



MuleSoft®

# Anypoint Platform Essentials

## Student Manual

Mule Runtime 3.7.1  
Anypoint Studio 5.2.1  
August 21, 2015

## Table of Contents

<b>SETUP INSTRUCTIONS.....</b>	<b>4</b>
<b>MODULE 2: BUILDING INTEGRATION APPLICATIONS WITH ANYPOINT STUDIO...7</b>	
Walkthrough 2-1: Create your first Mule application .....	8
Walkthrough 2-2: Run, test, and debug an application .....	12
Walkthrough 2-3: Read and write message properties.....	17
Walkthrough 2-4: Read and write variables .....	20
<b>MODULE 3: CONSUMING WEB SERVICES.....</b>	<b>23</b>
Walkthrough 3-1: Consume a RESTful web service.....	24
Walkthrough 3-2: Pass arguments to a RESTful web service .....	29
Walkthrough 3-3: Consume a RESTful web service that has a RAML definition .....	33
Walkthrough 3-4: Consume a SOAP web service .....	40
<b>MODULE 4: CONNECTING TO ADDITIONAL RESOURCES .....</b>	<b>46</b>
Walkthrough 4-1: Connect to a database (MySQL).....	47
Walkthrough 4-2: Connect to a file (CSV).....	53
Walkthrough 4-3: Connect to a JMS queue (ActiveMQ).....	58
Walkthrough 4-4: Connect to a SaaS application (Salesforce).....	65
Walkthrough 4-5: Find and install not-in-the-box connectors .....	73
<b>MODULE 5: TRANSFORMING DATA.....</b>	<b>76</b>
Walkthrough 5-1: Load external content into a message .....	77
Walkthrough 5-2: Write your first DataWeave transformation .....	80
Walkthrough 5-3: Transform basic Java, JSON, and XML data structures .....	85
Walkthrough 5-4: Transform complex data structures .....	91
Walkthrough 5-5: Use DataWeave operators .....	104
Walkthrough 5-6: Transform data sources that have associated metadata .....	115
Walkthrough 5-7: Pass arguments to a SOAP web service .....	125
Walkthrough 5-8: Transform a data source to which you add custom metadata.....	129
<b>MODULE 6: REFACTORIZING MULE APPLICATIONS .....</b>	<b>135</b>
Walkthrough 6-1: Separate applications into multiple configuration files.....	136
Walkthrough 6-2: Encapsulate global elements in a separate configuration file .....	141
Walkthrough 6-3: Create and run multiple applications .....	143
Walkthrough 6-4: Create and reference flows and subflows .....	150

<b>MODULE 7: HANDLING ERRORS .....</b>	<b>156</b>
Walkthrough 7-1: Handle a messaging exception .....	157
Walkthrough 7-2: Handle multiple messaging exceptions .....	160
Walkthrough 7-3: Create and use global exception handlers .....	168
Walkthrough 7-4: Specify a global default exception strategy .....	172
<b>MODULE 8: CONTROLLING MESSAGE FLOW .....</b>	<b>175</b>
Walkthrough 8-1: Multicast a message .....	176
Walkthrough 8-2: Route messages based on conditions .....	182
Walkthrough 8-3: Filter messages .....	188
Walkthrough 8-4: Pass messages to an asynchronous flow .....	194
<b>MODULE 9: PROCESSING RECORDS .....</b>	<b>196</b>
Walkthrough 9-1: Process items in a collection individually .....	197
Walkthrough 9-2: Create a batch job for records in a file .....	202
Walkthrough 9-3: Create a batch job for records in a database .....	207
Walkthrough 9-4: Restrict processing using a poll watermark .....	210
Walkthrough 9-5: Restrict processing using a message enricher and a batch step filter .....	214
<b>MODULE 10: BUILDING RESTFUL INTERFACES WITH RAML AND APIKIT .....</b>	<b>222</b>
Walkthrough 10-1: Use API Designer to define an API with RAML .....	223
Walkthrough 10-2: Use API Designer to simulate an API.....	228
Walkthrough 10-3: Use Anypoint Studio to create a RESTful API interface from a RAML file .....	234
Walkthrough 10-4: Use Anypoint Studio to implement a RESTful web service .....	239
<b>MODULE 11: DEPLOYING APPLICATIONS .....</b>	<b>244</b>
Walkthrough 11-1: Use application properties .....	245
Walkthrough 11-2: Dynamically specify property files .....	250
Walkthrough 11-3: (Optional) Deploy an application to the cloud.....	252
Walkthrough 11-4: (Optional) Deploy an application on-prem .....	259

# Setup instructions

To complete the exercises in this class, you need:

1. A computer with at least 3GB RAM, 2GHz CPU, and 4GB storage

<https://developer.mulesoft.com/docs/display/current/Hardware+and+Software+Requirements>

These are the requirements to run Anypoint Studio.

2. Internet access with access to the following ports and hosts:

Port	Host
80	mulesoft.com (and subdomains)
80	mulesoft-training.com (and subdomains)
80	cloudhub.io (and subdomains)
80	salesforce.com (and subdomains)
3306	iltmdb.mulesoft-training.com
61616	54.69.115.37

Note: You can use telnet to test the availability of these ports. For example, in a Terminal/Command window, enter the following command and see if it connects:

`telnet iltmdb.mulesoft-training.com 3306`

If you are using Windows and need to install a Telnet Client, see

[https://technet.microsoft.com/en-us/library/Cc771275\(v=WS.10\).aspx](https://technet.microsoft.com/en-us/library/Cc771275(v=WS.10).aspx).

If you have restricted internet access and cannot access some of these ports and hosts, you can instead use VirtualBox and a MuleSoft Training server image. Instructions for installing and setting up the server are included below.

You also need to install and/or set up BEFORE class:

1. Java SE Development Kit 8 (JDK 8 NOT JDK 6 or JDK 7)

<http://www.oracle.com/technetwork/java/javase/downloads/index.html>

This is required for Anypoint Studio.

Note: Instructions for checking what version of Java you have can be found here:

[https://www.java.com/en/download/help/version\\_manual.xml](https://www.java.com/en/download/help/version_manual.xml).

2. Anypoint Studio with embedded Mule 3.7 runtime

<https://www.mulesoft.com/lp/dl/studio>

Download, install, and start it to make sure it runs BEFORE class.

Note: If you have any issues, make sure the JDK and Anypoint Studio are BOTH 64-bit or BOTH 32-bit and that you have enough memory (at least 3 GB) to run Anypoint Studio.



### **3. An Anypoint Platform account**

<http://anypoint.mulesoft.com>

*Note: You can sign up for a free, 30-day trial account or you can use your company account (if you already have one).*

### **4. A Salesforce Developer account (NOT a standard account)**

<https://developer.salesforce.com/en/signup>

You CANNOT use a standard account, which will not give you API access.

You should receive an activation email within 5-10 minutes of creating the account. Click the link it contains to activate your email and set a password and password question. You will then be logged in to Salesforce. You can also log in at <http://salesforce.com> using your developer credentials.

### **5. A Salesforce API Access token**

<http://salesforce.com>

In Salesforce, click your name at the top of the screen and select My Settings. On the left side of the page, select Personal > Reset My Security Token and click the Reset Security Token button. A security token will be sent to your email in a few minutes. You will need this token to make API calls to Salesforce from your Mule applications.

### **6. Course materials**

<http://training.mulesoft.com/login.html>

The MuleSoft Learning Management System (LMS) hosts all of the resources, links, and files for the class. Browse to <http://training.mulesoft.com/login.html> and then enter your email and password (the password used when registering for the course). If you did not set a password or forgot it, go to [http://training.mulesoft.com/reset\\_password.html](http://training.mulesoft.com/reset_password.html).

Once you are logged in, click the Anypoint Platform Essentials 3.7 Course link under My Learning Plan. Near the bottom of the page, you will see the course resources. Click the links to download the student manual (PDF), the student files (ZIP), and the course slides (ZIP). If you have restricted internet access and plan to use VirtualBox and the training server image, also download the MuleSoft Training 3.6 Open Virtualization Archive (OVA).

*Note: Some students find it useful to put the student manual on a tablet and reference that during class.*

### **7. VirtualBox (only if you have restricted internet access)**

<https://www.virtualbox.org/wiki/Downloads>

Download, install, and set it up to run the MuleSoft Training server image BEFORE class.

Follow the setup instructions below and/or watch the setup video: <https://vimeo.com/123113815>.

After installing VirtualBox, open it and select File > Import Appliance. Browse to the mulesoft-training3.6.ova file you downloaded in the previous step. After the image is imported, you should see a mulesoft-training3.6 virtual machine listed in the VirtualBox Manager.

Double-click the mulesoft-training3.6 virtual machine to start it. The server needs to be running to complete the course exercises. Verify the server starts. You do not need to log in, but if you

use SSH the username and password are vagrant. To stop the virtual machine, right-click it in the VirtualBox Manager and select Close > Power Off.

The virtual machine uses ports 61111, 8112, 4406, 8111, and 2222. If you have any port conflicts, you will need to change these values. First, power off the virtual machine and then click the Settings button. Click the Network button and expand the Advanced section. Click the Port Forwarding button and change any of the host port values as necessary.

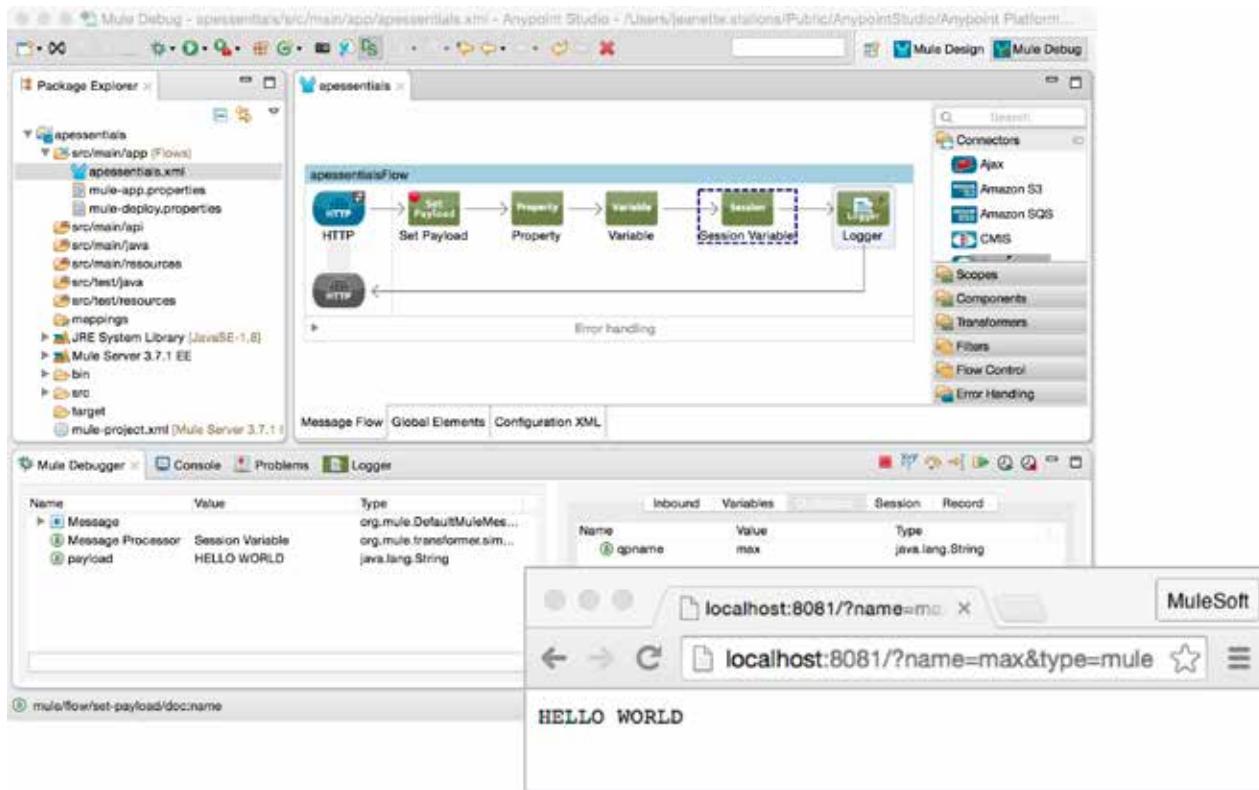
## 8. Mule 3.7 runtime with Mule Management Console (optional)

<http://mulesoft-training.com/mule-runtime-bundle-3.7>

In public classes, you will deploy to the cloud. In private classes, you will deploy to the cloud or an on-prem, standalone Mule runtime. Step-by-step instructions for both are included in the student manual.

If you want to follow the instructions to deploy to an on-prem Mule runtime, download the bundle from the above link before class. You will start the server during class.

# Module 2: Building Integration Applications with Anypoint Studio



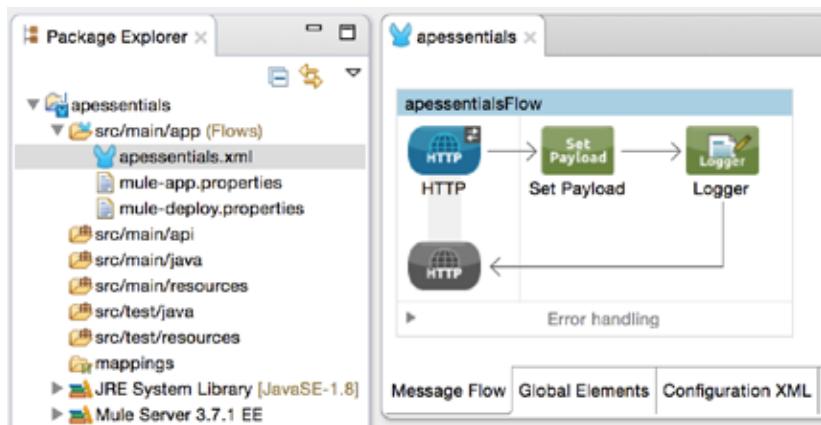
In this module, you will learn:

- About Mule applications, flows, messages, and message processors.
- To use Anypoint Studio to create flows graphically using connectors, transformers, components, scopes, and flow control elements.
- To build, run, test, and debug Mule applications.
- To read and write message properties.
- To write expressions with Mule Expression Language (MEL).
- To create variables.

## Walkthrough 2-1: Create your first Mule application

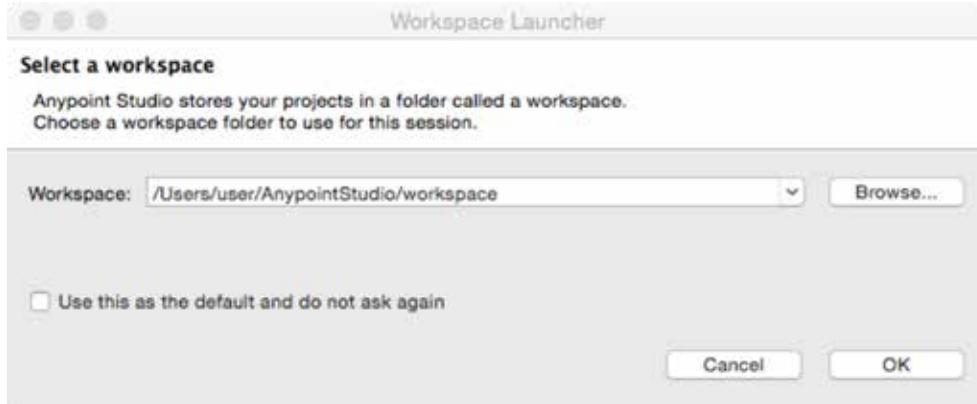
In this walkthrough, you will build your first Mule application. You will:

- Create a new Mule project with Anypoint Studio.
- Add a connector to receive requests at an endpoint.
- Display a message in the Anypoint Studio console.
- Set the message payload.



### Launch Anypoint Studio

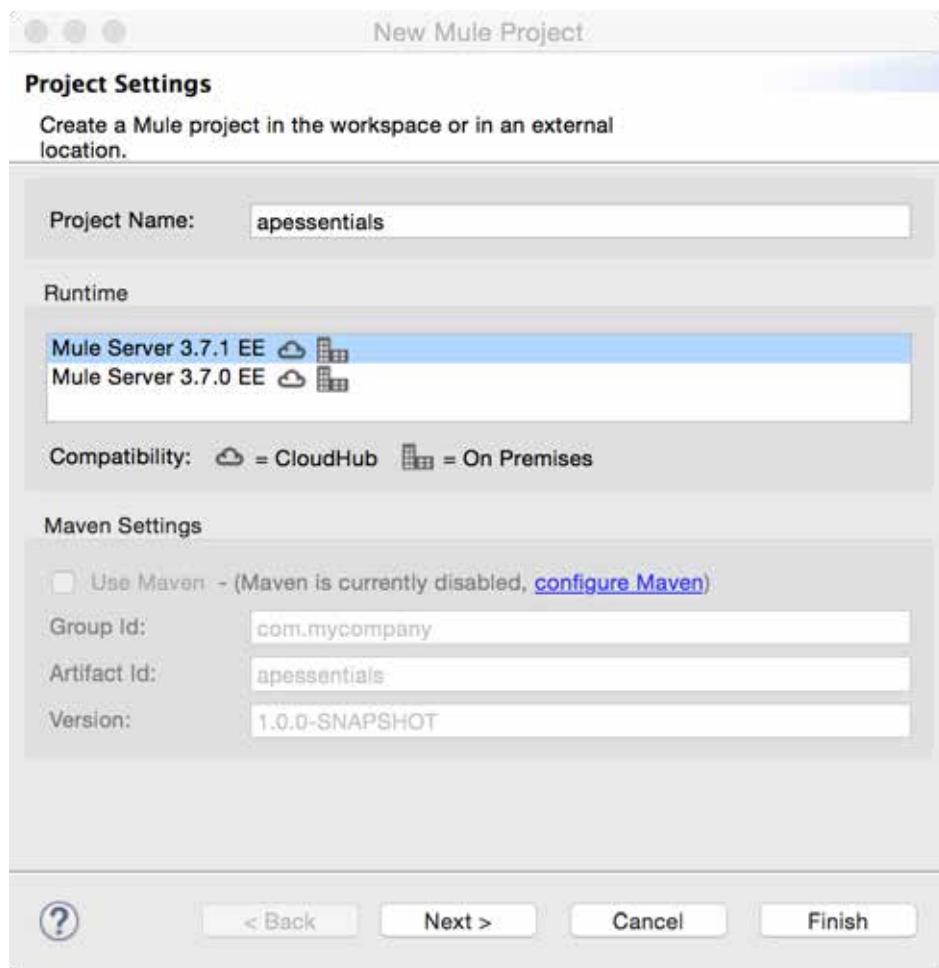
1. Open Anypoint Studio.
2. In the Workspace Launcher dialog box, look at the location of the default workspace; change the workspace location if you want.



3. Click OK to select the workspace; Anypoint Studio should open.
4. If you get a Welcome Page, click the X on the tab to close it.
5. If you get an Updates Available pop-up in the lower-right corner of the application, click it and install the available updates.

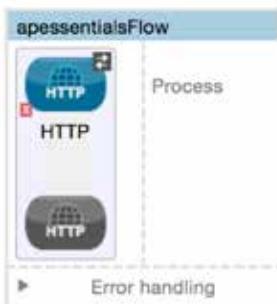
## Create a project

6. Select File > New > Mule Project.
7. Set the Project Name to apessentials.
8. Ensure the Runtime is set to the latest version of the Mule Server.
9. Click Finish.



## Create an HTTP connector endpoint to receive requests

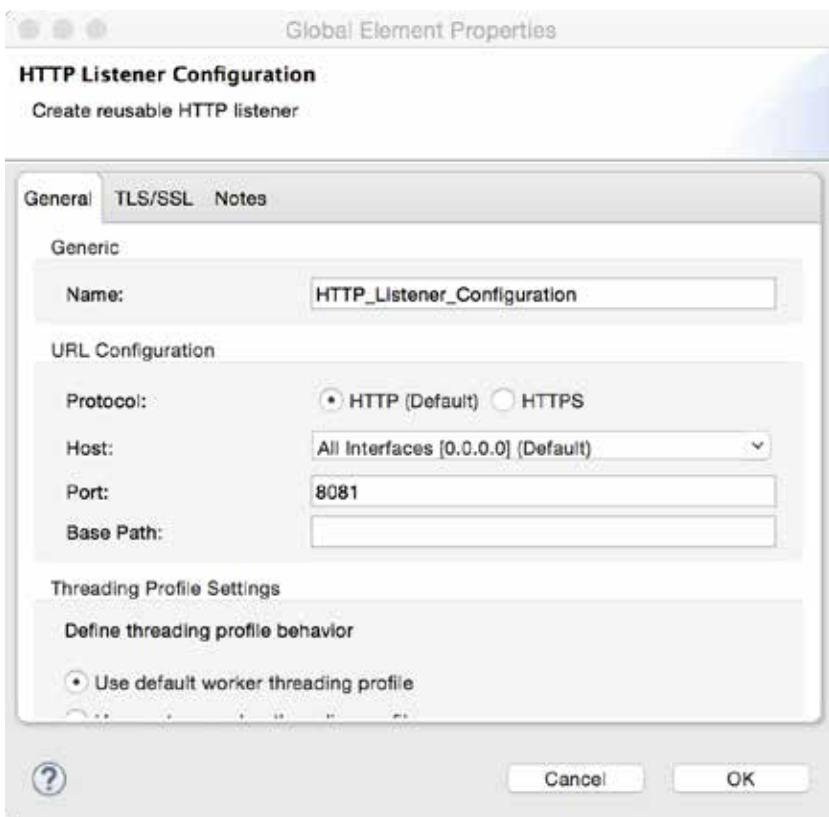
10. Drag an HTTP connector from the palette to the canvas.



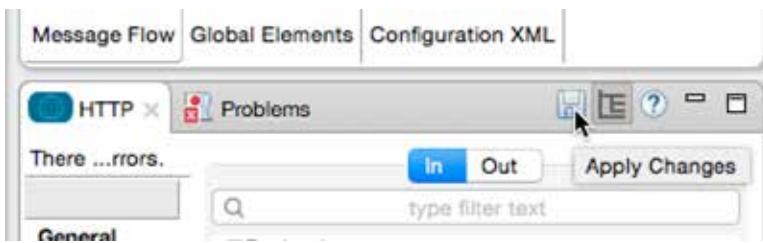
11. Double-click the HTTP connector endpoint.
12. In the Mule Properties view, click the Add button next to connector configuration.



13. In the Global Element Properties dialog box, look at the default values and click OK.

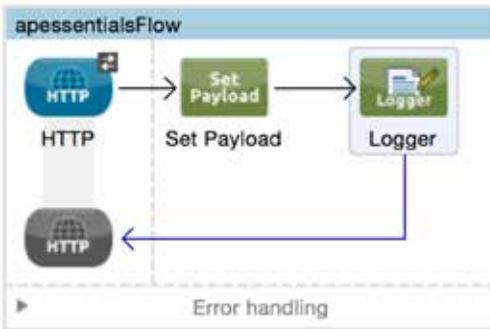


14. Click the Apply Changes button; the errors in the Problems view should disappear.



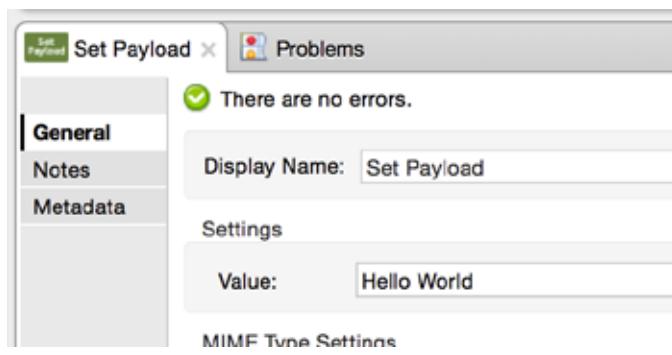
## Display data

15. Drag a Set Payload transformer from the palette into the process section of the flow.
16. Drag in a Logger component and drop it after the Set Payload transformer.



## Configure the Set Payload transformer

17. Double-click the Set Payload transformer.
18. In the Properties view, set the value field to Hello World.



19. Click the Configuration XML tab at the bottom of the canvas and examine the corresponding XML.

```
8  http://www.mulesoft.org/schema/mule/http http://www.mulesoft.org/schema/mule/http/current/mule-http.xsd
9      <http:listener-config name="HTTP_Listener_Configuration" host="0.0.0.0" port="8081" doc:name="HTTP"
10     <flow name="apessimalsFlow">
11         <http:listener config-ref="HTTP_Listener_Configuration" path="/" doc:name="HTTP"/>
12         <set-payload value="Hello World" doc:name="Set Payload"/>
13         <logger level="INFO" doc:name="Logger"/>
14     </flow>
15 </mule>
16
```

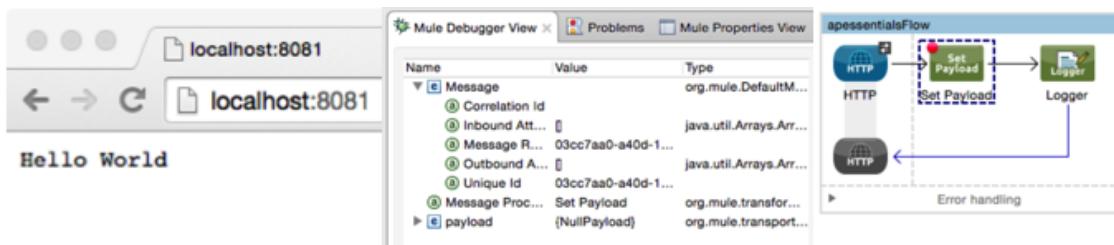
Message Flow Global Elements Configuration XML

20. Click the Message Flow tab to return to the canvas.
21. Click the Save button or press Ctrl+S to save the file.

## Walkthrough 2-2: Run, test, and debug an application

In this walkthrough, you will run, test, and debug your first Mule application. You will:

- Run a Mule application using the embedded Mule runtime.
- Make an HTTP request to the endpoint via a web browser or a tool like cURL or Postman.
- Receive a response with the text Hello World.
- Redeploy an application.
- Use the Mule Debugger to debug an application.



### Run the application

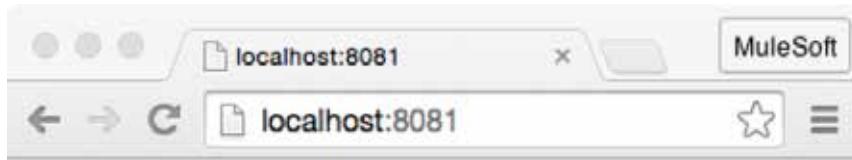
1. From the main menu bar, select Run > Run As > Mule Application.
2. Watch the Console view; it should display information letting you know that both the Mule runtime and the apessimist application started.

The screenshot shows the 'Console' tab in the Mule Studio interface. It displays deployment logs for the 'apessimist' application. The logs show the application starting, the deployment directory watcher being active, and the Mule runtime kicking every 5000ms. It also shows the startup summary deployment listener output, which includes domain and application status information. The 'default' domain is listed as deployed.

```
apessimist [Mule Applications] /Library/Java/JavaVirtualMachines/jdk1.8.0_45.jdk/Contents/Home/bin/java (Aug 19, 2015, 9:59:51 AM)
+++++
+ Started app 'apessimist'
+
INFO 2015-08-19 09:59:55,360 [main] org.mule.module.launcher.DeploymentDirectoryWatcher:
+++++
+ Mule is up and kicking (every 5000ms)
+
INFO 2015-08-19 09:59:55,407 [main] org.mule.module.launcher.StartupSummaryDeploymentListener:
*****
*      - + DOMAIN + - -      * - - + STATUS + - - *
*****
* default                                * DEPLOYED   *
*****
*      - - + APPLICATION + - -      *      - - + DOMAIN + - -      * - - + STATUS + - - *
*****
* apessimist                           * default          * DEPLOYED   *
*****
```

## Test the application

- Send an HTTP request to <http://localhost:8081> through a browser or tool like cURL or Postman; you should see Hello World displayed.



- Return to the console in Anypoint Studio.
- Examine the last entry.

```
Mule Properties Problems Console x
apessentials [Mule Applications] /Library/Java/JavaVirtualMachines/jdk1.8.0_45.jdk/Contents/Home/bin/java (Aug 19, 2015, 9:59:51 AM)
Message properties:
  INVOCATION scoped properties:
    INBOUND scoped properties:
      accept=text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
      accept-encoding=gzip, deflate, sdch
      accept-language=en-US,en;q=0.8
      connection=keep-alive
      host=localhost:8081
      http.listener.path=/
      http.method=GET
      http.query.params=ParameterMap{[]}
      http.query.string=
      http.relative.path=/
      http.remote.address=/0:0:0:0:0:0:1:57174
      http.request.path=/
      http.request.uri=/
      http.scheme=http
      http.uri.params=ParameterMap{[]}
      http.version=HTTP/1.1
      upgrade-insecure-requests=1
      user-agent=Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10_2) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/45.0.2423.122 Safari/537.36
    OUTBOUND scoped properties:
    SESSION scoped properties:
}
```

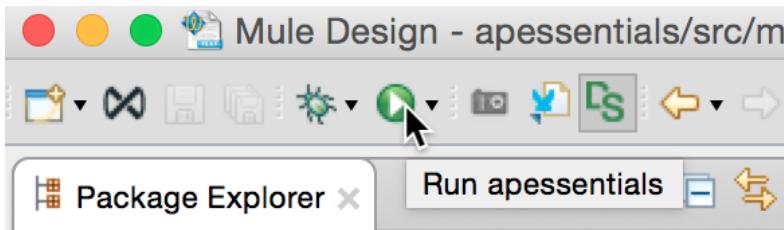
The screenshot shows the 'Console' tab in Anypoint Studio. It displays the message properties for a recent request. The properties listed include various HTTP headers and parameters, such as 'accept', 'accept-encoding', 'accept-language', 'connection', 'host', 'http.listener.path', 'http.method', 'http.query.params', 'http.query.string', 'http.relative.path', 'http.remote.address', 'http.request.path', 'http.request.uri', 'http.scheme', 'http.uri.params', 'http.version', 'upgrade-insecure-requests', and 'user-agent'. The 'user-agent' property shows a specific version of Mozilla's browser engine.

- Click the red Terminate button to stop the application and the Mule runtime.
- Answer the following questions.
  - What triggered all the output you saw in the console?
  - What is the last thing displayed?
  - What Java class represents the payload?
  - What are the inbound and outbound properties on the message?

## Rerun the application

- Change the value of the Set Payload transformer to a different value.
- Save the file.

10. Click the Run button and watch the console; you should see the application is redeployed but the Mule runtime is also restarted.



*Note: You may want to modify your perspective so you always see the console. In Eclipse, a perspective is a specific arrangement of views in specific locations. You can rearrange the perspective by dragging and dropping tabs and views to different locations. Use the Window menu in the main menu bar to save and reset perspectives.*

## Redeploy the application

11. Change the value of the Set Payload transformer again to a different value.
12. Click the Apply Changes button in the upper-right corner of the Properties view and watch the console; you should see the application is redeployed but the Mule runtime is not restarted.

```
<terminated> apessentials [Mule Applications] [/Library/java/javaVirtualMachines/jdk1.8.0_45.jdk/Contents/Home/bin/java Aug 19, 2015, 9:59:41,298 [Mule.app.deployer.monitor.1.thread.1] org.mule.modules.oauth2.provider.agen
=====
+ DevKit Extensions (0) used in this application
+
=====
INFO 2015-08-19 10:03:41,300 [Mule.app.deployer.monitor.1.thread.1] org.mule.lifecycle.AbstractLifecycleM
INFO 2015-08-19 10:03:41,300 [Mule.app.deployer.monitor.1.thread.1] org.mule.construct.FlowConstructLifecycle
INFO 2015-08-19 10:03:41,301 [Mule.app.deployer.monitor.1.thread.1] org.mule.processor.SedaStageLifecycle
INFO 2015-08-19 10:03:41,311 [Mule.app.deployer.monitor.1.thread.1] org.mule.module.management.agent.Wrap
INFO 2015-08-19 10:03:41,315 [Mule.app.deployer.monitor.1.thread.1] org.mule.DefaultMuleContext:
=====
* Application: apessentials
* OS encoding: /, Mule encoding: UTF-8
*
* Agents Running:
*   DevKit Extension Information
*   Batch module default engine
*   JMX Agent
*   Wrapper Manager
=====
INFO 2015-08-19 10:03:41,315 [Mule.app.deployer.monitor.1.thread.1] org.mule.module.launcher.MuleDeployment
=====
+ Started app 'apessentials'
=====
```

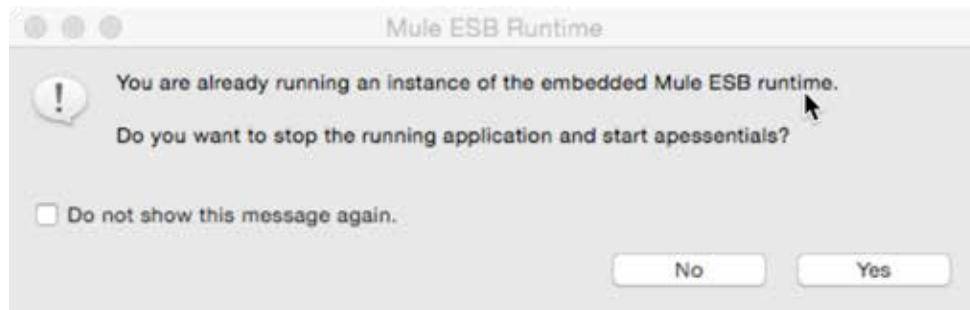
The screenshot shows the Eclipse IDE's Console view with deployment logs. The logs indicate the application 'apessentials' was started successfully. It shows various system and application-specific log entries, including lifecycle events for components like FlowConstructLifecycle and SedaStageLifecycle, and information about agents running.

## Debug the application

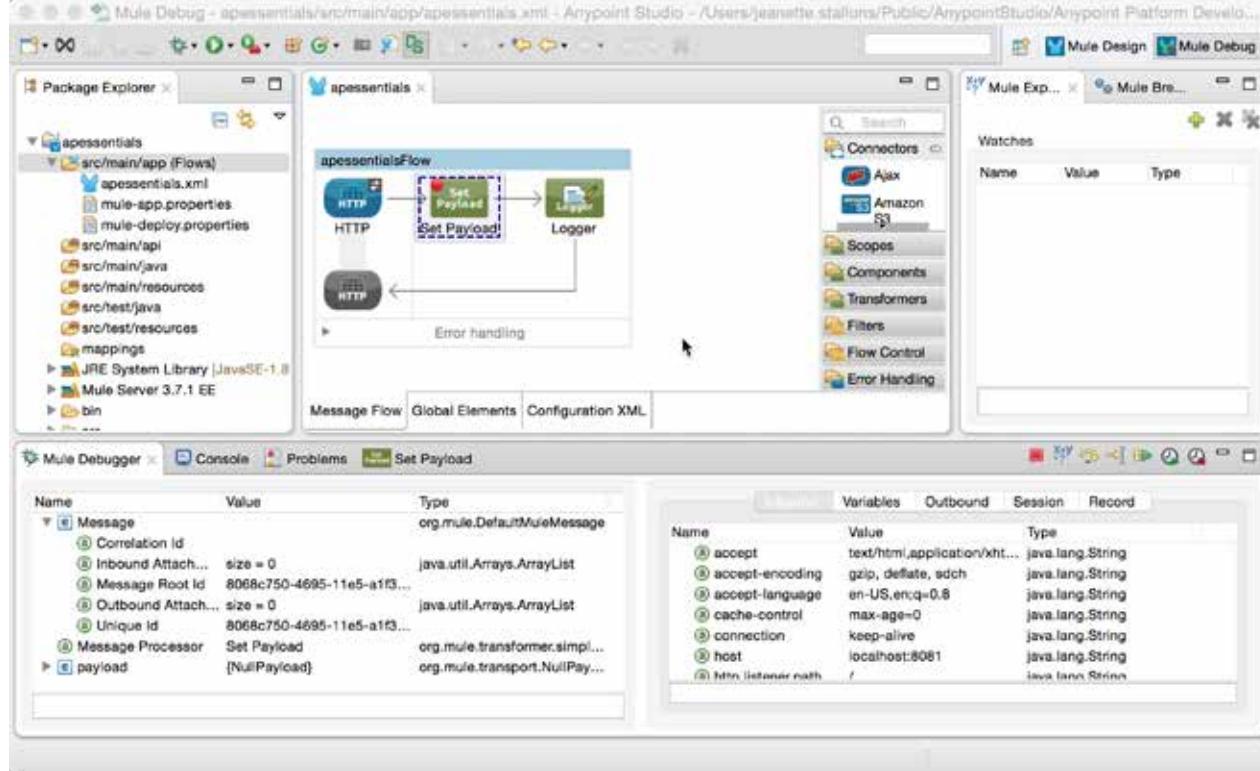
13. Right-click the Set Payload component and select Toggle breakpoint.
14. Select Run > Debug or click the Debug button in the main menu bar.
15. If you get an Accept incoming network connections dialog box, click Allow.
16. If you get a Confirm Perspective Switch dialog box, click Yes.

*Note: If you do not want to get this dialog box again, check Remember my decision.*

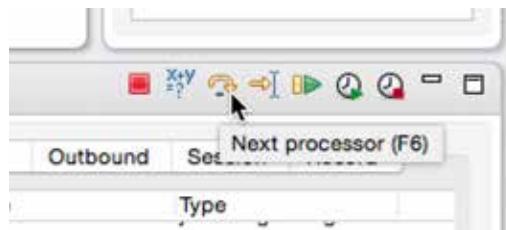
17. In the Mule ESB Runtime window, click Yes to stop the Mule runtime and restart it connected to the Mule Debugger.



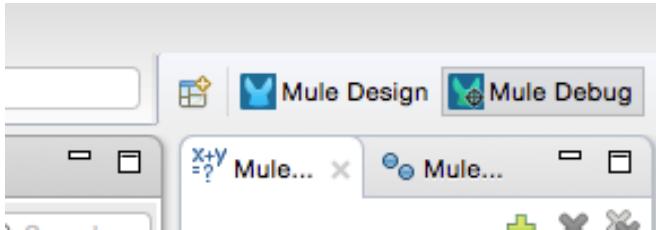
18. Make another request to <http://localhost:8081> with a browser or other tool like cURL or Postman.  
19. Return to Anypoint Studio and look at the Mule Debugger view.  
20. Drill-down and explore the properties and variables.



21. Click the Next processor button.



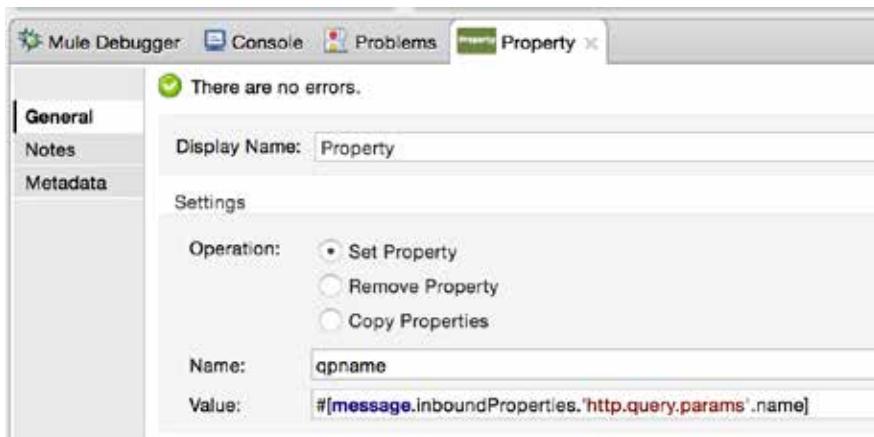
22. Look at the new value of the payload.
23. Step through the rest of the application.
24. Stop the application and the Mule runtime.
25. Click the Mule Design tab in the upper-right corner of the application to switch perspectives.



## Walkthrough 2-3: Read and write message properties

In this walkthrough, you will manipulate message properties. You will:

- Write MEL expressions.
- Use the Debugger to read inbound and outbound message properties.
- Use the Property transformer to set outbound message properties.



### Use an expression to set the payload

1. Navigate to the Properties view for the Set Payload transformer.
2. Change the value to an expression.

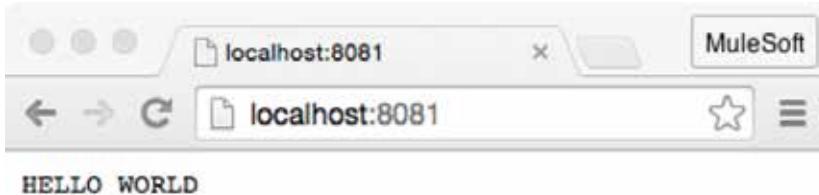
```
#['Hello World']
```

3. Click the Apply Changes button and run the application; you have to run it instead of redeploying it because you stopped the Mule runtime when you stopped the Debugger.
4. Make a request to <http://localhost:8081>; the application should work as before.
5. Return to the Set Payload expression and use autocomplete to use the toUpperCase() method to return the value in upper case.

```
#['Hello World'].toUpperCase()
```

*Note: Press Ctrl+Space to trigger autocomplete if it does not appear.*

6. Click the Apply Changes button to redeploy the application.
7. Make a request to the endpoint; the return string should now be in upper case.



## Use an expression to display info to the console

8. Navigate to the Properties view for the Logger component.
9. Set the message to display the http.query.params property of the message inbound properties.

Message:	# <b>[message.inboundProperties.'http.query.params']</b>
Level:	INFO (Default)
Category:	

10. Apply the changes and debug the application.
11. Make a request to the endpoint with a couple of query parameters; for example, <http://localhost:8081/?name=max&type=mule>.
12. Return to the Mule Debugger view and locate your query parameters.

Inbound	Variables	Outbound	Session	Record
Name	Value	Type		
HTTP_METHOD	GET	java.lang.String		
http.query.params	size = 2	org.mule.module.http.int...		
0	name=max	java.util.AbstractMap\$Si...		
key	name	java.lang.String		
value	max	java.lang.String		
1	type=mule	java.util.AbstractMap\$Si...		
http.query.string	name=max&type=mule	java.lang.String		
http.relative.path	/	java.lang.String		
http.remote.address	/0:0:0:0:0:0:1:58091	java.lang.String		

13. Step through the application and watch the property values.
14. Navigate to the console; you should see a ParameterMap object listed.  

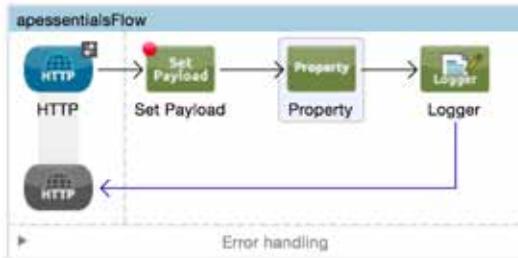
```
INFO 2015-08-19 10:29:32,305 [[apessentials].HTTP_Listener_Configuration.worker.01] org.mule.api.processor.LoggerMessageProcessor: ParameterMap[[name=[max], type=[mule]]]
```
15. Modify the Logger to display one of your query parameters.  

```
#[message.inboundProperties.'http.query.params'.name]
```
16. Click Apply Changes to redeploy the application and make a request to the endpoint with query parameters again.
17. Click the Resume button in the Mule Debugger view.
18. Return to the console; you should now see the value of the parameter displayed.  

```
INFO 2015-08-19 10:27:14,586 [[apessentials].HTTP_Listener_Configuration.worker.01] org.mule.api.processor.LoggerMessageProcessor: max
```

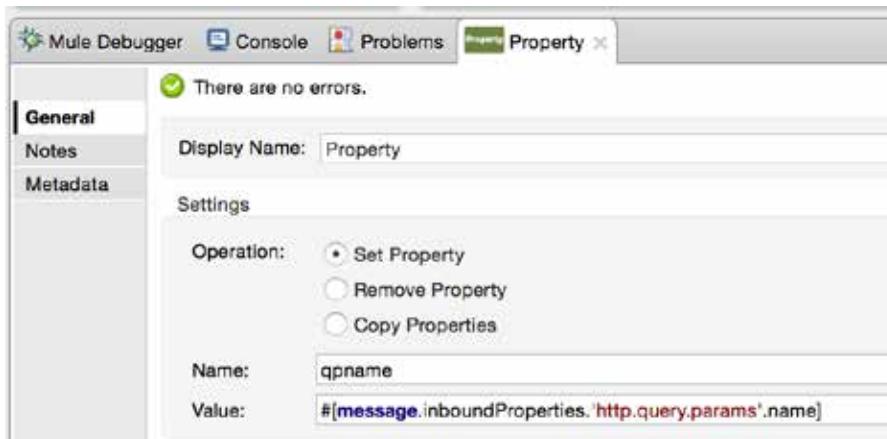
## Set an outbound property

19. Add a Property transformer between the Set Payload and Logger processors.



*Note: The Message Properties transformer has been deprecated; it has been replaced by the collection of Property, Variable, Session Variable, and Attachment transformers.*

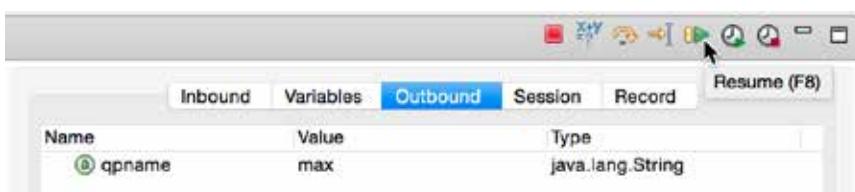
20. In the Properties view for the Property transformer, select Set Property.
21. Set the name to qpname (or some other value) and the value to an expression that evaluates one of your query parameters.



22. Modify the Logger to display the value of this new outbound property.

```
# [message.outboundProperties.qpname]
```

23. Click the Apply Changes button to redeploy the application with a connection to the Mule Debugger.
24. Make another request to the endpoint with query parameters.
25. Click the Outbound tab in the Mule Debugger view.
26. Step to the Logger; you should see your new outbound property.

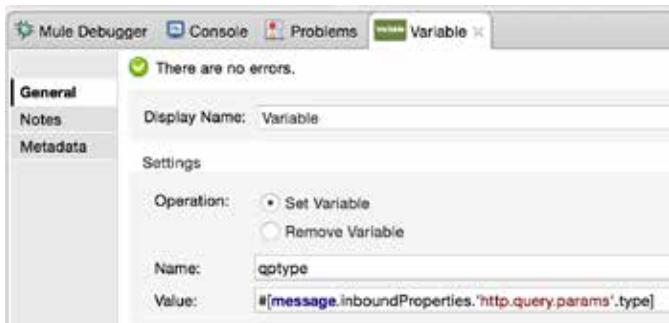


27. Click the Resume button.
28. Look at the console; you should see the value of your variable displayed.

## Walkthrough 2-4: Read and write variables

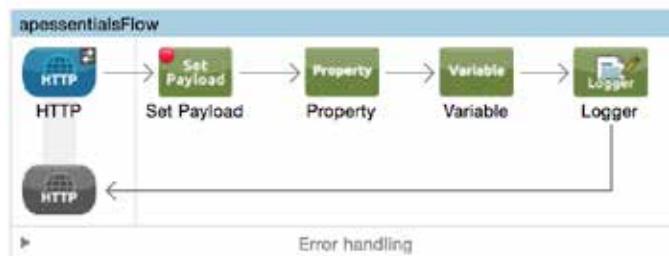
In this walkthrough, you will create flow and session variables. You will:

- Use the Variable transformer to create flow variables.
- Use the Session transformer to create session variables.

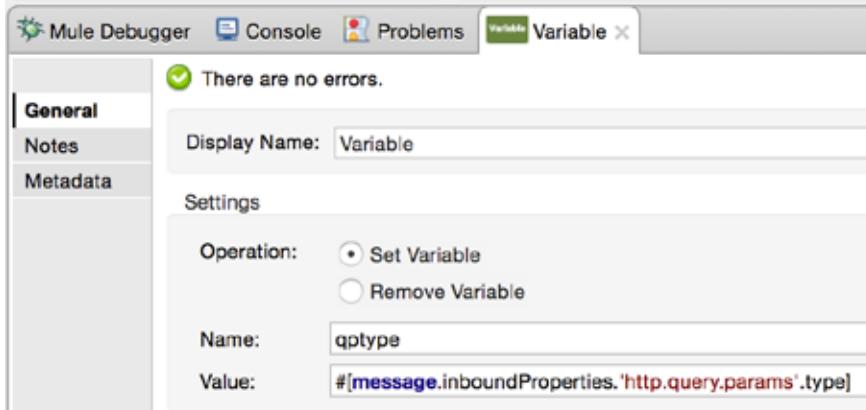


### Create a flow variable

1. Add a Variable transformer between the Property and Logger processors.



2. In the Variable Properties view, select Set Variable.
3. Set the name to qptype (or some other value) and the value to your second query parameter.



4. Modify the Logger to also display the value of this new variable.

```
#['Name: ' + message.outboundProperties.qpname + ' Type: ' +  
flowVars.qptype]
```

*Note: The flowVars is optional.*

5. Save the file (or click Apply Changes) to redeploy the application in debug mode.
6. Make a request to the endpoint with query parameters again.
7. Click the Variables tab in the Mule Debugger view.
8. Step to the Logger; you should see your new flow variable.

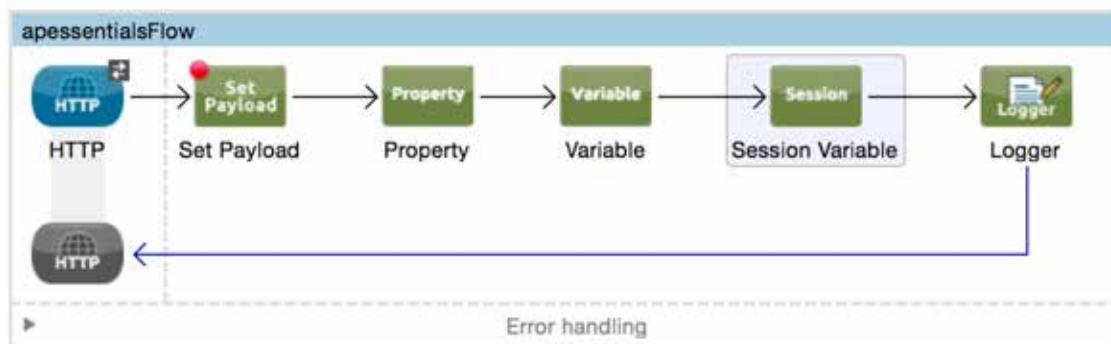
Inbound	Variables	Outbound	Session	Record						
	<table border="1"><thead><tr><th>Name</th><th>Value</th><th>Type</th></tr></thead><tbody><tr><td>qptype</td><td>mule</td><td>java.lang.String</td></tr></tbody></table>	Name	Value	Type	qptype	mule	java.lang.String			
Name	Value	Type								
qptype	mule	java.lang.String								

9. Click the Resume button.
10. Look at the console; you should see the value of your outbound property and the value of your new flow variable displayed.

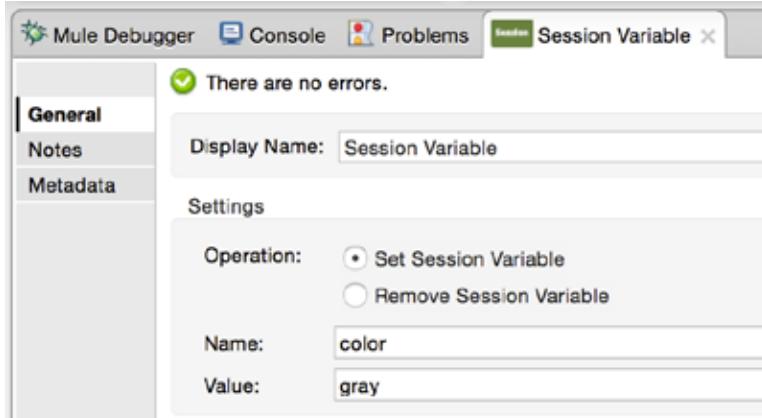
```
INFO 2015-08-19 10:45:33,189 [[apessentials].HTTP_Listener_Configuration.worker.01] org.mule.api.processor.LoggerMessageProcessor: Name: max Type: mule
```

## Create a session variable

11. Add a Session Variable transformer between the Variable and Logger processors.



12. In the Session Variable Properties view, select Set Session Variable.
13. Set the name to color (or some other value) and give it any value, static or dynamic.



14. Modify the Logger to also display the session variable.

```
#[ 'Name: ' + message.outboundProperties.qpname + ' Type: ' +  
flowVars.qptype + ' Color: ' + sessionVars.color]
```

15. Save and redeploy the application and make a request to the endpoint with query parameters again.

16. Click the Session tab in the Mule Debugger view.

17. Step to the Logger; you should see your new session variable.

Inbound	Variables	Outbound	Session	Record
Name	Value	Type		
	color	gray	java.lang.String	

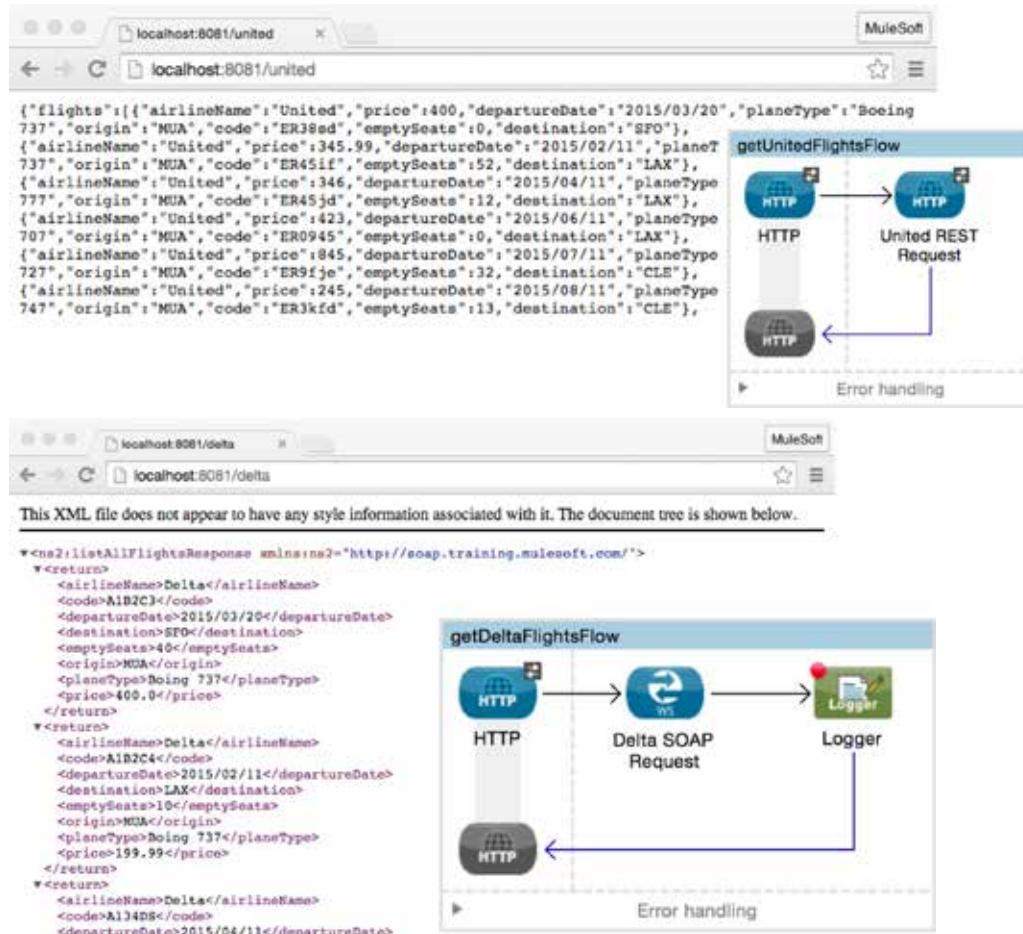
18. Click the Resume button.

19. Look at the console; you should see the value of your session variable displayed.

20. Stop the Mule runtime.

*Note: You will explore the persistence of these variables in a later module, Refactoring Mule Applications.*

# Module 3: Consuming Web Services



## In this module, you will learn:

- About RESTful and SOAP based web services.
- What RAML is and how it can be used.
- To consume RESTful web services with and without RAML definitions.
- To consume SOAP web services.

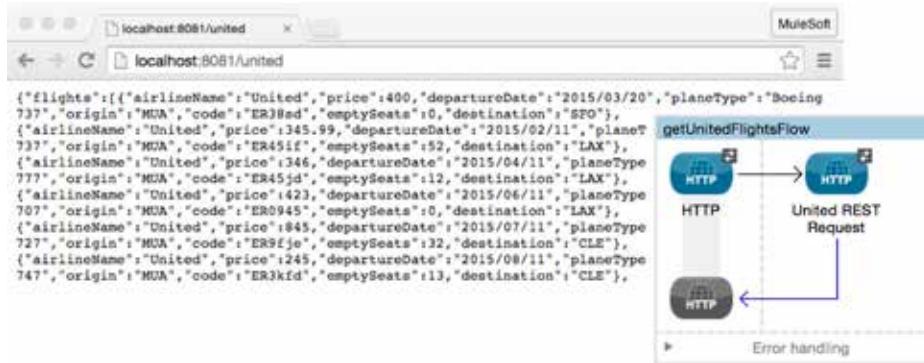


MuleSoft

## Walkthrough 3-1: Consume a RESTful web service

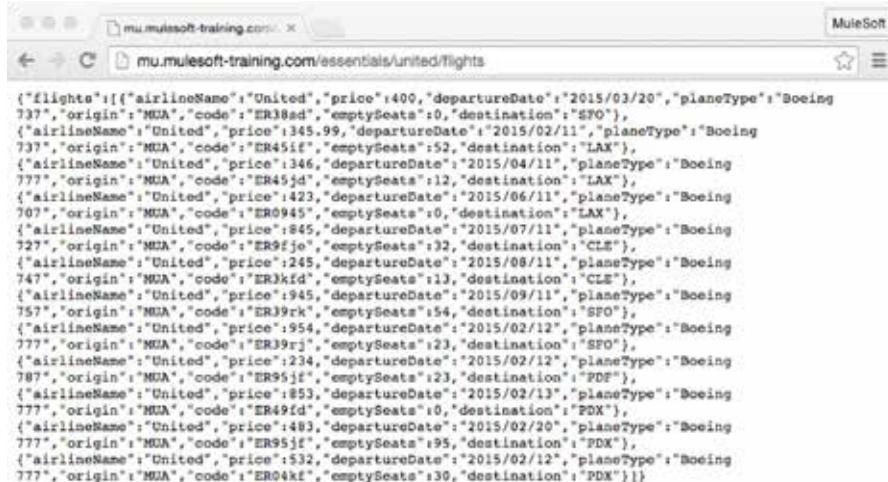
In this walkthrough and many others, you will work on building a Mule United Airline (MUA) application that returns flights for Delta, United, and American airlines. In this walkthrough, you will consume a RESTful web service that returns a list of all United flights as JSON. You will:

- Create a second flow and rename flows.
- Add an HTTP Listener connector endpoint to receive requests at <http://localhost:8081/united>.
- Add an HTTP Request connector endpoint to consume a RESTful web service for United flight data.



### Make a request to the web service

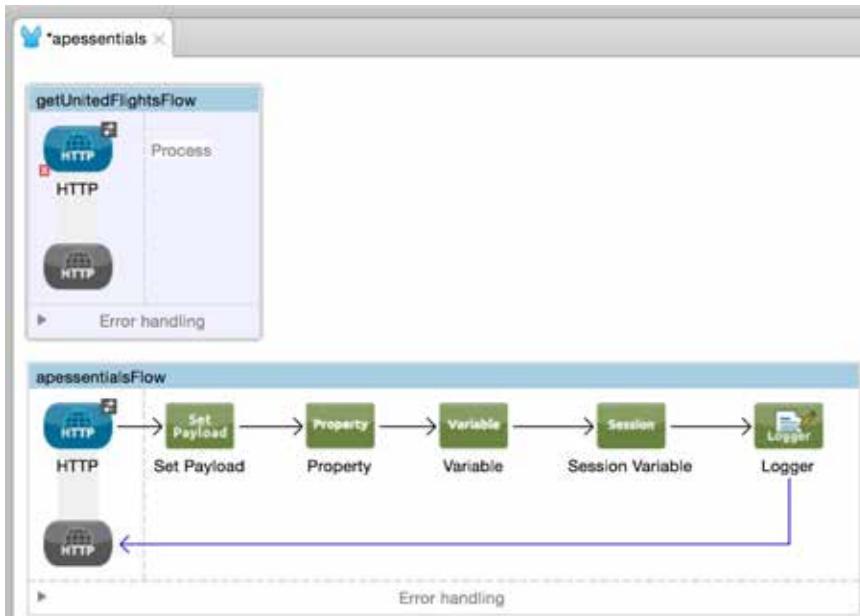
1. In your computer's system explorer, navigate to the student files folder for the course:  
APEssentials3.7\_studentFiles\_{date}.
2. Open the course snippets.txt file.
3. Make a request to the United RESTful web service URL listed in the course snippets.txt file; you should see JSON data for the United flights returned.
4. Look at the destination values; you should see SFO, LAX, CLE, PDX, and PDF.



## Add a new flow with an HTTP Listener connector endpoint

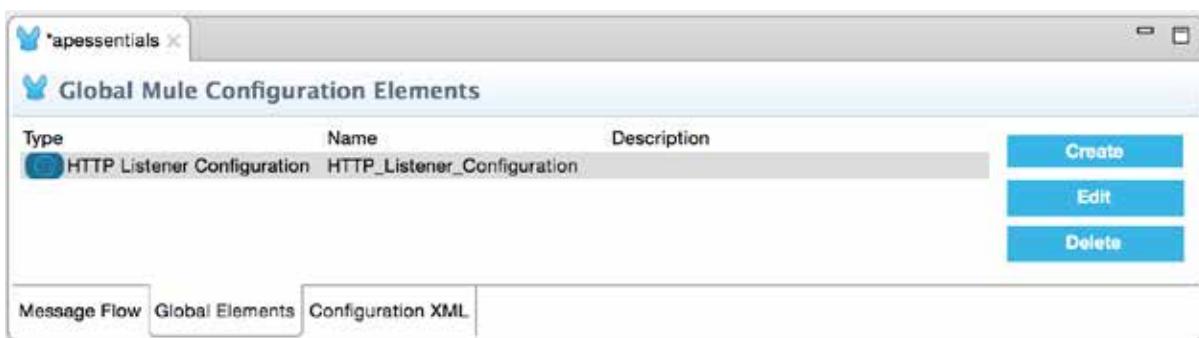
5. Return to apessentials.xml.
6. Drag out another HTTP connector and drop it in the canvas above the existing apessentialsFlow.
7. Double-click the name of the flow in the canvas and give it a new name of getUnitedFlightsFlow.

*Note: You can set the name in the Properties view or directly in the blue banner.*



## Look at the global elements

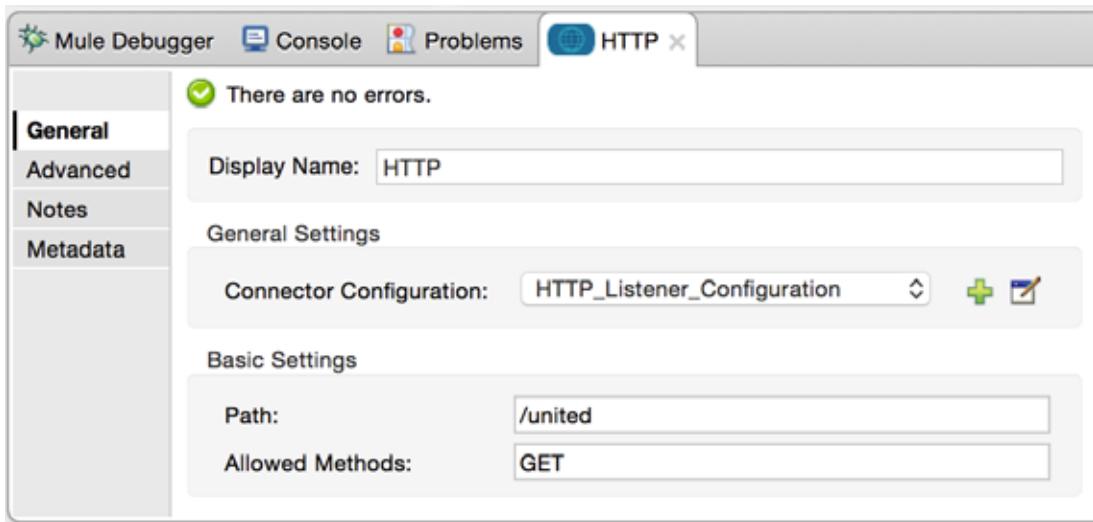
8. Click the Global Element tabs at the bottom of the canvas.
9. Select the HTTP Listener Configuration and click Edit (or double-click it).



10. In the Global Element Properties dialog box, review the HTTP\_Listener\_Configuration and click OK.

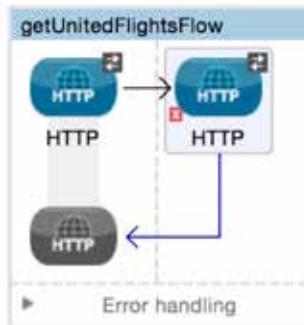
## Configure the HTTP Listener connector endpoint

11. Click the Message Flow tab.
12. Double-click the new HTTP Listener connector endpoint.
13. Set the connector configuration to the existing HTTP\_Listener\_Configuration.
14. Set the path to /united.
15. Set the allowed methods to GET.



## Add an HTTP Request connector endpoint

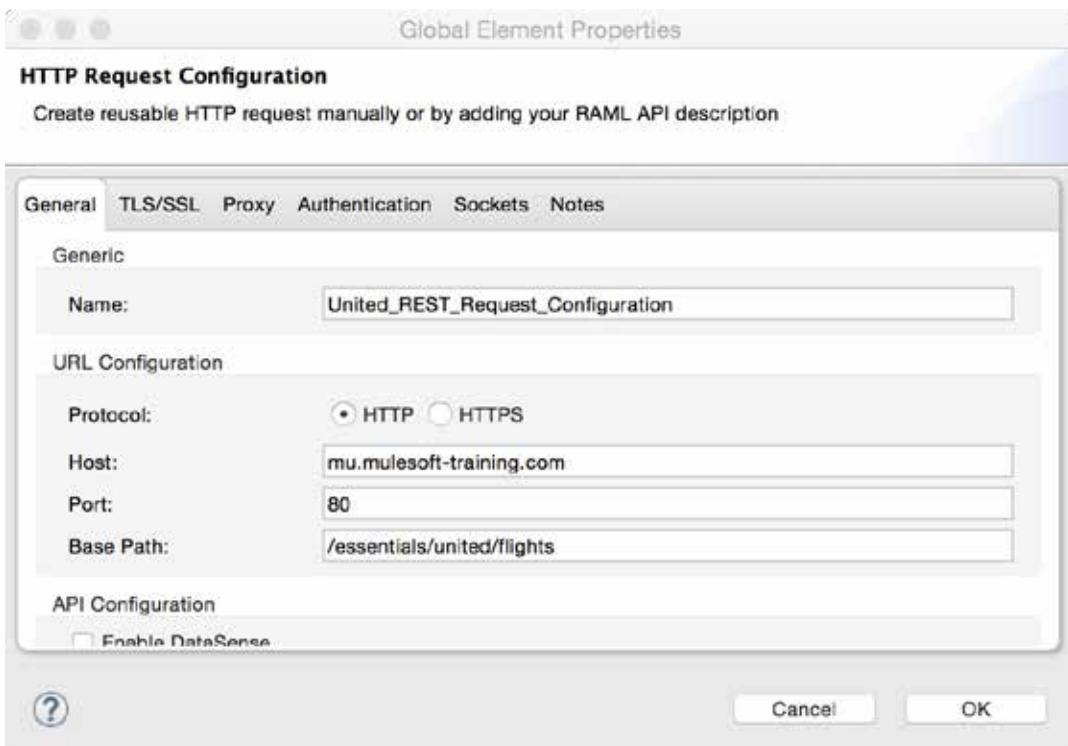
16. Drag out another HTTP connector and drop it into the process section of the flow.



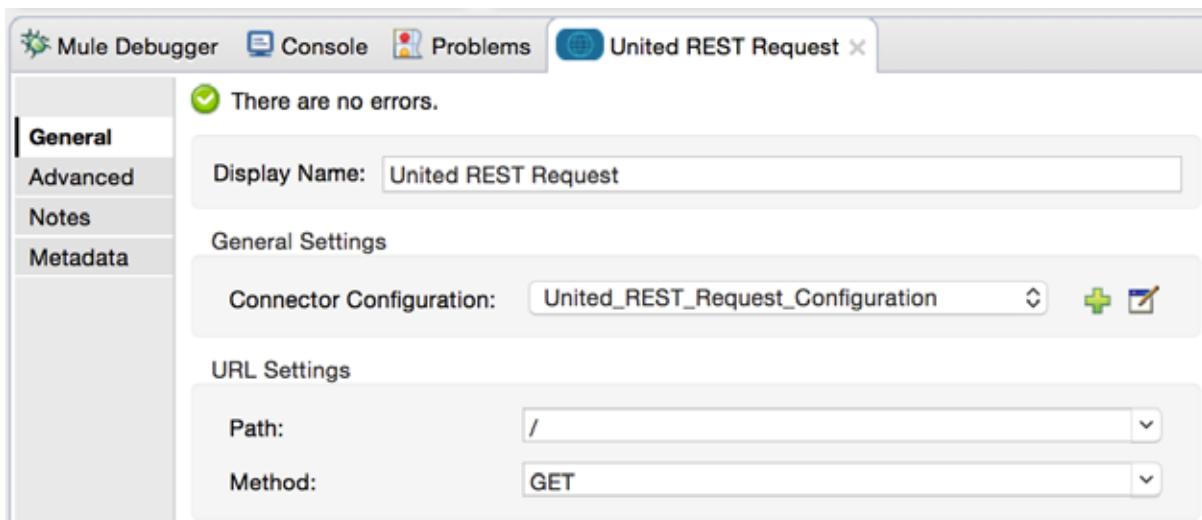
17. In the Properties view, change the display name to United REST Request.
18. Click the Add button next to connector configuration.

19. In the Global Element Properties dialog box, set the following values and click OK.

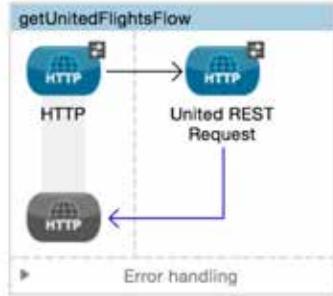
- Name: United\_REST\_Request\_Configuration
- Host: Use the value for the United web service host listed in the course snippets.txt file.
- Port: Use the value for the United web service port listed in the course snippets.txt file.
- Base Path: Base Path: Use the value for the United web service base path listed in the course snippets.txt file.



20. In the Properties view, set the path to / and the method to GET.



21. Click the Global Elements tab at the bottom of the canvas and see the new global configuration element.
22. Return to the Message Flow view.



## Test the application

23. Save the file and run the application.
24. Make a request to <http://localhost:8081/united>; you should see JSON flight data returned.

```

{
  "flights": [
    {"airlineName": "United", "price": 400, "departureDate": "2015/03/20", "planeType": "Boeing 737", "origin": "MUA", "code": "ER38sd", "emptySeats": 0, "destination": "SFO"}, {"airlineName": "United", "price": 345.99, "departureDate": "2015/02/11", "planeType": "Boeing 737", "origin": "MUA", "code": "ER45if", "emptySeats": 52, "destination": "LAX"}, {"airlineName": "United", "price": 346, "departureDate": "2015/04/11", "planeType": "Boeing 777", "origin": "MUA", "code": "ER45jd", "emptySeats": 12, "destination": "LAX"}, {"airlineName": "United", "price": 423, "departureDate": "2015/06/11", "planeType": "Boeing 707", "origin": "MUA", "code": "ER0945", "emptySeats": 0, "destination": "LAX"}, {"airlineName": "United", "price": 845, "departureDate": "2015/07/11", "planeType": "Boeing 727", "origin": "MUA", "code": "ER9fje", "emptySeats": 32, "destination": "CLE"}, {"airlineName": "United", "price": 245, "departureDate": "2015/08/11", "planeType": "Boeing 747", "origin": "MUA", "code": "ER3kfd", "emptySeats": 13, "destination": "CLE"}, {"airlineName": "United", "price": 945, "departureDate": "2015/09/11", "planeType": "Boeing 757", "origin": "MUA", "code": "ER39rk", "emptySeats": 54, "destination": "SFO"}, {"airlineName": "United", "price": 954, "departureDate": "2015/02/12", "planeType": "Boeing 777", "origin": "MUA", "code": "ER39rj", "emptySeats": 23, "destination": "SFO"}, {"airlineName": "United", "price": 234, "departureDate": "2015/02/12", "planeType": "Boeing 787", "origin": "MUA", "code": "ER95jf", "emptySeats": 23, "destination": "PDX"}, {"airlineName": "United", "price": 853, "departureDate": "2015/02/13", "planeType": "Boeing 777", "origin": "MUA", "code": "ER49fd", "emptySeats": 0, "destination": "PDX"}, {"airlineName": "United", "price": 483, "departureDate": "2015/02/20", "planeType": "Boeing 777", "origin": "MUA", "code": "ER95jf", "emptySeats": 95, "destination": "PDX"}, {"airlineName": "United", "price": 532, "departureDate": "2015/02/12", "planeType": "Boeing 777", "origin": "MUA", "code": "ER04kf", "emptySeats": 30, "destination": "PDX"} ]
}
  
```

## Walkthrough 3-2: Pass arguments to a RESTful web service

In this walkthrough, you will retrieve United flights for a specific destination by setting the destination as a URI parameter. You will:

- Modify the HTTP Request connector endpoint to use a URI parameter for the destination.
- Set the destination to a static value.
- Set the destination to a dynamic query parameter value.
- Create a variable to set the destination.

*Note: In a later module, you will add an HTML form to the application for destination selection.*

The screenshot shows the configuration of a REST endpoint in MuleSoft Anypoint Studio. The 'Path' is set to '/{destination}' and the 'Method' is 'GET'. Under the 'Parameters' section, there is a single parameter named 'destination' with a value of '\${flowVars.destination}'. Below the configuration, a preview window shows the JSON response for flights to CLE:

```
{"flights": [{"airlineName": "United", "price": 845, "departureDate": "2015/07/11", "planeType": "Boeing 727", "origin": "MUA", "code": "ER9fje", "emptySeats": 32, "destination": "CLE"}, {"airlineName": "United", "price": 245, "departureDate": "2015/08/11", "planeType": "Boeing 747", "origin": "MUA", "code": "ER3kfd", "emptySeats": 13, "destination": "CLE"}]}
```

### Make a request to the web service in a browser or another tool

1. Make a request to the United RESTful web service URL for a destination listed in the course snippets.txt file; you should see JSON data for only the flights to SFO.

The screenshot shows a web browser window with the URL 'mu.mulesoft-training.com/essentials/united/flights/SFO'. The response is a JSON object containing flight information for SFO:

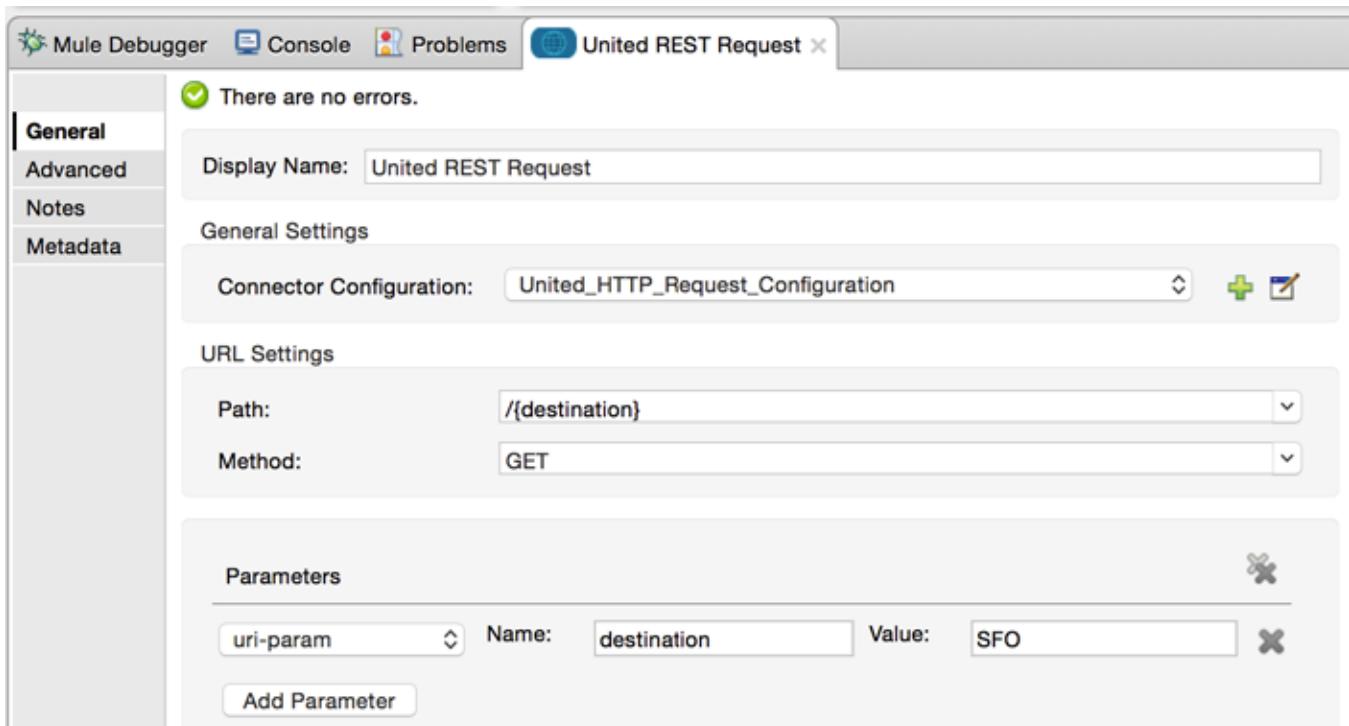
```
{"flights": [{"airlineName": "United", "price": 400, "departureDate": "2015/03/20", "planeType": "Boeing 737", "origin": "MUA", "code": "ER38sd", "emptySeats": 0, "destination": "SFO"}, {"airlineName": "United", "price": 945, "departureDate": "2015/09/11", "planeType": "Boeing 757", "origin": "MUA", "code": "ER39rk", "emptySeats": 54, "destination": "SFO"}, {"airlineName": "United", "price": 954, "departureDate": "2015/02/12", "planeType": "Boeing 777", "origin": "MUA", "code": "ER39rj", "emptySeats": 23, "destination": "SFO"}]}
```

2. Make additional requests for destinations of LAX, CLE, PDX, or PDF.

### Add a URI parameter with a static value

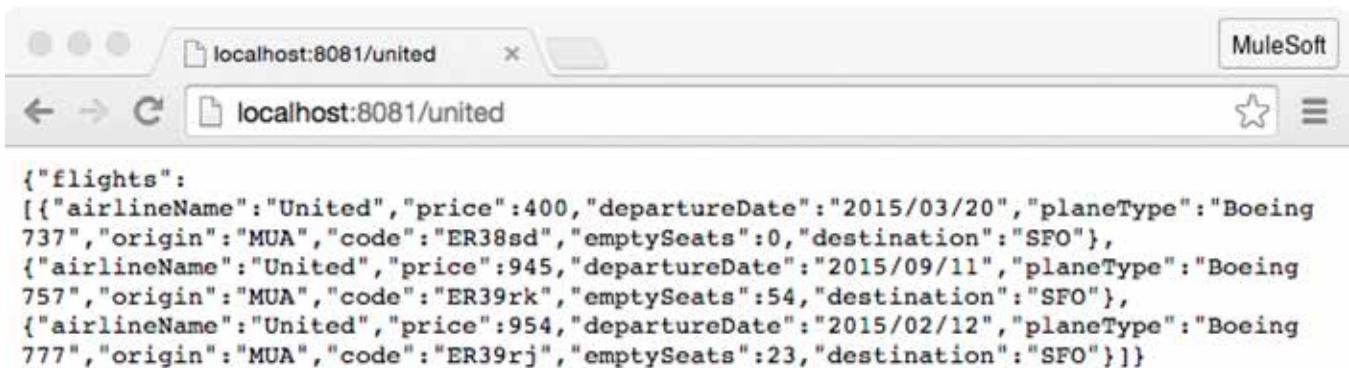
3. Return to ap essentials.xml.
4. Double-click the United REST Request endpoint.

