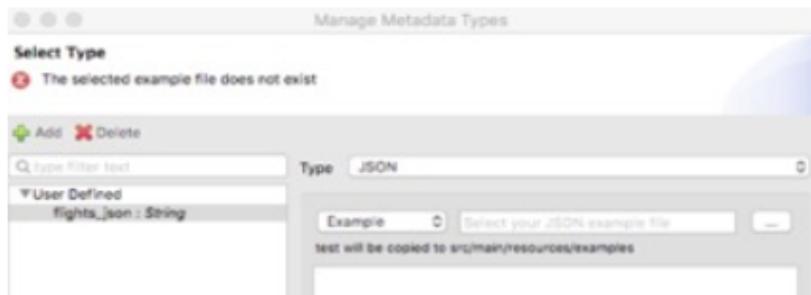


15. In the Create new type dialog box, set the type id to flights_json.

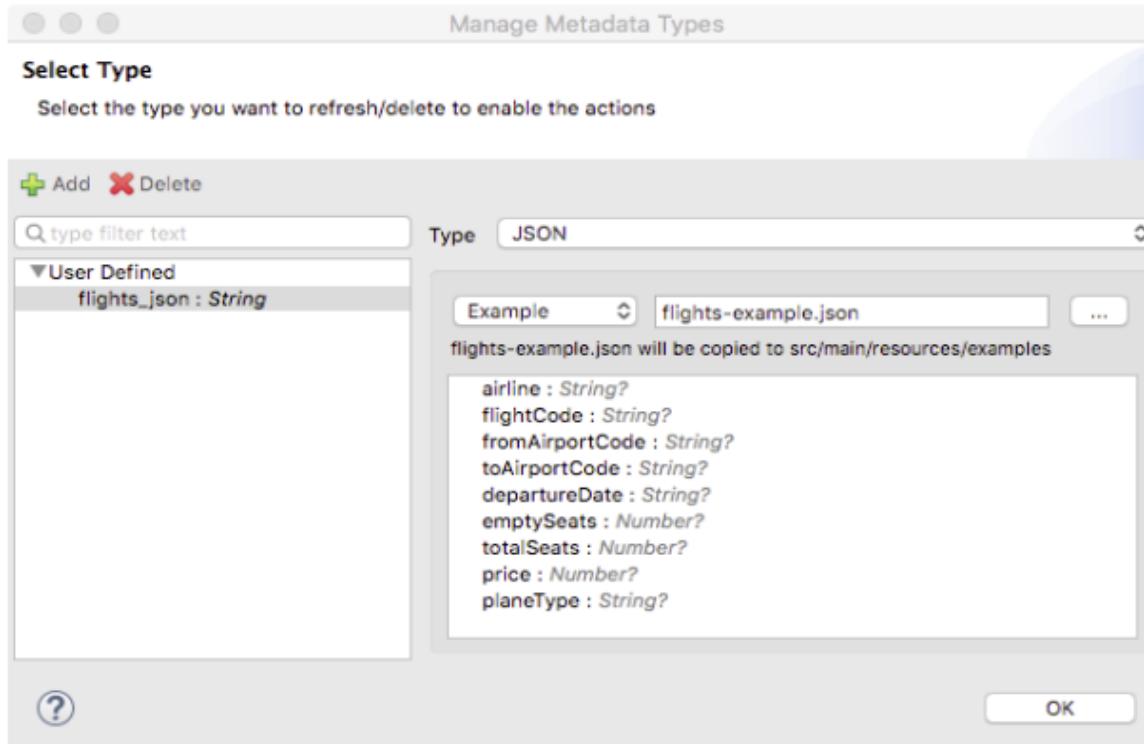


16. Click Create type.
17. In the Select metadata type dialog box, set the type to JSON.

18. In the drop-down menu beneath the type, select Example.



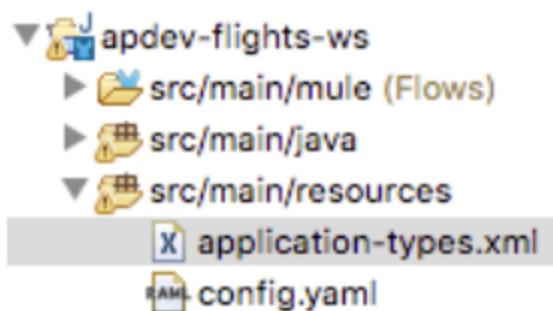
19. Click the browse button and select the flights-example.json file in src/test/resources.



