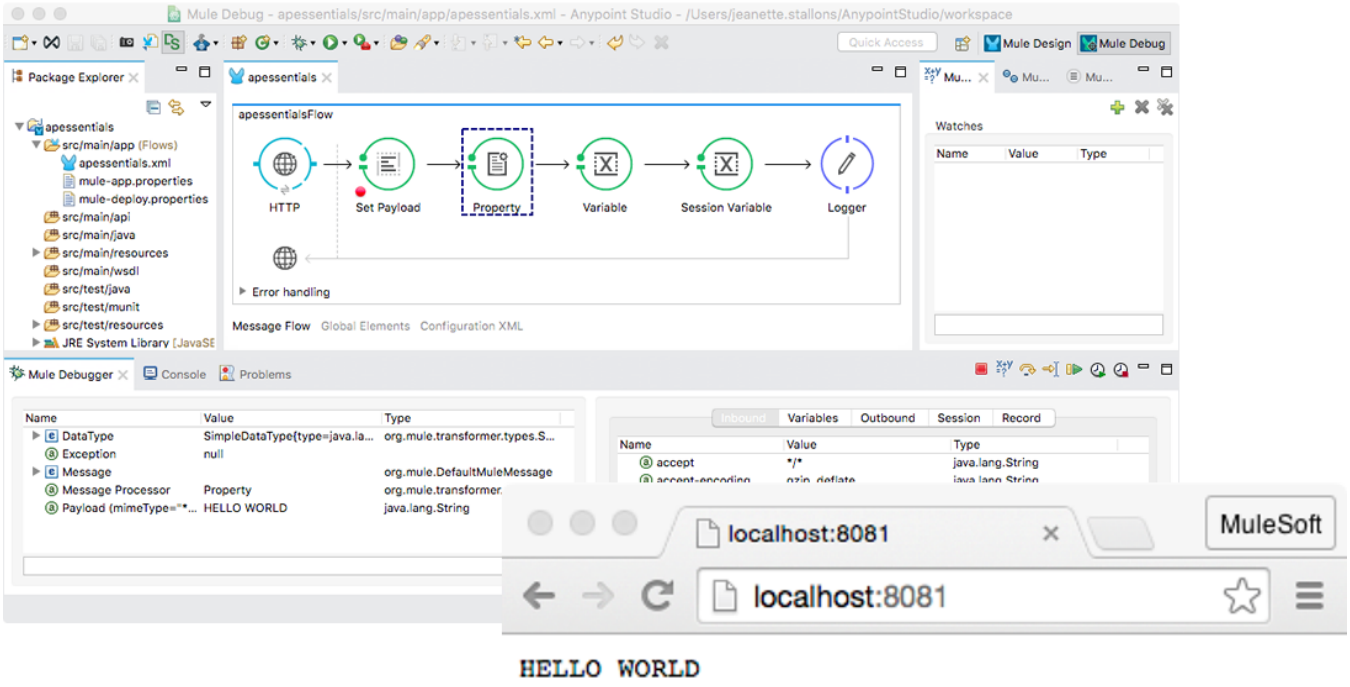
Anypoint Platform Essentials

Student Manual

Mule runtime 3.8
May 20, 2015

Table of Contents

Module 2: Building Integration Applications with Anypoint Studio



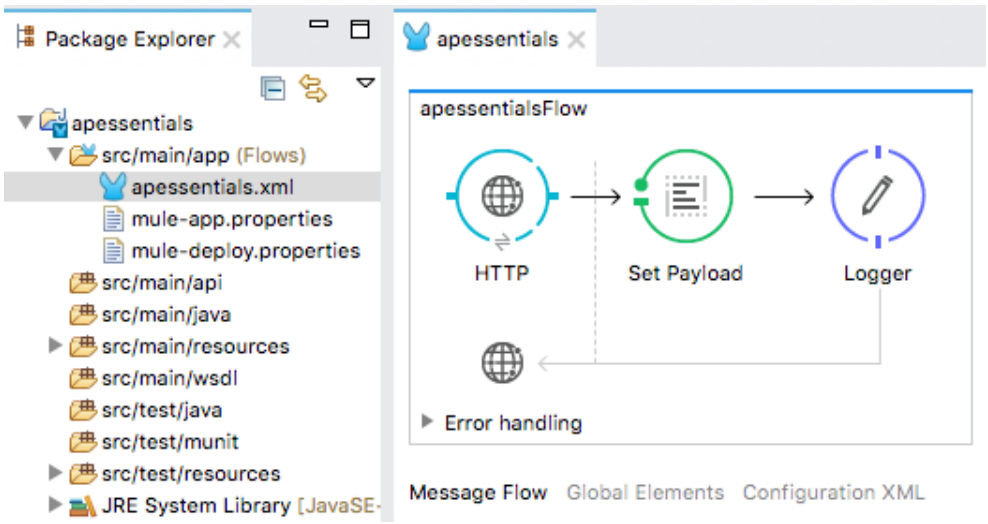
In this module, you will learn:

- 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

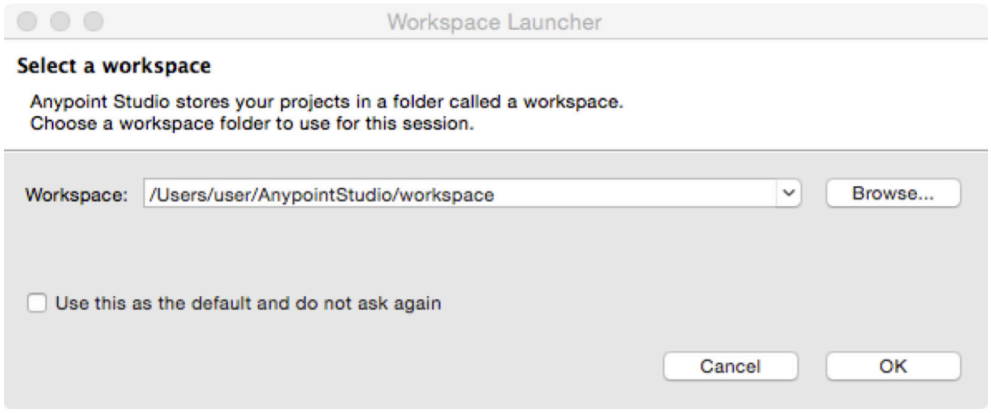
In this walkthrough, you will build your first Mule application. You will:

- 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.

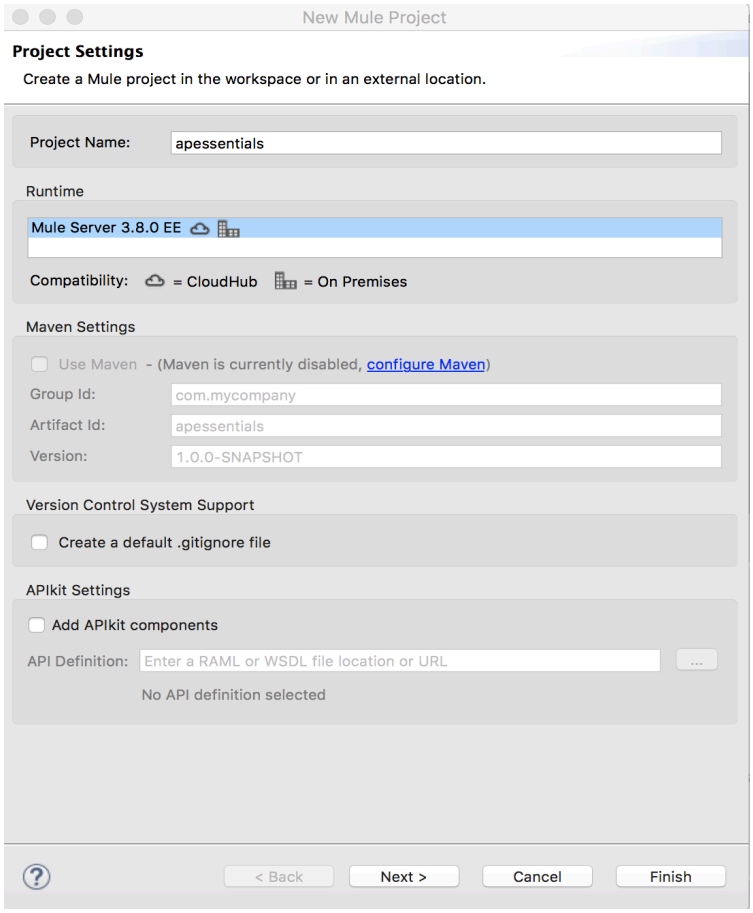


3. Click OK to select the workspace; Anypoint Studio should open.
4. If you get a Welcome Page, click the X on the tab to close it.

5. If you get an Updates Available pop-up in the lower-right corner of the application, click it and install the available updates.

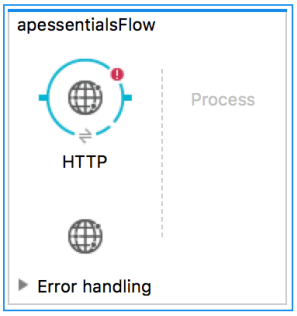
Create a project

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.

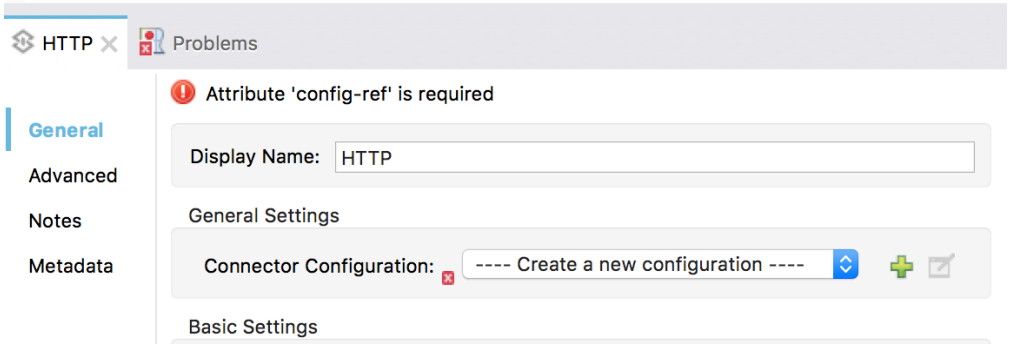


Create an HTTP connector endpoint to receive requests

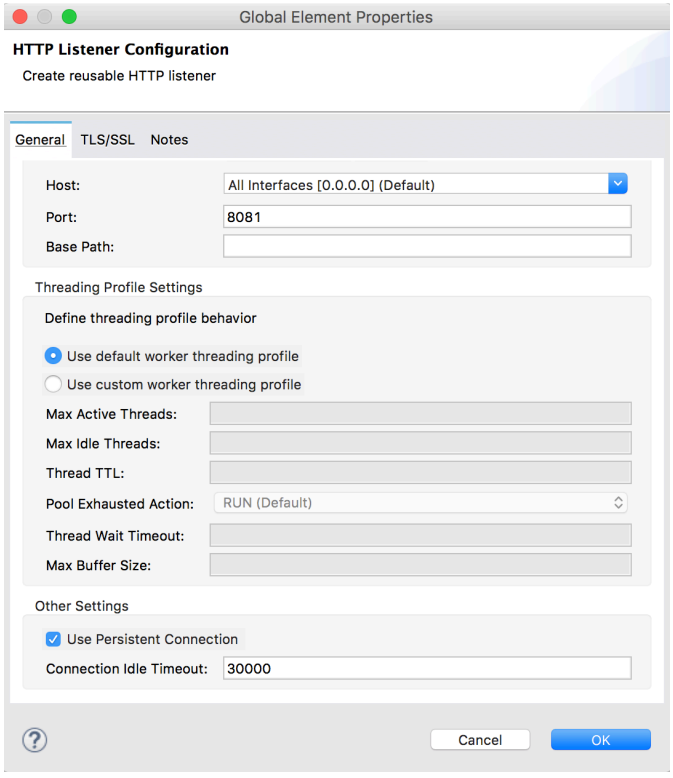
10. Drag an HTTP connector from the palette to the canvas.



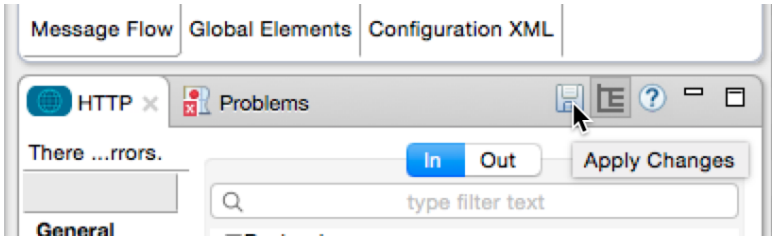
11. Double-click the HTTP connector endpoint.
12. In the Mule Properties view, click the Add button next to connector configuration.



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

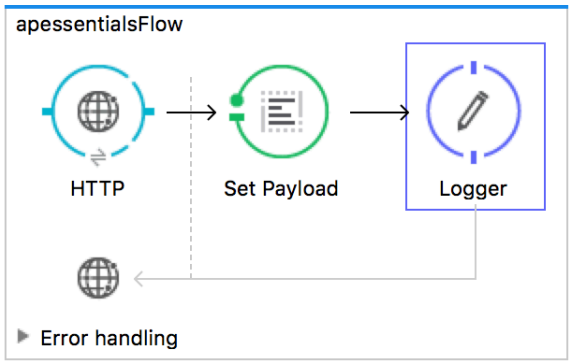


14. Click the Apply Changes button; the errors in the Problems view should disappear.



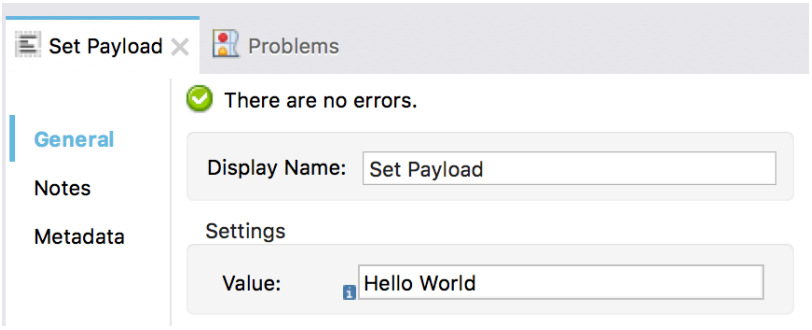
Display data

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.

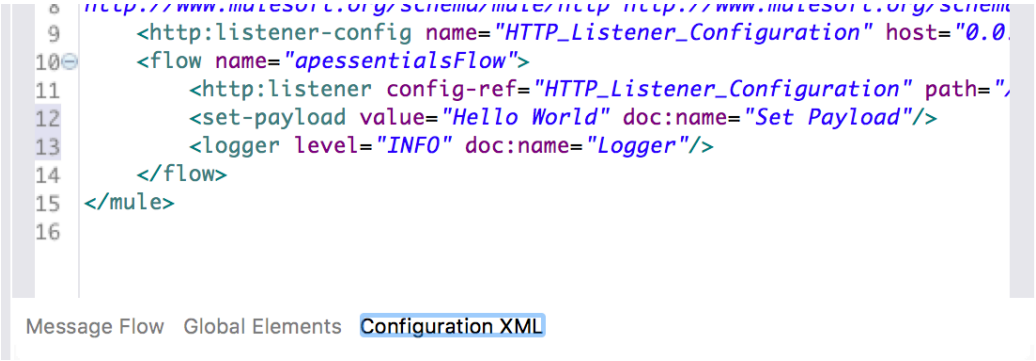


Configure the Set Payload transformer

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.

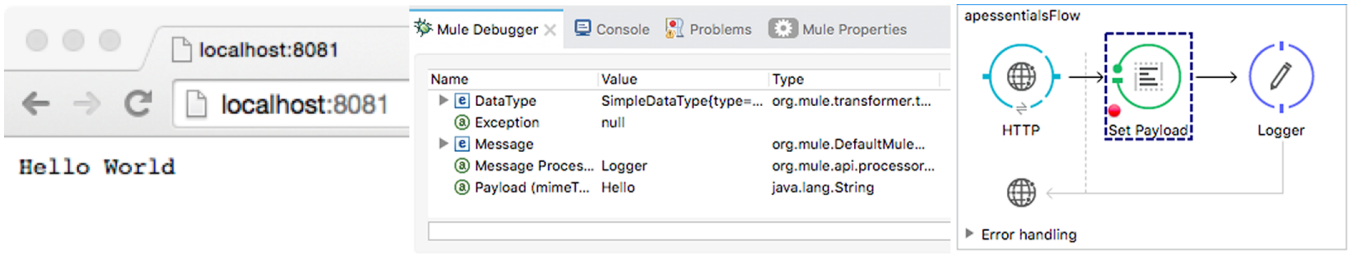


20. Click the Message Flow tab to return to the canvas.
21. Click the Save button or press Ctrl+S to save the file.

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

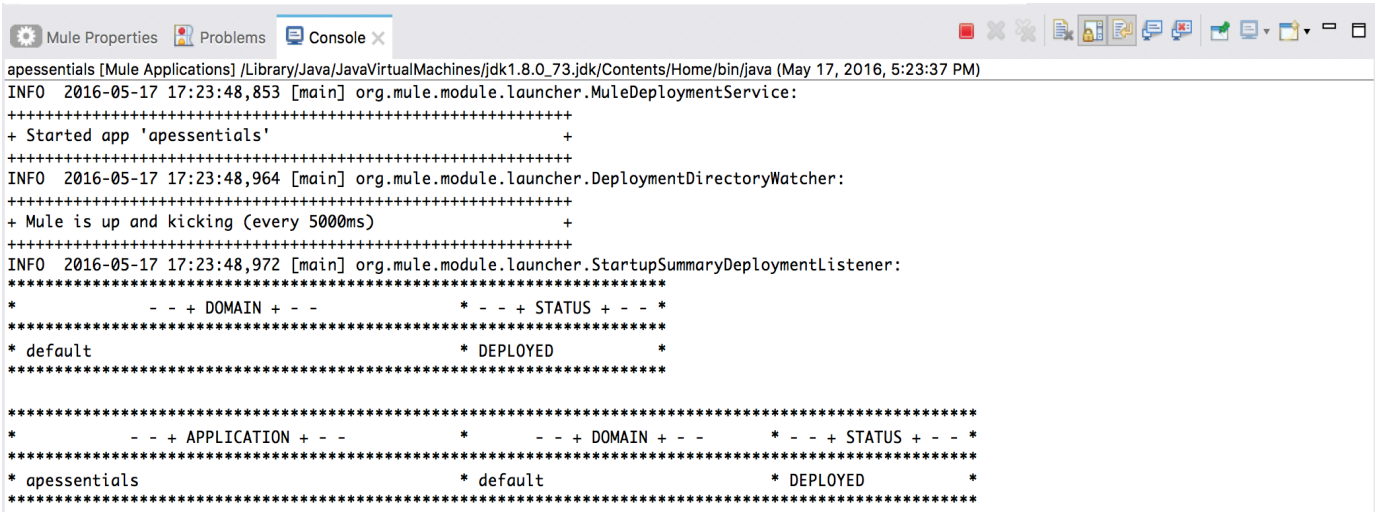
In this walkthrough, you will run, test, and debug your first Mule application. You will:

- 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.



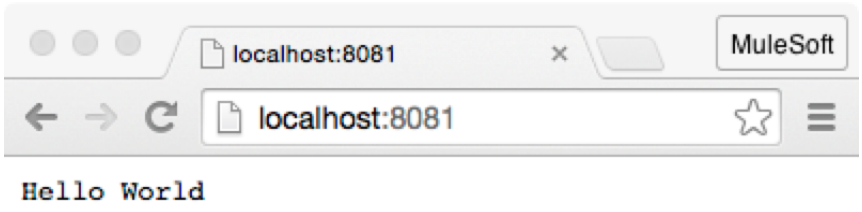
Run the application

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 apessentials application started.

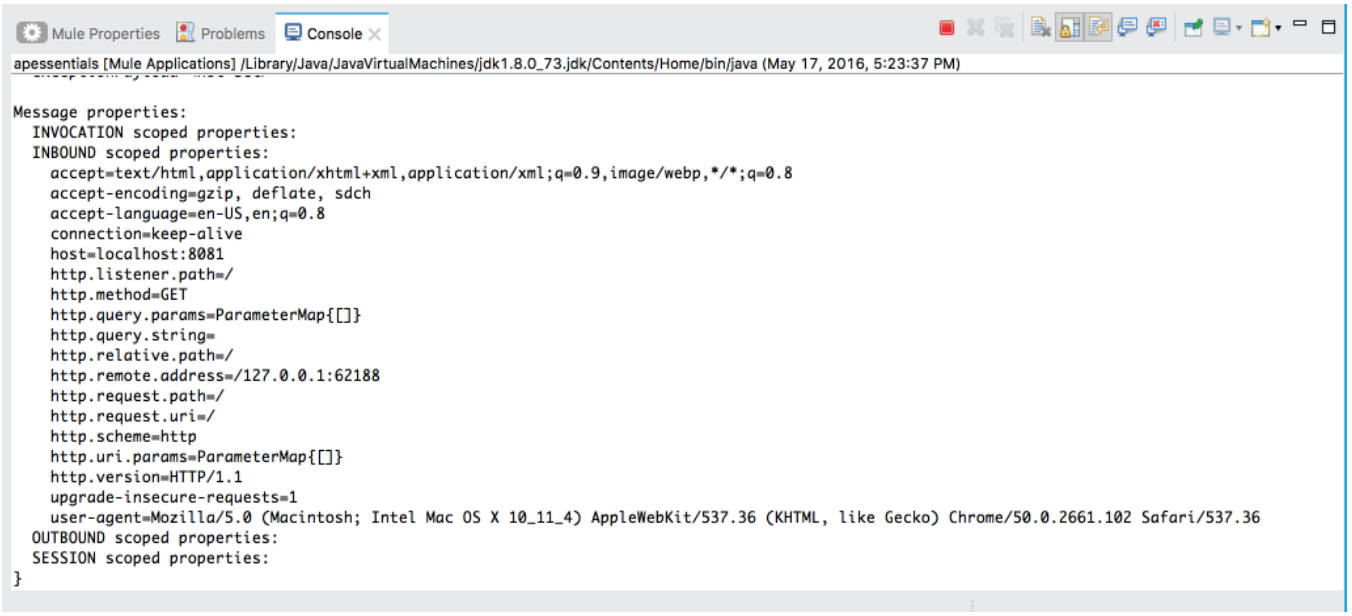


Test the application

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



4. Return to the console in Anypoint Studio.
5. Examine the last entry.

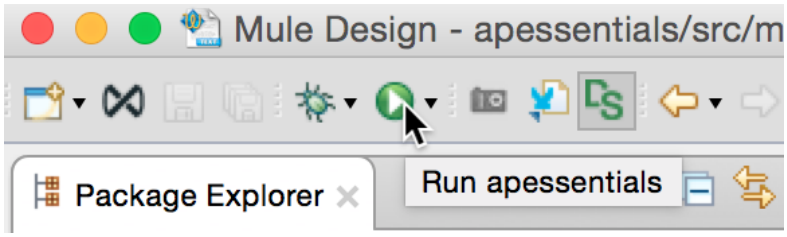


6. Click the red Terminate button to stop the application and the Mule runtime.
7. Answer the following questions.
   - What triggered all the output you saw in the console?
   - What is the last thing displayed?
   - What Java class represents the payload?
   - What are the inbound and outbound properties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

Rerun the application

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

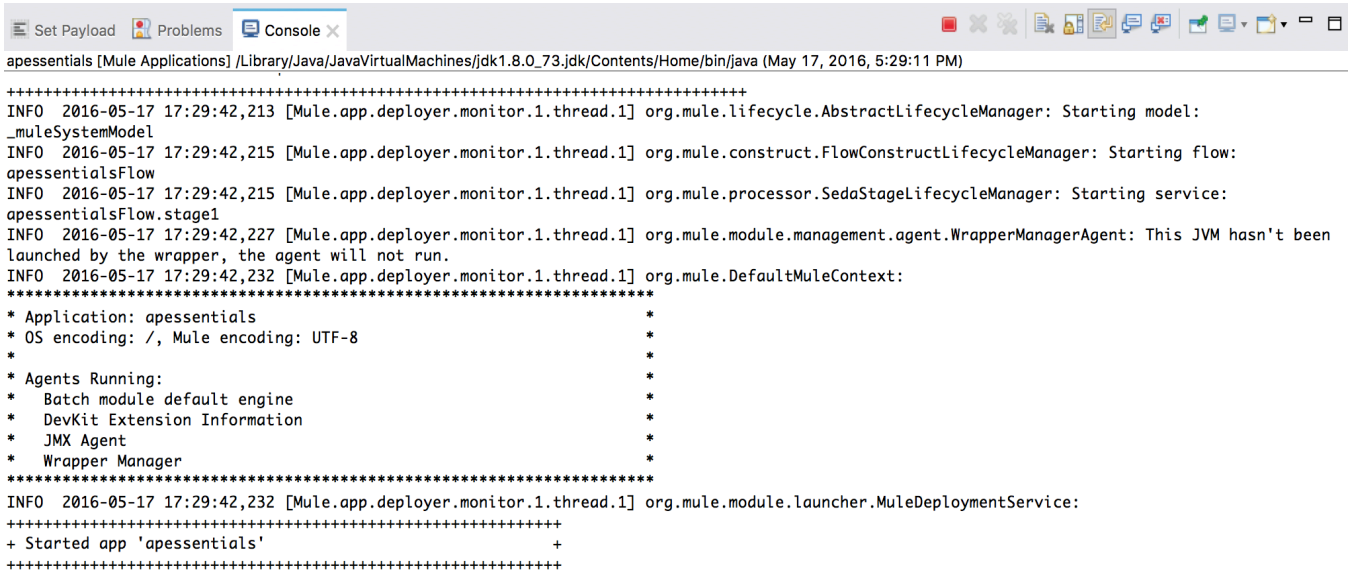
10. Click the Run button and watch the console; you should the application is redeployed but the Mule runtime is also restarted.