5. In the Properties view, click the Add Parameter button.
6. Set the following parameter values.
  - Parameter type: uri-param
  - Name: destination
  - Value: SFO
7. Change the United REST Request endpoint path to `{destination}`.

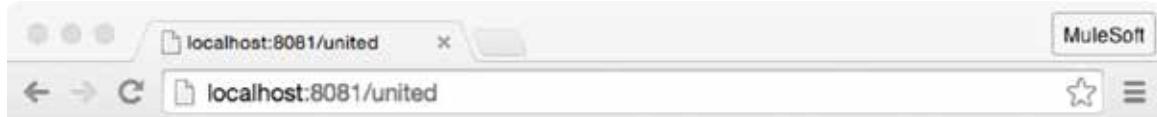


## Test the application

8. Save the application to redeploy the application and make a request to <http://localhost:8081/united/>; you should only get the SFO flights.



9. Modify the United REST Request endpoint and set the URI parameter value to LAX.
10. Save and redeploy the application and make another request to <http://localhost:8081/united/>; you should now only get the LAX flights.



```
{
  "flights": [
    {"airlineName": "United", "price": 345.99, "departureDate": "2015/02/11", "planeType": "Boeing 737", "origin": "MUA", "code": "ER45if", "emptySeats": 52, "destination": "LAX"},
    {"airlineName": "United", "price": 346, "departureDate": "2015/04/11", "planeType": "Boeing 777", "origin": "MUA", "code": "ER45jd", "emptySeats": 12, "destination": "LAX"},
    {"airlineName": "United", "price": 423, "departureDate": "2015/06/11", "planeType": "Boeing 707", "origin": "MUA", "code": "ER0945", "emptySeats": 0, "destination": "LAX"}]
}
```

## Add a URI parameter with a dynamic value

11. Return to the Properties view for the United REST Request endpoint.
12. Change the value of the uri-param from LAX to an expression for the value of a query parameter called code.

```
##[message.inboundProperties.'http.query.params'.code]
```

## Test the application

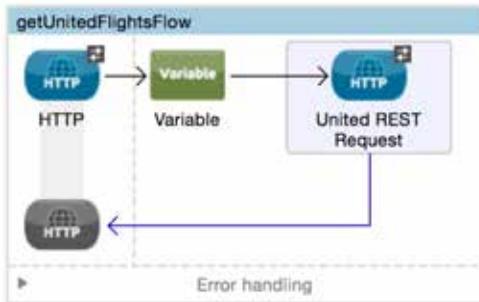
13. Save and redeploy the application and make a request to <http://localhost:8081/united/?code=CLE>; you should only get the CLE flights.



```
{
  "flights": [
    {"airlineName": "United", "price": 845, "departureDate": "2015/07/11", "planeType": "Boeing 727", "origin": "MUA", "code": "ER9fje", "emptySeats": 32, "destination": "CLE"},
    {"airlineName": "United", "price": 245, "departureDate": "2015/08/11", "planeType": "Boeing 747", "origin": "MUA", "code": "ER3kfd", "emptySeats": 13, "destination": "CLE"}]
}
```

## Create a variable to set the destination

14. Add a Variable transformer before the United REST Request endpoint.



15. In the Variable Properties view, change the display name to Set Destination.

16. Set the operation to Set Variable and the name to destination.

17. Use a ternary expression to set the value to 'SFO' or the value of a query parameter called code.

```
#[(message.inboundProperties.'http.query.params'.code == empty) ?  
'SFO' : message.inboundProperties.'http.query.params'.code]
```

Operation:	<input checked="" type="radio"/> Set Variable <input type="radio"/> Remove Variable
Name:	destination
Value:	#[(message.inboundProperties.'http.query.params'.code == empty) ? 'SFO' : message.inboundProperties.'http.query.params'.code]

18. Navigate to the Properties view for the United REST connector endpoint.

19. Modify the URI parameter to use the new destination variable.

Parameters	
uri-param	Name: destination Value: #[flowVars.destination]

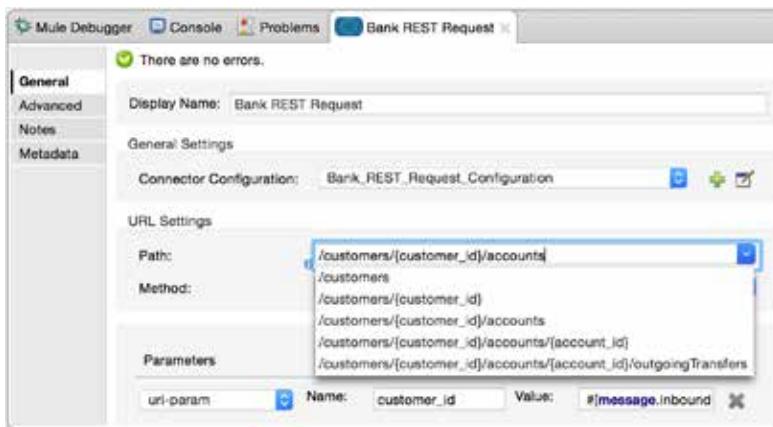
20. Save and redeploy the application and make a request to <http://localhost:8081/united>; you should see only flights to SFO again.

21. Make another request to <http://localhost:8081/united?code=PDX>; you should now see flights to PDX.

## Walkthrough 3-3: Consume a RESTful web service that has a RAML definition

In this walkthrough, you will consume a RESTful web service containing some simple bank account data that has a RAML definition file. You will:

- Add a third flow to the application.
- Add an HTTP Listener connector endpoint to receive requests at <http://localhost:8081/bank>.
- Add an HTTP Request connector endpoint to consume a RESTful web service defined with a RAML file.



### Add a new flow with an HTTP Listener connector endpoint

1. Return to apessentials.xml.
2. Drag out another HTTP connector and drop it in the canvas between the two existing flows.
3. Rename the flow to getBankAccountsFlow.



4. In the Properties view for the endpoint, set the connector configuration to the existing `HTTP_Listener_Configuration`.

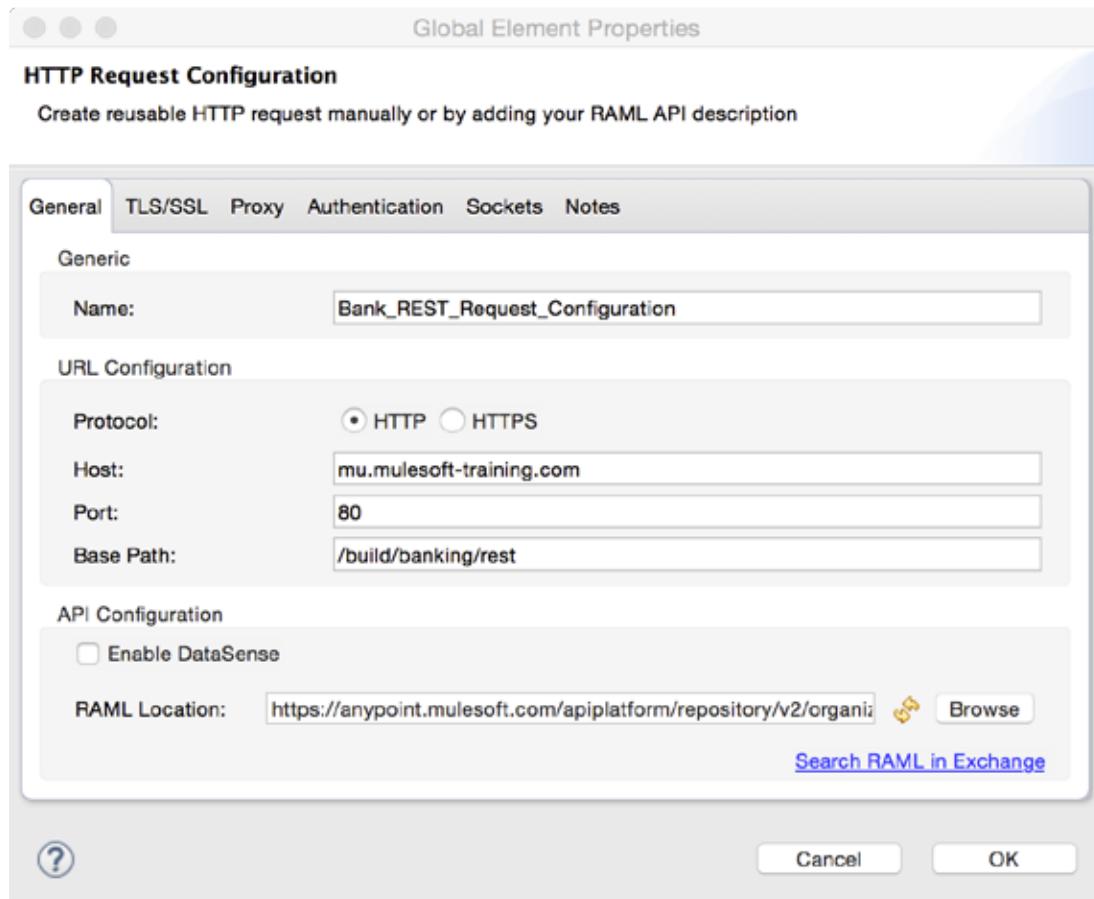


- Set the path to /bank.
- Set the allowed methods to GET.

## Add an HTTP Request connector with a RAML location

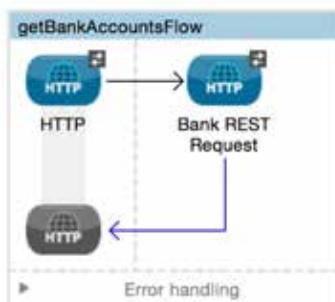
- Drag out another HTTP connector and drop it in the process section of the new flow.
- In the Properties view, change the name to Bank REST Request.
- Click the Add button next to connector configuration.
- In the Global Element Properties dialog box, change the name to Bank\_REST\_Request\_Configuration.
- Set the RAML location to the Banking RAML URL listed in the course snippets.txt file.
- Wait for the RAML to be parsed and the host, port, and base path fields to be populated and then click OK.

*Note: If the fields did not populate, click the Reload RAML button next to the RAML location.*



- Click the Global Elements tab at the bottom of the canvas and see the new global configuration element.

14. Return to the Message Flow view.



15. In the Bank REST Request Properties view, click the drop-down button for the path; you should see the available resources (paths) for the RESTful web service defined by the RAML file.

Attribute '...is required

General

Advanced

Notes

Metadata

Display Name: Bank REST Request

General Settings

Connector Configuration: Bank\_REST\_Request\_Configuration

URL Settings

Path:

Method:

Parameters

/customers  
/customers/{customer\_id}  
/customers/{customer\_id}/accounts  
/customers/{customer\_id}/accounts/{account\_id}  
/customers/{customer\_id}/accounts/{account\_id}/outgoingTransfers

16. Select the /customers path.

17. Set the method to GET.

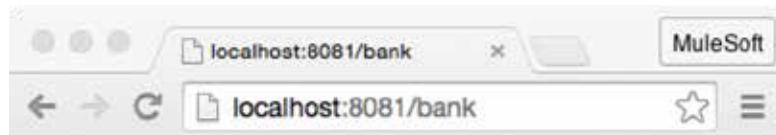
URL Settings

Path: /customers

Method: GET

## Test the application

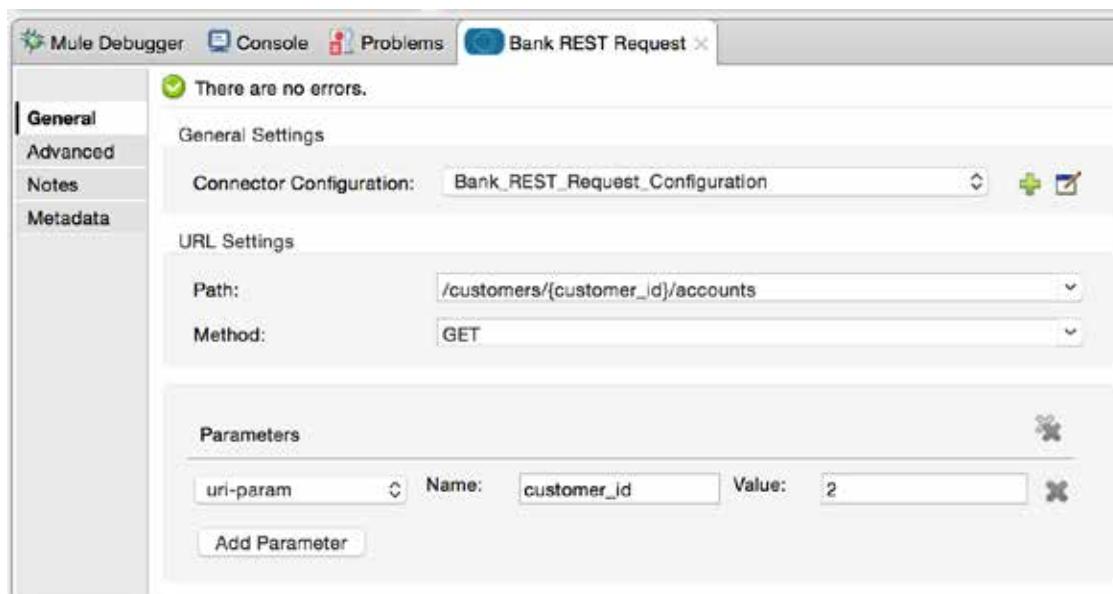
18. Save to redeploy the application and make a request to <http://localhost:8081/bank>; you should see the customers returned as JSON.



```
[ { "customer_id" : 1, "last_name" : "Lloyd", "first_name" : "Sam" }, { "customer_id" : 2, "last_name" : "Newberry", "first_name" : "Frank" }, { "customer_id" : 3, "last_name" : "Webber", "first_name" : "Mule" } ]
```

## Modify the request to use a path requiring a URI parameter

19. Return to the Properties view for the Bank REST Request endpoint.
20. Change the path to `/customers/{customer_id}/accounts` and the method to GET; a URI parameter with the name `customer_id` should have been created automatically.
21. Set the `customer_id` parameter to a value of 2.



## Test the application

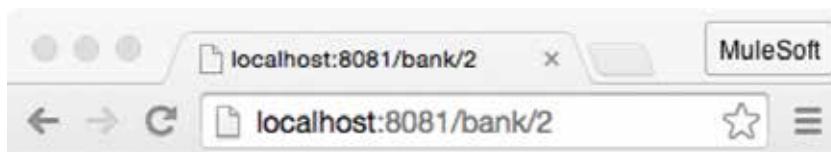
22. Save to redeploy the application make a request to <http://localhost:8081/bank>; you should see the account info for the customer with an ID of 2 returned as JSON.



```
[ {  
    "balance" : 40000.00,  
    "customer_id" : 2,  
    "account_id" : 587  
, {  
    "balance" : 5000.29,  
    "customer_id" : 2,  
    "account_id" : 611  
} ]
```

23. Make a request and try to pass the customer ID as a URI parameter:

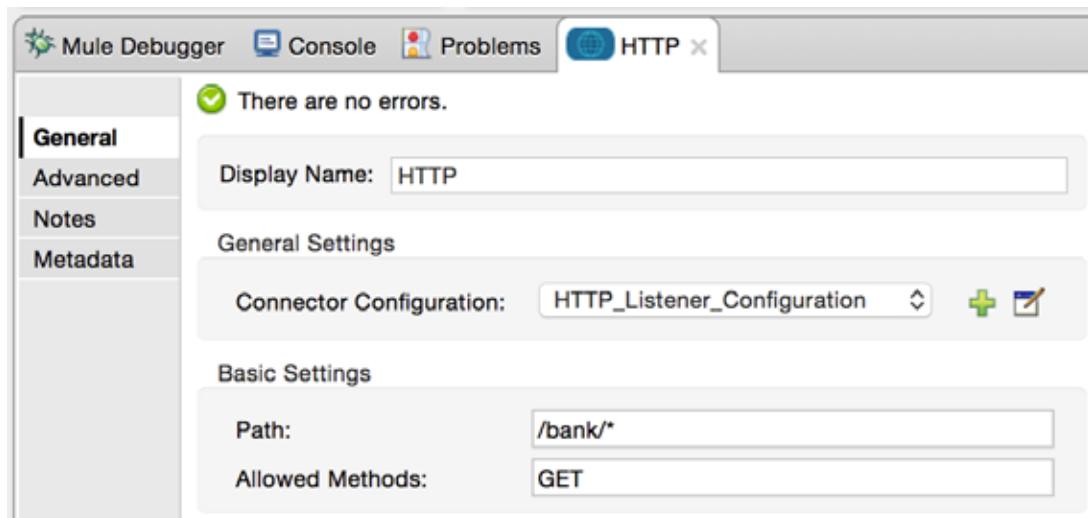
<http://localhost:8081/bank/2>; you should get a Resource not found response.



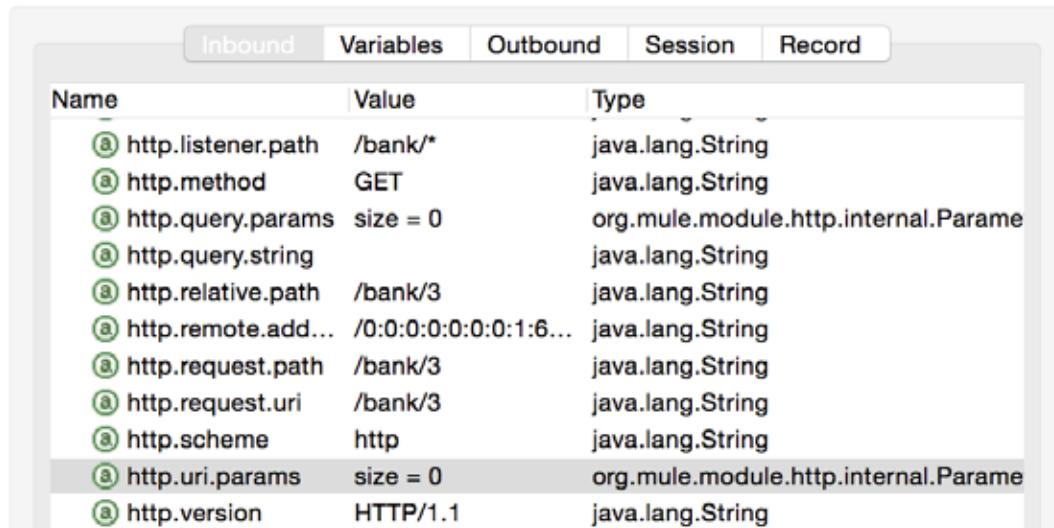
Resource not found.

## Get the value of an HTTP Listener endpoint URI parameter

24. Return to the Properties view for the HTTP Listener endpoint.  
25. Change the path to use a wildcard to specify any path starting with /bank/.

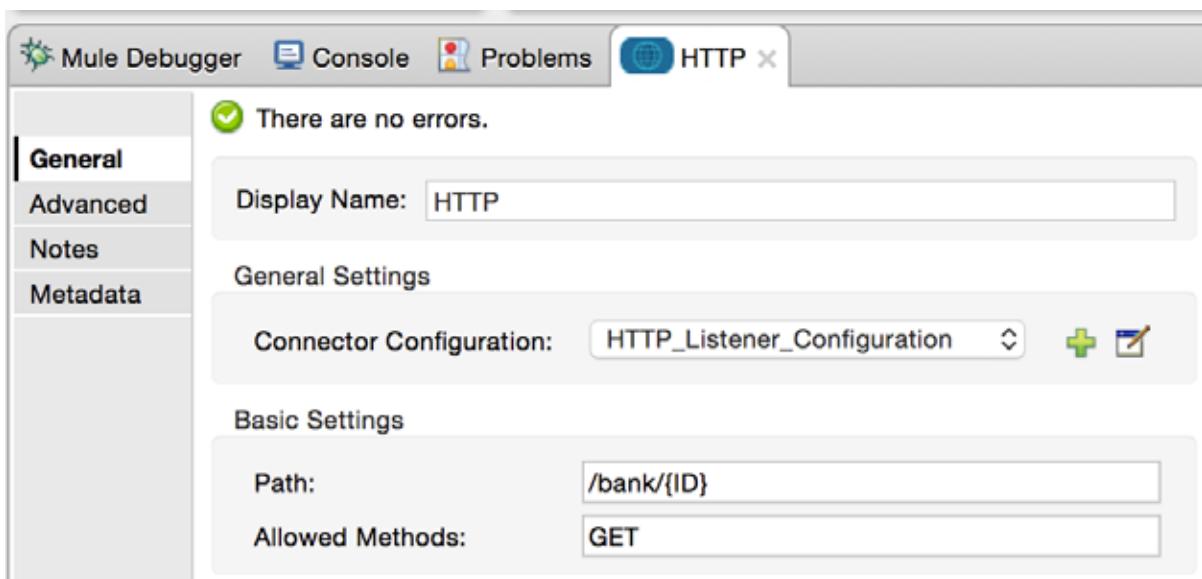


26. Add a breakpoint to the Bank REST Request endpoint.
27. Save the file and debug the application.
28. Make a request to <http://localhost:8081/bank/3>.
29. In the Mule Debugger view, locate the http.request.uri and http.uri.params inbound properties.



Name	Value	Type
⑥ http.listener.path	/bank/*	java.lang.String
⑥ http.method	GET	java.lang.String
⑥ http.query.params	size = 0	org.mule.module.http.internal.Parameters
⑥ http.query.string		java.lang.String
⑥ http.relative.path	/bank/3	java.lang.String
⑥ http.remote.add...	/0:0:0:0:0:0:1:6...	java.lang.String
⑥ http.request.path	/bank/3	java.lang.String
⑥ http.request.uri	/bank/3	java.lang.String
⑥ http.scheme	http	java.lang.String
⑥ http.uri.params	size = 0	org.mule.module.http.internal.Parameters
⑥ http.version	HTTP/1.1	java.lang.String

30. Step through the rest of the application; you should still get the account info for the customer with an ID of 2.
31. Return to the Properties view for the HTTP Listener endpoint.
32. Change the path to specify a URI parameter called ID: /bank/{ID}.



33. Save the file to redeploy the application in debug mode.
34. Make a request to <http://localhost:8081/bank/4>.

35. In the Mule Debugger view, locate the http.request.uri and http.uri.params inbound properties.

Inbound			Variables	Outbound	Session	Record
Name	Value	Type				
③ http.relative.path	/bank/4	java.lang.String				
③ http.remote.add...	/0:0:0:0:0:0:1:6...	java.lang.String				
③ http.request.path	/bank/4	java.lang.String				
③ http.request.uri	/bank/4	java.lang.String				
③ http.scheme	http	java.lang.String				
▼ ④ http.uri.params	size = 1	org.mule.module.http.internal.ParameterMap				
▼ ⑤ 0	ID=4	java.util.AbstractMap\$SimpleEntry				
⑥ key	ID	java.lang.String				
⑥ value	4	java.lang.String				
③ http.version	HTTP/1.1	java.lang.String				

36. Step through the rest of the application; you should still get the account info for the customer with an ID of 2

## Set the HTTP Request endpoint URI parameter to a dynamic value

37. Return to the Properties view for the Bank REST Request endpoint.

38. Change the value of the customer\_id uri-param from 2 to an expression for the value of an HTTP Listener endpoint URI parameter called ID.

```
##[message.inboundProperties.'http.uri.params'.ID]
```

## Test the application

39. Save the file and run the application.

40. Make another request to <http://localhost:8081/bank/3>; you should now see the account info for the customer with an ID of 3.



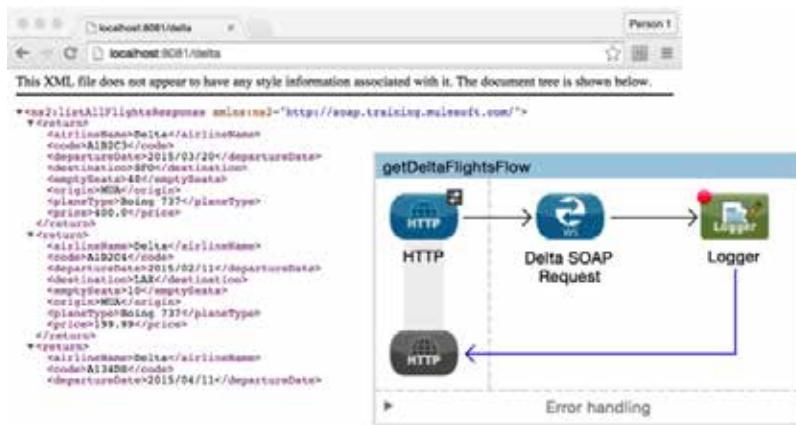
```
[ { "balance" : 50000.00, "customer_id" : 3, "account_id" : 588 }, { "balance" : 9999.99, "customer_id" : 3, "account_id" : 600 } ]
```



## Walkthrough 3-4: Consume a SOAP web service

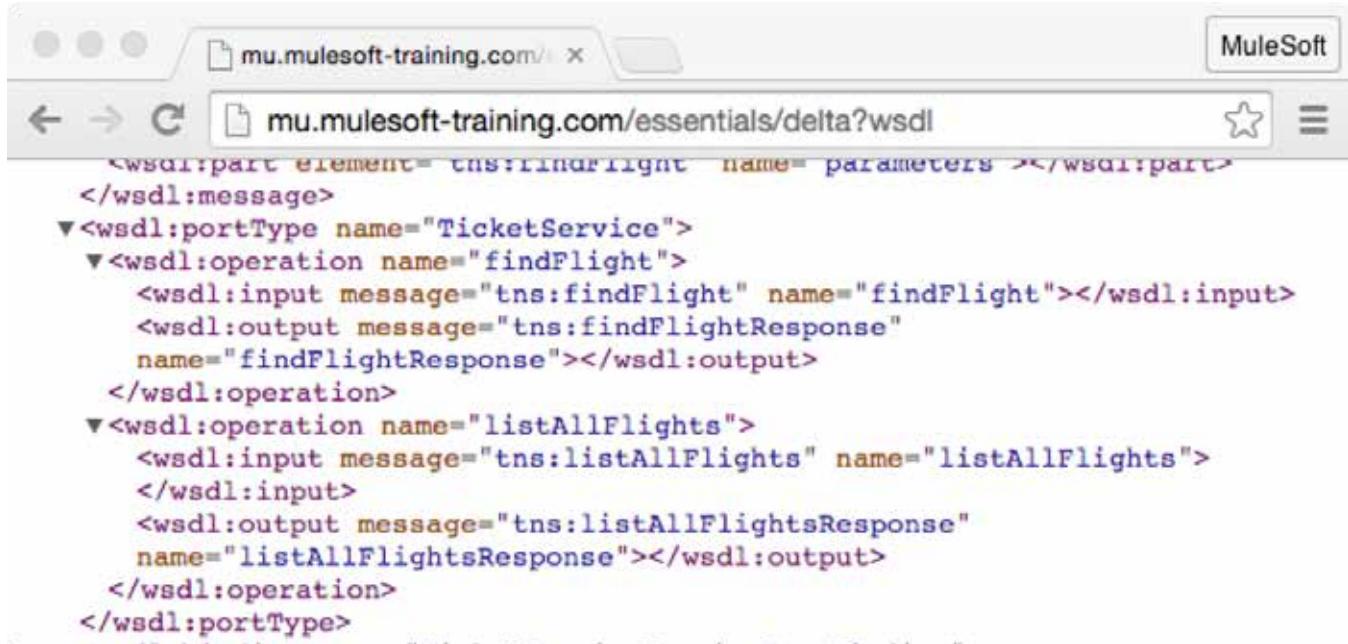
In this walkthrough, you will consume a SOAP web service that returns a list of flights for Delta airlines. You will:

- Create a fourth flow with an endpoint to receive requests at <http://localhost:8081/delta>.
- Add a Web Service Consumer connector to consume a SOAP web service for Delta flight data.
- Use the DOM to XML transformer to display the SOAP response.



### Browse the WSDL

1. Make a request to the Delta SOAP web service WSDL listed in the course snippets.txt file; you should see the web service WSDL returned.
2. Browse the WSDL; you should find references to operations listAllFlights and findFlight.

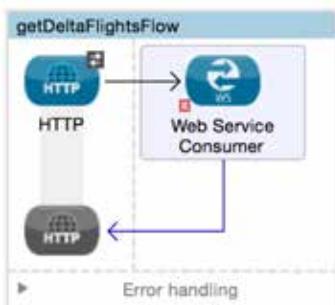


## Create a new flow with an HTTP Listener connector endpoint

3. Return to apessentials.xml.
4. Drag out another HTTP connector and drop it in the canvas above the existing flows.
5. Give the flow a new name of getDeltaFlightsFlow.
6. In the Properties view for the endpoint, set the connector configuration to the existing HTTP\_Listener\_Configuration.
7. Set the path to /delta.
8. Set the allowed methods to GET.

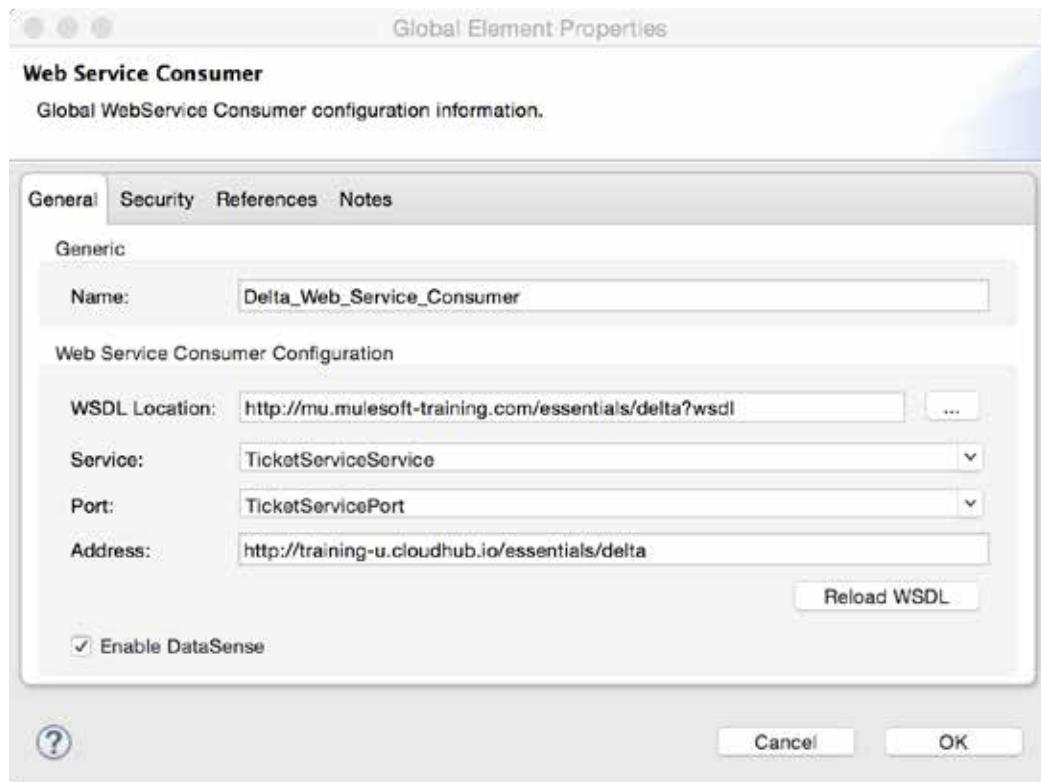
## Add a Web Service Consumer connector endpoint

9. Drag out a Web Service Consumer connector and drop it in the process section of the flow.

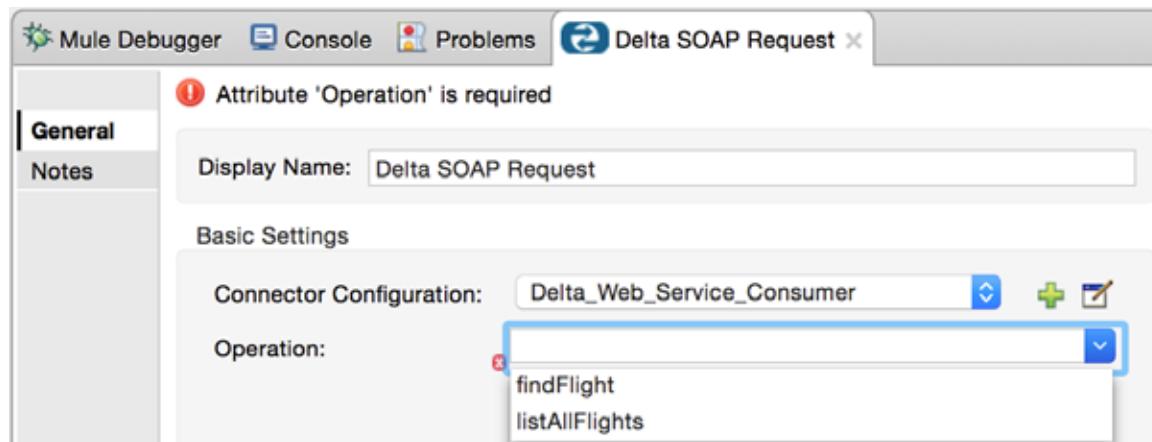


10. In the Properties view, set the display name to Delta SOAP Request.
11. Click the Add button next to connector configuration.
12. In the Global Element Properties dialog box, change the name to Delta\_Web\_Service\_Consumer.
13. Set the WSDL Location to the Delta SOAP web service WSDL listed in the course snippets.txt file.

14. Wait for the WSDL to be parsed and the service, port, and address fields to be automatically populated.



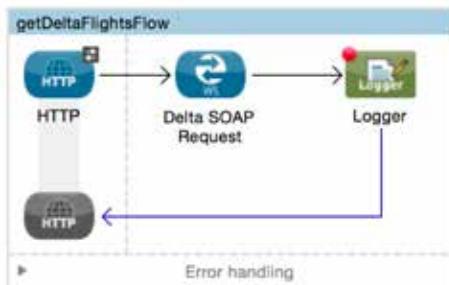
15. Click OK.  
16. Switch to the Global Elements view and see the new global configuration element.  
17. Return to the Message Flow view.  
18. In the Delta SOAP Request Properties view, click the operation drop-down button; you should see all of the web service operations listed.



19. Select the listAllFlights operation.

## Debug the application

20. Add a Logger component after the Delta SOAP Request endpoint.
21. In the Logger Properties view, set the message to #[payload].
22. Add a breakpoint to the Logger.



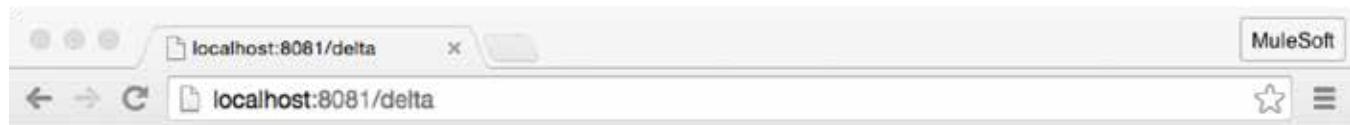
23. Save the file and debug the application.
24. Make a request to <http://localhost:8081/delta>.
25. In the Mule Debugger view, drill-down into the payload variable.

Name	Value	Type
Message	org.mule.DefaultMuleMessage	org.mule.api.processor.LoggerMessageProcessor
Message Processor	Logger	org.apache.cxf.staxutils.DepthXMLStreamReader
payload	org.mule.module.ws.consumer.NamespaceRestorerXMLStreamReader@2db53627	java.lang.Integer
depth	0	org.mule.module.ws.consumer.NamespaceRestorerXMLStreamReader
reader	org.mule.module.ws.consumer.NamespaceRestorerXMLStreamReader@2db53627	javanet.staxutils.events.EventAllocator
first	false	javanet.staxutils.events.EventAllocator
namespaces	[xmlns:ns2="http://soap.training.mulesoft.com/"]	java.util.ArrayList
nsBlacklist	[http://schemas.xmlsoap.org/soap/envelope/]	java.util.ArrayList
reader	org.mule.module.cxf.support.StreamCloseable@2db53627	java.util.List
scope	<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">	java.util.List
0	<soap:Body>	javanet.staxutils.events.StartElementEvent
1	<ns2:listAllFlightsResponse xmlns:ns2="http://soap.training.mulesoft.com/"/>	javanet.staxutils.events.StartElementEvent
2	<ns2:flights>	javanet.staxutils.events.StartElementEvent
attributes	null	java.util.Map
location	[row,col {unknown-source}]: [1,82]	com.ctc.wstx.io.WstxInputLocation
name	(http://soap.training.mulesoft.com/)/listAllFlightsResponse	javax.xml.namespace.QName

26. Step to the end of the application.
27. Look at the console; you should see the type of object listed.

```
INFO 2015-08-19 12:38:25,676 [[apessentials].HTTP_Listener_Configuration.worker.01] org.mule.api.processor.LoggerMessageProcessor: org.mule.module.ws.consumer.NamespaceRestorerXMLStreamReader@2db53627
```

28. Return to the browser; you should see the XML SOAP response displayed there.

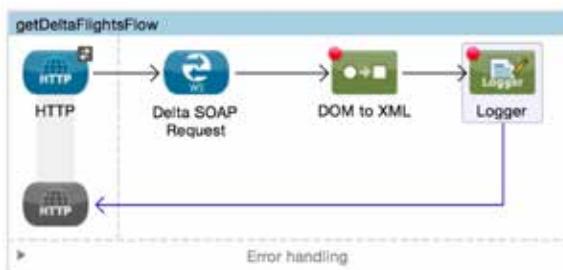


This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
<ns2:listAllFlightsResponse xmlns:ns2="http://soap.training.mulesoft.com/">
  <return>
    <airlineName>Delta</airlineName>
    <code>A1B2C3</code>
    <departureDate>2015/03/20</departureDate>
    <destination>SFO</destination>
    <emptySeats>40</emptySeats>
    <origin>MUA</origin>
    <planeType>Boing 737</planeType>
    <price>400.0</price>
  </return>
  <return>
    <airlineName>Delta</airlineName>
    <code>A1B2C4</code>
    <departureDate>2015/02/11</departureDate>
    <destination>LAX</destination>
    <emptySeats>10</emptySeats>
    <origin>MUA</origin>
    <planeType>Boing 737</planeType>
    <price>199.99</price>
  </return>
  <return>
    <airlineName>Delta</airlineName>
    <code>A134DS</code>
    <departureDate>2015/04/11</departureDate>
```

## Display the payload as a String

29. Add a DOM to XML transformer before the Logger.

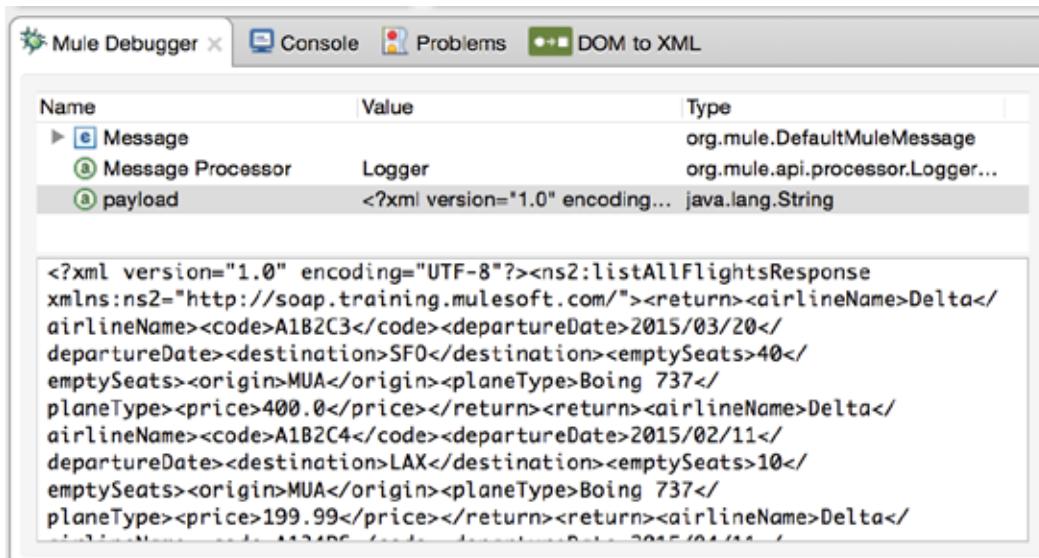


## Test the application

30. Save the file to redeploy the application.

31. Make another request to <http://localhost:8081/delta>.

32. Step to the Logger; you should see the payload is now a String.



The screenshot shows the Mule Debugger interface with the following details:

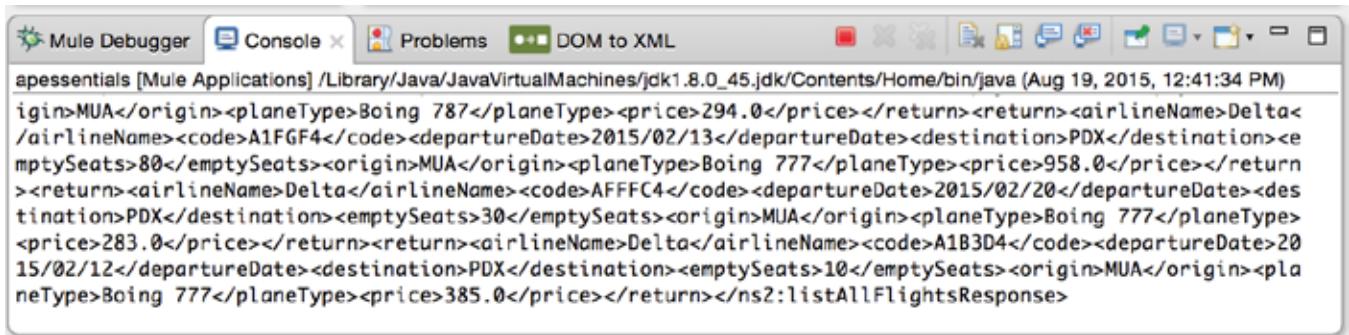
Name	Value	Type
Message		org.mule.DefaultMuleMessage
Message Processor	Logger	org.mule.api.processor.Logger...
payload	<?xml version="1.0" encoding...>	java.lang.String

The payload value is displayed as a large XML string:

```
<?xml version="1.0" encoding="UTF-8"?><ns2:listAllFlightsResponse
xmlns:ns2="http://soap.training.mulesoft.com/"><return><airlineName>Delta</
airlineName><code>A1B2C3</code><departureDate>2015/03/20</
departureDate><destination>SFO</destination><emptySeats>40</
emptySeats><origin>MUA</origin><planeType>Boing 737</
planeType><price>400.0</price></return><return><airlineName>Delta</
airlineName><code>A1B2C4</code><departureDate>2015/02/11</
departureDate><destination>LAX</destination><emptySeats>10</
emptySeats><origin>MUA</origin><planeType>Boing 737</
planeType><price>199.99</price></return><return><airlineName>Delta</
airlineName><code>A1F5D4</code><departureDate>2015/01/11</
departureDate><destination>PDX</destination><emptySeats>80</
emptySeats><origin>MUA</origin><planeType>Boing 777</planeType><price>958.0</price></return>
<return><airlineName>Delta</airlineName><code>AFFFC4</code><departureDate>2015/02/20</departureDate><des
tination>PDX</destination><emptySeats>30</emptySeats><origin>MUA</origin><planeType>Boing 777</planeType>
<price>283.0</price></return><return><airlineName>Delta</airlineName><code>A1B3D4</code><departureDate>20
15/02/12</departureDate><destination>PDX</destination><emptySeats>10</emptySeats><origin>MUA</origin><pla
neType>Boing 777</planeType><price>385.0</price></return></ns2:listAllFlightsResponse>
```

33. Resume the application.

34. Look at the console; you should see the XML SOAP response displayed there.



The screenshot shows the Mule Debugger interface with the following details:

Name	Value	Type
Message		org.mule.DefaultMuleMessage
Message Processor	Logger	org.mule.api.processor.Logger...
payload	<?xml version="1.0" encoding="UTF-8"?><ns2:listAllFlightsResponse xmlns:ns2="http://soap.training.mulesoft.com/"><return><airlineName>Delta</ airlineName><code>A1B2C3</code><departureDate>2015/03/20</ departureDate><destination>SFO</destination><emptySeats>40</ emptySeats><origin>MUA</origin><planeType>Boing 737</ planeType><price>400.0</price></return><return><airlineName>Delta</ airlineName><code>A1B2C4</code><departureDate>2015/02/11</ departureDate><destination>LAX</destination><emptySeats>10</ emptySeats><origin>MUA</origin><planeType>Boing 737</ planeType><price>199.99</price></return><return><airlineName>Delta</ airlineName><code>A1F5D4</code><departureDate>2015/01/11</ departureDate><destination>PDX</destination><emptySeats>80</ emptySeats><origin>MUA</origin><planeType>Boing 777</planeType><price>958.0</price></return> <return><airlineName>Delta</airlineName><code>AFFFC4</code><departureDate>2015/02/20</departureDate><des tination>PDX</destination><emptySeats>30</emptySeats><origin>MUA</origin><planeType>Boing 777</planeType> <price>283.0</price></return><return><airlineName>Delta</airlineName><code>A1B3D4</code><departureDate>20 15/02/12</departureDate><destination>PDX</destination><emptySeats>10</emptySeats><origin>MUA</origin><pla neType>Boing 777</planeType><price>385.0</price></return></ns2:listAllFlightsResponse>	

35. Stop the runtime.



# Module 4: Connecting to Additional Resources



In this module, you will learn:

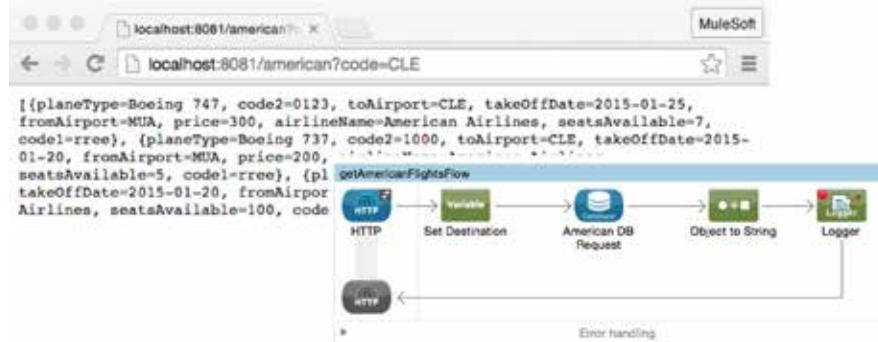
- To connect to databases.
- To connect to files.
- To connect to JMS queues.
- To connect to SaaS applications.
- To discover and install connectors not bundled with Anypoint Studio.
- About developing your own custom connectors with Anypoint Connector DevKit.

## Walkthrough 4-1: Connect to a database (MySQL)

In this walkthrough, you will connect to a MySQL database that contains information about American flights. You will:

- Create a new flow to receive requests at <http://localhost:8081/american>.
- Add a Database connector endpoint that connects to a MySQL database.
- Write a query to return all flights from the database for the static destination SFO.
- Modify the query to use a dynamic destination from a query parameter.
- Use the Object to String transformer to return the results as a string.

*Note: You will get a destination dynamically from a form in a later module.*

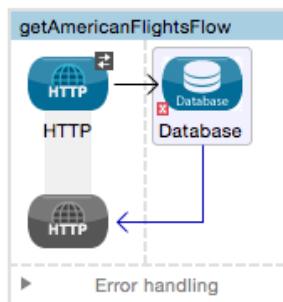


### Create a new flow with an HTTP Listener connector endpoint

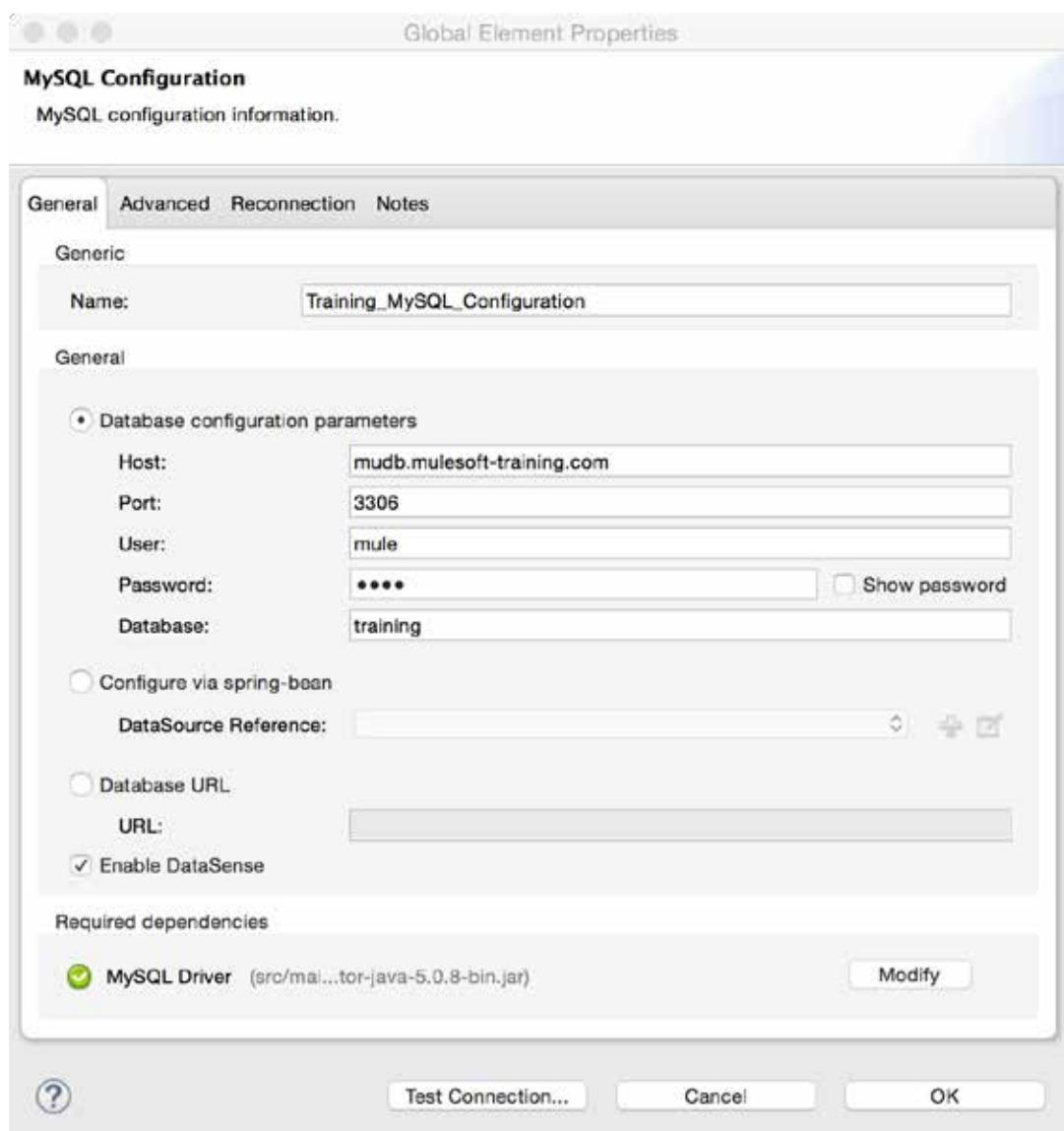
1. Return to ap essentials.xml.
2. Create a new flow above the Delta flow and name it getAmericanFlightsFlow.
3. In the HTTP Properties view, set the connector configuration to the existing HTTP\_Listener\_Configuration.
4. Set the path to /american and the allowed methods to GET.

### Add and configure a Database connector endpoint

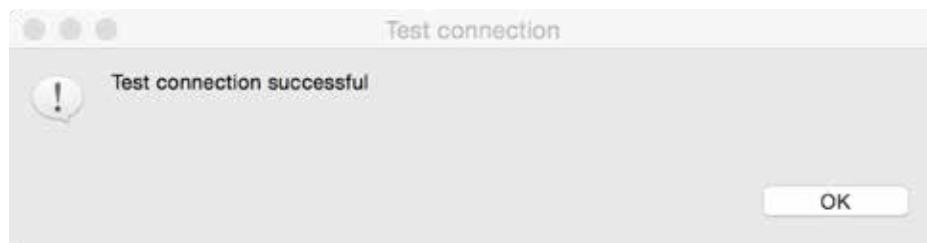
5. Add a Database connector to the process section of the flow.



6. In the Database Properties view, change the display name to American DB Request.
7. Click the Add button next to connector configuration.
8. In the Choose Global Type dialog box, select Connector Configuration > MySQL Configuration and click OK.
9. In the Global Element Properties dialog box, set the name to Training\_MySQL\_Configuration.
10. Set the server, port, user, password, and database values to the values listed in the course snippets.txt file for the MySQL database.
11. Click the Add File button next to the MySQL driver required dependency.
12. Browse to the mysql-connector-java-{version}-bin.jar file located in the APEssentials3.7\_studentFiles\_{date}/jars folder and click Open.



13. Back in the Global Element Properties dialog box, click the Test Connection button; you should get a successful test dialog box.



*Note: If your database connectivity test fails, make sure you are not behind a firewall restricting access to port 3306. If you cannot access port 3306, use the setup instructions to install and set up VirtualBox to use the MuleSoft Training server image.*

14. Click OK to close the dialog box.  
15. Click OK to close the Global Element Properties dialog box.

## Write a query to return all flights

16. In the Properties view for the database endpoint, select the Select operation.  
17. In the parameterized query text box, write a query to select all records from the flights database.

```
SELECT *  
FROM flights
```

## Test the application

18. Save the file and run the application.  
19. Make a request to <http://localhost:8081/american>; you should get some garbled plain text displayed or contained in a download file, which is the tool's best representation of an ArrayList.

A screenshot of a browser window. The address bar shows 'File Path : ~/Downloads/american'. The page content is a large block of garbled text, likely the serialized ArrayList representation of flight data.

```
["isr java.util.LinkedList$]J`" xpw sr$org.mule.util.CaseInsensitiveHashMapù  
Boeing 787t code2t 0001t toAirportt LAXt takeOffDatesr  
java.sql.Date 'Fh?5fó xr java.util.DatehjÅ KYt xpw K Z~xt fromAirportt MUAt price  
Boeing 747q~t 0123q~t CLEq~  
sq~w K fxq~t MUAq~sq~,q~t American Airlinesq~t 7q~t rreexsq~w?@ q~t  
Boeing 777q~t 0192q~t LAXq~  
sq~w K Z~xq~t MUAq~sq~,q~t American Airlinesq~t noneq~t rreexsq~w?@ q~t  
Airbus 800q~t 100q~t KULq~  
sq~w K Z~xq~t MUAq~sq~ Üq~tMas Airlinesq~t 20q~t masxsq~w?@ q~t  
Boeing 737q~t 1000q~t CLEq~  
sq~w K 7~xq~t MIAq~sq~ »q~t American Airlinesq~t 50q~t rreexsq~w?@ q~t
```

## Debug the application

20. Add a Logger component after the Database connector endpoint and add a breakpoint to it.
21. Save the file, debug the application, and make a request to <http://localhost:8081/american>.
22. In the Mule Debugger view, drill-down and look at the data contained in the payload.
23. Find the name of the field for the flight destination.

The screenshot shows the Mule Debugger interface with the 'Logger' tab selected. The payload is displayed as a list of flight records. Each record is a map with fields like planeType, code2, toAirport, takeOffDate, fromAirport, price, airlineName, seatsAvailable, and code1. A tooltip for one of the records shows the full JSON representation of the payload.

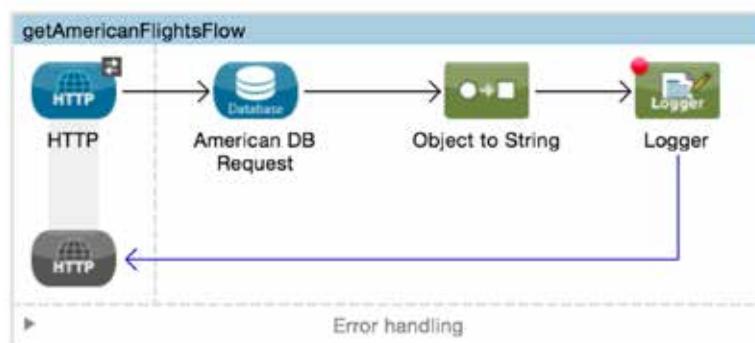
Name	Value	Type
Message		org.mule.DefaultMuleMessage
Message Processor	Logger	org.mule.api.processor.LoggerMessageProcessor
payload	size = 12	java.util.LinkedList
e 0	{planeType=Boeing 787, code2=00...}	org.mule.util.CaseInsensitiveHashMap
e 1	planeType=Boeing 787	org.apache.commons.collections.map.AbstractMap
e 2	code2=0001	org.apache.commons.collections.map.AbstractMap
e 3	toAirport=LAX	org.apache.commons.collections.map.AbstractMap
e 4	takeOffDate=2015-01-20	org.apache.commons.collections.map.AbstractMap
e 5	fromAirport=MUA	org.apache.commons.collections.map.AbstractMap
e 6	price=541	org.apache.commons.collections.map.AbstractMap
e 7	airlineName=American Airlines	org.apache.commons.collections.map.AbstractMap
e 8	seatsAvailable=none	org.apache.commons.collections.map.AbstractMap
e 9	code1=rree	org.apache.commons.collections.map.AbstractMap
e 10	{planeType=Boeing 747, code2=01...	org.mule.util.CaseInsensitiveHashMap
e 11	{planeType=Boeing 777, code2=45...	org.mule.util.CaseInsensitiveHashMap
e 12	{planeType=Boeing 737, code2=45...	org.mule.util.CaseInsensitiveHashMap

```
{planeType=Boeing 787, code2=0001, toAirport=LAX, takeOffDate=2015-01-20, fromAirport=MUA, price=541, airlineName=American Airlines, seatsAvailable=none, code1=rree}
```

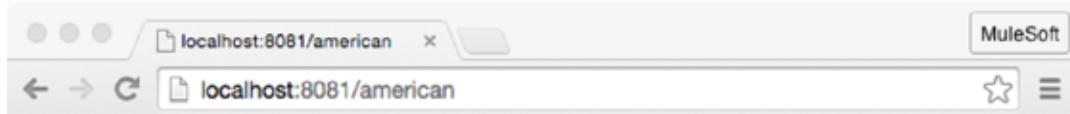
24. Stop the Mule Debugger.

## Display the return data

25. Set the Logger message to #[payload].
26. Add an Object to String transformer before the Logger component.



27. Save and run the application.
28. Make another request to <http://localhost:8081/american>; you should see the query results displayed.



29. Look at the console; you should also see the query results there.

The screenshot shows the Mule Studio interface with the "Console" tab selected. The log output shows the same JSON array of flight records as the browser, indicating the query results are being processed and outputted.

```
INFO 2015-08-19 13:13:19,297 [[apessentials].HTTP_Listener_Configuration.worker.01] org.mule.api.processor.org.LoggerMessageProcessor: [{"planeType": "Boeing 787", "code2": "0001", "toAirport": "LAX", "takeOffDate": "2015-01-20", "fromAirport": "MUA", "price": 541, "airlineName": "American Airlines", "seatsAvailable": "none", "code1": "rree"}, {"planeType": "Boeing 747", "code2": "0123", "toAirport": "CLE", "takeOffDate": "2015-01-25", "fromAirport": "MUA", "price": 300, "airlineName": "American Airlines", "seatsAvailable": 7, "code1": "rree"}, {"planeType": "Boeing 777", "code2": "0192", "toAirport": "LAX", "takeOffDate": "2015-01-20", "fromAirport": "MUA", "price": 300, "airlineName": "American Airlines", "seatsAvailable": "none", "code1": "rree"}, {"planeType": "Airbus 800", "code2": "100", "toAirport": "KUL", "takeOffDate": "2015-01-20", "fromAirport": "MUA", "price": 500, "airlineName": "Mas Airlines", "seatsAvailable": 20, "code1": "mas"}, {"planeType": "Boeing 737", "code2": "1000", "toAirport": "CLE", "takeOffDate": "2015-01-20", "fromAirport": "MUA", "price": 200, "airlineName": "American Airlines", "seatsAvailable": 5, "code1": "rree"}, {"planeType": "Boeing 737", "code2": "1093", "toAirport": "SFO", "takeOffDate": "2015-02-11", "fromAirport": "MUA", "price": 142, "airlineName": "American Airlines", "seatsAvailable": 1, "code1": "rree"}, {"planeType": "Boeing 787", "code2": "1112", "toAirport": "CLE", "takeOffDate": "2015-01-20", "fromAirport": "MUA", "price": 954, "airlineName": "American Airlines", "seatsAvailable": 100, "code1": "rree"}, {"planeType": "Boeing 777", "code2": "1994", "toAirport": "SFO", "takeOffDate": "2015-01-01", "fromAirport": "MUA", "price": 676, "airlineName": "American Airlines", "seatsAvailable": "none", "code1": "rree"}, {"planeType": "Boeing 737", "code2": "2000", "toAirport": "SFO", "takeOffDate": "2015-02-20", "fromAirport": "MUA", "price": 300, "airlineName": "American Airlines", "seatsAvailable": 30, "code1": "rree"}, {"planeType": "Boeing 737", "code2": "3000", "toAirport": "SFO", "takeOffDate": "2015-02-01", "fromAirport": "MUA", "price": 900, "airlineName": "American Airlines", "seatsAvailable": "none", "code1": "rree"}, {"planeType": "Boeing 777", "code2": "4511", "toAirport": "LAX", "takeOffDate": "2015-01-15", "fromAirport": "MUA", "price": 900, "airlineName": "American Airlines", "seatsAvailable": 100, "code1": "rree"}, {"planeType": "Boeing 737", "code2": "4567", "toAirport": "SFO", "takeOffDate": "2015-01-20", "fromAirport": "MUA", "price": 456, "airlineName": "American Airlines", "seatsAvailable": 100, "code1": "rree"}]
```

*Note: To set the width of the console, select Anypoint Studio > Preferences > Run/Debug > Console and select Fixed width console and set a maximum character width.*

## Set a flow variable to hold the value of a query parameter

30. Select the Set Destination transformer in the getUnitedFlightsFlow and from the main menu, select Edit > Copy (or press Cmd+C/Ctrl+C).
31. Click in the process section of the getAmericanFlightsFlow and from the main menu, select Edit > Paste (or press Cmd+V/Ctrl+V); you should see a copy of the transformer added to the flow.
32. Move the transformer before the American DB Request endpoint.

33. In the Variable Properties view, change the display name to Set Destination and review the expression.



## Modify the query to use a dynamic destination

34. In the Properties view for the Database connector endpoint, modify the query to use the destination flow variable.

```

SELECT *
FROM flights
WHERE toAirport = #[flowVars.destination]

```

35. Save to redeploy the application and make another request to <http://localhost:8081/american>; you should now only see flights to SFO.

```

[{"planeType": "Boeing 737", "code2": 1093, "toAirport": "SFO", "takeOffDate": "2015-02-11", "fromAirport": "MUA", "price": 142, "airlineName": "American Airlines", "seatsAvailable": 1, "code1": "rree"}, {"planeType": "Boeing 777", "code2": 1994, "toAirport": "SFO", "takeOffDate": "2015-01-01", "fromAirport": "MUA", "price": 676, "airlineName": "American Airlines", "seatsAvailable": null, "code1": "rree"}, {"planeType": "Boeing 737", "code2": 2000, "toAirport": "SFO", "takeOffDate": "2015-02-20", "fromAirport": "MUA", "price": 300, "airlineName": "American Airlines", "seatsAvailable": 30, "code1": "rree"}, {"planeType": "Boeing 737", "code2": 3000, "toAirport": "SFO", "takeOffDate": "2015-02-01", "fromAirport": "MUA", "price": 900, "airlineName": "American Airlines", "seatsAvailable": null, "code1": "rree"}, {"planeType": "Boeing 737", "code2": 4567, "toAirport": "SFO", "takeOffDate": "2015-01-20", "fromAirport": "MUA", "price": 456, "airlineName": "American Airlines", "seatsAvailable": 100, "code1": "rree"}]

```

36. Make another request to <http://localhost:8081/american?code=CLE>; you should now see flights to CLE.

```

[{"planeType": "Boeing 747", "code2": 0123, "toAirport": "CLE", "takeOffDate": "2015-01-25", "fromAirport": "MUA", "price": 300, "airlineName": "American Airlines", "seatsAvailable": 7, "code1": "rree"}, {"planeType": "Boeing 737", "code2": 1000, "toAirport": "CLE", "takeOffDate": "2015-01-20", "fromAirport": "MUA", "price": 200, "airlineName": "American Airlines", "seatsAvailable": 5, "code1": "rree"}, {"planeType": "Boeing 787", "code2": 1112, "toAirport": "CLE", "takeOffDate": "2015-01-20", "fromAirport": "MUA", "price": 954, "airlineName": "American Airlines", "seatsAvailable": 100, "code1": "rree"}]

```



## Walkthrough 4-2: Connect to a file (CSV)

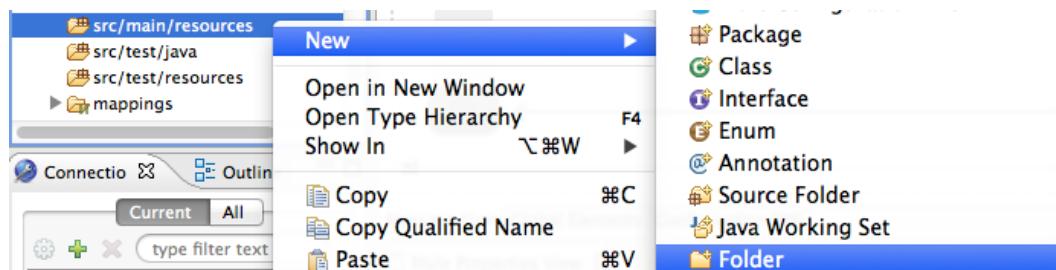
In this walkthrough, you will load data from a local CSV file. You will:

- Add and configure a File connector endpoint to watch an input directory, read the contents of any added files, and then rename the files and move them to an output folder.
- Add a CSV file to the input directory and watch it renamed and moved.
- Restrict the type of file read.
- Use the File to String transformer to return the results as a string.

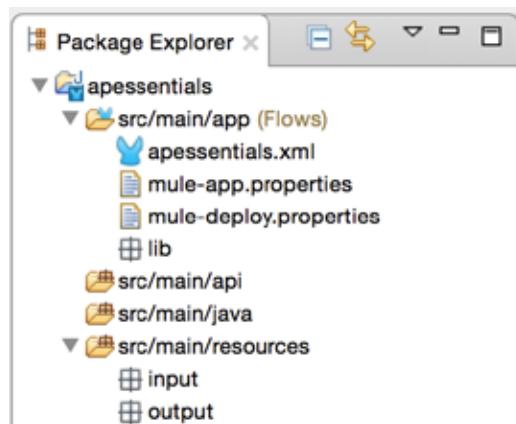


### Create new directories

1. Return to apessentials.xml.
2. Right-click the src/main/resources folder in the Package Explorer and select New > Folder.

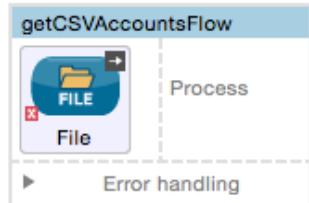


3. Set the folder name to input and click Finish.
4. Create a second folder called output.

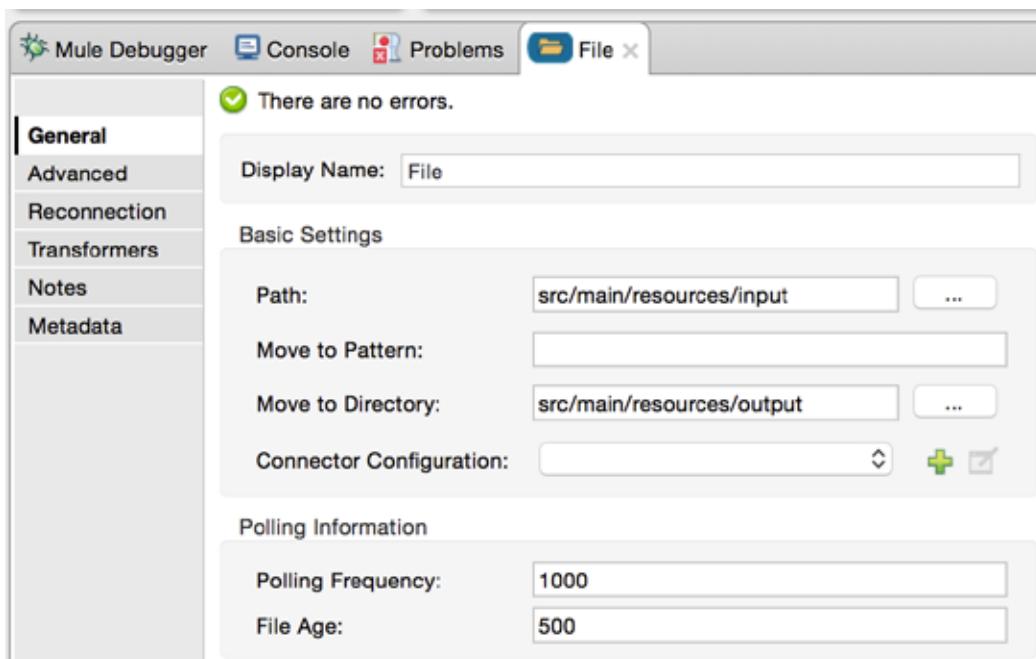


## Add and configure a File connector endpoint

5. Drag a File connector from the palette and drop it in the canvas to create a new flow.
6. Rename the flow to getCSVAccountsFlow.



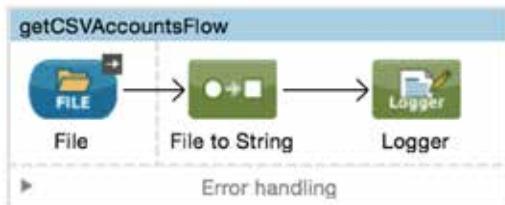
7. In the File Properties view, set the path to src/main/resources/input.
8. Set the move to directory to src/main/resources/output.



9. Look at the default polling information; the endpoint checks for incoming messages every 1000 milliseconds and sets a minimum of 500 milliseconds a file must wait before it is processed.

## Display the file contents

10. Add a File to String transformer to the process section of the flow.
11. Add a Logger after the transformer.



12. In the Logger Properties view, set the message to display the payload.

## Run the application

13. Save the file to redeploy the application.

14. Leave the application running.

## Add a CSV file to the input directory

15. Right-click src/main/resources in the Package Explore and select Show In > System Explorer.

16. In another window, navigate to the student files folder for the course:

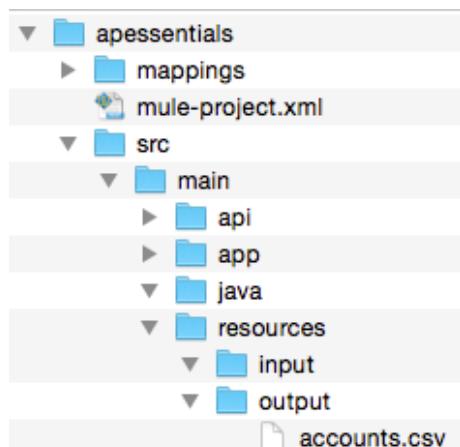
APEssentials3.7\_studentFiles\_{date}.

17. Locate the resources/accounts.csv file.

18. Make a copy of this file and put it in the src/main/resources folder of the apessentials project.

## Test the application

19. Drag the CSV into the input folder; you should see it almost immediately moved to the output folder.



20. Drag it back into the input folder; it will be read again and then moved to the output folder.

21. Leave this folder open.

## Debug the application

22. Return to apessentials.xml.

23. Add a breakpoint to the Logger.

24. Debug the application.

25. Move the accounts.csv file from the output folder back to the input folder.

*Note: You can do this in your operating system window or in Anypoint Studio. If you use Anypoint Studio, you will need to right-click on the project and select Refresh to see the file.*

26. Wait until code execution stops at the Logger.

27. Drill-down and look at the payload.

28. Look at the inbound message properties and locate the original filename.

The screenshot shows the Mule Debugger interface. On the left, the 'Inbound' tab of the properties view is selected, displaying the following data:

Name	Value	Type
Message	org.mule.DefaultM...	org.mule.api.proce...
Message Pr...	Logger	org.mule.api.proce...
payload	Billing Street,Billing...	java.lang.String

Below this, the payload content is expanded:

```
Billing Street,Billing City,Billing
Country,Billing State,Name,BillingPostalCode
111 Boulevard Hausmann,Paris,France,,Dog Park
Industries,75008
400 South St,San Francisco,USA,CA,Iguana Park
Industries,91156
777 North St,San Francisco,USA,CA,Cat Park
```

On the right, the 'Record' tab of the properties view is selected, showing the following data:

Name	Value	Type
directory	/Users/jeanette.stallo...	java.lang.String
fileSize	267	java.lang.Long
MULE_ORIGIN...	endpoint.file.src.main...	java.lang.String
originalDirectory	/Users/jeanette.stallo...	java.lang.String
originalFilename	accounts.csv	java.lang.String
timestamp	1420586276000	java.lang.Long

The 'accounts.csv' entry is highlighted.

29. Click the Resume button in the Mule Debugger view.

30. Stop the debugger.

## Restrict the type of file read

31. Open the Properties view for the File endpoint.

32. Click the Add button next to file name regex filter.

33. In the File Name Regex Filter dialog box, set the pattern to \*.csv and uncheck case sensitive.

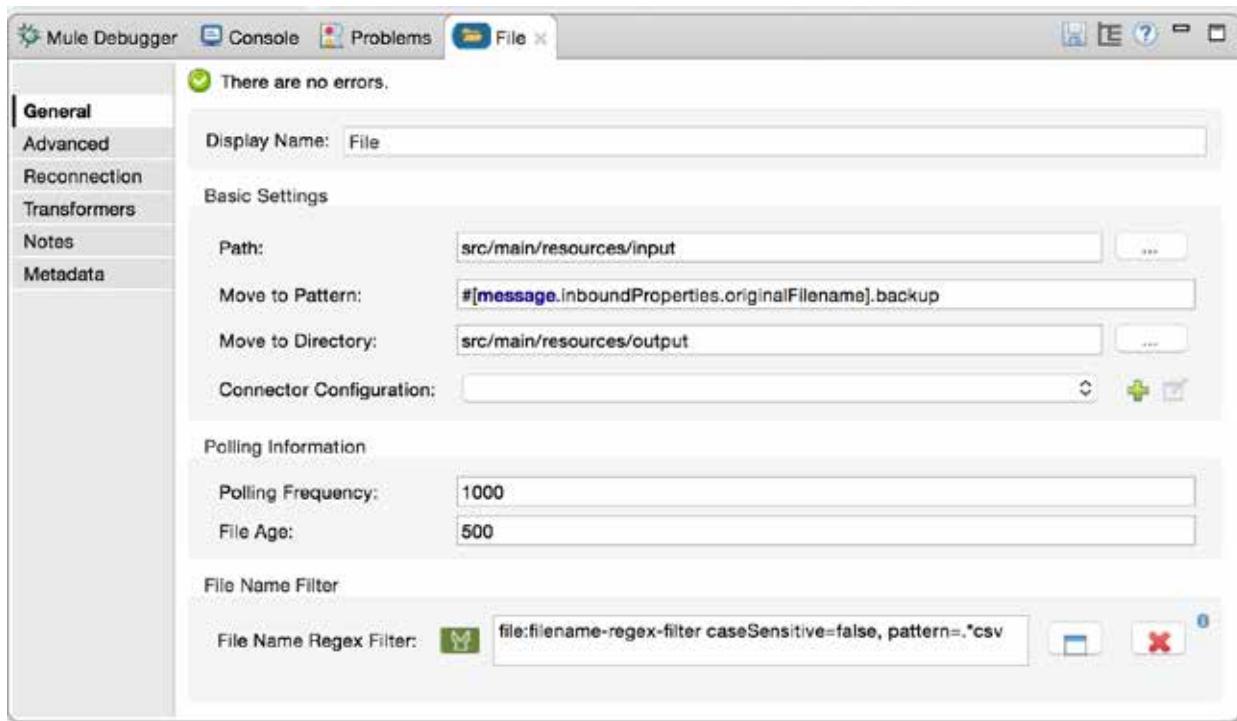


34. Click Finish.

## Rename the file

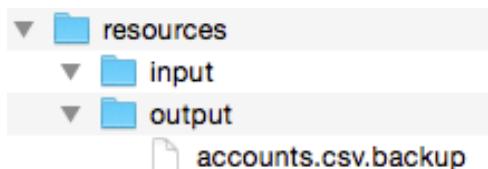
- Set the move to pattern to append .backup to the original filename.