20. In the Select metadata type dialog box, click OK.

Locate the new metadata definition file

21. Locate application-types.xml in src/main/resources.



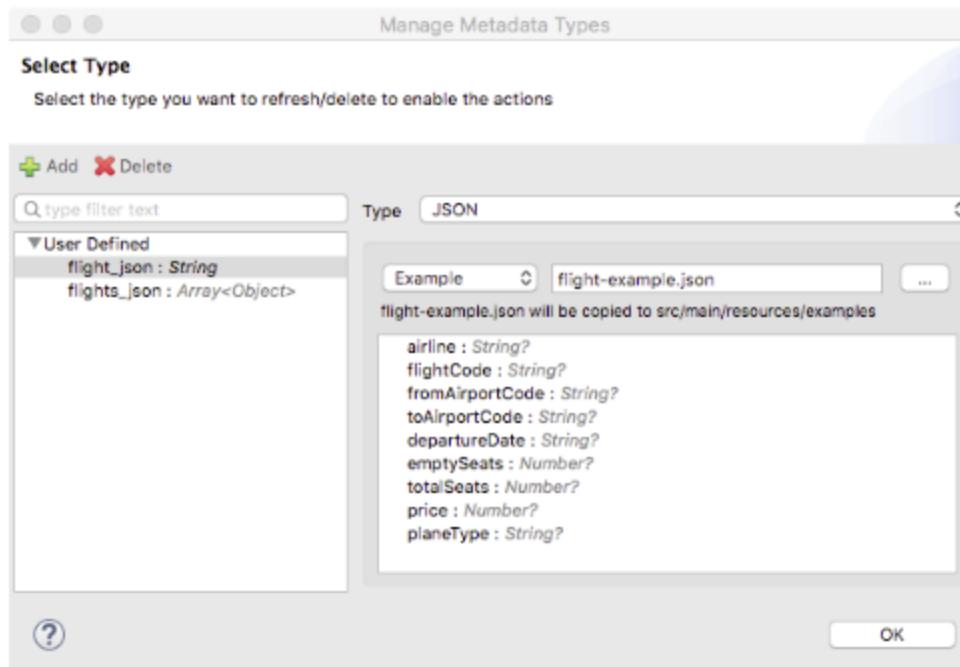
22. Open the file and review the code.

The screenshot shows the contents of the 'application-types.xml' file in a code editor. The XML code defines a single type named 'flights_json' which is formatted as JSON. It includes an example located at 'examples/flights-example.json'.

```
<?xml version='1.0' encoding='UTF-8'?>
<types:mule xmlns:types="http://www.mulesoft.org/schema/mule/types">
    <types:catalog>
        <types:type name="flights_json" format="json">
            <types:example format="json" location="examples/flights-example.json"/>
        </types:type>
    </types:catalog>
</types:mule>
```