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.



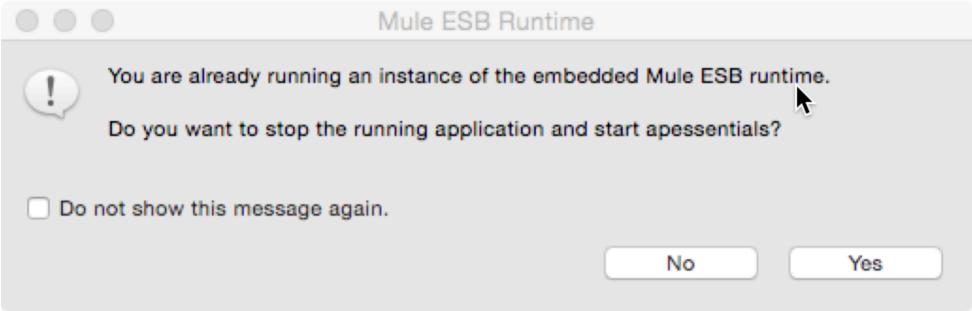
Debug the application

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.

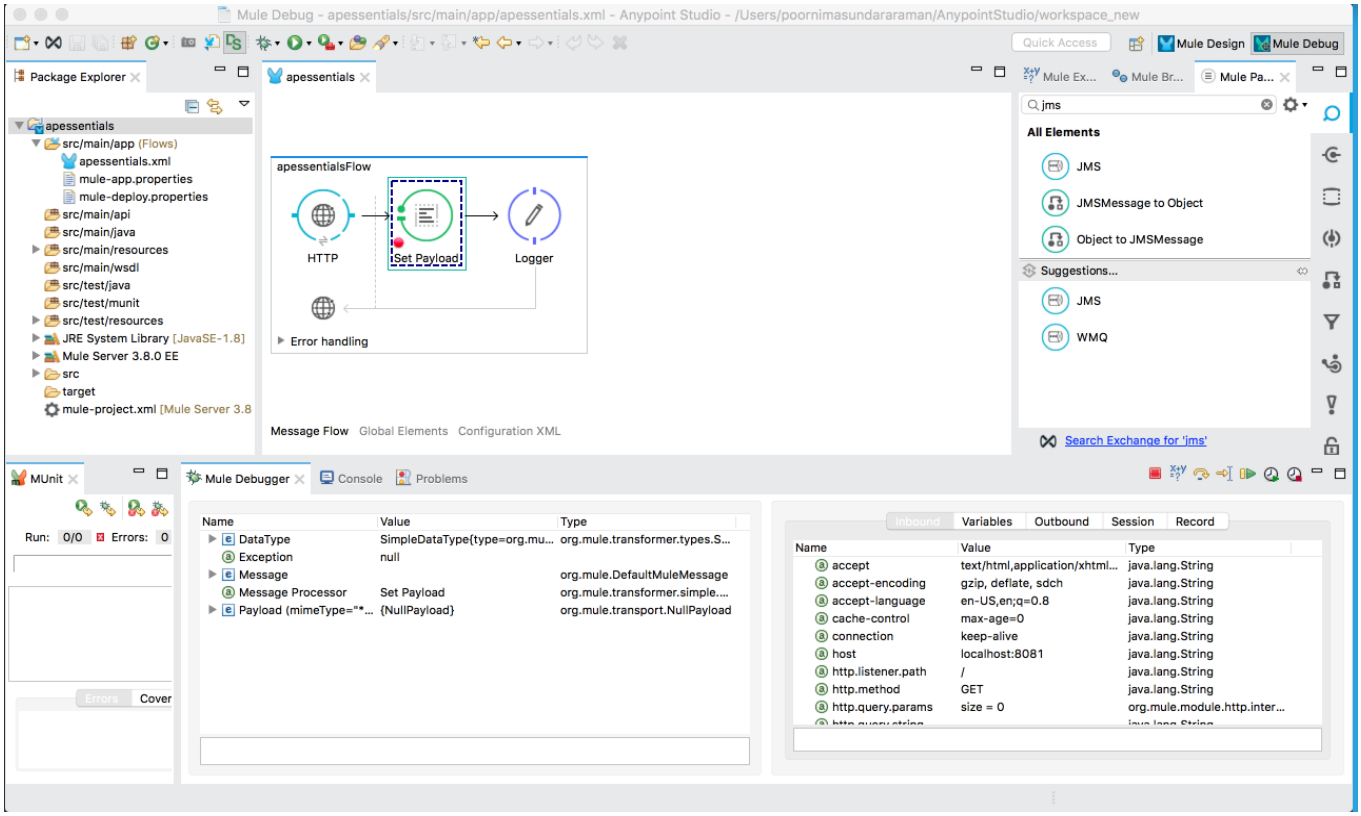
15. If you get an Accept incoming network connections dialog box, click Allow.
16. If you get a Confirm Perspective Switch dialog box, click Yes.

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

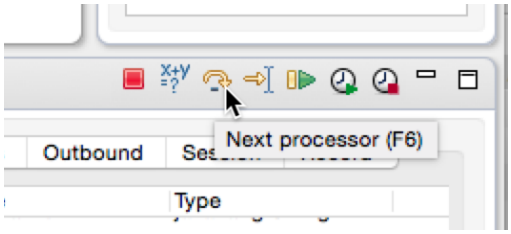
17. In the Mule ESB Runtime window, click Yes to stop the Mule runtime and restart it connected to the Mule Debugger.



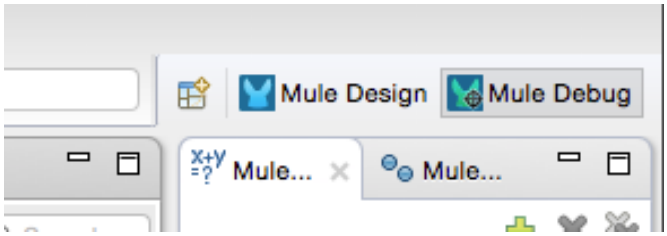
18. Make another request to http://localhost:8081 with a browser or other tool like cURL or Postman.
19. Return to Anypoint Studio and look at the Mule Debugger view.
20. Drill-down and explore the properties and variables.



21. Click the Next processor button.



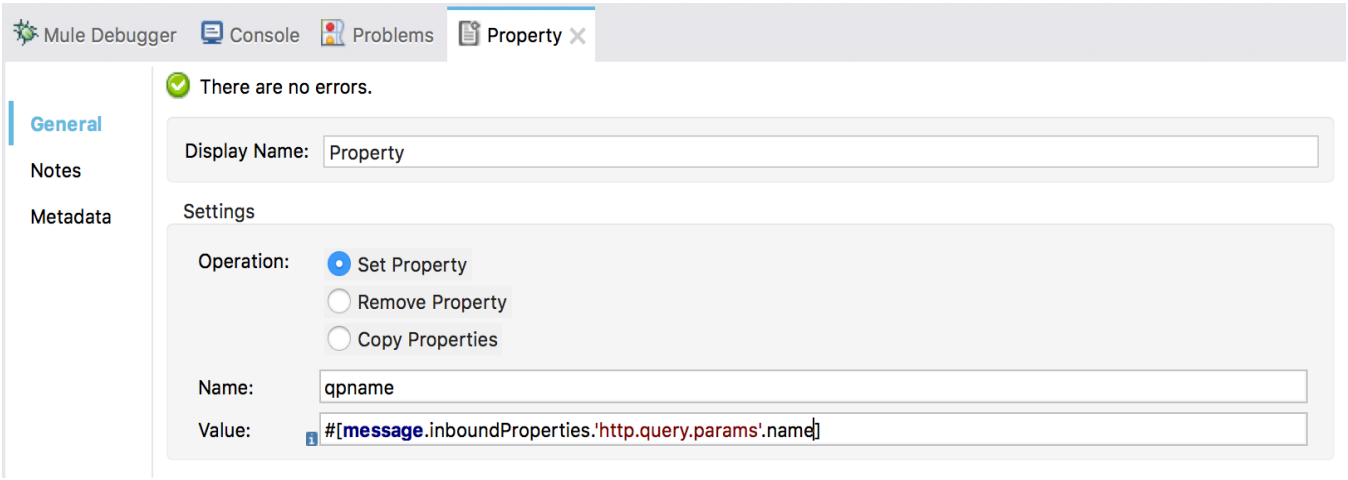
22. Look at the new value of the payload.
23. Step through the rest of the application.
24. Stop the application and the Mule runtime.
25. Click the Mule Design tab in the upper-right corner of the application to switch perspectives.



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.

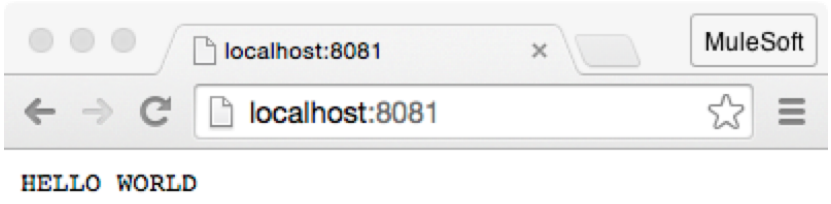
   ```
   #['Hello World']
   ```

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()]
   ```

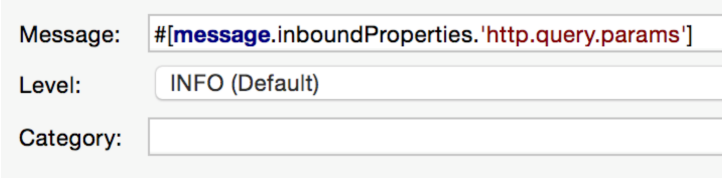
   Note: Press Ctrl+Space to trigger autocomplete if it does not appear.

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.

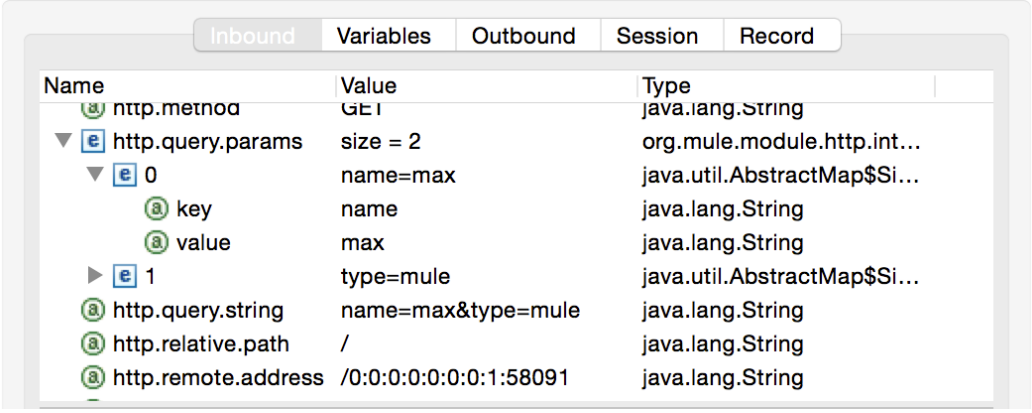


Use an expression to display info to the console

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.



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.



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.process
or.LoggerMessageProcessor: ParameterMap{[name=[max], type=[mule]]}
```

15. Modify the Logger to display one of your query parameters.

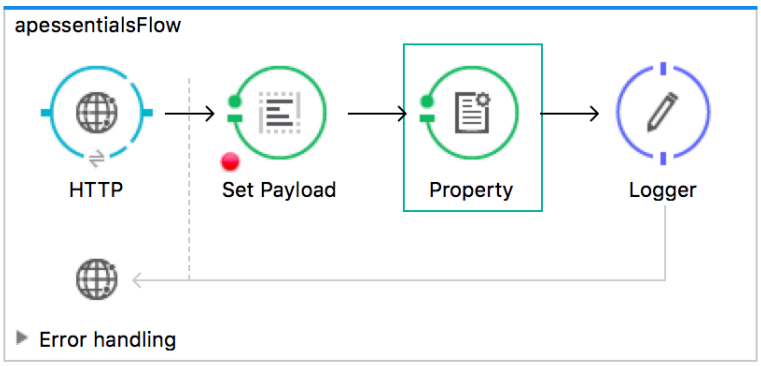
```
#[message.inboundProperties.'http.query.params'.name]
```

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.process
or.LoggerMessageProcessor: max
```

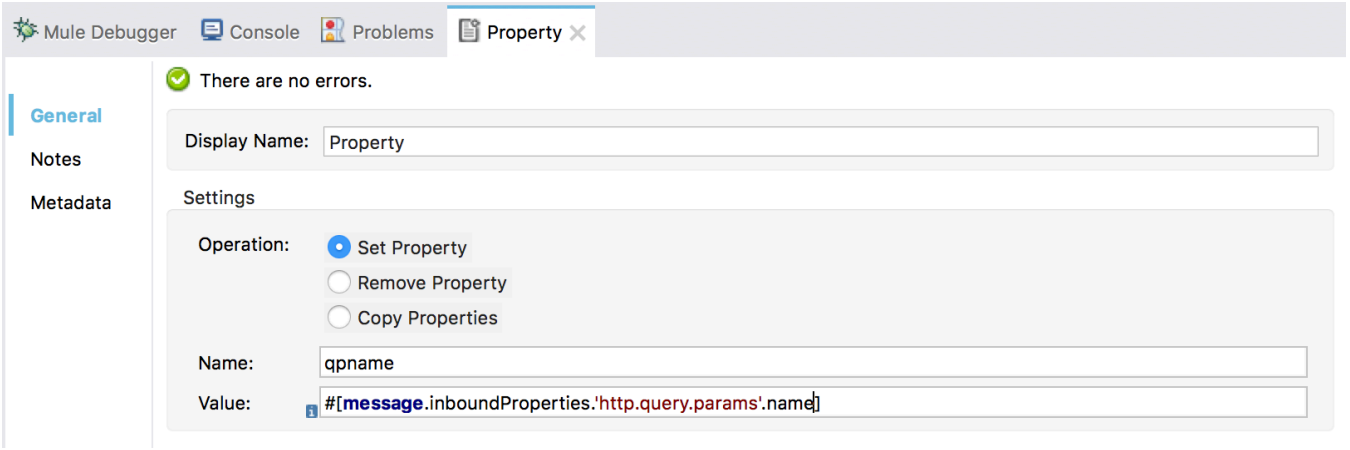
Set an outbound property

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.

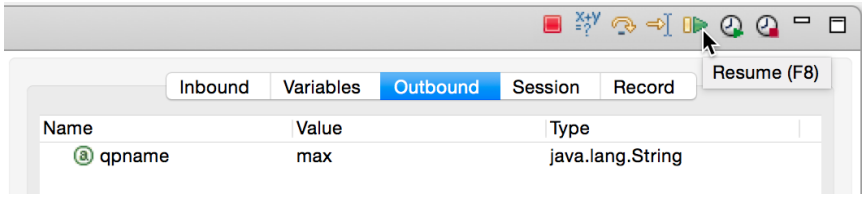


22. Modify the Logger to display the value of this new outbound property.

```
#[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.

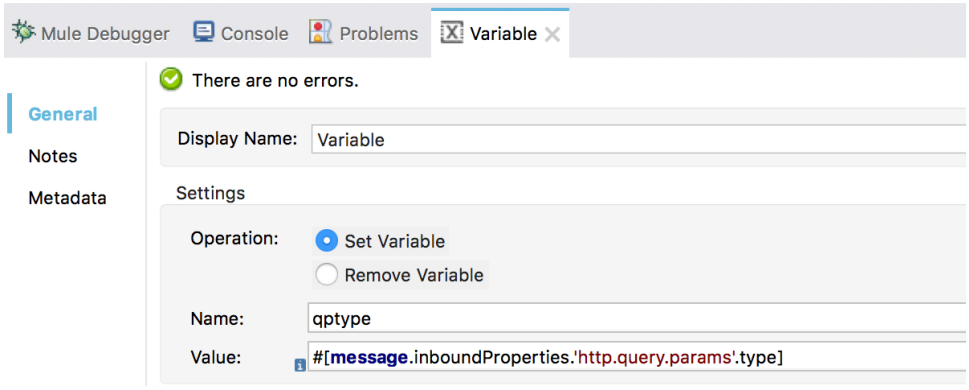


27. Click the Resume button.
28. Look at the console; you should see the value of your variable displayed.

Walkthrough 2-4: Read and write variables

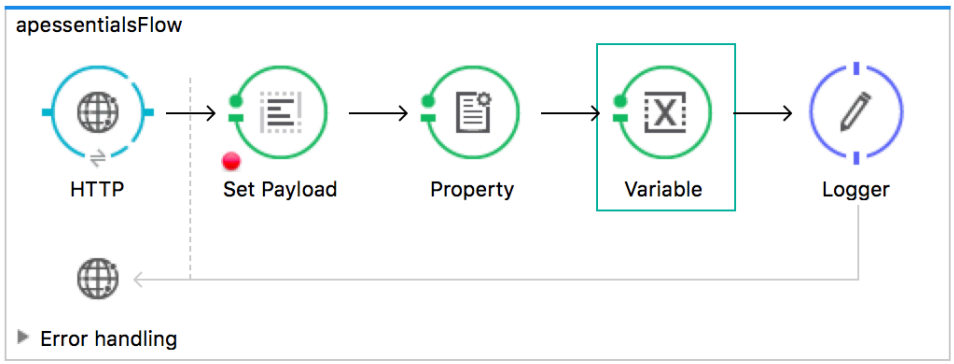
In this walkthrough, you will create flow and session variables. You will:

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

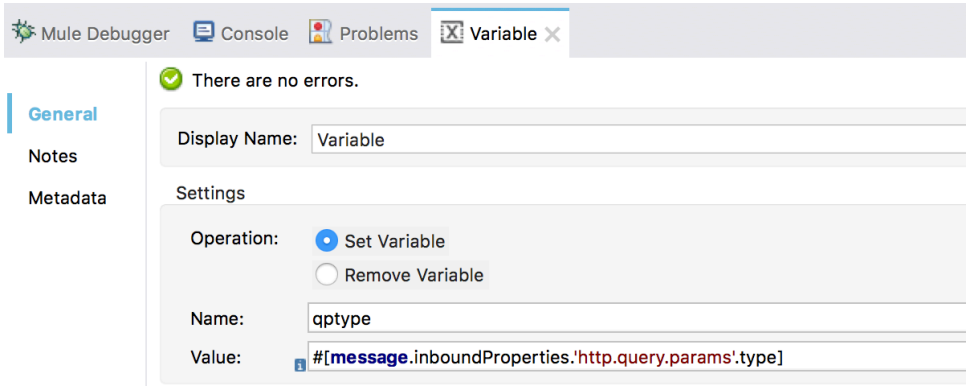


Create a flow variable

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.