```
##[message.inboundProperties.originalFilename].backup
```



## Test the application

- Save the file and run the application.
- Move the accounts.csv file from the output folder back to the input folder; you should see it appear in the output folder with its new name.



- Look at the console; you should see the contents of the file displayed.

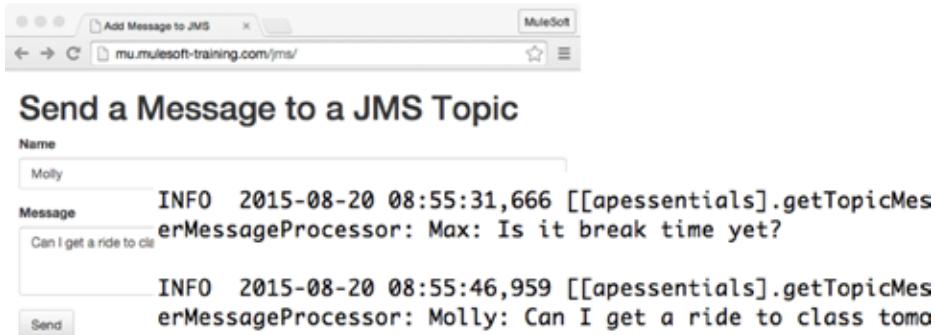
```
INFO 2015-08-19 13:44:16,688 [[apessentials].getCSVAccountsFlow.stage1.02] org.mule.api.processor.Logger
MessageProcessor: Billing Street,Billing City,Billing Country,Billing State,Name,BillingPostalCode
111 Boulevard Hausmann,Paris,France,,Dog Park Industries,75008
400 South St,San Francisco,USA,CA,Iguana Park Industries,91156
777 North St,San Francisco,USA,CA,Cat Park Industries,91156
```

- Move the accounts.csv.backup file from the output folder back to the input folder; it should not be processed and moved to the output folder.

## Walkthrough 4-3: Connect to a JMS queue (ActiveMQ)

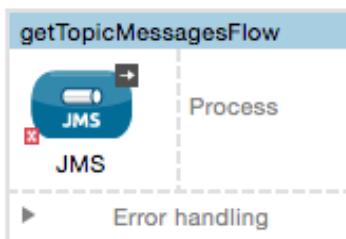
In this walkthrough, you will read and write messages from a JMS topic. You will:

- Create a flow accessible at <http://localhost:8081/jms>.
- Add and configure an ActiveMQ connector.
- Use a JMS endpoint to retrieve messages from a JMS topic.
- Add messages to the topic using a web form.
- Use a JMS endpoint to send messages to a JMS topic.



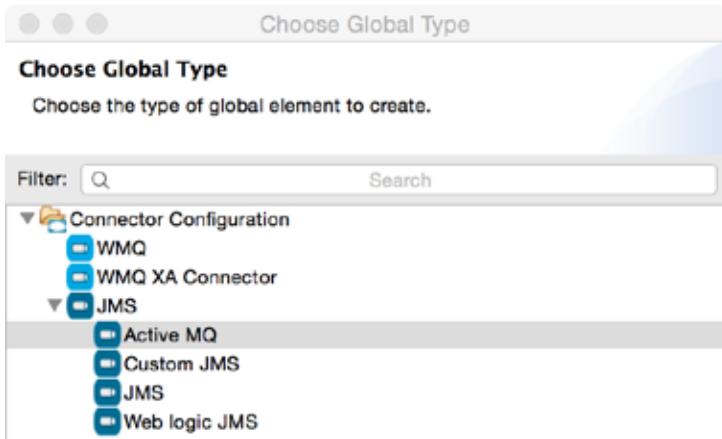
### Create a JMS inbound-endpoint

1. Return to apessentials.xml.
2. Drag out a JMS connector and drop it at the bottom of the canvas to create a new flow.
3. Give the flow a new name of getTopicMessagesFlow.

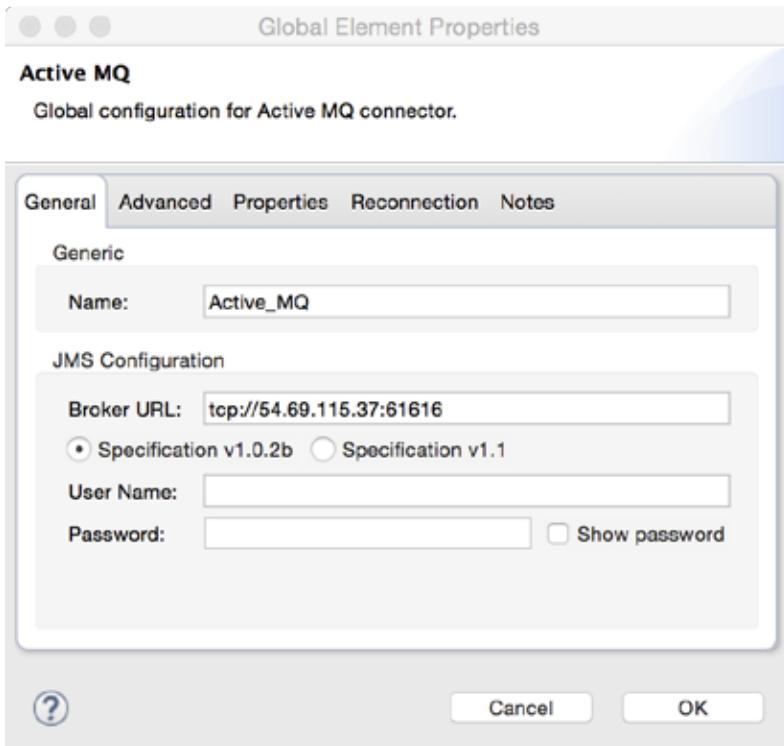


4. In the JMS Properties view, select topic and set it to apessentials.
5. Click the Add button next to connector configuration.

6. In the Choose Global Type dialog box, select Connector Configuration > JMS > Active MQ and click OK.



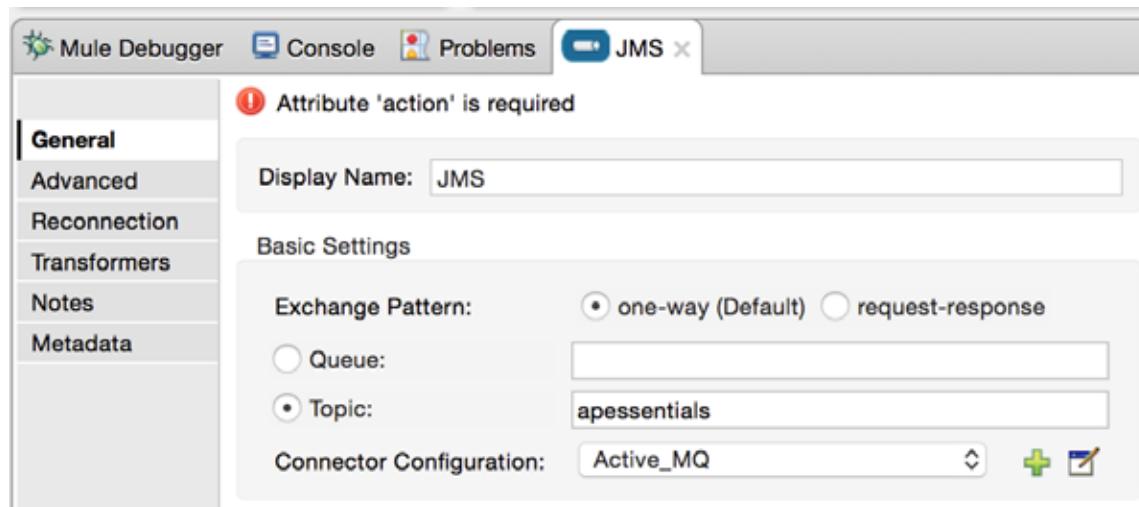
7. In the Global Element Properties dialog box, change the broker URL to the JMS ActiveMQ Broker URL listed in the course snippets.txt file.
8. Set the Specification to v1.1.



9. Click the Advanced tab and take a look at the various settings.
10. Click OK.
11. Click the Apply Changes button in the Properties view.

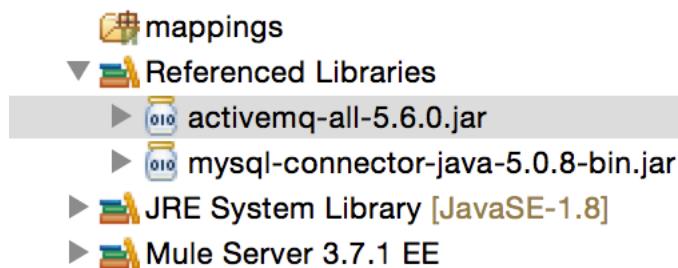
12. If you see an Attribute 'action' is required warning, ignore it.

*Note: This is a bug in the Anypoint Studio January 2015 release.*



## Add the ActiveMQ library

13. In the Package Explorer, right-click apessentials and select New > Folder.
14. In the New Folder dialog box, set the folder name to lib and click Finish.
15. Locate activemq-all-{version}.jar file located in the APEssentials3.7\_studentFiles\_{date}/jars folder.
16. Copy and paste or drag the JAR file into your lib folder.
17. Right-click the JAR file in the Package Explorer and select Build Path > Add to Build Path; you should now see it under Referenced Libraries.



*Note: Adding activemq-all.jar can create conflicts with other dependencies in projects, so it is recommended that you only add only the jar files you need in relation to what you need ActiveMQ for. For more details, see the documentation at <http://www.mulesoft.org/documentation/display/current/ActiveMQ+Integration>.*

## Test the application and receive messages

18. Add a Logger to the process section of the flow and set its message to #[payload].
19. Save the file and run the application.
20. Make a request to the JMS form URL listed in the course snippets file.
21. In the form, enter your name and a message and click Send.



### Send a Message to a JMS Topic

Name

Message

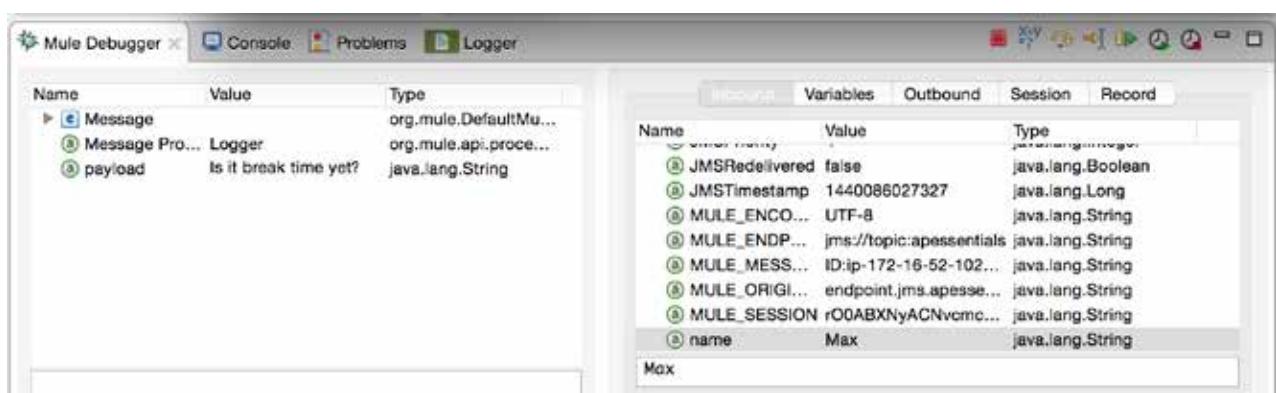
22. Return to the console in Anypoint Studio; you should see your message along with those from your classmates – but you don't see the names of the people who sent the messages.

```
INFO 2015-08-20 08:52:41,053 [[apessentials].getTopicMessagesFlow.stage1.02] org.mule.api.processor.LoggerMessageProcessor: Is it break time yet?
```

```
INFO 2015-08-20 08:52:57,613 [[apessentials].getTopicMessagesFlow.stage1.02] org.mule.api.processor.LoggerMessageProcessor: Can I get a ride to class tomorrow?
```

## Debug the application

23. Add a breakpoint to the Logger.
24. Debug the application.
25. Make another request to the JMS form URL and submit another message.
26. In the Mule Debugger view, look at the payload and inbound message properties.



Name	Type
Message	org.mule.DefaultMu...
Message Pro... Logger	org.mule.api.proce...
payload	java.lang.String

Inbound	Variables	Outbound	Session	Record
Name	Value	Type		
④ JMSRedelivered	false	java.lang.Boolean		
④ JMSTimestamp	1440086027327	java.lang.Long		
④ MULE_ENCO...	UTF-8	java.lang.String		
④ MULE_MESS...	jms://topic:apessentials	java.lang.String		
④ MULE_ORIGI...	endpoint.jms.apesse...	java.lang.String		
④ MULE_SESSION	rO0ABXNyACNvcmc...	java.lang.String		
④ name	Max	java.lang.String		

27. Stop the Mule Debugger.

## Display names with the messages

28. In the Logger Properties view, change the message to use an expression to display the name and message.

```
# [message.inboundProperties.name + ":" + payload]
```

## Test the application

29. Save the file and run the application.

30. Return to the message form and submit a new message.

31. Look at the console; you should now see names along with messages.

```
INFO 2015-08-20 08:55:31,666 [[apessentials].getTopicMessagesFlow.stage1.02] org.mule.api.processor.LoggerMessageProcessor: Max: Is it break time yet?
```

```
INFO 2015-08-20 08:55:46,959 [[apessentials].getTopicMessagesFlow.stage1.02] org.mule.api.processor.LoggerMessageProcessor: Molly: Can I get a ride to class tomorrow?
```

## Create a JMS outbound-endpoint

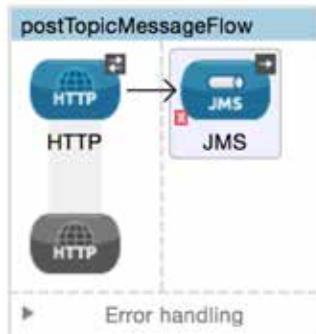
32. Drag out another HTTP connector and drop it in the canvas to create a new flow.

33. Give the flow a new name of postTopicMessageFlow.

34. In the HTTP Properties view, set the connector configuration to the existing  
HTTP\_Listener\_Configuration.

35. Set the path to /jms and the allowed methods to GET.

36. Drag out another JMS connector and drop it into the process section of the flow.



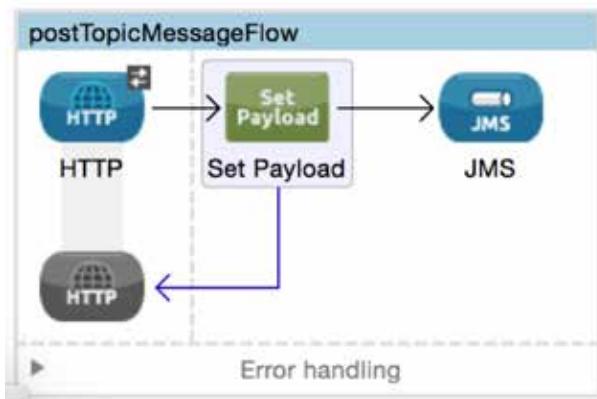
37. In the JMS Properties view, select topic and set it to apessentials.

38. Set the connector configuration to the existing Active\_MQ.

39. If you see an Attribute 'action' is required warning, ignore it.

## Set a message

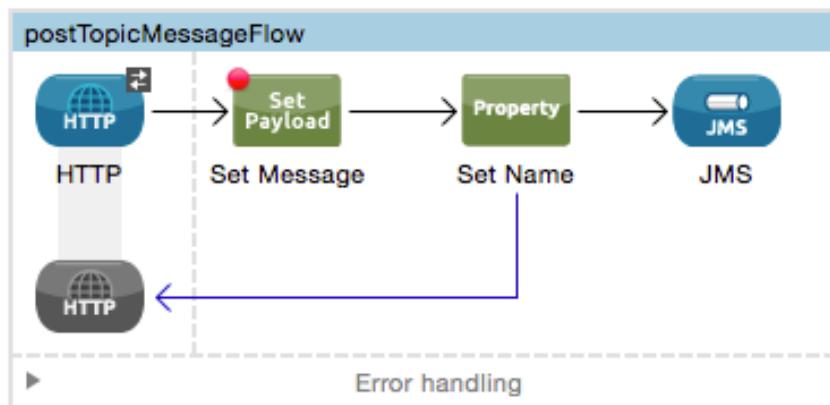
40. Add a Set Payload transformer between the HTTP and JMS connector endpoints.



41. In the Set Payload Properties view, change the display name to Set Message and set the value to a message query parameter.

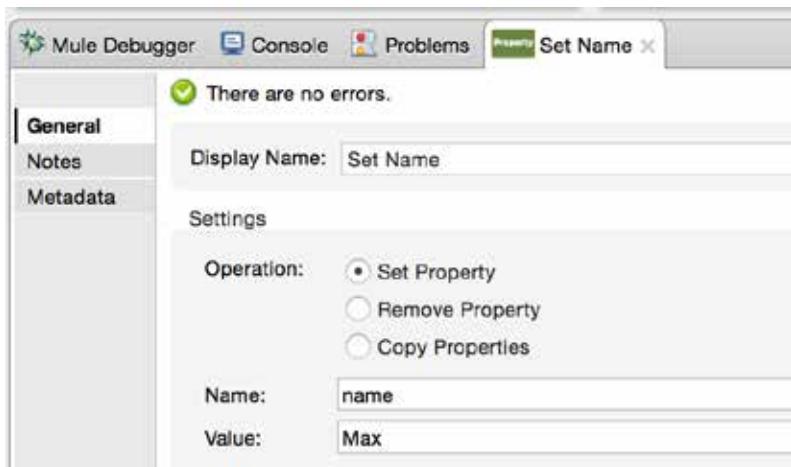


42. Add a breakpoint to the Set Payload transformer.  
43. Add a Property transformer after the Set Payload transformer.  
44. In the Properties view, change the display name to Set Name.



45. Select Set Property and set the name to name and the value to your name.

*Note: You can set this to a query parameter instead if you prefer.*



## Test the application and post messages

46. Save the file to redeploy the application and make a request to

<http://localhost:8081/jms?message=Hello>.

47. Look at the console; you should see your name and message displayed – along with those of your classmates.

```
INFO 2015-08-20 09:01:28,411 [[apessentials].getTopicMessagesFlow.stage1.02] org.mule.api.processor.LoggerMessageProcessor: Max: Hello
```

## Walkthrough 4-4: Connect to a SaaS application (Salesforce)

In this walkthrough, you will retrieve account records from Salesforce. You will:

- Browse Salesforce data on <http://salesforce.com>.
- Create a flow accessible at <http://localhost:8081/sfdc>.
- Add a Salesforce connector endpoint to retrieve accounts for a specific postal code.
- Use the Query Builder to write a query.
- Use the Object to JSON transformer to return the results as a string.

The screenshot shows the Salesforce mobile app interface. At the top, there's a banner with the text "Take Salesforce with you wherever you go. Run your business from any mobile device with the Salesforce Mobile App." Below the banner, the main screen displays a list of accounts. A search bar is at the top left, and a navigation bar with tabs like Home, Chatter, Campaigns, Leads, Accounts, Contacts, Opportunities, Forecasts, Contracts, Orders, Cases, and Solutions is at the top right. On the left, there's a sidebar with "Recent Items" and a "Create New..." button. The main content area shows a table with columns: Name, Value, and Type. One row is highlighted with a yellow background, showing the value "#[ payload.next() ]".

*Note: To complete this walkthrough, you need a Salesforce Developer account. Instructions for creating a Salesforce developer account and getting an API access token are included in the Setup instructions at the beginning of this student manual.*

### Look at existing Salesforce account data

1. In a browser, go to <http://login.salesforce.com/> and log in with your Salesforce Developer account.
2. Click the Accounts link in the main menu bar.
3. In the view drop-down, select All Accounts and click the Go button.

The screenshot shows the Salesforce web interface. At the top, there's a banner with the text "Take Salesforce with you wherever you go. Run your business from any mobile device with the Salesforce1 Mo...". Below the banner, the main screen displays a list of accounts. A search bar is at the top left, and a navigation bar with tabs like Home, Chatter, Campaigns, Leads, Accounts, Contacts, Opportunities, Forecasts, and Contracts is at the top right. On the left, there's a sidebar with "Recent Items" and a "Create New..." button. The main content area shows a table with columns: Name, Value, and Type. One row is highlighted with a yellow background, showing the value "#[ payload.next() ]".

- Look at the existing account data; a Salesforce Developer account is populated with some sample data.

The screenshot shows a Salesforce interface for managing accounts. At the top, there's a navigation bar with a folder icon, the text 'All Accounts', and links for 'Edit | Delete | Create New View'. Below this is a header row with a 'New Account' button and a refresh icon. A navigation menu above the main table includes letters A through N. The main table has columns for 'Action', 'Account Name', 'Billing State/Prov...', and 'Phone'. Six rows of account data are listed:

Action	Account Name	Billing State/Prov...	Phone
<a href="#">Edit</a>   <a href="#">Del</a>   <a href="#">+</a>	Burlington Textiles ...	NC	(336) 222-7000
<a href="#">Edit</a>   <a href="#">Del</a>   <a href="#">+</a>	Dickenson plc	KS	(785) 241-6200
<a href="#">Edit</a>   <a href="#">Del</a>   <a href="#">+</a>	Edge Communicati...	TX	(512) 757-6000
<a href="#">Edit</a>   <a href="#">Del</a>   <a href="#">+</a>	Express Logistics a...	OR	(503) 421-7800
<a href="#">Edit</a>   <a href="#">Del</a>   <a href="#">+</a>	GenePoint	CA	(650) 867-3450
<a href="#">Edit</a>   <a href="#">Del</a>   <a href="#">+</a>	Grand Hotels & Res...	IL	(312) 596-1000

- Notice that countries and postal codes are not displayed by default.
- Click the Create New View link next to the drop-down displaying All Accounts.
- Set the view name to All Accounts with Postal Code.
- Locate the Select Fields to Display section.
- Select Billing Zip/Postal Code as the available field and click the Add button.
- Add the Billing Country and Account Number fields.
- Use the Up and Down buttons to order the fields as you prefer.
- Click the Save button; you should now see all the accounts with postal codes and countries.

The screenshot shows the same Salesforce interface after creating a new view. The title bar now says 'All Accounts with Postal Code'. The main table has been updated to include four new columns: 'Billing Zip/Postal Code', 'Billing Country', 'Billing State/Province', and 'Account Number'. The data remains the same as in the previous screenshot, with the new columns providing more detailed information for each account entry.

Action	Account Name	Billing Zip/Postal Code	Billing Country	Billing State/Province	Account Number
<a href="#">Edit</a>   <a href="#">Del</a>   <a href="#">+</a>	Burlington Textiles Co...	27215	USA	NC	CD656092
<a href="#">Edit</a>   <a href="#">Del</a>   <a href="#">+</a>	Dickenson plc	66045	USA	KS	CC634267
<a href="#">Edit</a>   <a href="#">Del</a>   <a href="#">+</a>	Edge Communications			TX	CD451796
<a href="#">Edit</a>   <a href="#">Del</a>   <a href="#">+</a>	Express Logistics a...			OR	CC947211
<a href="#">Edit</a>   <a href="#">Del</a>   <a href="#">+</a>	GenePoint			CA	CC978213

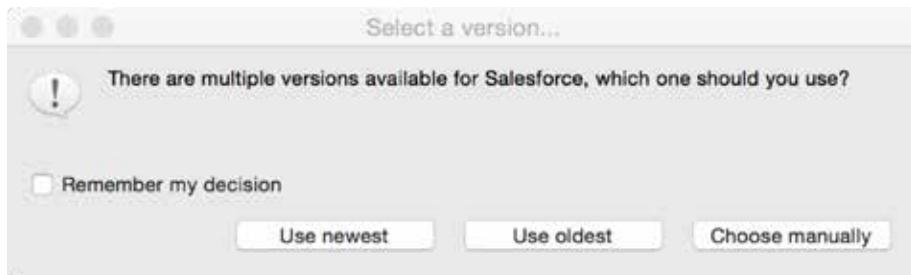
## Create a new project and flow

- Return to apessentials.xml in Anypoint Studio.
- Drag out an HTTP connector and drop it above the getCSVAccountsFlow to create a new flow.
- Give the flow a new name of getSFDCAccountsFlow.
- In the HTTP Properties view, set the connector configuration to the existing `HTTP_Listener_Configuration`.
- Set the path to `/sfdc` and the allowed methods to GET.

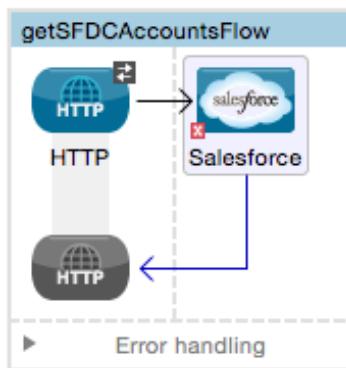


## Add a Salesforce connector endpoint

18. Drag out a Salesforce connector and drop it in the process section of the flow.
19. In the Select a version dialog box, click Use newest.

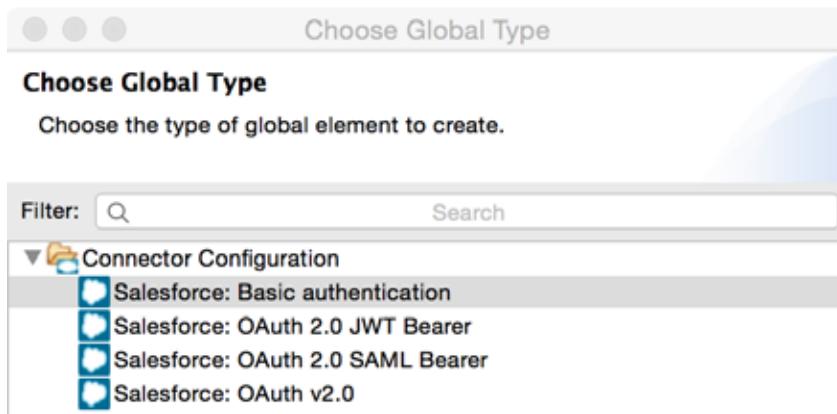


20. Examine the flow.



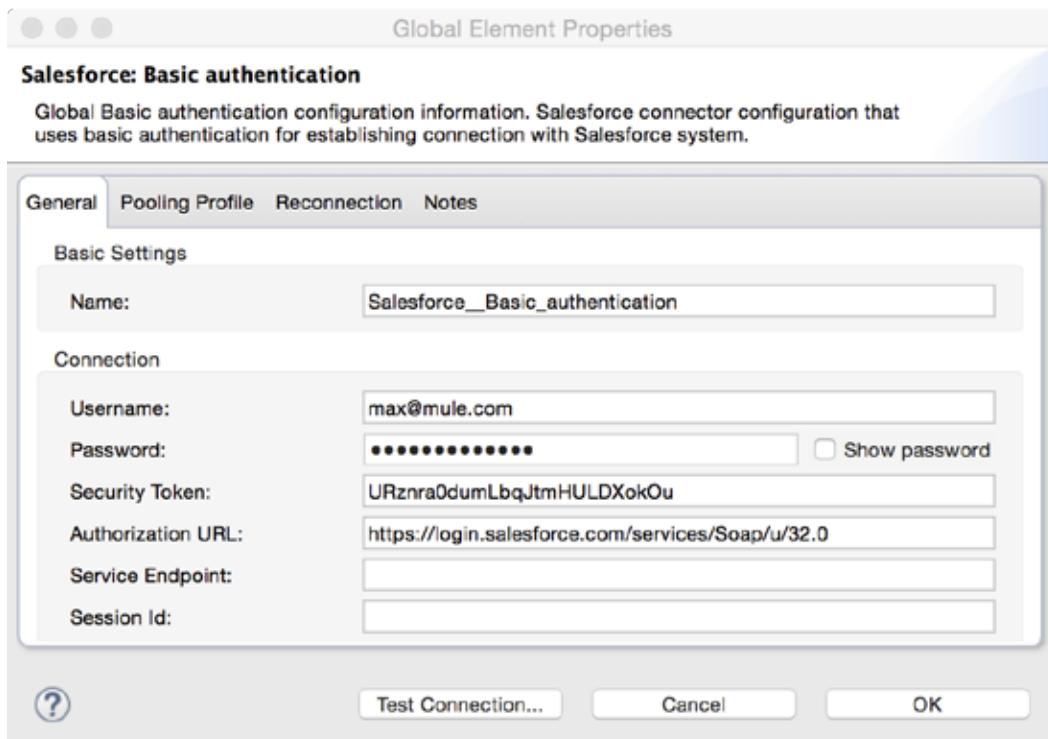
## Configure the Salesforce connector

21. In the Salesforce Properties view, click the Add button next to Connector Configuration.
22. In the Choose Global Type dialog box, select Connector Configuration > Salesforce: Basic authentication and click OK.



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

*Note: Instructions for creating a Salesforce Developer account and getting a security token are included in the Setup instructions at the beginning of this student manual.*



24. Click the Test Connection button; you will get a Test Connection dialog box letting you know if the connection succeeds or fails.



25. Click OK to close the Test connection dialog box.  
26. If your test connection failed, fix it; do not proceed until it is working.

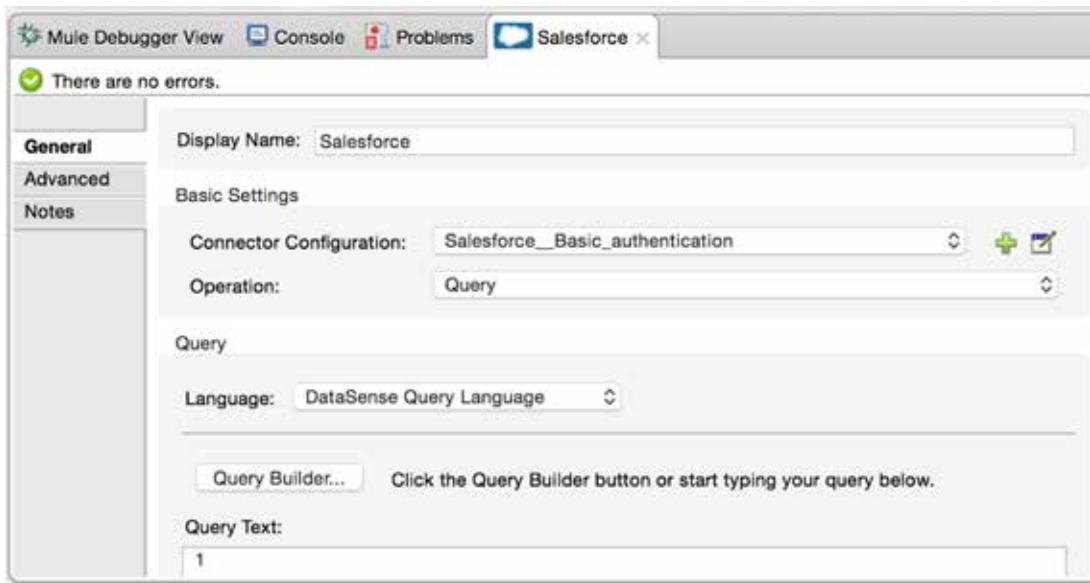
*Note: If it failed, check to see if you have any extra whitespace in your entries.*

27. Click OK to close the Global Elements Properties dialog box.

## Write the query

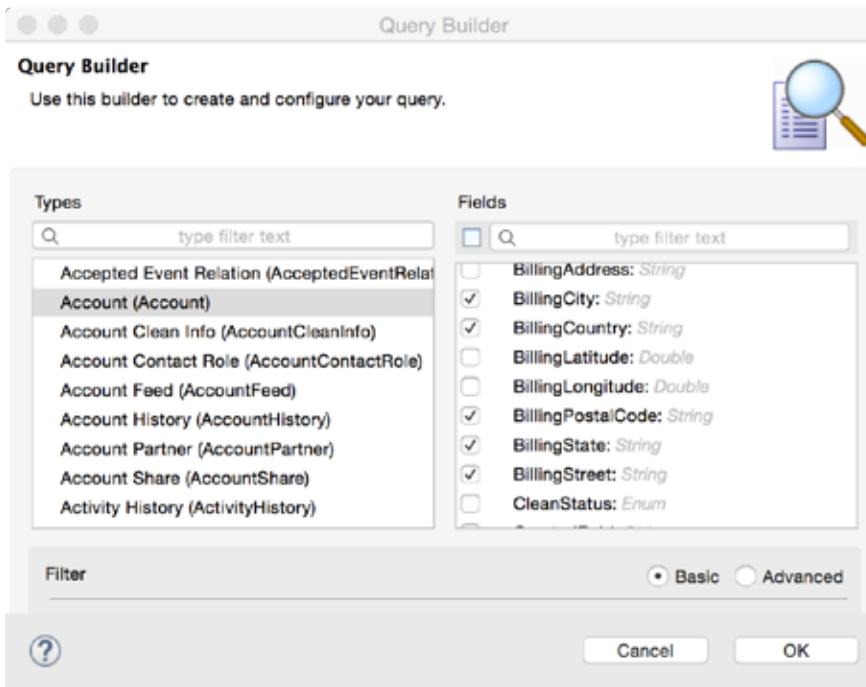
28. In the Salesforce Properties view, select an operation of Query.

29. Click the Query Builder button.



30. In the Query Builder dialog box, select a type of Account (Account).

31. Select fields BillingCity, BillingCountry, BillingPostalCode, BillingState, BillingStreet, and Name.



32. Click OK.

33. Examine the generated query.

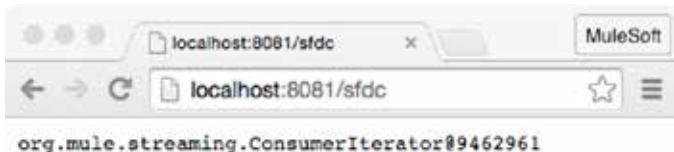


## Test the application

34. Save the file to redeploy the application.

*Note: If you get a SAXParseException, stop the Mule runtime and run the application again. If you still get an exception, close and reopen the project. If that does not work, restart Anypoint Studio.*

35. Make a request to <http://localhost:8081/sfdc>; you should see the type of Java object created.



## Debug the application

36. Add a Logger component after the Salesforce endpoint.
37. Set the Logger message to #[payload].
38. Add a breakpoint to the Logger.
39. Save the file and debug the application.
40. Make another request to <http://localhost:8081/sfdc>.
41. Step to the Logger and drill-down into the payload variable.
42. Click the Evaluate Mule Expression button in the Mule Debugger view.



43. Enter the following expression and press the Enter key.

```
#[payload.next()]
```

44. Expand the results; you should see the account data for the first account.

Name	Value	Type
#[ payload.next() ]		
▼ e #[payload.next()]	size = 8	java.util.HashMap
► e 0	BillingCountry=null	java.util.HashMap\$Node
► e 1	BillingCity=null	java.util.HashMap\$Node
► e 2	BillingStreet=Kings Par...	java.util.HashMap\$Node
► e 3	BillingPostalCode=null	java.util.HashMap\$Node
► e 4	Id=null	java.util.HashMap\$Node
► e 5	type=Account	java.util.HashMap\$Node
► e 6	BillingState=UK	java.util.HashMap\$Node
► e 7	Name=United Oil & G...	java.util.HashMap\$Node

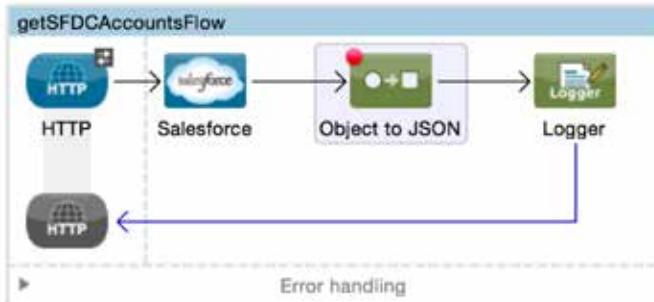


45. Click anywhere outside the expression window to close it.

46. Stop the Mule Debugger.

## Display the return data

47. Add an Object to JSON transformer before the Logger component.



48. Save the file and run the application.

49. Make another request to <http://localhost:8081/sfdc>; you should see the Salesforce accounts displayed.

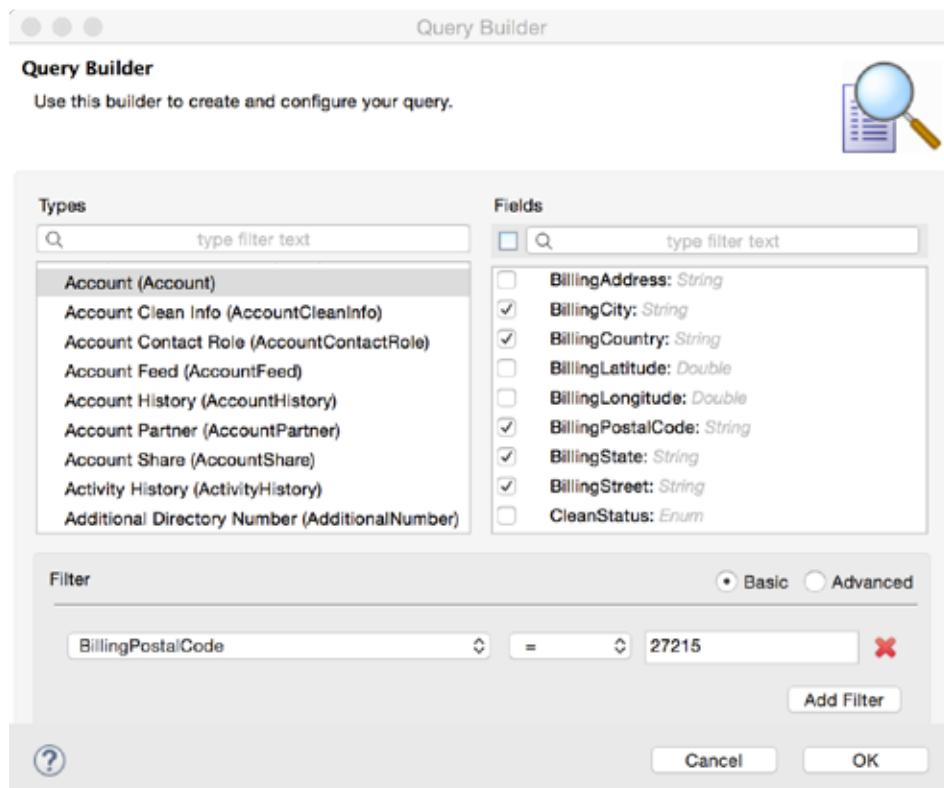
```
[{"BillingCountry":null,"BillingCity":"Mountain View","BillingStreet":"345 Shoreline Park\nMountain View, CA 94043\nUSA","BillingPostalCode":null,"Id":null,"type":"Account","BillingState":"CA","Name":"GenePoint"}, {"BillingCountry":null,"BillingCity":null,"BillingStreet":"Kings Park, 17th Avenue, Team Valley Trading Estate,\nGateshead, Tyne and Wear NE26 3HS\nUnited Kingdom","BillingPostalCode":null,"Id":null,"type":"Account","BillingState":"UK","Name":"United Oil & Gas, UK"}, {"BillingCountry":null,"BillingCity":"Singapore","BillingStreet":"9 Tagore Lane\nSingapore, Singapore 787472\nSingapore","BillingPostalCode":null,"Id":null,"type":"Account","BillingState":"Singapore","Name":"United Oil & Gas, Singapore"}, {"BillingCountry":null,"BillingCity":"Austin","BillingStreet":"312 Constitution Place\nAustin, TX 78767\nUSA","BillingPostalCode":null,"Id":null,"type":"Account","BillingState":"TX","Name":"Austin Oil & Gas, Inc."}]
```

## Modify the query

50. In the Salesforce Properties view, click the Query Builder button.

51. In the Query Builder dialog box, click the Add Filter button.

52. Create a filter for BillingPostalCode equal to one of the postal codes (like 27215) in your Salesforce account data.



53. Click OK.

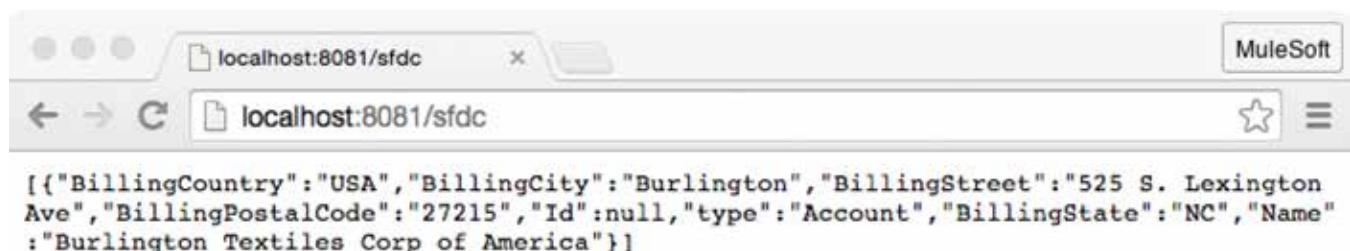
54. Examine the generated query.

```
Query Text:  
1 SELECT BillingCity,BillingCountry,BillingPostalCode,BillingState,BillingStreet,Name  
2 FROM Account  
3 WHERE BillingPostalCode = '27215'
```

## Test the application

55. Save the file to redeploy the application.

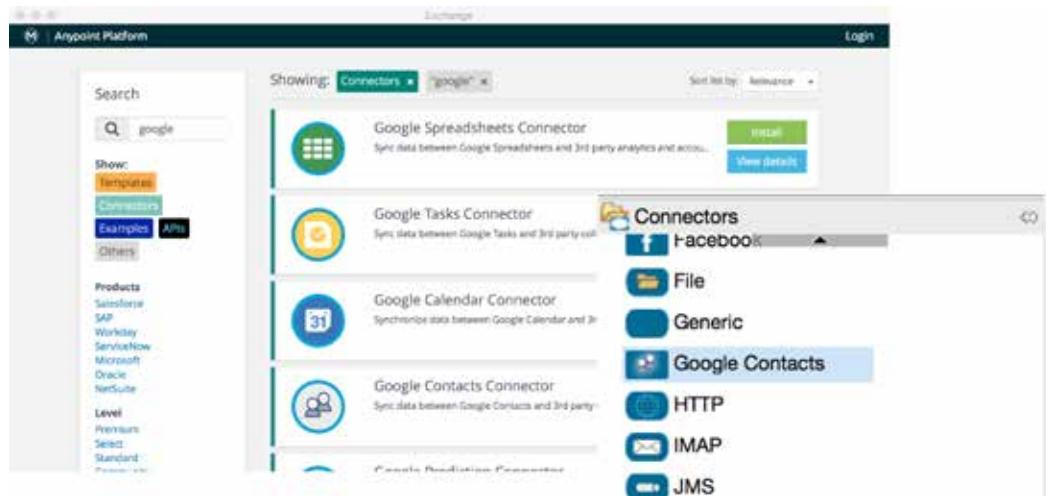
56. Make another request to <http://localhost:8081/sfdc>; you should only see the accounts with the specified postal code displayed.



## Walkthrough 4-5: Find and install not-in-the-box connectors

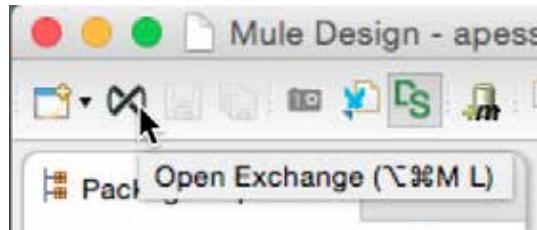
In this walkthrough, you will learn how to add a new connector to Anypoint Studio. You will:

- Browse the Anypoint Exchange.
- Add a new connector to Anypoint Studio.

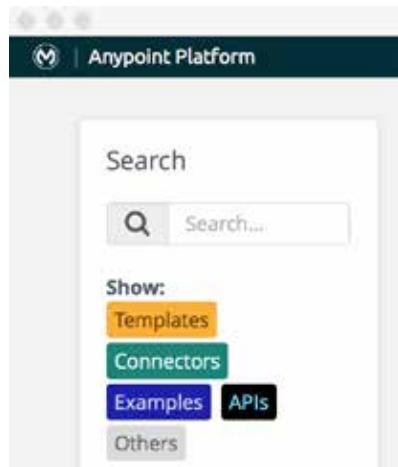


### Browse the Anypoint Exchange from Anypoint Studio

1. In Anypoint Studio, click the Open Exchange button; the Anypoint Exchange should open in a new window.



2. Click the Connectors button.



3. Browse the connectors, exploring the different levels and categories.

The screenshot shows the Anypoint Platform Exchange interface. The top navigation bar includes the MuleSoft logo, 'Anypoint Platform', 'Exchange', and 'Login'. On the left, there's a sidebar with a search bar, a 'Show' dropdown menu (Templates, Connectors, Examples, APIs, Others), a 'Products' section listing various enterprise systems like Salesforce, SAP, Workday, etc., and a 'Level' section (Premium, Select, Standard). The main content area is titled 'Showing: Connectors' and lists four connectors:

- Salesforce Analytics Cloud Connector: 'Install' (green button), 'View details' (blue button)
- Amazon Simple Storage Service (S3) Connector: 'Installed' (grey button), 'View details' (blue button)
- Zuora Connector: 'Install' (green button), 'View details' (blue button)
- Zendesk Connector: 'Install' (green button), 'View details' (blue button)

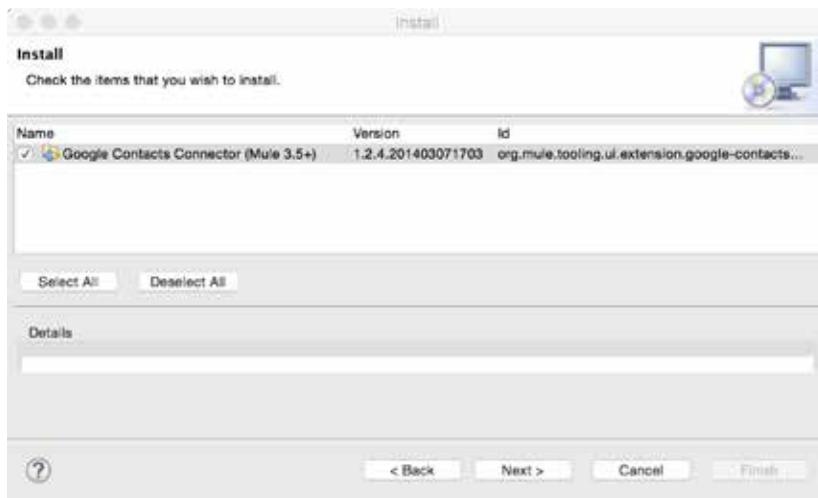
4. Locate the Google Contacts connector (or any other connector you are interested in).  
5. Click the View details button and browse the connector's documentation.

The screenshot shows the detailed documentation for the Google Contacts Connector. The top navigation bar is identical to the previous screenshot. The left sidebar lists other connectors: Google Spreadsheets Connector, Google Tasks Connector, Google Calendar Connector, and Google Contacts Connector (which is highlighted with a green border). The main content area for the Google Contacts Connector includes:

- Google Contacts Connector**: 'Install' (green button), 'Share URL', 'Contact us'
- A note: 'Please install this Connector to rate it.'
- Description**: A detailed paragraph explaining how to integrate Google Contacts using the connector.
- Level**: Community
- Commonly connects to**: Marketo, SAP, Zuora
- Category**: Business-Process-Administration

## Install the connector

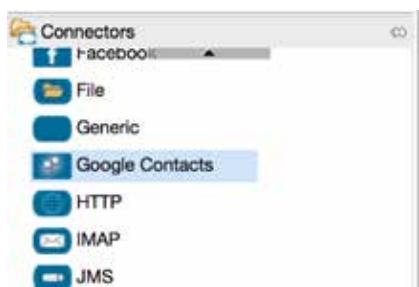
6. In the Anypoint Exchange, click the connector's Install button.
7. If you get a Terms of Service dialog box, enter your personal information and click Install.
8. Wait until an Install dialog box appears in Anypoint Studio.



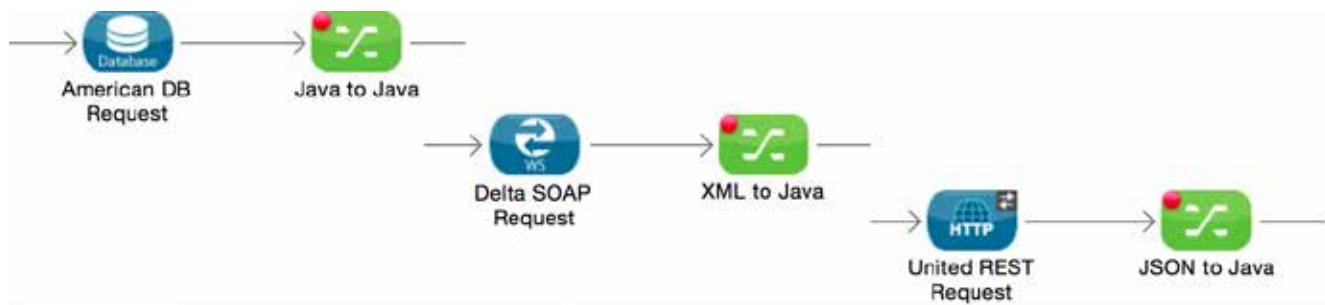
*Note: If you do not see the connector listed, resize the Install dialog box.*

*Note: You can also install connector's directly from Anypoint Studio. From the main menu bar, select Help > Install New Software. In the Install dialog box, click the Work with drop-down button and select Anypoint Connectors Update Site. Drill-down into Community and select the Google Contacts Connector (or some other connector).*

9. Click Next.
10. On the Install Details page, click Next.
11. On the Review Licenses page, select the I accept the terms radio button.
12. Click Finish; the software will be installed and then you will get a message to restart Anypoint Studio.
13. In the Software Updates dialog box, click Yes to restart Anypoint Studio.
14. After Anypoint Studio restarts, locate the new connector in the Connectors section of the palette.



# Module 5: Transforming Data



Name	Value
root : ArrayList	
[0] : LinkedHashMap	
airlineName : String	????
departureDate : String	????
destination : String	????
emptySeats : Integer	1
flightCode : String	????
origination : String	????
planeType : String	????
price : Integer	2

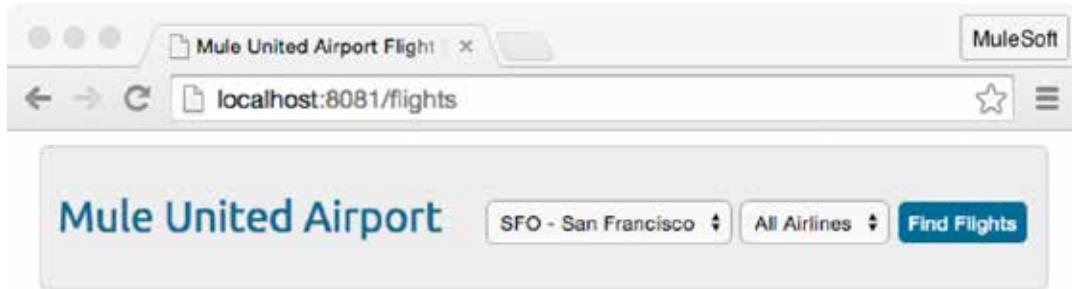
In this module, you will learn:

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

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

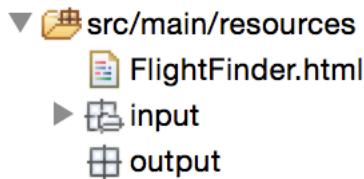
In this walkthrough, you will add an HTML form to your application that will eventually serve as a front end for the MUA integration application. You will:

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



### Add an HTML file to the project

1. On your computer, navigate to the student files folder for the course, APESentials3.7\_studentFiles\_{date}.
2. Copy and paste (or drag) the resources/FlightFinder.html file into your project's src/main/resources folder.



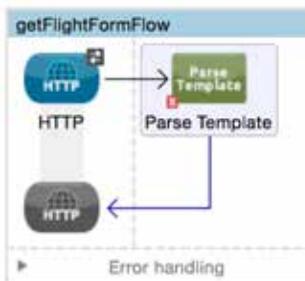
*Note: You will examine the HTML code soon – after you load the form and see what it looks like.*

### Create a flow

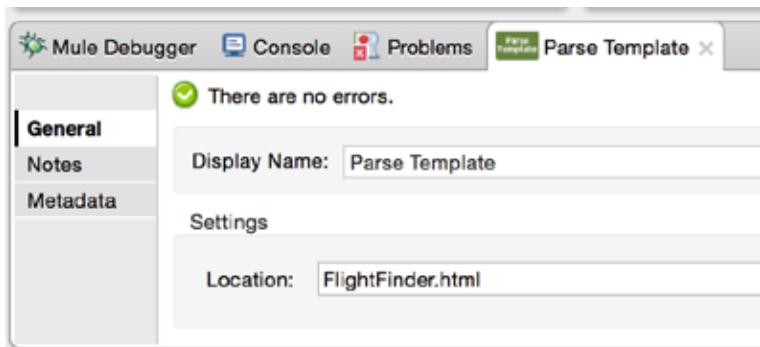
3. Return to apessentials.xml.
4. Drag out another HTTP connector to create a new flow at the top of the canvas and name it getFlightFormFlow.
5. In the HTTP Properties view, set the connector configuration to the existing HTTP\_Listener\_Configuration.
6. Set the path to /flights and set the allowed methods to GET.

## Set the payload to HTML and JavaScript

7. Add a Parse Template transformer to the process section of the flow.



8. In the Properties view for the transformer, set the location to FlightFinder.html.



## Debug the application

9. Add a breakpoint to the Parse Template transformer.
10. Add a Logger component after the transformer.
11. Save the file and debug the application.
12. In a browser, make a request to <http://localhost:8081/flights>.
13. Step through the flow and watch the value of the payload change.

The screenshot shows the 'Mule Debugger View' window. The 'payload' row is selected in the table, showing its type as 'java.lang.String'. The payload content is displayed below, showing an HTML document structure.

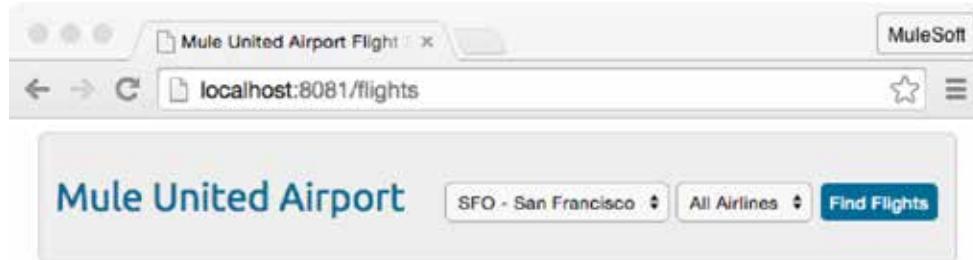
Name	Value	Type
Message		org.mule.DefaultMuleMessage
Message Processor	Logger	org.mule.api.processor.LoggerMess...
payload	<!DOCTYPE html PUBLIC "-//W3C//D...>	java.lang.String

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
    <title>Mule United Airport Flight System</title>
    <style>
        @import url(@import url(http://fonts.googleapis.com/css?family=Ubuntu+Roboto:400,300););
    </style>
</head>
<body>
    width: 100%;
    background-color: #fff;
    font-weight: 300;
    font-family: 'Roboto', sans-serif;
</body>
```



14. Step through the rest of the flow and then return to the browser window; you should see the HTML form.



15. Click the Find Flights button; what happens?

## Review the HTML form code

16. Open FlightFinder.html in Anypoint Studio.

17. Locate the form at the bottom and find the names of the select boxes and their option values.

```
181<form id="myForm" name="myForm">
182  <select id="destination" name="destination" class="select1">
183    <option value="SFO">SFO - San Francisco</option>
184    <option value="LAX">LAX - Los Angeles</option>
185    <option value="CLE">CLE - Cleveland</option>
186    <option value="PDX">PDX - Portland</option>
187    <option value="PDF">PDF - Adobe</option>
188    <option value="FOO">FOO - Mars</option>
189  </select>
190  <select id="airline" name="airline" class="select2">
191    <option value="all">All Airlines</option>
192    <option value="united">United</option>
193    <option value="delta">Delta</option>
194    <option value="american">American</option>
195  </select>
196  <input class="button" name="submit" type="button" value="Find Flights"
197  onClick="return ajaxFunction();">
198</form>
```

18. Locate to where the form data is posted.

```
169
170      ajaxRequest.open("POST", "/flights", true);
171      ajaxRequest.setRequestHeader("Content-type", "application/json");
172      ajaxRequest.send(formData);
```

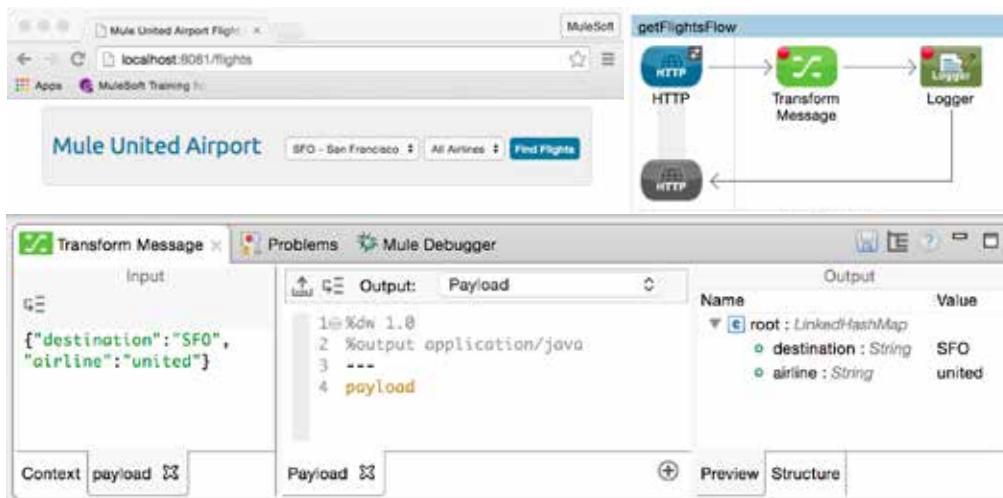
19. Locate in what format the data is that is posted.

```
98  function ajaxFunction() {
99    var destinationMenu = document.getElementById("destination");
100   var airlineMenu = document.getElementById("airline");
101
102   var jsonObject = {
103     "destination" : destinationMenu.options[destinationMenu.selectedIndex].value,
104     "airline" : airlineMenu.options[airlineMenu.selectedIndex].value
105   };
106
107   var formData = JSON.stringify(jsonObject);
```

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

In this walkthrough, you will work with the data posted from the HTML form. You will:

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



### Create a new flow

1. Return to ap essentials.xml.
2. Drag out another HTTP connector to create a new flow and name it getFlightsFlow
3. In the HTTP Properties view, set the connector configuration to the existing HTTP\_Listener\_Configuration.
4. Set the path to /flights (the same as the form flow) and set the allowed methods to POST.

### Look at the data posted from the form

5. Add a Logger to the flow and add a breakpoint to it.
6. Save the file to redeploy the application in debug mode.
7. Make another request to <http://localhost:8081/flights>.
8. Step through or remove the breakpoints in the getFlightsFormFlow to get to the form in the browser window.
9. Click the Find Flights button.

10. Look at the Mule Debugger view; you should see the return data is a BufferInputStream.
11. Look at the inbound properties; you should see the content-type is set to application/json.

The screenshot shows the Mule Studio interface with the 'Mule Debugger' tab selected. It displays two tables: 'Inbound Properties' and 'Outbound Properties'.

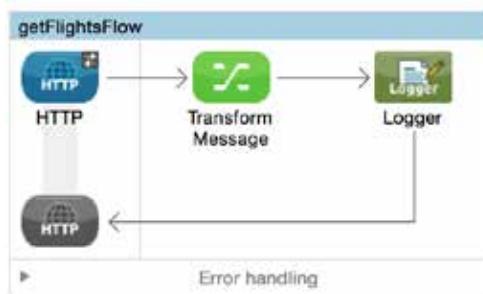
Name	Type
Message	org.mule.DefaultMuleMessage
Message Processor	org.mule.api.processor.LoggerMessageProcessor
payload	org.glassfish.grizzly.util.BufferInputStream

Name	Type
accept-encoding	java.lang.String
accept-language	java.lang.String
connection	java.lang.String
content-length	java.lang.String
content-type	application/json
host	java.lang.String
http.listener.path	/flights

## Add a DataWeave Transform Message component

12. Add a DataWeave Transform Message component before the Logger.



13. In the Transform Message Properties view, look at the Input, Transform, and Output sections.
14. In the Transform section, locate the drop-down menu at the top of the view that sets the output type; it should be set to Payload.
15. Beneath it, locate the directive that sets the output to application/java.
16. Delete the existing DataWeave expression (the curly braces) and set it to payload.

The screenshot shows the 'Transform Message' properties view. The 'Output' section is selected, showing 'Payload' as the type. The 'Transform' section contains the following DataWeave code:

```

1@%dw 1.0
2 %output application/java
3 ---
4 payload
  
```

## Debug the application

17. Add a breakpoint to the Transform Message component (if it does not have one).
18. Save the file to redeploy the application in debug mode.
19. Make a request to <http://localhost:8081/flights> and submit the form again.

20. Step to the Logger in the getFlightsFlow; you should see the form data is now a LinkedHashMap.

The screenshot shows the Mule Debugger interface with the 'Transform Message' tab selected. The 'payload' entry under 'Message Processor' is expanded, showing two entries: '0' and '1'. The value for '0' is 'destination=SFO' and for '1' is 'airline=all'. A preview window below shows the JSON representation: '{destination=SFO, airline=all}'.

Name	Value	Type
Message		org.mule.DefaultMuleMessage
Message Processor	Logger	org.mule.api.processor.LoggerMe...
payload	{destination=SFO, airline=all}	java.util.LinkedHashMap
0	destination=SFO	java.util.LinkedHashMap\$Entry
1	airline=all	java.util.LinkedHashMap\$Entry

```
{destination=SFO, airline=all}
```

21. Click the Resume button.

## Add sample data and preview

22. In the Input section of the Transform Message Properties view, click Payload: Unknown.  
23. Click the Sample Data button.

The screenshot shows the 'Transform Message' properties view. The 'Input' section has a dropdown menu open, with 'Payload : Unknown' selected. Below the dropdown, there are buttons for 'Sample Data' and 'Inbound Properties'.

24. In the Sample data format dialog box, select JSON and click OK.

The screenshot shows a 'Sample data format' dialog box. It has a dropdown menu labeled 'Select the format of your sample data' with 'JSON' selected. At the bottom are 'Cancel' and 'OK' buttons.

25. In the new payload window that is created in the Input section, replace the sample JSON data with {"destination":"SFO", "airline":"united"}.

*Note: You can copy this value from the course snippets.txt file.*

26. Click the Preview tab in the Output section; you should see sample output of type Java.

The screenshot shows the Mule Studio interface with the 'Transform Message' component selected. The 'Input' pane contains the JSON payload: {"destination": "SFO", "airline": "united"}. The 'Output' pane shows the mapping steps: 1 %dw 1.0, 2 %output application/java, 3 ---, 4 payload. The 'Payload' pane displays the resulting Java object: root : LinkedHashMap, destination : String SFO, airline : String united. The 'Preview' tab is selected at the bottom.

## Change the output type

27. In the Transform section, change the output type from application/java to application/xml.

28. Look at the Preview tab; you should get an error.

The screenshot shows the Mule Studio interface with the 'Transform Message' component selected. The 'Input' pane contains the JSON payload: {"destination": "SFO", "airline": "united"}. The 'Output' pane shows the mapping steps: 1 %dw 1.0, 2 %output application/xml, 3 ---, 4 payload. The 'Payload' pane displays the resulting XML output: { "destination": "SFO", "airline": "united" }. A red error message at the bottom states: "Can not update preview. Validate your mappings." The 'Preview' tab is selected at the bottom.

29. Change the output type to application/json; in the Preview tab, you should see sample JSON output.

The screenshot shows the Mule Studio interface with the 'Transform Message' component selected. The 'Input' pane contains the JSON payload: {"destination": "SFO", "airline": "united"}. The 'Output' pane shows the mapping steps: 1 %dw 1.0, 2 %output application/json, 3 ---, 4 payload. The 'Payload' pane displays the resulting JSON output: { "destination": "SFO", "airline": "united" }. The 'Preview' tab is selected at the bottom.

## Debug the application

15. Save the file to redeploy the application in debug mode.

16. Make a request to <http://localhost:8081/flights> and submit the form again.

17. Step to the Logger in the getFlightsFlow; you should see the form data is now of type WeaveOutputHandler.

Name	Value	Type
Message	org.mule.DefaultMuleMessage	
Message Processor	Logger	org.mule.api.processor.LoggerMessageProcessor
payload	com.mulesoft.weave.mule.WeaveMessage... com.mulesoft.weave.mule.WeaveMessageProcessor.WeaveOutputHandler	com.mulesoft.weave.mule.WeaveOutputHandler
encodingName	UTF-8	java.lang.String
engine	com.mulesoft.weave.engine.Engine@fb32ca7	com.mulesoft.weave.engine.Engine
outputModule	com.mulesoft.weave.module.JsonModul...	com.mulesoft.weave.module.JsonModule
readers	Map(payload -> com.mulesoft.weave.rea...)	scala.collection.immutable.Map.Map1
variableContext	Map(lookup -> com.mulesoft.weave.mul...)	scala.collection.immutable.HashMap.HashTrieMap

18. Click the Resume button.

19. Stop the Mule runtime.

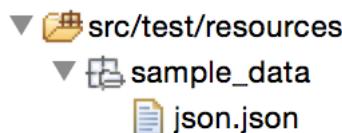
## Change the output type

20. In the Transform section, change the output type from application/json back to application/java.  
 21. Save the file.

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

## Examine the code

22. In the Package Explorer, locate the sample data in src/test/resources.

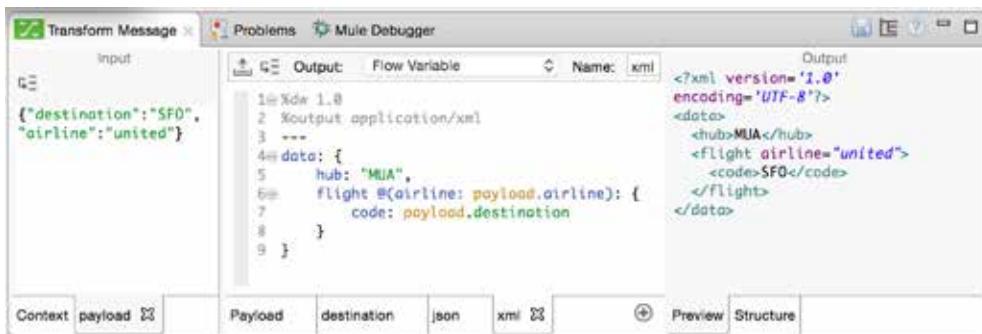


23. Switch to the Configuration XML view.  
 24. Locate the DataWeave code.  
 25. Switch back to the Message Flow view.

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

In this walkthrough, you will continue to work with the data posted from the HTML form. You will:

- Write transformations to store data in multiple outputs as different types of data.
- Create a second transformation to store the destination in a flow variable.
- Create a third transformation to output data as JSON.
- Create a fourth transformation to output data as XML.



### Create a second transformation to store destination in a flow variable

1. Return to getFlightsFlow in apessentials.xml.
2. Return to the Properties view for the Transform Message component.
3. In the Transform section, click the Add New Transformation button in the lower-right corner.



4. In the new myVar tabbed window that appears in the Transform section, make sure the Output is set to Flow Variable.
5. Set the name to destination.
6. Set the DataWeave expression to payload.destination.

- Look at the preview in the Output section; you should see the string SFO.

Name	Value
root : String	SFO

## Debug the application

- Save the file and debug the application.
- Make a request to <http://localhost:8081/flights>, select LAX and Delta, and submit the form again.
- Step to the Logger and click the Variables tab; you should see now see your flow variable in addition to the transformed payload.

Name	Value	Type
destination	LAX	java.lang.String

- Click the Resume button.

*Note: You will continue working with this flow in a later walkthrough. You will add functionality to get the requested flight data from the other flows and return it back to the form.*

## Create a third transformation to output data as JSON

- In the Transform Message Properties view, click the Add New Transformation button.
- In the new window, leave the Output set to Flow Variable.
- Set the name to json.
- Set the DataWeave expression to payload.
- Change the output type to application/json.

17. Look at the preview in the Output section.

The screenshot shows the Mule Studio interface with a 'Transform Message' component selected. The 'Input' tab displays the JSON payload: {"destination": "SFO", "airline": "united"}. The 'Output' tab shows the DataWeave expression: %dw 1.0 %output application/json payload. The resulting 'Output' is a JSON object with 'destination' and 'airline' fields. Below the tabs are buttons for Context, payload, Payload, destination, json, Preview, and Structure.

```
%dw 1.0
%output application/json
payload
```

```
{
  "destination": "SFO",
  "airline": "united"
}
```

18. Change the DataWeave expression to data: payload.

The screenshot shows the Mule Studio interface with a 'Transform Message' component selected. The 'Input' tab displays the same JSON payload. The 'Output' tab shows the DataWeave expression: %dw 1.0 %output application/json data: payload. The resulting 'Output' is a JSON object with a 'data' field containing the original payload. Below the tabs are buttons for Context, payload, Payload, destination, json, Preview, and Structure.

```
%dw 1.0
%output application/json
data: payload
```

```
{
  "data": {
    "destination": "SFO",
    "airline": "united"
  }
}
```

19. Change the DataWeave expression to data: {}.

The screenshot shows the Mule Studio interface with a 'Transform Message' component selected. The 'Input' tab displays the same JSON payload. The 'Output' tab shows the DataWeave expression: %dw 1.0 %output application/json data: {}. The resulting 'Output' is a JSON object with a 'data' field that is an empty object. Below the tabs are buttons for Context, payload, Payload, destination, json, Preview, and Structure.

```
%dw 1.0
%output application/json
data: {}
```

```
{
  "data": {}
}
```

20. Add a field called hub and set it to "MUA".

The screenshot shows the Mule Studio interface with a 'Transform Message' component selected. The 'Input' tab displays the same JSON payload. The 'Output' tab shows the DataWeave expression: %dw 1.0 %output application/json data: {hub: "MUA"}. The resulting 'Output' is a JSON object with a 'data' field containing a sub-object with a 'hub' key set to 'MUA'. Below the tabs are buttons for Context, payload, Payload, destination, json, Preview, and Structure.

```
%dw 1.0
%output application/json
data: {hub: "MUA"}
```

```
{
  "data": {
    "hub": "MUA"
  }
}
```

21. Add a field called code and set it to the destination property of the payload.

22. Add a field called airline and set it to the airline property of the payload.

The screenshot shows the Mule Studio interface with the 'Transform Message' component selected. The 'Input' tab displays the JSON payload: `{"destination": "SFO", "airline": "united"}`. The 'Output' tab shows the DataWeave expression: `%dw 1.0 output application/json`  
---  
data: {  
 hub: "MUA",  
 code: payload.destination,  
 airline: payload.airline  
}. The 'Output' panel on the right shows the resulting JSON object: `{ "data": { "hub": "MUA", "code": "SFO", "airline": "united" } }`. Below the tabs are buttons for Context, payload, Payload, destination, json, Preview, and Structure.

## Create a fourth transformation to output data as XML

23. Click the Add New Transformation button.

24. In the new window, leave the Output set to Flow Variable.

25. Set the name to xml.

26. Set the DataWeave expression to payload.

27. Change the output type to application/xml; you should get an error message in the preview tab.

The screenshot shows the 'Transform Message' component editor. The 'Input' tab has the same JSON payload as before. The 'Output' tab shows the DataWeave expression: `%dw 1.0 output application/xml`  
---  
payload. The 'Output' panel on the right shows a table with a single row: `root : LinkedHashMap  
destination : String SFO  
airline : String united`. A red error message at the bottom says: `Can not update previe...lilate your mappings.`. Below the tabs are buttons for Context, payload, Payload, destination, json, xml, Preview, and Structure.

28. Change the DataWeave expression to data: payload.

The screenshot shows the 'Transform Message' component editor. The 'Input' tab has the same JSON payload. The 'Output' tab shows the DataWeave expression: `%dw 1.0 output application/xml`  
---  
data: payload. The 'Output' panel on the right shows the generated XML: `<?xml version='1.0' encoding='UTF-8'?><data><destination>SFO</destination><airline>united</airline></data>`. Below the tabs are buttons for Context, payload, Payload, destination, json, xml, Preview, and Structure.

29. Click the json tab and copy the DataWeave expression.

30. Click the xml tab and replace the DataWeave expression with the one you just copied.
31. Modify the expression so the code and airline properties are child elements of a new element called flight.

```

Transform Message X Problems Mule Debugger
Input: {"destination":"SFO", "airline":"united"} Output: Flow Variable Name: xml
1 %dw 1.0
2 %output application/xml
3 ---
4 @data: {
5   hub: "MUA",
6   flight: {
7     code: payload.destination,
8     airline: payload.airline
9   }
10 }

Context payload X Payload destination json xml X Preview Structure

```

The screenshot shows the Mule Studio interface with a Transform Message component. The input is a JSON object with fields 'destination' and 'airline'. The output is set to 'Flow Variable' with the name 'xml'. The DataWeave code uses the %dw 1.0 syntax to define a variable 'data' containing a hub and a flight object. The flight object has a 'code' field set to 'payload.destination' and an 'airline' field set to 'payload.airline'. The XML output is generated based on this structure, with the flight element having 'code' and 'airline' as children.

32. Modify the expression so the airline is an attribute of the flight element.

```

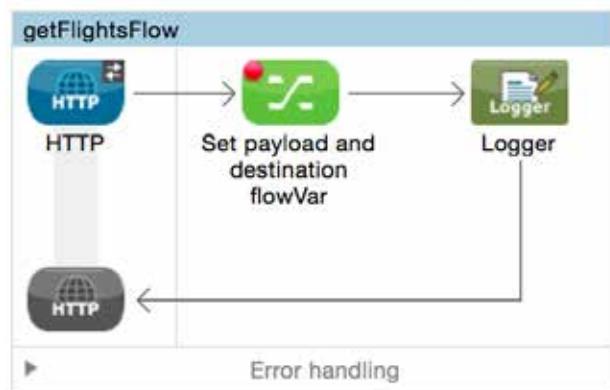
Transform Message X Problems Mule Debugger
Input: {"destination":"SFO", "airline":"united"} Output: Flow Variable Name: xml
1 %dw 1.0
2 %output application/xml
3 ---
4 @data: {
5   hub: "MUA",
6   flight @{airline: payload.airline}: {
7     code: payload.destination
8   }
9 }

Context payload X Payload destination json xml X Preview Structure

```

This screenshot shows the same Transform Message component as above, but with a modified DataWeave expression. The 'flight' element now has an attribute '@{airline: payload.airline}' instead of a separate 'airline' child element. The XML output reflects this change, where the 'flight' element contains the 'code' child and an 'airline' attribute.

33. Change the name of the component to Set payload and destination flowVar.



## Debug the application

34. Save the file to redeploy the application in debug mode.
35. Make a request to <http://localhost:8081/flights> and submit the form again.
36. Step to the Logger and click the Variables tab; you should now see three flow variables.

The screenshot shows the 'Variables' tab of the Mule Studio debugger. It lists three variables:

Name	Value	Type
destination	LAX	java.lang.String
json	{ "data": { "hub": "MUA", "code": "LAX", "airline": "delta" } }	java.lang.String
xml	<?xml version='1.0' encoding='UTF-8'?><flight><destination>LAX</destination><airlines><airline>delta</airline></airlines></flight>	java.lang.String

The 'json' variable's value is expanded below the table, showing a JSON object with a 'data' key containing hub, code, and airline information.

37. Stop the Debugger.

*Note: The json and xml flow variables were created for learning purposes. If you want, you can delete them from the DataWeave transformer.*

## Walkthrough 5-4: Transform complex data structures

In this walkthrough, you will work with static flight data not retrieved using a connector. You will:

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

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



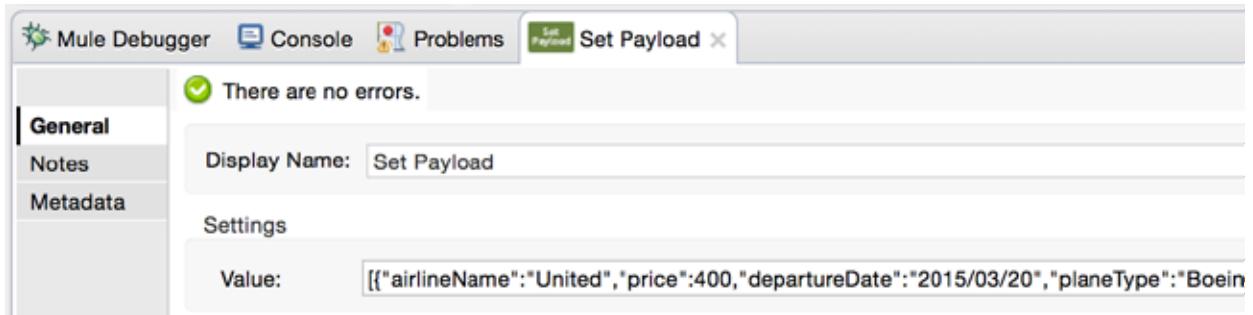
### Create a new flow

1. Return to ap essentials.xml.
2. Drag out another HTTP connector to create a new flow and name it transformStaticFlightsFlow
3. In the HTTP Properties view, set the connector configuration to the existing `HTTP_Listener_Configuration`.
4. Set the path to `/static` and set the allowed methods to GET.

### Add static JSON payload data

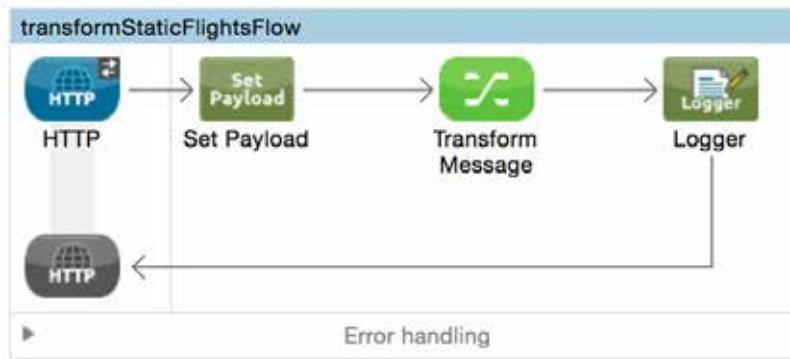
5. Add a Set Payload transformer to the flow.
6. Return to the course snippets.txt file and copy the value for the Example flights JSON.

7. In the Set Payload Properties view, paste the copied value into the value field.



## Transform the payload to Java

8. Add a DataWeave Transform Message component and a Logger to the flow.



9. In the Transform Message Properties view, set the DataWeave expression to payload.
10. Look at the Preview tab in the Output section; you should not see any preview of the data.

## Add sample data

11. In the Input section, click Payload: Unknown.
12. Click the Sample Data button.
13. In the Sample data format dialog box, select JSON and click OK.
14. In the new payload tab, replace the sample JSON with the JSON you used in the Set Payload transformer.

15. Look at the Preview tab in the Output section; you should now see a preview of the data and there should be three LinkedHashMap objects.

The screenshot shows the Mule Studio interface with the 'Transform Message' component selected. The 'Output' tab is active, displaying the payload structure. The payload is an ArrayList containing three LinkedHashMap objects. Each LinkedHashMap has properties: airlineName, price, departureDate, planeType, origination, flightCode, availableSeats, and destination. The first object has an airlineName of 'United', a price of 400, a departureDate of '2015/03/20', a planeType of 'Boeing 737', an origination of 'MUA', a flightCode of 'ER38sd', availableSeats of 0, and a destination of 'SFO'. The second and third objects have similar but slightly different values.

Name	Type	Value
root	ArrayList	[0] : LinkedHashMap airlineName : S United price : Integer 400 departureDate : 2015/03/20 planeType : Str Boeing 737 origination : Str MUA flightCode : Str ER38sd availableSeats : 0 destination : Str SFO
[1]	LinkedHashMap	
[2]	LinkedHashMap	

## Debug the application

16. Add a breakpoint to the Transform Message component.
17. Save the file and debug the application.
18. Make a request to <http://localhost:8081/static>.
19. Step to the Transform Message component; you should see the payload is of type String.

The screenshot shows the Mule Properties view with the 'Message Processor' set to 'Transform Message'. The 'payload' entry shows the value as a JSON string: [{"airlineName": "United", "price": 400, "departur...}]. The 'Type' column indicates it is a 'java.lang.String'.