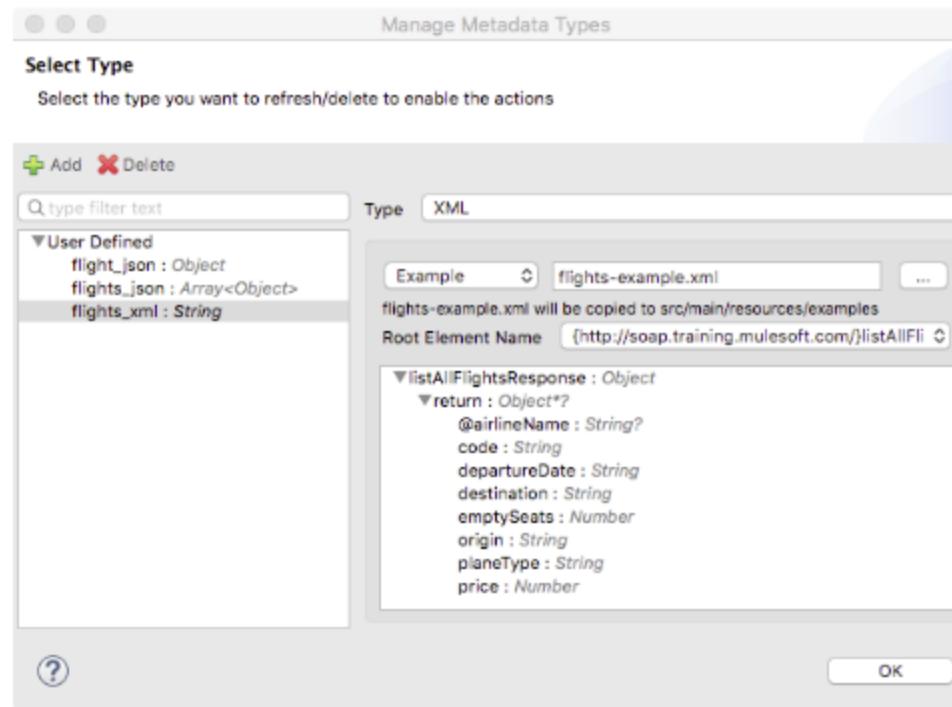
Create flight_json metadata type

23. Right-click the project and select Mule > Manage Metadata Types.
24. In the Select metadata type dialog box, click Add.
25. In the Create new type dialog box, set the type id to flight_json and click Create type.
26. In the Select metadata dialog box, set the type to JSON.
27. Select Example and browse to and select the flight-example.json file in src/test/resources.



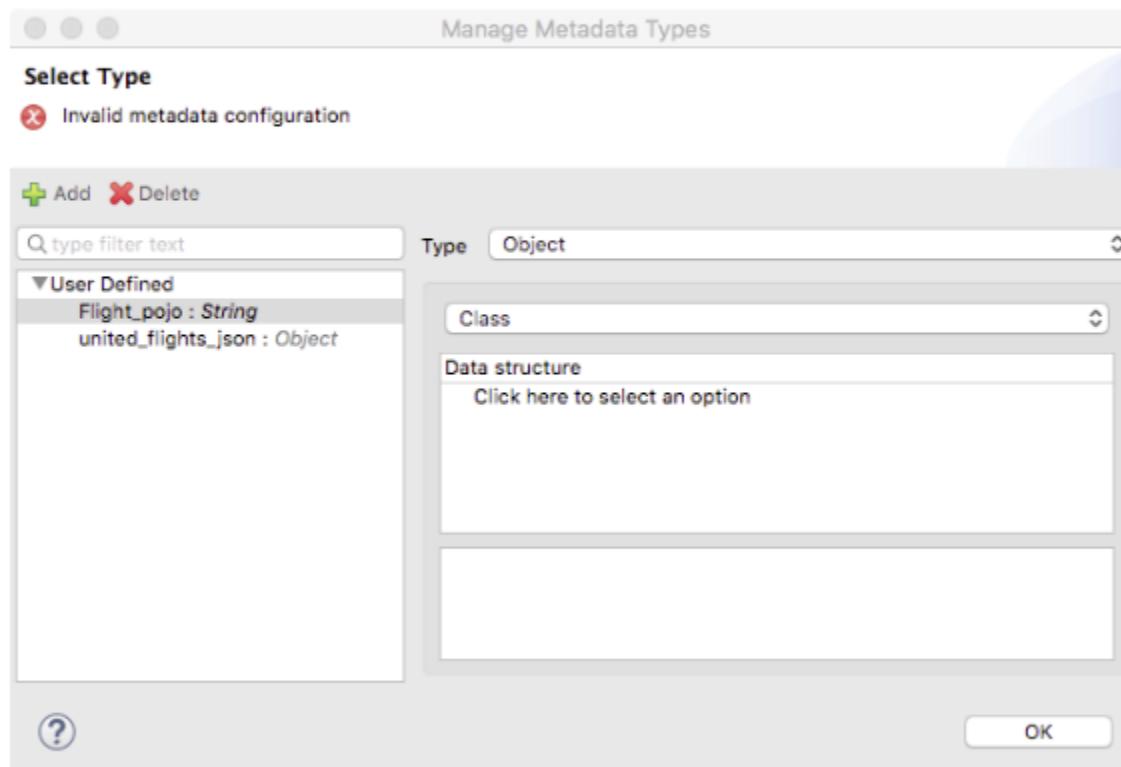
Create flights_xml metadata type

28. In the Select metadata type dialog box, click Add.
29. In the Create new type dialog box, set the type id to flights_xml and click Create type.
30. In the Select metadata dialog box, set the type to XML.
31. Select Example and browse to and select the flights-example.xml file in src/test/resources.