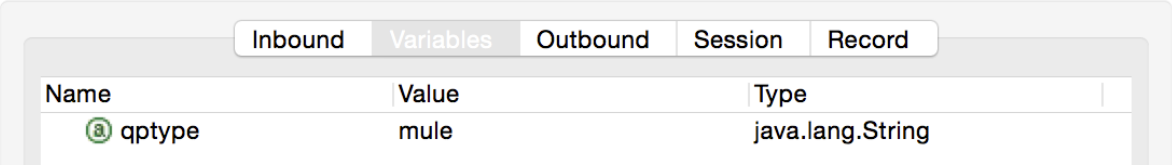


4. Modify the Logger to also display the value of this new variable.

   ```
   #['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.

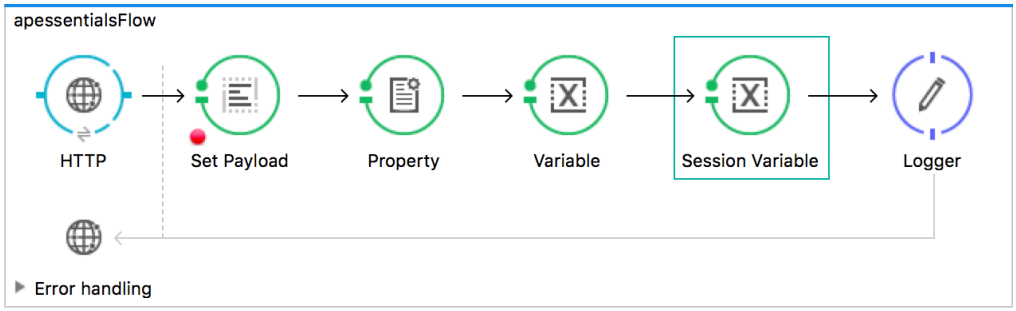


9. Click the Resume button.
10. Look at the console; you should see the value of your outbound property and the value of your new flow variable displayed.

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

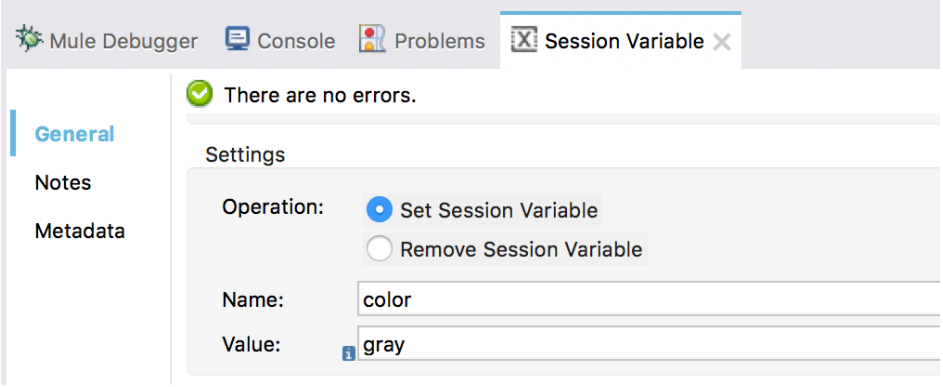
Create a session variable

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



12. In the Session Variable Properties view, select Set Session Variable.

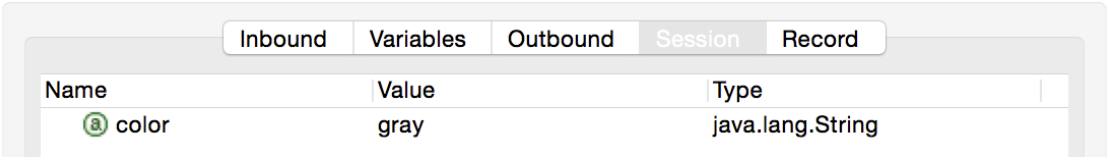
13. Set the name to color (or some other value) and give it any value, static or dynamic.



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

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

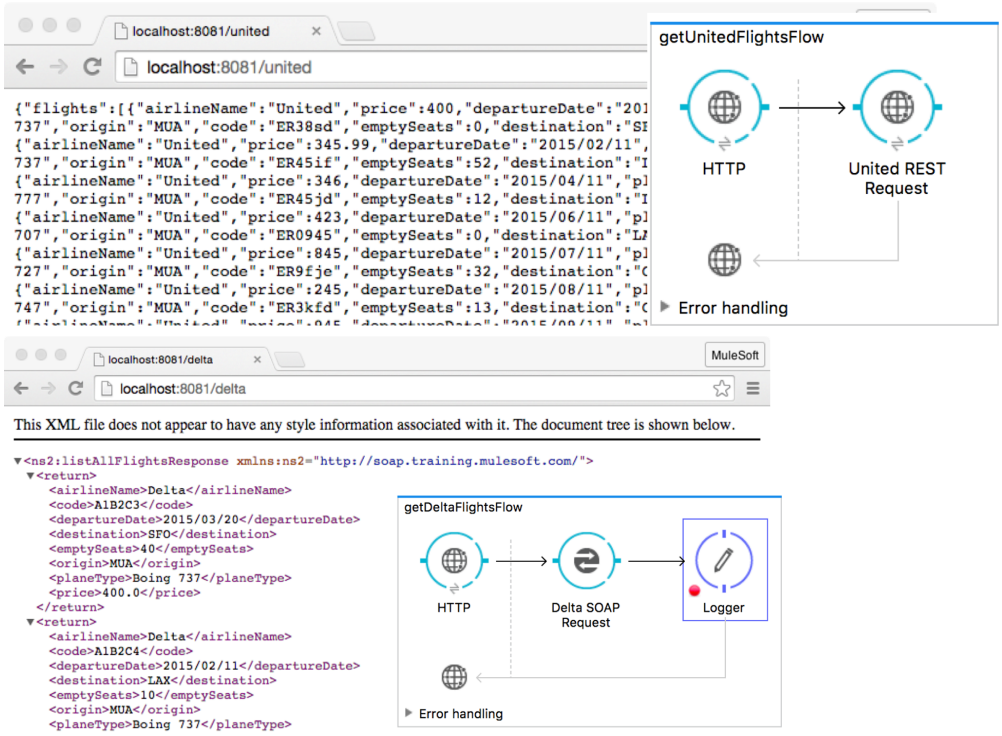
15. Save and redeploy the application and make a request to the endpoint with query parameters again.
16. Click the Session tab in the Mule Debugger view.
17. Step to the Logger; you should see your new session variable.



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



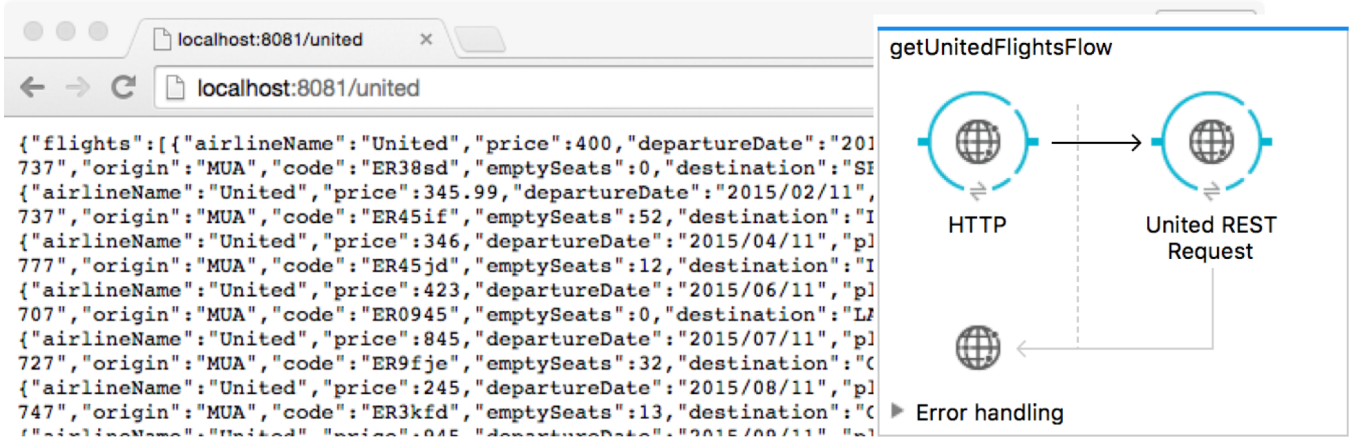
In this module, you will learn:

- 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.

Walkthrough 3-1: Consume a RESTful web service

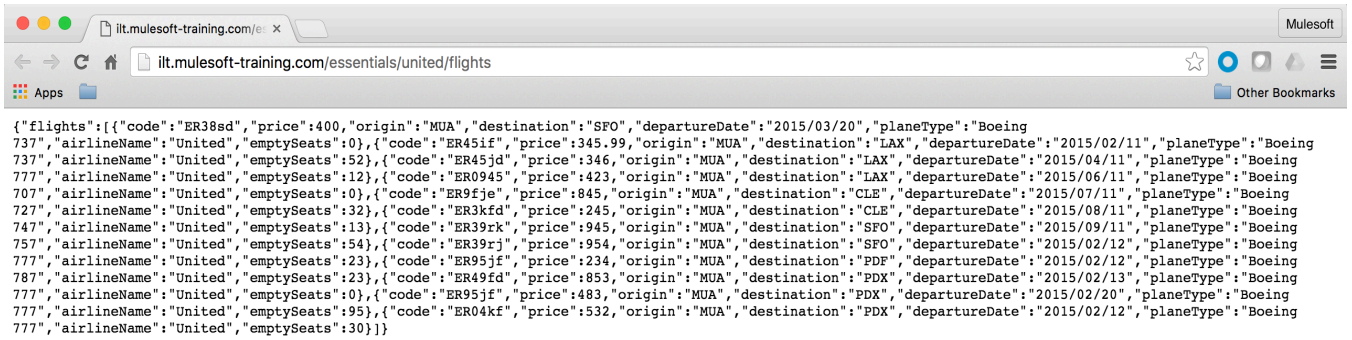
In this walkthrough and many others, you will work on building a Mule United Airline (MUA) application that returns flights for Delta, United, and American airlines. In this walkthrough, you will consume a RESTful web service that returns a list of all United flights as JSON. You will:

- 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

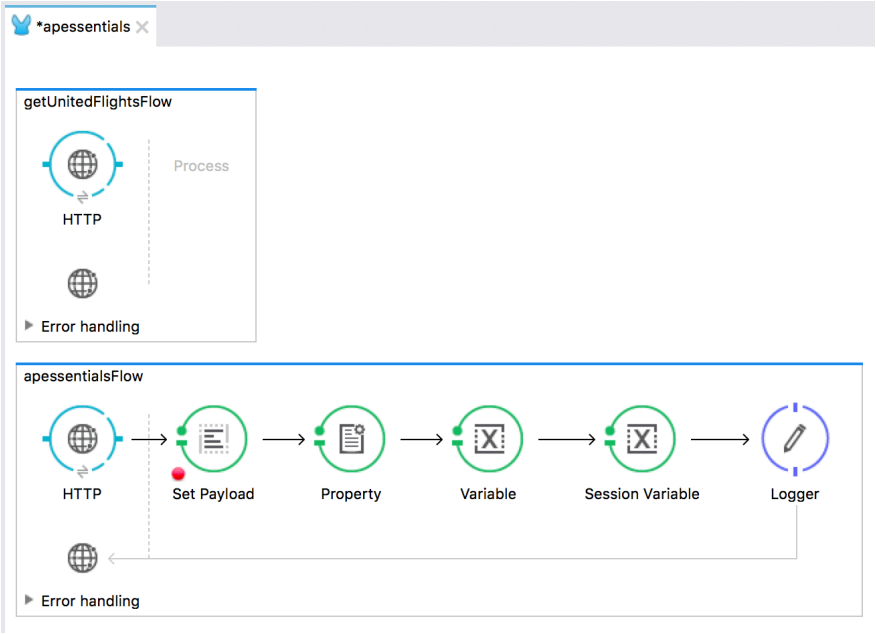
1. In your computer's file explorer, navigate to the student files folder for the course: APEssentials3.8_studentFiles_{date}.
2. Open the course snippets.txt file.
3. Make a request to the United RESTful web service URL listed in the course snippets.txt file; you should see JSON data for the United 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

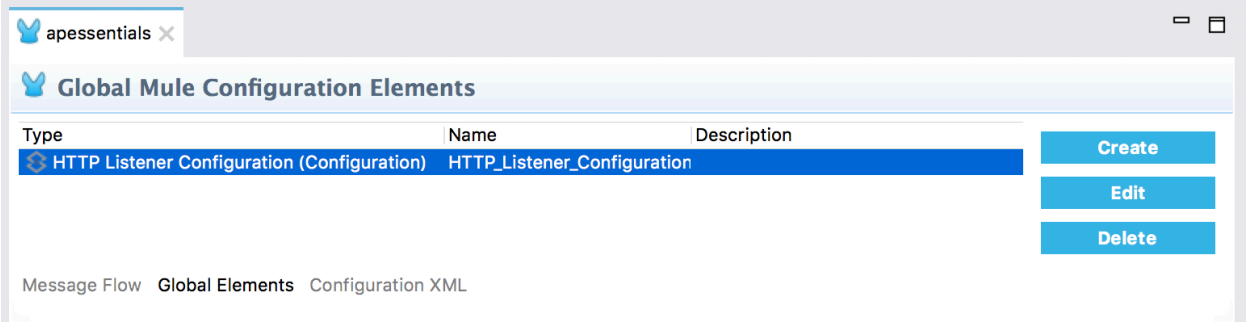
5. Return to apessentials.xml.
6. Drag out another HTTP connector and drop it in the canvas above the existing apessentialsFlow.
7. Double-click the name of the flow in the canvas and give it a new name of getUnitedFlightsFlow.

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



Look at the global elements

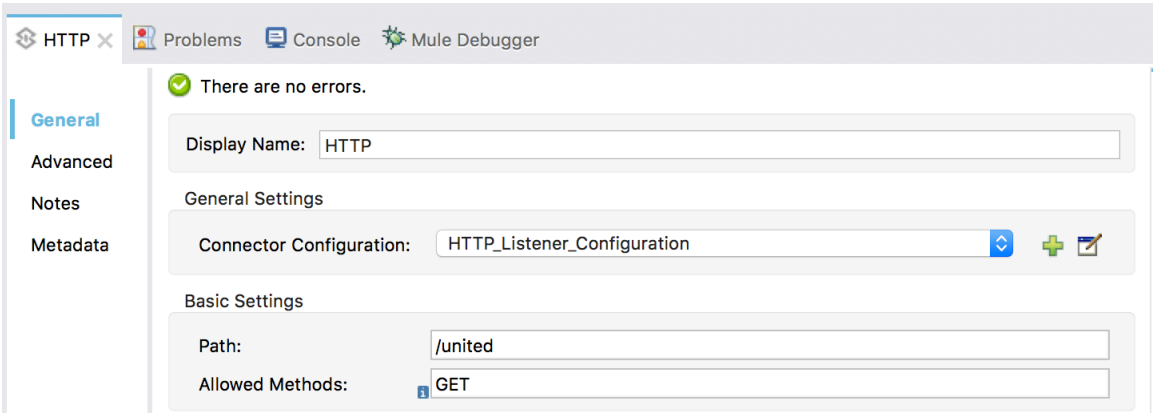
8. Click the Global Element tabs at the bottom of the canvas.
9. Select the HTTP Listener Configuration and click Edit (or double-click it).



10. In the Global Element Properties dialog box, review the HTTP_Listener_Configuration and click OK.

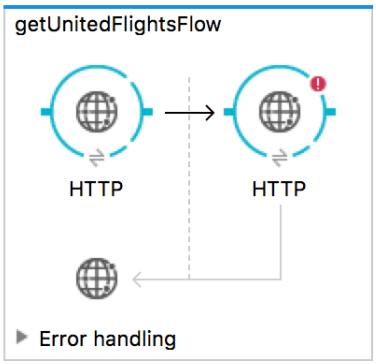
Configure the HTTP Listener connector endpoint

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

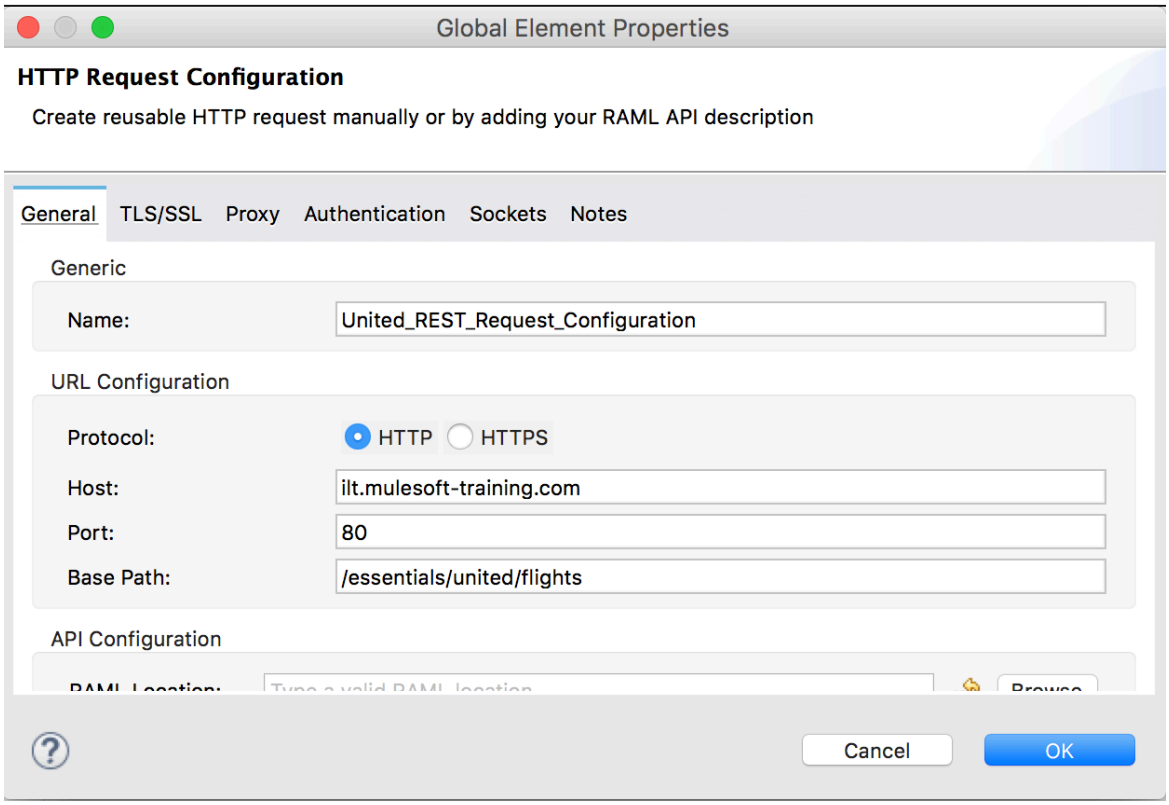
16. Drag out another HTTP connector and drop it into the process section of the flow.



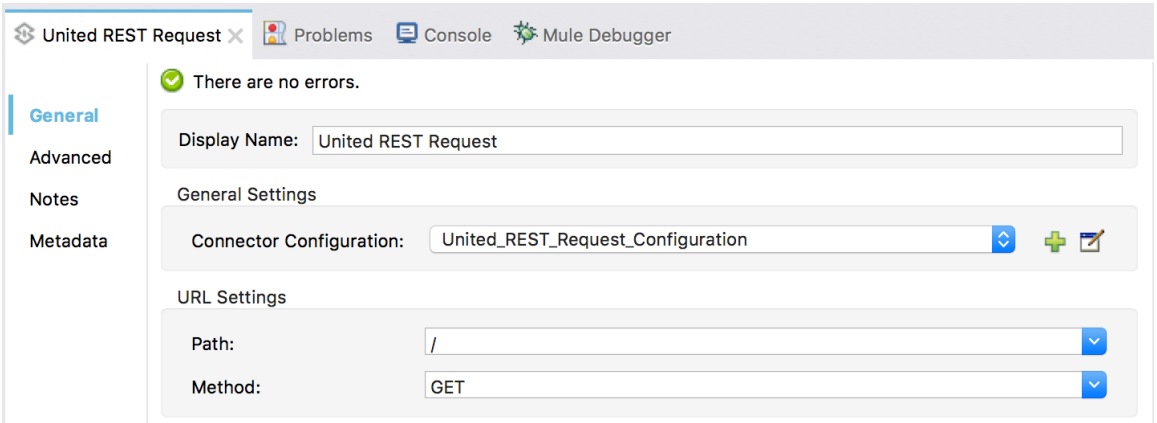
17. In the Properties view, change the display name to United REST Request.
18. Click the Add button next to connector configuration.

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.

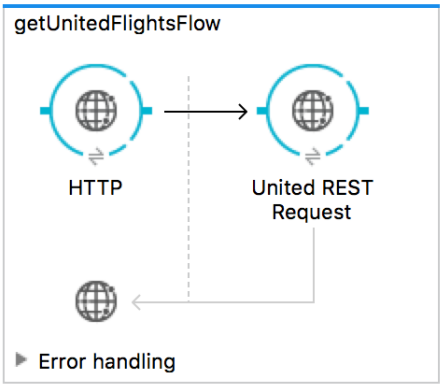


20. In the Properties view, set the path to / and the method to GET.



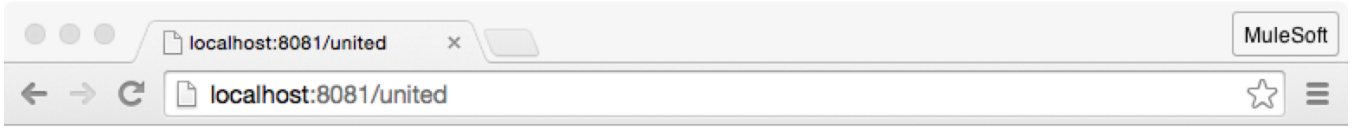
21. Click the Global Elements tab at the bottom of the canvas and see the new global configuration element.

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.



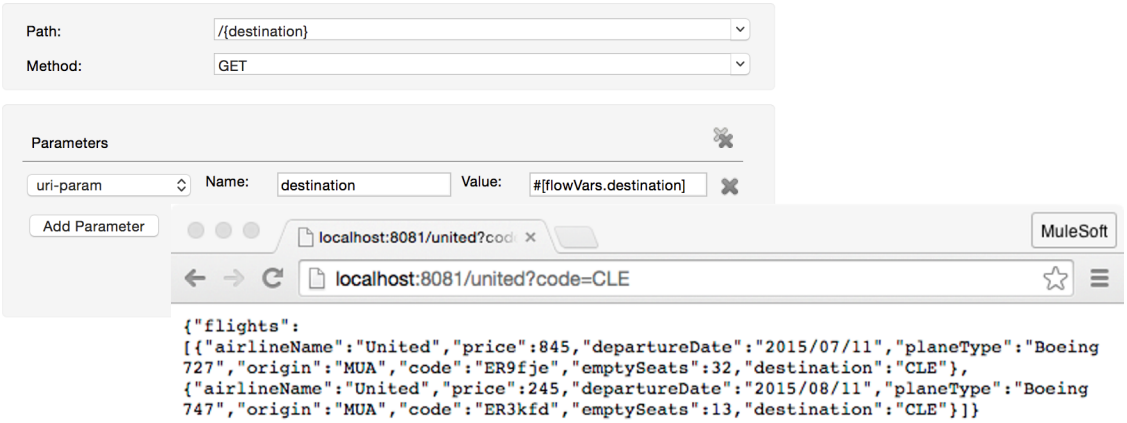

```
{"flights":[{"airlineName":"United","price":400,"departureDate":"2015/03/20","planeType":"Boeing
737","origin":"MUA","code":"ER38sd","emptySeats":0,"destination":"SFO"},
{"airlineName":"United","price":345.99,"departureDate":"2015/02/11","planeType":"Boeing
737","origin":"MUA","code":"ER45if","emptySeats":52,"destination":"LAX"},
{"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":"PDF"},
{"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:

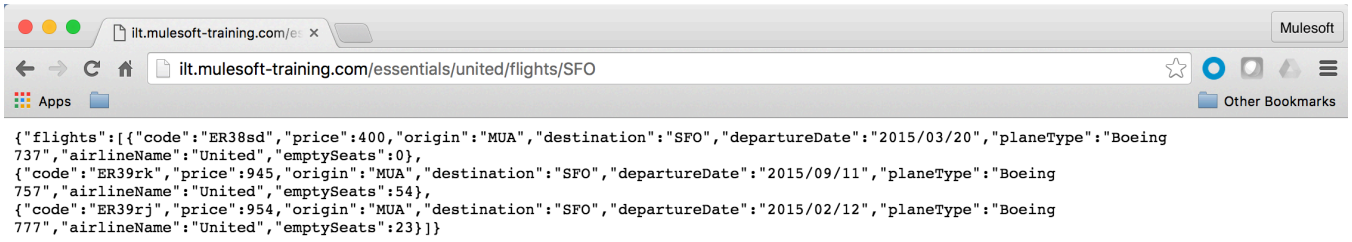
- Modify the HTTP Request connector endpoint to use a URI parameter for the destination.
- Set the destination to a static value.
- Set the destination to a dynamic query parameter value.
- Create a variable to set the destination.

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



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.

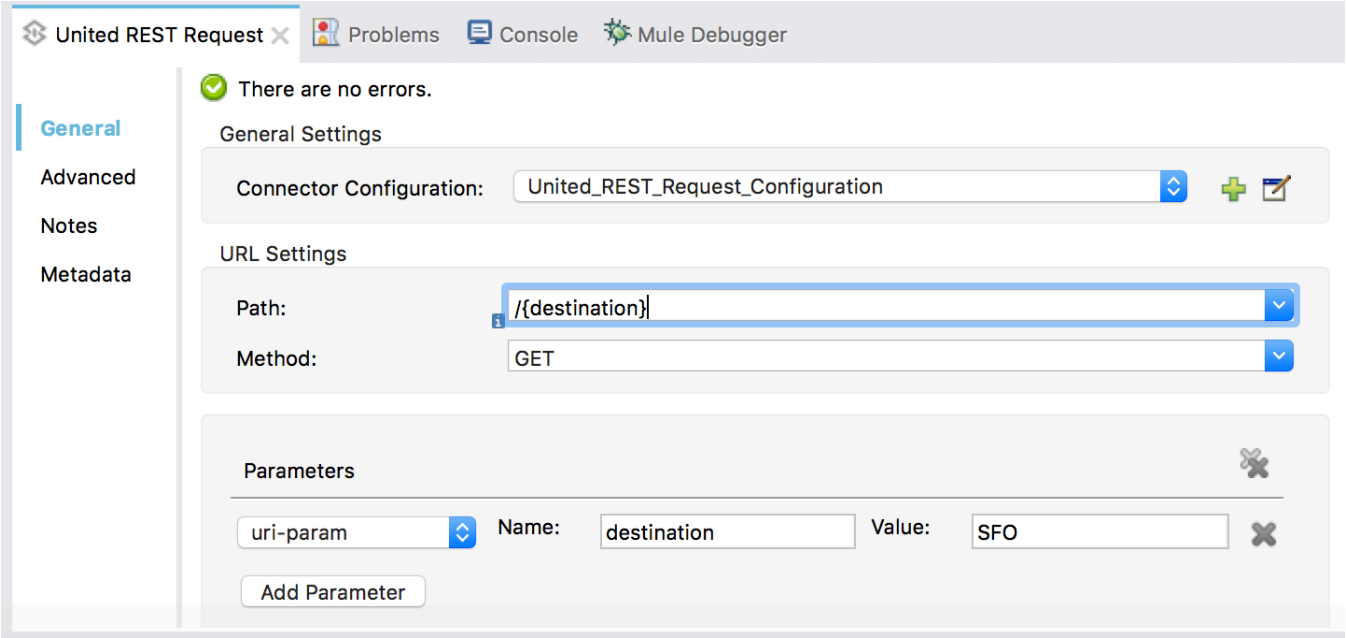


2. Make additional requests for destinations of LAX, CLE, PDX, or PDF.

Add a URI parameter with a static value

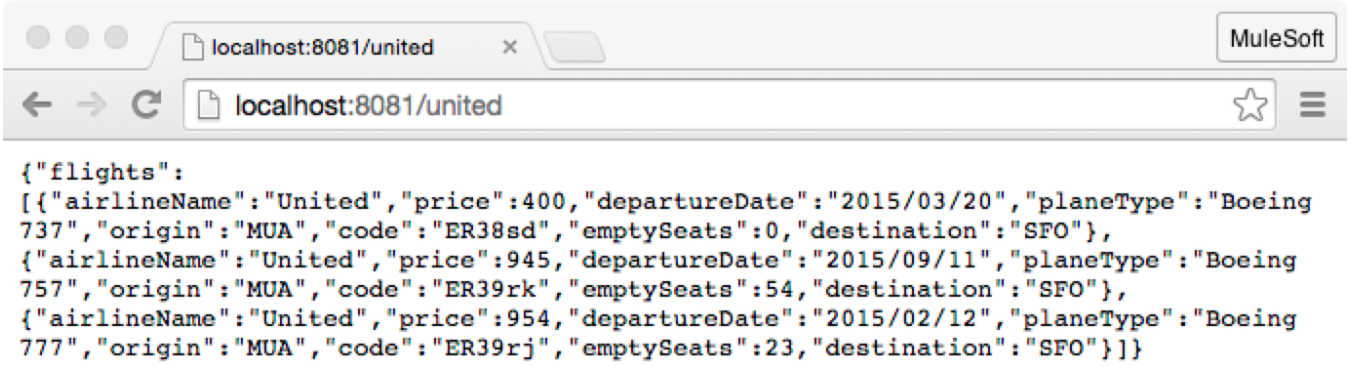
3. Return to apessentials.xml.
4. Double-click the United REST Request endpoint.
5. In the Properties view, click the Add Parameter button.