Name	Value	Type
Message		org.mule.DefaultMuleMessage
Message Processor	Transform Message	com.mulesoft.weave.mule.Weave...
payload	[{"airlineName": "United", "price": 400, "departur...}]	java.lang.String

20. Step to the Logger; you should see the payload is still of type String.

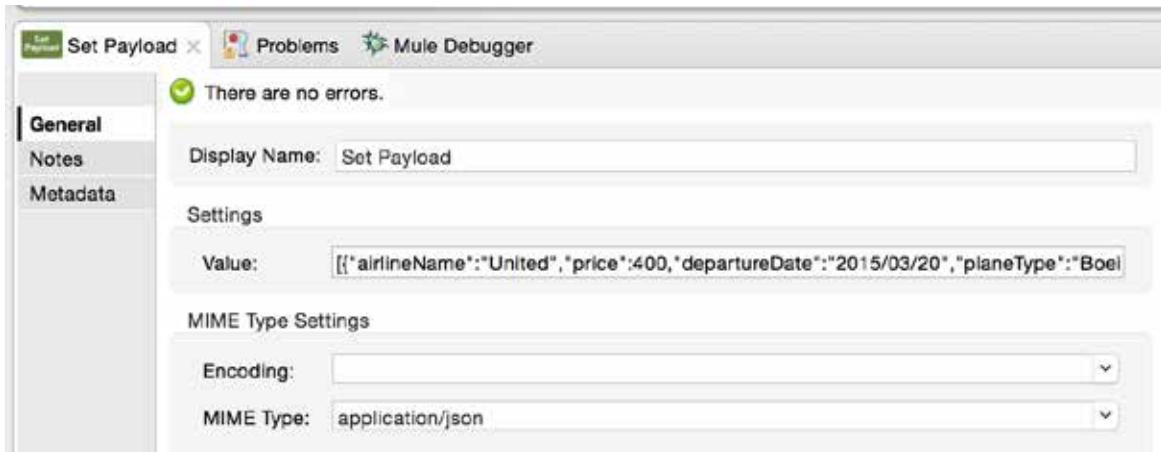
The screenshot shows the Mule Properties view with the 'Message Processor' set to 'Logger'. The 'payload' entry shows the same JSON string as before. The 'Type' column indicates it is a 'java.lang.String'.

Name	Value	Type
Message		org.mule.DefaultMuleMessage
Message Processor	Logger	org.mule.api.processor.LoggerMe...
payload	[{"airlineName": "United", "price": 400, "departur...}]	java.lang.String

21. Click the Resume button.

## Set the payload MIME type

22. In the Set Payload Properties view, set the MIME type to application/json.



23. Apply the changes and wait for the application to redeploy.

24. Make a request to <http://localhost:8081/static>.

25. Step to the Logger; the payload should now be of type ArrayList.

Name	Value	Type
► [e] Message		org.mule.DefaultMuleMessage
③ Message Processor	Logger	org.mule.api.processor.LoggerMe...
▼ [e] payload	[{"airlineName": "United", "price": 400, ...}	java.util.ArrayList
► [e] 0	{"airlineName": "United", "price": 400, ...}	java.util.LinkedHashMap
► [e] 1	{"airlineName": "United", "price": 945, ...}	java.util.LinkedHashMap
▼ [e] 2	{"airlineName": "United", "price": 954, ...}	java.util.LinkedHashMap
► [e] 0	airlineName=United	java.util.LinkedHashMap\$Entry
► [e] 1	price=954	java.util.LinkedHashMap\$Entry
► [e] 2	departureDate=2015/02/12	java.util.LinkedHashMap\$Entry
► [e] 3	planeType=Boeing 777	java.util.LinkedHashMap\$Entry
► [e] 4	origin=MUA	java.util.LinkedHashMap\$Entry
► [e] 5	code=ER39rj	java.util.LinkedHashMap\$Entry
► [e] 6	emptySeats=23	java.util.LinkedHashMap\$Entry
► [e] 7	destination=SFO	java.util.LinkedHashMap\$Entry

26. Click the Resume button; you should get a file download or garbled data depending upon if you are using a browser or some other tool to make your request.

## Return values for the first object in the collection

27. Change the DataWeave expression to payload[0]; in the Preview tab, you should see one LinkedHashMap object.

28. Change the DataWeave expression to payload[0].price; in the Preview tab, you should get 400, the first price value.
29. Change the DataWeave expression to payload[0].\*price; in the Preview tab, you should see an ArrayList with one integer value of 400 – the first price value.

The screenshot shows the Mule Studio interface with a 'Transform Message' component selected. The DataWeave editor on the left contains the following code:

```
%dw 1.0
%output application/java
---
[{"airlineName": "United", "price": 400, "departureDate": "2015/03/20", "planeType": "Boeing 737", "origination": "MUA", "flightCode": "ER38sd", "availableSeats": 0, "destination": "SFO"},
```

The Preview tab on the right shows the output structure:

Name	Value
root : ArrayList	
[0] : Integer	400

## Return collections of values

30. Change the DataWeave expression to return a collection of all the prices.

`payload.*price` or `payload.price`

31. Look at the Preview tab; you should see an ArrayList with three values.

The screenshot shows the Mule Studio interface with a 'Transform Message' component selected. The DataWeave editor on the left contains the following code:

```
%dw 1.0
%output application/java
---
4 payload.price
```

The Preview tab on the right shows the output structure:

Name	Value
root : ArrayList	
[0] : Integer	400
[1] : Integer	945
[2] : Integer	954

32. Change the DataWeave expression to return a collection of prices and available seats.

`[payload.price, payload.availableSeats]`

33. Look at the Preview tab; you should see an ArrayList with two ArrayLists.

The screenshot shows the Mule Studio interface with a 'Transform Message' component selected. The DataWeave editor on the left contains the following code:

```
%dw 1.0
%output application/java
---
4 [payload.*price, payload.availableSeats]
```

The Preview tab on the right shows the output structure:

Name	Value
root : ArrayList	
[0] : ArrayList	<ul style="list-style-type: none"> <li>[0] : Integer 400</li> <li>[1] : Integer 945</li> <li>[2] : Integer 954</li> </ul>
[1] : ArrayList	<ul style="list-style-type: none"> <li>[0] : Integer 0</li> <li>[1] : Integer 54</li> <li>[2] : Integer 23</li> </ul>

## Use the map operator to return object collections with different data structures

34. Change the DataWeave expression to payload map \$, in the Preview tab, you should see three Java objects again.

The screenshot shows the DataWeave editor with the following code:

```
1@%dw 1.0
2 %output application/java
3 ---
4 payload map $
```

The preview table shows the resulting structure:

Name	Value
root : ArrayList	
[0] : LinkedHashMap	
[1] : LinkedHashMap	
[2] : LinkedHashMap	
airlineName : String	United
price : Integer	954
departureDate : String	2015/02/12
planeType : String	Boeing 777
origination : String	MUA
flightCode : String	ER39rj
availableSeats : Integer	23
destination : String	SFO

35. Change the DataWeave expression to set a property called flight for each object that is equal to its index value in the collection.

The screenshot shows the DataWeave editor with the following code:

```
1@%dw 1.0
2 %output application/java
3 ---
4 payload map {
5   flight: $$
6 }
```

The preview table shows the resulting structure:

Name	Value
root : ArrayList	
[0] : LinkedHashMap	0
[1] : LinkedHashMap	1
[2] : LinkedHashMap	2

36. Change the DataWeave expression to create a property called dest for each object that is equal to the value of the destination field in the payload collection.

The screenshot shows the DataWeave editor with the following code:

```
1@%dw 1.0
2 %output application/java
3 ---
4 payload map {
5   flight: $$,
6   dest: $.destination
7 }
```

The preview table shows the resulting structure:

Name	Value
root : ArrayList	
[0] : LinkedHashMap	0
dest : String	SFO
[1] : LinkedHashMap	1
dest : String	SFO
[2] : LinkedHashMap	2
dest : String	SFO

37. Change the DataWeave expression to dynamically add each of the properties present on the payload objects.

The screenshot shows the Mule Studio DataWeave editor. The payload is defined as:

```
1@%dw 1.0
2 %output application/java
3 ---
4@payload map {
5   flight: $$,
6   ($$): $ 
7 }
```

The output pane shows the resulting structure:

Name	Value
root : ArrayList	
[0] : LinkedHashMap	
[1] : LinkedHashMap	
[2] : LinkedHashMap	2
2 : LinkedHashMap	
airlineName : String	United
price : Integer	954
departureDate : String	2015/02/12
planeType : String	Boeing 777
origination : String	MUA
flightCode : String	ER39rj
availableSeats : Integer	23
destination : String	SFO

Below the editor are tabs for Payload, Preview, and Structure.

38. Change the DataWeave expression so the name of each object is flight+index and you get flight0, flight1, and flight2.

The screenshot shows the Mule Studio DataWeave editor. The payload is defined as:

```
1@%dw 1.0
2 %output application/java
3 ---
4@payload map {
5   'flight$$': $ 
6 }
```

The output pane shows the resulting structure:

Name	Value
root : ArrayList	
[0] : LinkedHashMap	
[1] : LinkedHashMap	
[2] : LinkedHashMap	
flight2 : LinkedHashMap	
airlineName : String	United
price : Integer	954
departureDate : String	2015/02/12
planeType : String	Boeing 777
origination : String	MUA
flightCode : String	ER39rj
availableSeats : Integer	23
destination : String	SFO

Below the editor are tabs for Payload, Preview, and Structure.

## Change the output to JSON and test the application

39. Change the DataWeave expression output type from application/java to application/json.  
40. Save the file and run the application.

41. Make a request to <http://localhost:8081/static>; you should see the correct JSON returned.



The screenshot shows a web browser window with the address bar containing "localhost:8081/static". The page content displays a JSON array of flight objects:

```
[{"flight0": {"airlineName": "United", "price": 400, "departureDate": "2015/03/20", "planeType": "Boeing 737", "origination": "MUA", "flightCode": "ER38sd", "availableSeats": 0, "destination": "SFO"}, {"flight1": {"airlineName": "United", "price": 945, "departureDate": "2015/09/11", "planeType": "Boeing 757", "origination": "MUA", "flightCode": "ER39rk", "availableSeats": 54, "destination": "SFO"}, {"flight2": {"airlineName": "United", "price": 954, "departureDate": "2015/02/12", "planeType": "Boeing 777", "origination": "MUA", "flightCode": "ER39rj", "availableSeats": 23, "destination": "SFO"}]
```

## Change the expression to output valid XML and test the application

42. Change the DataWeave expression output type from application/json to application/xml; you should get an error in the Preview tab.
43. Change the DataWeave expression to create a root node called flights; you should still get an error in the Preview tab.
44. Change the expression to map each item in the input array to an XML element.

```
flights: {(payload map {  
    'flight$$':$  
})}
```

45. Save the file and run the application.

46. Make a request to <http://localhost:8081/static>; you should see the XML returned.

The screenshot shows the 'Set Payload' properties dialog. The 'Payload' tab is selected, displaying the Groovy code:

```
1@%dw 1.0
2 %output application/xml
3 ---
4 flights: {${payload map {
5   'flight$': $}}
6 }
7 }
```

The 'Output' tab shows the resulting XML output:

```
<?xml version='1.0' encoding='UTF-8'?>
<flights>
  <flight0>
    <airlineName>United</airlineName>
    <price>400</price>
    <departureDate>2015/03/20</departureDate>
    <planeType>Boeing 737</planeType>
    <origination>MIA</origination>
    <flightCode>ER38sd</flightCode>
    <availableSeats>0</availableSeats>
    <destination>SFO</destination>
  </flight0>
  <flight1>
    <airlineName>United</airlineName>
    <price>945</price>
    <departureDate>2015/09/11</departureDate>
    <planeType>Boeing 757</planeType>
    <origination>MIA</origination>
```

47. Save the file and run the application.

48. Make a request to <http://localhost:8081/static>; you should see the XML returned.

The screenshot shows a web browser window with the URL <http://localhost:8081/static>. The page content is:

This XML file does not appear to have any style information associated with it. The document tree is shown below.

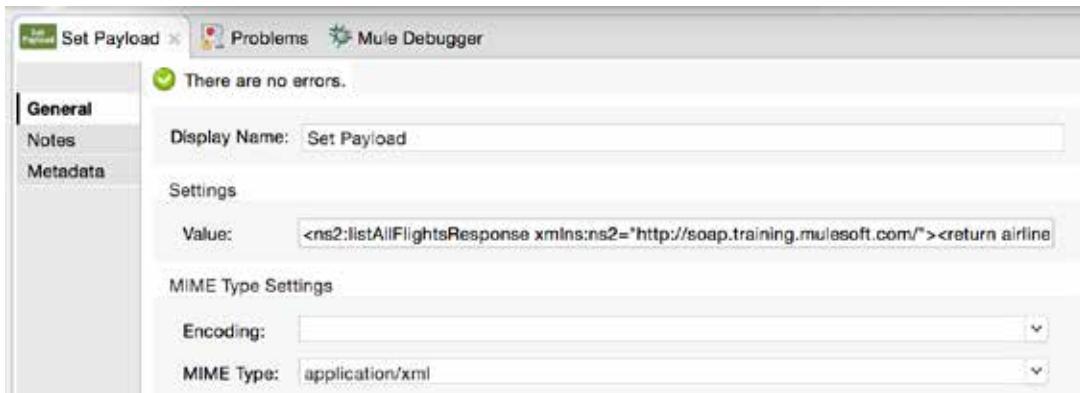
```
▼<flights>
  ▼<flight0>
    <airlineName>United</airlineName>
    <price>400</price>
    <departureDate>2015/03/20</departureDate>
    <planeType>Boeing 737</planeType>
    <origination>MIA</origination>
    <flightCode>ER38sd</flightCode>
    <availableSeats>0</availableSeats>
    <destination>SFO</destination>
  </flight0>
  ▼<flight1>
    <airlineName>United</airlineName>
    <price>945</price>
    <departureDate>2015/09/11</departureDate>
    <planeType>Boeing 757</planeType>
    <origination>MIA</origination>
    <flightCode>ER39rk</flightCode>
    <availableSeats>54</availableSeats>
    <destination>SFO</destination>
  </flight1>
```

## Add static XML payload data

49. Return to the course snippets.txt file and copy the value for the Example flights XML.

50. In the Set Payload Properties view, paste the copied value into the value field.

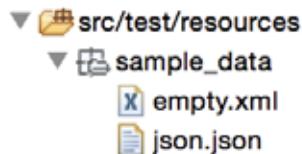
51. Change the MIME type to application/xml.



52. In the Input section of the Transform Message Properties view, click the x on the payload tab to delete the src/test/resources/sample\_data/json\_1.json file.

53. Click Payload: Unknown and then the Sample Data button.

54. In the Sample data format dialog box, select XML and click OK; a new sample data file empty.xml should be created.



55. In the Transform Message Properties view, replace the sample payload data with the Example flights XML you copied.

## Change the return structure of the XML

56. Look at the Preview tab; you should see all the flights are children of the flight0 element.

The screenshot shows the 'Transform Message' properties dialog in Mule Studio. The 'Input' tab displays XML code. The 'Output' tab shows a Groovy script mapping the payload to XML. The 'Preview' tab shows the resulting XML output.

**Input:**

```
<?xml version='1.0' encoding='UTF-8'?>
<ns2:listAllFlightsResponse
  xmlns:ns2="http://
  soap.training.mulesoft.com/"><return
  airlineName="United"><code>A1B2C3</code><departureDate>2015-10-20</departureDate><destination>SFO</destination><emptySeats>40</emptySeats><origin>MUA</origin><planeType>Boing 737</planeType><price>400.0</price></return><return airlineName="Delta"><code>A1B2C4</code><departureDate>2015-10-21</departureDate><destination>LAX</destination></return></listAllFlightsResponse>
```

**Output:**

```
<?xml version='1.0' encoding='UTF-8'?>
<flights>
  <flight0>
    <return airlineName="United">
      <code>A1B2C3</code>
      <departureDate>2015-10-20</departureDate>
      <destination>SFO</destination>
      <emptySeats>40</emptySeats>
      <origin>MUA</origin>
      <planeType>Boing 737</planeType>
      <price>400.0</price>
    </return>
    <return airlineName="Delta">
      <code>A1B2C4</code>
      <departureDate>2015-10-21</departureDate>
      <destination>LAX</destination>
    </return>
  </flight0>
</flights>
```



57. Change the DataWeave expression to loop over the payload.listAllFlightsResponse element; you should see the flights are now correctly transformed.

```

Transform Message
Input
<?xml version='1.0' encoding='UTF-8'?>
<ns2:listAllFlightsResponse
xmlns:ns2="http://
soap.training.mulesoft.com/"><return
airlineName="United"><code>A1B2C3</
code><departureDate>2015-10-20</
departureDate><destination>SFO</
destination><emptySeats>40</
emptySeats><origin>MUA</
origin><planeType>Boing 737</
planeType><price>400.0</price></
return><return
airlineName="Delta"><code>A1B2C4</
code><departureDate>2015-10-21</
departureDate>
Output
<?xml version='1.0'
encoding='UTF-8'?>
<flights>
<flight0>
<code>A1B2C3</code>
<departureDate>2015-10-20</
departureDate>
<destination>SFO</destination>
<emptySeats>40</emptySeats>
<origin>MUA</origin>
<planeType>Boing 737</
planeType>
<price>400.0</price>
</flight0>
<flight1>
<code>A1B2C4</code>
<departureDate>2015-10-21</
departureDate>

```

58. Change the DataWeave expression to loop over the payload.listAllFlightsResponse.return element.

```

Transform Message
Input
<?xml version='1.0'
encoding='UTF-8'?>
<ns2:listAllFlightsResponse
xmlns:ns2="http://
soap.training.mulesoft.com/"><r
eturn
airlineName="United"><code>A1B2
C3</
code><departureDate>2015-10-20<
/
departureDate><destination>SFO</
destination><emptySeats>40</
emptySeats><origin>MUA</
origin><planeType>Boing 737</
planeType>
Output
<?xml version='1.0'
encoding='UTF-8'?>
<flights>
<flight0>A1B2C3</flight0>
<flight1>2015-10-20</flight1>
<flight2>SFO</flight2>
<flight3>40</flight3>
<flight4>MUA</flight4>
<flight5>Boing 737</flight5>
<flight6>400.0</flight6>
</flights>

```

59. Change the DataWeave expression use \* to loop over the XML repeated return elements.

```

Transform Message
Input
<?xml version='1.0'
encoding='UTF-8'?>
<ns2:listAllFlightsResponse
xmlns:ns2="http://
soap.training.mulesoft.com/"><r
eturn
airlineName="United"><code>A1
B2C3</
code><departureDate>2015-10-2
0</
departureDate><destination>SF
O</
destination><emptySeats>40</
emptySeats><origin>MUA</
origin>
Output
<?xml version='1.0' encoding='UTF-8'?>
>
<flights>
<flight0>
<code>A1B2C3</code>
<departureDate>2015-10-20</
departureDate>
<destination>SFO</destination>
<emptySeats>40</emptySeats>
<origin>MUA</origin>
<planeType>Boing 737</planeType>
<price>400.0</price>
</flight0>
<flight1>
<code>A1B2C4</code>
<departureDate>2015-10-21</
departureDate>

```

60. Change the DataWeave expression to return XML with repeated flight elements.

```
flights: {(payload.listAllFlightsResponse.*return map {
    flight: $}
)}
```

61. Change the DataWeave expression to return flight elements with dest and price elements that map to the corresponding destination and price input values.

The screenshot shows the Mule Studio interface with a Transform Message component. The Input section contains an XML message with flight details. The Payload section contains the following DataWeave code:

```
1%> Xdw 1.0
2 %output application/xml
3 ---
4 flights: {(payload.listAllFlightsResponse.*return map {
5     flight: {
6         dest: $.destination,
7         price: $.price
8     }
9 })
10 )
11 }
```

The Output section shows the resulting XML output:

```
<?xml version='1.0' encoding='UTF-8'?>
<flights>
<flight>
<dest>SFO</dest>
<price>400.0</price>
</flight>
<flight>
<dest>LAX</dest>
<price>199.99</price>
</flight>
<flight>
<dest>PDX</dest>
<price>283.0</price>
</flight>
```

## Change the output structure to JSON

62. Change the transform output type from XML to JSON.

The screenshot shows the Mule Studio interface with a Transform Message component. The Input section contains the same XML message as before. The Payload section contains the following DataWeave code:

```
1%> Xdw 1.0
2 %output application/json
3 ---
4 flights: {(payload.listAllFlightsResponse.*return map {
5     flight: {
6         dest: $.destination,
7         price: $.price
8     }
9 })
10 )
11 }
```

The Output section shows the resulting JSON output:

```
{
  "flights": [
    "flight": {
      "dest": "SFO",
      "price": "400.0"
    },
    "flight": {
      "dest": "LAX",
      "price": "199.99"
    },
    "flight": {
      "dest": "PDX",
      "price": "283.0"
    }
  ]
}
```

Note: This is invalid JSON and should result in an error.

63. Remove the {{( and )}} around the payload map expression.

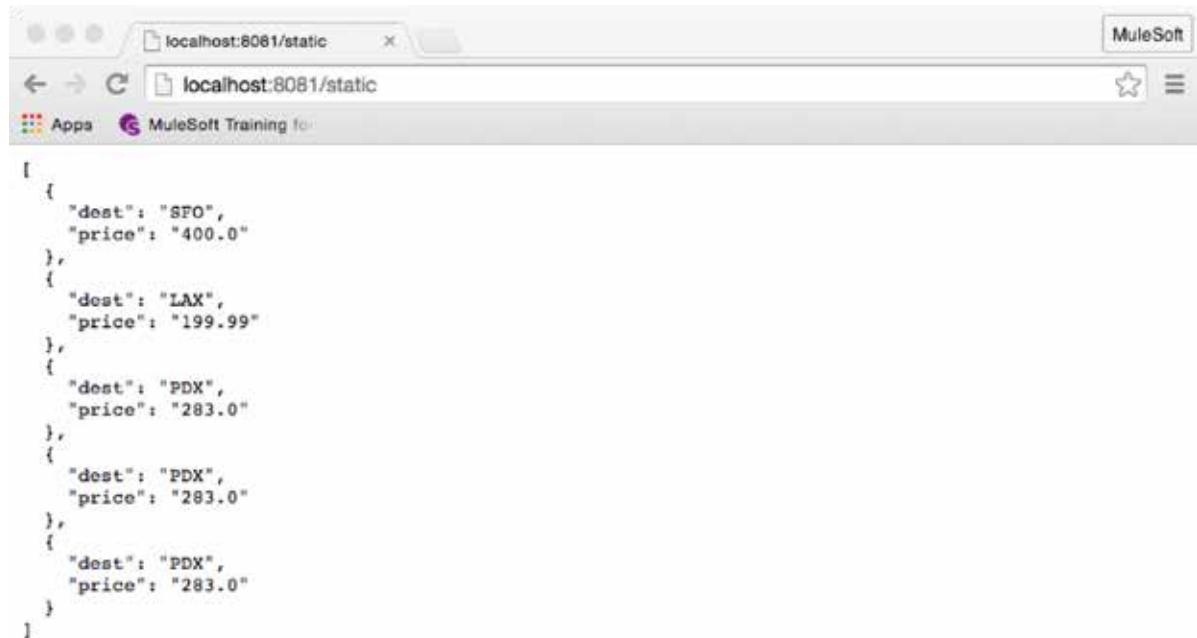
```
flights: payload.listAllFlightsResponse.*return map {
    flight: {
        dest: $.destination,
        price: $.price
    }
}
```

64. Change the DataWeave expression to return an array of objects without the flights and flight properties.

```
payload.listAllFlightsResponse.*return map {
    dest: $.destination,
    price: $.price
}
```

65. Save the file and run the application.

66. Make a request to <http://localhost:8081/static> and look at the JSON structure returned.



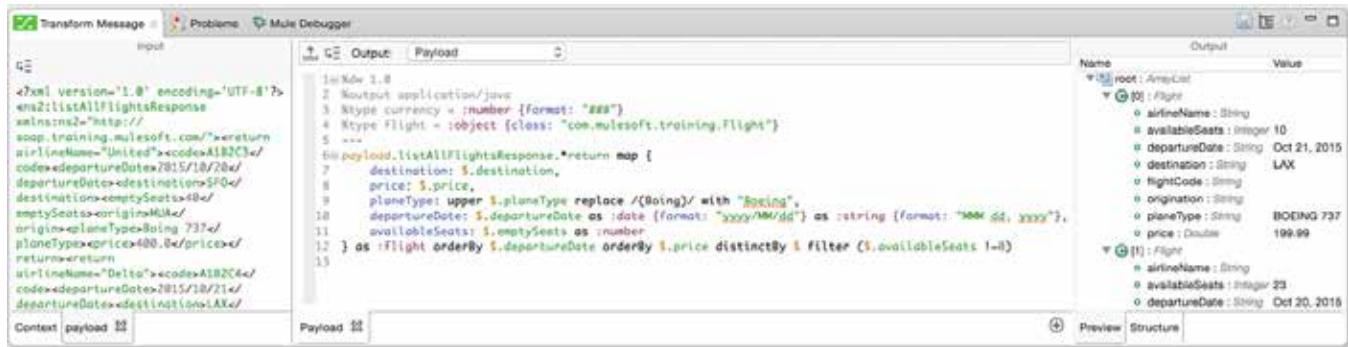
```
[{"dest": "SFO", "price": "400.0"}, {"dest": "LAX", "price": "199.99"}, {"dest": "PDX", "price": "283.0"}, {"dest": "PDX", "price": "283.0"}, {"dest": "PDX", "price": "283.0"}]
```



## Walkthrough 5-5: Use DataWeave operators

In this walkthrough, you will continue to work with the static flight data from the last exercise. You will:

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



```
dw 1.0
%output application/java
number {format: "###"}
object {class: "com.mulesoft.training.Flight"}
...
payload.listAllFlightsResponse.*return map {
    destination: $.destination,
    price: $.price,
    planeType: upper $.planeType replace /Boing/ with "Boeing",
    departureDate: $departureDate as date [format: "yyyy/MM/dd"] as string [format: "MM dd, yyyy"],
    availableSeats: $emptySeats as number
} as :Flight orderBy $.departureDate orderBy $.price distinctBy $.availableSeats +0
}
```

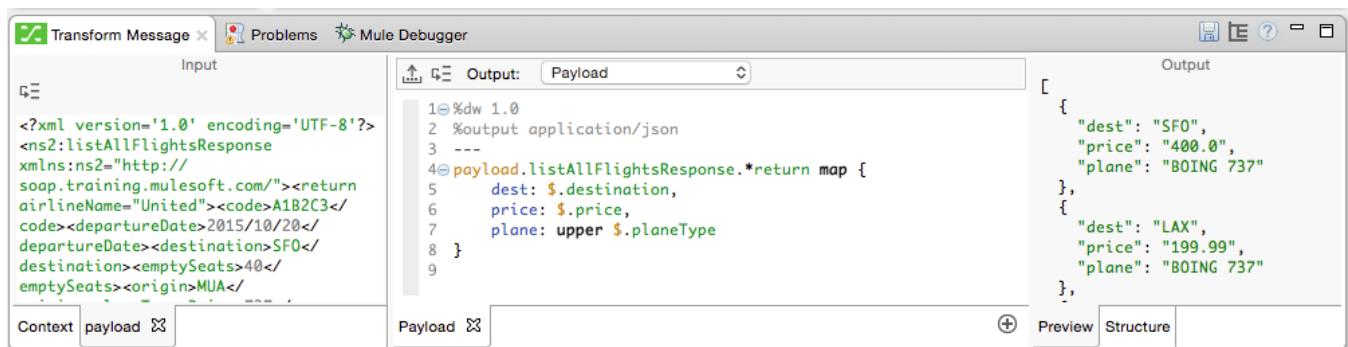
Name	Value
root	ArrayList
[0]	Flight
airlineName	United
availableSeats	10
departureDate	Oct 21, 2015
destination	LAX
flightCode	A1B2C3
origination	MUA
planeType	Boeing
price	199.99
[1]	Flight
airlineName	Delta
availableSeats	23
departureDate	Oct 20, 2015

### Format a string

- Return to transformStaticFlightsFlow in apessentials.xml.
- Change the DataWeave expression to return objects with a property called plane equal to the value of the input planeType values.
- Use the upper operator to return the value in uppercase.

```
plane: upper $.planeType
```

- Look at the Preview tab; you should see the uppercase plane fields.
- Look at the data type of the prices; you should see that they are strings (surrounded by quotation marks).



```
%dw 1.0
%output application/json
---
payload.listAllFlightsResponse.*return map {
    dest: $.destination,
    price: $.price,
    plane: upper $.planeType
}
```

Output
[{"dest": "SFO", "price": "400.0", "plane": "BOING 737"}, {"dest": "LAX", "price": "199.99", "plane": "BOING 737"}]

## Convert data types

6. Change the DataWeave expression to use the as operator to return the prices as numbers (not surrounded by quotation marks) instead of strings.

The screenshot shows the Mule Studio interface with the 'Transform Message' tab selected. The 'Input' pane contains an XML payload representing flight information. The 'Output' pane shows the resulting JSON payload. The DataWeave script used is:

```
%dw 1.0
%output application/json
---
@payload.listAllFlightsResponse.*return map {
    dest: $.destination,
    price: $.price as :number,
    plane: upper $.planeType
}
```

The 'Output' pane displays the JSON structure:

```
[{"dest": "SFO", "price": 400.0, "plane": "BOING 737"}, {"dest": "LAX", "price": 199.99, "plane": "BOING 737"}]
```

7. Add a departureDate field to the return object.

```
payload.listAllFlightsResponse.*return map {
    dest: $.destination,
    price: $.price as :number,
    plane: upper $.planeType
    departureDate: $.departureDate
}
```

8. Change the transform output type from json to java.

```
%output application/java
```

9. Look at the Preview tab; you should see the departureDate property is a String.

The screenshot shows the Mule Studio interface with the 'Transform Message' tab selected. The 'Input' pane contains the same XML payload as before. The 'Output' pane shows the resulting Java structure. The DataWeave script is identical to the previous one, but the output type is specified as 'application/java'. The 'Preview' tab is selected, showing the following structure:

Name	Type	Value
root	ArrayList	
0	LinkedHashMap	<ul style="list-style-type: none"><li>dest: String SFO</li><li>price: Integer 400</li><li>plane: String BOING 737</li><li>departureDate: String 2015/10/20</li></ul>
1	LinkedHashMap	<ul style="list-style-type: none"><li>dest: String LAX</li><li>price: Double 199.99</li></ul>

10. Change the DataWeave expression to use the as operator to convert departureDate to a date object.

11. Look at the Preview tab; you should get an error message.

Name	Value
root : ArrayList	
[0] : LinkedHashMap	
dest : String	SFO
price : Integer	400
plane : String	BOING 737
departureDate : String	2015/10/20
[1] : LinkedHashMap	
dest : String	LAX

Can not update preview. Validate your mappings.

12. Look at the format of the dates in the sample data in the payload tab in the Input section.

13. Use the format operator to specify the pattern of the input date strings.

departureDate: \$.departureDate as :date {format: "yyyy/MM/dd"}

Note: If you are not a Java programmer, you may want to look at the Java documentation for the `DateTimeFormatter` class to review documentation on pattern letters. See:

<https://docs.oracle.com/javase/8/docs/api/java/time/format/DateTimeFormatter.html>.

14. Look at the Preview tab; you should see departureDate is now a Date object.

Name	Value
root : ArrayList	
[0] : LinkedHashMap	
dest : String	SFO
price : Integer	400
plane : String	BOING 737
departureDate : Date	Tue Oct 20 12:00:00 PDT
[1] : LinkedHashMap	
dest : String	LAX
price : Double	199.99

## Format a date

15. Use the as operator to convert the dates to strings, which can then be formatted.

```
departureDate: $.departureDate as :date {format: "yyyy/MM/dd"} as :string
```

16. Look at the Preview tab; the departureDate is again a String.

The screenshot shows the Mule Studio interface with a 'Transform Message' component selected. The 'Input' tab displays an XML message structure. The 'Output' tab shows the transformed payload. The 'Preview' tab on the right displays the resulting data structure, which includes a list of flights with their destination, price, plane type, and now-formatted departure date ('2015/10/20').

Name	Value
root	ArrayList
[0]	LinkedHashMap
dest	SFO
price	400.0
plane	BOING 737
departureDate	2015/10/20
[1]	LinkedHashMap
dest	LAX
price	199.99

17. Use the format operator with any pattern letters and characters to format the date strings.

```
departureDate: $.departureDate as :date {format: "yyyy/MM/dd"}  
as :string {format: "MMM dd, yyyy"}
```

18. Look at the Preview tab; the departureDate properties should now be formatted.

The screenshot shows the Mule Studio interface with the same 'Transform Message' component. The transformation script now includes a format pattern 'MMM dd, yyyy' for the departure date. The output preview shows the departure dates now correctly formatted as 'Oct 20, 2015'.

Name	Value
root	ArrayList
[0]	LinkedHashMap
dest	SFO
price	400.0
plane	BOING 737
departureDate	Oct 20, 2015
[1]	LinkedHashMap
dest	LAX
price	199.99

19. Change the transform output from java to json.

```
%output application/json
```

20. Save the file and run the application.

21. Make a request to <http://localhost:8081/static>; you should see the formatted dates.

The screenshot shows a web browser window displaying a JSON response. The JSON data represents three flight records, each with a 'departureDate' field formatted as 'Oct 20, 2015'.

```
[{"dest": "SFO", "price": "400.0", "plane": "BOING 737", "departureDate": "Oct 20, 2015"}, {"dest": "LAX", "price": "199.99", "plane": "BOING 737", "departureDate": "Oct 21, 2015"}, {"dest": "PDX", "price": "283.0", "plane": "Airbus A320", "departureDate": "Oct 22, 2015"}]
```



## Format a number

22. Use the format operator in the DataWeave expression to format the prices with two decimal places.

```
price: $.price as :number {format: "###.##"},
```

23. Look at the Preview tab; the prices should be formatted.



```
%dw 1.0
output application/json
---
payload.listAllFlightsResponse.*return map {
    dest: $destination,
    price: $.price as :number {format: "###.##"},
    plane: upper $.planeType,
    departureDate: $.departureDate as :date {format: "yyyy/MM/dd"} as :string [f]
}
```

The screenshot shows the Anypoint Studio interface with a Transform Message component. The input is XML describing flight details. The DataWeave script uses the `format` operator to round trip the XML to JSON, specifically formatting the `price` field. The preview tab shows the resulting JSON output, which includes the formatted price values.

*Note: There is a bug in Anypoint Studio 5.2.1 (whose live preview uses Mule runtime 3.7.1) and Mule runtime 3.7.1 and the numbers are not formatted correctly.*

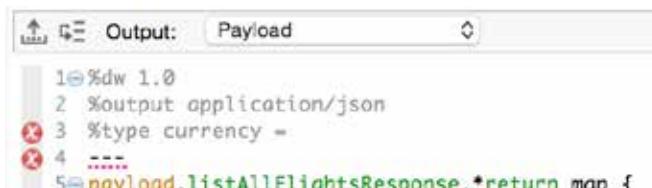
24. Save the file and run the application.  
25. Make a request to <http://localhost:8081/static>; you should see the formatted prices (but may not).  
26. Change the DataWeave expression to format the prices to zero decimal places.

```
price: $.price as :number {format: "###"},
```

27. Look at the Preview tab; you should (but may not) see the second price is now 200 instead of 199.99.

## Define a custom data type

28. In the header section of the Transform section, define a custom data type called currency.



```
%dw 1.0
output application/json
%type currency = :number {format: "###"}
```

29. Select the data coercion and formatting expression for the price in the DataWeave expression and move it to the currency data type definition.

```
%type currency = :number {format: "###"}
```

30. In the DataWeave expression, set the price to be of type currency.

```
price: $.price as :currency,
```

31. Look at the Preview tab; the prices should still be formatted.

*Note: There is a bug in Mule runtime 3.7.1 and the numbers are not formatted correctly.*

## Replace data values

32. Use the replace operator in the DataWeave expression to replace the string Boing with Boeing.

```
plane: upper $.planeType replace /(Boing)/ with "Boeing",
```

33. Look at the Preview tab; Boeing should be spelled correctly.



```
Input
<!-- XML code -->
<Output> application/json
<!-- DW Script -->
1 %dw 1.0
2 %output application/json
3 %type currency = :number {format: "###"}
4 ---
5 @payload.listAllFlightsResponse.*return map {
6   dest: $destination,
7   price: $.price as :currency,
8   plane: upper $.planeType replace /(Boing)/ with "Boeing",
9   departureDate: $.departureDate as :date {format: "yyyy/MM/dd"} as :string {f:
10 }
11 }
```

```
Output
[{"dest": "SFO", "price": 480.0, "plane": "BOEING 737", "departureDate": "Oct 20, 2015"}, {"dest": "LAX", "price": 199.99, "plane": "BOEING 737", "departureDate": "Oct 21, 2015"}, {"dest": "POX", "price": 283.0, "plane": "Airbus A320", "departureDate": "Oct 22, 2015"}]
```

## Order data

34. In the Preview tab, look at the flight prices; the flights should not be ordered by price.

35. Change the DataWeave expression to use the orderBy operator to order the objects by price.

```
payload.listAllFlightsResponse.*return map {
  ...
} orderBy $.price
```



36. Look at the Preview tab or run the application; the flights should now be ordered by price.

```
payload.listAllFlightsResponse.*return map {
    dest: $destination,
    price: $price as :currency,
    plane: upper $.planeType replace /(Boing)/ with "Boeing",
    departureDate: $departureDate as :date {format: "yyyy/MM/dd"} as :string,
    orderBy $.price
}
```

The preview tab shows the following sorted flight list:

```
[{"dest": "LAX", "price": 199.99, "plane": "BOEING 737", "departureDate": "Oct 21, 2015"}, {"dest": "PDX", "price": 283.0, "plane": "BOEING 777", "departureDate": "Oct 21, 2015"}, {"dest": "PDX", "price": 283.0, "plane": "BOEING 777", "departureDate": "Oct 21, 2015"}, {"dest": "SFO", "price": 400.0, "plane": "BOEING 777", "departureDate": "Oct 21, 2015"}]
```

37. Look at the PDX flights; they should not be ordered by departureDate.

38. Change the DataWeave expression to first sort by departureDate and then by price.

```
payload.listAllFlightsResponse.*return map {
    ...
} orderBy $.departureDate orderBy $.price
```

39. Look at the Preview tab or run the application; the flights should now be ordered by price and flights of the same price should be sorted by departureDate.

```
payload.listAllFlightsResponse.*return map {
    dest: $destination,
    price: $price as :currency,
    plane: upper $.planeType replace /(Boing)/ with "Boeing",
    departureDate: $departureDate as :date {format: "yyyy/MM/dd"} as :string,
    orderBy $.departureDate orderBy $.price
}
```

The preview tab shows the following sorted flight list:

```
[{"dest": "PDX", "price": 283.0, "plane": "BOEING 777", "departureDate": "Oct 20, 2015"}, {"dest": "PDX", "price": 283.0, "plane": "BOEING 777", "departureDate": "Oct 21, 2015"}, {"dest": "SFO", "price": 400.0, "plane": "BOEING 777", "departureDate": "Oct 21, 2015"}]
```

## Remove duplicate data

40. Use the distinctBy operator in the DataWeave expression to remove any duplicate objects.

```
payload.listAllFlightsResponse.*return map {
    ...
} orderBy $.departureDate orderBy $.price distinctBy $
```

41. Look at the Preview tab; you should now see only four flights instead of five.

```

<?xml version='1.0' encoding='UTF-8'?>
<ns2:listAllFlightsResponse xmlns:ns2="http://soap.training.mulesoft.com/">
<!--return
airlineName="United"><code>A182C3</code><departureDate>2015/10/28</departureDate><destination>LAX</destination><emptySeats>40</emptySeats><origin>HNL</origin><planeType>Boeing 737</planeType><price>400.0</price></returns><return
airlineName="Delta"><code>A1B2C4</code><departureDate>2015/10/21</departureDate><destination>LAX</destination><emptySeats>10</emptySeats>
    
```

```

1= Node 1.0
2= output application/json
3= XType: currency = :number {format: "###,##0"}
4= ==
5= payload.listAllFlightsResponse.*return map {
6=   dest: $.destination,
7=   price: $.price,
8=   plane: upper $.planeType replace /(@oing)/ with "(@ing)",
9=   departureDate: $.departureDate as :date {format: "yyyy/MM/dd"} as ist
10} orderBy $.departureDate orderBy $.price distinctBy $
11
[
  {
    "dest": "LAX",
    "price": "199.99",
    "plane": "BOEING 737",
    "departureDate": "Oct 21, 2015"
  },
  {
    "dest": "PDX",
    "price": "283.0",
    "plane": "BOEING 777",
    "departureDate": "Oct 28, 2015"
  },
  {
    "dest": "PDX",
    "price": "283.0",
    "plane": "BOEING 777",
    "departureDate": "Oct 21, 2015"
  },
  {
    "dest": "SFO",
    "price": "400.0",
    "plane": "BOEING 737",
    "departureDate": "Oct 28, 2015"
  }
]
    
```

*Note: If you still get five flights, remove the currency type coercion (or replace it with the currency expression) from the price field. This is a bug in Mule runtime 3.7.1.*

42. Add an availableSeats field that is equal to the emptySeats field and coerce it to a number.

availableSeats: \$.emptySeats as :number

43. Look at the Preview tab; you should see five flights again.

## Add Java classes file to the project

44. On your computer, navigate to the java folder in the student files folder for the course:

APEssentials3.7\_studentFiles\_{date}/java.

45. Copy and paste (or drag) the com.mulesoft.training folder into your Anypoint Studio project's src/main/java folder.

46. Open the Flight.java class and look at the names of the properties.

```

package com.mulesoft.training;
import java.util.Comparator;
public class Flight implements java
{
    String flightCode;
    String origination;
    int availableSeats;
    String departureDate;
    String airlineName;
    String destination;
    double price;
    String planeType;
}
    
```



## Transform objects to POJOs

47. Return to transformStaticFlightsFlow in apessentials.xml.

48. Change the transformation output from json to java.

```
%output application/java
```

49. Look at the Preview tab and see that LinkedHashMap objects are created.

The screenshot shows the Mule Studio interface with the 'Transform Message' tab selected. The 'Input' section contains an XML payload representing flight data. The 'Output' section shows the resulting DataWeave expression and its preview. The preview table displays two LinkedHashMap objects under the 'root' key. Each object has five entries: dest (String), price (String), plane (String), departureDate (String), and availableSeats (Integer). The first object corresponds to a flight from LAX to PDX on Oct 21, 2015, with a Boeing 737 and 10 available seats. The second object corresponds to a flight from PDX to BOEING 737 on Oct 20, 2015, with 283.0 available seats.

Name	Type	Value
root	Map<String, LinkedHashMap>	
[0]	LinkedHashMap	<ul style="list-style-type: none"><li>dest : String</li><li>price : String</li><li>plane : String</li><li>departureDate : String</li><li>availableSeats : Integer</li></ul>
[1]	LinkedHashMap	<ul style="list-style-type: none"><li>dest : String</li><li>price : String</li><li>plane : String</li><li>departureDate : String</li><li>availableSeats : Integer</li></ul>
[2]	LinkedHashMap	

50. In the header section of the Transform section, define a custom data type called flight that is a com.mulesoft.training.Flight Java object.

```
%type flight = :object {class: "com.mulesoft.training.Flight"}
```

51. In the DataWeave expression, set the map objects to be of type flight.

```
payload.listAllFlightsResponse.*return map {  
    ...  
} as :flight orderBy $.departureDate orderBy $.price distinctBy $
```

52. In the Transform section, look at the warning you get: Key 'dest' not found.

```
3 %type currency = :number {format: "###"}  
4 %type flight = :object {class: "com.mulesoft.training.Flight"}  
5 ---  
6 payload.listAllFlightsResponse.*return map {  
    7 dest: $.destination,  
    8 price: $.price,  
    9 plane: upper $.planeType replace /(Boing)/ with "Boeing",  
10 departureDate: $.departureDate as :date {format: "yyyy/MM/dd"} as :string  
11 availableSeats: $.emptySeats as :number  
12 } as :flight orderBy $.departureDate orderBy $.price distinctBy $
```

53. Change the name of the dest key to destination.

```
destination: $.destination,
```

54. Change the name of the plane key to planeType.

```
planeType: $.planeType,
```

55. Look at the Preview tab; the objects should now be of type Flight.

The screenshot shows the Mule Studio interface with the 'Transform Message' component selected. The 'Input' tab displays an XML payload with flight data. The 'Output' tab shows a DataWeave script that processes this input. The 'Payload' tab shows the resulting JSON output. To the right, the 'Preview' tab displays the structure of the output, showing an array of 'Flight' objects with properties like airlineName, availableSeats, departureDate, destination, flightCode, origination, planeType, and price. One object in the array has an availableSeats value of 0.

```
1<!Nm 1.0>
2<output application/json>
3<type currency = :number {format: "###"!}>
4<type flight = :object {class: "com.mulesoft.training.Flight"}>
5<!--
6<payload.listAllFlightsResponse.*return map {
7    destination: $destination,
8    price: $price,
9    planeType: upper $.planeType replace /(Boing)/ with "Boeing",
10   departureDate: $.departureDate as :date {format: "yyyy/MM/dd"} as :string {
11     availableSeats: $.emptySeats as :number
12   } as :flight orderBy $.departureDate orderBy $.price distinctBy $>
13
```

Name	Value
root	ArrayList
[0]	Flight
airlineName	MUA
availableSeats	10
departureDate	Oct 21, 2015
destination	LAX
flightCode	C3
origination	Delta
planeType	BOEING 737
price	199.99
[1]	Flight
airlineName	MLA
availableSeats	0
departureDate	Oct 20, 2015

56. Save the file and debug the application.

57. Make a request to <http://localhost:8081/static>.

58. Step to the Logger and drill-down into the payload; the objects should be of type Flight.

The screenshot shows the Mule Properties view with the 'Mule Properties' tab selected. It displays a table of the payload structure. The payload is an ArrayList of five Flight objects. Each Flight object has properties: airlineName (null), availableSeats (40), departureDate (Oct 20, 2015), destination (SFO), flightCode (null), origination (null), planeType (BOEING 737), and price (400.0).

Name	Value	Type
e payload	size = 5	java.util.ArrayList
e 0	com.mulesoft.training.Flight@92...	com.mulesoft.training.Flight
e 1	com.mulesoft.training.Flight@7ff...	com.mulesoft.training.Flight
e 2	com.mulesoft.training.Flight@59...	com.mulesoft.training.Flight
e 3	com.mulesoft.training.Flight@6a...	com.mulesoft.training.Flight
e 4	com.mulesoft.training.Flight@7d...	com.mulesoft.training.Flight
airlineName	null	java.lang.String
availableSeats	40	java.lang.Integer
departureDate	Oct 20, 2015	java.lang.String
destination	SFO	java.lang.String
flightCode	null	java.lang.String
origination	null	java.lang.String
planeType	BOEING 737	java.lang.String
price	400.0	java.lang.Double

59. Stop the debugger.

## Filter data

60. Return to transformStaticFlightsFlow in apessentials.xml.

61. In the Preview tab, look at the values of the availableSeats properties.

62. Add a filter to the DataWeave expression that removes any objects that have availableSeats equal to 0.

```
payload.listAllFlightsResponse.*return map {
```

```
...
```



```
} as :flight orderBy $.departureDate orderBy $.price distinctBy $  
filter ($.availableSeats !=0)
```

63. Look at the Preview tab; you should no longer see the flight that had no seats.

The screenshot shows the Mule Studio interface with a Transform Message component open. The left pane displays the XML payload:

```
<?xml version='1.0'  
encoding='UTF-8'?>  
<ns2:listAllFlightsRe  
sponse  
xmlns:ns2="http://  
soop.training.mulesof  
t.com/"><return  
airlineName="United">  
<code>A182C3</  
code><departureDates>  
01/18/20</  
departureDate><destin  
ation>SF0</  
destination><emptySea  
ts>48</
```

The right pane shows the preview of the transformed data:

Name	Value
root	ArrayList
[0]	Flight
airlineName	String
availableSeats	10
departureDate	Oct 21, 2015
destination	String LAX
flightCode	String
origination	String
planeType	String BOEING 737
price	Double 199.99
[1]	Flight
airlineName	String
availableSeats	23
departureDate	Oct 20, 2015

The preview shows two flight objects, one with 10 available seats and another with 23 available seats, both departing on Oct 21, 2015.

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

In this walkthrough, you will work with the American and Delta flight data. You will:

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



### Add a DataWeave Transform Message component

1. Locate getAmericanFlightsFlow in ap essentials.xml.
2. Delete the Object to String transformer.
3. Replace it with a Transform Message component.



4. Change the name of the Transform Message component to Java to Java.

- Look at the Input section of the Transform Message Properties view; the payload should automatically be set to a List<Map> with correct properties because the Database connector configuration is DataSense enabled by default.

The screenshot shows the Mule Debugger interface with the 'Input' tab selected. The payload is defined as a List<Map>. Each element in the list contains the following attributes:

- airlineName : String
- code1 : String
- code2 : String
- fromAirport : String
- planeType : String
- price : Short
- seatsAvailable : String
- takeOffDate : Date
- toAirport : String

Below the payload, there are sections for 'Flow Variables' and 'Session Variables'. At the bottom, there are tabs for 'Context' and 'payload', with 'payload' being the active tab.

*Note: If the payload is not automatically set to a List<Map>, you probably need to refresh the connector metadata. In the American DB Request Properties view, click the Output tab in the Metadata section and see if the payload is shown to be of type List<Map>. If it is not, click the Refresh Metadata link beneath the metadata. If this does not work, select Anypoint > Preferences / Window > Preferences and navigate to Anypoint Studio > DataSense and make sure Automatic DataSense metadata retrieval is selected and click OK. If that does not work, try restarting Anypoint Studio.*

- In the Transform section, leave the output set to Payload and of type application/java.
- Delete the existing DataWeave expression (the curly braces) and set it to payload.
- Look at the Preview tab in the Output section; you should see the output will be the same data type as the input but it does not have any sample values.

The screenshot shows the Mule Debugger interface with the 'Output' tab selected. The structure tree shows the following hierarchy:

- root : ArrayList
- e [0] : LinkedHashMap
  - airlineName : S ????
  - price : Integer 1
  - seatsAvailable : ????
  - toAirport : Strin ????
  - planeType : Stri ????
  - fromAirport : St ????
  - code2 : String ????
  - code1 : String ????
  - takeOffDate : D Wed Oct 01 12:00:

At the bottom, there are tabs for 'Structure' and 'Preview', with 'Preview' being the active tab.

## Add sample data and preview

9. In the Input section of the Transform Message Properties view, click Payload: List<Map>.
10. Click the Sample Data button.
11. In the new payload tab, replace the “????” with sample values.

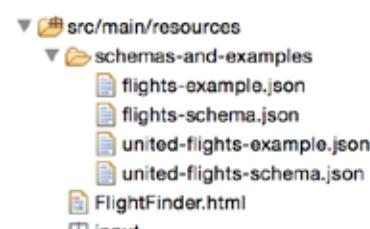
```
%dw 1.0
%output application/java
---
[{
    airlineName: "American",
    price: 450,
    seatsAvailable: "20",
    toAirport: "PDX",
    planeType: "Boeing 747",
    fromAirport: "ORD",
    code2: "pxhjs",
    code1: "5576",
    takeOffDate: "2015-10-01"
}]
```

12. Look at the Preview tab in the Output section; you should see sample values.

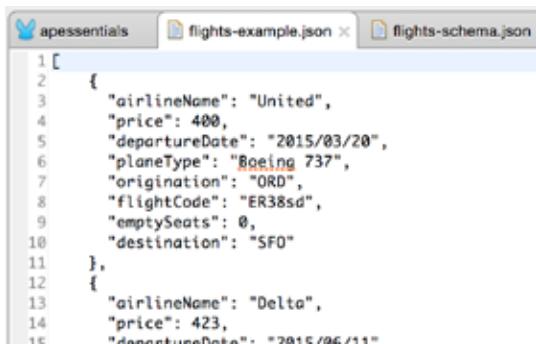
Output	
Name	Value
root : ArrayList	
[0] : LinkedHashMap	
airlineName : String	American
price : Integer	450
seatsAvailable : String	20
toAirport : String	PDX
planeType : String	Boeing 747
fromAirport : String	ORD
code2 : String	pxhjs
code1 : String	5576
takeOffDate : Date	Thu Oct 01 00:00:

## Look at the flights schema and example files

13. On your computer, navigate to the resources folder in the student files folder for the course: APEssentials3.7\_studentFiles\_{date}.
14. Copy and paste (or drag) the schemas-and-examples folder into your Anypoint Studio project's src/main/resources folder.



15. Open the flights-schema.json and flights-example.json files and examine the code.

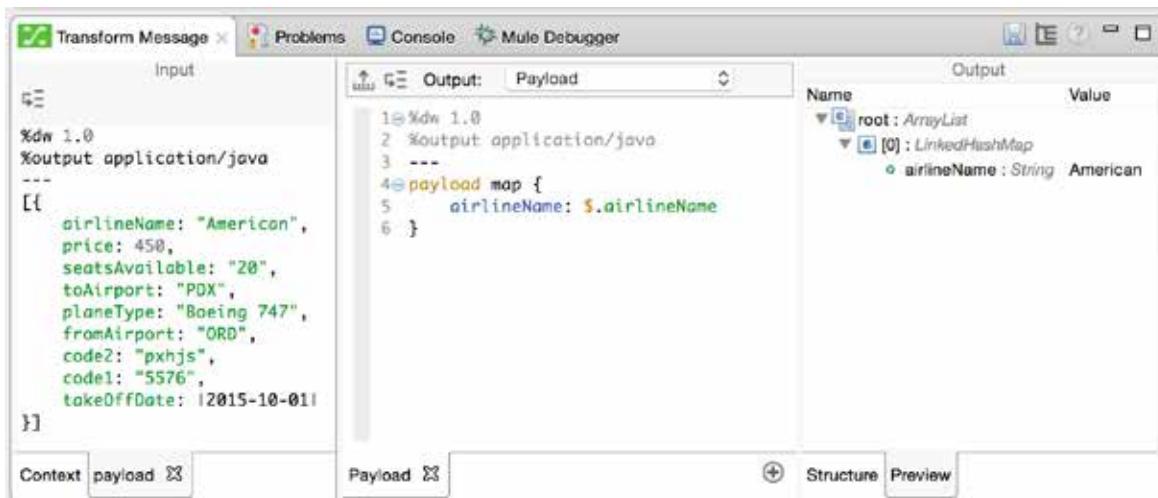


```
1 [
2   {
3     "airlineName": "United",
4     "price": 400,
5     "departureDate": "2015/03/20",
6     "planeType": "Boeing 737",
7     "origination": "ORD",
8     "flightCode": "ER38sd",
9     "emptySeats": 0,
10    "destination": "SFO"
11  },
12  {
13    "airlineName": "Delta",
14    "price": 423,
15    "departureDate": "2015/06/11"
```

## Change the output structure

16. Change the DataWeave expression to payload map { }.

17. Inside the {}, define a new property called airlineName and map it to the payload airlineName property; be sure to use auto-completion!



The screenshot shows the Mule Studio interface with the 'Transform Message' tab selected. On the left, the 'Input' pane displays a JSON array of flight records. In the center, the 'Output' pane shows the DataWeave script:

```
%dw 1.0
%output application/java
---
[{
  airlineName: "American",
  price: 450,
  seatsAvailable: "20",
  toAirport: "PDX",
  planeType: "Boeing 747",
  fromAirport: "ORD",
  code2: "pxhjs",
  code1: "5576",
  takeOffDate: "2015-10-01"
}]
```

The 'Payload' tab is selected in the bottom navigation bar. On the right, the 'Output' pane displays the resulting payload structure:

Name	Value
root : ArrayList	
[0] : LinkedHashMap	
airlineName : String	American

18. Add additional properties with the following mappings:

- departureDate < takeOffDate
- destination < toAirport
- emptySeats < seatsAvailable
- flightCode < code1 concatenated with code2
- origination < fromAirport
- planeType < planeType
- price < price

*Note: You can also copy this code from the course snippets.txt file.*

The screenshot shows the Mule Studio interface with the DataWeave editor open. The payload is defined as:

```

1@%dw 1.0
2 %output application/java
3 ---
4@payload map {
5   airlineName: $.airlineName,
6   departureDate: $.takeOffDate,
7   destination: $.toAirport,
8   emptySeats: $.seatsAvailable,
9   flightCode: $.code1 ++ $.code2,
10  origination: $.fromAirport,
11  planeType: $.planeType,
12  price: $.price
13
14 }

```

The resulting payload structure is displayed in a tree view:

Name	Value
root	ArrayList
[0]	LinkedHashMap
airlineName	S American
departureDate	Thu Oct 01 00:00:00
destination	Str PDX
emptySeats	S 20
flightCode	Str 5576pxhjs
origination	Str ORD
planeType	Str Boeing 747
price	Integer 450

Below the editor, there are tabs for Payload, Structure, and Preview.

## Debug the application

19. Save the file and debug the application.
20. Make a request to <http://localhost:8081/american>.
21. Watch the payload value and step through to the Logger.
22. Drill-down into the data for the first and second flights in the payload and look at the data types and values; be sure to look at emptySeats.

The screenshot shows the Mule Studio interface with the Mule Debugger tab selected. The payload structure is displayed in a table:

Name	Value	Type
Message		org.mule.DefaultMuleMessage
Message Processor	Logger	org.mule.api.processor.Logger...
payload	[[airlineName=American Airlines,...	java.util.ArrayList
0	{airlineName=American Airlines,...	java.util.LinkedHashMap
1	{airlineName=American Airlines,...	java.util.LinkedHashMap
0	airlineName=American Airlines	java.util.LinkedHashMap\$Entry
1	departureDate=Thu Jan 01 00:00:00	java.util.LinkedHashMap\$Entry
2	destination=SFO	java.util.LinkedHashMap\$Entry
3	emptySeats=none	java.util.LinkedHashMap\$Entry
4	flightCode=rree1994	java.util.LinkedHashMap\$Entry
5	origination=MUA	java.util.LinkedHashMap\$Entry
6	planeType=Boeing 777	java.util.LinkedHashMap\$Entry
7	price=676	java.util.LinkedHashMap\$Entry
2	{airlineName=American Airlines,...	java.util.LinkedHashMap

23. Stop the Mule Debugger.

## Format the data

24. In the DataWeave expression, format the price field as a number with the pattern "###.##".

```
price: $.price as :number {format: "###.##"}
```

*Note: You can also define and use a custom data type.*

25. In the DataWeave expression, add logic to set the emptySeats field to a number unless it is equal to "none" in which case set it to 0.

```
emptySeats: $.seatsAvailable as :number unless $.seatsAvailable=="none" otherwise 0,
```

## Debug the application

26. Save the file and debug the application.
27. Make a request to <http://localhost:8081/american>.
28. Watch the payload value and step through to the Logger.
29. Drill-down into the data for the first and second flights in the payload and look at the data types and values and make sure emptySeats and price are set correctly.

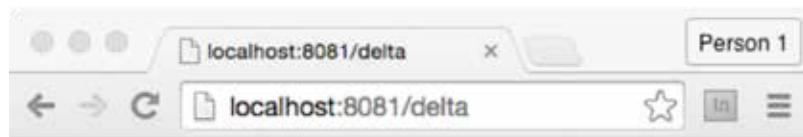
Name	Value	Type
Message		org.mule.DefaultMuleMessage
Message Processor	Logger	org.mule.api.processor.Logger...
payload	[{"airlineName=American Airlines",...}	java.util.ArrayList
0	{airlineName=American Airlines,...}	java.util.LinkedHashMap
1	{airlineName=American Airlines,...}	java.util.LinkedHashMap
0	airlineName=American Airlines	java.util.LinkedHashMap\$Entry
1	departureDate=Thu Jan 01 00:00:00	java.util.LinkedHashMap\$Entry
2	destination=SFO	java.util.LinkedHashMap\$Entry
3	emptySeats=0	java.util.LinkedHashMap\$Entry
4	flightCode=rree1994	java.util.LinkedHashMap\$Entry
5	originination=MIA	java.util.LinkedHashMap\$Entry

30. Stop the Mule Debugger.

## Review the Delta flight data

31. Locate getDeltaFlightsFlow.
32. Run the application and make a request to <http://localhost:8081/delta>.

33. Look at the names of the XML elements and the value of the planeType element.



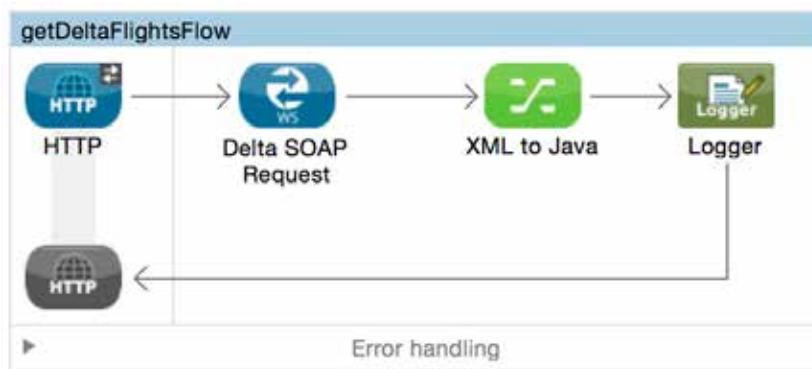
```
<ns2:listAllFlightsResponse
  xmlns:ns2="http://soap.training.mulesoft.com/">
  <return>
    <airlineName>Delta</airlineName>
    <code>A1B2C3</code>
    <departureDate>2015/03/20</departureDate>
    <destination>SFO</destination>
    <emptySeats>40</emptySeats>
    <origin>MUA</origin>
    <planeType>Boing 737</planeType>
    <price>400.0</price>
  </return>
  <return>
    <airlineName>Delta</airlineName>
```

## Review at the flights schema and example files

34. Return to or open the flights-schema.json and flights-example.json files and examine the code.

## Transform the XML to Java

35. Return to getDeltaFlightsFlow.
36. Delete the DOM to XML transformer.
37. Replace it with a Transform Message component.
38. Change the name of the component to XML to Java.



39. Look at the Input section of the XML to Java Properties view; the payload should automatically be set to a `Xml<listAllFlightsResponse>` with correct properties because the Web Service Consumer connector configuration is DataSense enabled by default.

The screenshot shows the 'XML to Java' properties view with the 'Input' tab selected. The 'Payload' section is expanded, showing the structure of a `listAllFlightsResponse` XML element. It contains a `return` list of flight details, each with attributes like `airlineName`, `code`, `departureDate`, `destination`, `emptySeats`, `origin`, `planeType`, and `price`. Below the payload, there are sections for 'Flow Variables' and 'Session Variables', and a 'Context' section at the bottom.

40. In the Transform section, leave the output set to Payload and of type application/java.

41. Delete the existing DataWeave expression (the curly braces) and set it to payload.

42. In the Preview tab, look at the resulting output structure.

The screenshot shows the 'Preview' tab of the XML to Java Properties view. On the left, the 'Payload' section shows the Mule configuration: `%dw 1.0`, `%output application/java`, and the variable `payload`. On the right, the 'Output' section displays the resulting data structure as a table:

Name	Value
<code>root : LinkedHashMap</code>	
<code>listAllFlightsResponse : LinkedHashMap</code>	
<code>return : LinkedHashMap</code>	
<code>airlineName : String</code>	????
<code>code : String</code>	????
<code>departureDate : String</code>	????
<code>destination : String</code>	????
<code>emptySeats : String</code>	1
<code>origin : String</code>	????
<code>planeType : String</code>	????
<code>price : String</code>	2

## Change the output structure

43. Change the DataWeave expression to payload.listAllFlightsResponse and look at the preview.

The screenshot shows the Mule Studio DataWeave editor with the following configuration:

- Output:** Payload
- Payload:** (empty)
- DataWeave Expression:**

```
1@%dw 1.0
2 %output application/java
3 ---
4 payload.listAllFlightsResponse
```
- Output Structure Preview:**

Name	Output	Value
root : LinkedHashMap		
return : LinkedHashMap		
airlineName : String	????	
code : String	????	
departureDate : String	????	
destination : String	????	
emptySeats : String	1	
origin : String	????	
planeType : String	????	
price : String	2	

44. Change the DataWeave expression to payload.listAllFlightsResponse.return and look at the preview.

The screenshot shows the Mule Studio DataWeave editor with the following configuration:

- Output:** Payload
- Payload:** (empty)
- DataWeave Expression:**

```
1@%dw 1.0
2 %output application/java
3 ---
4 payload.listAllFlightsResponse.return
```
- Output Structure Preview:**

Name	Output	Value
root : LinkedHashMap		
airlineName : String	????	
code : String	????	
departureDate : String	????	
destination : String	????	
emptySeats : String	1	
origin : String	????	
planeType : String	????	
price : String	2	

45. Modify the DataWeave expression to used the map operator to populate a new airline property with the value of the airlineName property in each object; you should get a message that the preview cannot be updated.

The screenshot shows the Mule Studio DataWeave editor with the following configuration:

- Output:** Payload
- Payload:** (empty)
- DataWeave Expression:**

```
1@%dw 1.0
2 %output application/java
3 ---
4 payload.listAllFlightsResponse.return map {
5     airlineName: $.airlineName
6 }
```
- Output Structure Preview:**

Name	Output	Value
root : ArrayList		
[0] : LinkedHashMap		
[1] : LinkedHashMap		
[2] : LinkedHashMap		
[3] : LinkedHashMap		
[4] : LinkedHashMap		
[5] : LinkedHashMap		
[6] : LinkedHashMap		
[7] : LinkedHashMap		

A red message at the bottom of the preview panel states: "Can not update preview. Validate your mappings."

46. Change the DataWeave expression to payload.listAllFlightsResponse.\*return.

The screenshot shows the Mule Studio DataWeave editor. The top bar has 'Output: Payload'. The code area contains:

```
1 %dw 1.0
2 %output application/java
3 ---
4 payload.listAllFlightsResponse.*return map {
5     airlineName: $.airlineName
6 }
```

The preview pane shows the output structure:

Name	Value
root : ArrayList	
[0] : LinkedHashMap	
airlineName : \$ ????	

Below the editor are tabs: Payload (selected), Structure, and Preview.

47. Add additional properties with the following mappings:

- departureDate < departureDate
- destination > destination
- emptySeats < emptySeats as a number
- flightCode < code
- origination < origin
- planeType < planeType with the string Boing replaced with Boeing
- price < price and format it as a number with the pattern “###.##”

*Note: You can also copy this code from the course snippets.txt file.*

The screenshot shows the Mule Studio DataWeave editor with the following code:

```
1 %dw 1.0
2 %output application/java
3 ---
4 payload.findFlightResponse.*return map {
5     airlineName: $.airlineName,
6     departureDate: $.departureDate,
7     destination: $.destination,
8     emptySeats: $.emptySeats as :number,
9     flightCode: $.code,
10    origination: $.origin,
11    planeType: $.planeType replace /(Boing)/ with "Boeing",
12    price: $.price as :number {format: "###.##"}
13 }
```

The preview pane shows the output structure, which includes the mapped properties like 'emptySeats' as a number and 'planeType' with the replacement applied.

## Debug the application

48. Save the file and debug the application.
49. Make a request to <http://localhost:8081/delta>.
50. Watch the payload value and step through to the Logger.
51. Drill-down into the data and make sure it all mapped correctly.
52. Stop the Mule Debugger.

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

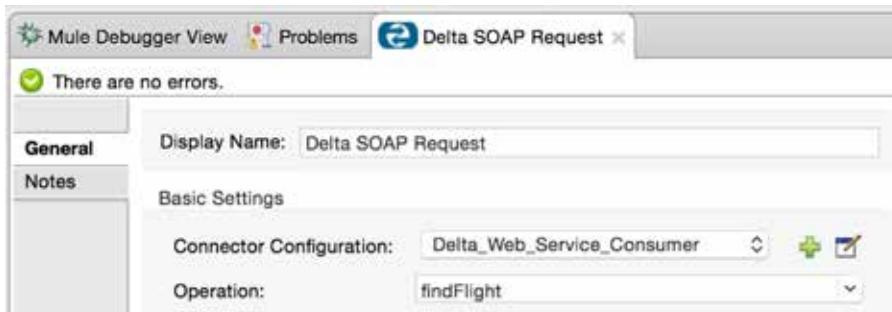
In this walkthrough, you will continue to work with the Delta flight data. You will:

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



### Call a different web service operation

1. Return to getDeltaFlightsFlow.
2. In the Properties view for the Delta SOAP Request endpoint, change the operation to findFlight.



3. In the Transform Message Properties view, change the DataWeave expression to reference findFlightResponse instead of listAllFlightsResponse.

```
payload.findFlightResponse.*return map {
```

### Debug the application

4. Apply the changes and debug the application.
5. Make a request to <http://localhost:8081/delta>.
6. Return to Anypoint Studio and step through the flow; you should get an error.

7. Look at the console; you should see a SoapFaultException because you are calling an operation that requires parameters but you have not passed it any.

```
ERROR 2015-07-08 19:05:22,005 [[apessentials-mod04].HTTP_Listener_Configuration.worker.01] org.mule.exception.DefaultMessagingExceptionStrategy:
*****
Message : No binding operation info while invoking unknown method with params unknown.. Message payload is of type:
NullPayload
Type : org.mule.module.ws.consumer.SoapFaultException
Code : MULE_ERROR--2
Payload : {NullPayload}
JavaDoc : http://www.mulesoft.org/docs/site/current3/apidocs/org/mule/module/ws/consumer/SoapFaultException.html
*****
```

8. Stop the debugger.

## Use DataWeave to pass parameters to the web service

9. Add a Transform Message component to the left of the Delta SOAP Request endpoint.  
 10. Change its name to Pass Destination.



11. Look at the Pass Destination Properties view.

Output
<pre> 1 %dw 1.0 2 %output application/xml 3 %namespace ns0 http://soap.training.mulesoft.com/ 4 --- 5 { 6     ns0#findFlight: { 7         destination: "????" 8     } 9 }</pre>
<b>Payload</b>
<b>Structure</b> <b>Preview</b>

12. Set the destination to SFO and look at the preview.

The screenshot shows the Mule Studio interface with the 'Preview' tab selected. On the left, the 'Payload' tab displays the following XML code:

```
1 %dw 1.0
2 %output application/xml
3 %namespace ns0 http://soap.training.mulesoft.com/
4 ---
5 {
6     ns0:findFlight: {
7         destination: "SFO"
8     }
9 }
```

On the right, the 'Output' tab shows the generated XML response:

```
<?xml version='1.0' encoding='UTF-8'?>
<ns0:findFlight xmlns:ns0="http://
soap.training.mulesoft.com/">
<destination>SFO</destination>
</ns0:findFlight>
```

Below the tabs are buttons for 'Payload', 'Structure', and 'Preview'.

## Debug the application

13. Save the file and debug the application.
14. Make a request to <http://localhost:8081/delta>.
15. In the Mule Debugger, step though to the Logger; you should now see only the SFO flights.

The screenshot shows the Mule Studio interface with the 'Mule Debugger' tab selected. It displays a table of variables and their types:

Name	Value	Type
Message		org.mule.DefaultMuleMessage
Message Processor	Logger	org.mule.api.processor.Log...
payload	[{airline=Delta, code=A1B2..., {airline=Delta, code=A1B2..., {airline=Delta, code=A1BT..., {airline=Delta, code=A1424..., {airline=Delta, {code=A14244, {origination=MUA, {plane=Boing 787, {price=294.0, {emptySeats=10, {departureDate=2015/02/12, {destination=SFO	java.util.ArrayList
0	{airline=Delta, code=A1B2..., {airline=Delta, code=A1B2..., {airline=Delta, code=A1BT..., {airline=Delta, code=A1424..., {airline=Delta, {code=A14244, {origination=MUA, {plane=Boing 787, {price=294.0, {emptySeats=10, {departureDate=2015/02/12, {destination=SFO	java.util.LinkedHashMap
1	{airline=Delta, code=A1B2..., {airline=Delta, code=A1B2..., {airline=Delta, code=A1BT..., {airline=Delta, code=A1424..., {airline=Delta, {code=A14244, {origination=MUA, {plane=Boing 787, {price=294.0, {emptySeats=10, {departureDate=2015/02/12, {destination=SFO	java.util.LinkedHashMap
2	{airline=Delta, code=A1B2..., {airline=Delta, code=A1B2..., {airline=Delta, code=A1BT..., {airline=Delta, code=A1424..., {airline=Delta, {code=A14244, {origination=MUA, {plane=Boing 787, {price=294.0, {emptySeats=10, {departureDate=2015/02/12, {destination=SFO	java.util.LinkedHashMap
0	airline=Delta	java.util.LinkedHashMap\$Entry
1	code=A14244	java.util.LinkedHashMap\$Entry
2	origination=MUA	java.util.LinkedHashMap\$Entry
3	plane=Boing 787	java.util.LinkedHashMap\$Entry
4	price=294.0	java.util.LinkedHashMap\$Entry
5	emptySeats=10	java.util.LinkedHashMap\$Entry
6	departureDate=2015/02/12	java.util.LinkedHashMap\$Entry
7	destination=SFO	java.util.LinkedHashMap\$Entry

16. Step through the rest of the flow.

## Set a flow variable to hold the value of a query parameter

17. Select the Set Destination transformer in getUnitedFlightsFlow and from the main menu, select Edit > Copy (or press Cmd+C/Ctrl+C).
18. Click in the process section of getDeltaFlightsFlow and from the main menu, select Edit > paste (or press Cmd+V/Ctrl+V); you should see a copy of the transformer added to the flow.
19. Move the transformer before the Pass Destination component.

20. In the Variable Properties view, change the display name to Set Destination and review the expression.



## Modify the input argument to use a dynamic destination from a query parameter

21. In the Pass Destination Properties view, replace the literal of SFO with a reference to the destination flow variable.

```

<ns0:findFlight destination='FlowVars.destination' />

```

## Debug the application

22. Save the file and redeploy the application.
23. Make a request to <http://localhost:8081/delta> and step through to the Logger; you should still only see flights to SFO.
24. Make another request to <http://localhost:8081/delta?code=CLE> and step through to the Logger;; you should now only see flights to CLE.

Name	Type
Message	org.mule.DefaultMuleMessage
Message Processor	org.mule.api.processor.LoggerMessageProcessor
payload	java.util.ArrayList
0	java.util.LinkedHashMap
0	java.util.LinkedHashMap\$Entry
1	java.util.LinkedHashMap\$Entry
2	java.util.LinkedHashMap\$Entry
3	java.util.LinkedHashMap\$Entry
4	java.util.LinkedHashMap\$Entry
5	java.util.LinkedHashMap\$Entry
6	java.util.LinkedHashMap\$Entry
7	java.util.LinkedHashMap\$Entry
1	java.util.LinkedHashMap
2	java.util.LinkedHashMap

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

In this walkthrough, you will work with the United flight data. You will:

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



### Review the United flight data

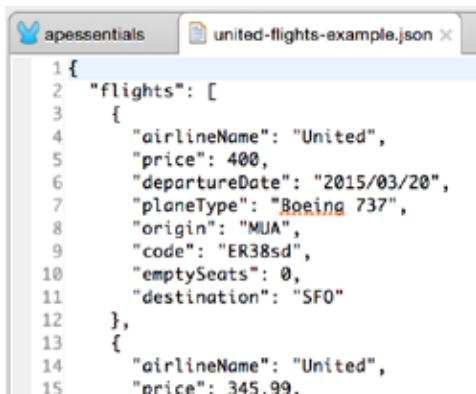
1. Run the application and make a request to <http://localhost:8081/united>.
2. Look at the names of the object keys.

```
{"flights": [{"airlineName": "United", "price": 400, "departureDate": "2015/03/20", "planeType": "Boeing 737", "origin": "MUA", "code": "ER38sd", "emptySeats": 0, "destination": "SFO"}, {"airlineName": "United", "price": 945, "departureDate": "2015/09/11", "planeType": "Boeing 757", "origin": "MUA", "code": "ER39rk", "emptySeats": 54, "destination": "SFO"}, {"airlineName": "United", "price": 954, "departureDate": "2015/02/12", "planeType": "Boeing 777", "origin": "MUA", "code": "ER39rj", "emptySeats": 23, "destination": "SFO"}]}
```

### Review at the flights schema and example files

3. Open the united-flights-example.json file located in the src/main/resources/schemas-and-examples folder and examine the code.

- Open the united-flights-schema.json file located in the src/main/resources/schemas-and-examples folder and examine the code.



```

apessentials   united-flights-example.json
1 {
2   "flights": [
3     {
4       "airlineName": "United",
5       "price": 400,
6       "departureDate": "2015/03/20",
7       "planeType": "Boeing 737",
8       "origin": "MUA",
9       "code": "ER38sd",
10      "emptySeats": 0,
11      "destination": "SFO"
12    },
13    {
14      "airlineName": "United",
15      "price": 345.99.

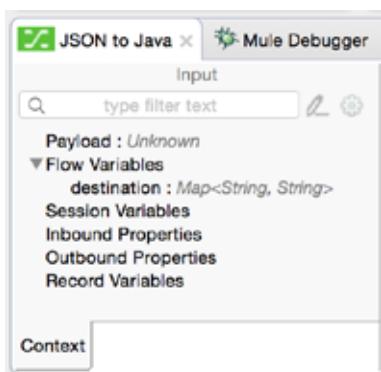
```

## Add a DataWeave Transform Message component

- Locate getUnitedFlightsFlow.
- Add a Transform Message component after the United REST Request.
- Change its name to JSON to Java.
- Add a Logger after the component.

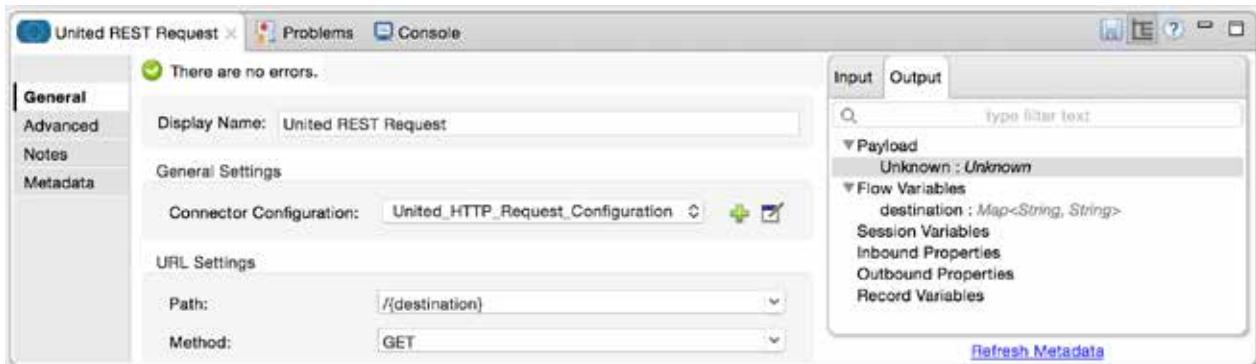


- In the Transform Message Properties view, leave the output set to Payload and of type application/java.
- Look at the Input section of the Transform Message Properties view; the payload is of type unknown.