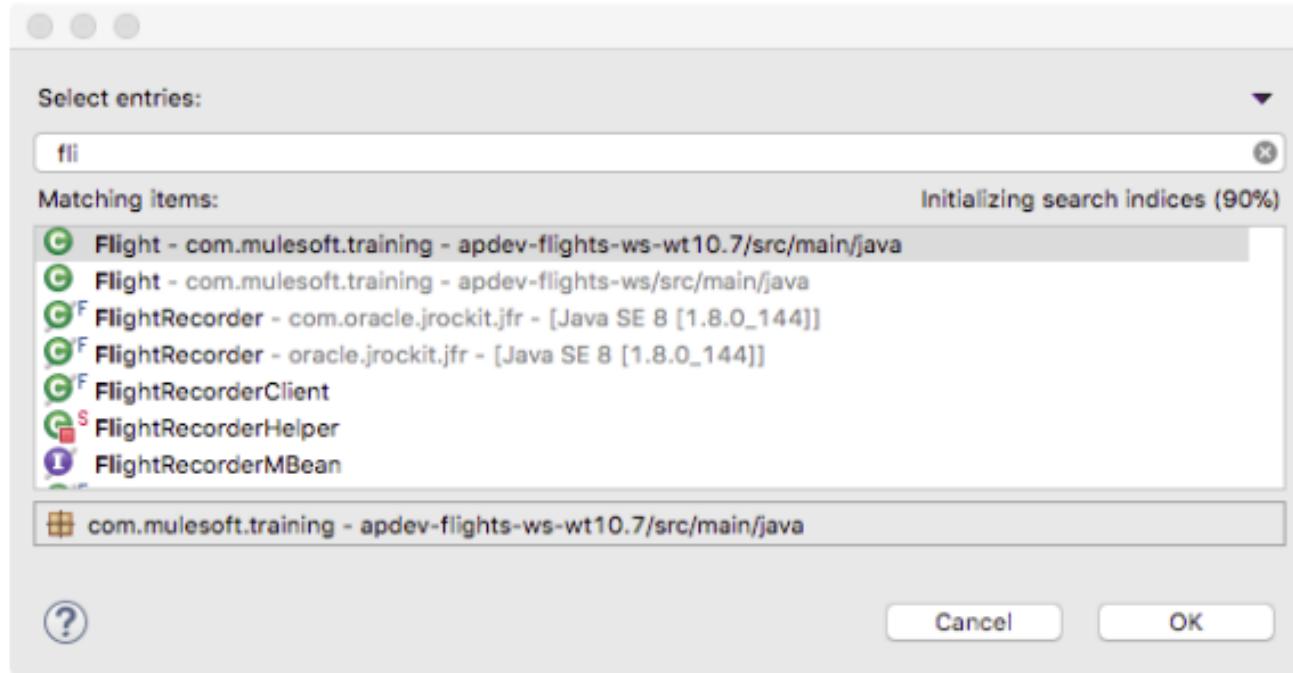
Create Flight metadata type

32. In the Select metadata type dialog box, click the Add button.
33. In the Create new type dialog box, set the type id to Flight_pojo and click Create type
34. In the Select metadata type dialog box, set the type to Object.