6. Set the following parameter values.
   - Parameter type: uri-param
   - Name: destination
   - Value: SFO
7. Change the United REST Request endpoint path to /{destination}.



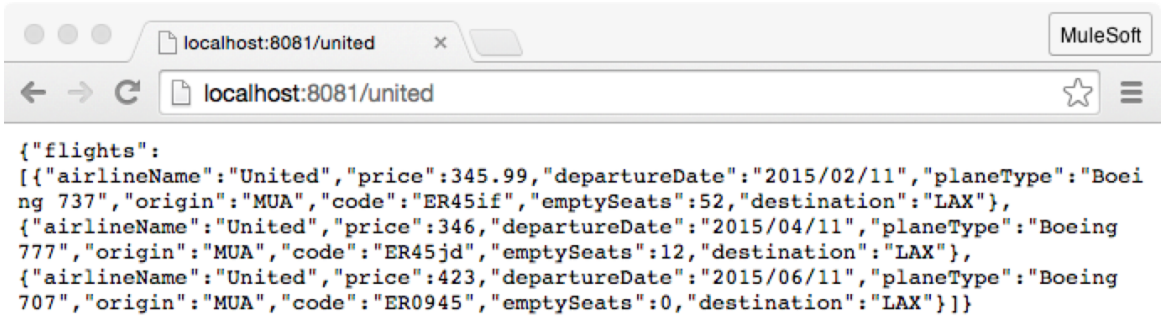
Test the application

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.



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.

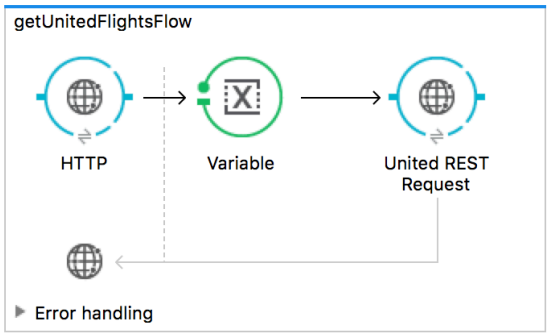
```
#[message.inboundProperties.'http.query.params'.code]
```

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.

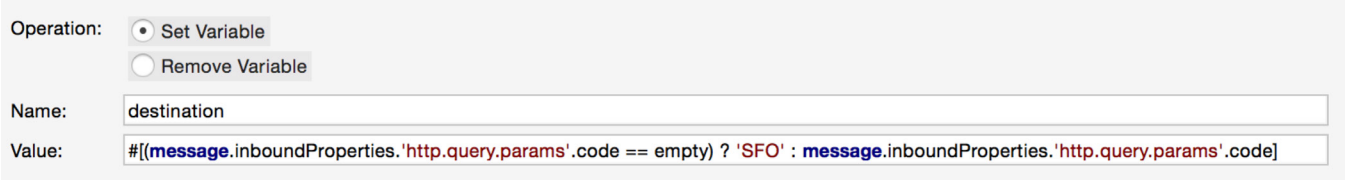
Create a variable to set the destination

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

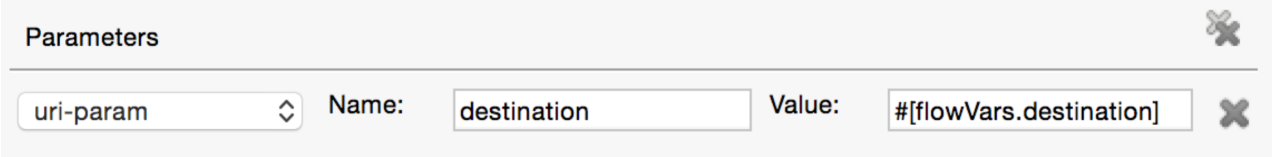


15. In the Variable Properties view, change the display name to Set Destination.
16. Set the operation to Set Variable and the name to destination.
17. Use a ternary expression to set the value to 'SFO' or the value of a query parameter called code.

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



18. Navigate to the Properties view for the United REST connector endpoint.
19. Modify the URI parameter to use the new destination variable.

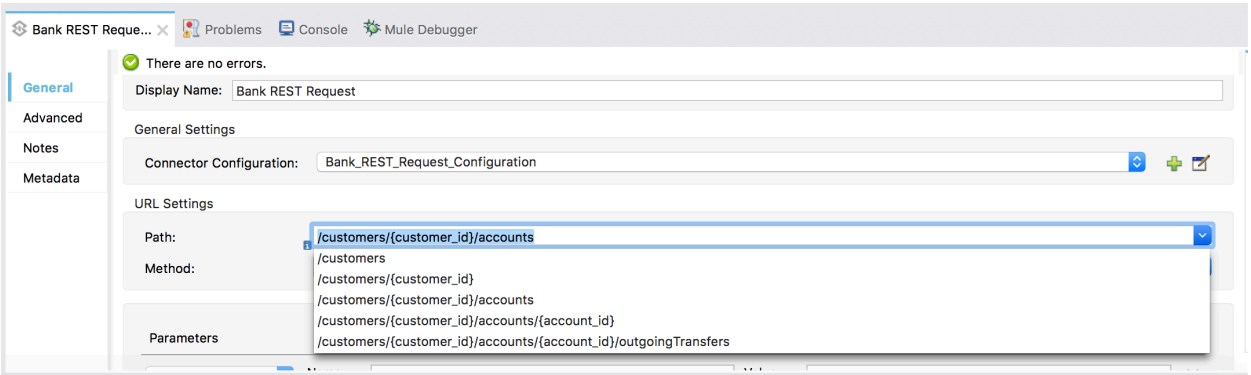


20. Save and redeploy the application and make a request to http://localhost:8081/united; you should see only flights to SFO again.
21. Make another request to http://localhost:8081/united?code=PDX; you should now see flights to PDX.

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

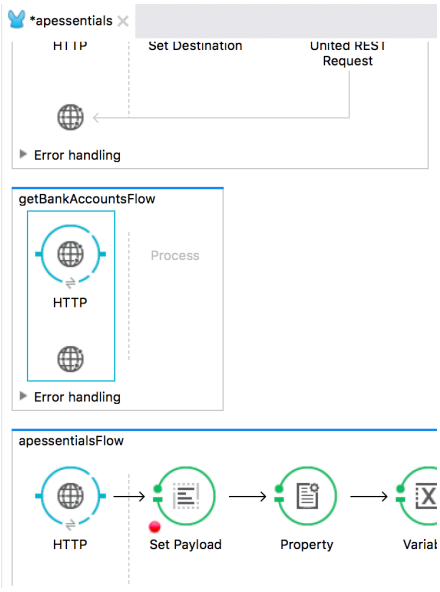
In this walkthrough, you will consume a RESTful web service containing some simple bank account data that has a RAML definition file. You will:

- 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.

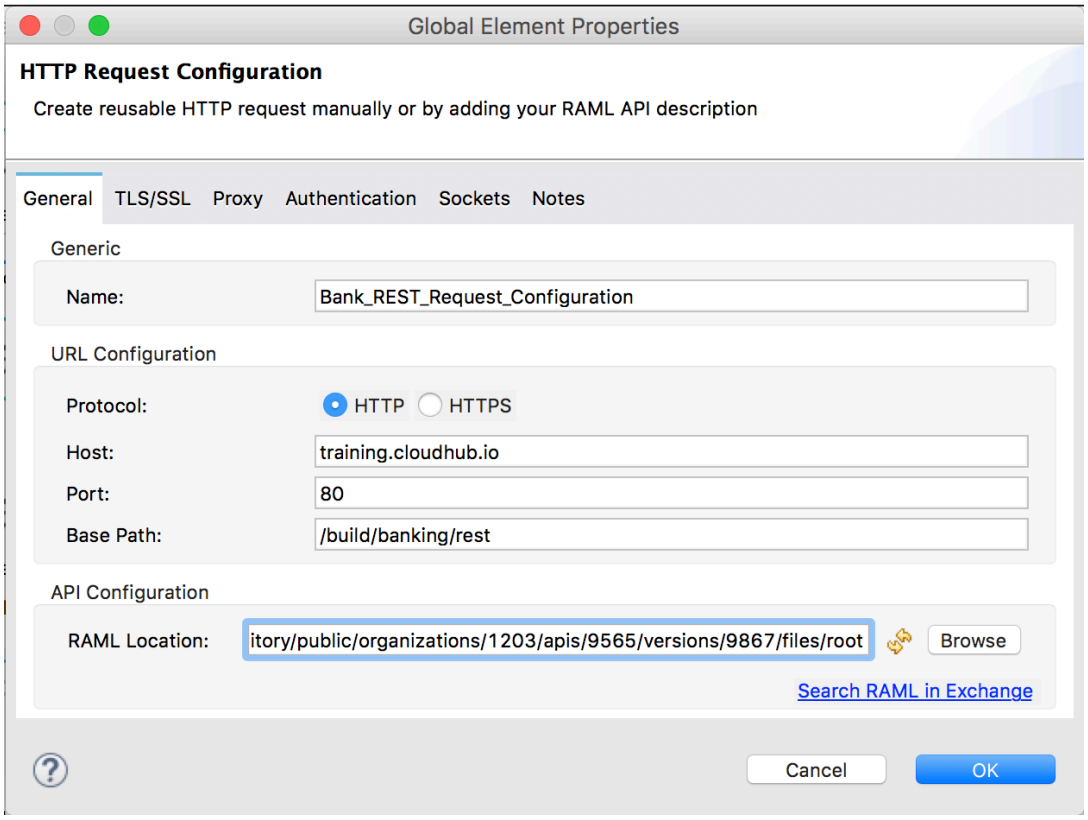


4. In the Properties view for the endpoint, set the connector configuration to the existing HTTP_Listener_Configuration.
5. Set the path to /bank.
6. Set the allowed methods to GET.

Add an HTTP Request connector with a RAML location

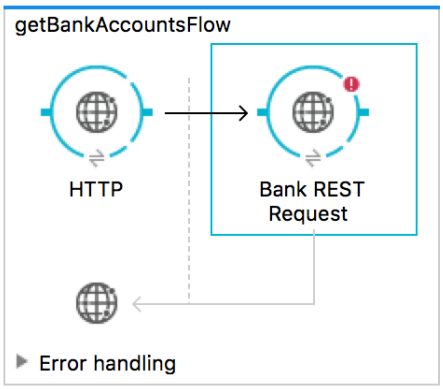
7. Drag out another HTTP connector and drop it in the process section of the new flow.
8. In the Properties view, change the name to Bank REST Request.
9. Click the Add button next to connector configuration.
10. In the Global Element Properties dialog box, change the name to Bank_REST_Request_Configuration.
11. Set the RAML location to the Banking RAML URL listed in the course snippets.txt file.
12. 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.

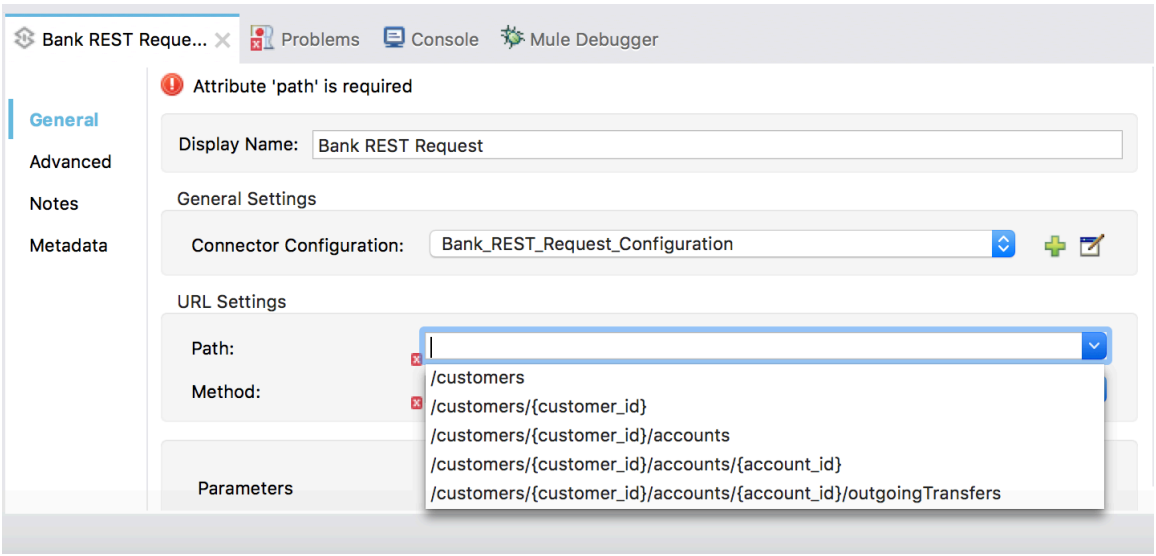


13. Click the Global Elements tab at the bottom of the canvas and see the new global configuration element.

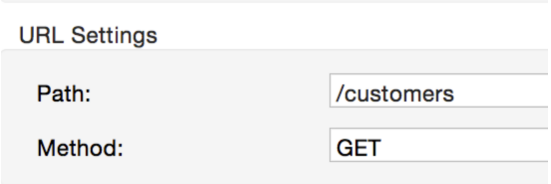
14. Return to the Message Flow view.



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.

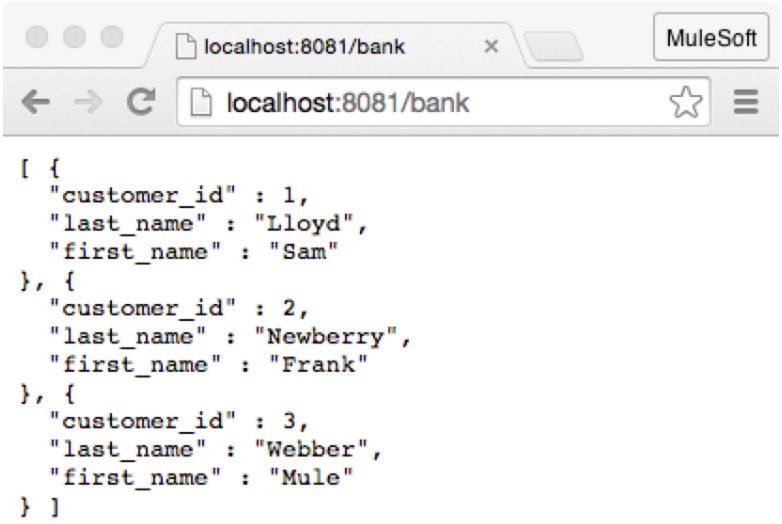


16. Select the /customers path.
17. Set the method to GET.



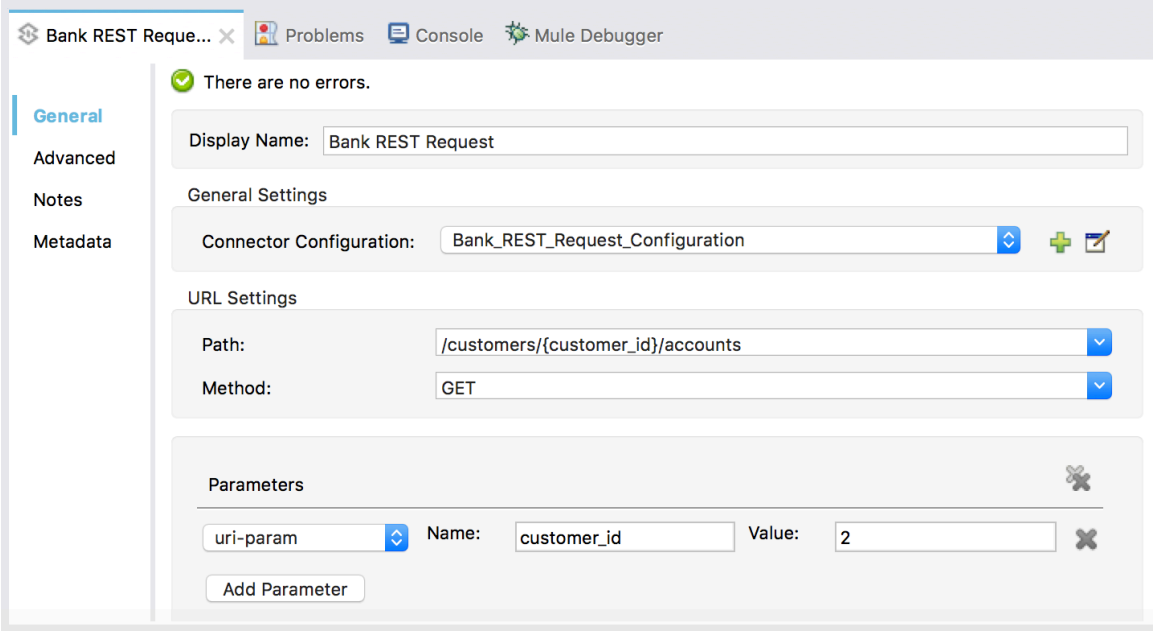
Test the application

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



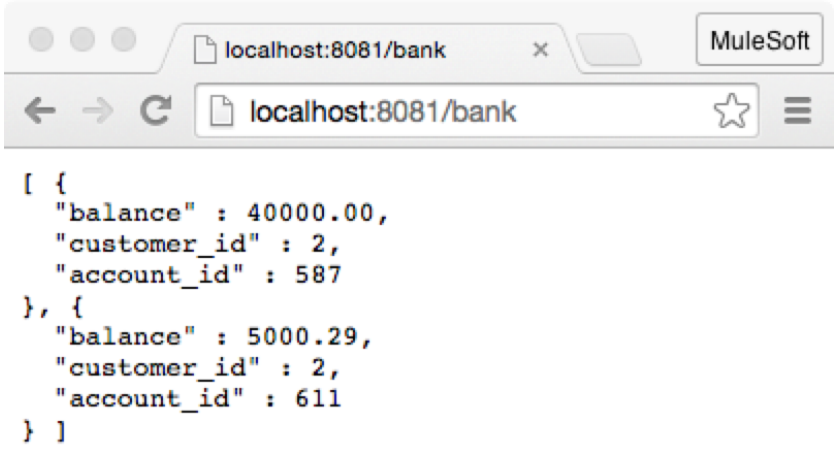
Modify the request to use a path requiring a URI parameter

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.

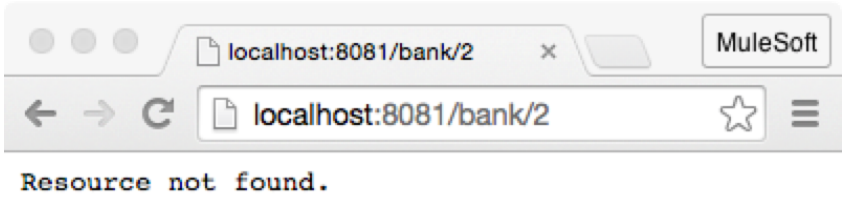


Test the application

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.

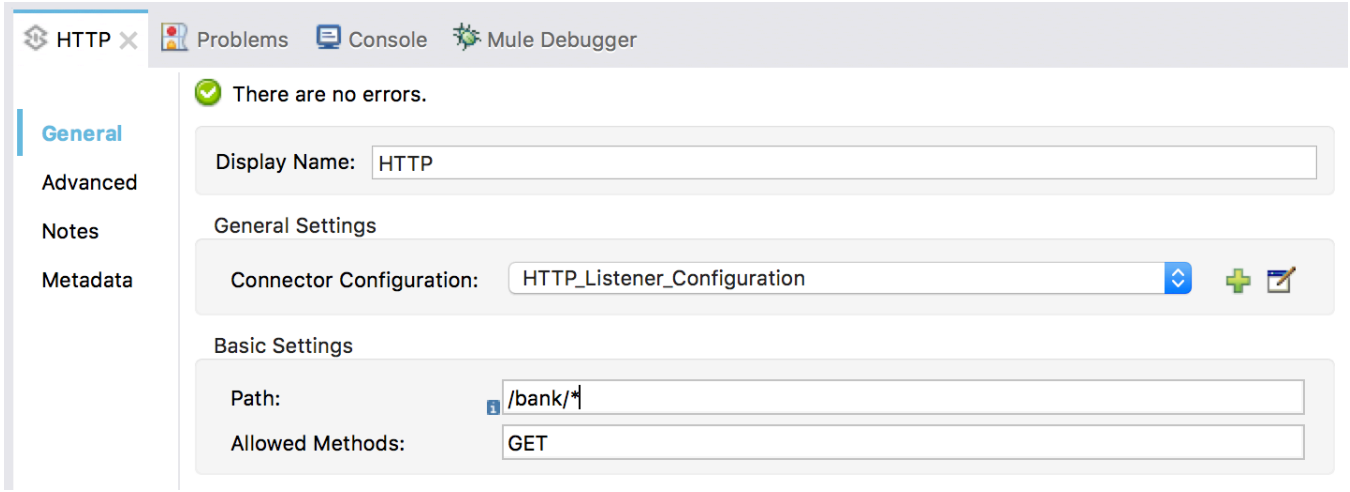


23. Make a request and try to pass the customer ID as a URI parameter: http://localhost:8081/bank/2; you should get a Resource not found response.



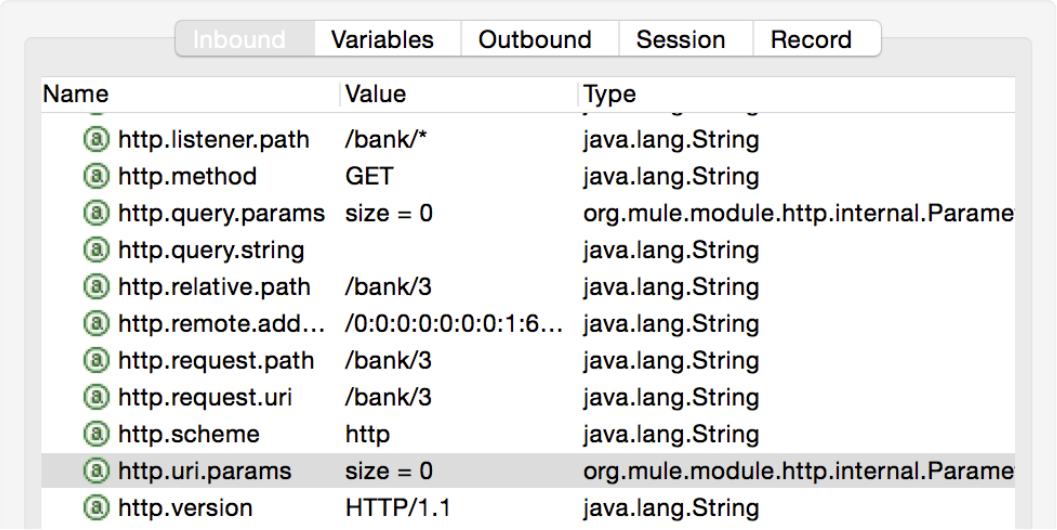
Get the value of an HTTP Listener endpoint URI parameter

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/.