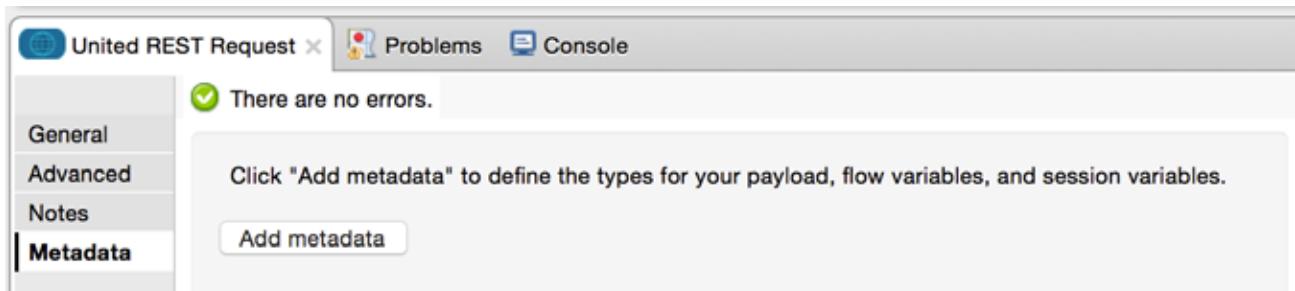
## Add metadata to the HTTP Request endpoint

11. In the United REST Request Properties view, click the Output tab in the Metadata section; you should see the payload is of type unknown.



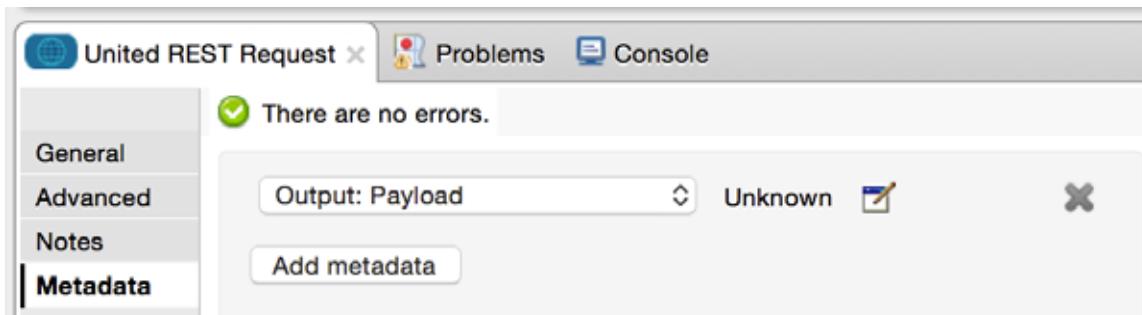
12. Click the Metadata link in the left-side navigation.

13. Click the Add metadata button.



14. In the drop-down menu that appears, select Output: Payload.

15. Click the Edit button located to the right of the drop-down menu.



16. In the Define Type dialog box, select Create new type.

17. Set the type to JSON.

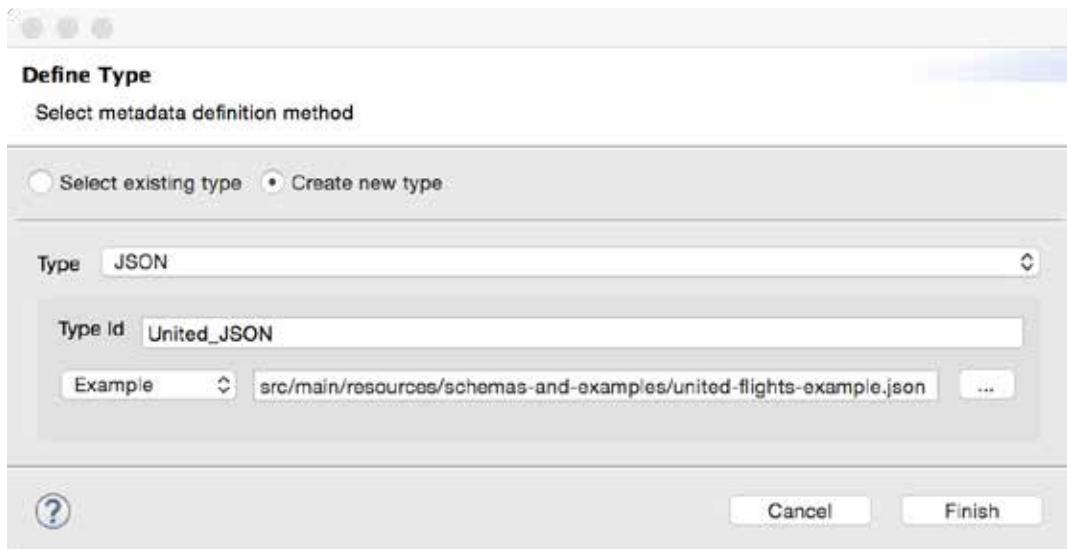
18. Set the type id to United\_JSON.

19. In the drop-down menu, select Example.

20. Click the browse button.

21. In the Open dialog box browse to src/main/resources/schemas-and-examples/united-flights-example.json and click Open.

22. In the Define Type dialog box, click Finish.



23. Look at the Output tab in the Metadata section again; you should now see the payload is of type JSON with a flights property that is a list of objects with specific fields.

Input   Output

Q   type filter text

▼ Payload

  ▼ Json : Json

    ▼ flights : List<Json>

      airlineName : String  
      code : String  
      departureDate : String  
      destination : String  
      emptySeats : integer  
      origin : String  
      planeType : String  
      price : Integer

  ▼ Flow Variables

    destination : Map<String, String>

[Refresh Metadata](#)

## Transform the JSON to a collection of Java objects

24. Return to the JSON to Java component Properties view.

25. Look at the Input section again; the payload should now be of type Json.

26. Expand the flights List; you should see the field names of each object.

The screenshot shows the Mule Debugger interface with the following payload structure:

```
Payload : Json
  flights : List<Json>
    airlineName : String
    code : String
    departureDate : String
    destination : String
    emptySeats : Integer
    origin : String
    planeType : String
    price : Integer
```

Below the payload, there are sections for Flow Variables (destination: Map<String, String>) and Session Variables. A Context section is also present at the bottom.

27. Delete the existing DataWeave expression (the curly braces) and set it to payload.

The screenshot shows the Transform Message tool with the following configuration:

Input

Output: Payload

```
1 %dw 1.0
2 %output application/java
3 ---
4 payload
```

28. Look at the Output section; the output structure should be set to a Map.

## Change the output structure

29. Change the DataWeave expression to payload.flights; the output structure should now be a List of Map objects.

The screenshot shows the Transform Message tool with the following configuration:

Output: Payload

```
1 %dw 1.0
2 %output application/java
3 ---
4 payload.flights
```

The payload structure is expanded to show:

```
Payload
  List<Map> : List
    airlineName : String
    code : String
    departureDate : String
    destination : String
    emptySeats : Double
    origin : String
    planeType : String
    price : Double
```

Below the payload, there are sections for Flow Variables (destination: Map<String, String>) and Session Variables.

30. Modify the DataWeave expression to use the map operator.

31. In the transformation function, set the following mappings:

- airlineName < airlineName
- departureDate < departureDate
- destination > destination
- emptySeats < emptySeats as a number
- flightCode < code
- origination < origin
- planeType < planeType
- price < price and format it as a number with the pattern “###.##”

*Note: If you want, you can copy these lines of code from the Delta DataWeave expression.*

The screenshot shows the Mule Studio interface with the DataWeave editor open. The editor has two panes: 'Payload' on the left and 'Output' on the right. The Payload pane contains the following Delta DataWeave code:

```
1@dw 1.0
2 @output application/java
3 ---
4 @payload.flights map {
5   airlineName: $.airlineName,
6   departureDate: $.departureDate,
7   destination: $.destination,
8   emptySeats: $.emptySeats as :number,
9   flightCode: $.code,
10  origination: $.origin,
11  planeType: $.planeType,
12  price: $.price as :number {format: "##.##"}}
13 }
```

The Output pane shows the resulting payload structure:

- Payload (List<Map>):
  - airlineName: String
  - code: String
  - departureDate: String
  - destination: String
  - emptySeats: Double
  - origin: String
  - planeType: String
  - price: Double
- Flow Variables:
  - destination: Map<String, String>
- Session Variables

## Debug the application

32. Save the file and debug the application.

33. Make a request to <http://localhost:8081/united>.

34. Watch the payload value and step through to the Logger.

35. Drill-down into the data and make sure it all mapped correctly.

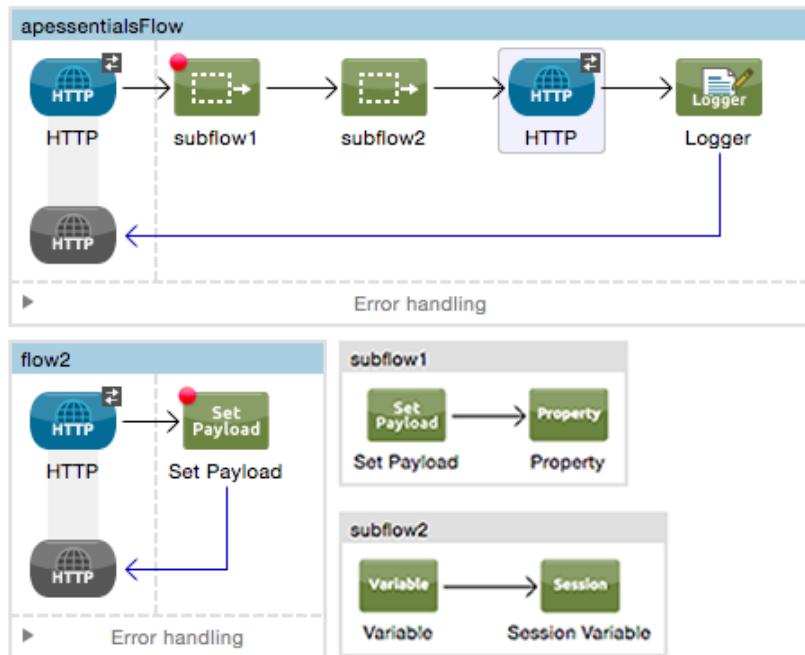
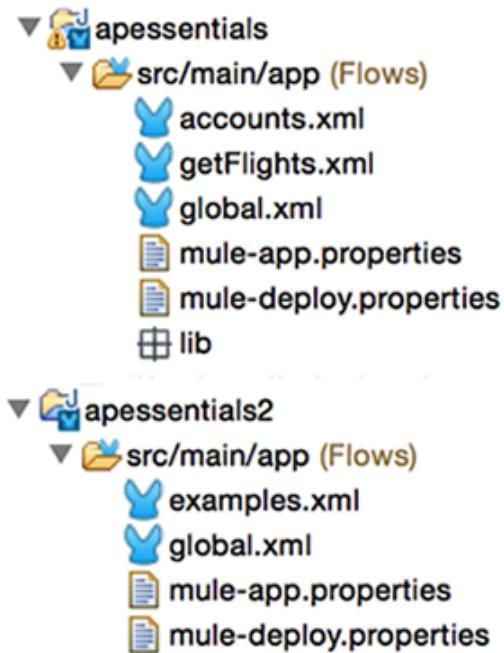
36. Stop the Mule Debugger.

The screenshot shows the Mule Studio 'Mule Debugger' tool window. It displays a table of mapped values for the 'payload' variable. The columns are 'Name', 'Value', and 'Type'. The data is as follows:

Name	Value	Type
Message		org.mule.DefaultMuleMessage
Message Processor	Logger	org.mule.api.processor.Logger...
payload	[{airlineName=United, departureDa...}...]	java.util.ArrayList
@ 0	{airlineName=United, departureDa...}	java.util.LinkedHashMap
@ 0	airlineName=United	java.util.LinkedHashMap\$Entry
@ 1	departureDate=2015/03/20	java.util.LinkedHashMap\$Entry
@ 2	destination=SFO	java.util.LinkedHashMap\$Entry
@ 3	emptySeats=0	java.util.LinkedHashMap\$Entry
@ 4	flightCode=ER38sd	java.util.LinkedHashMap\$Entry
@ 5	origination=MUA	java.util.LinkedHashMap\$Entry
@ 6	planeType=Boeing 737	java.util.LinkedHashMap\$Entry
@ 7	price=400	java.util.LinkedHashMap\$Entry
@ 1	{airlineName=United, departureDa...}	java.util.LinkedHashMap



# Module 6: Refactoring Mule Applications



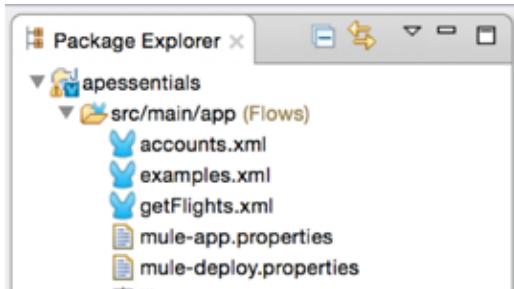
In this module, you will learn:

- To separate applications into multiple configuration files.
- To encapsulate global elements in a separate configuration file.
- To create and run multiple applications.
- To create and reference flows and subflows.
- About variable persistence through subflows and flows and across transport barriers.

## Walkthrough 6-1: Separate applications into multiple configuration files

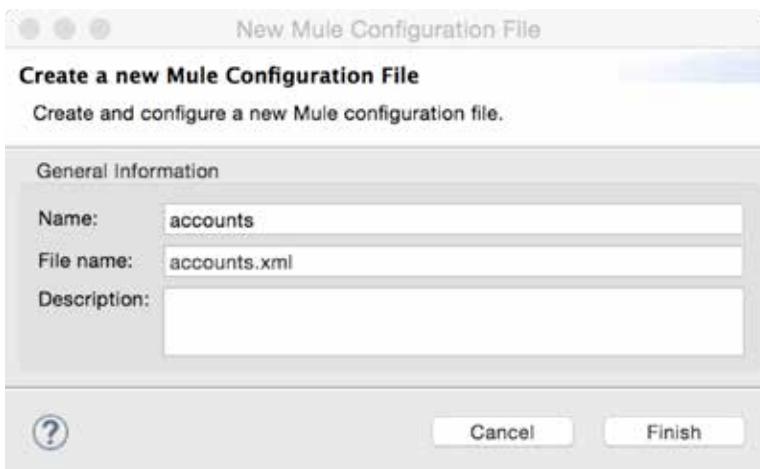
In this walkthrough, you will refactor your monolithic apessentials application. You will:

- Separate the account flows into accounts.xml.
- Move the rest of the example flows into examples.xml.
- Rename apessentials.xml to getFlights.xml.



### Move account flows to a new configuration file

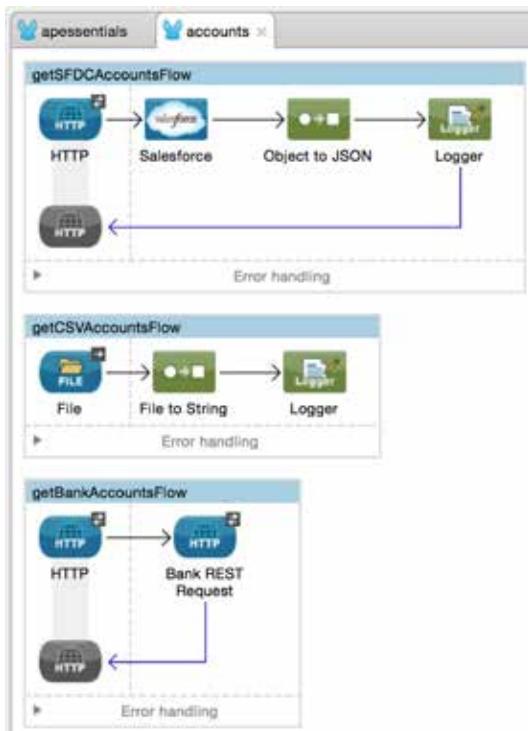
1. Right-click the apessentials project in the Package Explorer and select New > Mule Configuration File.
2. Set the name to accounts and click Finish.



3. In accounts.xml, switch to the Configuration XML view.
4. Place some empty lines between the start and end mule tags.
5. Return to apessentials.xml and switch to the Configuration XML view.
6. Select and cut getSFDCAccountsFlow, getCSVAccountsFlow, and getBankAccountsFlow.
7. Return to accounts.xml and paste the flows inside the mule tags.

*Note: If these flows are not adjacent, you will need to repeat the process several times.*

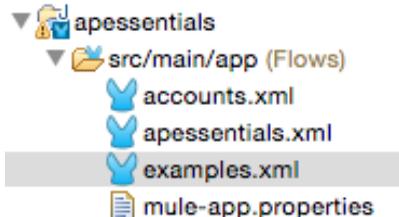
8. Switch to the Message Flow view.



9. Save all the files by selecting File > Save All, clicking the Save All button, or pressing Shift+Cmd+S.

## Move non-flight flows to a new configuration file

10. Right-click the apessentials project in the Package Explorer and select New > Mule Configuration File.
11. Set the name to examples and click Finish.

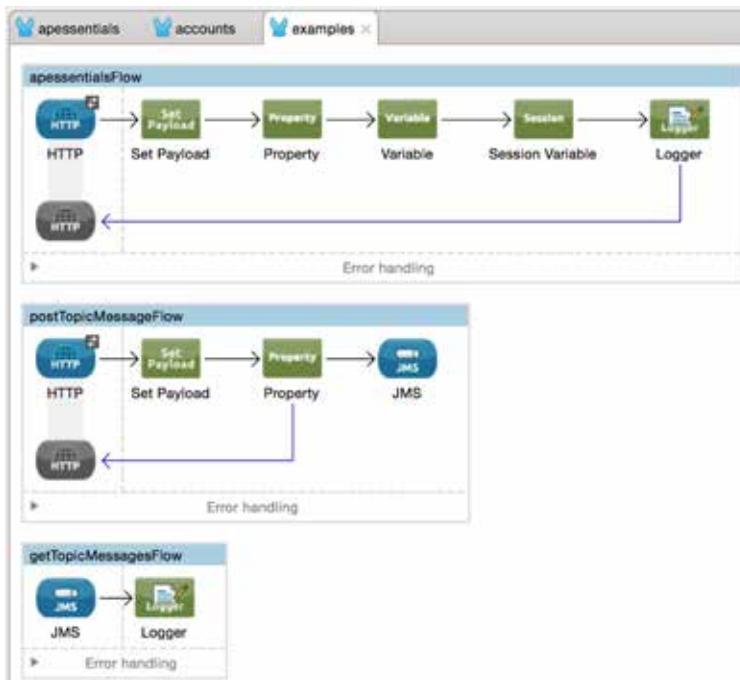


12. In examples.xml, switch to the Configuration XML view.
13. Place some empty lines between the start and end mule tags.
14. Return to apessentials.xml and select and cut apessentialsFlow, getTopicMessagesFlow, and postTopicMessageFlow.
15. Return to examples.xml and paste the flows inside the mule tags.

*Note: If these flows are not adjacent, you will need to repeat the process several times.*

16. Switch to the Message Flow view.

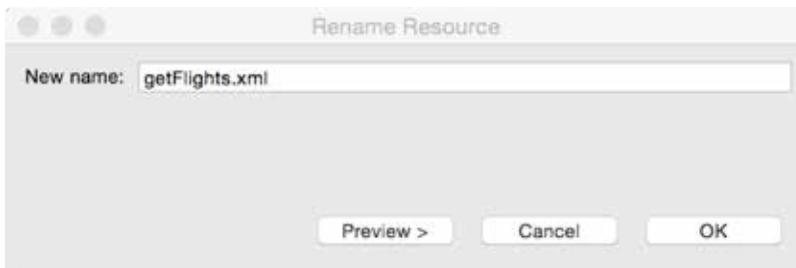
17. Save all the files.



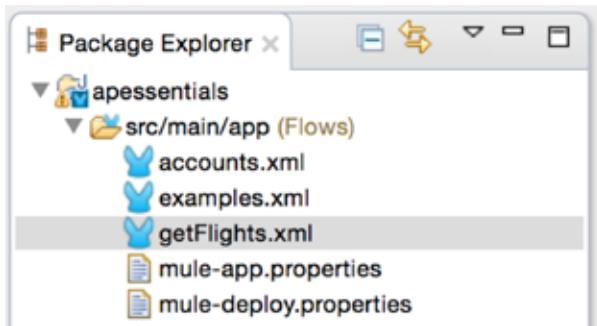
## Rename the apessentials configuration file

18. Right-click apessentials.xml in the Package Explorer and select Refactor > Rename.

19. In the Rename Resource dialog box, enter a new name of getFlights.xml and click OK.



20. Double-click getFlights.xml in the Package Explorer to reopen it.



## Examine the global elements

21. Click the Global Elements tab at the bottom of the canvas.
22. Notice that all of the global elements are still defined in getFlights.xml.

Type	Name
HTTP Listener Configuration	HTTP_Listener_Configuration
HTTP Request Configuration	United_HTTP_Request_Configuration
HTTP Request Configuration	Bank_REST_Request_Configuration
Web Service Consumer	Delta_Web_Service_Consumer
MySQL Configuration	Training_MySQL_Configuration
Active MQ	Active_MQ
Salesforce: Basic authentication	Salesforce

## Test the application

23. Run the application; you should get an error that the prefix for sfdc:query is not bound.

```
ERROR 2015-08-20 14:11:19,395 [main] org.mule.module.launcher.application.DefaultMuleApplication: null
org.xml.sax.SAXParseException: The prefix "sfdc" for element "sfdc:query" is not bound.
    at org.apache.xerces.util.ErrorHandlerWrapper.createSAXParseException(Unknown Source) ~[?:?]
    at org.apache.xerces.util.ErrorHandlerWrapper.fatalError(Unknown Source) ~[?:?]
```

## Add namespaces and schema locations

24. Return to the Configuration XML view for accounts.xml.
25. Look for the sfdc prefix in the beginning of the code; you should see that there is not a sfdc namespace definition or schema location.
26. Place the cursor after <sfdc:query and press Ctrl+Space.

```
14<flow name="getSFDCAccountsFlow">
15    <http:listener config-ref="HTTP_Listener_Con
16        path="/sfdc" allowedMethods="GET" doc:n
17        <sfdc:query config-ref="Salesforce"
18            query="<sfdc:query
19                doc:nam<sfdc:query-all
20                <json:objec<sfdc:query-result-stream
21                    doc:nam<sfdc:query-single
```

27. In the autocomplete pop-up, select sfdc:query and press Enter.
28. Delete the new config-ref attribute that was added.

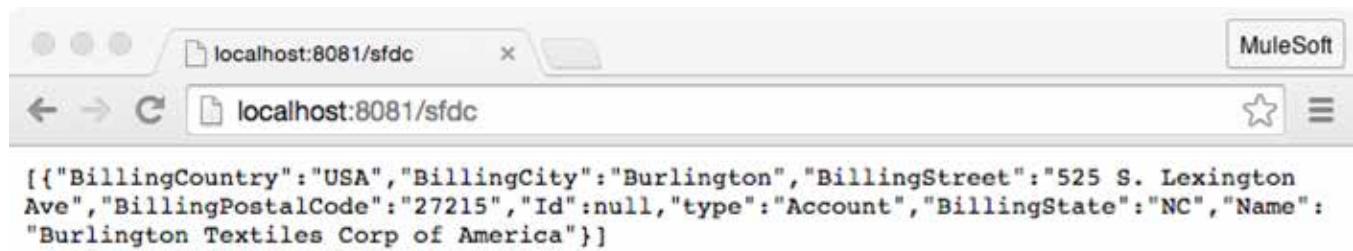
29. Locate the new xmlns and schemaLocation for sfdc.

```
3<mule xmlns:sfdc="http://www.mulesoft.org/schema/mule/sfdc" xmlns:file="<br/>4    xmlns:http="http://www.mulesoft.org/schema/mule/http" xmlns:json="ht<br/>5    xmlns="http://www.mulesoft.org/schema/mule/core" xmlns:doc="http://<br/>6    xmlns:spring="http://www.springframework.org/schema/beans" version="<br/>7    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"<br/>8    xsi:schemaLocation="http://www.mulesoft.org/schema/mule/sfdc http://<br/>9    http://www.mulesoft.org/schema/mule/file http://www.mulesoft.org/schema/<br/>10   mule-file.xsd"></mule>
```

## Test the application

30. Save all the files and run the application.

31. Make a request to <http://localhost:8081/sfdc>; you should now get account results.



## Walkthrough 6-2: Encapsulate global elements in a separate configuration file

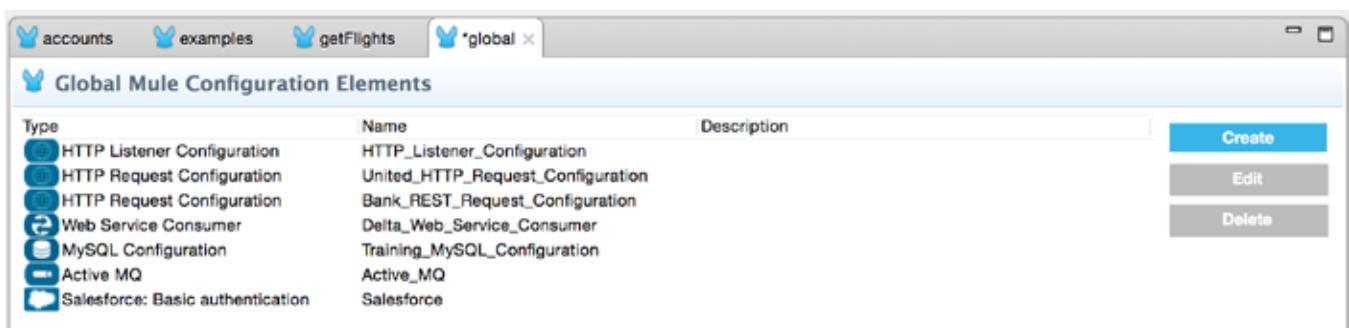
In this walkthrough, you will separate global elements into a new configuration file. You will:

- Create a configuration file for just global elements.
- Move all the existing global elements to this file.



### Create a new configuration file for the global elements

1. Right-click the 'getFlights.xml' file in the Package Explorer and select Copy.
2. Right-click the 'src/main/app' folder and select > Paste.
3. In the Name Conflict dialog box, enter a name of 'global.xml' and click OK.
4. Open 'global.xml'.
5. Shift-click all the flows to select them all and then right-click and select Delete.
6. Click the Global Elements tab; you should see all the existing global elements.
7. Save the file.



### Delete the global elements in 'getFlights.xml'

8. Return to 'getFlights.xml'.

9. Switch the editor to the Configuration XML view.

```
22 http://www.mulesoft.org/schema/mule/sfdc http://www.mulesoft.or
23 http://www.mulesoft.org/schema/mule/json http://www.mulesoft.or
24 http://www.mulesoft.org/schema/mule/ee/dw http://www.mulesoft.c
25     <http:listener-config name="HTTP_Listener_Configuration" hc
26     <http:request-config name="United_HTTP_Request_Configuratio
27     <http:request-config name="Bank_REST_Request_Configuration"
28         <http:raml-api-configuration location="https://anypoint
29     </http:request-config>
30     <ws:consumer-config name="Delta_Web_Service_Consumer" wsdlL
31     <db:mysql-config name="Training_SQL_Configuration" host="
32     <jms:activemq-connector name="Active_MQ" specification="1.1
33     <sfdc:config name="Salesforce" username="YOUR_USERNAME" pas
34     <flow name="transformStaticFlightsFlow">
35         <http:listener config-ref="HTTP_Listener_Configuration"
36             path="/static" allowedMethods="GET" doc:name="HTTP"
```

10. Select and delete the eight global elements defined before the flows.

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

11. Save the file.

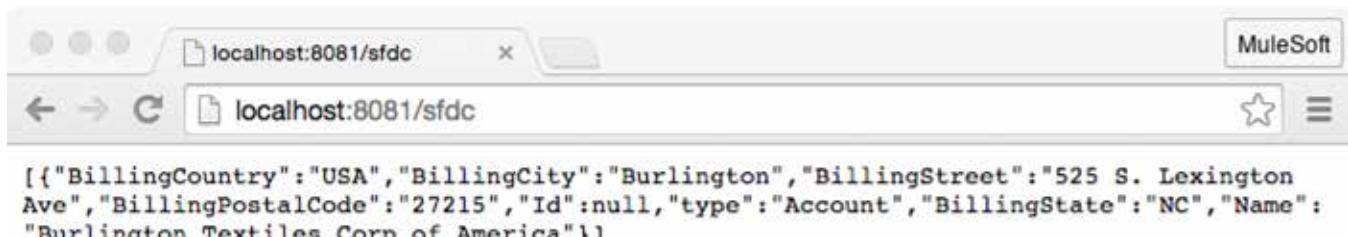
12. Return to the Message Flow view.

13. Double-click the HTTP Listener connector of any flow; the connector configuration should still be set to HTTP\_Listener\_Configuration, which is now defined in global.xml.

## Test the application

14. Run the application.

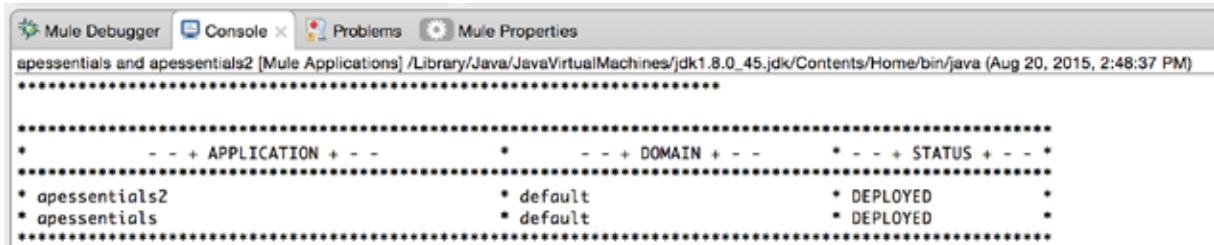
15. Make a request to <http://localhost:8081/sfdc>; you should still get account results.



## Walkthrough 6-3: Create and run multiple applications

In this walkthrough, you will separate your project into two projects. You will:

- Create a new project and application called apessentials2.
- Move the examples configuration file into the apessentials2 application.
- Create new global connector configurations in the new project.
- Create a new run configuration to run multiple applications simultaneously.

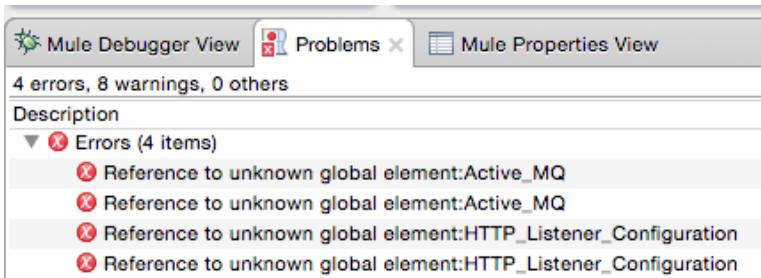


### Create a new project and application

1. Select File > New > Mule Project.
2. Set the name to apessentials2 and click Finish.
3. Right-click the apessentials2.xml file in the Package Explorer and select Delete.
4. In the Delete dialog box, click OK.

### Move the examples flow into the new project

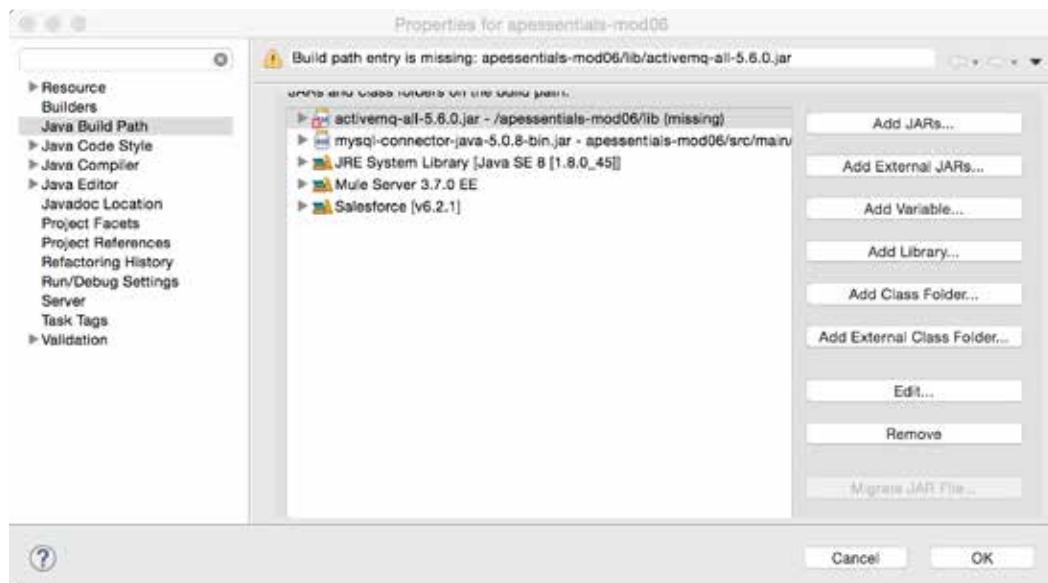
5. Drag examples.xml from the apessentials project to the src/main/app folder of the apessentials2 project; you should get unknown global element errors.



### Move activemq JAR into the new project

6. Right-click apessentials2 and select New > Folder.
7. In the New Folder dialog box, set the folder name to lib and click Finish.
8. Drag activemq-all-{version}.jar.xml from the apessentials/lib folder to the apessentials2/lib folder.

9. Right-click activemq-all-{version}.jar and select Build Path > Add to Build Path; you should see the JAR file added to Referenced Libraries.
10. In global.xml, switch to the Global Elements view.
11. Select the Active MQ configuration element and click Delete.
12. Save the file.
13. Right-click apessentials in the Package Explorer and select Properties.
14. Select Java Build Path and click the Libraries tab.
15. Select activemq-all-{version}.jar and click Remove.

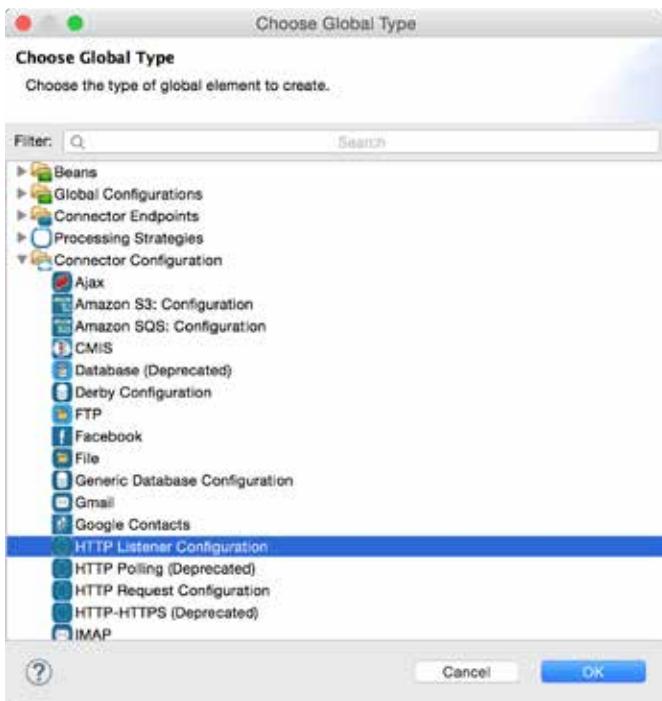


16. Click OK.

## Create new global elements

17. Right-click the apessentials2 project and select New > Mule Configuration File.
18. In the New Mule Configuration File dialog box, set the name to global and click Finish.
19. In global.xml, switch to the Global Elements view.
20. Click the Create button.

21. Select Connector Configuration > HTTP Listener Configuration and click OK.

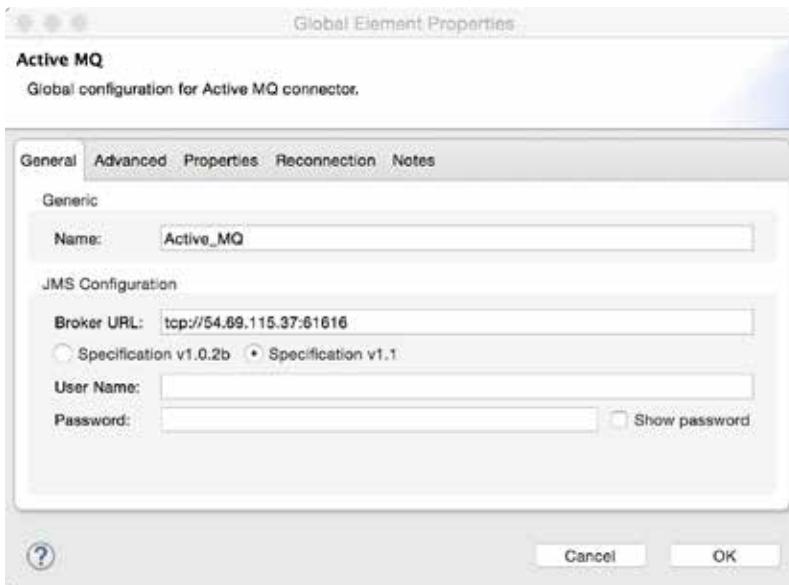


22. In the Global Elements dialog box, click OK.

23. Click the Create button again and select Connector Configuration > JMS > Active MQ and click OK.

24. In the Global Elements dialog box, set the Broker URL to tcp://54.69.115.37:61616.

25. Select Specification v1.1 and click OK.

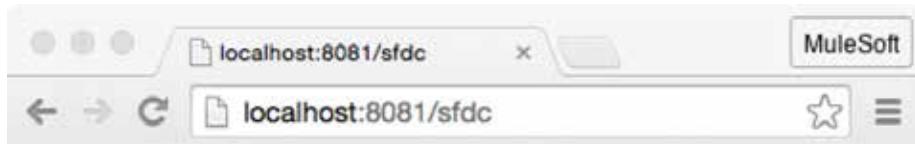


*Note: You could also copy and paste these definitions from the apessentials global.xml file.*

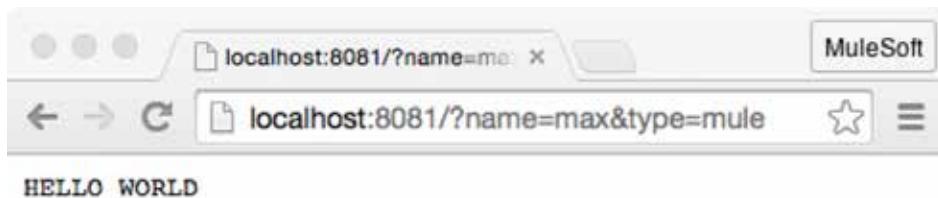
26. Save all the files.

## Run the new application

27. Right-click apessentials2 and select Run As > Mule Application.
28. Make a request to <http://localhost:8081/sfdc>; you should get Resource not found – that application is not running.

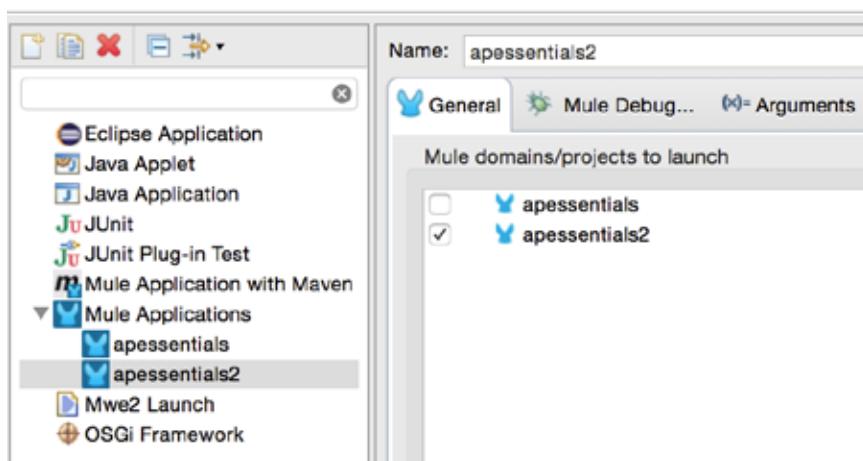


29. Make a request to <http://localhost:8081/?name=max&type=mule> using your own query parameter values; you should get HELLO WORLD.

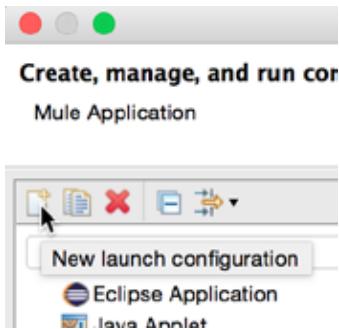


## Create a run configuration to run multiple applications

30. Return to Anypoint Studio.
31. On the main menu bar, select Run > Run Configurations.
32. Click apessentials in the left menu bar under Mule Applications; you should see that the apessentials project is selected to launch.
33. Click apessentials2 in the left menu bar; you should see that the apessentials2 project is selected to launch.

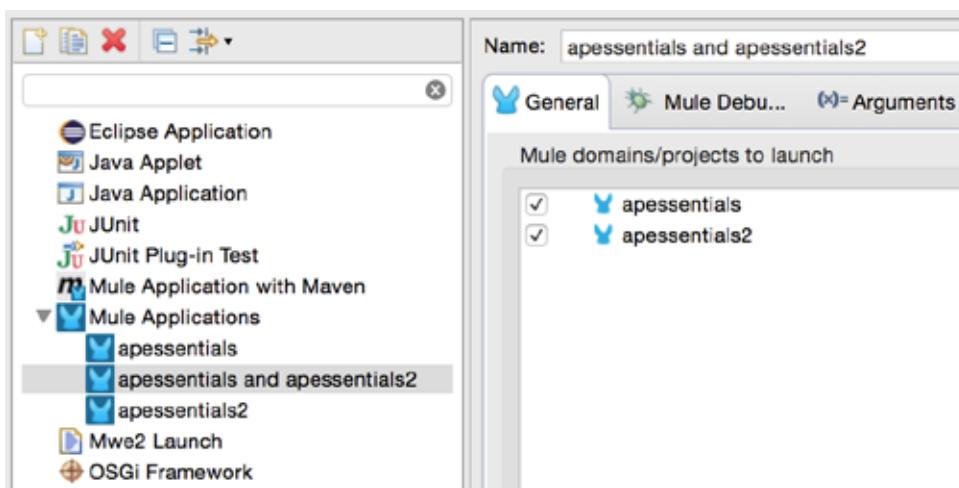


34. Click the New Launch Configuration button.



35. Set the name to apessentials and apessentials2.

36. On the General tab, select both the apessentials and apessentials2 projects.



## Run multiple applications

37. Click Run.

38. Look at the console; apessentials should have deployed and APESSENTIALS2 failed.

```
*****
*          - - + APPLICATION + - -          *          - - + DOMAIN + - -          *          - - + STATUS + - - *
*****
* apessentials2                                * default                      * FAILED
* apessentials                                * default                      * DEPLOYED
*****
*****
```

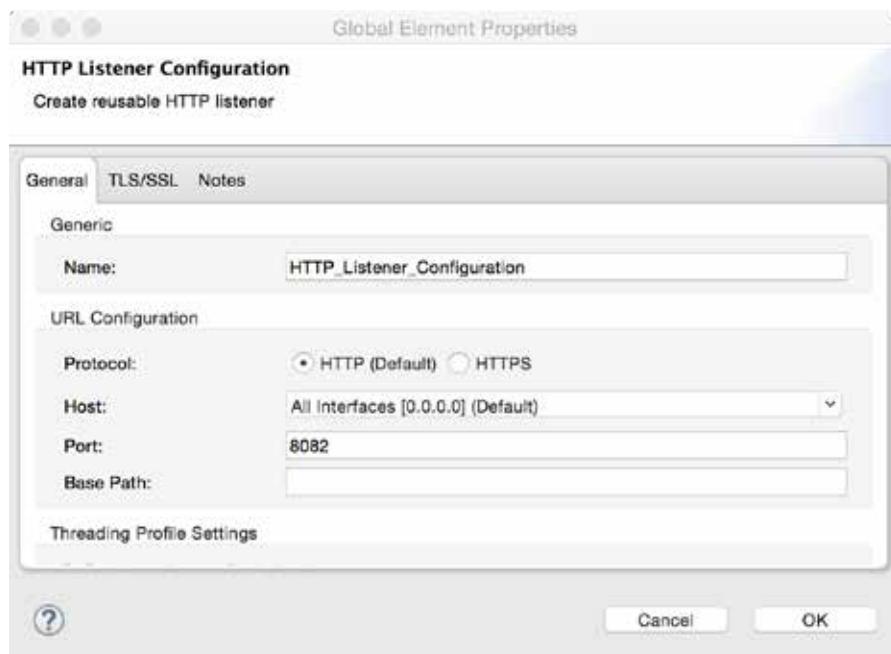
39. Scroll up the console; you should see an address already in use error.

```
INFO 2015-08-20 14:38:38,594 [main] org.mule.util.queue.QueueXaResourceManager: :
ERROR 2015-08-20 14:38:38,612 [main] org.mule.module.launcher.application.DefaultApplication
java.net.BindException: Address already in use
    at sun.nio.ch.Net.bind0(Native Method) ~[?:1.8.0_45]
    at sun.nio.ch.Net.bind(Net.java:437) ~[?:1.8.0_45]
```

40. Navigate to the Global Elements view in global.xml in apessentials2.

41. Double-click the HTTP Listener Configuration (or select it and click the Edit button).

42. In the Global Elements Properties dialog box, change the port to 8082 and click OK.

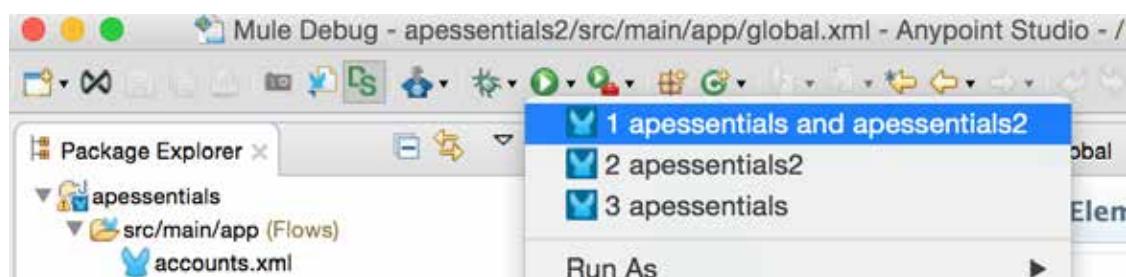


43. Save the file; both applications should now be successfully deployed – apessentials was already deployed.

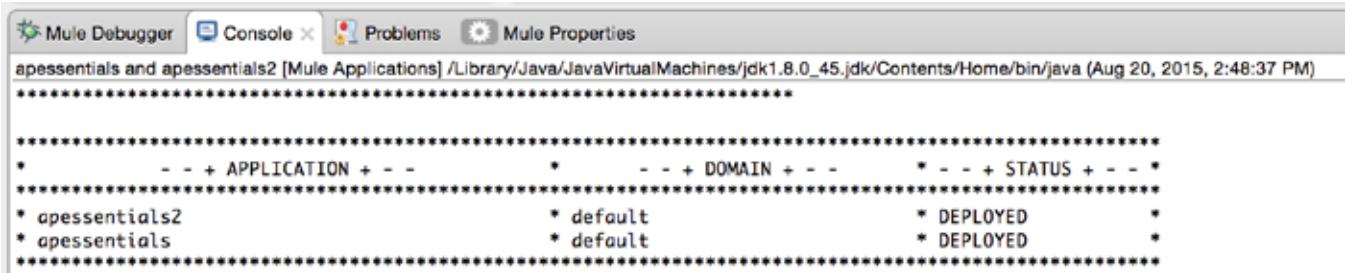
```
+-----+
+ Started app 'apessentials2' +
+-----+
```

## Test the application

44. Make a request to <http://localhost:8081/sfdc>; you should see Salesforce data.
45. Make a request to <http://localhost:8082/?name=max&type=mule> using your own query parameter values; you should get HELLO WORLD.
46. Stop the Mule runtime.
47. Click the arrow next to the Run button in the Anypoint Studio toolbar and select apessentials and apessentials2.



48. Look at the console; you should both applications deployed.



The screenshot shows the Mule Debugger interface with the 'Console' tab selected. The output window displays deployment logs for two applications:

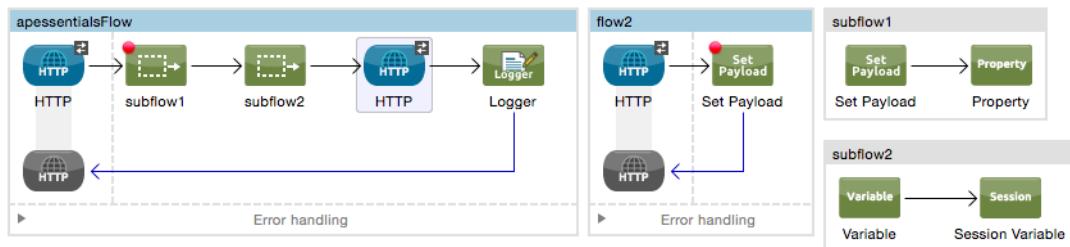
```
* - - + APPLICATION + - - * - - + DOMAIN + - - * - - + STATUS + - - *
* apessentials2           * default          * DEPLOYED      *
* apessentials            * default          * DEPLOYED      *
```

49. Stop the Mule runtime.

## Walkthrough 6-4: Create and reference flows and subflows

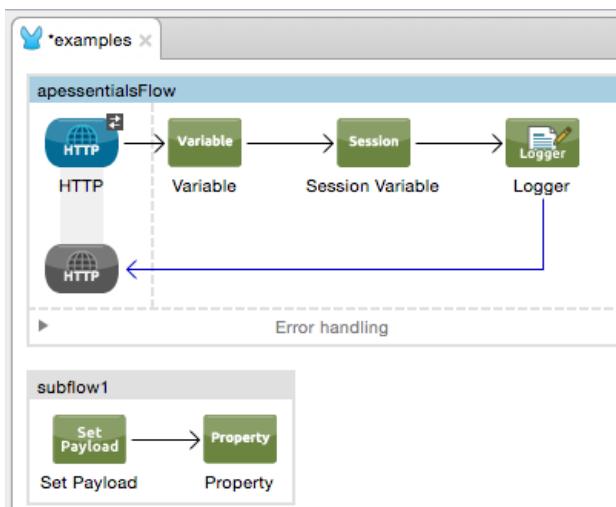
In this walkthrough, you will work with the apessentialsFlow in examples.xml. You will:

- Extract processors into separate subflows and flows.
- Use the Flow Reference component to reference other flows.
- Create and reference synchronous and asynchronous flows.
- Explore variable persistence through flows, subflows, and across transport barriers.



### Create a subflow

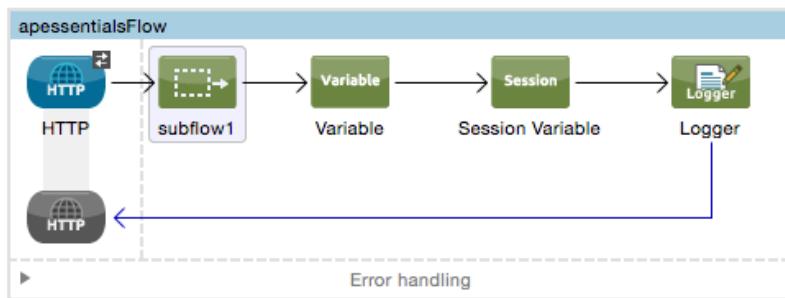
1. Open examples.xml in apessentials2.
2. Drag a Sub Flow scope element to the canvas.
3. Select the Set Payload and Property transformers in the apessentialsFlow and drag them into the subflow.
4. Change the name of the subflow to subflow1.



### Reference a subflow

5. Drag a Flow Reference component from the palette and drop it into the apessentialsFlow between the HTTP Listener connector endpoint and the Variable transformer.

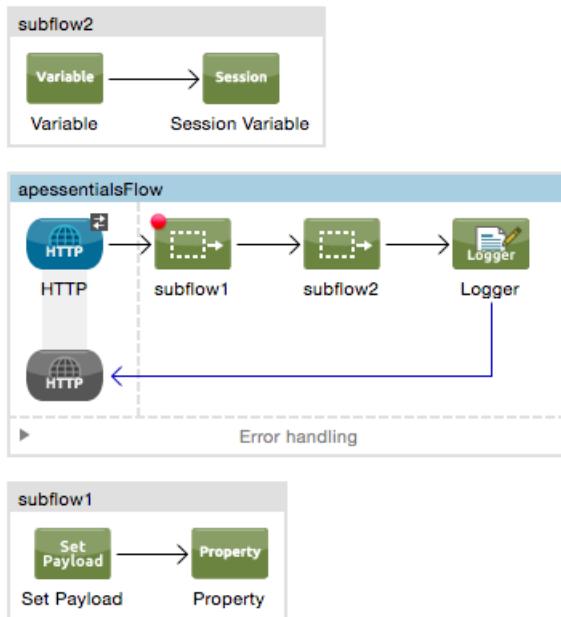
6. In the Flow Reference Properties view, set the flow name to subflow1.



7. Add a breakpoint to the subflow1 Flow Reference.

## Extract processors into a subflow

8. Right-click the Variable and Session Variable transformers and select Extract to > Sub Flow.
9. In the Extract Flow dialog box, set the flow name to subflow2.
10. Leave the target Mule configuration set to current and click OK.
11. Look at the new Flow Reference Properties view; the flow name should already be set to subflow2.

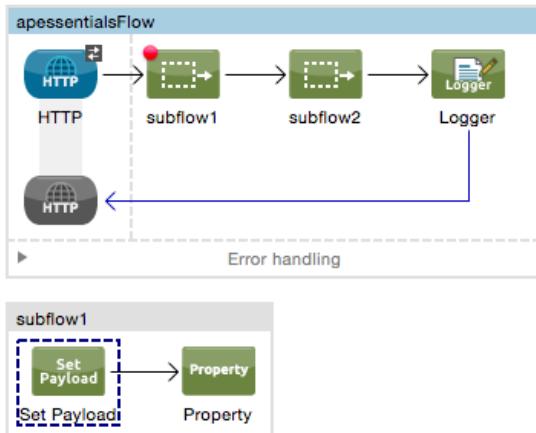


12. Drag subflow2 below subflow1.

13. Save the file.

## Debug the application

14. Click the Debug button and select apessimentials2 to just debug apessimentials2.
15. Make a request to <http://localhost:8082/?name=max&type=mule> using your own query parameter values.
16. Step through the application, watching messages move into and out of the subflows.



17. Make another request to <http://localhost:8082/?name=max&type=mule> using your own query parameter values.
18. Step through the application again, this time watching the values of the inbound properties, variables, outbound properties, and session variables.

## Create a second flow

19. Drag a Flow scope element to the canvas.
20. Change the flow name to flow2.
21. Add a Set Payload transformer to the process section of the flow.
22. Add a breakpoint to it.



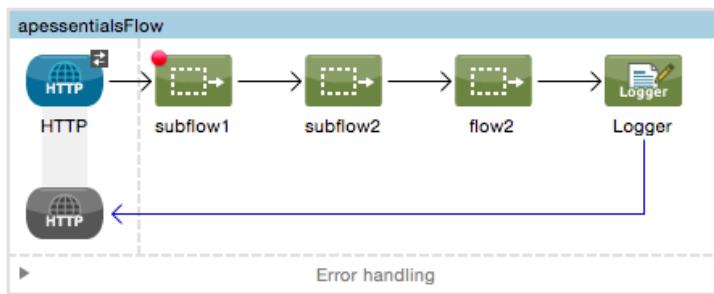
23. In the Set Payload Properties view, set the value to a string, like flow2.

## Reference a flow

24. Drag a Flow Reference component from the palette and drop it into the apessimentialsFlow between the subflow2 Flow Reference and the Logger.

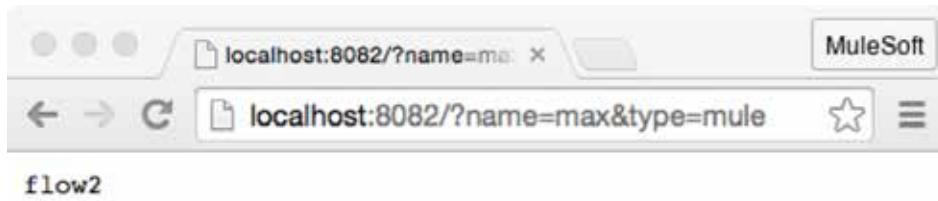


25. In the Flow Reference Properties view, set the flow name to flow2.



## Debug the application

26. Save the file to redeploy the application in debug mode.
27. Make a request to <http://localhost:8082/?name=max&type=mule> using your own query parameter values.
28. Step through the application, watching the flow move into and out of the second flow; at the end, you should see the new payload value set in the second flow returned.

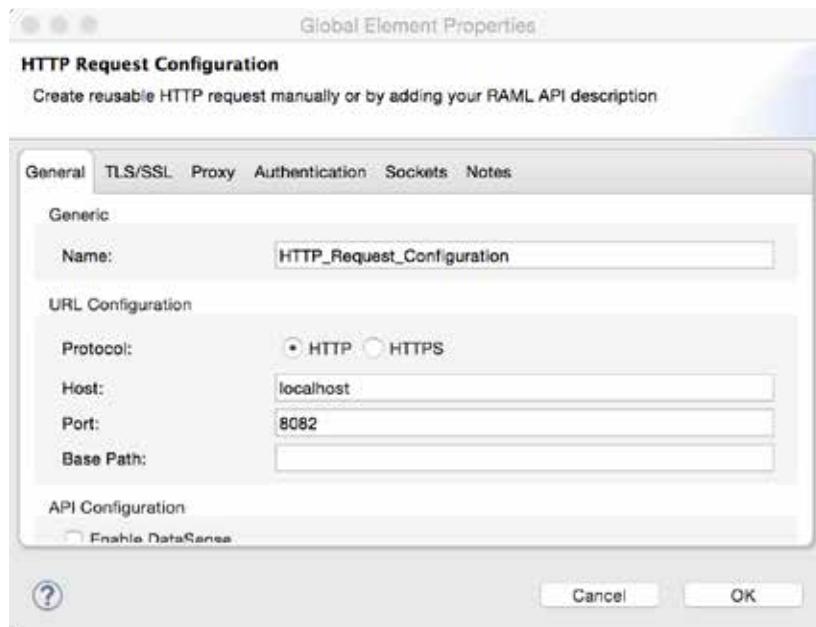


29. Make another request to <http://localhost:8082/?name=max&type=mule> using your own query parameter values.
30. Step through the application again, this time watching the values of the inbound properties, variables, outbound properties, and session variables.

## Create a transport barrier for the second flow

31. In global.xml for apessentials2, switch to the Global Elements view.
32. Click Create.
33. In the Choose Global Type dialog box, select Connector Configuration > HTTP Request Configuration and click OK.

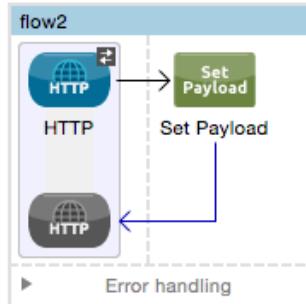
34. In the Global Elements dialog box, set the host to localhost, the port to 8082, and click OK.



35. In examples.xml, drag an HTTP connector into the Source section of flow2.

36. In the HTTP Properties view, set the connector configuration to the existing `HTTP_Listener_Configuration`.

37. Set the path to `/flow2` and the allowed methods to GET.

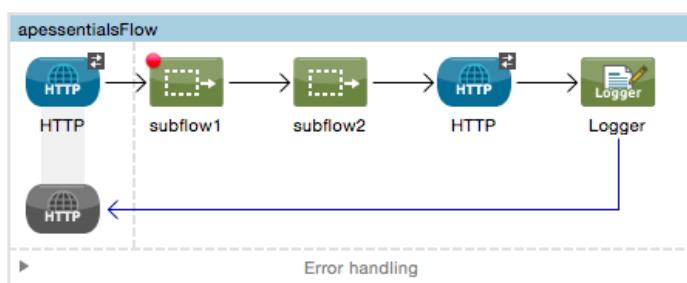


38. Add a second HTTP connector after the `flow2` reference in `apessimistFlow`.

39. Delete the `flow2` reference.

40. In the HTTP Properties view, set the connector configuration to `HTTP_Request_Configuration`.

41. Set the path to `/flow2` and the method to GET.



## Debug the application

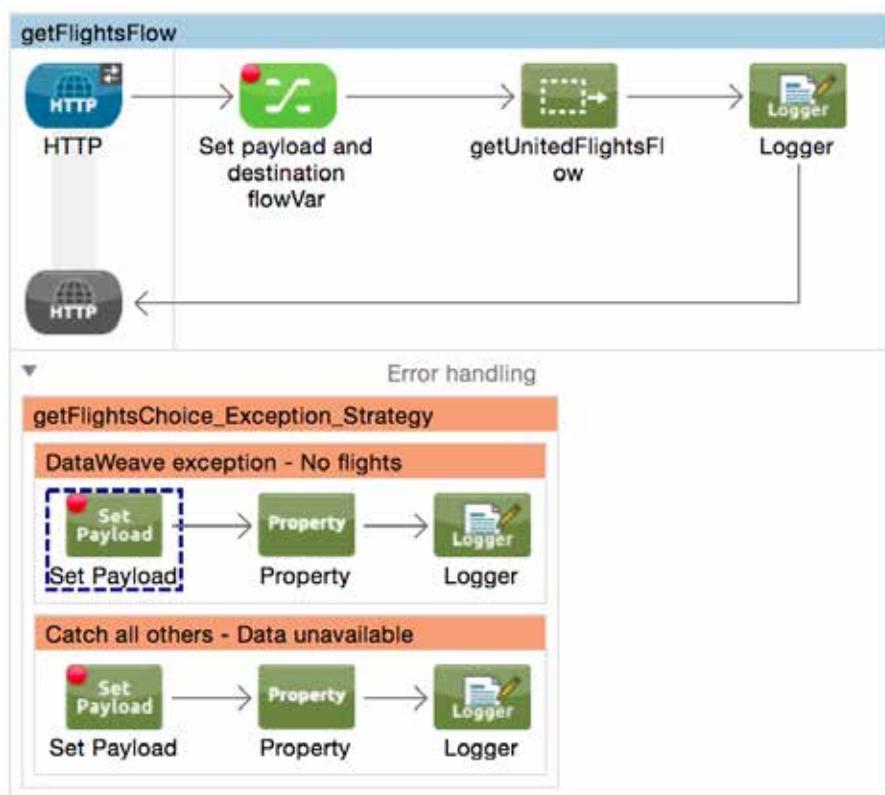
42. Save all the files to redeploy the application.
43. Make another request to <http://localhost:8082/?name=max&type=mule> using your own query parameter values.
44. Step through the application again, pressing the Resume button to move into flow2.
45. Watch the values of the inbound properties, outbound properties, flow variables, and session variables as you move into flow2 and then back to apessentialsFlow.

*Note: Ignore any TimeoutException you may get.*

*Note: Session variables are persisted across some but not all transport barriers. As you see here, they are not propagated across the HTTP transport barrier.*



# Module 7: Handling Errors



In this module, you will learn:

- About the different types of exception strategies.
- To handle messaging exceptions in flows.
- To create and use global exception handlers.
- To specify a global default exception strategy.

## Walkthrough 7-1: Handle a messaging exception

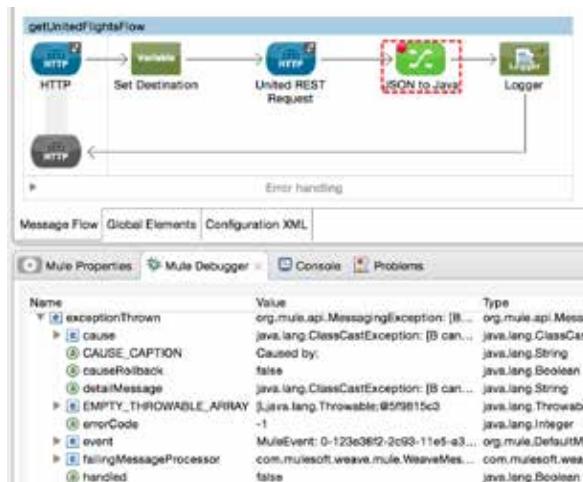
In this walkthrough, you will handle an exception thrown by the United flow when a destination with no flights is used. You will:

- Add a Catch Exception Strategy to a flow.
- Catch the exception and send an error message back to the requester.
- Reference the exception object inside an exception handler.
- Create and catch a web service request error.



### Test the application for when United returns no results

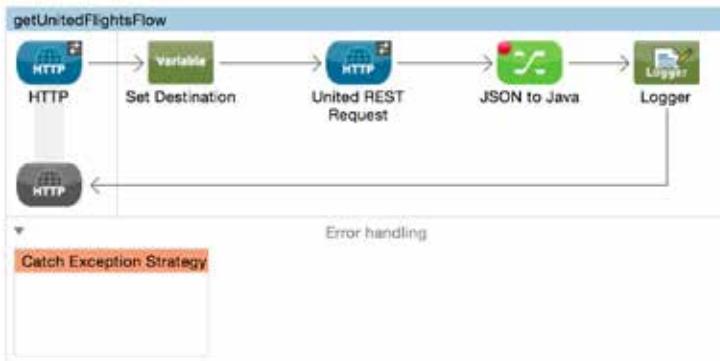
1. Return to `getFlights.xml` and debug the application.
2. Make a request to <http://localhost:8081/united?code=LAX>.
3. Step through the application and make sure you get flight results.
4. Make a request to <http://localhost:8081/united?code=FOO>.
5. Step through the application and when you get the error, drill-down into the `exceptionThrown` object in the debugger.



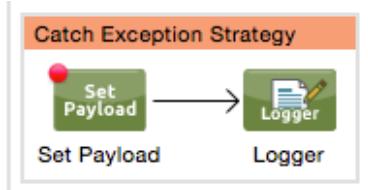
6. Click the Resume button.

## Add a catch exception strategy

7. Return to getUnitedFlightsFlow and click the arrow to expand the Error handling section.
8. Drag and drop a Catch Exception Strategy from the Error Handling section of the palette into the Error handling section of the flow.



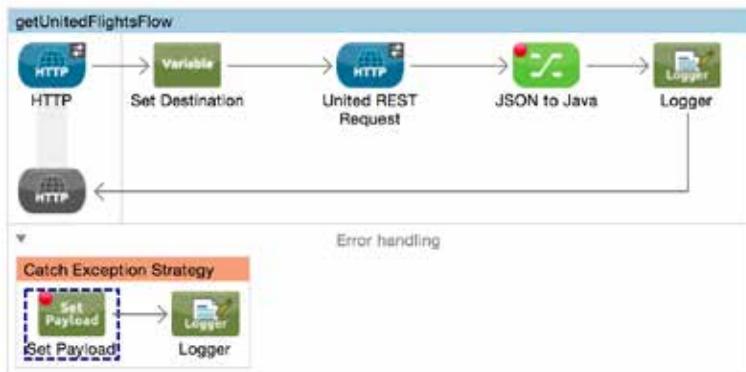
9. Add a Set Payload transformer to the Catch Exception Strategy.
10. In the Set Payload Properties view, set the value the following MEL expression:  
NO FLIGHTS to #[flowVars.destination]. ERROR: #[exception]
11. Add a breakpoint to the transformer.
12. Add a Logger after the transformer.



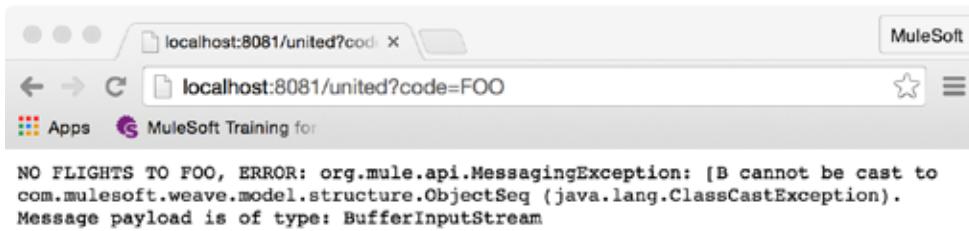
## Test the application

13. Save the file and debug the application.
14. Make another request to <http://localhost:8081/united?code=FOO>.

15. Step through the application; you should see the exception thrown in getUnitedFlightsFlow and handled by the exception handler.



16. Step to the end of the application; you should see your message in the browser window.



## Make a RESTful web service request error

17. Navigate to the United REST Request endpoint in getUnitedFlightsFlow.  
18. Change the path to `/{destination}`.

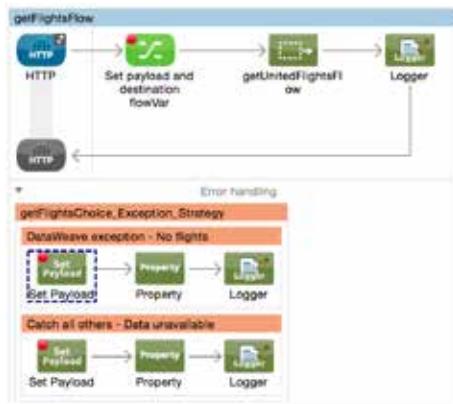
## Test the application

19. Save the file and debug the application.  
20. Make a request to <http://localhost:8081/united>.  
21. Step to the end of the application; the exception is handled by the same exception handler.

## Walkthrough 7-2: Handle multiple messaging exceptions

In this walkthrough, you will handle multiple types of exceptions thrown by the United flow. You will:

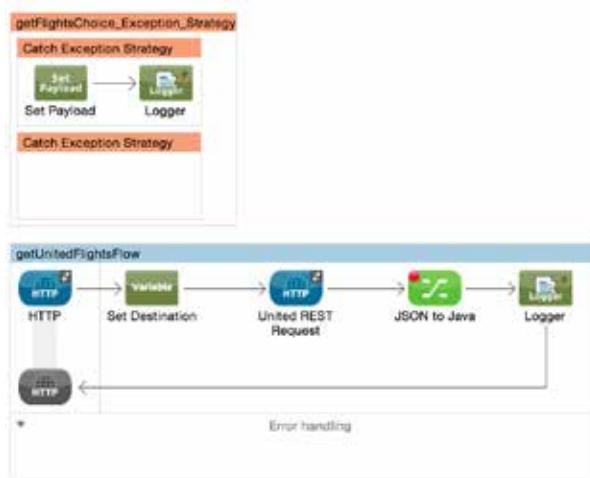
- Add and configure a Choice Exception Strategy.
- Set HTTP status codes in the exception handler.
- Let an exception bubble up and be handled by the calling flow.



### Add a choice exception strategy

1. Drag a Choice Exception Strategy from the palette and drop it above getUnitedFlightsFlow.
2. Drag the Catch Exception Strategy from getUnitedFlightsFlow and drop it in the Choice Exception Strategy.
3. Drag a second Catch Exception Strategy from the palette and drop it in the Choice Exception Strategy.

*Note: If you are having difficulty adding it, try dropping it on the orange banner of the Catch Exception Strategy.*



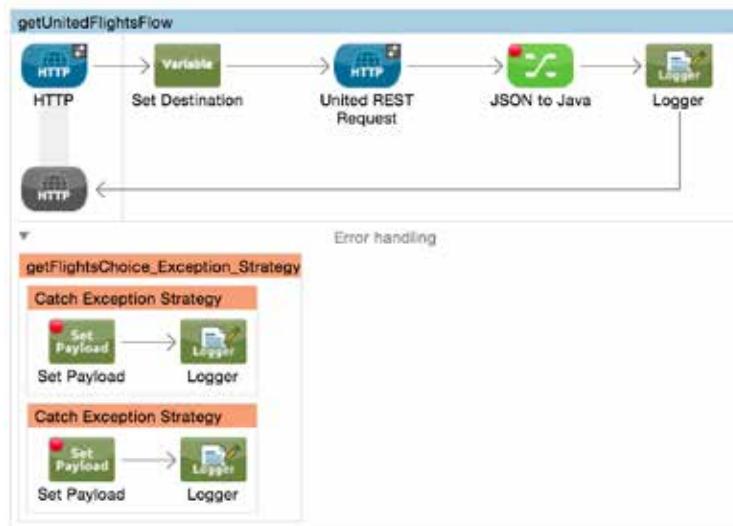
4. Drag the Choice Exception Strategy and drop it in the Error handling section of getUnitedFlightsFlow.

*Note: If you are having difficulty adding it because the canvas is scrolling, try changing the order of the flows on the canvas and then dragging and dropping.*

5. Add a Set Payload transformer and a Logger to the new Catch Exception Strategy.
6. In the Set Payload Properties view, set the value the following MEL expression:

```
DATA IS UNAVAILABLE. TRY LATER. ERROR: #[exception]
```

7. Add a breakpoint to the transformer.



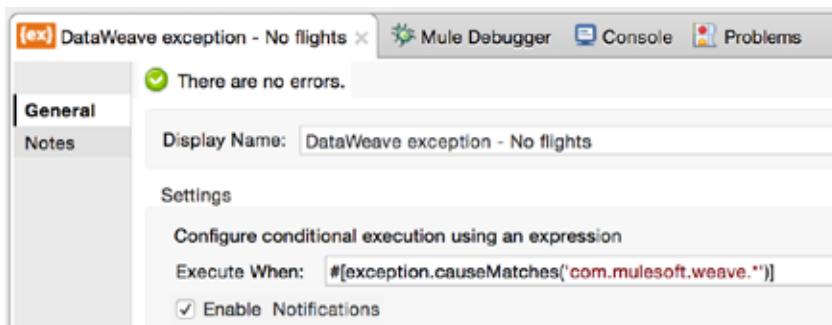
## Configure the choice exception strategy

8. Set the name of the first Catch Exception Strategy to DataWeave exception - No flights.
9. Set the name of the second Catch Exception Strategy to Catch all others - Data unavailable.



10. In the Properties view for the first Catch Exception Strategy, add an expression so it executes when a com.mulesoft.weave.\* exception is thrown; use the causeMatches() method.

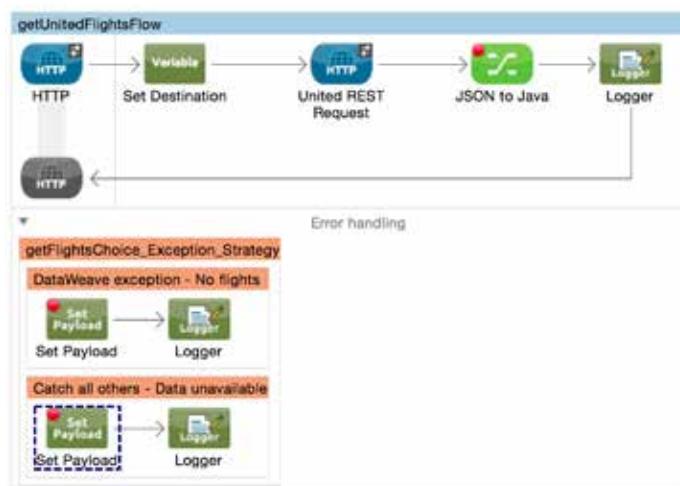
```
##[exception.causeMatches('com.mulesoft.weave.*')]
```



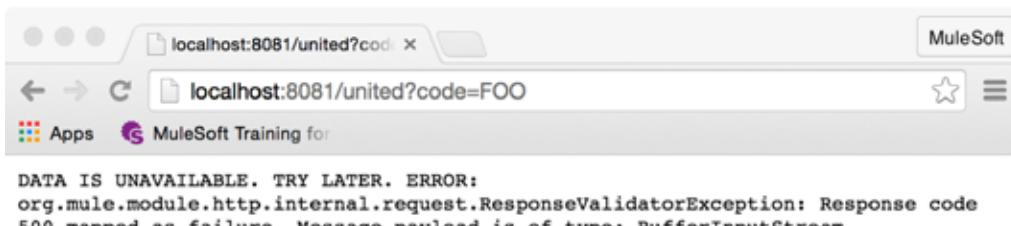
11. In the Properties view for the second Catch Exception Strategy, do not set an execute when expression.

## Test the application

12. Save the file and debug the application.
13. Make another request to <http://localhost:8081/united>.
14. Step through the application; the exception should be handled by the second catch block.



15. Step to the end of the application; you should see the second error message.

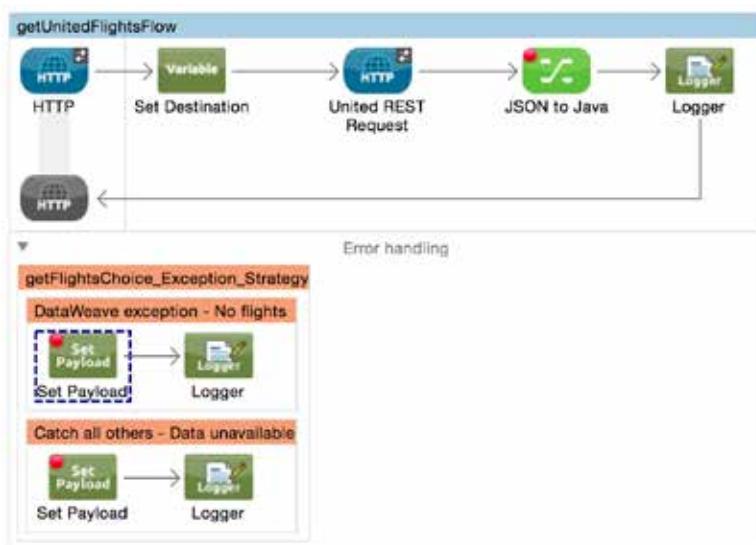


## Fix the RESTful web service request error

16. Navigate to the United REST Request endpoint in getUnitedFlightsFlow.
17. Change the path back to /{destination}.

## Test the application

18. Save the file and debug the application.
19. Make a request to <http://localhost:8081/united?code=FOO>.
20. Step through the application; the exception should be handled by the first Catch Exception Strategy.



21. Step to the end of the application; you should see the first error message.

```
NO FLIGHTS TO FOO. ERROR: org.mule.api.MessagingException: Exception while
executing:
payload.flights map {

Type mismatch
found :name, :binary
required :name, :object
(com.mulesoft.weave.mule.exception.WeaveExecutionException). Message payload is
of type: BufferInputStream
```

22. If you are using Postman, locate the http status code value.

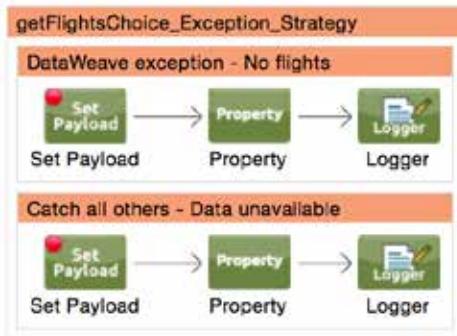
The screenshot shows the Postman interface with the following details:

- Header Bar:** Shows "Builder", "Runner", "Import", "Disabled", "Sign In", and "Supporters".
- Request URL:** http://localhost:8081/united?code=FOO
- Method:** GET
- Params:** None
- Send:** Button to send the request.
- Authorization:** Set to "No Auth".
- Headers:** (0)
- Body:** (Empty)
- Pre-request script:** (Empty)
- Tests:** (Empty)
- Status:** 200 OK (though the response indicates an error)
- Time:** 4883 ms
- Response Body (Pretty):**

```
f 1 NO FLIGHTS TO FOO. ERROR: org.mule.api.MessagingException: Exception while executing:
2 payload.flights map {
3
4 Type mismatch
5     found :name, :binary
6     required :name, :object (com.mulesoft.weave.mule.exception.WeaveExecutionException). Message
payload is of type: BufferInputStream
```
- Buttons:** "Pretty", "Raw", "Preview", "HTML", "Copy", "Search", "Scroll to response".