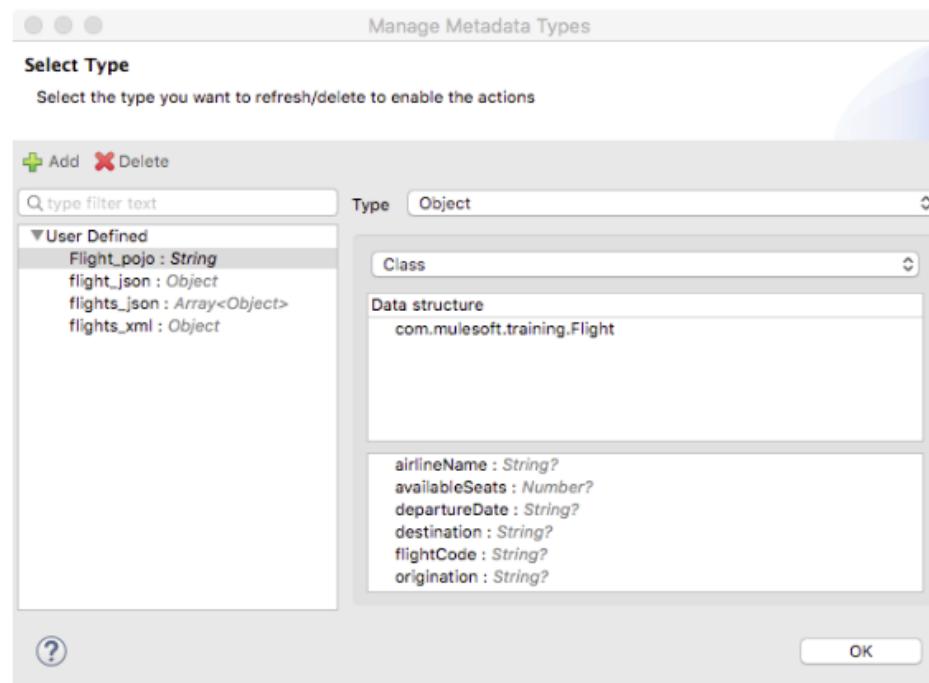


35. In the Data structure section, click the Click here to select an option link.
36. In the drop-down menu that appears, select Java object.

37. In the dialog box that opens, type fli and then in the list of classes that appears, select Flight – com.mulesoft.training.com.



38. Click OK.



39. In the Manage Metadata Types dialog box, click OK.

Review the metadata definition file

40. Return to application-types.xml in src/main/resources.

```
<?xml version='1.0' encoding='UTF-8'?>
<types:mule xmlns:types="http://www.mulesoft.org/schema/mule/types">
  <types:catalog>
    <types:type name="flights_json" format="json">
      <types:example format="json" location="examples/flights-example.json"/>
    </types:type>
    <types:type name="flights_xml" format="xml">
      <types:example format="xml" element="{http://soap.training.mulesoft.com/}list"/>
    </types:type>
    <types:type name="flight_json" format="json">
      <types:example format="json" location="examples/flight-example.json"/>
    </types:type>
    <types:type name="Flight_pojo" format="java">
      <types:shape format="java" element="com.mulesoft.training.Flight"/>
    </types:type>
  </types:catalog>
</types:mule>
```

41. Close the file.

Summary



- Separate functionality into **multiple applications** to allow managing and monitoring of them as separate entities
- Mule applications are **Maven** projects
 - A project's **POM** is used by Maven to build, report upon, and document a project
 - Maven builds an artifact (a Mule deployable archive JAR) from multiple dependencies (module JARs)
- Separate application functionality into **multiple configuration files** for easier development and maintenance
 - Encapsulate **global elements** into their own separate configuration file
- Share resources between applications by creating a **shared domain**

- Define **application properties** in a YAML file and reference them as \${prop}
- Application **metadata** is stored in application-types.xml
- Create applications composed of multiple **flows** and **subflows** for better readability, maintenance, and reusability
- Use **Flow Reference** to calls flows synchronously
- Use the **VM connector** to pass messages between flows using asynchronous queues

DIY Exercise 7-1: Track data through a Mule application

Time estimate: 1 hour

Objectives

In this exercise, you track data through an application that has several flows. You will:

- Track data through flows when a Flow Reference is used.
- Replace a Flow Reference with an HTTP Request to provide a connection boundary.
- Track data through flows when a connection boundary is in place.
- Ensure Mule event data is accessible in a flow before and after it crosses a connection boundary.

Scenario

You are tasked with tracking crucial data for a project for logging and auditing purposes. You need to save the Mule event payload, variables, and attributes as data passes between different flows and servers.

Import the starter project

Import `/files/module07/structure-mod07-starter.jar` (in the `MUFundamentals_DIYexercises.zip` that you can download from the Course Resources) into Anypoint Studio.

Debug the project

Debug the project and make a GET request to <http://localhost:8081/flights?destination=SFO> with a header named Requester-ID set to "AJ46".

Answer the following questions

- What is the result of the GET request to /flights?
- Are the payload, variables, and attributes accessible in all of the flows in the Mule application?

Replace the mockData Flow Reference with an HTTP Request

Replace the mockData Flow Reference with an HTTP Request configured to make a GET request to <http://mu.learn.mulesoft.com/american/{destination}>, where destination is a URI parameter that passes the value from the destination query parameter.

Answer the following questions

- What is the result of a web client request to /flights?
- Are the payload, variables, and attributes accessible in all of the flows in the Mule application?
- What data is accessible after crossing a connection boundary (with the HTTP Request)?

Make the header value accessible in all flows

Modify the Mule application to make the requester-id header accessible in all of the flows.