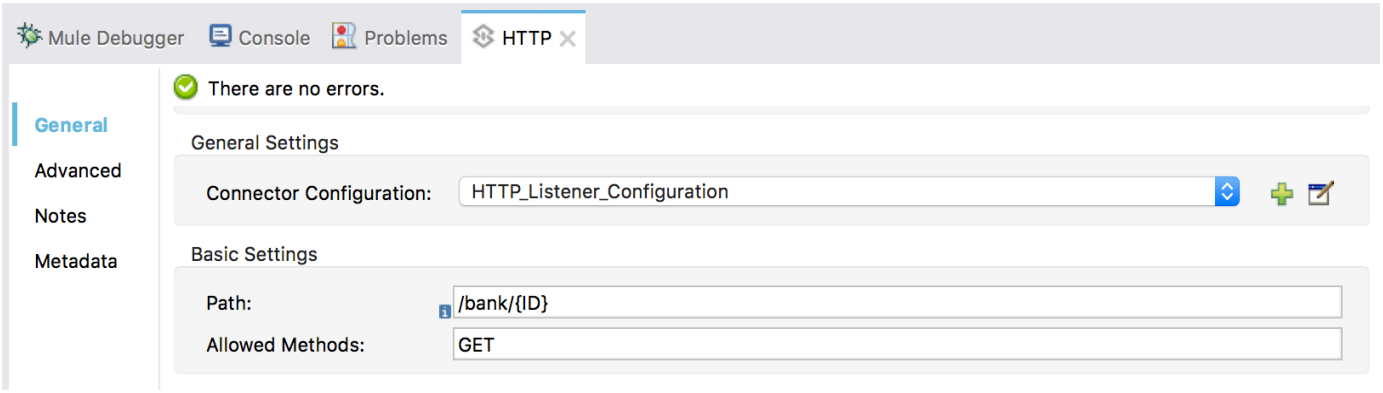


26. Add a breakpoint to the Bank REST Request endpoint.

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.

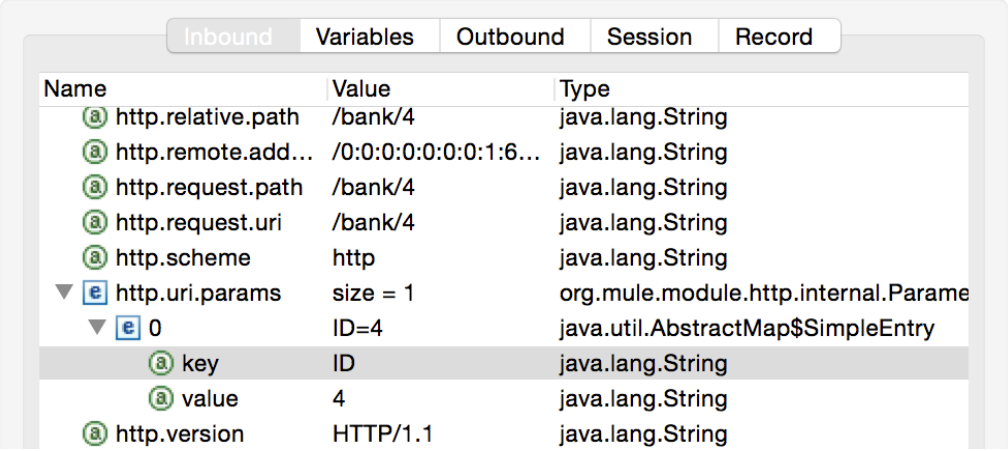


30. Step through the rest of the application; you should still get the account info for the customer with an ID of 2.
31. Return to the Properties view for the HTTP Listener endpoint.
32. Change the path to specify a URI parameter called ID: /bank/{ID}.



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.



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

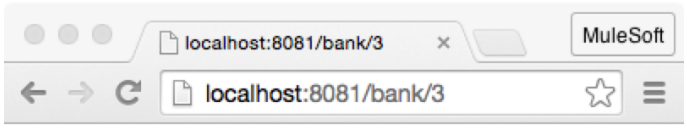
Set the HTTP Request endpoint URI parameter to a dynamic value

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.

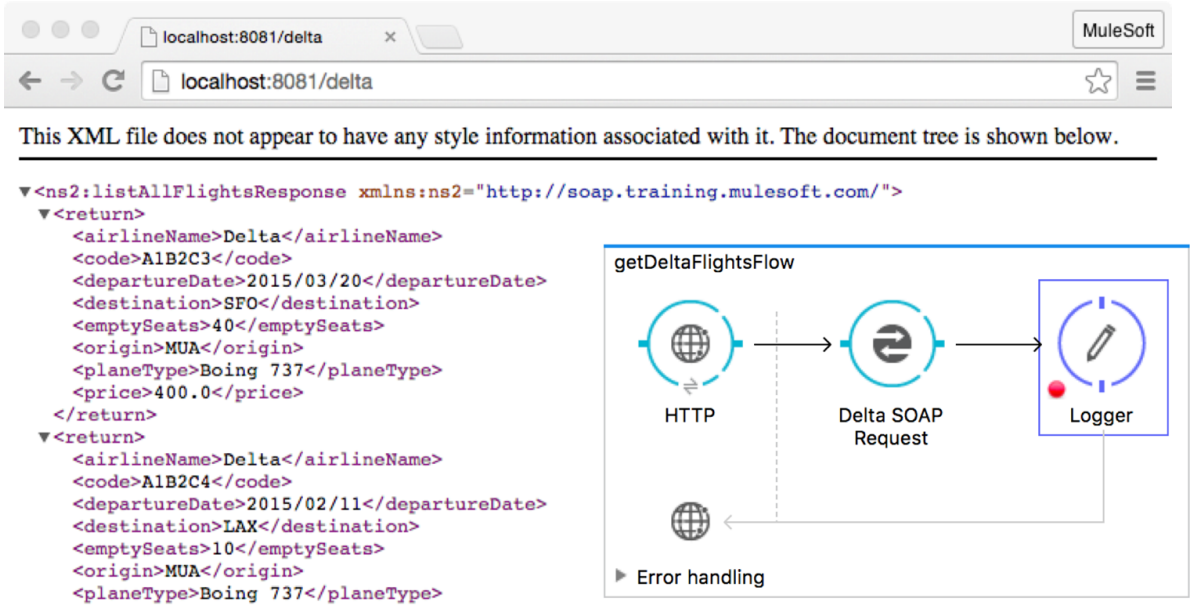



```
[ {
  "balance" : 50000.00,
  "customer_id" : 3,
  "account_id" : 588
}, {
  "balance" : 9999.99,
  "customer_id" : 3,
  "account_id" : 600
} ]
```

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

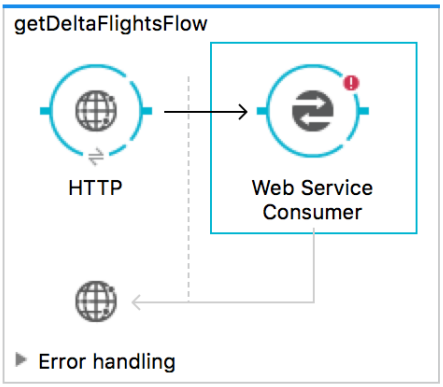
1. Make a request to the Delta SOAP web service WSDL listed in the course snippets.txt file; you should see the web service WSDL returned.
2. Browse the WSDL; you should find references to operations listAllFlights and findFlight.

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

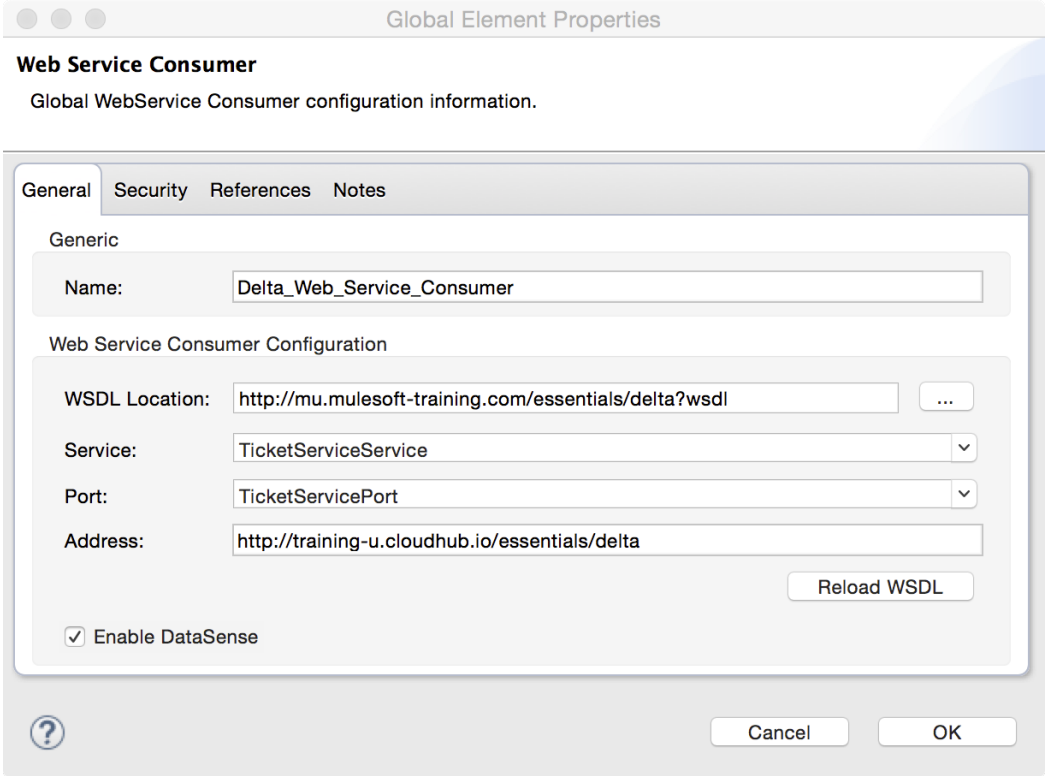
9. Drag out a Web Service Consumer connector and drop it in the process section of the flow.



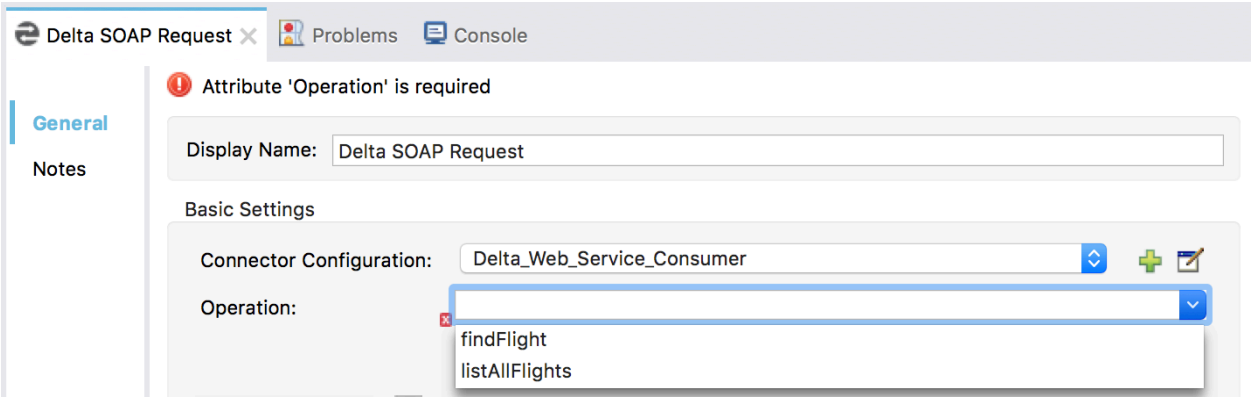
10. In the Properties view, set the display name to Delta SOAP Request.
11. Click the Add button next to connector configuration.
12. In the Global Element Properties dialog box, change the name to Delta_Web_Service_Consumer.
13. Set the WSDL Location to the Delta SOAP web service WSDL listed in the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

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

    Note: If the fields do not populate automatically, select the service and the port from the drop-down menus.



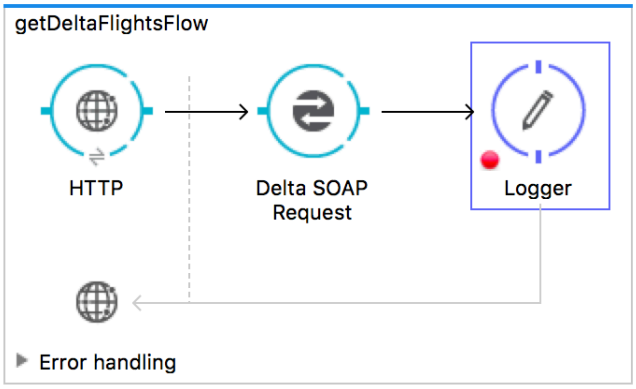
15. Click OK.
16. Switch to the Global Elements view and see the new global configuration element.
17. Return to the Message Flow view.
18. In the Delta SOAP Request Properties view, click the operation drop-down button; you should see all of the web service operations listed.



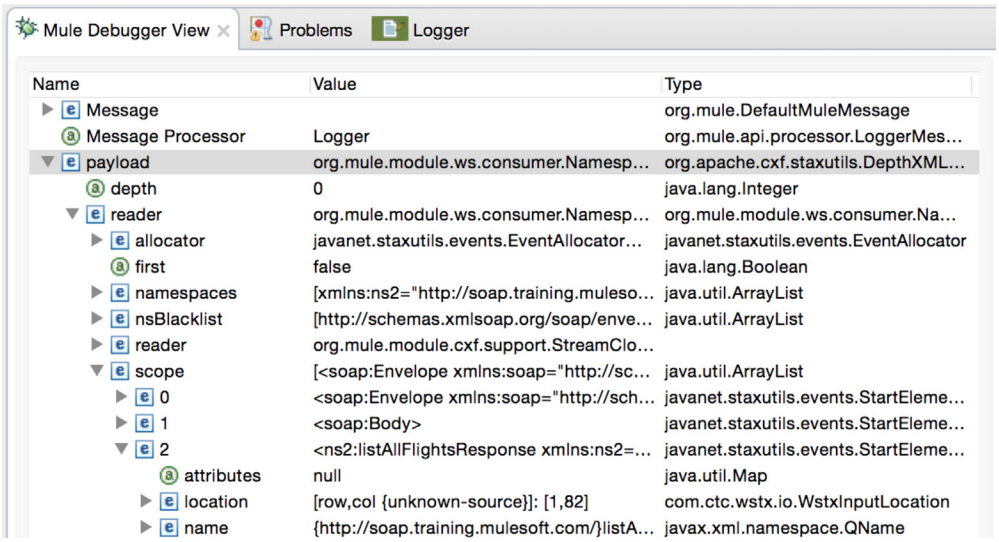
19. Select the listAllFlights operation.

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.



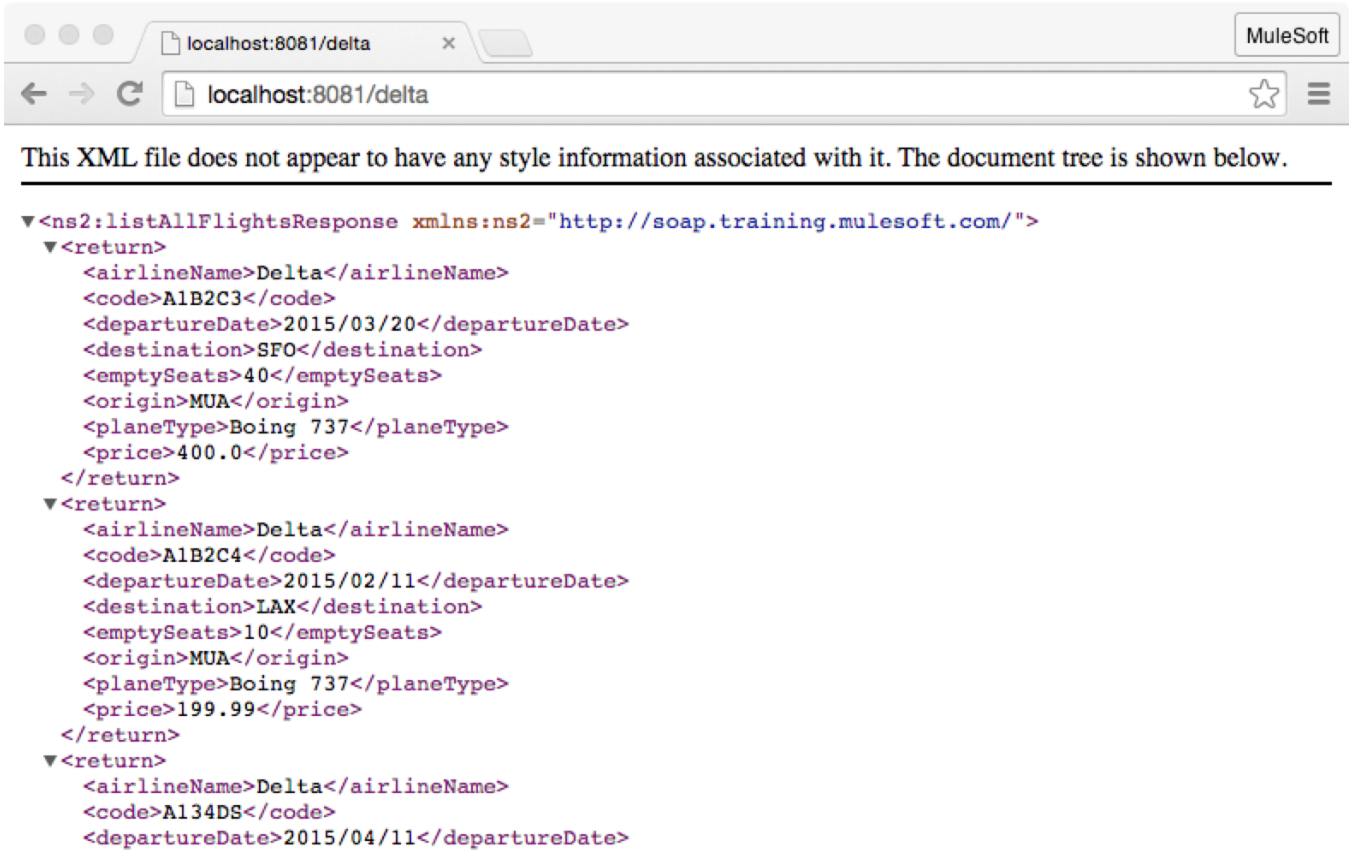
23. Save the file and debug the application.
24. Make a request to http://localhost:8081/delta.
25. In the Mule Debugger view, drill-down into the payload variable.



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

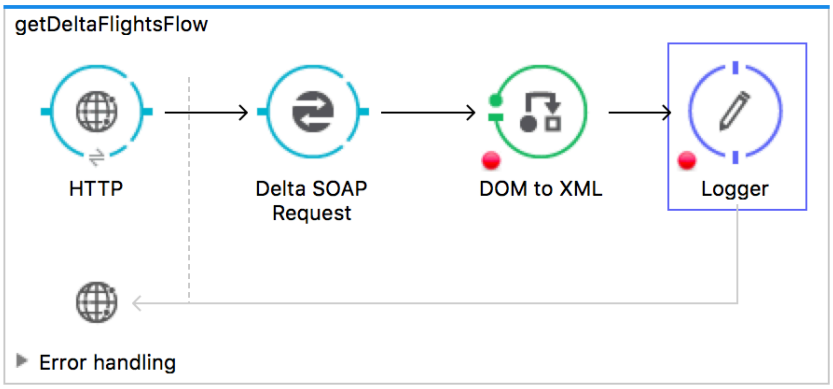
```
INFO  2015-08-19 12:38:25,676 [[apessentials].HTTP_Listener_Configuration.worker.01] org.mule.api.process
or.LoggerMessageProcessor: org.mule.module.ws.consumer.NamespaceRestorerXMLStreamReader@2db53627
```

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



Display the payload as a String

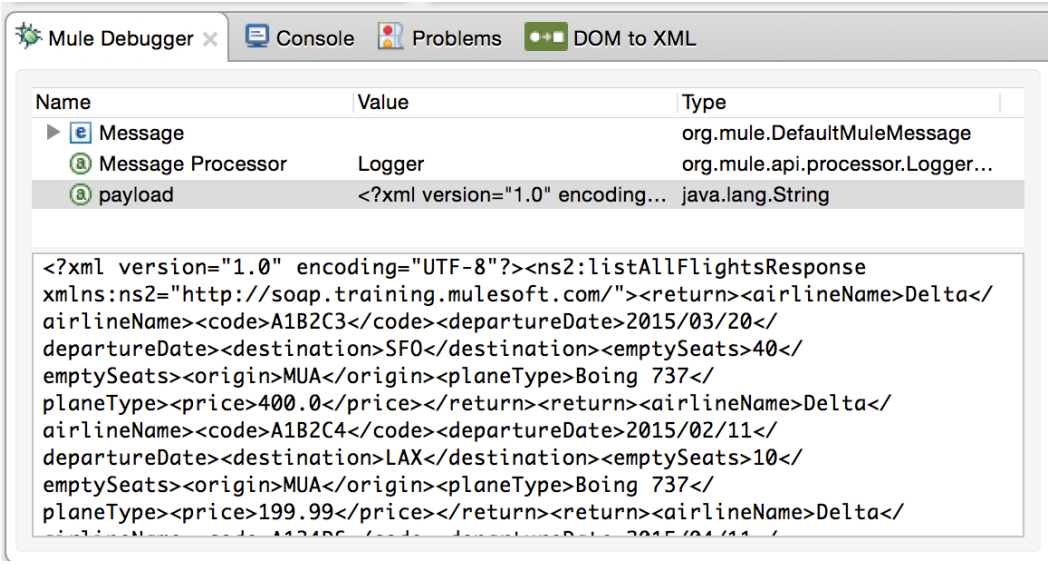
29. Add a DOM to XML transformer before the Logger.



Test the application

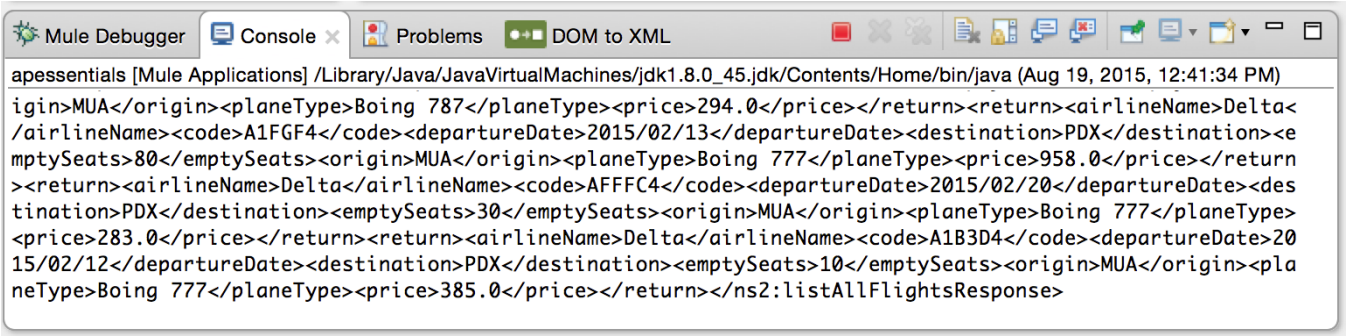
30. Save the file to redeploy the application.
31. Make another request to http://localhost:8081/delta.