## Set http status codes

23. Add a Property transformer before the Logger in both Catch Exception Strategies.



24. In the first Property transformer Properties view, set the property http.status to 400.

The screenshot shows the Mule Studio Properties view for a Property transformer:

- General Tab:** Selected.
- Notes:** There are no errors.
- Display Name:** Property
- Settings:**
  - Operation:** Set Property (radio button selected)
  - Remove Property
  - Copy Properties
- Name:** http.status
- Value:** 400

25. In the second Property transformer Properties view, set the property http.status to 500.

## Test the application

26. Save the file and debug the application.

27. Make a request to <http://localhost:8081/united?code=FOO>.

28. Step through the application; you should see the outbound http.status property set to 400.

Inbound Variables Outbound Session Record		
Name	Value	Type
④ http.status	400	java.lang.String

29. Step to the end of the application; if you are using Postman, you should see the new http status code value.

The screenshot shows the Postman interface. At the top, it says "Builder" and "Runner". Below that is a toolbar with "Import" and other icons. The URL bar shows "http://localhost:8081...". The main area has tabs for "Params", "Send", and "Tests". Under "Authorization", it says "No Auth". The "Body" tab is selected, showing a JSON response:

```
f 1 NO FLIGHTS TO FOO. ERROR: org.mule.api.MessagingException: Exception while executing:  
2 payload.flights map {  
3   ^  
4     Type mismatch  
5       found :name, :binary  
6       required :name, :object (com.mulesoft.weave.mule.exception.WeaveExecutionException). Message  
payload is of type: BufferInputStream
```

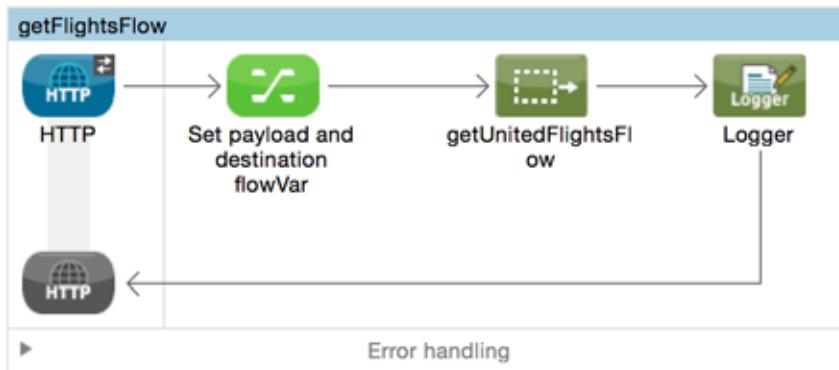
At the bottom right of the body panel, there is a "Scroll to response" button.

## Call the United flow from another flow

30. Locate getFlightsFlow.

31. Add a Flow Reference before the Logger.

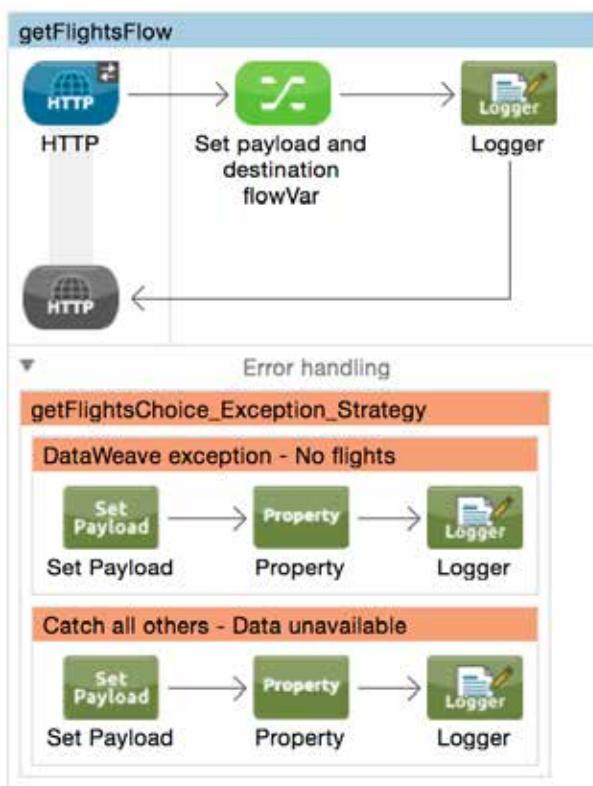
32. Set the flow name for the Flow Reference to getUnitedFlightsFlow.



## Move a catch exception strategy to the calling flow

33. Expand the Error handling section of getFlightsFlow.

34. Move the Choice Exception Strategy from getUnitedFlightsFlow to getFlightsFlow.



## Test the application

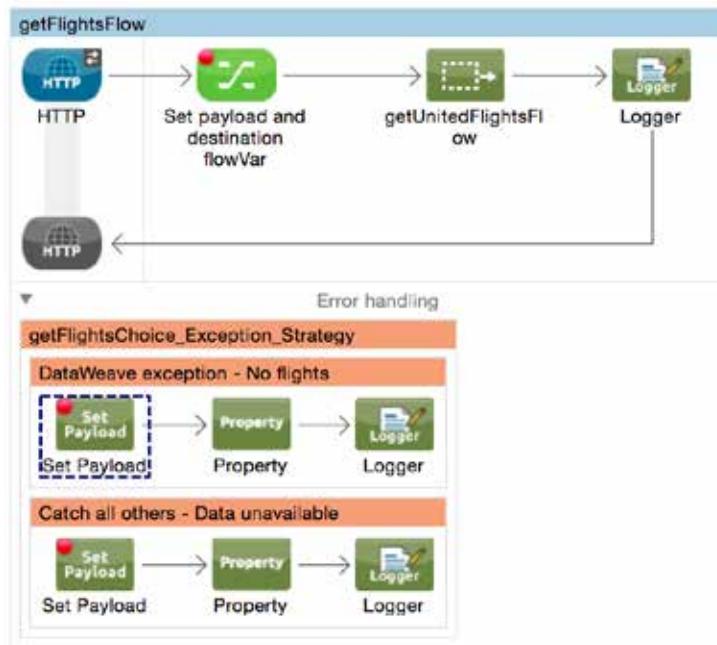
35. Save the file and debug the application.

36. Make sure there are breakpoints in the flow and the two Catch Exception Strategies.

37. Make a request to <http://localhost:8081/flights> and submit the form.

38. Step through the application; you should see the United flow called and the exception caught by the calling flow.

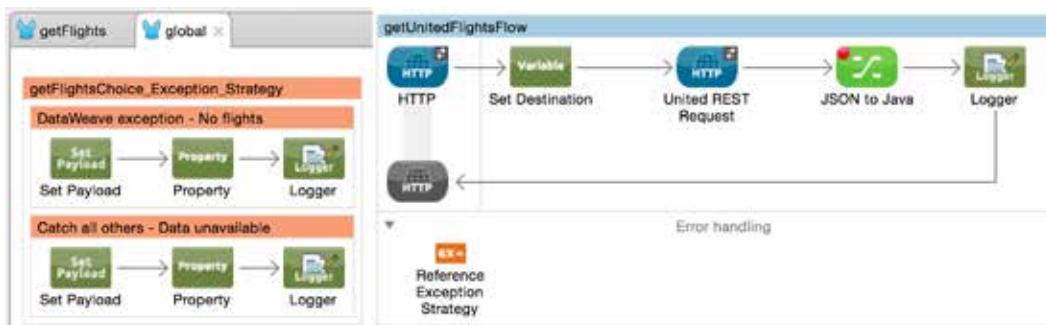
*Note: You will have to click the Resume button to move into the exception strategy after the exception is thrown by the United flow.*



## Walkthrough 7-3: Create and use global exception handlers

In this walkthrough, you will create a global exception handler in the MUA application. You will:

- Create a global exception handler.
- Reference and use the global exception handler in flows.



### Test the application for when Delta returns no results

1. Return to getFlights.xml and debug the application.
2. Make a request to <http://localhost:8081/delta?code=LAX>.
3. Step through the application and make sure you get flight results.
4. Make a request to <http://localhost:8081/delta?code=FOO>.
5. Step through the application and when you get the error, drill-down into the exceptionThrown object in the debugger.



6. Click the Resume button.

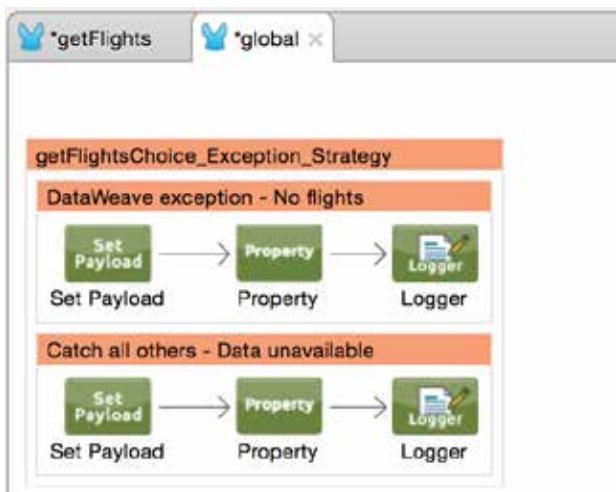
### Create a global exception handler

7. In getFlights.xml, switch to the Configuration XML view.

8. Locate the Choice Exception Strategy and select it and cut it.

```
<flow-ref name="getUnitedFlightsFlow" doc:name="getUnitedFlightsFlow"
<logger level="INFO" doc:name="Logger" />
<choice-exception-strategy doc:name="getFlightsChoice_Exception_Strate
<catch-exception-strategy
    when="#[exception.causeMatches('com.mulesoft.weave.*')]" doc:r
        <set-payload
            value="NO FLIGHTS TO #[flowVars.destination]. ERROR: #[exc
                doc:name="Set Payload" />
            <set-property propertyName="http.status" value="400"
                doc:name="Property" />
        <logger level="INFO" doc:name="Logger" />
    </catch-exception-strategy>
    <catch-exception-strategy doc:name="Catch all others - Data unava
        <set-payload value="DATA IS UNAVAILABLE. TRY LATER. ERROR: #[e
            doc:name="Set Payload" />
        <set-property propertyName="http.status" value="500"
            doc:name="Property" />
        <logger level="INFO" doc:name="Logger" />
    </catch-exception-strategy>
</choice-exception-strategy>
</flow>
```

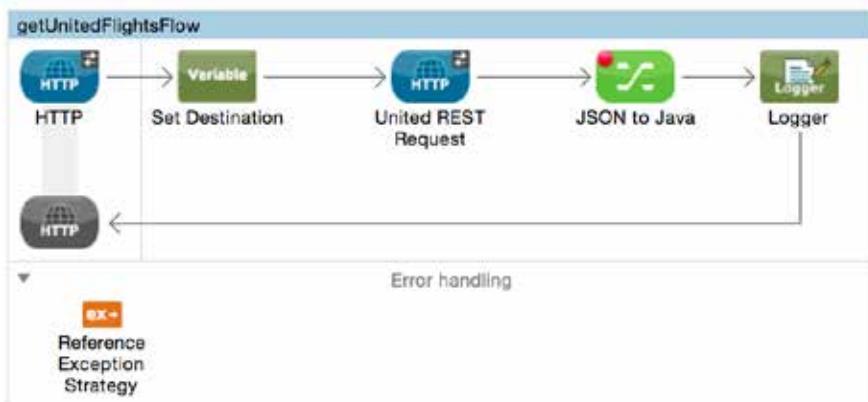
9. Open global.xml and go to the Configuration XML view.
10. Return to getFlights.xml and cut the getFlightsChoice\_Exception\_Strategy from the getFlightsFlow.
11. Paste the Choice Exception Strategy after the global configuration elements inside the start and end mule tags.
12. Switch to the Message Flow view; you should see the Choice Exception Strategy.



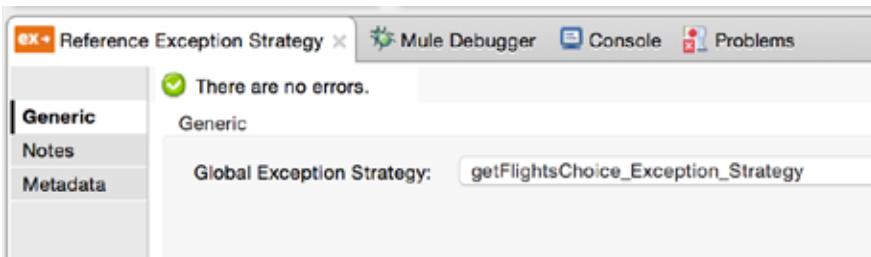
## Use the global exception handler

13. Return to getFlights.xml and switch to the Message Flow view.
14. Locate the getUnitedFlightsFlow and expand its Error handling section.

15. Drag a Reference Exception Strategy from the palette and drop it in the Error handling section.



16. In the Properties view, set the global exception strategy to getFlightsChoice\_Exception\_Strategy.



17. Drag a Reference Exception Strategy to the error handling section of getDeltaFlightsFlow.

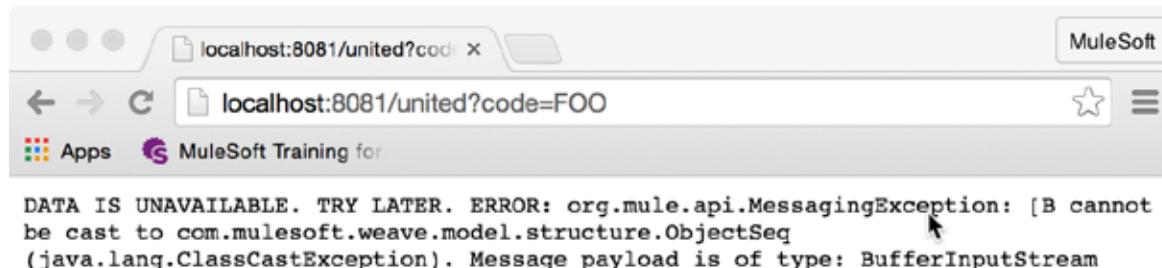
18. In the Properties view, set the global exception strategy to getFlightsChoice\_Exception\_Strategy.

*Note: You could add a Reference Exception Strategy to the rest of the flows, but instead, you will create a global default exception strategy in the next walkthrough.*

## Test the application

19. Save all the files and run the application.

20. Make a request to <http://localhost:8081/united?code=FOO>; you should see the data unavailable error message displayed.



21. Make a request to <http://localhost:8081/delta?code=FOO>; you should see the data unavailable error message displayed.



## Walkthrough 7-4: Specify a global default exception strategy

In this walkthrough, you will change the default exception handling for the application. You will:

- Create a global configuration element in the global.xml file.
- Specify a default exception strategy in the global configuration element.
- Remove the existing exception handling strategies.
- Use the default exception handling strategy.

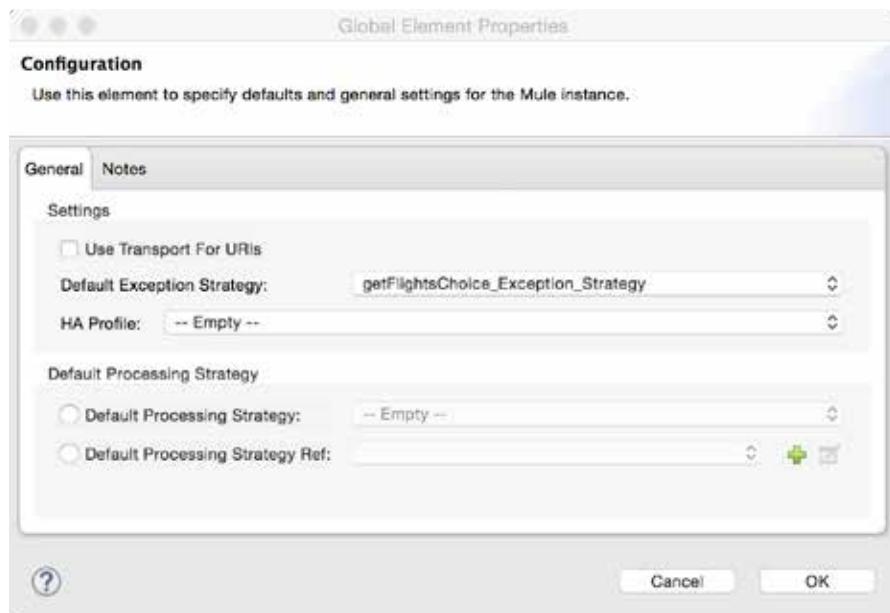


### Specify a global default exception strategy

1. Return to global.xml.
2. Switch to the Global Elements view.
3. Click the Create button.
4. In the Choose Global Type dialog box, select Global Configurations > Configuration and click OK.



5. In the Global Element Properties dialog box, set the default exception strategy to globalChoice\_Exception\_Strategy and click OK.



6. Locate your new global configuration element.

Type	Name
HTTP Listener Configuration	HTTP_Listener_Configuration
HTTP Request Configuration	United_HTTP_Request_Configuration
HTTP Request Configuration	Bank_REST_Request_Configuration
Web Service Consumer	Delta_Web_Service_Consumer
MySQL Configuration	Training_SQL_Configuration
Salesforce: Basic authentication	Salesforce
Configuration	Configuration

## Remove the existing exception strategy references

7. Return to getFlights.xml.
8. Delete the Reference Exception Strategy in getUnitedFlightsFlow.
9. Delete the Reference Exception Strategy in getDeltaFlightsFlow.

## Test the application

10. Save all the files and run the application.
11. Make a request to <http://localhost:8081/united?code=FOO>; you should still see the no flights error message displayed.



```
Type mismatch
  found :name, :binary
  required :name, :object (com.mulesoft.weave.mule.exception.WeaveExecutionException).
Message payload is of type: BufferInputStream
```

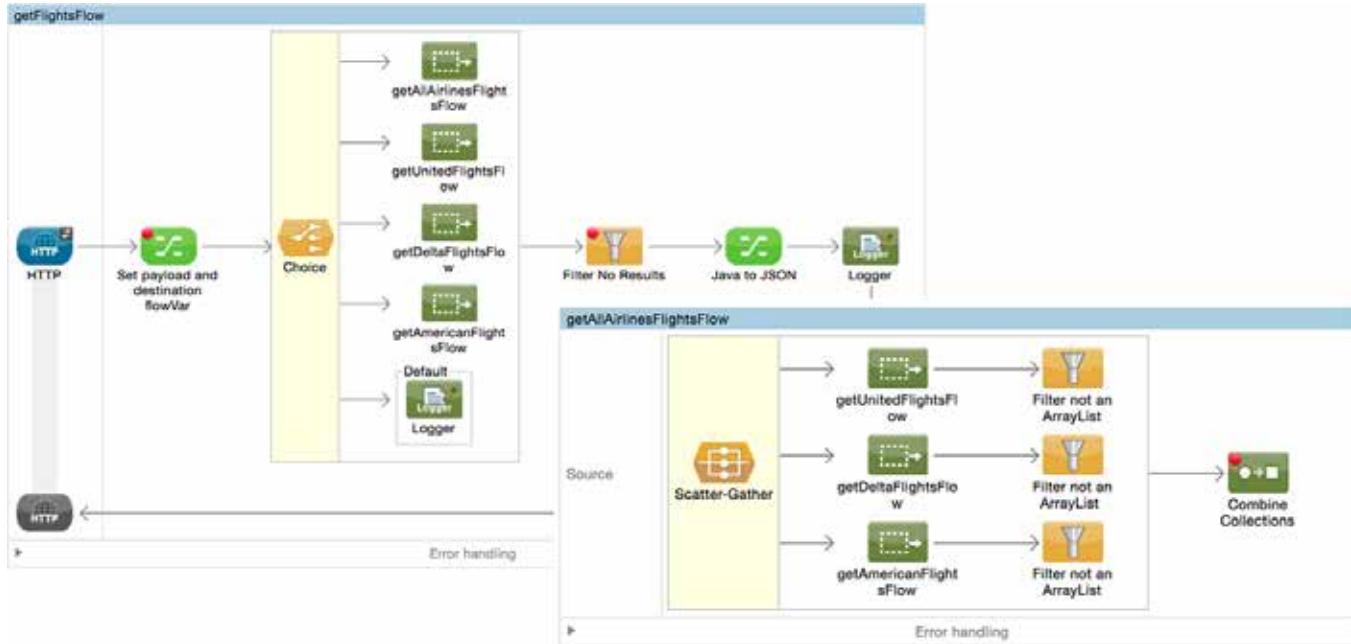
12. Make a request to <http://localhost:8081/delta?code=FOO>; you should still see the no flights error message displayed.



```
Cannot coerce a :string to a :object
(com.mulesoft.weave.mule.exception.WeaveExecutionException). Message payload is
of type: DepthXMLStreamReader
```

13. Stop the Mule runtime.

# Module 8: Controlling Message Flow



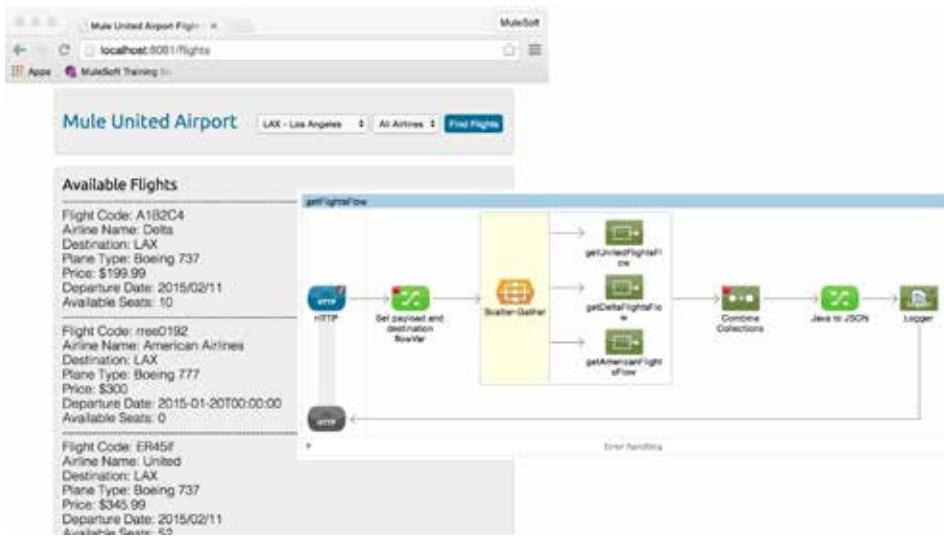
In this module, you will learn:

- About flow control and filter elements.
- To multicast a message.
- To route message based on conditions.
- To filter messages.
- About synchronous and asynchronous flows.
- To create an asynchronous flow.

## Walkthrough 8-1: Multicast a message

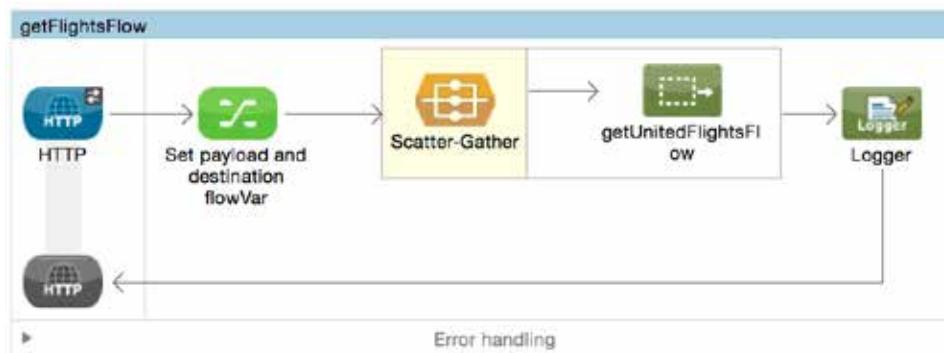
In this walkthrough, you will create one flow that calls each of the three airline services and combines and returns the results back to the web form. You will:

- Use a Scatter-Gather router to concurrently call all three flight services.
- Use a Combine Collections transformer to combine a collection of three ArrayLists of objects into one collection.
- Use DataWeave to sort the flights and return them as JSON to the form.



### Add a router to call all three airline services

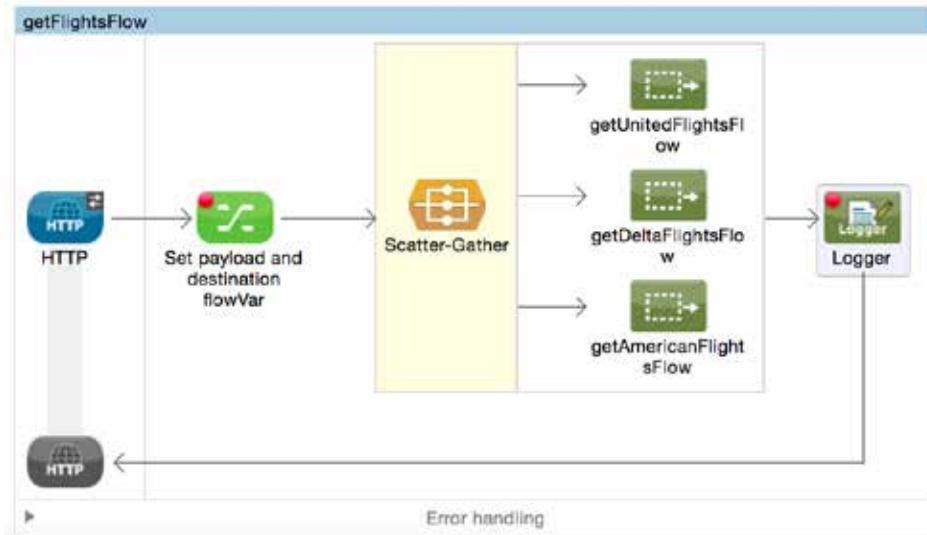
1. Return to getFlights.xml.
2. Add a Scatter-Gather flow control element before the Flow Reference in getFlightsFlow.
3. Drag the Flow Reference component into the Scatter-Gather router.



4. Add a second Flow Reference component to the Scatter-Gather router.
5. In the Properties view for the Flow Reference, set the flow name to getDeltaFlightsFlow.
6. Add a third Flow Reference component to the Scatter-Gather router.



7. In the Properties view for the Flow Reference, set the flow name to getAmericanFlightsFlow.
8. Add a breakpoint to the Set payload and destination flowVar component.
9. Add a breakpoint to the Logger.
10. Save the file.



## Test the application

11. Debug the application.
12. Make a request to <http://localhost:8081/flights> and submit the form.
13. Step through the application to the Logger; you should step through each of the airline flows.
14. Look the Mule Debugger view and see what the data type of the payload is after the router.

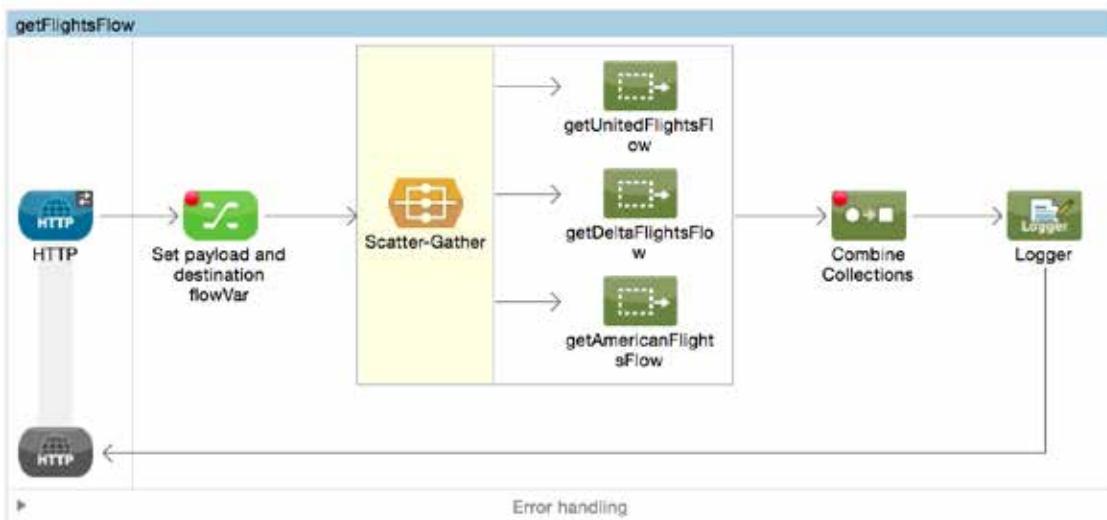
Name	Type
message Processor	org.mule.api.processor.LoggerMessageProcessor
payload	java.util.concurrent.CopyOnWriteArrayList
0	[com.mulesoft.training.Flight@7fed6...]
1	[com.mulesoft.training.Flight@78f59...]
2	[com.mulesoft.training.Flight@16512...]
3	[com.mulesoft.training.Flight@57a10c4...]
4	[com.mulesoft.training.Flight@4894eb12...]
airlineName	java.lang.String
availableSeats	java.lang.Integer
departureDate	java.lang.String
destination	java.lang.String
flightCode	java.lang.String

*Note: If you step through to the end of the application, ignore any errors you get.*

15. Click the Resume button.

## Flatten the results

16. Add a Combine Collections transformer before the Logger.



## Test the application

17. Save the file to redeploy the application.
18. Make a request to <http://localhost:8081/flights> and submit the form.
19. Step through the application to the Logger after the router; you should see the payload is now one ArrayList of HashMaps.

The screenshot shows the Mule Properties view with the following details:

Name	Value	Type
Message		org.mule.DefaultMuleMessage
Message Processor	Logger	org.mule.api.processor.LoggerMess...
payload	<pre>[{"airlineName": "United", "departureDate": "..."}, {"airlineName": "United", "departureDate": "..."}, {"airlineName": "United", "departureDate": "..."}, {"airlineName": "American Airlines", "de..."}, {"airlineName": "United", "departureDate": "..."}, {"airlineName": "Delta", "departureDate": ...}]</pre>	java.util.ArrayList
0	<pre>{"airlineName": "United", "departureDate": "..."}</pre>	java.util.LinkedHashMap
1	<pre>{"airlineName": "United", "departureDate": "..."}</pre>	java.util.LinkedHashMap
10	<pre>{"airlineName": "United", "departureDate": "..."}</pre>	java.util.LinkedHashMap
2	<pre>{"airlineName": "American Airlines", "de..."}</pre>	java.util.LinkedHashMap
3	<pre>{"airlineName": "United", "departureDate": "..."}</pre>	java.util.LinkedHashMap
	<pre>{"airlineName": "Delta", "departureDate": ...}</pre>	java.util.LinkedHashMap

20. Step through to the end of the application; you should see a jumble of data returned.

```
srjava.util.ArrayList{size=1093}mulesoft.training.Flight{B+}{Ia:departureDate=q-Ldestination=q-L flightCode=q-Lorigin=q-LplaneType=q-xp=a@tAmerican Airlinest 2015/02/11tSFOTree1093tMUAt Boeing 737sq~ @r@tDelta 2015/02/12tSFOTreeA14244tMUAt Boeing 787sq~ @r@tAmerican Airlinest 2015/02/20tSFOTree2000tMUAt Boeing 737sq~ @ytUnitedt 2015/03/20tSFOTER38sdstMUAt Boeing 737sq~ (@ytDelta 2015/03/20tSFOTreeA1B2C3tMUAt Boeing 737sq~d@tAmerican Airlinest 2015/01/20tSFOTree4567tMUAt Boeing 737sq~ @t@tDelta 2015/02/12tSFOTreeA1BTT4tMUAt Boeing 777sq~ @tAmerican Airlinest 2015/01/01tSFOTree1994tMUAt Boeing 777sq~ @tAmerican Airlinest 2015/02/01tSFOTree3000tMUAt Boeing 737sq~6@t@tUnited 2015/09/11tSFOTER39rktMUAt Boeing 757sq~ @t@tUnited 2015/02/12tSFOTER39rjtMUAt Boeing 777x
```

## Review HTML form code to see what data format it expects

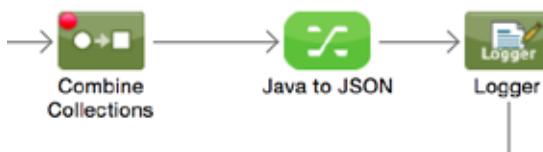
21. Open FlightFinder.html in src/main/resources and locate in what format it expects the data to be sent back.

```
132
133     if (ajaxRequest.status == 200) {
134         var response = ajaxRequest.responseText;
135         document.getElementById("myDiv").style.display = "block";
136
137         try {
138             var transformed = JSON.parse(response);
139
140             document.getElementById("myDiv").innerHTML = "<h2>Available Flights</h2>";
141             for ( var i = 0; i < transformed.length; i++) {
142                 document.getElementById("myDiv").innerHTML += "-----";
143                 + transformed[i].flightCode + "<br>";
144                 document.getElementById("myDiv").innerHTML += "Airline Name: "
145                 + transformed[i].airlineName + "-----";
```

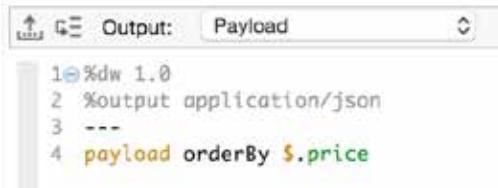
## Return the data as JSON and sort by price

22. In getFlights.xml, add a Transform Message component after the Combine Collections transformer.

23. Change its name to Java to JSON



24. In the Java to JSON Properties view, change the output type to application/json.
25. Write a DataWeave expression to transform the payload and order by price.



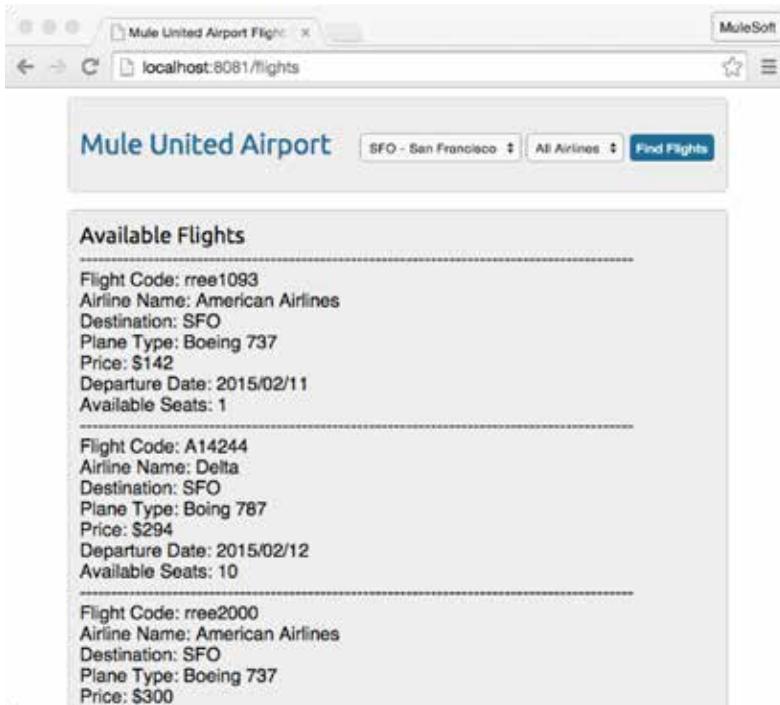
```

↑ %dw 1.0
2 %output application/json
3 ---
4 payload orderBy $.price

```

## Test the application

26. Save the file and run the application.
27. Make a request to <http://localhost:8081/flights> and submit the form; you should see SFO flights for all three airlines returned and the flights should be sorted by price.



**Mule United Airport**

SFO - San Francisco | All Airlines | Find Flights

**Available Flights**

Flight Code: ree1093	Airline Name: American Airlines
Destination: SFO	Plane Type: Boeing 737
Price: \$142	Departure Date: 2015/02/11
Available Seats: 1	
Flight Code: A14244	Airline Name: Delta
Destination: SFO	Plane Type: Boeing 787
Price: \$294	Departure Date: 2015/02/12
Available Seats: 10	
Flight Code: ree2000	Airline Name: American Airlines
Destination: SFO	Plane Type: Boeing 737
Price: \$300	

28. Submit the form for different destinations; flights for SFO are returned every time.

## Use the destination in the airline flows

29. Locate the Set Destination transformer in the getAmericanFlightsFlow.

*Note: There are two options at this point: 1) to remove the HTTP endpoints and Set Destination transformers from each of the airline flows or 2) to use a more complicated expression to check and see if a destination flow variable already exists so that the individual airline flows can still be called and tested individually.*

30. Modify the value so it only uses the expression to set the destination flowVars variable if it does not already exist.

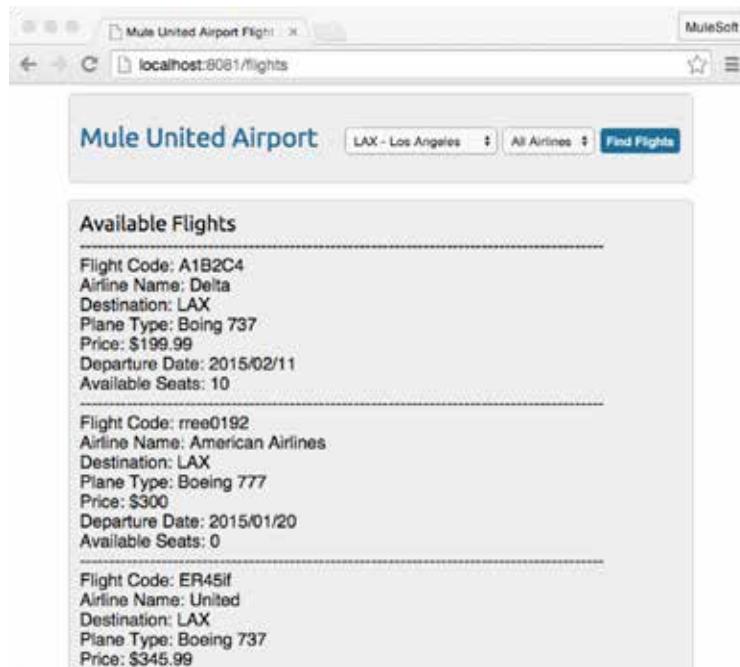
```
#[(flowVars.destination ==empty &&
message.inboundProperties.'http.query.params'.code == empty) ? 'SFO' :
(flowVars.destination != empty ? flowVars.destination :
message.inboundProperties.'http.query.params'.code)]
```

*Note: This expression is located in the course snippets.txt file and can be copied from there.*

31. Use the same expression in the Set Destination in getDeltaFlightsFlow and getUnitedFlightsFlow.

## Test the application

32. Save the file to redeploy the application.
33. Make a request to <http://localhost:8081/flights> and submit the form; you should still see SFO flights.
34. Submit the form for a different destination including LAX, CLE, or PDX; you should see the correct flights returned.



The screenshot shows a web browser window with the title "Mule United Airport Flight". The address bar displays "localhost:8081/flights". The main content area is titled "Mule United Airport" and shows flight search results for "LAX - Los Angeles" with "All Airlines". There are three flight entries listed:

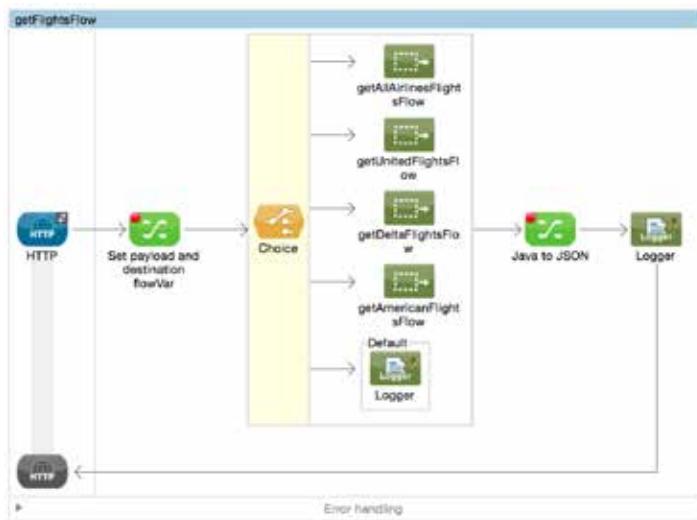
Flight Code	Airline Name	Destination	Plane Type	Price	Departure Date	Available Seats
A1B2C4	Delta	LAX	Boeing 737	\$199.99	2015/02/11	10
rree0192	American Airlines	LAX	Boeing 777	\$300	2015/01/20	0
ER45if	United	LAX	Boeing 737	\$345.99		

35. Submit the form for a destination of PDF or FOO; you should get an error in the console but no return error message to the form.
36. Submit the form for SFO, LAX, CLE, or PDX and a different airline; you should still see flights for all three airlines returned.

## Walkthrough 8-2: Route messages based on conditions

In this walkthrough, you will create a flow that routes messages based on conditions. You will:

- Use a Choice router to get flight results for all three airlines or only a specific airline.
- Set the router paths based on the airline value sent from the flight form.



### Look at selected airline values in HTML form

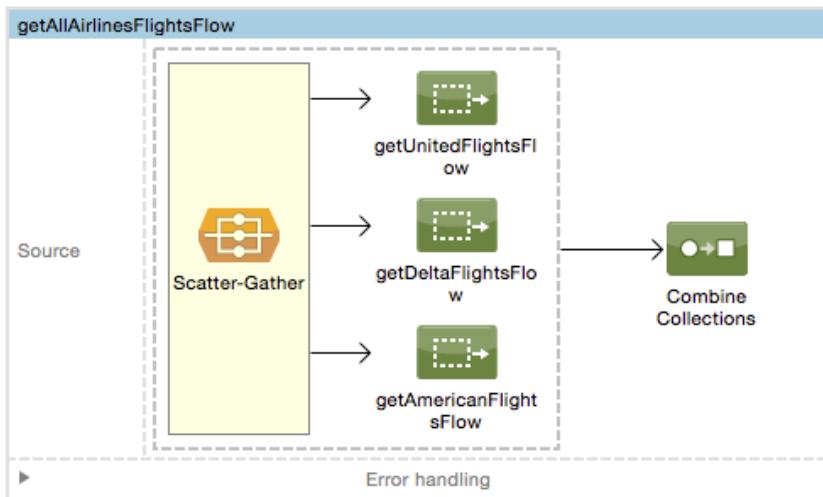
1. In Anypoint Studio, return to FlightFinder.html.
2. Locate the select box that sets the airline and see what values are set and returned.

```
190<select id="airline" name="airline" class="select2">
191    <option value="all">All Airlines</option>
192    <option value="united">United</option>
193    <option value="delta">Delta</option>
194    <option value="american">American</option>
195</select>
```

### Move the Scatter-Gather to a separate flow

3. Return to getFlights.xml.
4. Shift+click to select the Scatter-Gather router and the Combine Collections transformer.
5. Right-click and select Extract to > Flow.
6. In the Extract Flow dialog box, set the flow name to getAllAirlinesFlightsFlow.

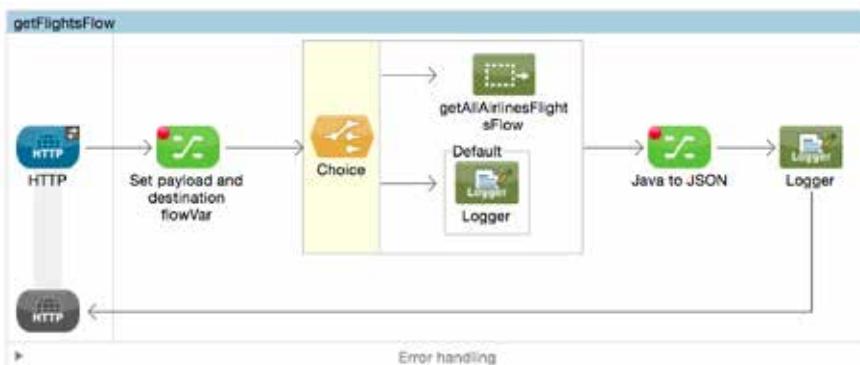
- Leave the target Mule configuration set to current and click OK.



- In `getFlightsFlow`, double-click the new Flow Reference.
- Change its display name to `getAllAirlinesFlightsFlow`.
- Move `getAllAirlinesFlightsFlow` beneath `getFlightsFlow`.
- Add a breakpoint to the `Combine Collections` transformer.

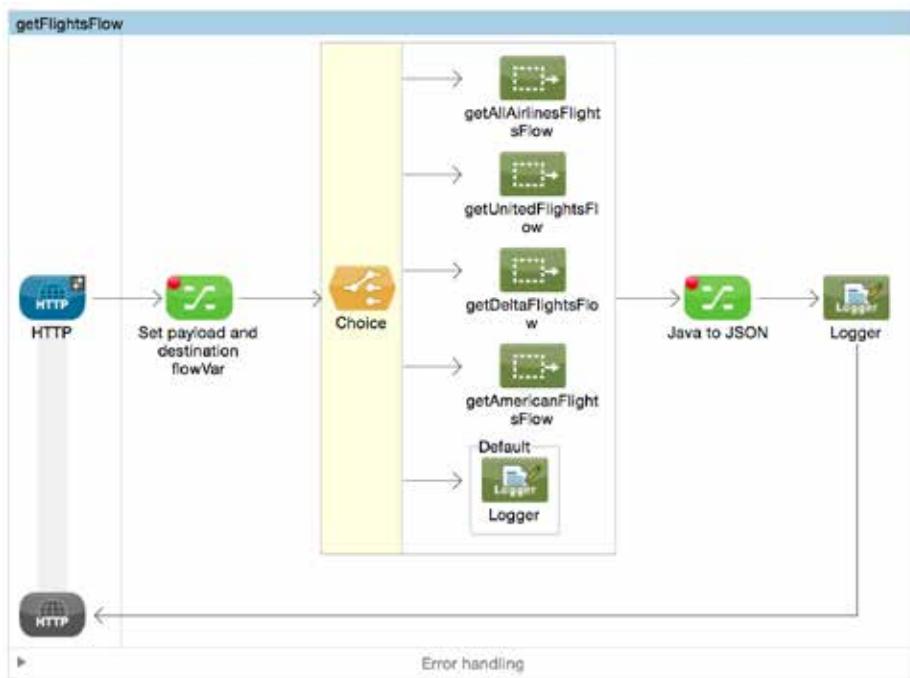
## Add a Choice router

- In `getFlightsFlow`, add a Choice flow control element between the Set payload and destination `flowVar` component and the `getAllAirlinesFlightsFlow` flow reference.
- Drag the `getAllAirlinesFlightsFlow` flow reference into the router.
- Add a new Logger component to the default branch.



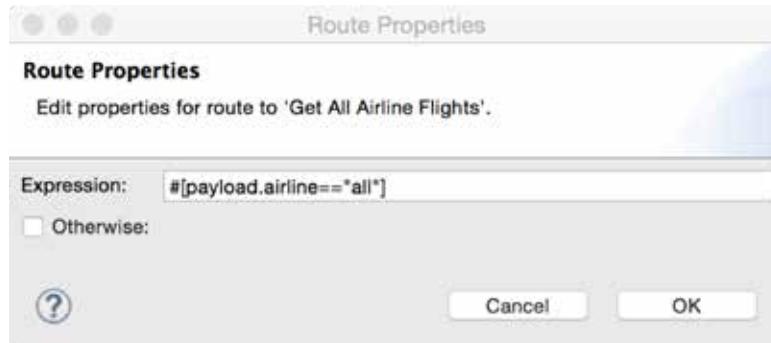
- Add three additional Flow Reference components to the Choice router to create a total of five branches.
- In the Properties view for the first new flow reference, set the flow name to `getUnitedFlightsFlow`.
- In the Properties view for the second flow reference, set the flow name to `getDeltaFlightsFlow`.

18. In the Properties view for the third flow reference, set the flow name to getAmericanFlightsFlow.



## Configure the Choice router

19. In the Properties view for the Choice router, double-click the getAllAirlinesFlightsFlow route.  
20. In the Route Properties dialog box, set the expression to #[payload.airline == "all"] and click OK.



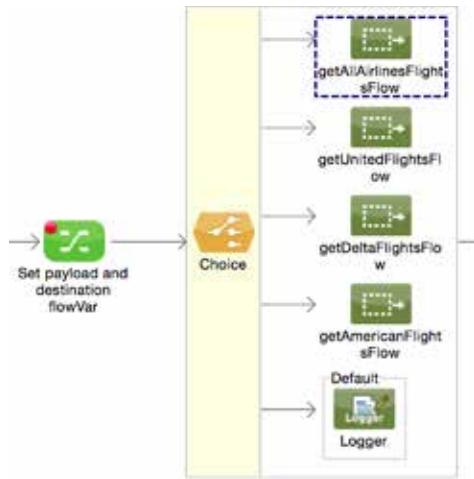
21. Set a similar expression for the United route, routing to it when payload.airline is equal to united.  
22. Set a similar expression for the Delta route, routing to it when payload.airline is equal to delta.

23. Set a similar expression for the American route, routing to it when payload.airline is equal to american.

When	Route Message to
<code>#[payload.airline == "all"]</code>	<code>getAllAirlinesFlightsFlow</code>
<code>#[payload.airline == "united"]</code>	<code>getUnitedFlightsFlow</code>
<code>#[payload.airline == "delta"]</code>	<code>getDeltaFlightsFlow</code>
<code>#[payload.airline == "american"]</code>	<code>getAmericanFlightsFlow</code>
Default	Logger

## Debug the application

24. Save the file and debug the application.
25. Make a request to <http://localhost:8081/flights>.
26. On the form page, leave SFO and All Airlines selected and click Find Flights.
27. Return to the Mule Debugger view and step through the application; you should see the Choice router pass the message to the getAllAirlinesFlightsFlow branch.



28. Click the Resume button until flight data for all airlines is returned back to the form page.
29. On the form page, select SFO and United and click Find Flights.
30. Return to the Mule Debugger view and step through the application; you should see the Choice router pass the message to the United branch.

31. Click the Resume button until flight data is returned back to the form page; you should see only United flights.

The screenshot shows a web browser window titled "Mule United Airport Flight". The URL in the address bar is "localhost:8081/flights". The page header includes "MuleSoft" and a "Find Flights" button. The main content area is titled "Available Flights" and lists three flight options:

- Flight Code: ER38sd  
Airline Name: United  
Destination: SFO  
Plane Type: Boeing 737  
Price: \$400  
Departure Date: 2015/03/20  
Available Seats: 0
- Flight Code: ER39rk  
Airline Name: United  
Destination: SFO  
Plane Type: Boeing 757  
Price: \$945  
Departure Date: 2015/09/11  
Available Seats: 54
- Flight Code: ER39rj  
Airline Name: United  
Destination: SFO  
Plane Type: Boeing 777  
Price: \$954  
Departure Date: 2015/02/12  
Available Seats: 23

## Test the application

32. Stop the debugger and run the application.
33. Make a request to <http://localhost:8081/flights>.
34. Test the Delta and American choices with SFO, LAX, or CLE; you should see the appropriate flights returned.
35. Test the PDF location with United; you should see one flight.

The screenshot shows a web browser window titled "Mule United Airport Flight". The URL in the address bar is "localhost:8081/flights". The page header includes "MuleSoft" and a "Find Flights" button. The main content area is titled "Available Flights" and lists one flight option:

- Flight Code: ER95jf  
Airline Name: United  
Destination: PDF  
Plane Type: Boeing 787  
Price: \$234  
Departure Date: 2015/02/12  
Available Seats: 23

36. Test the PDF location with Delta; you should get an error in the console.  
 37. Test the PDF location with American; you should get no error, but no results.

38. Test the PDF location with All Airlines; you should get no results and an error in the console.

## Debug the application

39. Debug the application.  
 40. Make a request to <http://localhost:8081/flights> and submit the form with PDF and All Airlines.  
 41. Step through the application to the Combine Collections transformer; you should see different types of objects returned.

Name	Value	Type
▶ <b>e</b> Message		org.mule.DefaultMessageCollection
ⓐ Message Processor	Combine Collections	org.mule.transformer.simple.Co...
▼ <b>e</b> payload	[[{"airlineName=United, departure...	java.util.concurrent.CopyOnWri...
▶ <b>e</b> 0	[{"airlineName=United, departureD...]	java.util.ArrayList
ⓐ 1	NO FLIGHTS TO PDF. ERROR: org....	java.lang.String
ⓐ 2	[]	java.util.ArrayList

42. Step to the Java to JSON component; you should see the payload still contains the error message in addition to the valid flight results to PDF.  
 43. Step to the end of the application; you should not get the United results to PDF.

*Note: You could write a custom aggregation strategy for the Scatter-Gather component with Java to handle this situation, but instead you will use the simpler approach of filtering out the exception messages.*

## Walkthrough 8-3: Filter messages

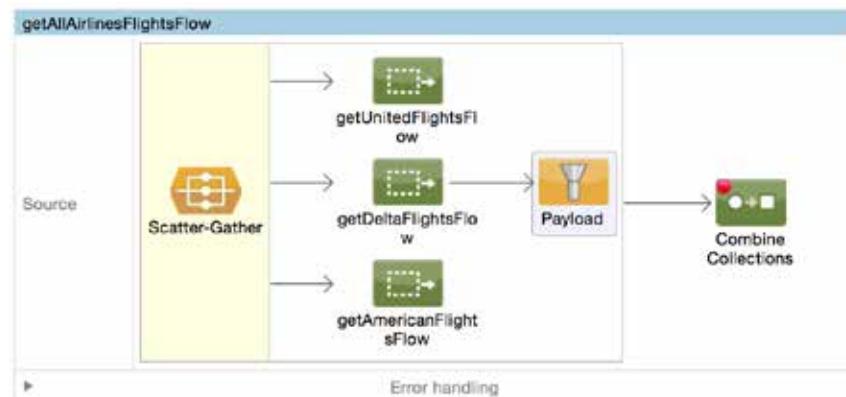
In this walkthrough, you will continue to work with the airline flight results. You will:

- Filter the results in the multicast to ensure they are ArrayLists and not exception strings.
- Use the Payload and Expression filters.
- Create and use a global filter.

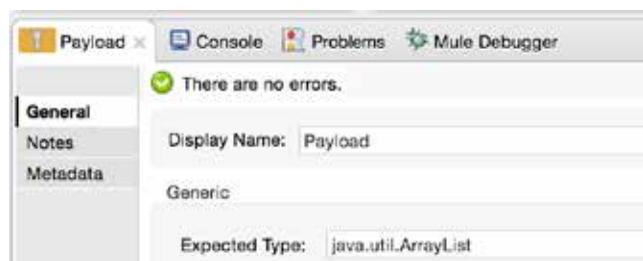


### Add a filter

1. Return to getFlights.xml.
2. Drag out a Payload Flow Control element from the palette and drop it after the getDeltaFlightsFlow reference in the Scatter-Gather.

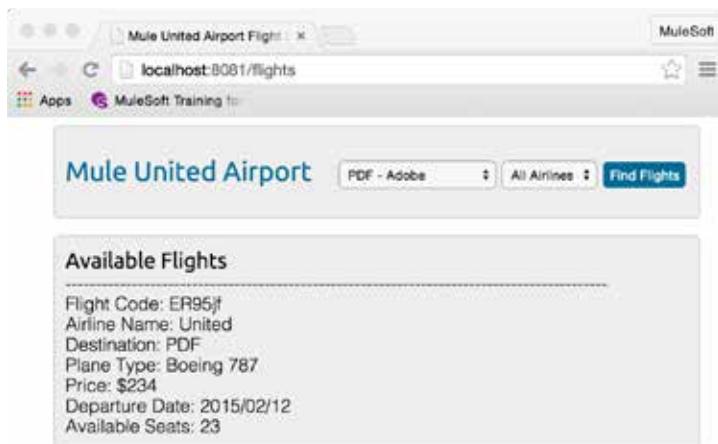


3. In the Payload Properties view, set the expected type to java.util.ArrayList.



## Test the application

4. Save the file and debug the application.
5. Make a request to <http://localhost:8081/flights> and submit the form with PDF and All Airlines.
6. Step through the Java to JSON component; this time you should see the payload only contains the ArrayList and not the error string.
7. Step to the end of the application; you should see the United results to PDF returned to the form.

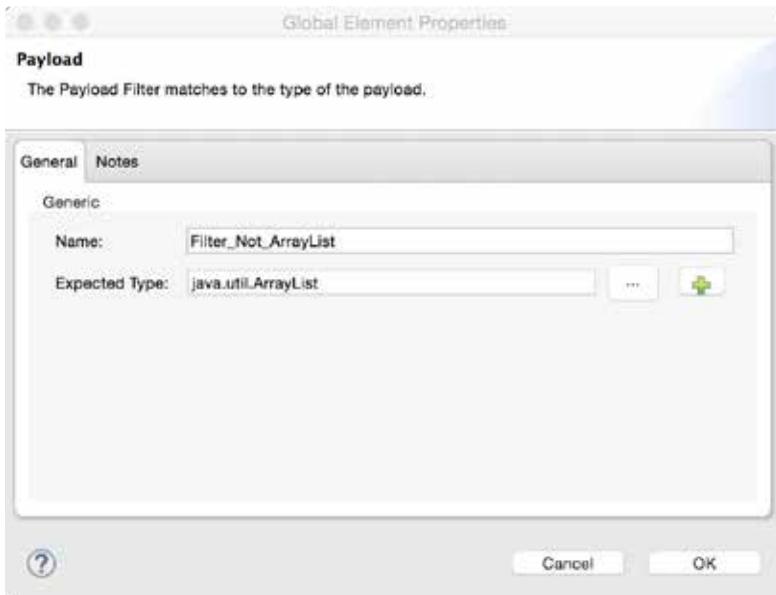


## Create a global filter

8. Delete the Payload filter.
9. Return to global.xml.
10. Switch to the Global Elements view and click Create.
11. In the Choose Global Type dialog box, select Filters > Payload and click OK.

12. In the Global Elements dialog box, set the name to Filter\_Not\_ArrayList.

13. Set the expected type to java.util.ArrayList and click OK.



14. See your new global filter in the list.

A screenshot of the 'Global Mule Configuration Elements' list. The list shows various configuration types and their names. The 'Type' column includes icons for HTTP Listener Configuration, HTTP Request Configuration, Web Service Consumer, MySQL Configuration, Salesforce: Basic authentication, Configuration, and Payload. The 'Name' column lists the corresponding names: 'HTTP\_Listener\_Configuration', 'United\_HTTP\_Request\_Configuration', 'Bank\_REST\_Request\_Configuration', 'Delta\_Web\_Service\_Consumer', 'Training\_MySQL\_Configuration', 'Salesforce', 'Configuration', and 'Filter\_Not\_ArrayList'. The 'Configuration' entry is highlighted with a blue selection bar at the top.

## Use the global filter

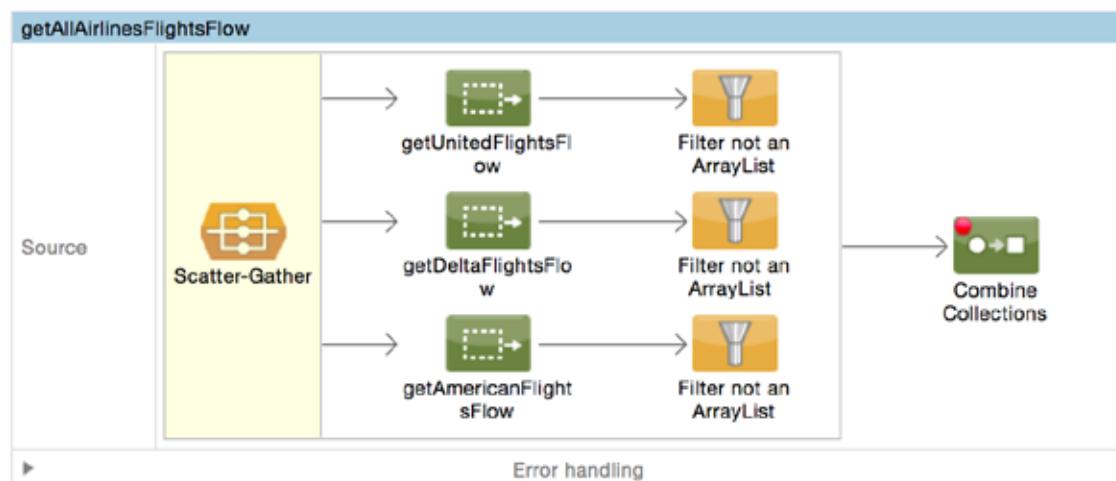
15. Return to getFlights.xml.

16. Drag a Filter Reference from the palette and drop it after getDeltaFlightsFlow in the Scatter-Gather.

17. In the Filter Reference Properties view, set the display name to Filter not an ArrayList and set the global reference to Filter\_Not\_ArrayList.



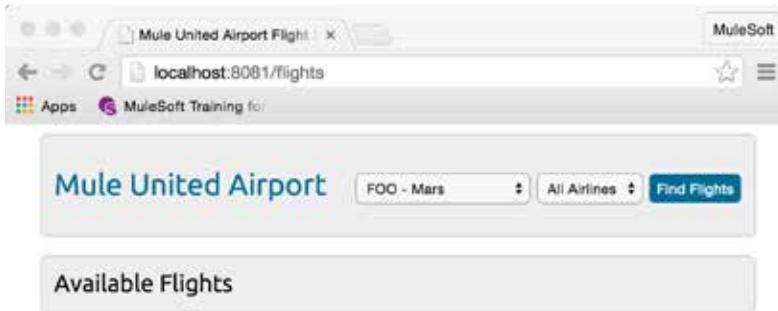
18. Add a Filter Reference after getAmericanFlightsFlow.  
19. In the Filter Reference Properties view, set the display name to Filter not an ArrayList and set the global reference to Filter\_Not\_ArrayList.  
20. Copy and paste the Filter Reference in the Scatter-Gather.  
21. Drag it after getUnitedFlightsFlow and change its name to Filter not an ArrayList.



## Test the application

22. Save all the files and run the application.  
23. Make a request to <http://localhost:8081/flights> and submit the form with PDF and All Airlines; you should still see the United results.

24. Submit the form for FOO and All Airlines; you should get no results and no message.



## Look at the HTML form and find the display if no flights are returned

25. In Anypoint Studio, return to or open FlightFinder.html.

26. Locate the code that sets the text if there is no flights are returned.

```
158     catch(e) {
159         if (response) {
160             document.getElementById("myDiv").innerHTML = response;
161         }
162     } else{
163         document.getElementById("myDiv").innerHTML = "There are no available flights";
164     }
165 }
```

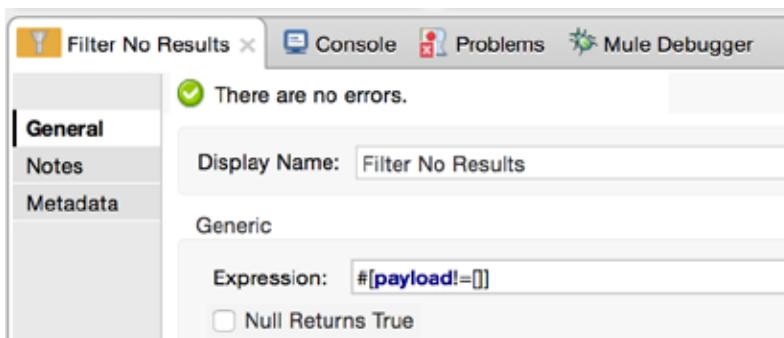
## Add an Expression filter to filter no results

27. In the getFlightsFlow, add an Expression filter after the Choice router.

28. In the Expression Properties view, set the display name to Filter No Results.

29. Set the expression to see if the payload is equal to an empty Array.

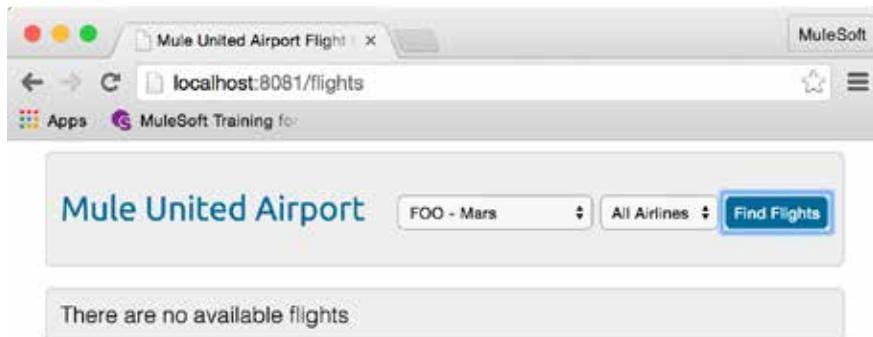
```
##[payload!=[]]
```



## Test the application

30. Save the file to redeploy the application.

31. Submit the form for FOO and All Airlines; you should see the message there are no available flights in the browser window.



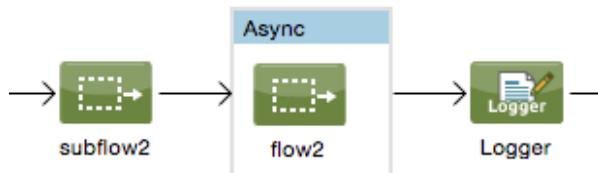
44. Test the PDF location with American; you should get the no available flights message.

*Note: If you test the PDF location with United or Delta, you should get no results and an error in the console: Execution of the expression "exception.causeMatches('com.mulesoft.weave.\*') failed. This is a bug in 3.7.1. As a workaround, you can modify the global Catch Exception Strategy for DataWeave expressions to #[message.?exception.causeMatches('com.mulesoft.weave.\*')], but you will also need to add some more logic to return the desired results.*

## Walkthrough 8-4: Pass messages to an asynchronous flow

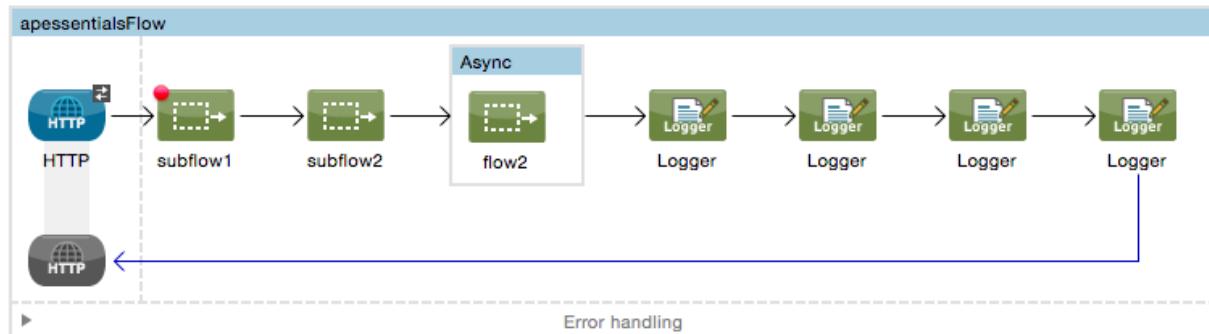
In this walkthrough, you will work in the apessentials2 project and create and pass messages to an asynchronous flow. You will:

- Use the Async scope element to create an asynchronous flow.
- Use the Mule Debugger to watch messages flow through both flows.

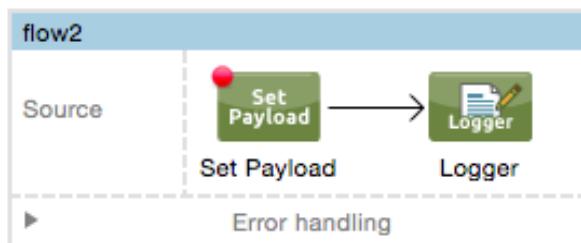


### Create an asynchronous flow

1. Return to examples.xml in apessentials2.
2. Drag an Async scope element from the palette and drop it into the apessentialsFlow between the HTTP Request endpoint and the Logger.
3. Delete the HTTP Request endpoint in apessentialsFlow.
4. Drag a Flow Reference into the Async scope.
5. In the Flow Reference Properties view, set the flow name to flow2.
6. Add three more Logger components after the Async scope.



7. Delete the HTTP Listener endpoint in flow2.
8. Add a Logger component after the Set Payload in flow2.
9. Make sure there is a breakpoint on the Set Payload in flow2.

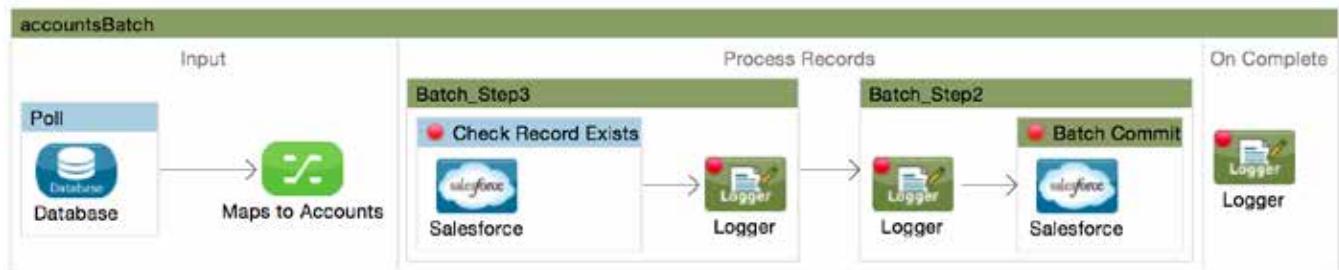


## Debug the application

10. Save the file and debug the application.
11. Make another request to <http://localhost:8082?name=pliny&type=cat> using your own query parameter values.
12. Step through the application again; you should see a copy of the message passed to the asynchronous flow2.
13. Watch the values of the payload change in both flows.



# Module 9: Processing Records



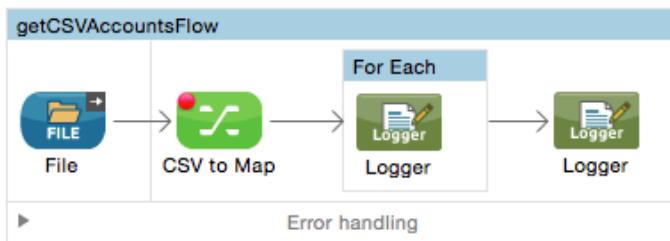
In this module, you will learn:

- Process items in a collection individually.
- Use DataWeave with CSV files.
- Use the Batch Job element (EE) to process individual records.
- Synchronize data from a CSV file to a SaaS application.
- Synchronize data from a legacy database to a SaaS application.

## Walkthrough 9-1: Process items in a collection individually

In this walkthrough, you will split a collection and process each item in it individually. You will:

- Add metadata to a File endpoint.
- Read a CSV file and use DataWeave to convert it to a collection of objects.
- Use the For Each scope element to process each item in a collection individually.



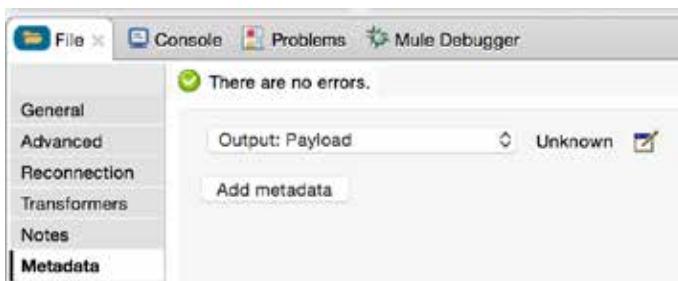
### Modify the File endpoint to not rename the files

1. Open accounts.xml.
2. In the Package Explorer, expand the src/main/resources/input folder.
3. Right-click accounts.csv.backup and select Refactor > Rename.
4. In the Rename Resource dialog box, set the new name to accounts.csv and click OK.
5. In getCSVAccountsFlow, delete the File-to-String transformer.
6. In the File Properties view, delete the move to pattern.

*Note: This will make it easier to test the application because you won't have to keep renaming the file as you move it back to the input directory.*

### Add File endpoint metadata

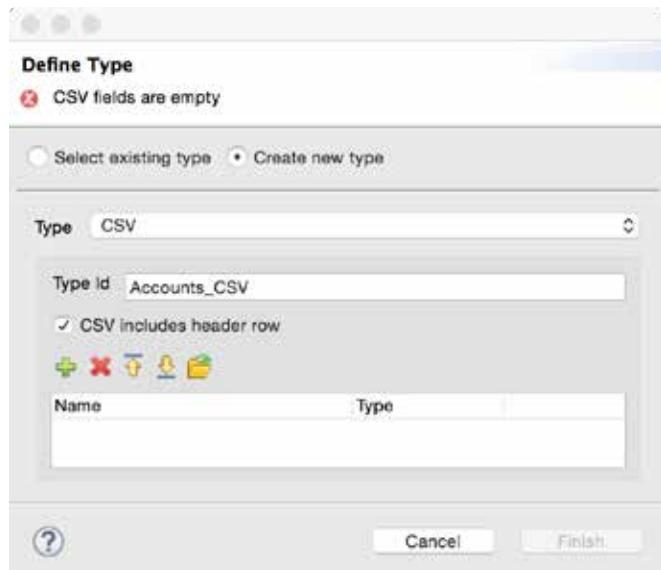
7. In the File Properties view, click Metadata in the left-side navigation.
8. Click the Add metadata button.
9. In the drop-down menu that appears, make sure Output: Payload is selected.



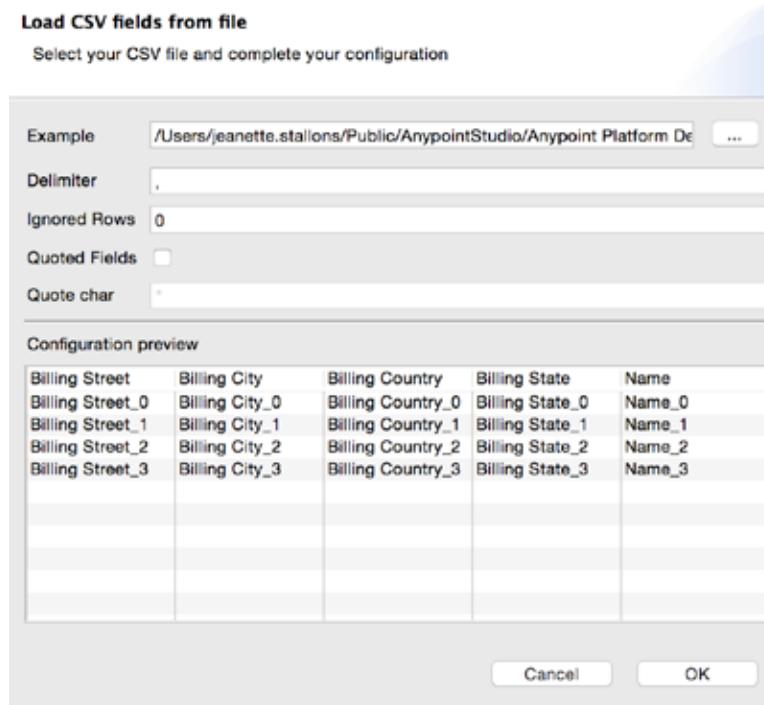
10. Click the Edit button next to the drop-down menu.
11. In the Define Type dialog box, select Create new type.



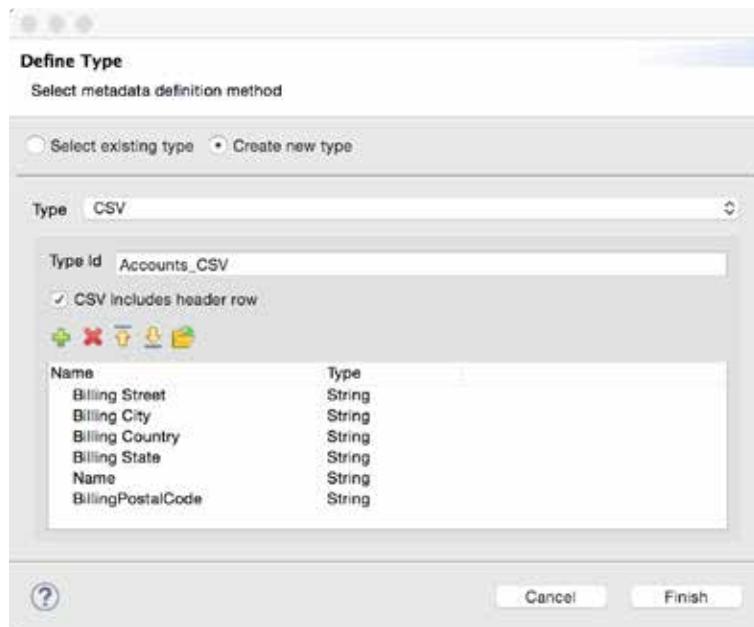
12. Change the type to CSV.
13. Set the Type Id to Accounts\_CSV.
14. Make sure CSV includes header row is checked.
15. Click the Load from example button (the folder icon).



16. In the Load CSV fields from file dialog box, click the Browse button next to Example.
17. In the Select CSV example dialog box, browse to the project's src/main/resources/input folder, select accounts.csv, and click Open.
18. In the Load CSV fields from file dialog box, click OK.



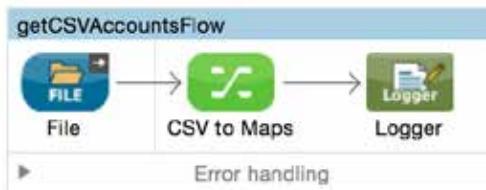
19. In the Define Type dialog box, click Finish.



## Use DataWeave to convert a CSV file to a collection of objects

20. Add a Transform Message component between the File and the Logger.

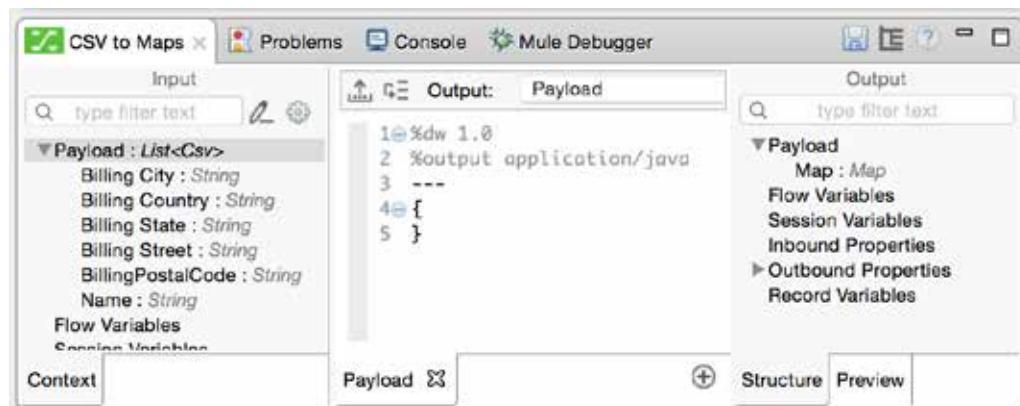
21. Change the name of the component to CSV to Maps.



22. In the Input section of the Transform Message Properties view, make sure the Payload is specified as a List<Csv> with appropriate fields.