In the Transform Message component for the processData flow, add a date key that references the date header returned from the HTTP GET request to the `mu.learn.mulesoft.com/{destination}` endpoint.

Verify your solution

Load the solutions `/files/module07/solutions/structure-mod07-solution1.jar` (in the `MUFundamentals_DIYexercises.zip` that you can download from the Course Resources) and compare your solution.

DIY Exercise 7-2: Define and use application properties

Time estimate: 1.5 hours

Objectives

In this exercise, you configure a Mule application to preserve data after it crosses a connection boundary and then define and use application properties for all HTTP Request operation configuration information. You will:

- Preserve data submitted to a flow using a connection boundary.
- Make HTTP requests asynchronously.
- Add property placeholders to a Mule application.
- Create a configuration properties YAML file to store property placeholders in a hierarchy.
- Save common values to global properties.

Scenario

Additional data processing requirements are added for the project. To process the data, you need to make a request to an external endpoint (simulated by making an HTTP Request to another flow in your Mule application). You are also tasked with making the project more maintainable by defining and using application properties for the configuration information for all HTTP Request operations.

Import the starter project

Use your solution from exercise 7-1 or import `/files/module07/structure-mod07-solution1.jar` (in the `MUFundamentals_DIYexercises.zip` that you can download from the Course Resources) into Anypoint Studio.

Run the project

Run the project and make a request to `http://localhost:8081/flights?destination=SFO` with a header named Requester-ID set to "AJ46". Verify that the payload, the destination variable, and the Requester-ID header are accessible in all of the Mule application's flows.

Add an HTTP Listener to the processData flow

Add an HTTP Listener to the processData flow and configure the HTTP Listener to listen for POST requests to `/processData`.

Replace the processData Flow Reference component with an HTTP Request

Replace the processData Flow Reference component with an HTTP Request configured to make a POST request to `http://localhost:8081/processData`. Run the project and make the same request.

Answer the following questions

- What is the result of the request to /flights?
- Are the payload, variables, and attributes accessible in all of the flows in the Mule application?
- What data is accessible within the processData flow?
- What data is still accessible in the flow after completing the HTTP Request operation?

Make all data accessible after the external call

Modify the Mule application to preserve the destination, requester-id, and date values. Rename the date key to flight-date-return in the Transform Message component in the processData flow.

Make the HTTP Request's POST occur asynchronously

Move the HTTP Request into an Async scope. Run the project and make the same request.

Answer the following questions

- What is the result of the request to /flights?
- Are the payload, variable, and attributes accessible throughout all flows called in the Mule application?
- What data is preserved within the processData flow?

Add and use property placeholders and configuration properties files

Create a YAML configuration property file with the name env-properties.yaml. Create a \${http.port} property placeholder that is set to 8081 in env-properties.yaml. Add a \${http.host} property placeholder set to "0.0.0.0".

Next, add property placeholders for all configuration information for all HTTP Request operations including host, port, paths, and url.

Once all properties have been created in env-properties.yaml, refactor the Mule application to refer to these property values in env-properties.yaml. There should be no hard-coded values in any of the HTTP operations or HTTP configurations.

Structure the Mule application to increase project maintainability

Move all common connector configurations and the configuration property to a Mule configuration file with the name global.xml.

Add an environment system property name to the configuration file name

Create a configuration property to refer to the configuration properties file \${env}-properties.yaml. In a debug configuration, set the VM argument -Denv=dev, then verify you can run the Mule application and submit requests with a REST client to <http://localhost:8081/flights?destination=SFO> with a header named Requester-ID set to "AJ46".

Answer the following questions

- What happens if you do not set the -M-Denv=dev parameters when you run the Mule application?
- What happens if you instead try to use -M-Denv=stage when you run the Mule application?

Verify your solution

Load the solutions `/files/module07/solutions/structure-mod07-solution2.jar` (in the `MUFundamentals_DIYexercises.zip` that you can download from the Course Resources) and compare your solution.

Answer the following questions

- Within one Mule application, are variables and attributes passed from one flow to another when called via a Flow Reference component?
- Within one Mule application, are variables and attributes transferred from one connection to another (HTTP, JMS, VM)?
- Within one Mule application, do variables get preserved after a response from a connection is returned?
- Within one Mule application, do attributes get preserved after a response from a connection is returned?
- What is another method to transfer variables from one connection to another connection?