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



33. Resume the application.

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



35. Stop the runtime.

Module 4: Connecting to Additional Resources

In this module, you will learn:

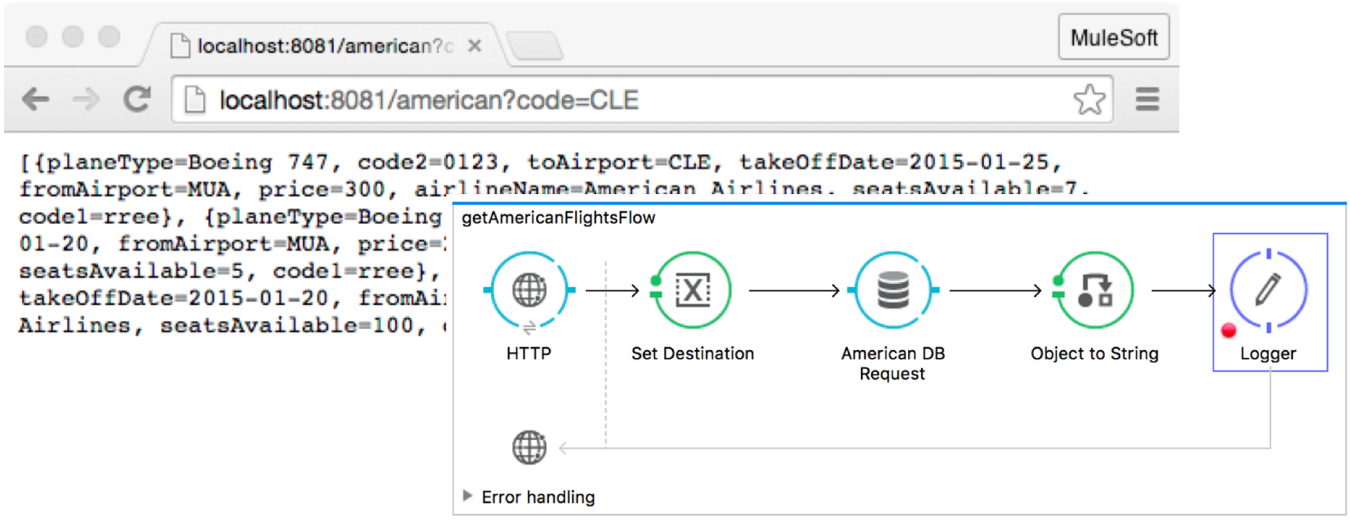
- To connect to databases.
- To connect to files.
- To connect to JMS queues.
- To connect to SaaS applications.
- To discover and install connectors not bundled with Anypoint Studio.
- About developing your own custom connectors with Anypoint Connector DevKit.

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.

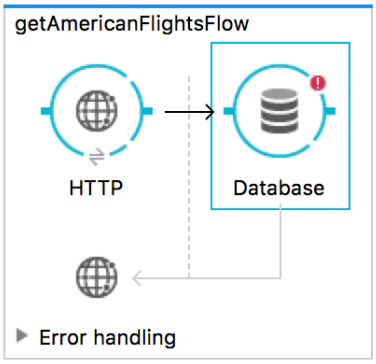


Create a new flow with an HTTP Listener connector endpoint

1. Return to apessentials.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.

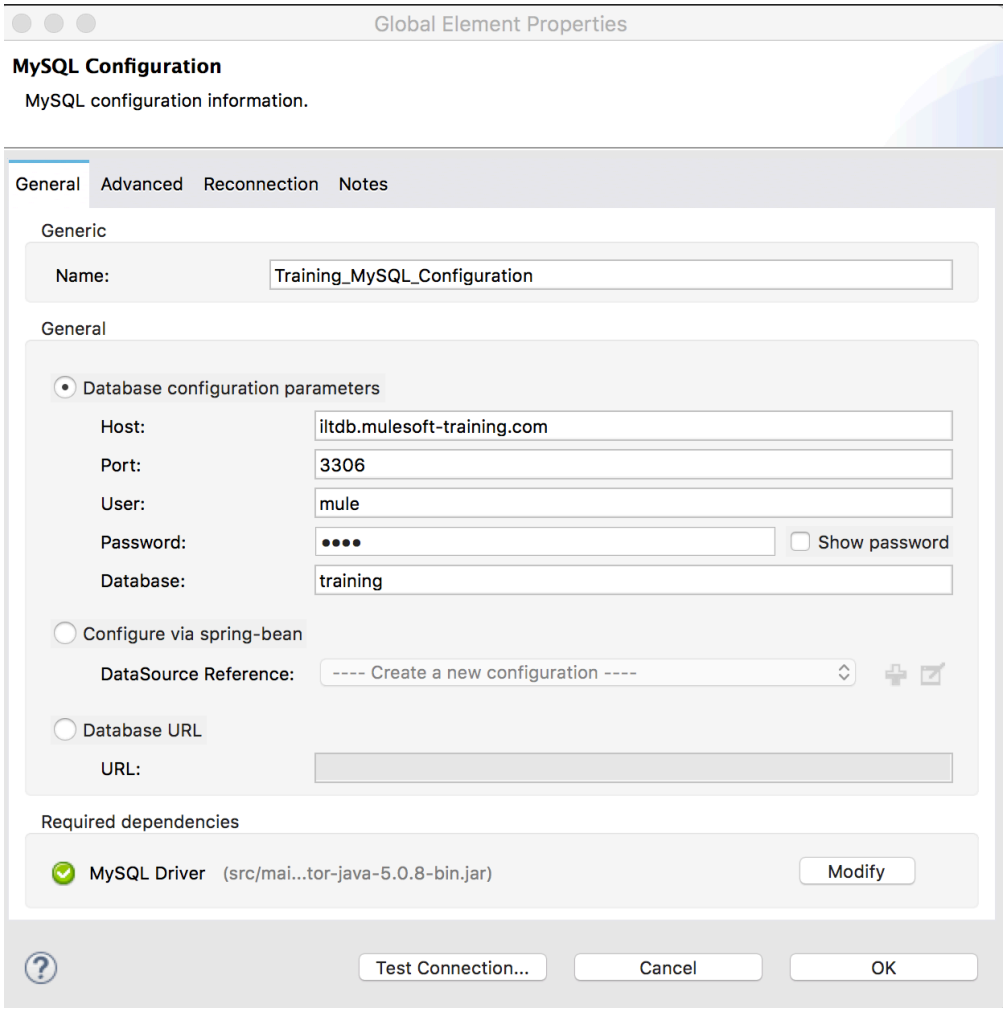
Add and configure a Database connector endpoint

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

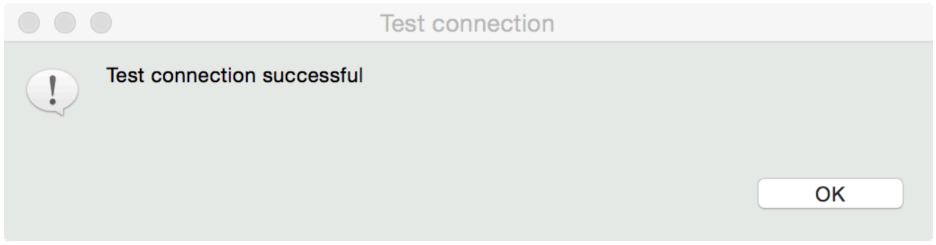


6. In the Database Properties view, change the display name to American DB Request.
7. Click the Add button next to connector configuration.
8. In the Choose Global Type dialog box, select Connector Configuration > MySQL Configuration and click OK.
9. In the Global Element Properties dialog box, set 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 APEssentials3.8_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.

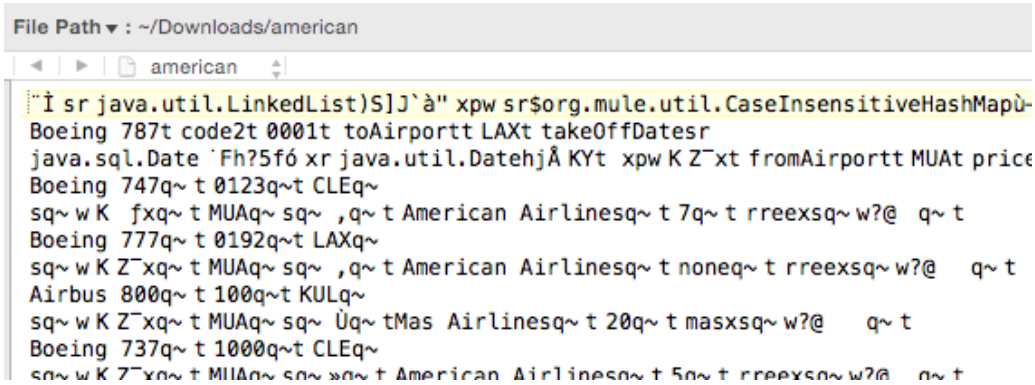
Write a query to return all flights

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

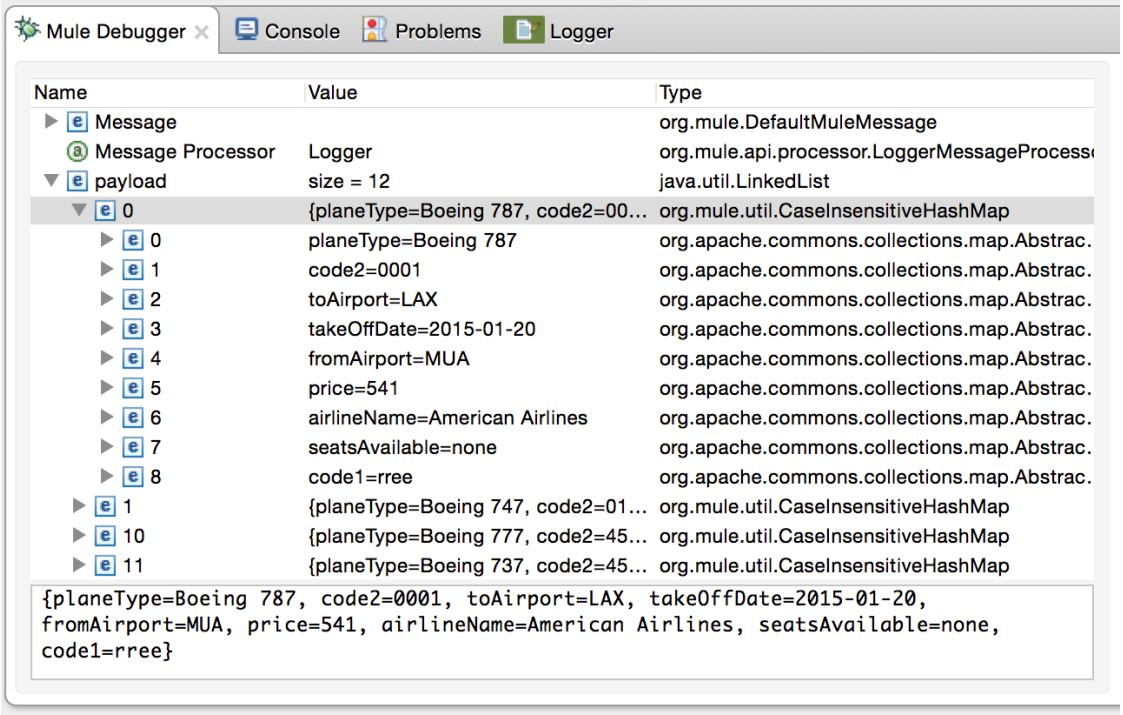
18. Save the file and run the application.
19. Make a request to http://localhost:8081/american; you should get some garbled plain text displayed or contained in a download file, which is the tool's best representation of an ArrayList.



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.

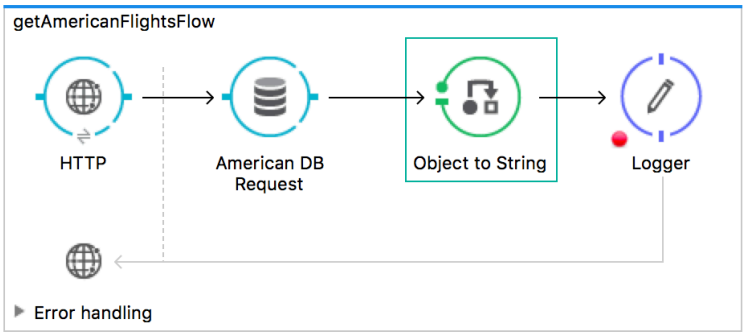
23. Find the name of the field for the flight destination.



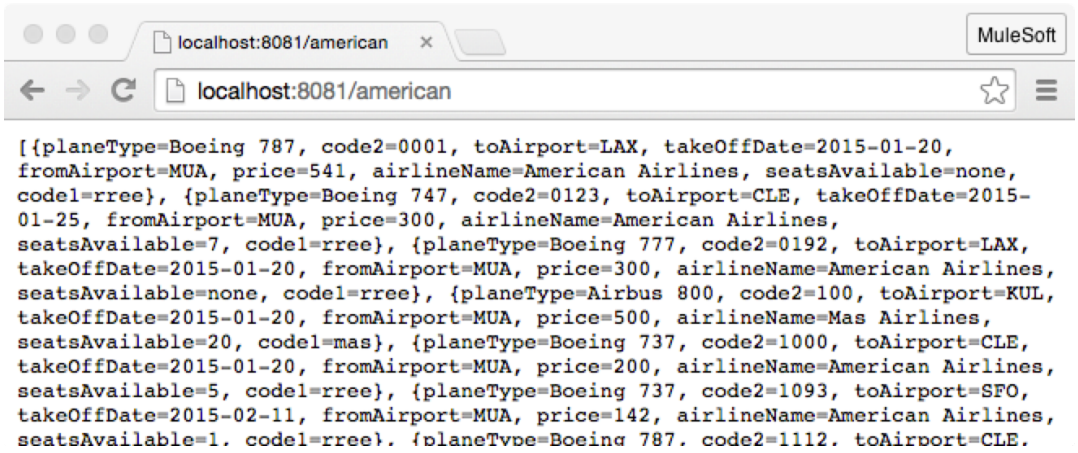
24. Stop the Mule Debugger.

Display the return data

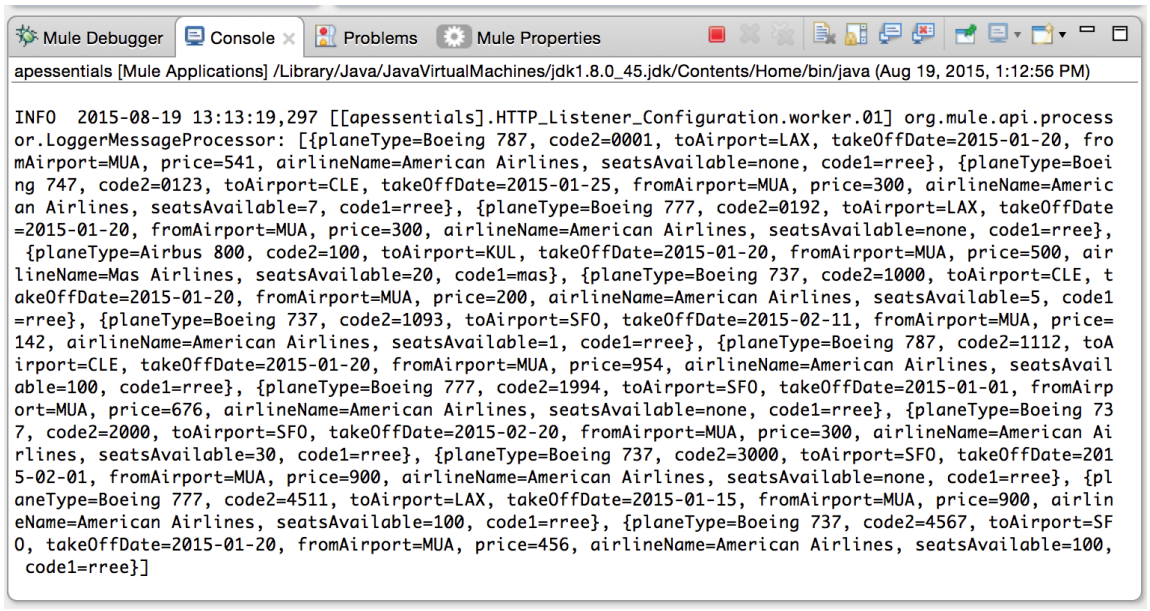
25. Set the Logger message to #[payload].
26. Add an Object to String transformer before the Logger component.



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.

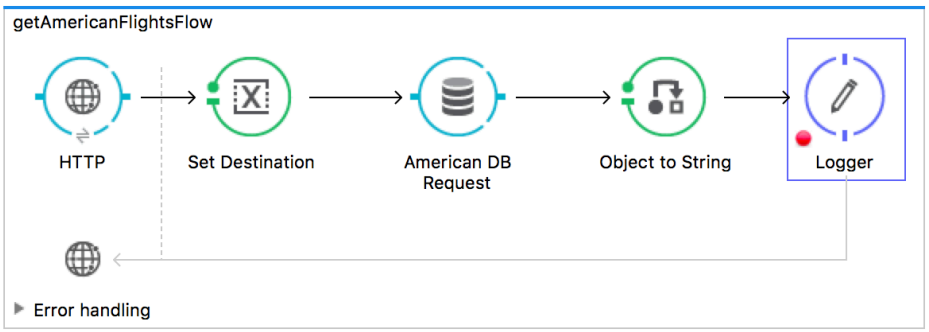


Note: To set the width of the console, select Anypoint Studio > Preferences > Run/Debug > Console and select Fixed width console and set a maximum character width.

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.

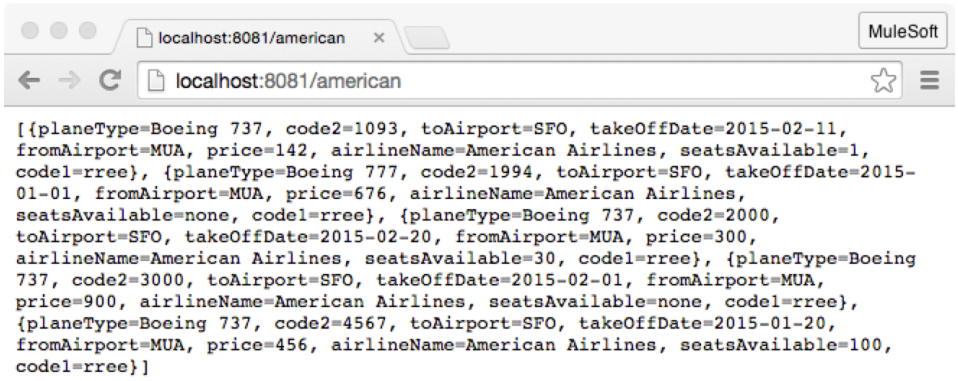


Modify the query to use a dynamic destination

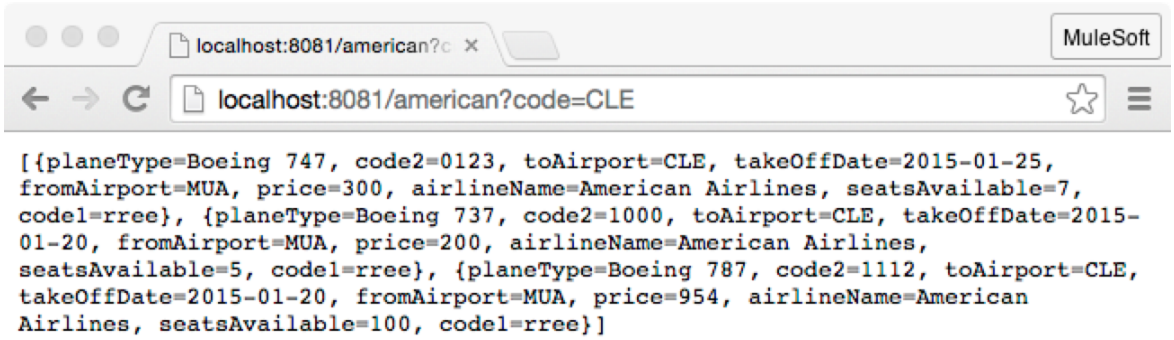
34. In the Properties view for the Database connector endpoint, modify the query to use the destination flow variable.

```
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.



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

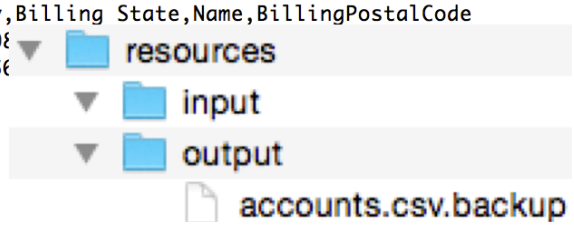


Walkthrough 4-2: Connect to a file (CSV)

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

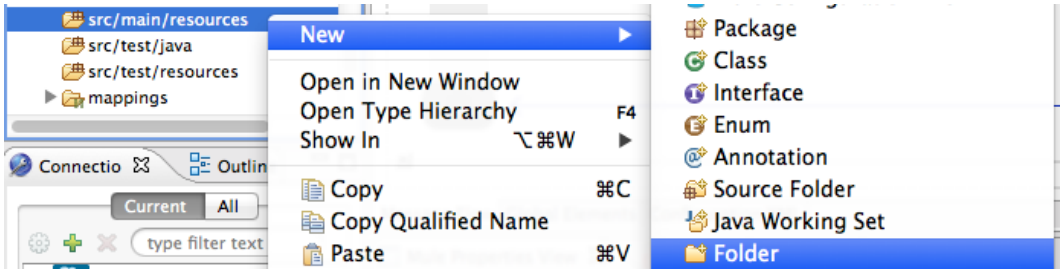
- Add and configure a File connector endpoint to watch an input directory, read the contents of any added files, and then rename the files and move them to an output folder.
- Add a CSV file to the input directory and watch it renamed and moved.
- Restrict the type of file read.
- Use the File to String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