23. Look at the Transform section, the output should be set to application/java.

24. Look at the Output section, the Payload should be set to a Map.



25. Write a DataWeave expression to transform the payload without making any changes to its structure or values.
26. Look at the preview.

The screenshot shows the DataWeave preview window with the following content:

```

1@%dw 1.0
2 %output application/java
3 ---
4 payload

```

**Output**

Name	Value
root : ArrayList	
[0] : LinkedHashMap	
• Billing Street : String	????
• Billing City : String	????
• Billing Country : String	????
• Billing State : String	????
• Name : String	????
• BillingPostalCode : String	????

Payload Structure Preview

27. Run the application.
28. Return to the console in Anypoint Studio; you should see the values for all the accounts in the CSV file displayed.

The screenshot shows the Mule Debugger console with the following log output:

```

INFO 2015-07-17 13:42:00,863 [[apessentials-mod09].connector.file.mule.default.receiver.01] org.mule.transport.FileMessageReceiver: Lock obtained on file: /Users/jeanette.stallions/Public/AnypointStudio/Anypoint Platform Development/APEssentials_on_37/apessentials-mod09/src/main/resources/input/accounts.csv
INFO 2015-07-17 13:42:01,682 [[apessentials-mod09].getCSVAccountsFlow.stage1.02] org.mule.api.processor.LogProcessor: [{Billing Street=111 Boulevard Hausmann, Billing City=Paris, Billing Country=France, Billing State=Dog Park Industries, Name=Dog Park Industries, BillingPostalCode=75008}, {Billing Street=400 South St, Billing City=San Francisco, Billing Country=USA, Billing State=CA, Name=Iguana Park Industries, BillingPostalCode=91156}, {Billing Street=777 North St, Billing City=San Francisco, Billing Country=USA, Billing State=CA, Name=Cat Park Industries, BillingPostalCode=91156}]

```

## Add a For Each scope element

29. Drag a For Each scope element form the palette and drop it after the Transform Message component.
30. Add a Logger to the For Each scope.



## Process each element

31. In the Logger Properties view, set the message to #[payload].

## Debug the application

32. Add a breakpoint to the Transform Message component.

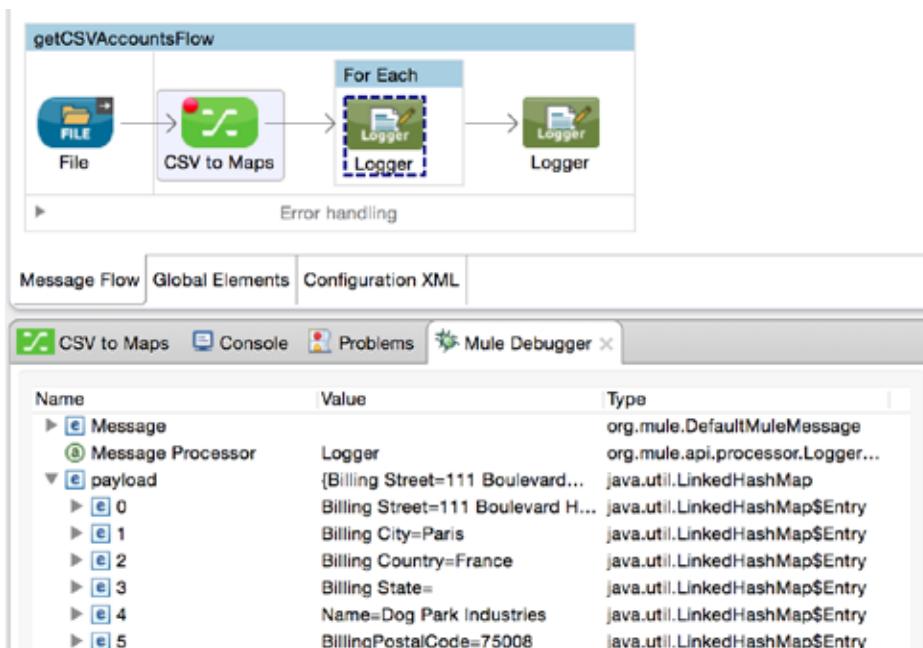
33. Save the file and debug the application.

34. Drag accounts.csv from the src/main/resources/output folder to the src/main/resources/input folder; application execution should stop at the Transform Message component.

35. In the Mule Debugger view, watch the payload value.

36. Step to the For Each scope; the payload should be an ArrayList of LinkedHashMaps.

37. Step through the For Each; the payload should be a LinkedHashMap.



38. Watch the console as you step through; you should see each record displayed.

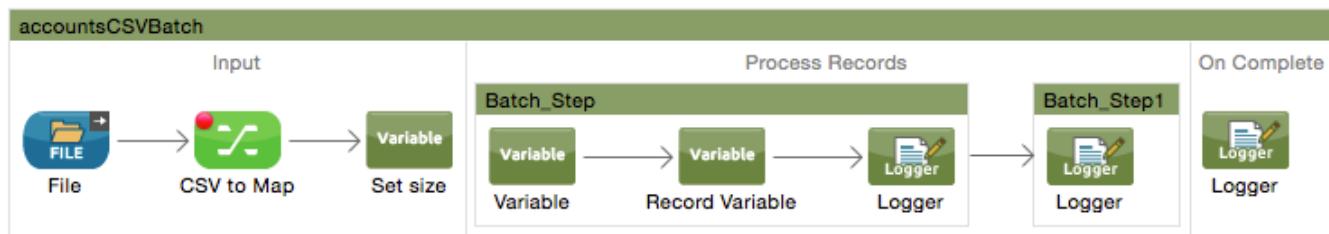
```
INFO 2015-07-17 13:47:29,712 [[apessentials-mod09].connector.file.mule.default.receiver.01] org.mule.transport.FileMessageReceiver: Lock obtained on file: /Users/jeanette.stallons/Public/AnypointStudio/Anypoint Platform Development/APEssentials_on_37/apessentials-mod09/src/main/resources/input/accounts.csv
INFO 2015-07-17 13:48:44,456 [[apessentials-mod09].getCSVAccountsFlow.stage1.02] org.mule.api.processor.LogggerMessageProcessor: {Billing Street=111 Boulevard Hausmann, Billing City=Paris, Billing Country=France, Billing State=, Name=Dog Park Industries, BillingPostalCode=75008}
INFO 2015-07-17 13:48:49,321 [[apessentials-mod09].getCSVAccountsFlow.stage1.02] org.mule.api.processor.LogggerMessageProcessor: {Billing Street=400 South St, Billing City=San Francisco, Billing Country=USA, Billing State=CA, Name=Iguana Park Industries, BillingPostalCode=91156}
INFO 2015-07-17 13:48:50,220 [[apessentials-mod09].getCSVAccountsFlow.stage1.02] org.mule.api.processor.LogggerMessageProcessor: {Billing Street=777 North St, Billing City=San Francisco, Billing Country=USA, Billing State=CA, Name=Cat Park Industries, BillingPostalCode=91156}
```

39. Step through to the Logger after the For Each; the payload should be an ArrayList again.

## Walkthrough 9-2: Create a batch job for records in a file

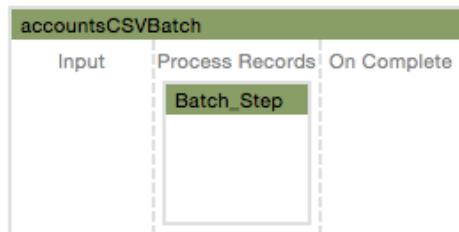
In this walkthrough, you will create a batch job to process records in a CSV file. You will:

- Create a new flow containing a batch job.
- Explore flow and record variable persistence across batch steps and phases.
- In the input phase, check for CSV files every second and convert them to a collection of objects.
- In the process records phase, create two batch steps for setting and tracking variables.
- In the on complete phase, display the number of records processed and failed.



### Create a batch job

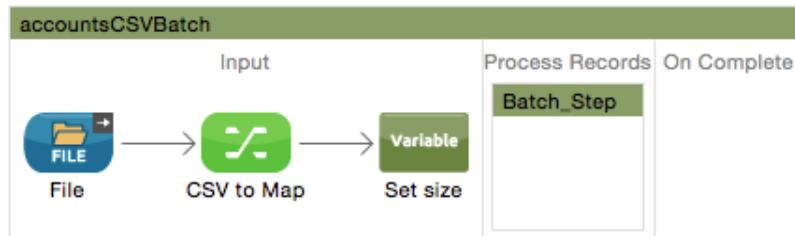
1. Return to accounts.xml.
2. Drag a Batch scope element from the palette and drop it above the getCSVAccountsFlow.
3. Change the batch job name to accountsCSVBatch.



### Add elements to the input phase

4. Select the File and Transform Message elements in getCSVAccountsFlow and select Edit > Copy or press Cmd+C/Ctrl+C.
5. Click in the accountsCSVBatch input phase and select Edit > Paste or press Cmd+V/Ctrl+V.

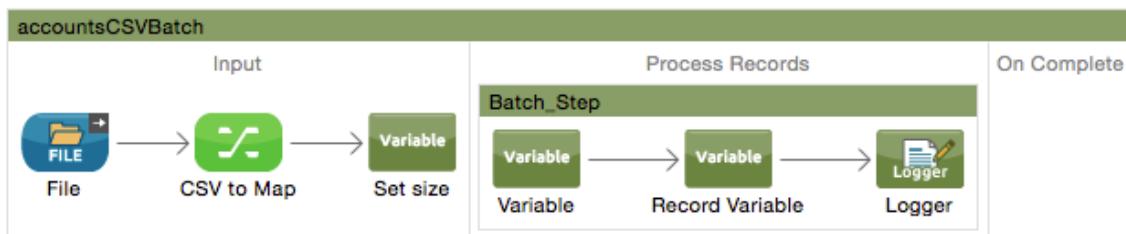
- Rename the elements to remove Copy\_of\_ from their names.
- Add a Variable transformer after the transformer in the input phase.
- In the Variable Properties view, change the display name to Set size and select Set Variable.



- Set the name to size and the value to #[payload.size()].

## Add processors to a batch step in the process records phase

- Add a Variable transformer to the batch step in the process records phase.
- In the Properties view, select Set Variable and set the name to fname and the value to #[payload.Name].
- Add a Record Variable transformer to the batch step.
- In the Properties view, select Set Record Variable and set the name to rname and the value to #[payload.Name].
- Add a Logger component to the batch step.



- In the Logger Properties view, set the message to display the record variable.

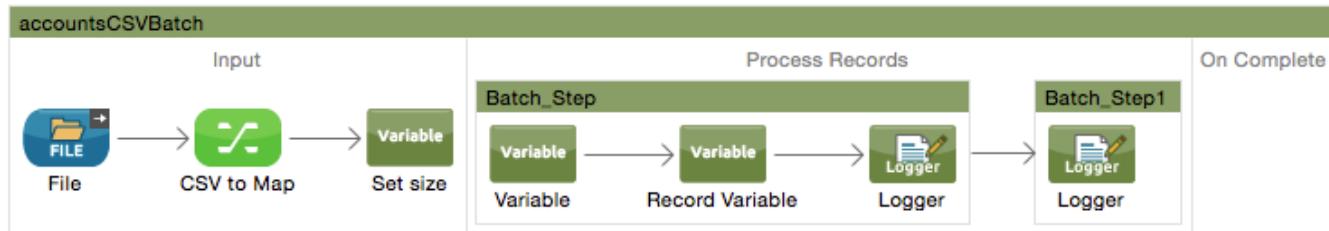
```
#['\n\nRecordVar: ' + recordVars.rname]
```

*Note: If you want to add line breaks, add one or more '\n' to your MEL expression – meaning it must be inside the #[].*

*Note: Anypoint Studio 5.2.0 does not display record variables in the Mule Debugger, so you are displaying the value in the console. If you are using a later release that does display record variables, you can skip the steps to display the record variable.*

## Create a second batch step

16. Drag a Batch Step scope element from the palette and drop it in the process records phase after the first batch step.
17. Add a Logger to the second batch step.



18. In the Logger Properties view, set the message to display the record variable.

```
#['\n\nRecordVar: ' + recordVars.rname]
```

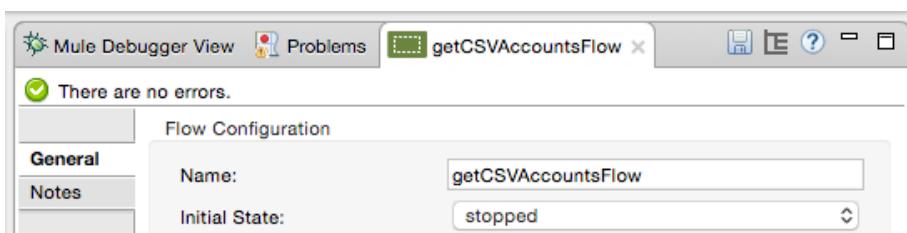
## Add processors to the on complete phase

19. Add a Logger component to the on complete phase.
20. In the Logger Properties view, set the message to display the number of processed records and failed records.

```
#['\n\nProcessed: ' + payload.processedRecords + ' Failed: ' + payload.failedRecords]
```

## Stop the getCSVAccountsFlow from running

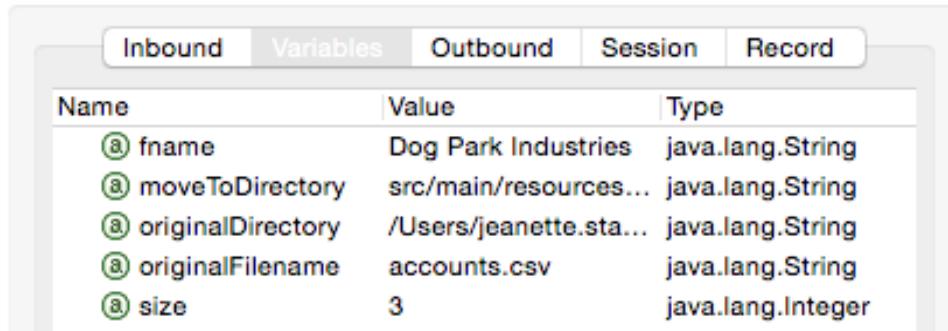
21. Double-click getCSVAccountsFlow.
22. In the Properties view, set the initial state to stopped.



## Debug the application

23. Add a breakpoint to the CSV to Maps component in accountCSVBatch.
24. Save the file to redeploy the application in debug mode.
25. After the application starts, drag the accounts.csv file from the output folder to the input folder.
26. In the Mule Debugger view, watch the payload value.

27. Step into the process records phase.
28. Click the Variables tab; you should see the size flow variable.
29. Step to the Logger in the first batch step; you should see the fname flow variable.



The screenshot shows the Mule Debugger interface with the 'Variables' tab selected. The table displays the following variables:

Name	Value	Type
⑧ fname	Dog Park Industries	java.lang.String
⑧ moveToDirectory	src/main/resources...	java.lang.String
⑧ originalDirectory	/Users/jeanette.sta...	java.lang.String
⑧ originalFilename	accounts.csv	java.lang.String
⑧ size	3	java.lang.Integer

30. Click the Record tab; you should see the rname flow variable.

*Note: Anypoint Studio 5.2.0 does not display record variables in the Mule Debugger.*

31. Click the Variables tab.
32. Step through the rest of the records in the first batch step and watch the payload and the fname variable.
33. Look at the console; you should see the values for the rname record variable displayed.

```
RecordVar: Dog Park Industries
INFO 2015-07-17 13:58:06,247 [batch-job-accountsCSV
eProcessor:
```

```
RecordVar: Iguana Park Industries
INFO 2015-07-17 13:58:08,180 [batch-job-accountsCSV
eProcessor:
```

```
RecordVar: Cat Park Industries
INFO 2015-07-17 13:58:08,220 [batch-job-accountsCSV
```

34. Step into and through the second batch step; watch the payload, the variables, the record variables, and the console.
35. Look at the console; you should see the values for the rname record variable displayed again for each record.

36. Step into the on complete phase; you should see the payload is an ImmutableBatchJobResult.

Name	Value
► <b>e</b> Message	
❸ Message Processor	Logger
▼ <b>e</b> payload	com.mulesoft.module.batch.ImmutableBatchJobResult@3e794b62
❸ batchJobInstanceld	20efbad0-aa8d-11e4-810c-fab1a430eb79
❸ elapsedTimeInMillis	146238
❸ failedOnCompletePhase	false
❸ failedOnInputPhase	false
❸ failedOnLoadingPhase	false
❸ failedRecords	0
❸ inputPhaseException	null
❸ loadedRecords	3
❸ loadingPhaseException	null
❸ onCompletePhaseException	null
❸ processedRecords	3
❸ serialVersionUID	4323747859995526737
► <b>e</b> stepResults	{Batch_Step1=com.mulesoft.module.batch.ImmutableBatchSte...
❸ successfulRecords	3
❸ totalRecords	3

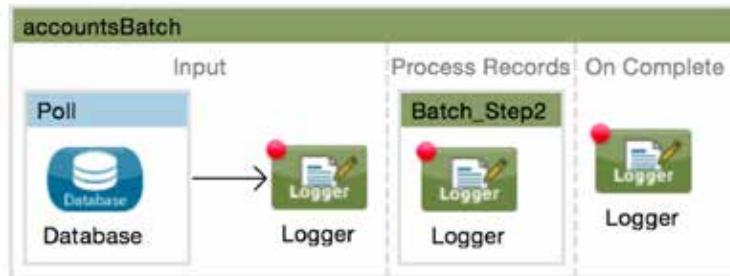
37. Step to the end of the application; you should see the number of process records phase and failed records in the console.

```
Processed: 3 Failed: 0
INFO 2015-07-17 13:59:07,285
faultBatchEngine: Finished exe
```

## Walkthrough 9-3: Create a batch job for records in a database

In this walkthrough, you will create a batch job that retrieves records from a legacy database. You will:

- Go to a form and add multiple accounts for a specific postal code to the database.
- Create a new flow containing a batch job that polls a MySQL database every 30 seconds for records with a specific postal code.
- Use the Poll scope.



### Get familiar with the data in the database

1. In a browser, go to the account list URL for the MySQL database listed in the course snippets.txt file.
2. Look at the existing data and the name of the columns, which match the names of the database fields.

accountID	name	street	city	state	postal	country
516	Jawshua	12312 Blue Rd	San Francisco	California	94108	United States

3. Click the Create More Accounts button.
4. Add one or more new records with a specific postal code; you will retrieve these records in this walkthrough and insert them into your Salesforce accounts in the next walkthrough.

## Create a batch job that polls a database

1. Return to accounts.xml.
2. Drag a Batch scope element onto the canvas from the palette to create a new flow.
3. Add a Poll scope element to the input phase.
4. In the Properties view, select the fixed frequency scheduler.
5. Set the frequency to 30 and the time unit to seconds.
6. Add a Database connector to the poll.



7. Set the connector configuration to the existing Training\_MySQL\_Configuration global element.
8. Set the operation to Select.
9. Add a query to select the data for your postal code.

```
SELECT *
FROM accounts
WHERE postal = '94108'
```

## Log each record to the console in the process records phase

10. Add a Logger component to the Process Records phase.
11. Display the record – the message payload – and other text or information you wish.

```
#['\n\nRECORD: ' + payload]
```

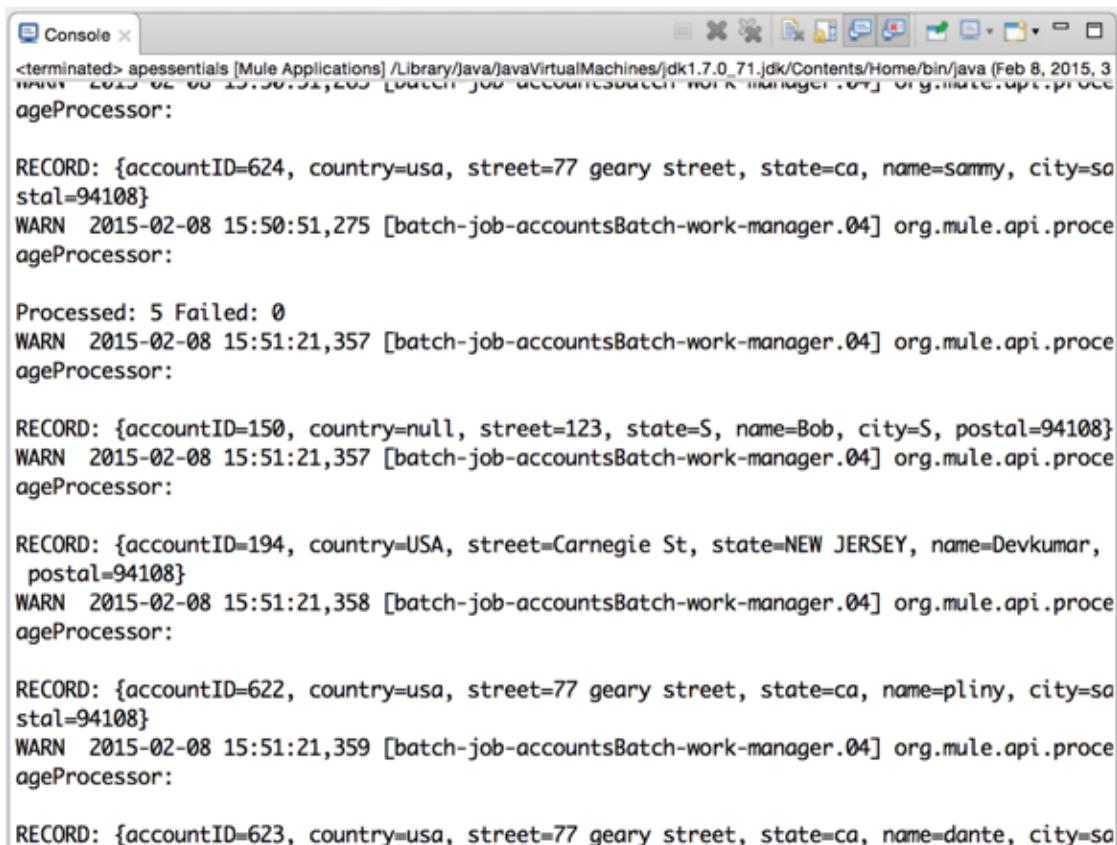
## Log processed records in on complete phase

12. Add a Logger component to the on complete phase.
13. Set the Logger message to display the number of processed records and failed records.

```
#['\n\nProcessed: ' + payload.processedRecords + ' Failed: ' +
payload.failedRecords]
```

## Test the application

- Run the application and watch the console; you should see records displayed every 30 seconds.



The screenshot shows a Java console window titled "Console". The logs output by the Mule application are as follows:

```
<terminated> apessentials [Mule Applications] /Library/Java/JavaVirtualMachines/jdk1.7.0_71.jdk/Contents/Home/bin/java (Feb 8, 2015, 3  
WARN 2015-02-08 15:50:51,203 [batch-job-accountsBatch-work-manager.04] org.mule.api.processor  
ageProcessor:  
  
RECORD: {accountID=624, country=usa, street=77 geary street, state=ca, name=sammy, city=sa  
stal=94108}  
WARN 2015-02-08 15:50:51,275 [batch-job-accountsBatch-work-manager.04] org.mule.api.processor  
ageProcessor:  
  
Processed: 5 Failed: 0  
WARN 2015-02-08 15:51:21,357 [batch-job-accountsBatch-work-manager.04] org.mule.api.processor  
ageProcessor:  
  
RECORD: {accountID=150, country=null, street=123, state=S, name=Bob, city=S, postal=94108}  
WARN 2015-02-08 15:51:21,357 [batch-job-accountsBatch-work-manager.04] org.mule.api.processor  
ageProcessor:  
  
RECORD: {accountID=194, country=USA, street=Carnegie St, state=NEW JERSEY, name=Devkumar,  
postal=94108}  
WARN 2015-02-08 15:51:21,358 [batch-job-accountsBatch-work-manager.04] org.mule.api.processor  
ageProcessor:  
  
RECORD: {accountID=622, country=usa, street=77 geary street, state=ca, name=pliny, city=sa  
stal=94108}  
WARN 2015-02-08 15:51:21,359 [batch-job-accountsBatch-work-manager.04] org.mule.api.processor  
ageProcessor:  
  
RECORD: {accountID=623, country=usa, street=77 geary street, state=ca, name=dante, city=sa
```

*Note: Right now, all records with matching postal code are retrieved – over and over again. In the next walkthrough, you will modify this so only new records with the matching postal code are retrieved.*



## Walkthrough 9-4: Restrict processing using a poll watermark

In this walkthrough, you will modify the poll so it only retrieves *new* database records with a specific postal code. You will:

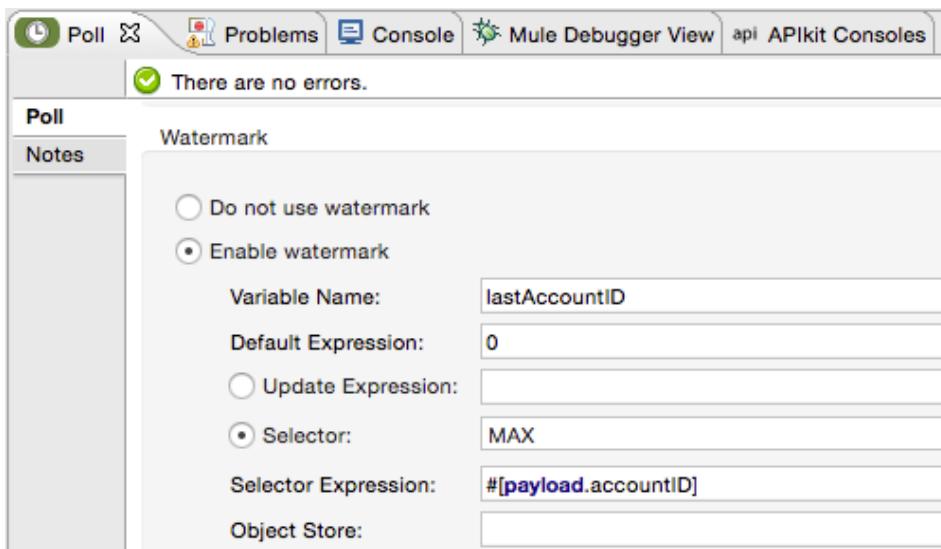
- Modify the Poll to use a watermark to keep track of the last record returned from the database.
- Modify the database query to use the watermark.
- Clear application data.

### Parameterized query:

```
SELECT *
FROM accounts
WHERE postal = '94108' AND accountID > #[flowVars.lastAccountID]
```

### Use a watermark to keep track of the last record

1. Return to accounts.xml.
2. In the Poll Properties view, select Enable watermark.
3. Set the watermark to store the max accountID returned by setting the following values.
  - Variable name: lastAccountID
  - Default expression: 0
  - Selector: MAX
  - Selector Expression: #[payload.accountID]



## Debug and examine the watermark value

4. Place a breakpoint on the Logger in the on complete phase.
5. Debug the application.
6. Find your watermark variable in the Variables section of the Mule Debugger view; initially, you should see a default value of zero.

Inbound	Variables	Outbound	Session	Record
Name	Value	Type		
⑧ lastAccountID	0	java.lang.String		
⑧ pollingFrequency	1000	java.lang.Long		

7. Run the application though until the next polling event – the next time it retrieves records from the accounts table; the watermark variable should now be equal to the max accountID for training accounts records with the postal code you are using.

The screenshot shows the Mule Studio interface. At the top, there's a toolbar with icons for Save, Undo, Redo, Cut, Copy, Paste, Find, Replace, and others. Below the toolbar is a tab bar with 'Message Flow' (selected), 'Global Elements', and 'Configuration XML'. The main workspace contains a message flow named 'accountsBatch'. The flow starts with a 'Poll' component (Database connector) which feeds into a 'Batch\_Step2' component. This is followed by two 'Logger' components. The second 'Logger' component has a red exclamation mark icon above it, indicating an error or warning. To the right of the message flow is a 'Variables' table showing the values of 'lastAccountID' (0) and 'pollingFrequency' (1000). A sidebar on the right lists various Mule components: Select, Connectors (Ajax, Anypoint Studio, Scopes, Async).

Name	Value	Type
⑧ lastAccountID	194	java.lang.Integer
⑧ pollingFrequency	1000	java.lang.Long

8. Resume the application multiple times; the same records should still be selected over and over again.

## Modify the database query to use the watermark

- In the Database Properties view, modify the query so it only returns records for your postal code and with accountID values greater than the watermark lastAccountID value.

Parameterized query:

```
SELECT *
FROM accounts
WHERE postal = '94108' AND accountID > #[flowVars.lastAccountID]
```

## Test the application

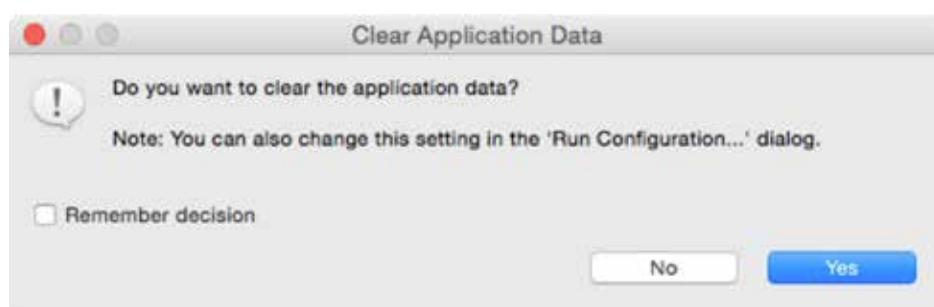
- Run the application and look at the console; you should see that no records are retrieved at all this time.

```
INFO 2015-07-17 14:11:58,254 [pool-54-t]
Processed: 0 Failed: 0
INFO 2015-07-17 14:11:58,254 [pool-54-t]
execution of onComplete phase for insta
```

*Note: By default, the watermark is stored in a persistent object store so its value is retained between different executions of the application.*

## Clear application data

- Select Run > Run Configurations.
- Make sure your apessentials project is selected and then on the General tab, change Clear Application Data from Never to Prompt.
- Run or debug your application again; you should be prompted to clear the application data.
- Click Yes.



- Look at the console; you should see the latest matching records retrieved from the legacy database again – but this time, only once.

16. Watch the console and see that all subsequent polling events retrieve no records.

```
Processed: 33 Failed: 0
INFO 2015-07-17 14:13:46,338 [batch-job-accountsBatch-work-manager.02] com.mulesoft
Finished execution of onComplete phase for instance b5d75e60-2cc8-11e5-9bca-027c6f5c1039
INFO 2015-07-17 14:13:46,339 [batch-job-accountsBatch-work-manager.02] com.mulesoft
Finished execution for instance 'b5d75e60-2cc8-11e5-9bca-027c6f5c1039' of job 'accountsBatch'
Successful records: 33. Failed Records: 0
INFO 2015-07-17 14:13:46,340 [batch-job-accountsBatch-work-manager.02] com.mulesoft

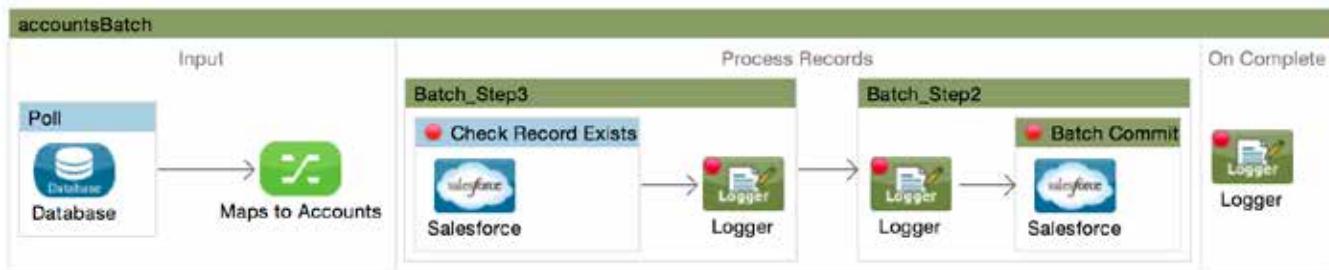
INFO 2015-07-17 14:13:46,351 [batch-job-accountsBatch-work-manager.02] com.mulesoft
h_Step2 finished processing all records for instance b5d75e60-2cc8-11e5-9bca-027c6f5c1039
INFO 2015-07-17 14:14:15,637 [pool-28-thread-1] com.mulesoft.module.batch.engine
32-2cc8-11e5-9bca-027c6f5c1039 for batch job accountsBatch
INFO 2015-07-17 14:14:15,637 [pool-28-thread-1] com.mulesoft.module.batch.engine
INFO 2015-07-17 14:14:15,637 [pool-28-thread-1] com.mulesoft.module.batch.engine
INFO 2015-07-17 14:14:15,645 [pool-28-thread-1] com.mulesoft.module.batch.engine
ase for instance 'c7791232-2cc8-11e5-9bca-027c6f5c1039' of job 'accountsBatch'
INFO 2015-07-17 14:14:15,645 [pool-28-thread-1] com.mulesoft.module.batch.engine
ase for instance c7791232-2cc8-11e5-9bca-027c6f5c1039 of job accountsBatch. 0 rec
INFO 2015-07-17 14:14:15,646 [pool-28-thread-1] com.mulesoft.module.batch.engine
-11e5-9bca-027c6f5c1039' of job 'accountsBatch' has no records to process. It's e
INFO 2015-07-17 14:14:15,646 [pool-28-thread-1] com.mulesoft.module.batch.engine
nComplete phase for instance c7791232-2cc8-11e5-9bca-027c6f5c1039 of job accountsBatch
INFO 2015-07-17 14:14:15,646 [pool-28-thread-1] org.mule.api.processor.LoggerMes

Processed: 0 Failed: 0
INFO 2015-07-17 14:14:15,647 [pool-28-thread-1] com.mulesoft.module.batch.engine
```

## Walkthrough 9-5: Restrict processing using a message enricher and a batch step filter

In this walkthrough, you will add logic to check and see if an account already exists in Salesforce before adding it. You will:

- Add a first batch step with a Message Enricher scope element that checks if a record already exists in Salesforce (an account with the same Name) and stores the result in a record variable and retains the original payload.
- Modify the second batch step to use a filter that only allows new records (records that don't already exist) to be processed.
- (Optional) Add the record(s) to Salesforce.



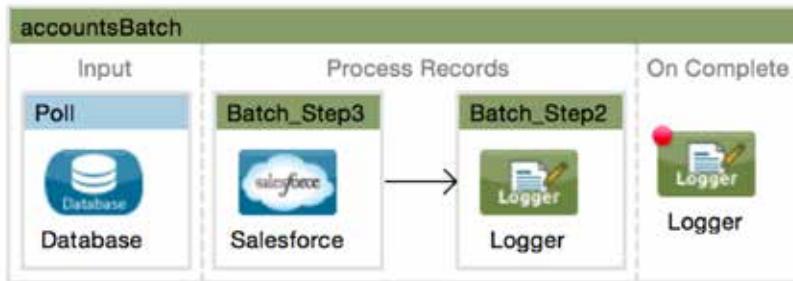
### Add a Salesforce account manually

1. In a browser, go to <http://salesforce.com> and log in with your Salesforce Developer account.
2. Click the Accounts link in the main menu bar.
3. In the view drop-down, select All Accounts with Postal Code and click the Go button.
4. Click the New Account button.
5. Enter an account name (one that matches one of the accounts you added with your postal code to the MySQL database) and click Save.

All Accounts with Postal Code			
New Account		A   B   C   D   E   F   G   H   I   J   K	
Action	Account Name	Billing Zip/Postal Code	Billing Country
Edit   Del   +	Burlington Textiles Co...	27215	USA
Edit   Del   +	Dickenson plc	66045	USA
Edit   Del   +	Edge Communications		
Edit   Del   +	Express Logistics and...		
Edit   Del   +	GenePoint		
Edit   Del   +	Grand Hotels & Resor...		
Edit   Del   +	Pliny		
Edit   Del   +	Pyramid Construction ...	75251	France
Edit   Del   +	sForce	94087	US
Edit   Del   +	United Oil & Gas Com		

## Check to see if a record already exists in Salesforce

6. Return to Anypoint Studio.
7. Return to accountsBatch in accounts.xml.
8. Drag out a new Batch Step scope element and drop it at the beginning of the process records phase.
9. Add a Salesforce connector to the new batch step.

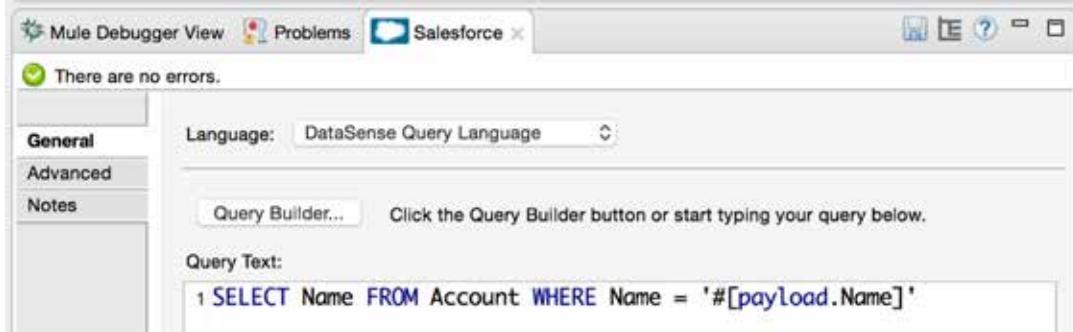


10. Configure the Salesforce connector endpoint to use the existing Salesforce connector configuration.
11. Set the operation to Query.
12. Click the Query Builder button.
13. Set the type to Account(Account).
14. Select a field of Name; it does not matter what you select, you just want to see if any records are returned.
15. Click the Add Filter button.
16. Create a filter to check if the Name field in the record being processed already exists in the database.

Name = #[payload.Name]

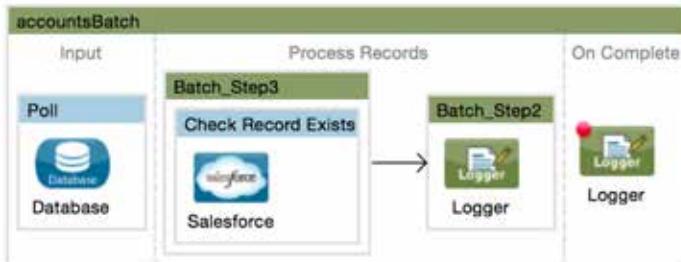
*Note: Right now, you can use payload.name or payload.Name because the payload is a caseSensitiveHashMap. Later this walkthrough, however, you will transform the Map to an Account object with a Name property and will have to refer to this as payload.Name.*

17. Click OK.



## Add a Message Enricher

18. Add a Message Enricher scope element to the first batch step.
19. Move the Salesforce connector endpoint so it is inside the message enricher.
20. In the Message Enricher Properties view, set the display name to Check Record Exists.

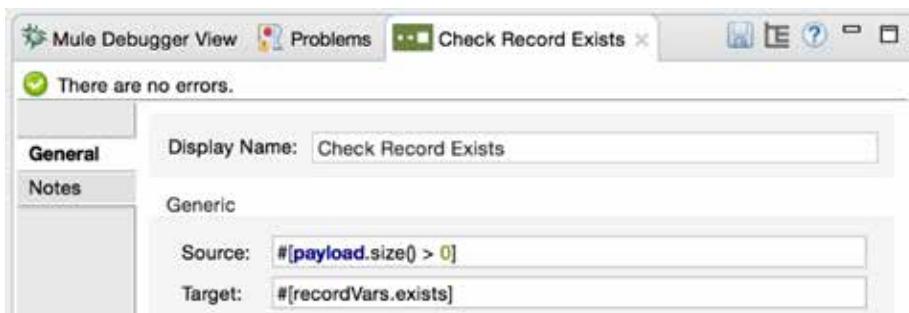


21. Set the source to an expression that checks to see if any records were returned, i.e. if there is a payload.

```
##[payload.size() > 0]
```

22. Set the target to assign the result of the source expression to a record variable called exists.

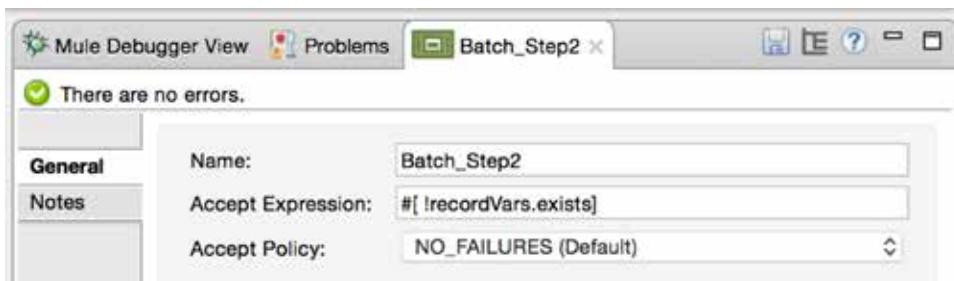
```
##[recordVars.exists]
```



## Set a filter for the insertion step

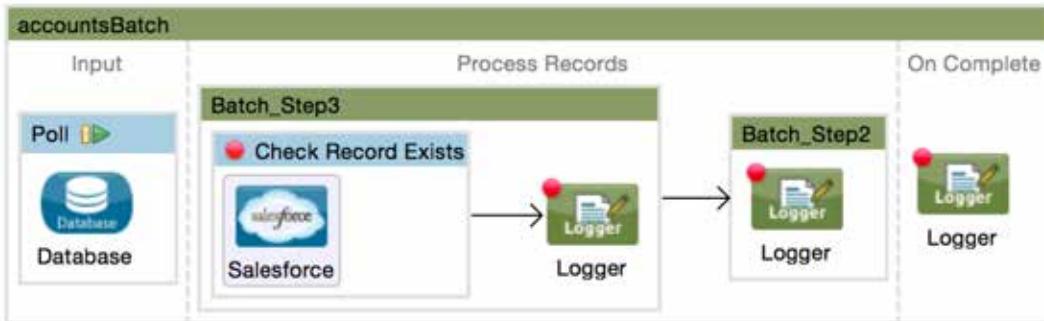
23. Double-click the second batch step that will/would insert the record into Salesforce.
24. In the Batch Step Properties view, set the Accept Expression so that only records that have a record variable exists set to false are processed.

```
##[ !recordVars.exists]
```



## Test the application

25. Add a breakpoint to the Message Enricher.
26. Add a Logger after the Message Enricher in the first batch step and add a breakpoint to it.



27. In the Logger Properties view, set the value to display the exists record variable.

```
#['\n\nRecordVar: ' + recordVars.exists]
```

28. Add a breakpoint to the Logger in the second batch step.
29. Debug the application and clear the application data.
30. Watch the console in Anypoint Studio; you should see the exists record variable set to false for all the records except the one you already added to Salesforce.

```
RecordVar: false
INFO 2015-07-17 14:29:04,416 [batch-job-account]

RecordVar: false
INFO 2015-07-17 14:29:04,731 [batch-job-account]

RecordVar: false
INFO 2015-07-17 14:29:05,025 [batch-job-account]

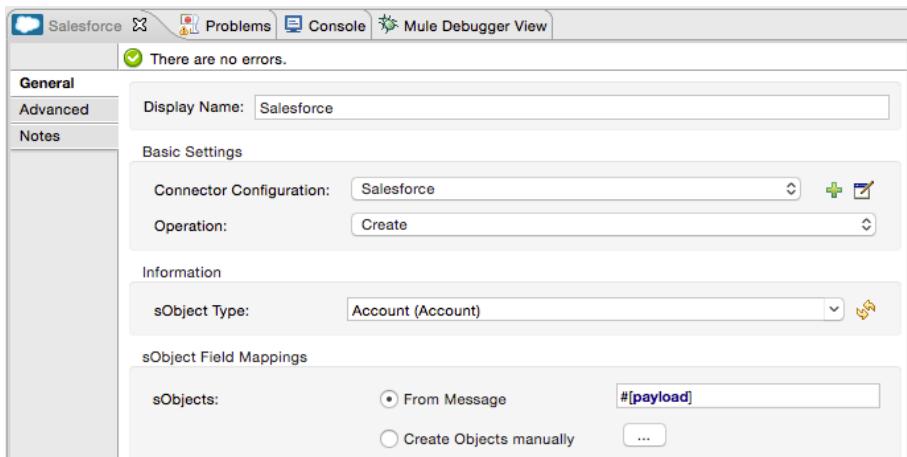
RecordVar: true
INFO 2015-07-17 14:29:05,396 [batch-job-account]

RecordVar: false
INFO 2015-07-17 14:29:05,689 [batch-job-account]
```

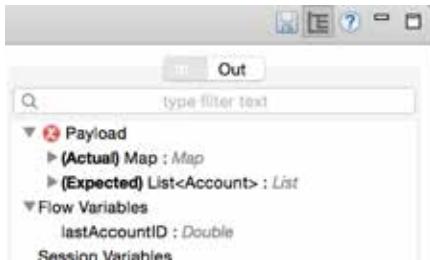
## Commit new account records to Salesforce (optional)

31. Add a Salesforce connector to the second batch step in the processing phase.
32. Configure the Salesforce connector endpoint to use the existing Salesforce connector configuration.
33. Set the operation to Create.
34. Set the sObject Type to Account (Account).

35. Leave the sObject Field Mappings to From Message #[payload].

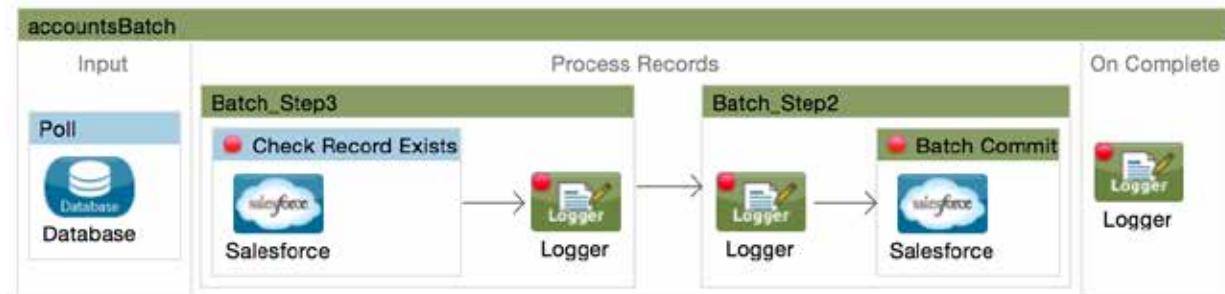


36. Look at the DataSense Explorer and see a problem indicated for the payload; it is a Map but the outbound connector is expecting a List of Account objects.



37. Drag out a Batch Commit scope and drop it in the second batch step.

38. Add the Salesforce endpoint to it.



39. In the Batch Commit Properties view, set the commit size to 100.

## Transform the record data

40. Copy the Salesforce endpoint in the second batch step and paste it after the poll in the input phase.

*Note: You are temporarily adding this Salesforce endpoint after the Database endpoint so DataSense will work to create the mappings. You could also add the DataWeave transformation to the process records phase, but this would add additional overhead.*

41. Add a Transform Message component between the Poll scope and the Salesforce endpoint.

42. Change the name of Transform Message component to Maps to Accounts.



43. Look at the Transform Message Properties view; DataSense should work with DataWeave to automatically create a scaffold for a transformation to convert the database List of Map objects to a Salesforce List of Account objects.

44. Modify the DataWeave expression to only keep the Name, BillingStreet, BillingCity, BillingState, BillingPostalCode, and BillingCountry fields.

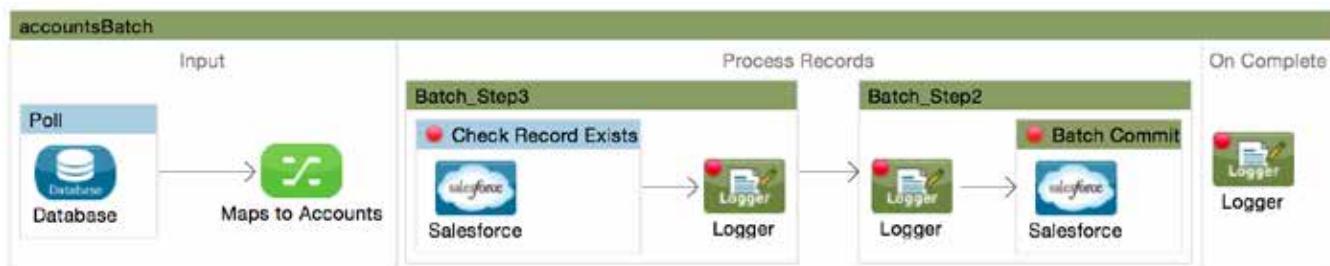
```
Output: Payload
1@%dw 1.0
2 %output application/java
3 ---
4@[
5   Name: "????",
6   BillingStreet: "????",
7   BillingCity: "????",
8   BillingState: "????",
9   BillingPostalCode: "????",
10  BillingCountry: "????"
11 ]]
```

45. Change the expression to use the map operator to map the payload.

46. Replace the placeholder values with references to the corresponding property in the input.

```
Output: Payload
1@%dw 1.0
2 %output application/java
3 ---
4@payload map [
5   Name: $.name,
6   BillingStreet: $.street,
7   BillingCity: $.city,
8   BillingState: $.state,
9   BillingPostalCode: $.postal,
10  BillingCountry: $.country
11 ]
```

47. Delete the Salesforce endpoint in the input phase.



48. Save the file.

## Test the application

49. Locate the Salesforce endpoint in getSFDCAccountsFlow.

50. Modify the query so it selects accounts with the postal code you have been using this module.

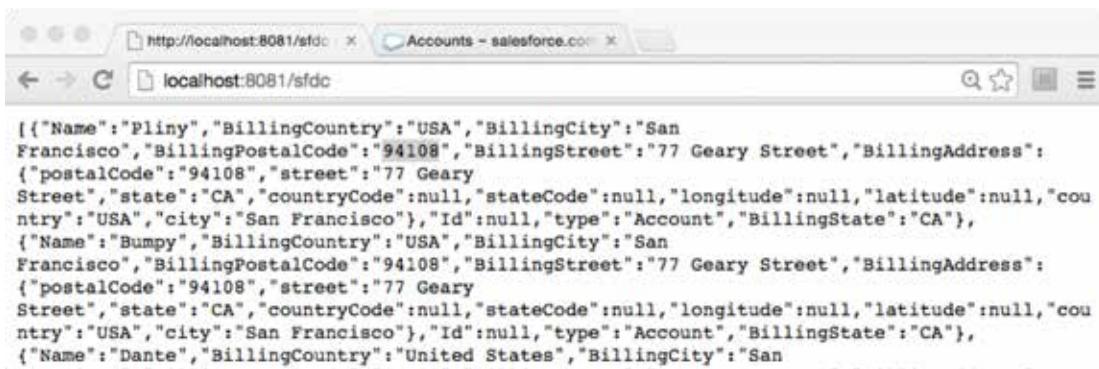
Query Text:

```
1 SELECT BillingCity,BillingCountry,BillingPostalCode,BillingState,BillingStreet,Name  
2 FROM Account  
3 WHERE BillingPostalCode = '94108'
```

51. Run the application and clear the application data.

52. Check the console and make sure you see your new records processed.

53. Make a request to <http://localhost:8081/sfdc>; you should see your new records from the legacy MySQL database now in the Salesforce database.



```
[{"Name": "Pliny", "BillingCountry": "USA", "BillingCity": "San Francisco", "BillingPostalCode": "94108", "BillingStreet": "77 Geary Street", "BillingAddress": {"postalCode": "94108", "street": "77 Geary Street", "state": "CA", "countryCode": null, "stateCode": null, "longitude": null, "latitude": null, "country": "USA", "city": "San Francisco"}, "Id": null, "type": "Account", "BillingState": "CA"}, {"Name": "Bumpy", "BillingCountry": "USA", "BillingCity": "San Francisco", "BillingPostalCode": "94108", "BillingStreet": "77 Geary Street", "BillingAddress": {"postalCode": "94108", "street": "77 Geary Street", "state": "CA", "countryCode": null, "stateCode": null, "longitude": null, "latitude": null, "country": "USA", "city": "San Francisco"}, "Id": null, "type": "Account", "BillingState": "CA"}, {"Name": "Dante", "BillingCountry": "United States", "BillingCity": "San
```

54. Run the application again; no records should be processed.

*Note: Remember you can also go the Salesforce web application and view, edit, and delete Account records.*

# Module 10: Building RESTful Interfaces with RAML and APIkit

The screenshot shows two windows side-by-side. The left window is the 'Designer' view of the Anypoint Platform, displaying a RAML file for the 'MUA Flights API'. The right window is a browser showing the resulting RESTful service at `localhost:8081/api/flights`, with a query parameter `?airline=delta`. Below the browser window is the APIkit flow configuration, titled 'getFlights(destination)mua-config'. The flow consists of several components: 'Source' (HTTP request), 'Set Payload' (with a JSON payload example), 'getFlightsFlow' (a complex flow with multiple steps and conditions), and 'Logger' (for logging the response).

```
1  raml 0.8
2  title: MUA Flights API
3  version: 1.0
4  #baseUri: http://server/api/{version}
5  #baseUrl: http://mocksvc.mulesoft.com/mock/v1/4711-4711-beta-0x0000000000
6
7  /flights:
8    /{destination}:
9      get:
10        queryParameters:
11          airline:
12            default: all
13            enum: [all, united, delta, american]
14        responses:
15          200:
16            body:
17              application/json:
18                examples:
19                  {"airlineName": "United", "price": 400, "depart": "2015-02-20T00:00:00Z", "origin": "MIA", "empty5": true}, {"airlineName": "United", "price": 945, "depart": "2015-02-20T00:00:00Z", "origin": "MIA", "empty5": true}, {"airlineName": "Delta", "price": 958, "depart": "2015-02-20T00:00:00Z", "origin": "MIA", "empty5": true}, {"airlineName": "American", "price": 954, "depart": "2015-02-20T00:00:00Z", "origin": "MIA", "empty5": true}
```

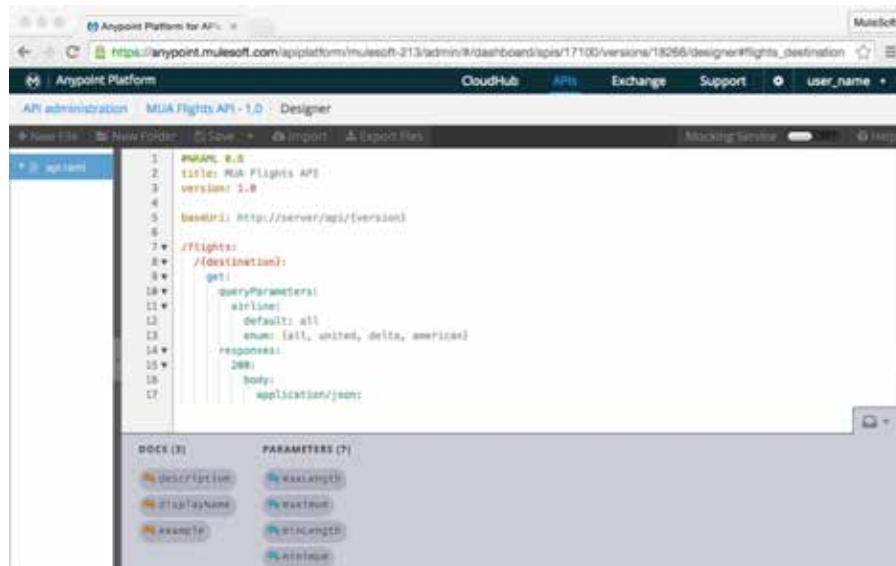
In this module, you will learn:

- To define an API with RAML.
- To create RAML files with Anypoint Designer.
- To implement a RAML file as a RESTful web service with Anypoint Studio and APIkit.

## Walkthrough 10-1: Use API Designer to define an API with RAML

In this walkthrough, you will create an API definition for MUA with RAML. You will:

- Add a new API to the Anypoint Platform.
- Use the API Designer to create a RAML file.
- Use a nested resource for the destination and a query parameter for the airline.

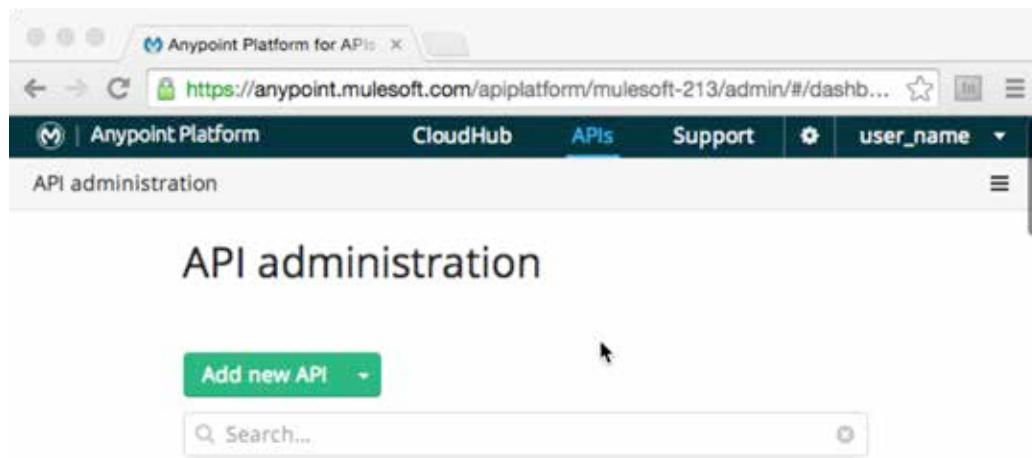


The screenshot shows the Anypoint Platform API Designer interface. The top navigation bar includes CloudHub, APIs, Exchange, Support, and user\_name. The main area displays a RAML file for the "MUA Flights API - 1.0" version. The RAML code defines a base URL and a /flights endpoint with a /{destination} nested resource. The /get method for this endpoint takes a query parameter "airline" with values "all", "united", "delta", and "america". The response is 200 OK with a JSON body. Below the code editor, there are sections for DOCS (3) and PARAMETERS (7), each listing several items with small icons.

```
1  *RAML 0.8
2  *title: MUA Flights API
3  *version: 1.0
4
5  *baseURL: http://server/api/{version}
6
7  */flights:
8    */{destination}:
9      *get:
10        *queryParameters:
11          *airline:
12            *default: all
13            *enum: [all, united, delta, america]
14        *responses:
15          *200:
16            *body:
17              application/json;
```

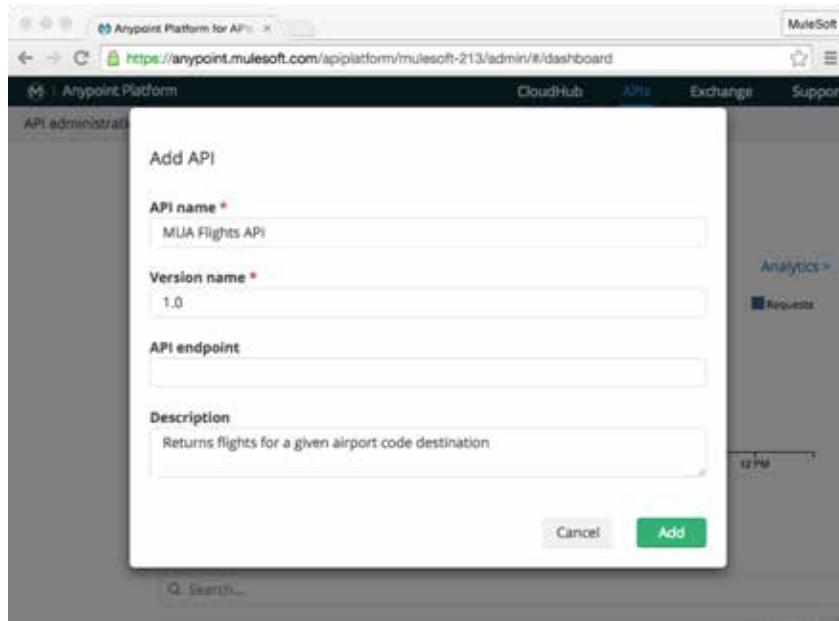
### Add a new API

1. In a browser, go to <http://anypoint.mulesoft.com> and log in.
15. Click the APIs link in the main menu bar.
16. Click the Add new API button.



17. In the Add API dialog box, enter the following and then click Add.

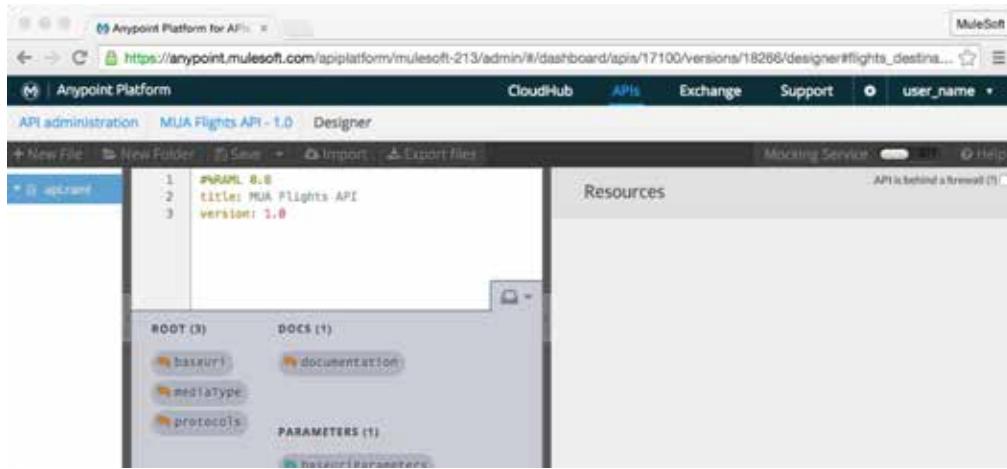
- API name: MUA Flights API
- Version name: 1.0
- Description: Returns flights for a given airport code destination



18. Take a look at the different sections and links for the API in the API administration.

A screenshot of the Anypoint Platform API administration page for the 'MUA Flights API - 1.0'. The page has tabs for 'CloudHub', 'APIs' (selected), 'Exchange', 'Support', and 'user\_name'. The main content area shows the API title 'MUA Flights API | 1.0' and a description 'Returns flights for a given airport code destination'. It includes three sections: 'API Definition' (with a link to 'Define API in API designer'), 'API Portal' (with a note 'No portal'), and 'API Status' (with a link to 'Configure endpoint').

19. Click the Define API in API designer link; the API Designer should open.



*Note: To change the background color from black to white, press Ctrl-Shift-T.*

## Add RAML root metadata

20. Place the cursor on a new line of code at the end of the file.  
21. Click the baseUri element in the API Designer shelf to add it to the RAML file.

*Note: Leave the default value #baseUri: http://server/api/{version} for now during development.*

## Add a RAML resource

22. Go to a new line of code and add a resource called flights.

```
/flights:
```

## Add a nested RAML resource

23. Indent by pressing the Enter key and then the Tab key.  
24. Add a nested resource to return flights for a particular destination.

```
5
6   /flights:
7     |  /{destination}:
```

## Add a RAML method

25. Go to a new line of code, press the Tab key, and then press the G key and then the Enter key to add a get method.  
26. Indent by pressing the Enter key and then the Tab key.

27. Scroll down in the API Designer shelf and locate and click responses.



```
1  #%RAML 0.8
2  title: MUA Flights API
3  version: 1.0
4  baseUri: http://server/api/{version}
5
6  ▼ /flights:
7  ▼   ▷ {destination}:
8  |   get:
9  |   |   responses:
```

The screenshot shows the MuleSoft API Designer shelf. At the top, there is a code editor window displaying RAML 0.8 code. Below the code editor are several buttons: 'ROOT (1)', 'DOCS (1)', 'protocols', 'description', 'PARAMETERS (3)', and 'SECURITY (1)'. A small icon of a car is visible on the left side of the shelf.

*Note: If you don't see the API Designer shelf, it is either minimized or there is an error in your code. To check if it is minimized, scroll down to the bottom of the browser window and look for its icon. If you see it, click it to expand it. If you don't see its icon and you also don't see the API Designer console, you probably have an error in your code, like a missing RAML definition.*



28. Indent and type 200::

```
responses:
```

```
  200:
```

29. Indent and type b and then press Enter to add a body parameter (or add it from the shelf).

30. Indent and type a and then press Enter to add application/json (or add it from the shelf).

```
1  #%RAML 0.8
2  title: MUA Flights API
3  version: 1.0
4  baseUri: http://server/api/{version}
5
6  ▼ /flights:
7  ▼   ▷ {destination}:
8  |   get:
9  |   |   responses:
10 |   |   |   200:
11 |   |   |   |   body:
12 |   |   |   |   |   application/json:
```

*Note: This is the minimal RAML file needed for Anypoint Studio and APIkit to generate the RESTful interface.*

## Add a query parameter

31. Go to a new line of code after get:.
32. Press the q key and then the Enter key to add queryParameters.
33. Indent and add a query parameter called airline.
34. Indent and add a default property and set it to all.
35. Return and add an enum property and set it to an array with values all, united, delta, and american.

```
5
6 ▼ /flights:
7 ▼   /{destination}:
8 ▼     get:
9 ▼       queryParameters:
10 ▼         airline:
11           | default: all
12           | enum: [all, united, delta, american]
13 ▼         responses:
14 ▼           200:
```

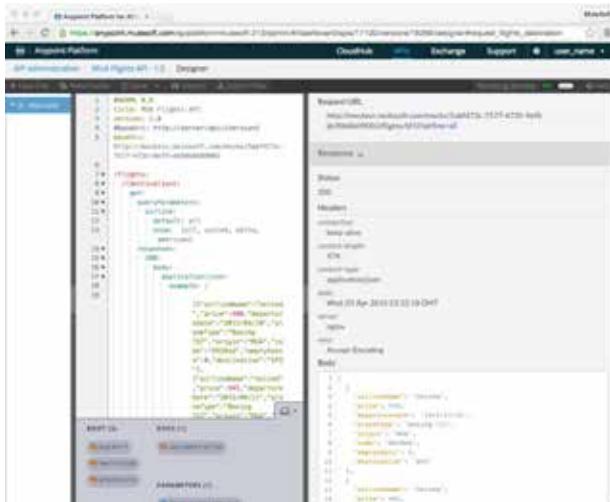
36. Click the Save button to save the RAML file.



## Walkthrough 10-2: Use API Designer to simulate an API

In this walkthrough, you will add example responses to the API definition and test it by running a live simulation. You will:

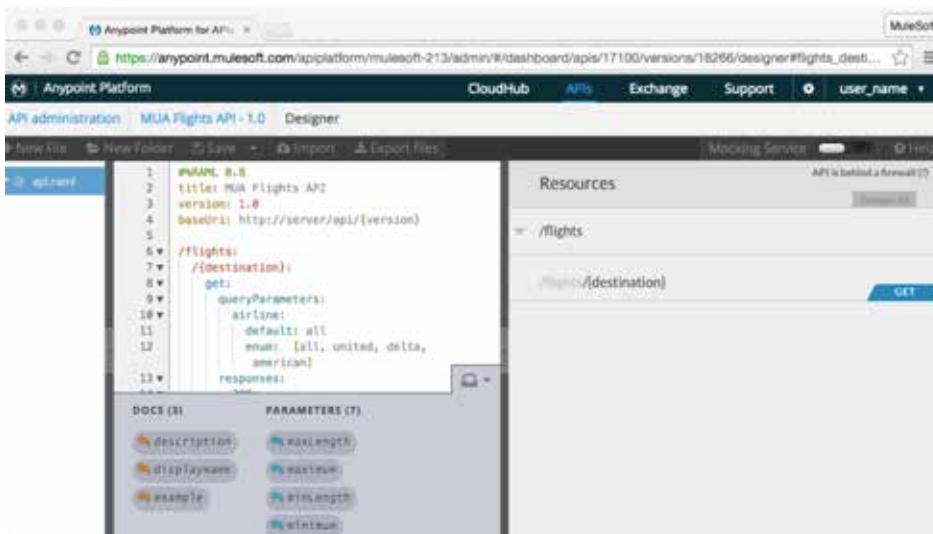
- Use the API console in API Designer.
- Use RAML to specify example responses for your API.
- Use the API Designer mocking service to run a live simulation of your API.



### Use the API Designer console

1. Return to your MUA Flights API in API Designer.
2. Look at the API console.

*Note: If you do not see the console, click the arrow located in the middle of the right edge of the browser window.*



37. Click the GET button for the /flights/{destination} resource; you should see the request information.

The screenshot shows the API documentation for the /flights/{destination} resource. At the top, there is a 'Request' section with a 'GET' button. Below it, under 'DESCRIPTION', is a placeholder for the response. Under 'URI PARAMETERS', there are two fields: 'destination' (required, string) and 'version' (required, (1.0)). Under 'QUERY PARAMETERS', there is a field for 'airline' with a note: 'one of (all, united, delta, american), default: all'. The entire interface has a clean, modern design with a light gray background and white text.

38. Scroll down to the Responses section and see the specification for a 200 status code.

39. Scroll back up and click the Try it button.

40. Enter a destination of SFO and click the GET button.

The screenshot shows the 'Try it' interface for the /flights/{destination} resource. It includes a 'Try it' button and an 'X' button. Under 'AUTHENTICATION', the 'Security Scheme' is set to 'Anonymous'. In the 'URI PARAMETERS' section, 'destination' is set to 'SFO' and 'version' is set to '1.0'. In the 'HEADERS' section, there is a plus sign icon. In the 'QUERY PARAMETERS' section, 'airline' is set to 'all'. At the bottom, there are three buttons: a blue 'GET' button, a black 'Clear' button, and a gray 'Reset' button. The overall layout is user-friendly with clear labeling and color-coded buttons.

41. Look at the response; you should get a 500 status code because the baseUri is not found.

The screenshot shows the API console interface. The 'Request' section contains a 'Request URL' field with the value `http://server/api/1.0/flights/SFO?airline=all`. The 'Response' section shows a 'Status' of 500. The 'Headers' section lists `connection: keep-alive`, `date: Wed, 01 Apr 2015 22:19:15 GMT`, `server: nginx`, and `transfer-encoding: chunked`. The 'Body' section contains the text `1 getaddrinfo ENOTFOUND`.

## Use the mocking service

42. Locate the Mocking Service slider in the menu bar above the console.

43. Slide it to on.

The screenshot shows the API console with the 'Mocking Service' slider set to 'ON'. The 'Resources' section shows a single entry under '/flights'.

44. Look at the new baseUri in the editor.

```
1  #%RAML 0.8
2  baseUri: http://mocksvc.mulesoft.com/mocks/824021bf-cb00-4b58-98f3-49fdbcec796c
3  title: MUA Flights API
4  version: 1.0
5  #baseUri: http://server/api/{version}
6
```

45. In the API console, click the Try it button for the /flights/{destination} resource again and then enter a destination and click GET; you should now get a 200 status code.

46. Scroll down and look at the body; you should get a general RAML message placeholder.

The screenshot shows the API Designer editor interface. On the left, there's a sidebar with 'Request' and 'Response' sections. Under 'Request', the 'URL' field contains 'http://mocksvc.mulesoft.com/mocks/824021bf-cb00-4b58-98f3-49fdbcec796c/flights/SFO?airline=all'. Under 'Response', the 'Status' is '200'. The 'Headers' section lists 'connection: keep-alive', 'content-length: 72', 'content-type: application/json', 'date: Wed, 01 Apr 2015 22:22:29 GMT', 'server: nginx', and 'vary: Accept-Encoding'. The 'Body' section contains a JSON placeholder: '1 { 2 "message": "RAML had no response information for application/json" 3 }'.

47. Close the /flights/{destination} window.

## Add examples

37. Return to the course snippets.txt file and copy the Example JSON.

```
[{"airlineName": "United", "price": 400, "departureDate": "2015/03/20", "planeType": "Boeing 737", "origination": "MUA", "flightCode": "ER38sd", "availableSeats": 0, "destination": "SFO"}, {"airlineName": "United", "price": 945, "departureDate": "2015/09/11", "planeType": "Boeing 757", "origination": "MUA", "flightCode": "ER39rk", "availableSeats": 54, "destination": "SFO"}, {"airlineName": "United", "price": 954, "departureDate": "2015/02/12", "planeType": "Boeing 777", "origination": "MUA", "flightCode": "ER39rj", "availableSeats": 23, "destination": "SFO"}]
```

48. Return to the API Designer editor and indent from the application/json element and add an example element.

49. Add a space and then a | after the example element.

50. Indent and paste the JSON array you just copied.

```
16 ▼ | | | | body:
17 ▼ | | | |   application/json:
18 ▼ | | | |     example: |
19 | | | |       [{"airlineName": "United", "price": 400, "departureDate": "2015/03/20", "planeType": "Boeing
737", "origination": "MUA", "flightCode": "ER38sd", "availableSeats": 0, "destination": "SFO"}, {"airlineName": "United", "price": 945, "departureDate": "2015/09/11", "planeType": "Boeing
757", "origination": "MUA", "flightCode": "ER39rk", "availableSeats": 54, "destination": "SFO"}, {"airlineName": "United", "price": 954, "departureDate": "2015/02/12", "planeType": "Boeing
777", "origination": "MUA", "flightCode": "ER39rj", "availableSeats": 23, "destination": "SFO"}]
```

## View the example data in the simulation

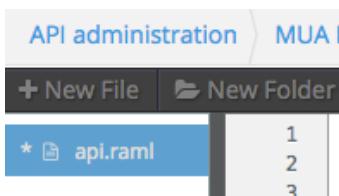
51. Return to the API console.
52. Click the GET button for the /flights/{destination} resource again.
53. Click Try it, enter a destination, and click GET.
54. Look at the response body; you should now get your example data.

Body

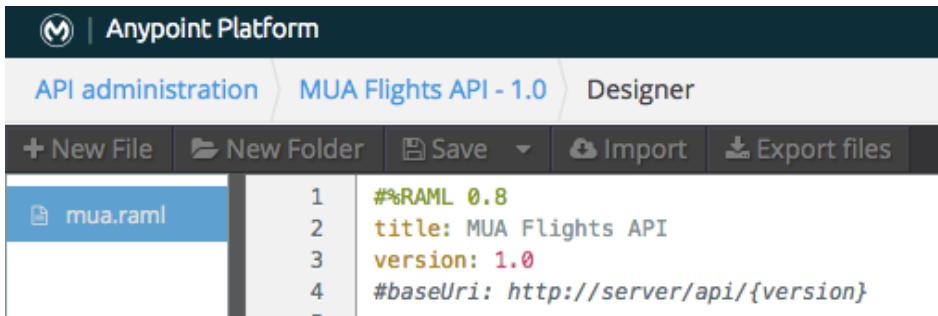
```
1 [ 
2   {
3     "airlineName": "United",
4     "price": 400,
5     "departureDate": "2015/03/20",
6     "planeType": "Boeing 737",
7     "origination": "MUA",
8     "flightCode": "ER38sd",
9     "availableSeats": 0,
10    "destination": "SFO"
11  },
12  {
13    "airlineName": "United",
14    "price": 945,
15    "departureDate": "2015/09/11",
16    "planeType": "Boeing 757",
17    "origination": "MUA",
18    "flightCode": "ER39rk".
```

## Save and export the RAML file

55. Slide the Mocking Service slider to off.
56. Save the file.
57. Right-click api.raml in the left pane and select Rename.

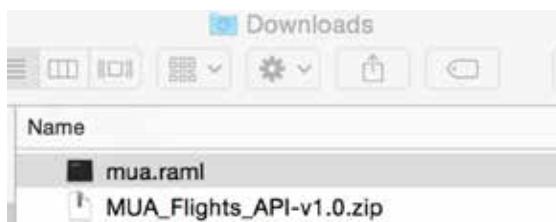


58. Change the name to mua.raml and click OK.
59. Click the Export files button to download the RAML file.



```
1  #%RAML 0.8
2  title: MUA Flights API
3  version: 1.0
4  #baseUri: http://server/api/{version}
```

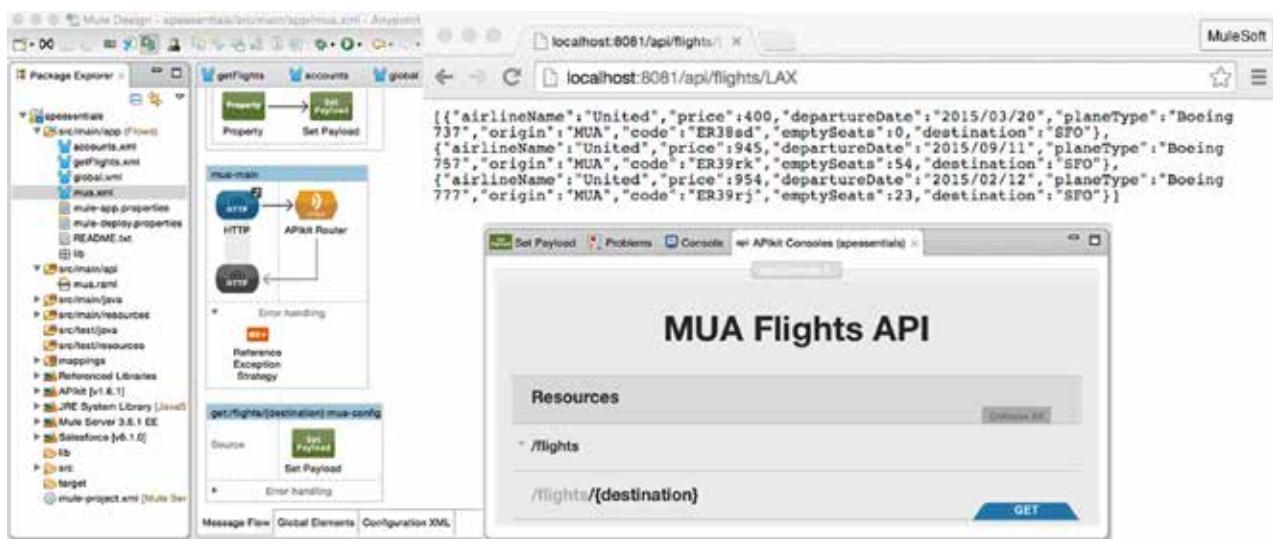
60. Locate the MUA\_Flights\_API-v1.0.zip file that is downloaded.
61. Expand the ZIP and locate the mua.raml file it contains.



## Walkthrough 10-3: Use Anypoint Studio to create a RESTful API interface from a RAML file

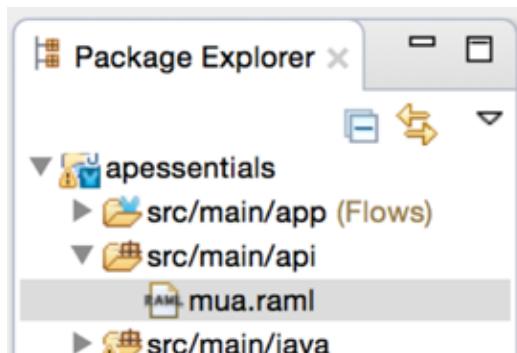
In this walkthrough, you will generate a RESTful interface from the RAML file. You will:

- Add a RAML file to your apessentials project.
- Use Anypoint Studio and APIkit to generate a RESTful web service interface from a RAML file.
- Test the web service in the APIkit Consoles view and a browser.