```
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
```

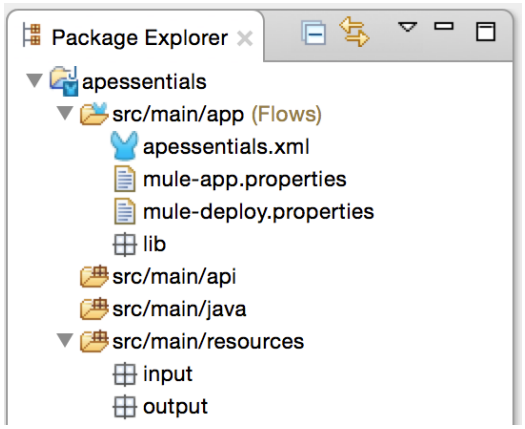


Create new directories

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

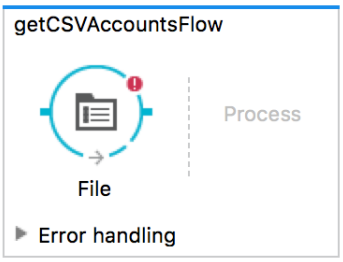


3. Set the folder name to input and click Finish.
4. Create a second folder called output.

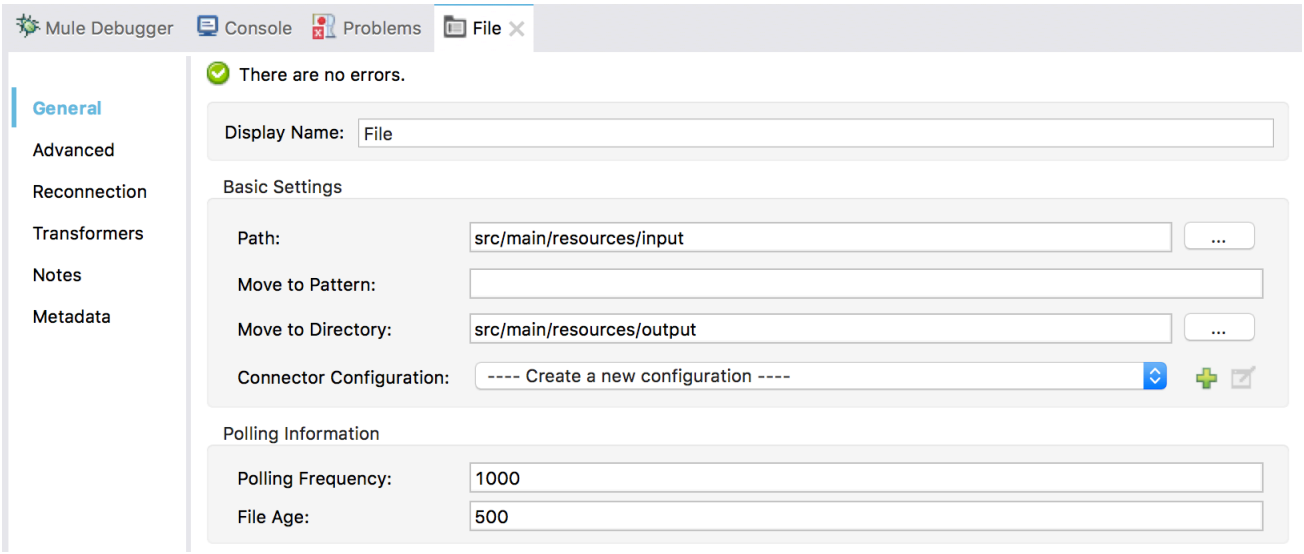


Add and configure a File connector endpoint

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.

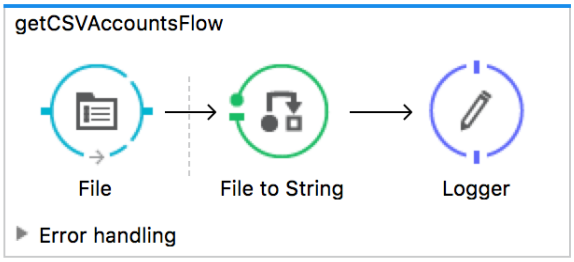


9. Look at the default polling information; the endpoint checks for incoming messages every 1000 milliseconds and sets a minimum of 500 milliseconds a file must wait before it is processed.

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

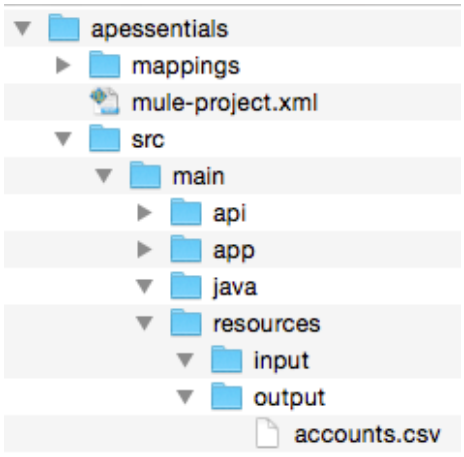
13. Save the file to redeploy the application.
14. Leave the application running.

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: APEssentials3.8_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.

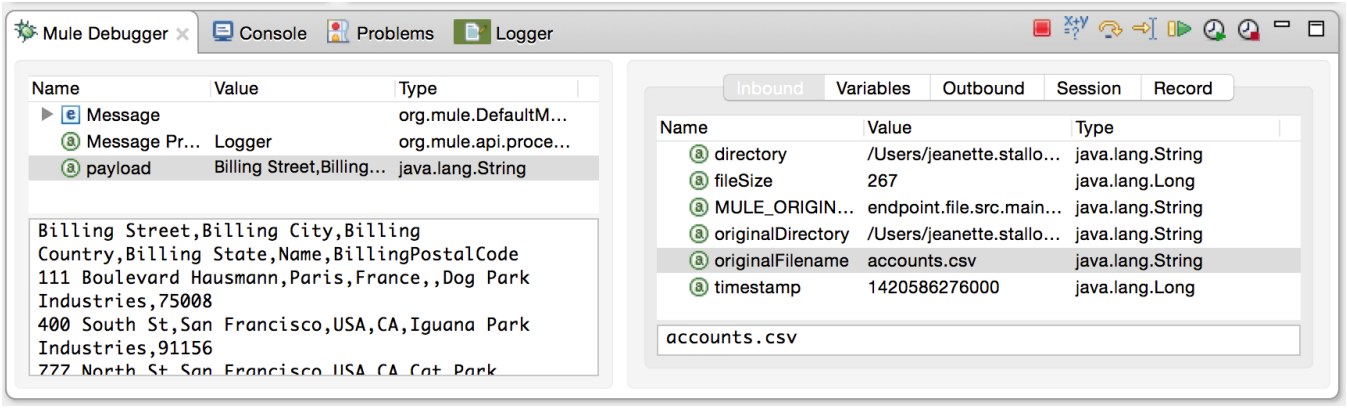
21. Leave this folder open.

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.

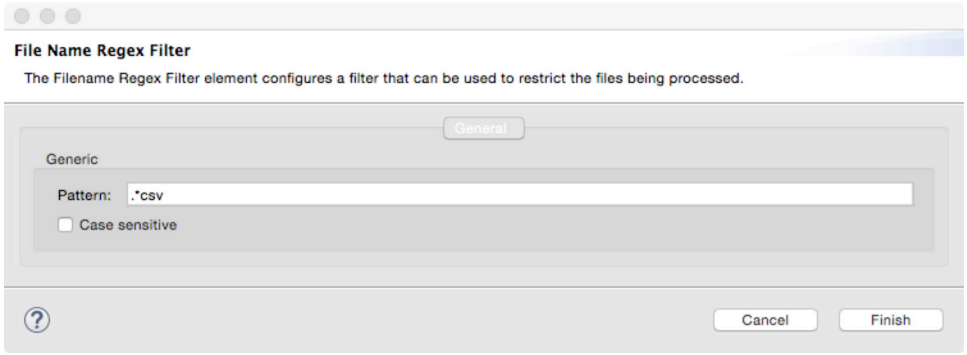


29. Click the Resume button in the Mule Debugger view.
30. Stop the debugger.

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.

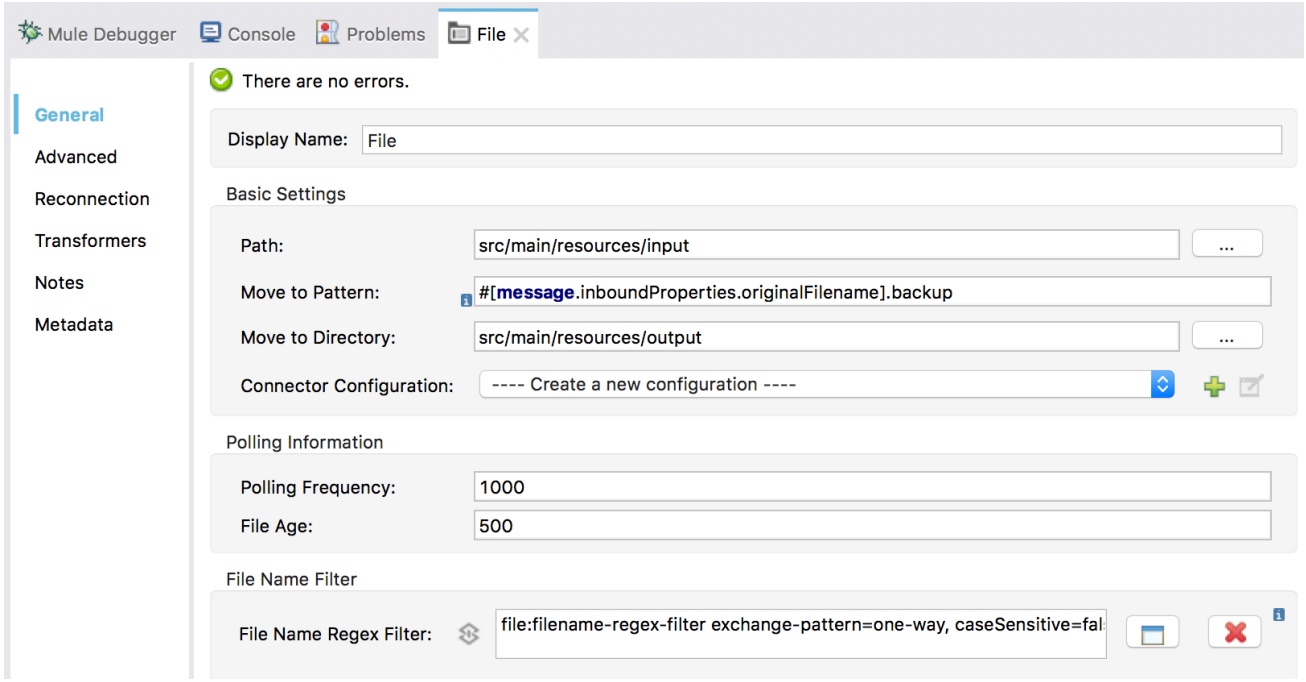


34. Click Finish.

Rename the file

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

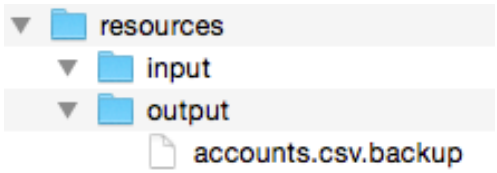
```
#[message.inboundProperties.originalFilename].backup
```



Test the application

36. Save the file and run the application.

37. 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.



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

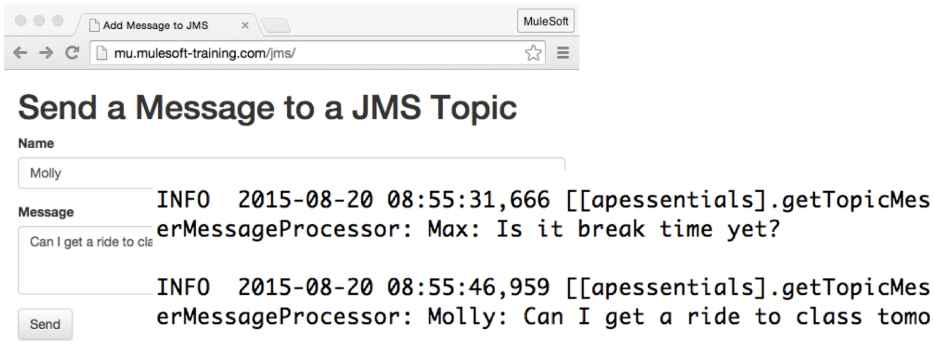
```
INFO  2015-08-19 13:44:16,688 [[apessentials].getCSVAccountsFlow.stage1.02] org.mule.api.processor.Logger
MessageProcessor: Billing Street,Billing City,Billing Country,Billing State,Name,BillingPostalCode
111 Boulevard Hausmann,Paris,France,,Dog Park Industries,75008
400 South St,San Francisco,USA,CA,Iguana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

39. 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)

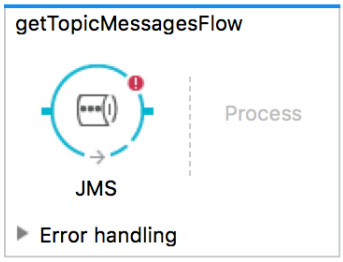
In this walkthrough, you will read and write messages from a JMS topic. You will:

- 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.



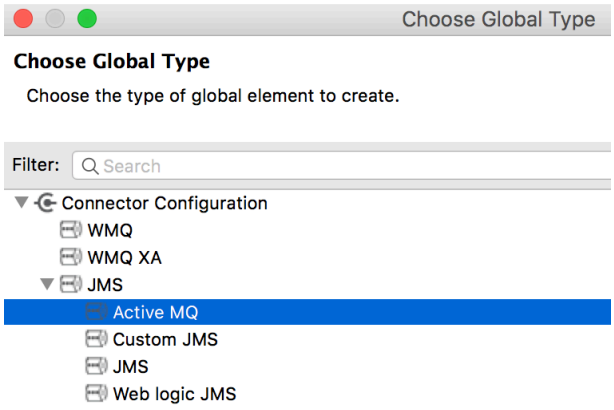
Create a JMS inbound-endpoint

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.

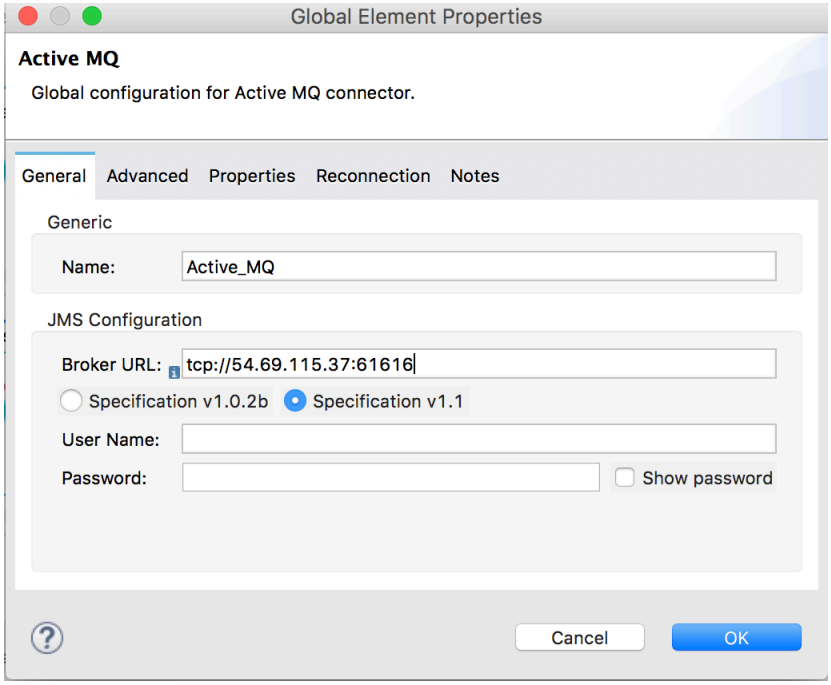


4. In the JMS Properties view, select topic and set it to apessentials.
5. Click the Add button next to connector configuration.

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



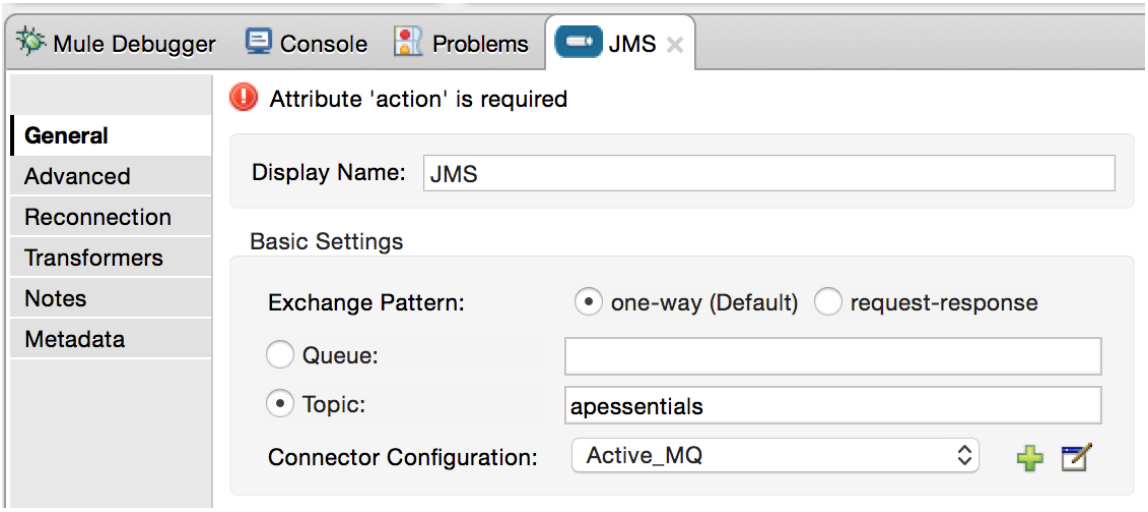
7. In the Global Element Properties dialog box, change the broker URL to the JMS ActiveMQ Broker URL listed in the course snippets.txt file.
8. Set the Specification to v1.1.



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.

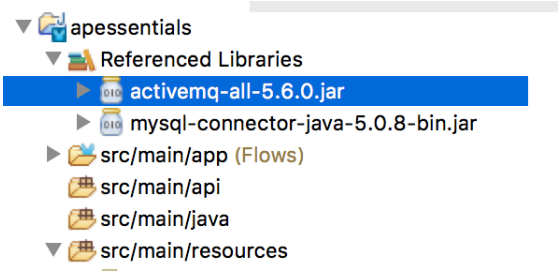
12. If you see an Attribute ‘action’ is required warning, ignore it.

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



Add the ActiveMQ library

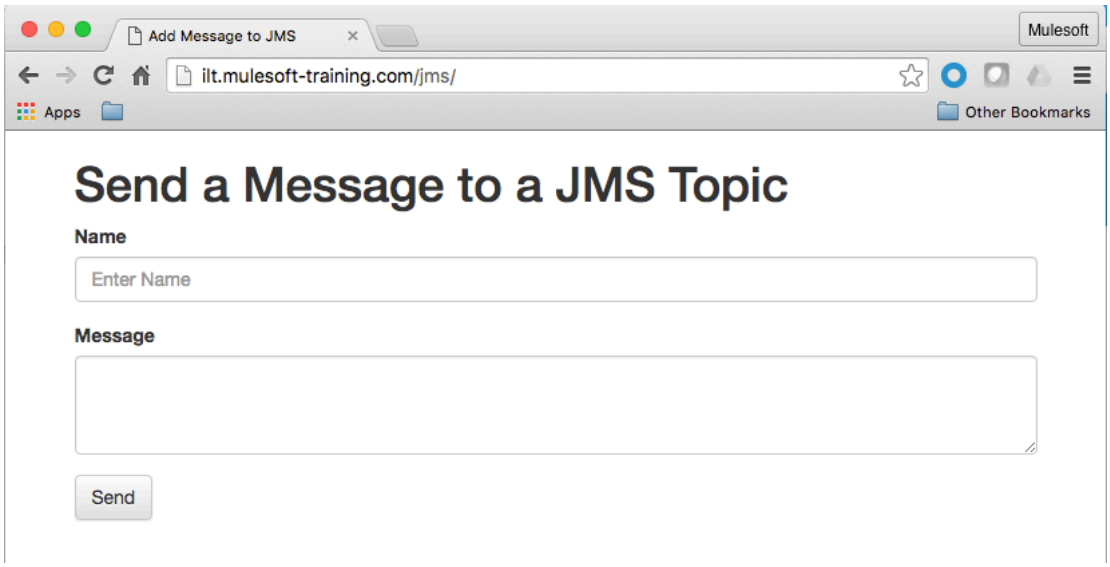
13. In the Package Explorer, right-click apessentials and select New > Folder.
14. In the New Folder dialog box, set the folder name to lib and click Finish.
15. Locate activemq-all-{version}.jar file located in the APEssentials3.8_studentFiles_{date}/jars folder.
16. Copy and paste or drag the JAR file 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.



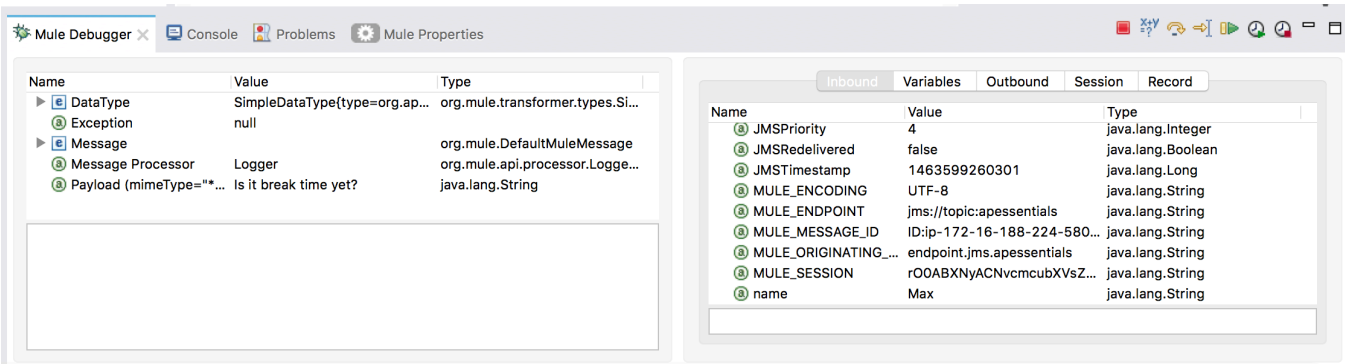
22. Return to the console in Anypoint Studio; you should see your message along with those from your classmates – but you don't see the names of the people who sent the messages.

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

INFO  2015-08-20 08:52:57,613 [[apessentials].getTopicMessagesFlow.stage1.02] org.mule.api.processor.Logg
erMessageProcessor: 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.



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

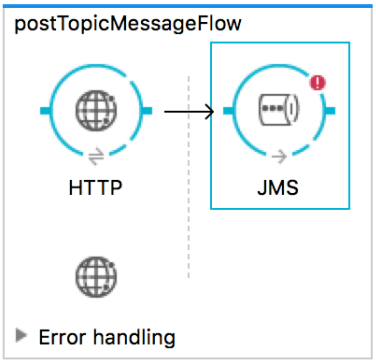
29. Save the file and run the application.
30. Return to the message form and submit a new message.
31. Look at the console; you should now see names along with messages.

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

INFO  2015-08-20 08:55:46,959 [[apessentials].getTopicMessagesFlow.stage1.02] org.mule.api.processor.Logg
erMessageProcessor: 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.

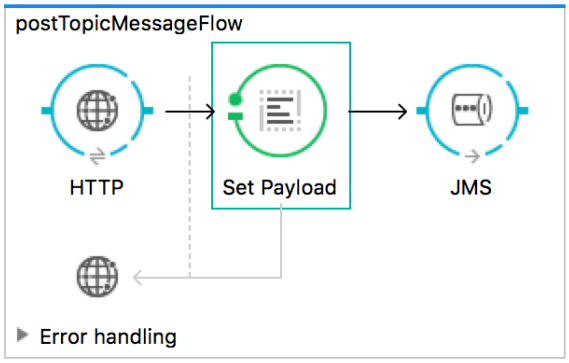
36. Drag out another JMS connector and drop it into the process section of the flow.



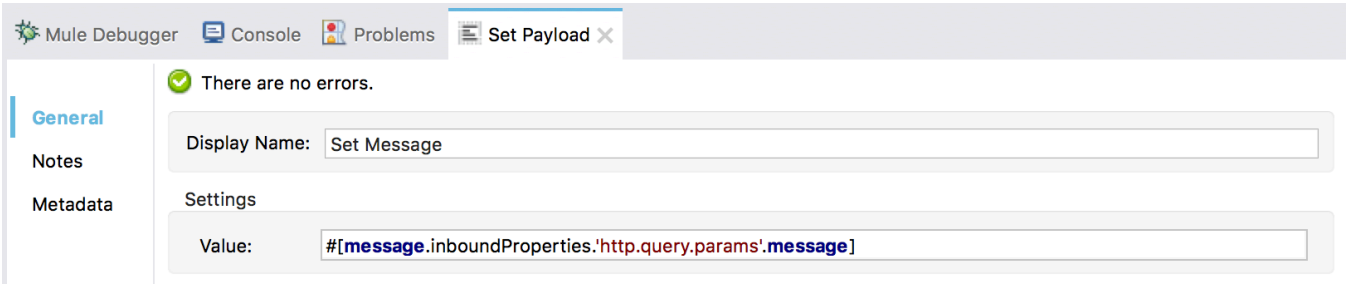
37. In the JMS Properties view, select topic and set it to apessentials.
38. Set the connector configuration to the existing Active_MQ.
39. If you see an Attribute ‘action’ is required warning, ignore it.

Set a message

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

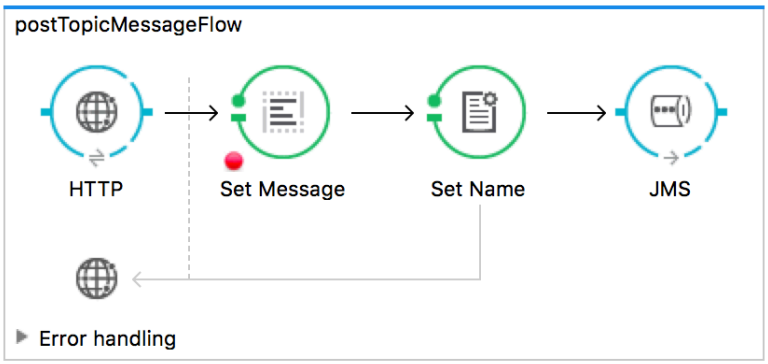


41. In the Set Payload Properties view, change the display name to Set Message and set the value to a message query parameter.



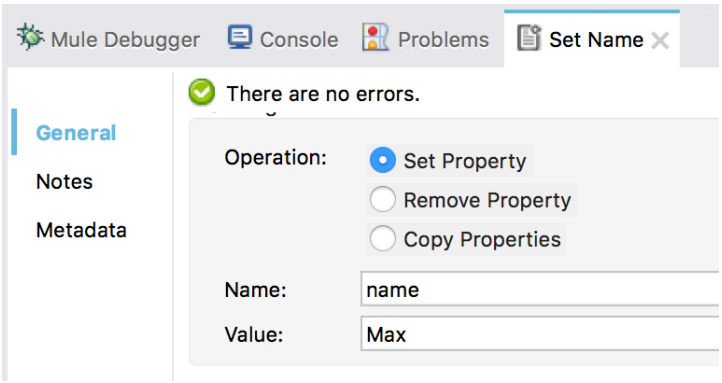
42. Add a breakpoint to the Set Payload transformer.
43. Add a Property transformer after the Set Payload transformer.