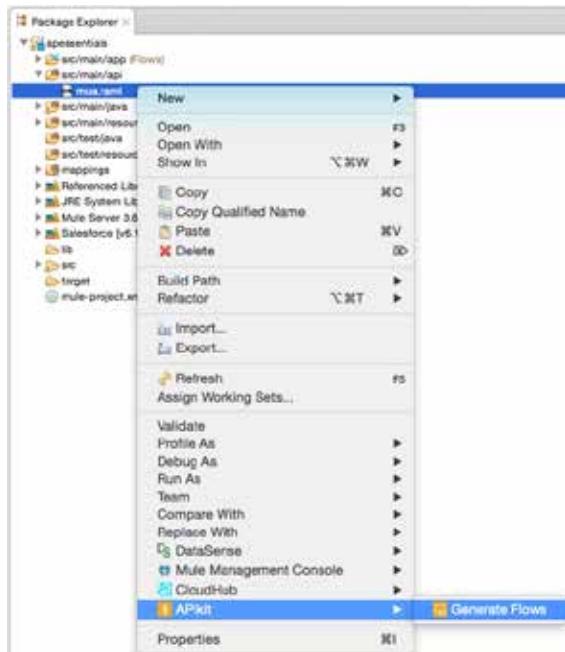
### Add the RAML file

1. Drag the mua.raml file into the src/main/api folder in your apessentials project.



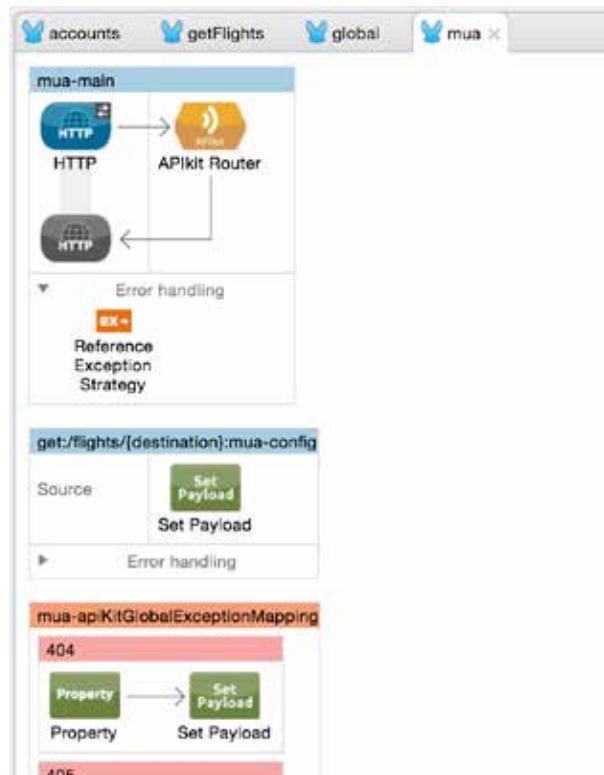
## Create a RESTful interface

2. Right-click the RAML file and select APIkit > Generate Flows.



*Note: If Generate Flows is disabled, there is an error in your RAML file and you need to return to API Designer and fix it and then re-add it to Anypoint Studio.*

3. Wait until a new mua.xml file is generated and opened.

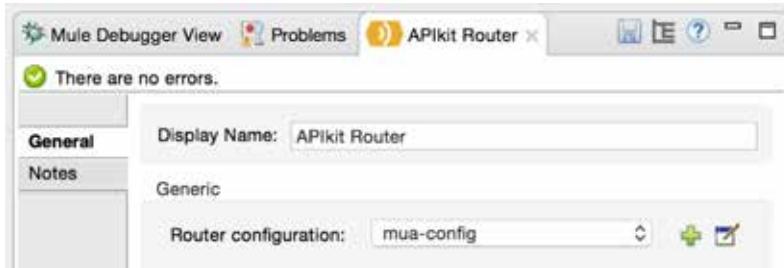


## Modify the HTTP endpoint

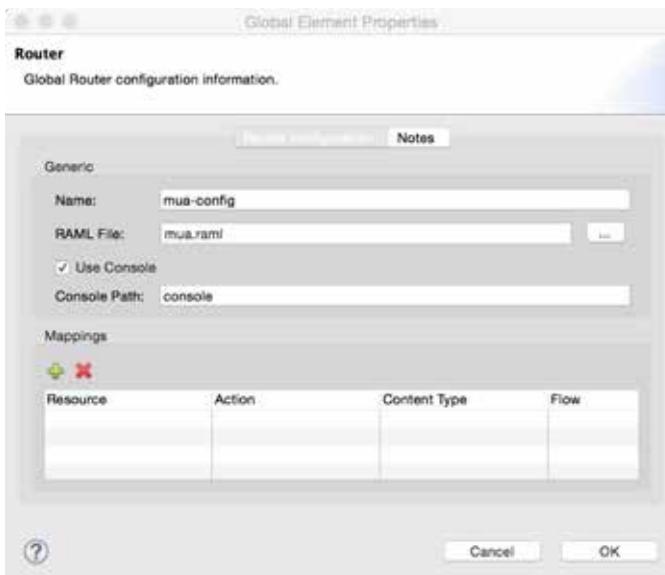
4. In mua.xml, double-click the HTTP connector in the mua-main flow.
5. Look at the path.
6. Click the Edit button next to Connector Configuration.
7. In the Global Element Properties dialog box, view the host and port values and click OK.
8. Change the connector configuration to the existing HTTP\_Listener\_Configuration.
9. Switch to the Global Elements view.
10. Select the mua-httpListenerConfig element and click Delete.
11. Return to the Message Flow view.

## Examine the APIkit router

12. Double-click the APIkit Router.
13. In the APIkit Router Properties view, see the router configuration is set to mua-config.
14. Click the Edit button next to Connector Configuration.



15. In the Global Element Properties dialog box, look at the values and then click OK.

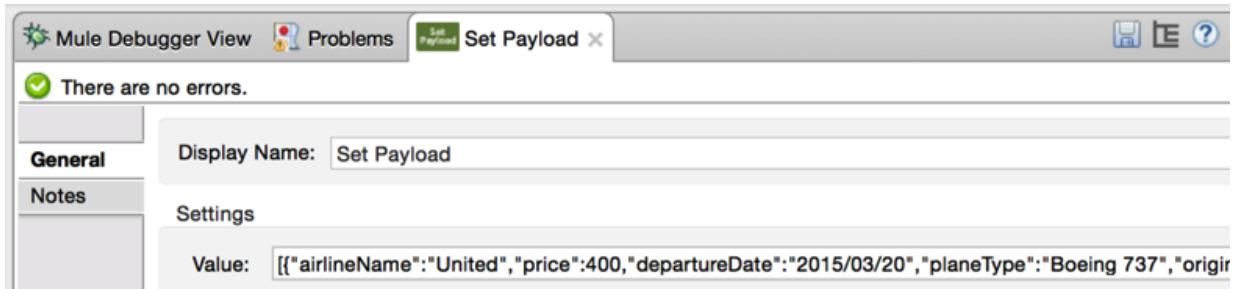


## Look at the generated resource flow

16. Look at the name of the other flow created: get:/flights/{destination}:mua-config.



17. Double-click the Set Payload transformer and look at the value.



18. Switch to the Configuration XML view and look at the generated code.

## Test the web service in the APIkit Consoles view

19. Run the application.
20. Look at the APIkit Consoles view that opens.

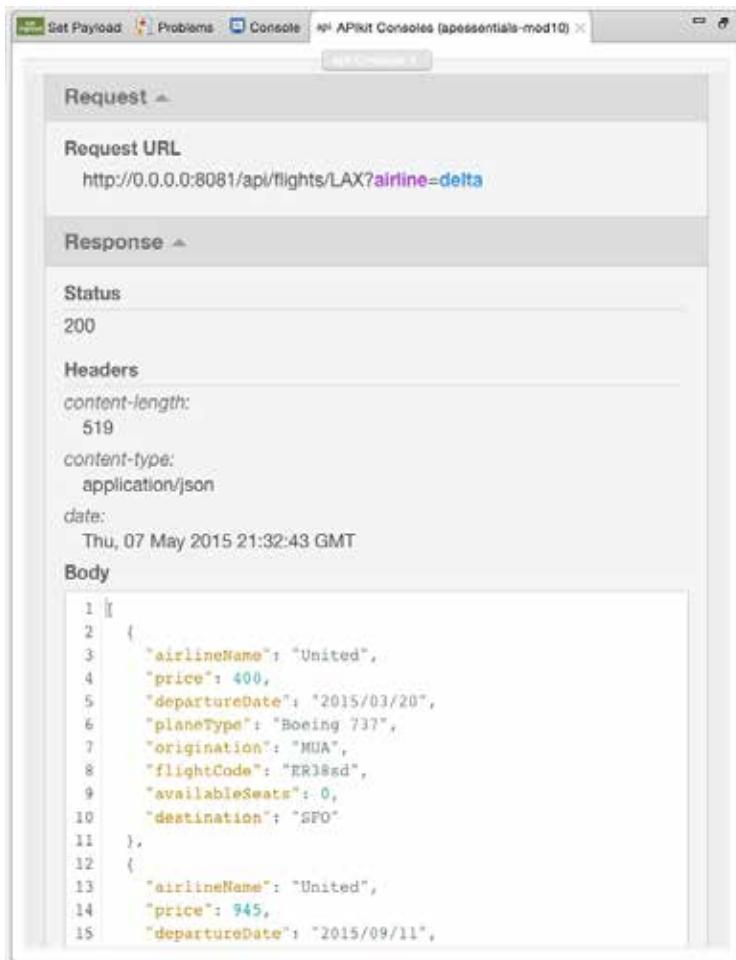


Note: If the API is not displayed in the APIkit Console, change your HTTP Listener Configuration host from 0.0.0.0 to localhost.

21. Click the GET button for /flights/{destination}.
22. Click the Try It button.
23. Enter a destination of LAX (or any other value).

24. Click in the airline query parameter field and select one of the enumerated values, like delta.

25. Click the GET button; you should see the example data displayed.



Request URL  
http://0.0.0.0:8081/api/flights/LAX?airline=delta

Status  
200

Headers  
content-length:  
519  
content-type:  
application/json  
date:  
Thu, 07 May 2015 21:32:43 GMT

Body

```
1 [
2   {
3     "airlineName": "United",
4     "price": 400,
5     "departureDate": "2015/03/20",
6     "planeType": "Boeing 737",
7     "origination": "MUA",
8     "flightCode": "ER38sd",
9     "availableSeats": 0,
10    "destination": "SFO"
11  },
12  {
13    "airlineName": "United",
14    "price": 945,
15    "departureDate": "2015/09/11",
```

## Test the web service in a browser

26. Go to a browser, and make a request to <http://localhost:8081/api/flights/LAX> (or any other destination); you should see the example data returned.



localhost:8081/api/flights/LAX

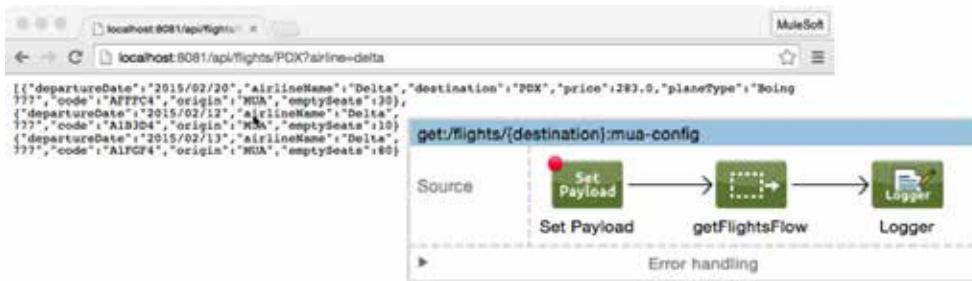
MuleSoft

```
[{"airlineName": "United", "price": 400, "departureDate": "2015/03/20", "planeType": "Boeing 737", "origination": "MUA", "flightCode": "ER38sd", "availableSeats": 0, "destination": "SFO"}, {"airlineName": "United", "price": 945, "departureDate": "2015/09/11", "planeType": "Boeing 757", "origination": "MUA", "flightCode": "ER39rk", "availableSeats": 54, "destination": "SFO"}, {"airlineName": "United", "price": 954, "departureDate": "2015/02/12", "planeType": "Boeing 777", "origination": "MUA", "flightCode": "ER39rj", "availableSeats": 23, "destination": "SFO"}]
```

## Walkthrough 10-4: Use Anypoint Studio to implement a RESTful web service

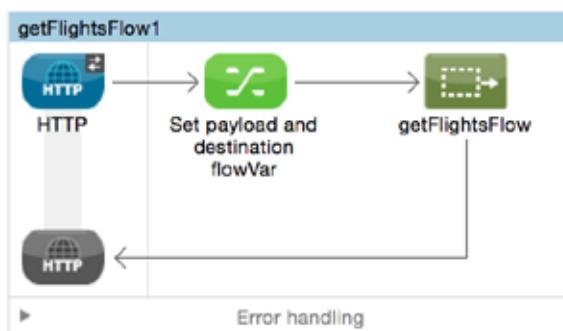
In this walkthrough, you will wire the RESTful web service interface up to your back end logic. You will:

- Split the getFlightsFlow into two flows: one that transforms the form post data into a FlightRequest object and the other that gets all the flights.
- Determine how to reference the web service destination and airline values.
- Call the backend flow.
- Test the web service in the APIkit Consoles view and a browser.



### Modify the existing flows

67. Return to getFlights.xml.
68. Drag out a Flow scope element from the palette and drop it at the top of the canvas.
69. Give it a display name of getFlightsFromFormFlow.
70. Drag the HTTP endpoint from getFlightsFlow and drop it in the source section of getFlightsFromFormFlow.
71. Drag the JSON to Object transformer from getFlightsForm and drop it in the process section of getFlightsFromFormFlow.
72. Drag out a Flow Reference component from the palette and drop it in the process section of getFlightsFromFormFlow.
73. In the Flow Reference Properties view, set the flow name to getFlightsFlow.



## Determine how to reference the web service destination and airline values

74. Return to mua.xml.
75. Add a Logger to the get:/flights/{destination} flow.
76. Add a breakpoint to the Set Payload transformer in the get:/flights/{destination} flow.
77. Save all the files and debug the application.
78. Make a request to <http://localhost:8081/api/flights/LAX>.
79. In the Mule Debugger view, look at the inbound properties; you should see a query parameter called airline with a value of all.

Inbound Variables Outbound Session Record			
Name	Value	Type	
⑧ connection	keep-alive	java.lang.String	
⑧ host	localhost:8081	java.lang.String	
⑧ http.listener.path	/api/*	java.lang.String	
⑧ http.method	GET	java.lang.String	
▼ ⑨ http.query.params {airline=all}	{airline=all}	java.util.HashMap	
▼ ⑩ 0	airline=all	java.util.HashMap\$Entry	
⑪ key	airline	java.lang.String	
⑪ value	all	java.lang.String	
⑧ http.query.string		java.lang.String	
⑧ http.remote.ad...	/0:0:0:0:0:0:1:61473	java.lang.String	

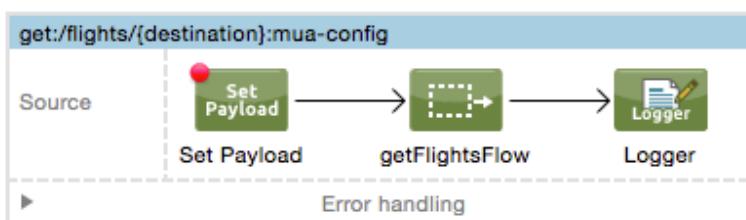
80. Click the Variables tab; you should see a variable called destination with a value of LAX.

Inbound Variables Outbound Session Record			
Name	Value	Type	
⑧ _ApikitResponseTran...	yes	java.lang.String	
⑧ _ApikitResponseTran...	application/json	java.lang.String	
► ⑨ _ApikitResponseTran...	[MimeType(type='appl...]	java.util.ArrayList	
⑧ destination	LAX	java.lang.String	

81. Click the Resume button.

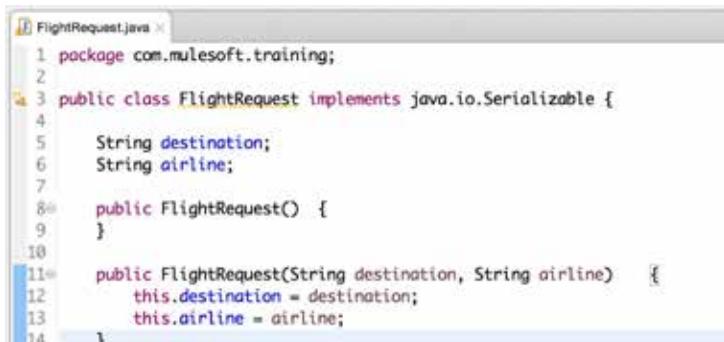
## Call the backend flow

82. Return to the get:/flights/{destination} flow.
83. Add a Flow Reference component after the Set Payload transformer.
84. In the Flow Reference Properties view, set the flow name to getFlightsFlow.



## Review the FlightRequest Java class

85. Open the com.mulesoft.training.FlightRequest.java class located in the project's src/main/java folder and examine the code.



```
FlightRequest.java X
1 package com.mulesoft.training;
2
3 public class FlightRequest implements java.io.Serializable {
4
5     String destination;
6     String airline;
7
8     public FlightRequest() {
9 }
10
11     public FlightRequest(String destination, String airline) {
12         this.destination = destination;
13         this.airline = airline;
14     }
}
```

## Set the payload to a Java object with flight request data

86. Double-click the Set Payload transformer in the get:/flights/{destination} flow.
87. In the Properties view, set the value to a new instance of com.mulesoft.training.FlightRequest, passing destination and message.inboundProperties.'http.query.params'.airline to it as arguments.

*Note: You can also copy this expression from the course snippets.txt file.*



## Test the web service in the APIkit Consoles view

88. Save the files and run the application.
89. In the APIkit Consoles view, click the GET button for /flights/{destination}.
90. Click the Try It button.

91. Enter a destination of LAX (or any other value) and click the GET button; you should now see the real data (for LAX) displayed.

```
Request URL: http://0.0.0.0:8081/api/flights/LAX?airline=all
Status: 200
Headers:
content-length: 1472
content-type: application/json
date: Wed, 01 Apr 2015 23:55:06 GMT
charset: UTF-8
Body:
1 [
2   {
3     "airlineName": "Delta",
4     "departureDate": "2015/02/11",
5     "destination": "LAX",
6     "price": 189.99,
7     "flightCode": "Delta 123",
8     "emptySeats": 1,
9     "origin": "MIA",
10    "planeType": "Boeing 737"
11  }
12 ]
```

92. Scroll up in the /flights/{destination} resource window.

93. Set the airline query parameter to united.

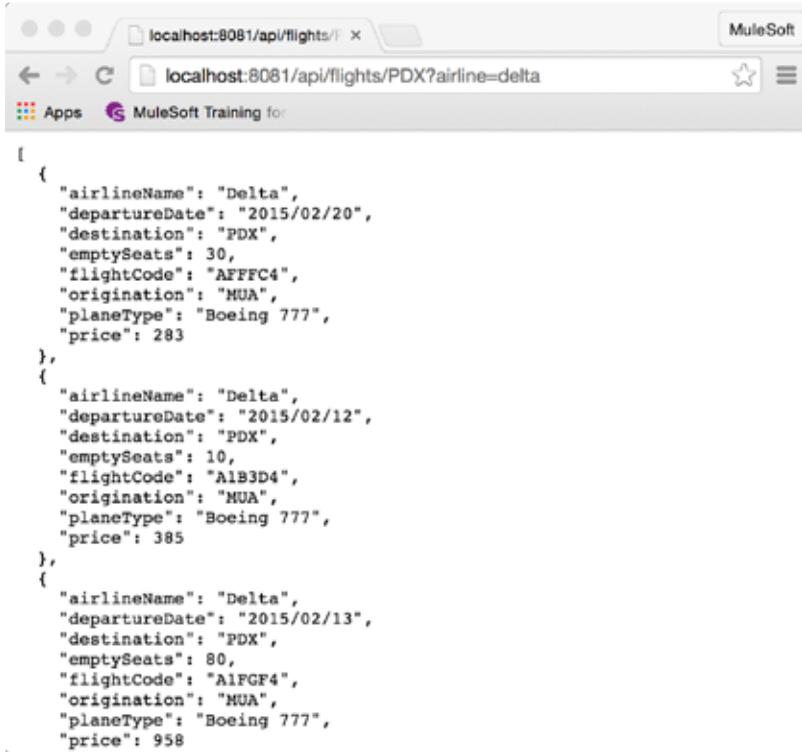
94. Click the GET button; you see should now see only the flights for LAX on United.

## Test the web service in a browser

95. Make a request to <http://localhost:8081/api/flights/CLE>; you should see data for all the flights to CLE returned.

```
localhost:8081/api/flights/CLE
[{"airlineName": "American Airlines", "departureDate": "2015-01-20T00:00:00", "destination": "CLE", "emptySeats": 5, "flightCode": "crwl000", "origin": "MIA", "planeType": "Boeing 737", "price": 200}, {"airlineName": "United", "departureDate": "2015-01-20T00:00:00", "destination": "CLE", "emptySeats": 13, "flightCode": "ERKfd1", "origin": "MIA", "planeType": "Boeing 747", "price": 245}, {"airlineName": "American Airlines", "departureDate": "2015-01-25T00:00:00", "destination": "CLE", "emptySeats": 7, "flightCode": "crw0123", "origin": "MIA", "planeType": "Boeing 747", "price": 300}]
```

96. Make another request to <http://localhost:8081/api/flights/PDX?airline=delta> to retrieve flights for PDX on Delta; ensure the correct data is returned.



A screenshot of a web browser window. The address bar shows the URL [localhost:8081/api/flights/PDX?airline=delta](http://localhost:8081/api/flights/PDX?airline=delta). The page content displays a JSON array of flight records:

```
[{"airlineName": "Delta", "departureDate": "2015/02/20", "destination": "PDX", "emptySeats": 30, "flightCode": "AFFFC4", "origination": "MUA", "planeType": "Boeing 777", "price": 283}, {"airlineName": "Delta", "departureDate": "2015/02/12", "destination": "PDX", "emptySeats": 10, "flightCode": "A1B3D4", "origination": "MUA", "planeType": "Boeing 777", "price": 385}, {"airlineName": "Delta", "departureDate": "2015/02/13", "destination": "PDX", "emptySeats": 80, "flightCode": "A1FGF4", "origination": "MUA", "planeType": "Boeing 777", "price": 958}]
```

# Module 11: Deploying Applications

Mule United Airport Flight - apessentials37.cloudhub.io/flights

MuleSoft

Mule United Airport

SFO - San Francisco | All Airlines | Find Flights

Available Flights

Flight Code: A14244  
Airline Name: Delta  
Destination: SFO  
Plane Type: Boeing 787  
Price: \$294  
Departure Date: 2015/02/12

Flight Code: A14244  
Airline Name: Delta  
Destination: SFO  
Plane Type: Boeing 787  
Price: \$294  
Departure Date: 2015/02/12

Anypoint Management Center - https://anypoint.mulesoft.com/cloudhub/#/console/home/applications

Anypoint Platform

CloudHub APIs Exchange Support

DEVELOPMENT Applications

Applications Deploy application Search Applications

Name	Server	Status	File
apessentials37	CloudHub	Started	apessentials.zip

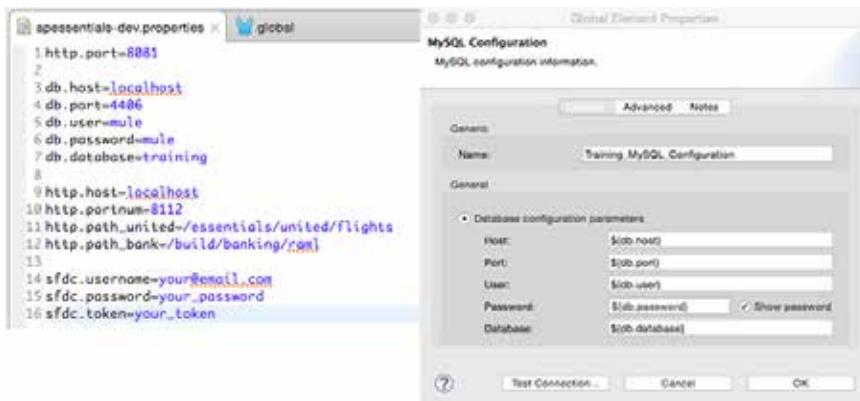
In this module, you will learn:

- About the options for deploying your applications.
- What CloudHub is.
- About when and how to use application properties.
- (Optional) To deploy and run applications in the cloud.
- (Optional) To deploy and run applications on-prem.

## Walkthrough 11-1: Use application properties

In this walkthrough, you will introduce properties into your Mule application. You will:

- Create a properties file for your application.
- Create a Properties Placeholder global element.
- Parameterize the HTTP Listener connector port.
- Define and use Database connector properties.



### Create a properties file

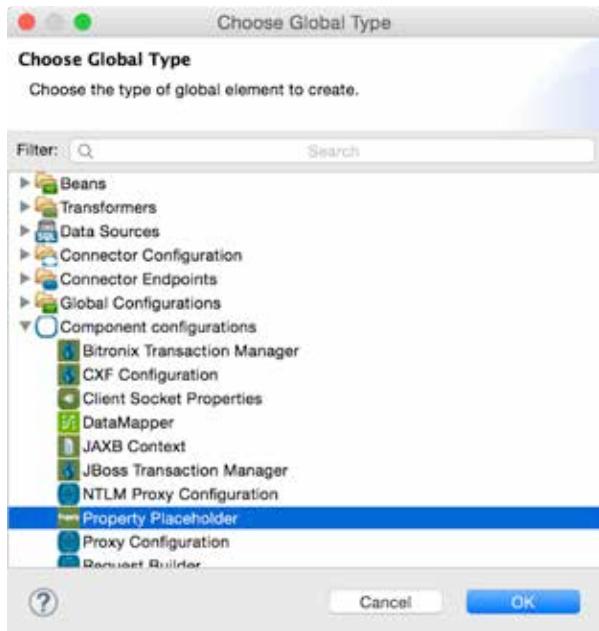
1. Right-click the src/main/resources folder in the Package Explorer and select New > File.
2. Name the file apessentials-dev.properties and click Finish.



### Create a Properties Placeholder global element

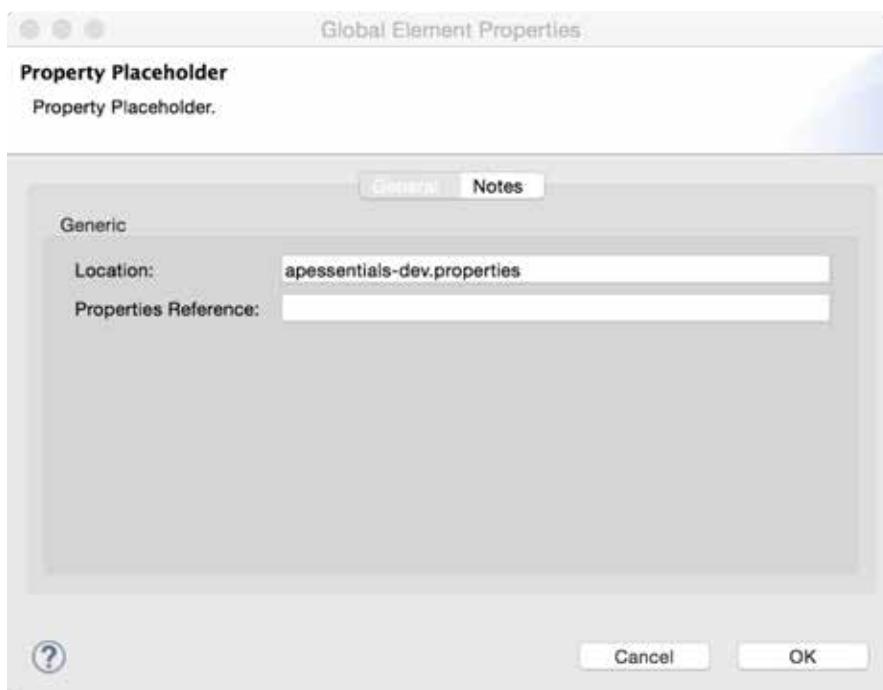
3. Open global.xml.
4. Navigate to the Global Elements view and click Create.

5. In the Choose Global Type dialog box, select Component configurations > Property Placeholder and click OK.



6. In the Global Element Properties dialog box, set the location to apessentials-dev.properties and click OK.

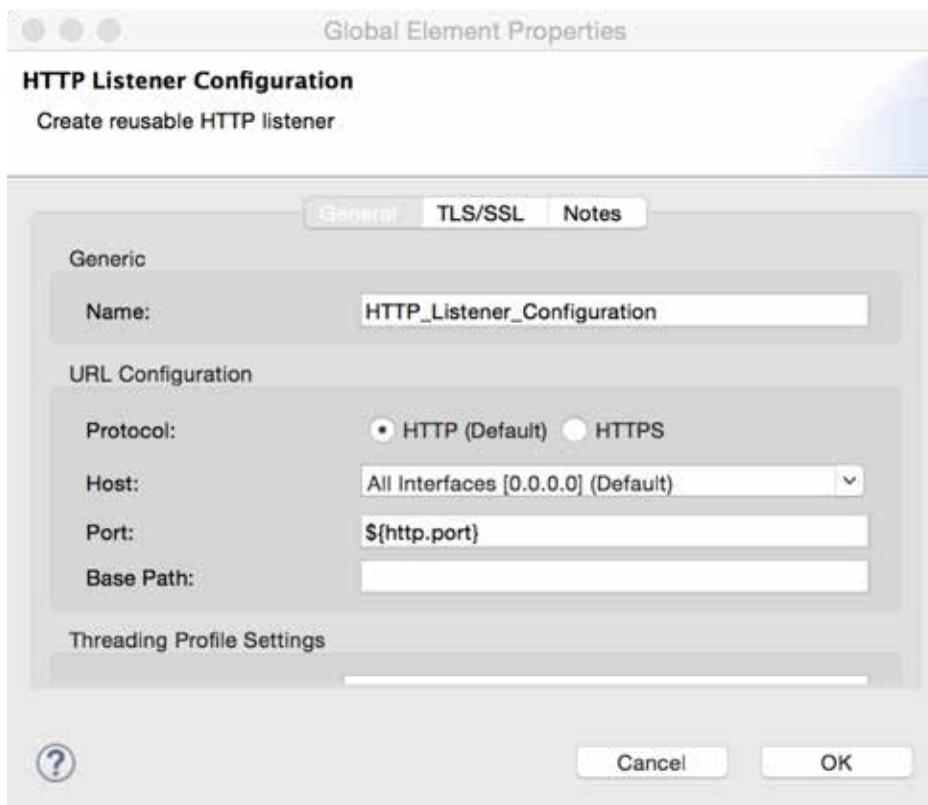
*Note: When setting the location, make sure you do not use a full path.*



## Parameterize the HTTP Listener port

7. Return to apessentials-dev.properties.
8. Create a property called http.port and set it to 8081.  

```
http.port=8081
```
9. Return to global.xml.
10. Double-click the HTTP Listener Configuration global element.
11. Change the port to the application property, \${http.port}.
12. Click OK.

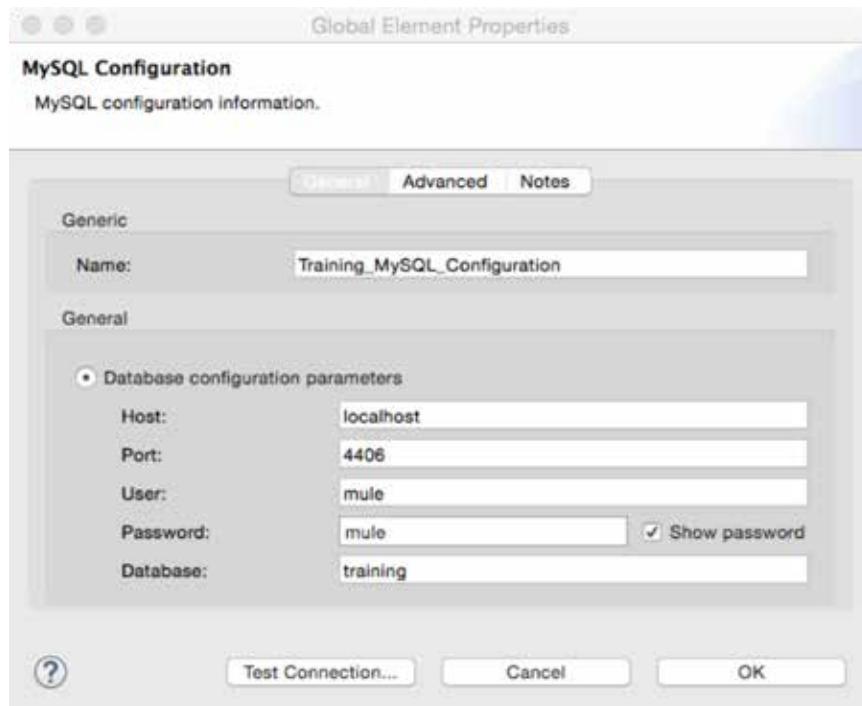


## Test the application

13. Save all the files and run the application.
14. Make a request <http://localhost:8081/flights> and confirm it still works.

## Parameterize database credentials

15. Return to global.xml.
16. Double-click the MySQL Configuration global element to edit it.



17. Copy the host value and then click OK.
18. Return to your properties file and create a property called db.host and paste the value you copied.
19. Create additional properties for the database port, user, password, and database values.

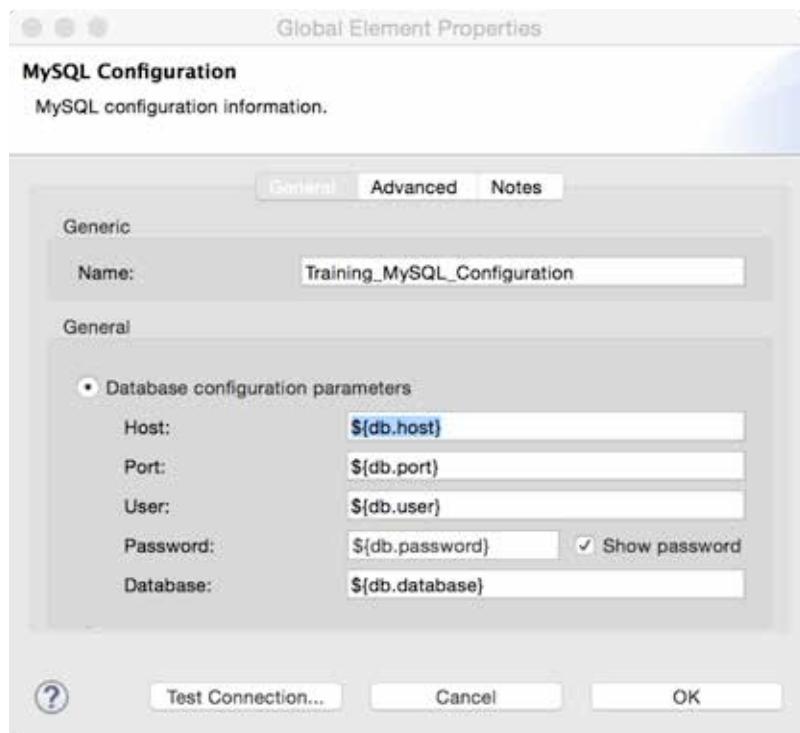
The screenshot shows a properties file named 'apessentials-dev.properties'. The file contains the following entries:

```
1 http.port=8081
2
3 db.host=localhost
4 db.port=4406
5 db.user=mule
6 db.password=mule
7 db.database=training
```

The file is currently open in a text editor, and the 'global' tab is selected in the tabs bar above the code area.

20. Save the file.

21. Return to the MySQL Configuration global element and replace the hard-coded configuration parameters with property placeholders.



22. Test the connection and make sure it still works.

23. Click OK.

## Test the application

24. Save the file and run the application.

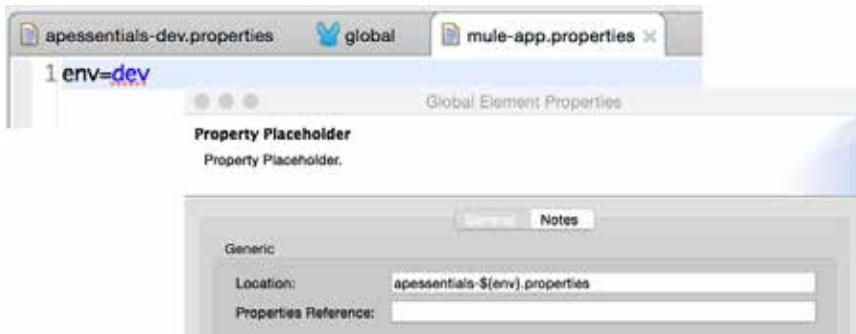
25. Make a request <http://localhost:8081/flights> and confirm it still works.

*Note: If you have time, also parameterize your Salesforce credentials and your HTTP Request properties.*

## Walkthrough 11-2: Dynamically specify property files

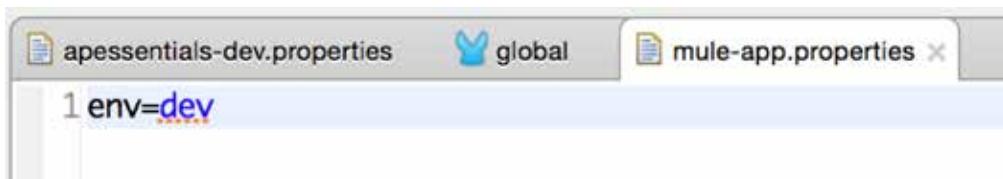
In this walkthrough, you will set the property file to use dynamically. You will:

- Define an environment property value in mule-app.properties.
- Use the environment property in the Property Placeholder.



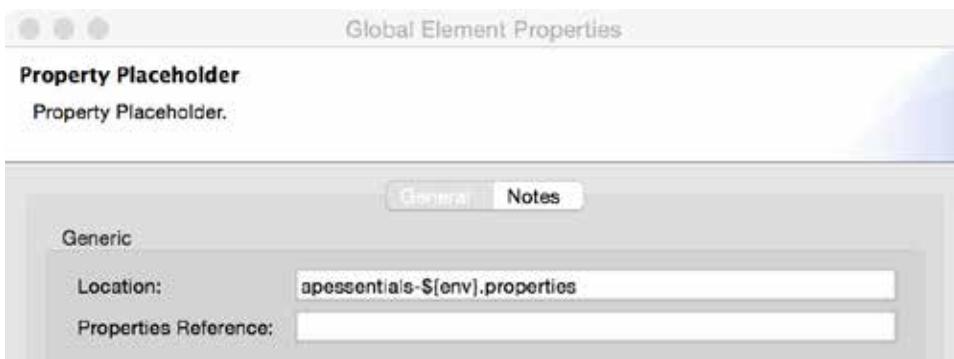
### Define an environment property value in mule-app.properties

1. Open mule-app.properties.
2. Define a property called env and set it to dev.
3. Save the file.



### Use the environment property in the Property Placeholder

4. Return to global.xml.
5. Double-click the Property Placeholder to edit it.
6. In the Global Element Properties dialog box, change the location to apessentials-\${env}.properties and click OK.



## Test the application

7. Save all the files to redeploy the application.
8. Make a request <http://localhost:8081/flights> and confirm it still works and returns flights.



## Walkthrough 11-3: (Optional) Deploy an application to the cloud

In this walkthrough, you will deploy your apessentials application to the cloud. You will:

- Deploy an application to CloudHub from Anypoint Studio.
- Run the application on its new, hosted domain.
- View application data in CloudHub.

The screenshot shows two windows side-by-side. On the left is Anypoint Studio displaying the 'Mule United Airport' application with flight details for a flight from SFO to SFO. On the right is the 'Anypoint Platform' interface, specifically the 'CloudHub' section under 'CloudHub'. It shows a table with one row for the 'apessentials37' application, which is deployed to 'CloudHub' and has a status of 'Started'. A green 'Deploy application' button is visible above the table.

## Access CloudHub

1. In a browser, go to <http://anypoint.mulesoft.com> and log in.

The screenshot shows a web browser window with the URL <https://anypoint.mulesoft.com/accounts/#/signin>. The page is titled 'Anypoint Platform' and features a 'Sign in' form with fields for 'Username' and 'Password', and a large green 'Sign in' button. There are links for 'Reset password' and 'Forgot username?' at the bottom of the form. The MuleSoft logo is in the top left corner, and there are 'Sign up' and 'Don't have an account?' links in the top right corner.

9. Click the CloudHub link in the main menu bar.

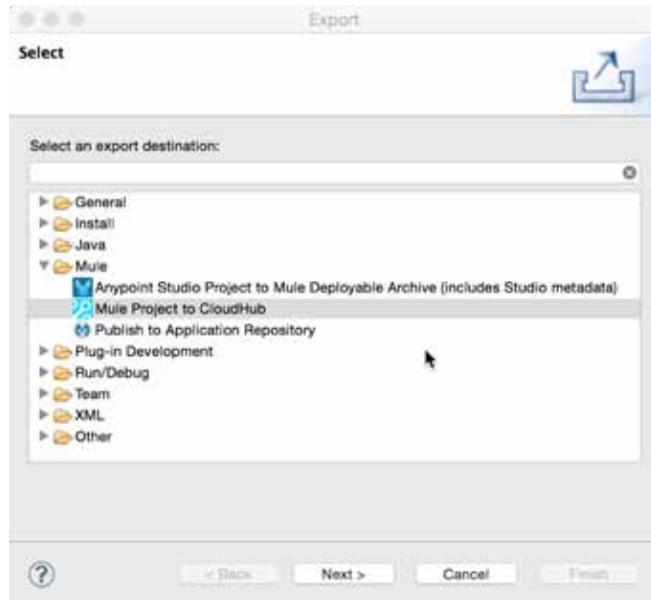
The screenshot shows a web browser window for the Anypoint Management Center at the URL <https://anypoint.mulesoft.com/cloudhub/#/console/home/applications>. The top navigation bar includes links for MuleSoft Training, Anypoint Platform, CloudHub (which is highlighted in blue), APIs, Exchange, Support, and a settings gear icon. Below the navigation bar, there are tabs for PRODUCTION and APPLICATIONS, with APPLICATIONS being the active tab. A green button labeled "Deploy application" is visible. A search bar with the placeholder "Search Applications" is also present.

10. Look at the applications; for a new account, there will be no applications listed.

## Deploy the application to CloudHub

11. Return to Anypoint Studio.
12. Select File > Export.
13. In the Export dialog box, select Mule > Mule Project to CloudHub.

*Note: You can also right-click the project in the Package Explorer and select CloudHub > Deploy to CloudHub.*



14. Click Next.
15. In the login dialog box, enter your Anypoint Platform credentials.
16. In the Deploy to CloudHub dialog box, select apessentials as the project to deploy.

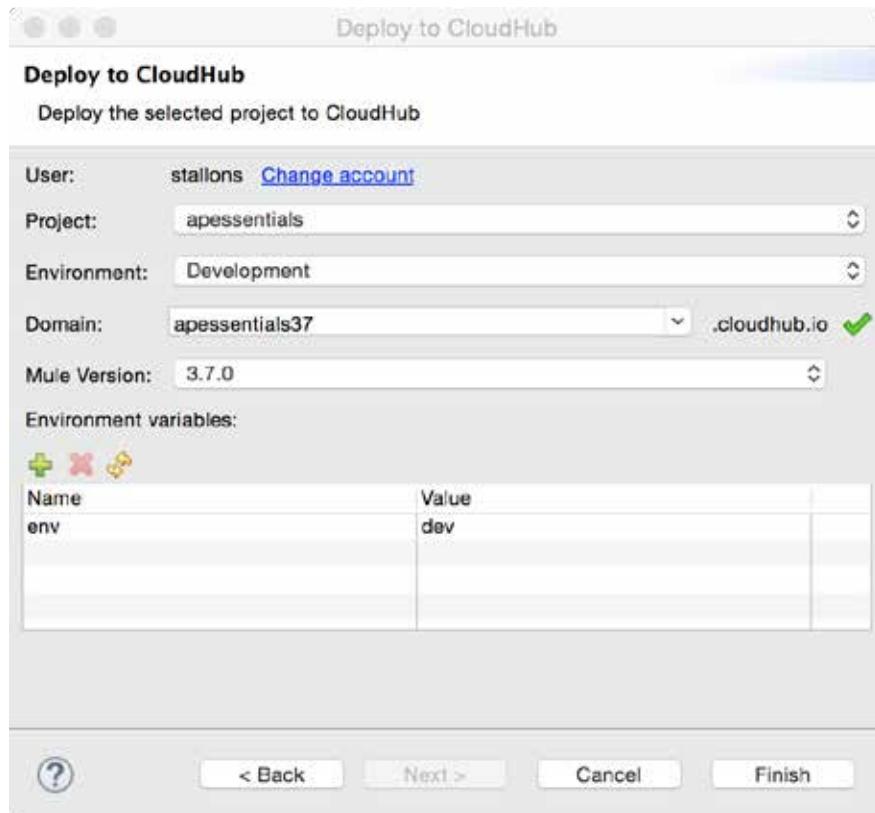
17. Wait while your credentials are used to successfully connect to the Anypoint Platform; once they are, the environment drop-down will be populated based on your CloudHub settings.
18. Select an environment; you may or may not have more than one option.
19. Enter a domain name – the URL that will be used to access the application on CloudHub.

*Note: Because the domain name must be unique across all applications on CloudHub, you may want to use your last name or company name as part of the domain name, for example, apessentials-{your\_lastname}. The availability of the domain is instantly checked and you will get a green check mark if it is available.*

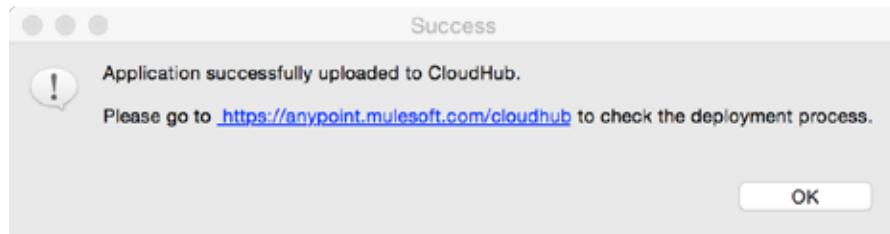
20. Set the Mule version to the version your project is using, like 3.7.1

*Note: If you don't know what version it is using, look at the Package Explorer and find a library folder with the name of the server being used, like Mule Server 3.7.1. EE.*

21. Look at the environment variables; you should see the env variable set to dev.



22. Click Finish.
23. Wait until you get a notice of a successful upload (or an error).



## Find the application on CloudHub

24. Return to CloudHub and see your application listed.
25. If you see a blue circle next to the application name, wait until it turns green (or red).

*Note: To follow the deployment process, mouse over the application name in CloudHub, select Logs, and then watch the Live View of the Logs file.*

A screenshot of the Anypoint Management Center interface, specifically the CloudHub section. The URL in the browser is https://anypoint.mulesoft.com/cloudhub/#/console/home/applications. The page shows a table of applications with the following data:

Name	Server	Status	File
apessentials37	CloudHub	Started	apessentials.zip

*Note: There will be a green circle next to the application if it was successfully deployed. There will be a red circle next to the application if it was not. If your application did not successfully deploy, mouse over its name in CloudHub, select Logs, and then examine the Logs to help figure out why the application did not deploy. If you had errors when deploying, troubleshoot them, fix them, and then redeploy.*

## Test the hosted application

26. Click the apessentials application in the application list on CloudHub; this takes you to the Dashboard for the application.
27. Click the link to the application on its new domain: apessentials-{your\_lastname}.cloudhub.io.

28. Click the link and then add the correct path, <http://apessentials-lastname/flights>; your application should work as before but now it is running on CloudHub.

Mule United Airport

SFO - San Francisco | All Airlines | Find Flights

### Available Flights

Flight Code: rree1093  
Airline Name: American Airlines  
Destination: SFO  
Plane Type: Boeing 737  
Price: \$142  
Departure Date: 2015-02-11T00:00:00  
Available Seats: 1

---

Flight Code: A14244  
Airline Name: Delta  
Destination: SFO  
Plane Type: Boeing 787  
Price: \$294  
Departure Date: 2015/02/12

*Note: You can also test your other endpoints: <http://apessentials-lastname/sfdc> and <http://apessentials-lastname/api/flights/SFO>,*

## Explore application data on CloudHub

29. Return to CloudHub; you should see your application listed.

Anypoint Management Center

https://anypoint.mulesoft.com/cloudhub/#/console/home/applications

CloudHub | APIs | Exchange | Support

### Applications

Deploy application

Name	Server	Status	File
apessentials37	CloudHub	Started	apessentials.zip

30. Click the application and then click the Manage Application button that appears on the right side.

31. Click settings in the left-side menu.

32. Change the worker size to 0.1 vCores.

The screenshot shows the Anypoint Management Console interface for managing applications. The application 'apessentials37' is selected. The 'Runtime' tab is active, displaying the following configuration:

- Runtime version: 3.7.0
- Worker size: 0.1 vCores
- Workers: 1

A warning message is displayed: "⚠ Your current subscription allows only one worker per application". The 'Properties' tab is also visible, showing the environment variable 'env=dev'.

33. Click Properties; you should see your env environment variable and its current value.

The screenshot shows the Anypoint Management Console interface for managing applications. The application 'apessentials37' is selected. The 'Properties' tab is active, displaying the following environment variable:

env	=	dev
-----	---	-----

34. Click Application Data in the left-side menu; you should see your watermark variable.

The screenshot shows the Anypoint Platform interface. The left sidebar has a 'DEVELOPMENT' tab and links for Applications, Dashboard, Insight, Logs, Alerts, Application Data (which is selected), and Queues. The main area shows a list titled 'apessentials37' with two items: 'Key' and 'lastAccountId'. There is a 'Delete' button and a search bar.

35. Click Schedules in the left-side menu; you should see the application poll and its current schedule.

36. Click the Disable button.

The screenshot shows the Anypoint Platform interface. The left sidebar has a 'DEVELOPMENT' tab and links for Applications, Dashboard, Insight, Logs, Alerts, Application Data, and Schedules (which is selected). The main area shows a table of scheduled polls. One row is highlighted with a green background, showing a poll named 'accountsBatch Poll' with a 'Last Run' time of '07/17 05:08:38 PM' and a 'Schedule' of 'Every 30 seconds'. There are buttons for 'Run now', 'Enable', and 'Disable'.

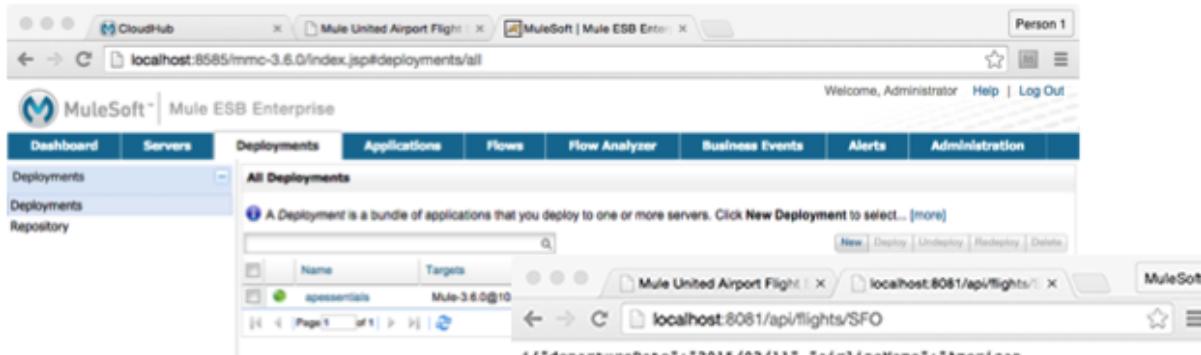
Name	Last Run	Schedule
accountsBatch Poll	07/17 05:08:38 PM	Every 30 seconds

## Walkthrough 11-4: (Optional) Deploy an application on-prem

In this lab, you will deploy your application to a local, standalone Mule runtime. You will:

- Package an application as a Mule deployable archive.
- Start a local, standalone Mule runtime and Mule Management console (MMC).
- Deploy an application to the Mule runtime.
- Run the application.

*Note: To complete this walkthrough, you need a standalone Mule runtime with Mule Management Console (MMC). If you have not already downloaded this bundle, refer to the setup instructions to get it.*



The screenshot shows the MuleSoft Mule ESB Enterprise Management Console interface. The top navigation bar includes CloudHub, Mule United Airport Flight, and MuleSoft | Mule ESB Enter. The main menu has tabs for Dashboard, Servers, Deployments, Applications, Flows, Flow Analyzer, Business Events, Alerts, and Administration. The Deployments tab is selected, showing a sub-menu with Deployments, Deployments, and Repository. Below this is a section titled "All Deployments" with a note: "A Deployment is a bundle of applications that you deploy to one or more servers. Click New Deployment to select... [more]". A table lists a single deployment entry: "apessentials" (Status: Mule-3.6.0@10). At the bottom of the page, there is a JSON snippet of flight data:

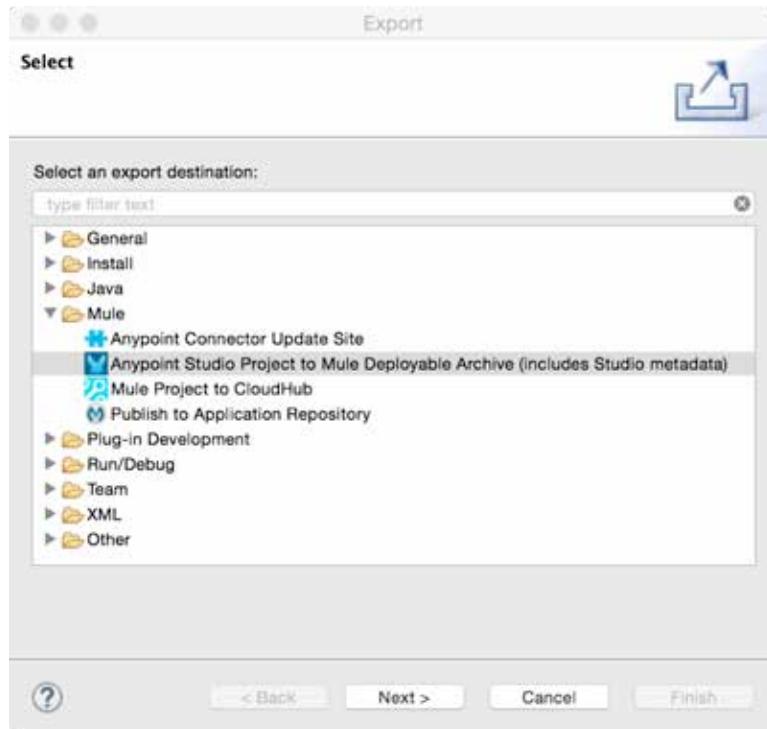
```
[[{"departureDate": "2015/02/11", "airlineName": "American Airlines", "destination": "SFO", "price": 142.0, "planeType": "Boeing 737", "code": "free1093", "origin": "MUA", "emptySeats": 0}, {"departureDate": "2015/02/12", "airlineName": "Delta", "destination": "SFO", "price": 294.0, "planeType": "Boeing 787", "code": "Al4244", "origin": "MUA", "emptySeats": 10}, {"departureDate": "2015/02/20", "airlineName": "American Airlines", "destination": "SFO", "price": 300.0, "planeType": "Boeing 737", "code": "free2000", "origin": "MUA", "emptySeats": 0}, {"departureDate": "2015/03/26", "airlineName": "United", "destination": "SFO", "price": 400.0, "planeType": "Boeing 737", "code": "ERJ8sd", "origin": "MUA", "emptySeats": 0}, {"departureDate": "2015/03/20", "airlineName": "Delta", "destination": "SFO", "price": 400.0, "planeType": "Boeing 737", "code": "AlB2CJ", "origin": "MUA", "emptySeats": 40}]]
```

### Package the application

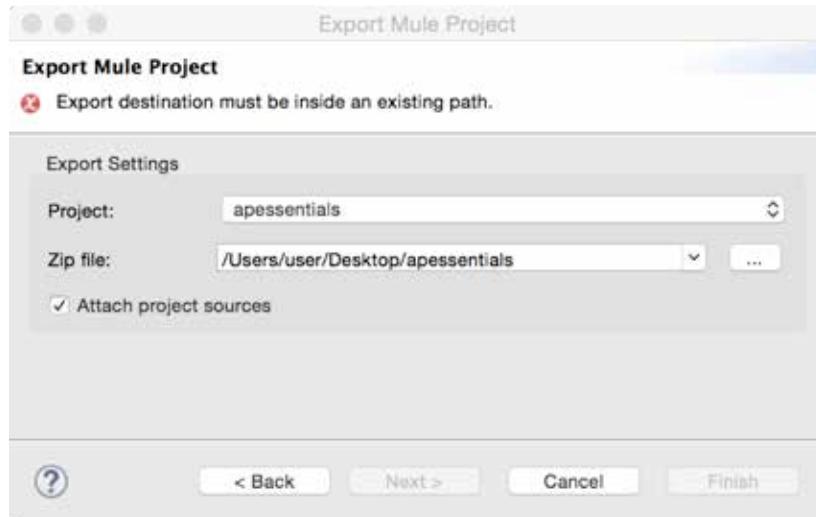
1. Return to Anypoint Studio and run the application to make sure it is working.
2. Click the red Terminate button to stop the embedded Mule runtime; YOU MUST DO THIS.
3. Select File > Export.

4. In the Export dialog box, select Mule > Anypoint Studio Project to Mule Deployable Archive and click Next.

*Note: You can also right-click a project in the Package Explorer and select Export > Anypoint Studio Project to Mule Deployable Archive.*

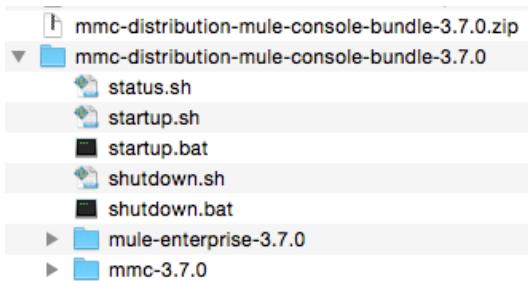


5. In the Export Mule Project dialog box, set the project to apessentials.
6. Browse to a location (like your desktop) and save the file as apessentials.



## Set the environment property in the Mule wrapper.conf file

7. Locate the mmc-distribution-mule-console-bundle-{version}.zip you downloaded for class as instructed in the setup instructions.
8. Unzip the file into some directory.



9. Navigate to the mmc-distribution-mule-console-bundle-3.7.X/mule-enterprise-3.7.X/conf folder.
10. Open wrapper.conf in a text editor.
11. Locate the last numbered wrapper.java.additional instruction near the top of the file.
12. Beneath it, create a new instruction with the number incremented by one that passes an argument called env equal to dev to the runtime when it starts.

wrapper.java.additional.15=-Denv=dev

*Note: Be sure to use the correct number. This number may be different than shown here for your version of the runtime.*

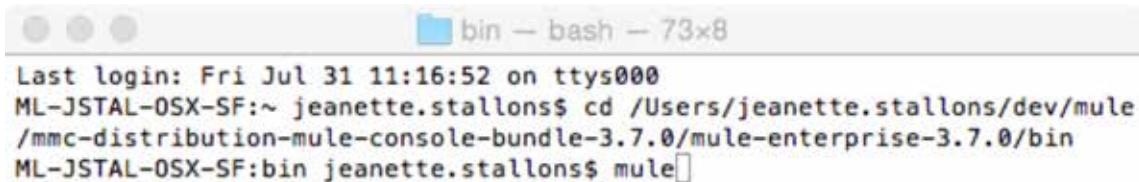
```
File Path : ~/dev/mule/mmc-distribution-mule-console-bundle-3.7.0/mule-enterprise-3.7.0/conf/wrapper.conf
wrapper.conf
1 #encoding=UTF-8
2 ****
3 # System Properties
4 ****
5 # Location of your Mule installation.
6 wrapper.java.additional.1=-Dmule.home="%MULE_HOME%"
7 wrapper.java.additional.1.stripquotes=TRUE
8 wrapper.java.additional.2=-Dmule.base="%MULE_HOME%"
9 wrapper.java.additional.2.stripquotes=TRUE
10
11 # Sets IPv4 addresses in order to avoid multicasting issues
12 wrapper.java.additional.3=-Djava.net.preferIPv4Stack=TRUE
13
14 # Needed to avoid a memory leak on mvn (see MULE-7658)
15 wrapper.java.additional.4=-Dmvel2.disable.jit=TRUE
16
17 # Limit HTTP module send and receive buffers size to 1MB by default to avoid running out
18 # of memory. This system property should be removed and direct memory increased as required
19 wrapper.java.additional.5=-Dorg.glassfish.grizzly.nio.transport.TCPNIOTransport.max-rec
20 wrapper.java.additional.6=-Dorg.glassfish.grizzly.nio.transport.TCPNIOTransport.max-sem
21
22 # Increase Permanent Generation Size from default of 64M
23 # Increase this value if you get "Java.lang.OutOfMemoryError: PermGen space error"
24 # This property is not used when running java 8 and may cause a warning.
25 wrapper.java.additional.7=-XX:PermSize=256M
26 wrapper.java.additional.8=-XX:MaxPermSize=256M
27
28 # GC settings
29 wrapper.java.additional.9=-XX:+HeapDumpOnOutOfMemoryError
30 wrapper.java.additional.10=-XX:+AlwaysPreTouch
31 wrapper.java.additional.11=-XX:+UseParNewGC
32 wrapper.java.additional.12=-XX:NewSize=512M
33 wrapper.java.additional.13=-XX:MaxNewSize=512M
34 wrapper.java.additional.14=-XX:MaxTenuringThreshold=8
35
36 wrapper.java.additional.15=-Denv=dev
37
38 # *** IMPORTANT ***
39 # If you enable any of the options below, you _must_ change the <n> to be a
40 # consecutive number (based on the number of additional properties) otherwise
41 # Java will not parse this properties file correctly!
42 # (see http://wrapper.tanukisoftware.org/doc/english/prop-java-additional-n.html)
43 # *** IMPORTANT ***
```



13. Save the file.

## Start the Mule runtime

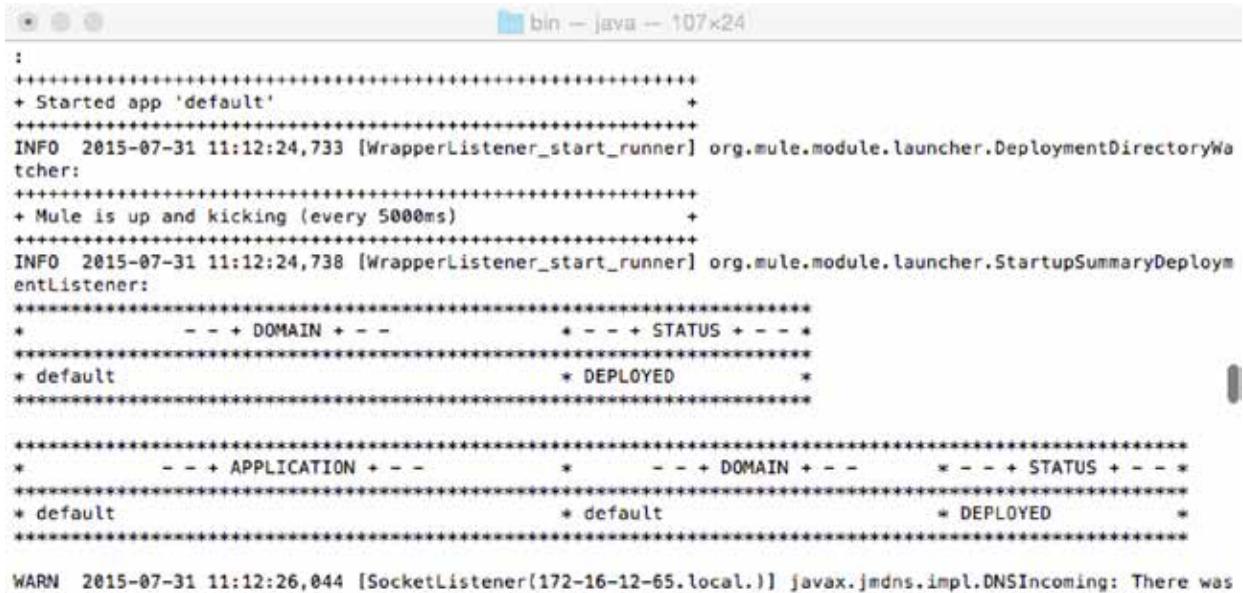
14. On your computer, open a Terminal or Command window.
15. Change to the /mmc-distribution-mule-console-bundle-3.7.X/mule-enterprise-3.7.X/bin directory.
16. Run the mule script: mule (Unix) or mule.bat (Windows).



```
Last login: Fri Jul 31 11:16:52 on ttys000
ML-JSTAL-OSX-SF:~ jeanette.stallons$ cd /Users/jeanette.stallons/dev/mule
/mmc-distribution-mule-console-bundle-3.7.0/mule-enterprise-3.7.0/bin
ML-JSTAL-OSX-SF:bin jeanette.stallons$ mule
```

17. Wait until you see a message that Mule is up and kicking.

*Note: You may need to scroll up to see this if you get a lot of DNS warnings after it.*



```
:
+-----+
+ Started app 'default' +
+-----+
INFO 2015-07-31 11:12:24,733 [WrapperListener_start_runner] org.mule.module.launcher.DeploymentDirectoryWatcher:
+-----+
+ Mule is up and kicking (every 5000ms) +
+-----+
INFO 2015-07-31 11:12:24,738 [WrapperListener_start_runner] org.mule.module.launcher.StartupSummaryDeploymentListener:
+-----+
*      - - + DOMAIN - - *      - - + STATUS - - *
*-----*-----*-----*
* default                                * DEPLOYED   *
*-----*-----*-----*
*-----*-----*-----*
*      - - + APPLICATION - - *      - - + DOMAIN - - *      - - + STATUS - - *
*-----*-----*-----*
* default                                * default     * DEPLOYED   *
*-----*-----*-----*
WARN 2015-07-31 11:12:26,844 [SocketListener(172-16-12-65.local.)] javax.jmdns.impl.DNSIncoming: There was
```

## Start MMC

18. On your computer, open a second Terminal or Command window.
19. Change to the /mmc-distribution-mule-console-bundle-3.7.X/mmc-3.7.X/apache-tomcat-7.0.52/bin directory.
20. Run the startup script: ./startup.sh (Unix) or startup.bat (Windows).

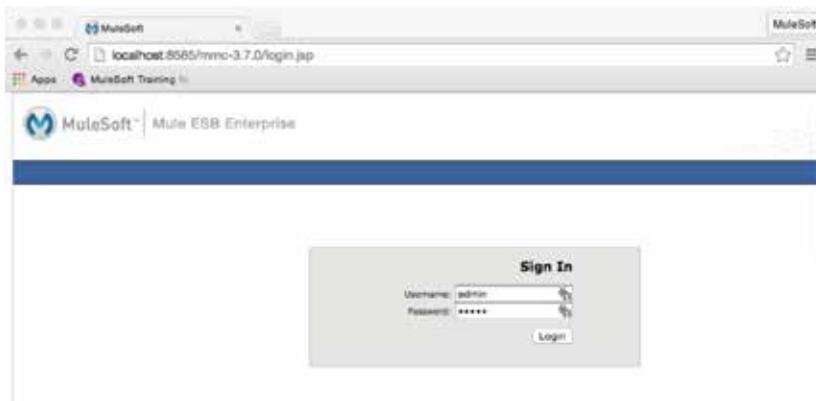
21. Wait until you see a message that Tomcat started in your console.

```
Last login: Fri Jul 31 11:21:07 on ttys001  
ML-JSTAL-OSX-5F:~ jeanette.stallons$ cd /Users/jeanette.stallons/dev/mule/mmc-distribution-mule-console-bundle-3.7.0/mmc-3.7.0/apache-tomcat-7.0.52/bin  
ML-JSTAL-OSX-5F:bin jeanette.stallons$ ./startup.sh  
Using CATALINA_BASE: /Users/jeanette.stallons/dev/mule/mmc-distribution-mule-console-bundle-3.7.0/mmc-3.7.0/apache-tomcat-7.0.52  
Using CATALINA_HOME: /Users/jeanette.stallons/dev/mule/mmc-distribution-mule-console-bundle-3.7.0/mmc-3.7.0/apache-tomcat-7.0.52  
Using CATALINA_TMPDIR: /Users/jeanette.stallons/dev/mule/mmc-distribution-mule-console-bundle-3.7.0/mmc-3.7.0/apache-tomcat-7.0.52/temp  
Using JRE_HOME: /Library/Java/JavaVirtualMachines/jdk1.8.0_45.jdk/Contents/Home  
Using CLASSPATH: /Users/jeanette.stallons/dev/mule/mmc-distribution-mule-console-bundle-3.7.0/mmc-3.7.0/apache-tomcat-7.0.52/bin/bootstrap.jar;/Users/jeanette.stallons/dev/mule/mmc-distribution-mule-console-bundle-3.7.0/mmc-3.7.0/apache-tomcat-7.0.52/temp/Tomcat started.  
ML-JSTAL-OSX-5F:bin jeanette.stallons$
```

22. In a browser window, navigate to <http://localhost:8585/mmc-3.7.0>.

*Note: Change the URL as necessary for a different version of the runtime.*

23. Enter a username and password of admin and click Login.



## Register the Mule runtime in MMC

24. In MMC, click the Servers tab.

25. Look at the All group and see if it has a number of 1 and a server registered.

Name	Version	Location	Flows	Type
Mule-3.7.0@192.168.1.126:7777	3.7.0	172.16.12.65	1/-/-	Single Server

26. If All is 0 and there is an Unregistered server, select it in the list, and click the Register button.

*Note: If the runtime does not register, follow the instructions in the next step.*

The screenshot shows the MuleSoft Enterprise MMC interface. The title bar says "MuleSoft | Mule ESB Enterprise". The URL in the address bar is "localhost:8585/mmc-3.7.0/index.jsp#discover". The left sidebar has tabs for "Dashboard", "Servers", "Deployments", "Applications", "Flows", "Flow Analyzer", "Business Events", "Alerts", and "Ad". The "Servers" tab is selected. A sub-menu under "Servers" shows groups: "All (0)", "Development (0)", "Production (0)", "Staging (0)", "Test (0)", and "Unregistered (1)". The "Unregistered (1)" item is highlighted with a blue selection bar. The main content area is titled "Unregistered Servers" and lists one entry: "Mule-3.7.0@192.168.1.125.7777" with version "3.7.0", agent version "\${project.version}", and host "192.168.1.125". There are buttons for "Register", "Register & Add to groups", and "New Server".

27. If All is 0 and Unregistered is 0, add and register the server as follows:

- Click the Servers tab.
- Click the Add drop-down menu and select New Server.
- Enter a server name of Max.
- Leave the Mule Agent URL as <http://localhost:7777/mmc-support>
- Click Add.

The screenshot shows the MuleSoft Enterprise MMC interface. The title bar says "MuleSoft | Mule ESB Enterprise". The URL in the address bar is "localhost:8585/mmc-3.7.0/index.jsp#servers/new". The left sidebar has tabs for "Dashboard", "Servers", "Deployments", "Applications", "Flows", "Flow Analyzer", and "Business Events". The "Servers" tab is selected. A sub-menu under "Servers" shows groups: "All (0)", "Development (0)", "Production (0)", "Staging (0)", "Test (0)", and "Unregistered (0)". The "Add Server" dialog is open. It has a note: "Enter a unique name for this server and the URL of the Mule agent using the format http://[hostname]:7777/mmc-support, where hostname is the host where Mule is running". It has fields for "Server Name" (set to "Max") and "Mule Agent URL" (set to "http://localhost:7777/mmc-support"). There are "Add" and "Cancel" buttons at the bottom.

## Deploy your application to the Mule runtime

28. Click the Deployments tab.
29. Click the New button.
30. Enter a deployment name of apessentials.
31. Select the server you just registered, Mule-3.7.X or Max.
32. Click the Upload New Application button.
33. Click the Browse button, browse to the zip file you created earlier, and click Open.
34. Click the Add button; you should see your archive listed.

The screenshot shows the MuleSoft ESB Enterprise interface. The top navigation bar includes links for Apps, MuleSoft Training, Welcome, Administrator, Help, and Log Out. The main menu has tabs for Dashboard, Servers, Deployments (which is selected), Applications, Flows, Flow Analyzer, Business Events, and Alerts. On the left, there's a sidebar with sections for Deployments, Deployments, and Repository. The central area is titled 'Deployment Name' with a placeholder 'Enter a unique name for this Deployment. To add a server, or server group to the Deployment, start typing its name... [more]' and a text input field containing 'apessentials'. Below this, there are two tabs: 'Applications' (selected) and 'Servers'. Under 'Applications', there are buttons for 'Add From Repository' and 'Upload New Application', and a table with one row: 'apessentials' (Name) and '201507311139' (Version). Under 'Servers', there is a dropdown menu labeled 'Select a server...' and a table with one row: 'Mule-3.7.0@192.168.1.125:7777'. At the bottom right are 'Cancel', 'Save', and 'Deploy' buttons. A footer at the bottom of the page includes links for Provide Feedback, Support, About Mule ESB Enterprise, and © MuleSoft, Inc. All rights reserved.

35. Click the Deploy button in the lower-right corner; you should see your application appear in the deployment list with a green circle next to it.

The screenshot shows the MuleSoft ESB Enterprise Management Console (MMC) interface. The title bar reads "MuleSoft | Mule ESB Enterprise". The address bar shows "localhost:8585/mmc-3.7.0/index.jsp#deployments/all". The left sidebar has "Deployments" selected. The main content area is titled "All Deployments" with a sub-instruction: "A Deployment is a bundle of applications that you deploy to one or more servers. Click New Deployment to select... [more]". Below this is a table with columns: Name, Targets, Applications, and Last Modified. A single row is visible: "apessentials" with "Targets: Mule-3.7.0@192.168.56.1" and "Applications: apessentials [2015073111139]" and "Last Modified: Fri Jul 31 11:40:52 GMT-700 2015". At the bottom right of the table, it says "Displaying 1 - 1 of 1".

*Note: If there is a red circle next to the application name, it was not deployed. Examine the log files to figure out why it failed. Return to the mmc-distribution-mule-console-bundle-3.7.X/mule-enterprise-3.7.X folder and open the logs folder. Open the mule-app-apessentials-{version}.log file and locate the error. Fix the errors, create a new deployable archive if necessary, and then redeploy the app in MMC.*

## Test the application

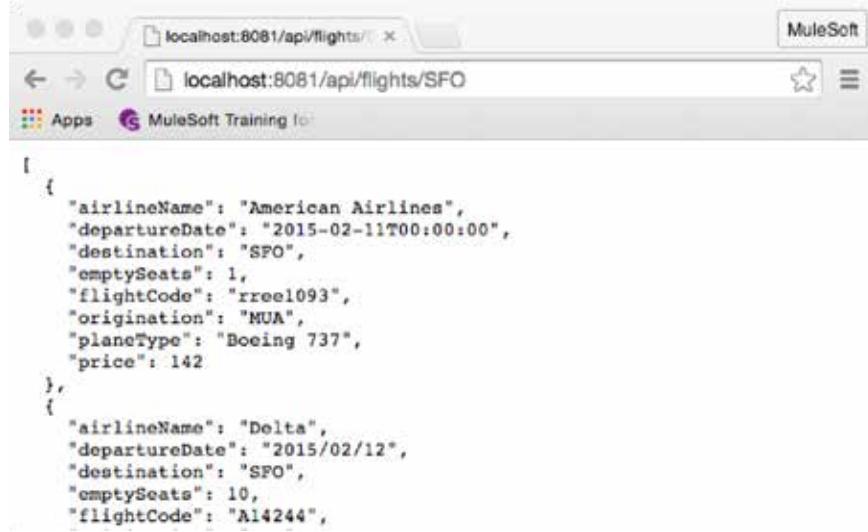
36. Make a request to <http://localhost:8081/flights> and find flights; your application should work as before but now it is running on your local standalone Mule runtime.

The screenshot shows a web browser window with the title "Mule United Airport Flight". The address bar shows "localhost:8081/flights". The page header includes "Person 1" and a navigation menu. The main content area is titled "Mule United Airport" with dropdown menus for "SFO - San Francisco" and "United" and a "Find Flights" button. Below this is a section titled "Available Flights" containing two flight entries:

Flight Code	Airline Name	Destination	Plane Type	Price
ER38sd	United	SFO	Boeing 737	\$400.00
ER39rk	United	SFO	Boeing 757	\$945.00



37. Make a request to <http://localhost:8082/api/flights/SFO>; you should get flight results.



A screenshot of a web browser window titled "localhost:8081/api/flights/SFO". The address bar shows the URL. The page content displays a JSON array of flight information. The first flight is from American Airlines, departing on 2015-02-11 at 00:00:00, destination SFO, price \$142. The second flight is from Delta, departing on 2015/02/12, destination SFO, price \$14244.

```
[{"airlineName": "American Airlines", "departureDate": "2015-02-11T00:00:00", "destination": "SFO", "emptySeats": 1, "flightCode": "rree1093", "origination": "MUA", "planeType": "Boeing 737", "price": 142}, {"airlineName": "Delta", "departureDate": "2015/02/12", "destination": "SFO", "emptySeats": 10, "flightCode": "A14244", "origination": "MUA", "planeType": "Boeing 737", "price": 14244}]
```

## Stop MMC and the Mule runtime

38. Return to the MMC Terminal/Command window.
39. Run the shutdown script: ./shutdown.sh (Unix) or shutdown.bat (Windows).
40. Close the Terminal/Command window.
41. Return to the Mule Terminal/Command window.
42. Stop the server by pressing Ctrl+C.
43. Wait until the runtime stops.



A screenshot of a terminal window titled "bin - bash - 106x18". The window shows the Mule runtime stopping process. It includes log entries from org.mule.module.launcher.domain.DefaultMuleDomain, org.mule.lifecycle.AbstractLifecycleManager, and org.mule.DefaultMuleContext. The output indicates the application "agent" shut down normally on 7/31/15 at 11:46 AM, after being up for 0 days, 0 hours, 11 mins, 20.271 sec.

```
INFO 2015-07-31 11:46:22,645 [WrapperListener_stop_runner] org.mule.module.launcher.domain.DefaultMuleDomain: 
+-----+
+ Disposing domain 'default' +
+-----+
INFO 2015-07-31 11:46:22,646 [WrapperListener_stop_runner] org.mule.lifecycle.AbstractLifecycleManager: Disposing: 'Mule Agent Core Extension'. Object is: AgentCoreExtension
INFO 2015-07-31 11:46:22,646 [WrapperListener_stop_runner] org.mule.lifecycle.AbstractLifecycleManager: Disposing RegistryBroker
INFO 2015-07-31 11:46:22,646 [WrapperListener_stop_runner] org.mule.lifecycle.AbstractLifecycleManager: Disposing model: _muleSystemModel
INFO 2015-07-31 11:46:22,837 [WrapperListener_stop_runner] org.mule.DefaultMuleContext: 
*-----*
* Application "agent" shut down normally on: 7/31/15 11:46 AM      *
* Up for: 0 days, 0 hours, 11 mins, 20.271 sec            *
*-----*
<-- Wrapper Stopped
ML-JSTAL-OSX-SF:bin jeanette.stallons$ []
```

44. Close the Terminal/Command window.