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

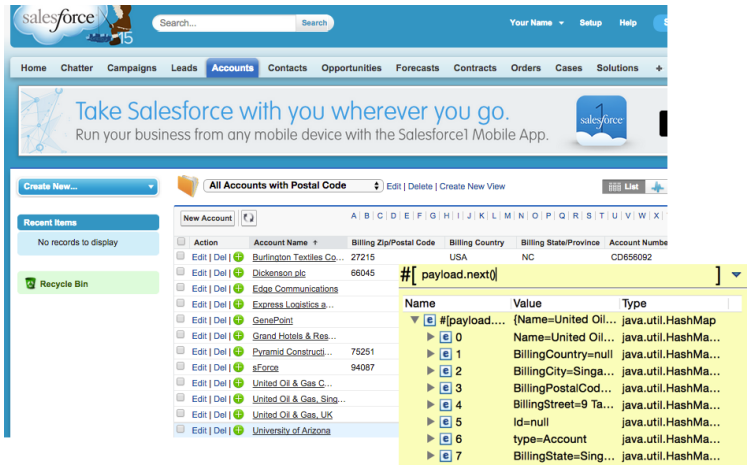
46. Save the file to redeploy the application and make a request to http://localhost:8081/jms?message=Hello.
47. Look at the console; you should see your name and message displayed – along with those of your classmates.

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

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

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

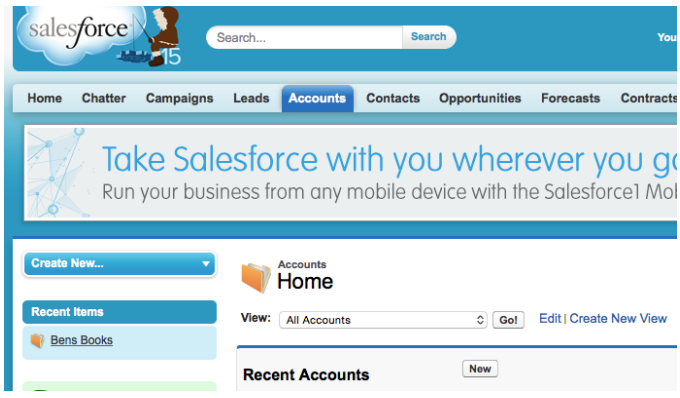
- Browse Salesforce data on http://salesforce.com.
- Create a flow accessible at http://localhost:8081/sfdc.
- Add a Salesforce connector endpoint to retrieve accounts for a specific postal code.
- Use the Query Builder to write a query.
- Use the Object to JSON transformer to return the results as a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



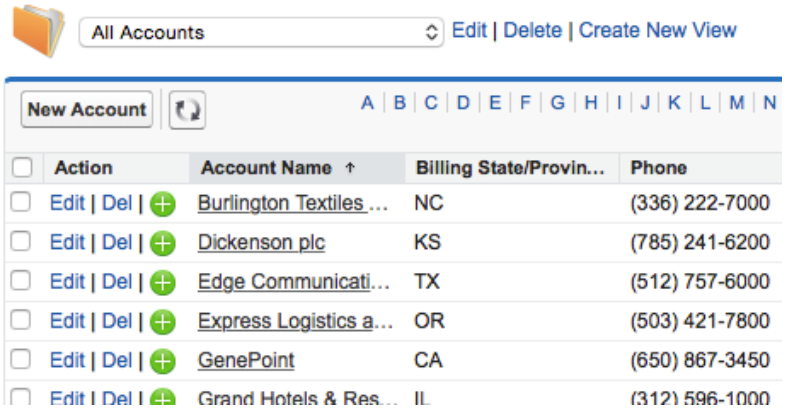
Note: To complete this walkthrough, you need a Salesforce Developer account. Instructions for creating a Salesforce developer account and getting an API access token are included in the Setup instructions at the beginning of this student manual.

Look at existing Salesforce account data

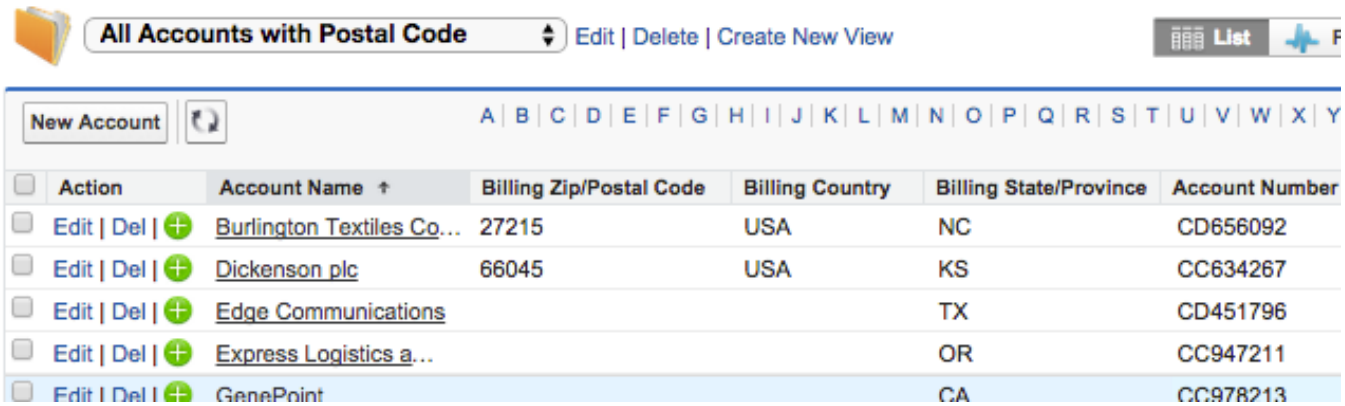
1. In a browser, go to http://login.salesforce.com/ and log in with your Salesforce Developer account.
2. Click the Accounts link in the main menu bar.
3. In the view drop-down, select All Accounts and click the Go button.



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



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

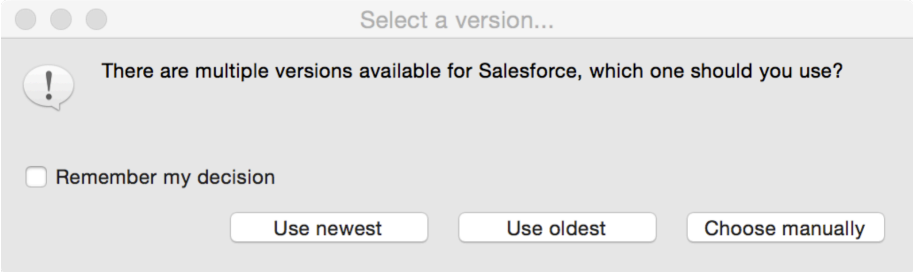


Create a new project and flow

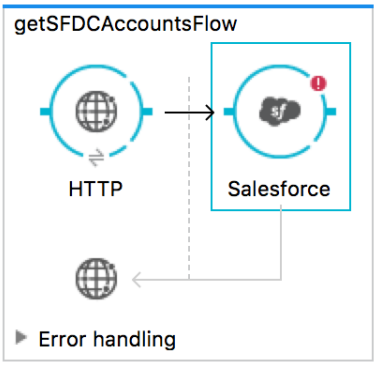
13. Return to apessentials.xml in Anypoint Studio.
14. Drag out an HTTP connector and drop it above the getCSVAccountsFlow to create a new flow.
15. Give the flow a new name of getSFDCAccountsFlow.
16. In the HTTP Properties view, set the connector configuration to the existing HTTP_Listener_Configuration.
17. 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.

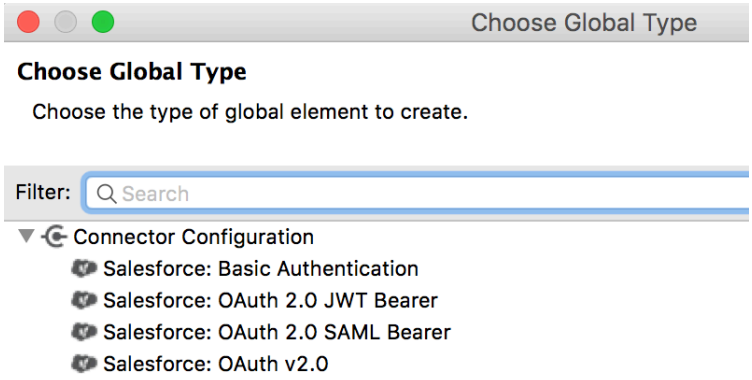


20. Examine the flow.



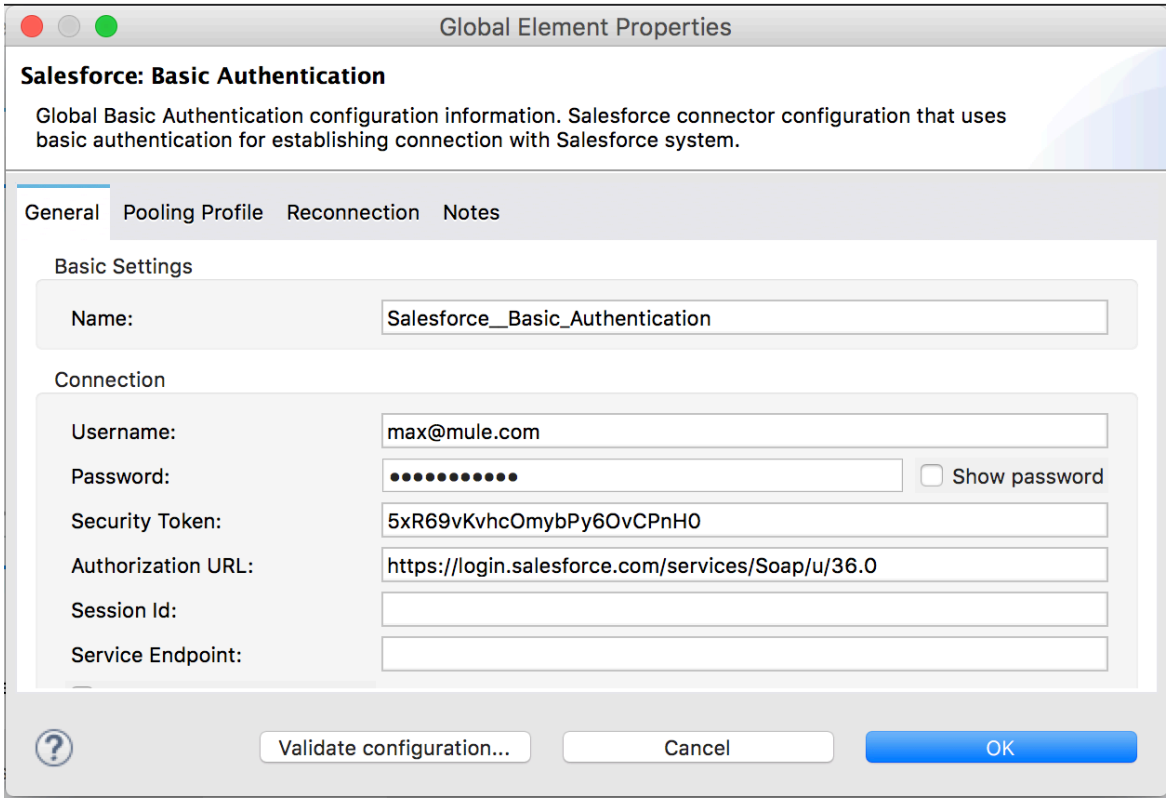
Configure the Salesforce connector

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.

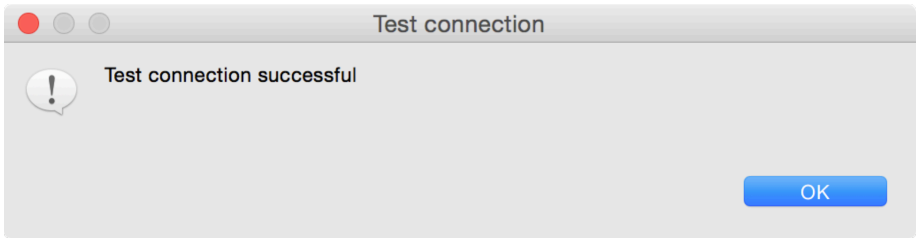


23. In the Global Element Properties dialog box, enter your email username, password, and security token.

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



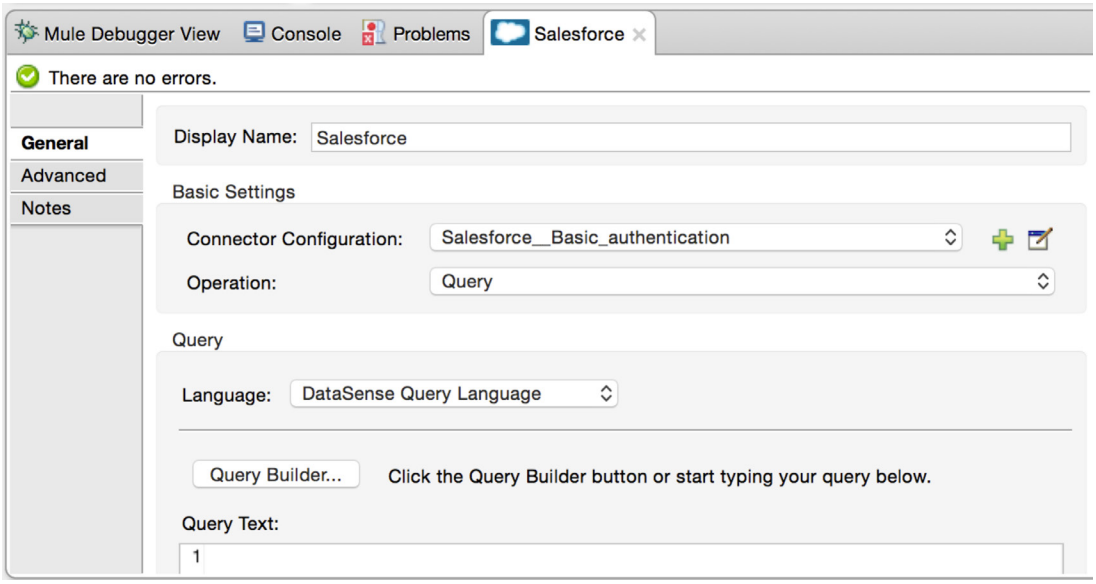
25. Click OK to close the Test connection dialog box.
26. If your test connection failed, fix it; do not proceed until it is working.

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

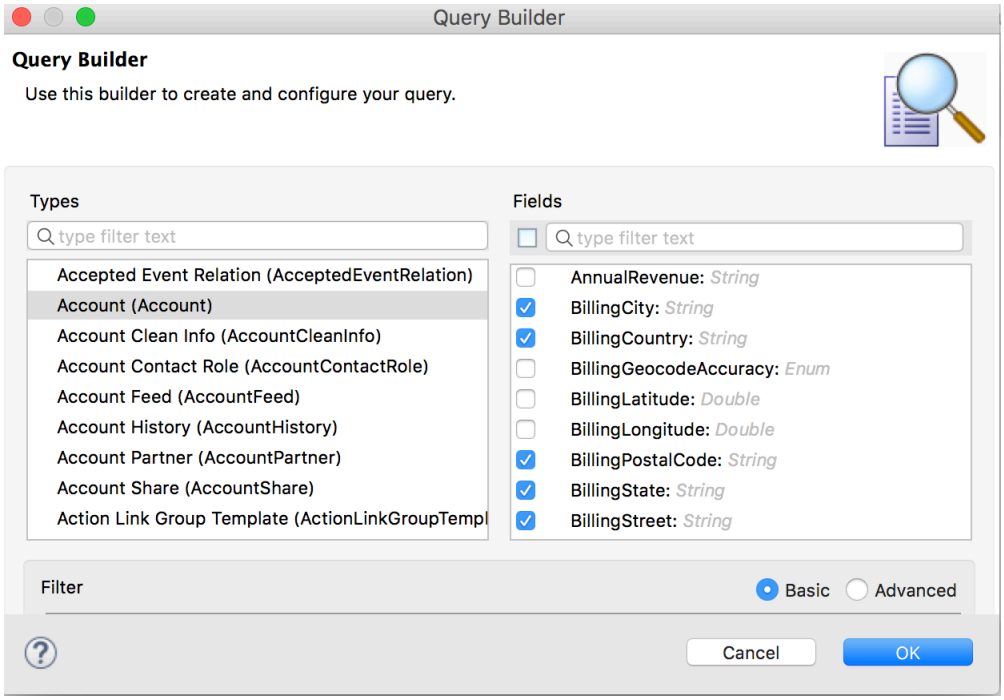
27. Click OK to close the Global Elements Properties dialog box.

Write the query

28. In the Salesforce Properties view, select an operation of Query.
29. Click the Query Builder button.

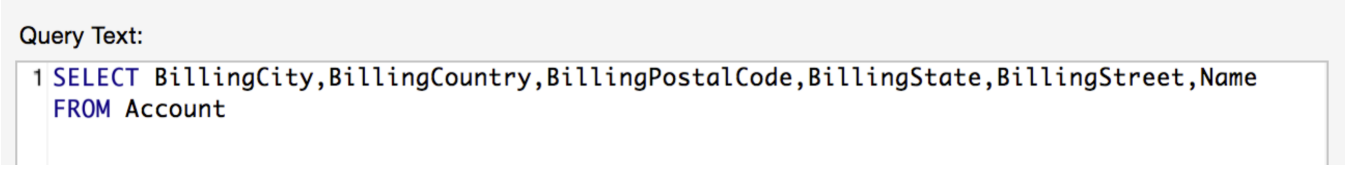


30. In the Query Builder dialog box, select a type of Account (Account).
31. Select fields BillingCity, BillingCountry, BillingPostalCode, BillingState, BillingStreet, and Name.



32. Click OK.

33. Examine the generated query.



Test the application

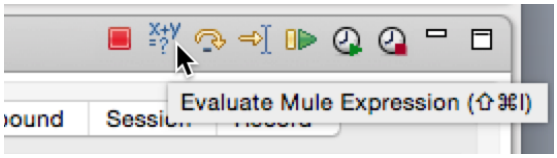
34. Save the file to redeploy the application.

    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

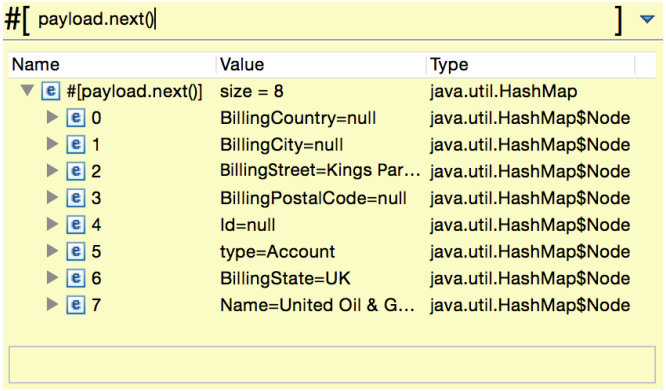
36. Add a Logger component after the Salesforce endpoint.
37. Set the Logger message to #[payload].
38. Add a breakpoint to the Logger.
39. Save the file and debug the application.
40. Make another request to http://localhost:8081/sfdc.
41. Step to the Logger and drill-down into the payload variable.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



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

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

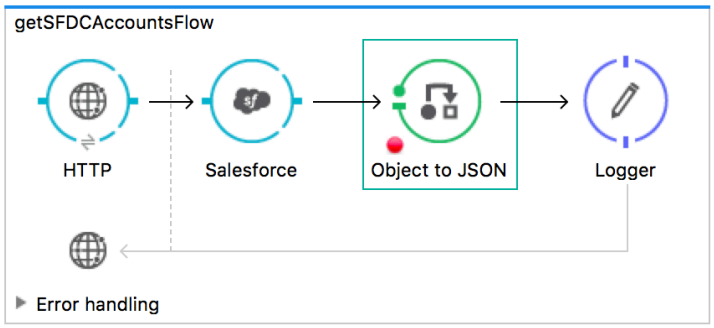
44. Expand the results; you should see the account data for the first account.



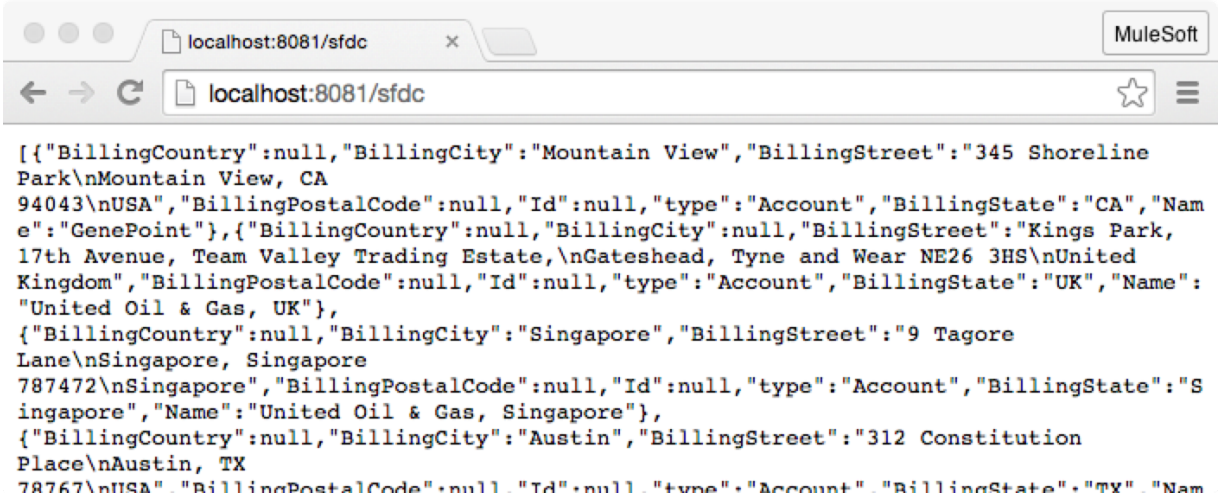
45. Click anywhere outside the expression window to close it.
46. Stop the Mule Debugger.

Display the return data

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

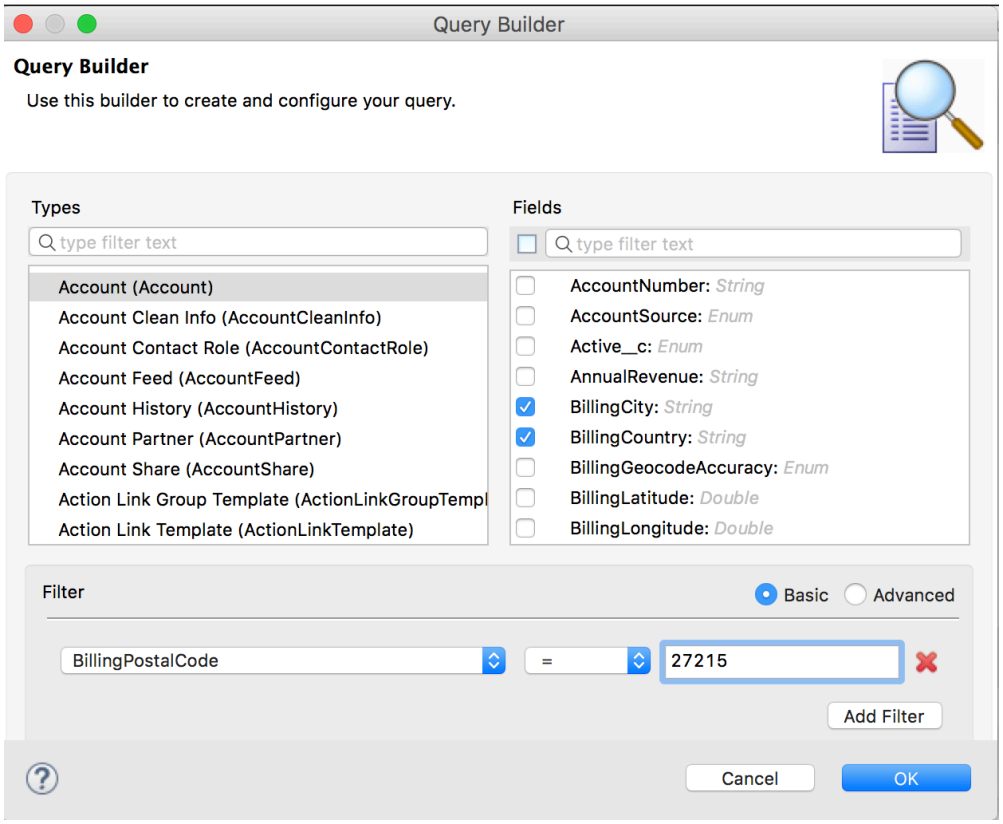


48. Save the file and run the application.
49. Make another request to http://localhost:8081/sfdc; you should see the Salesforce accounts displayed.



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.



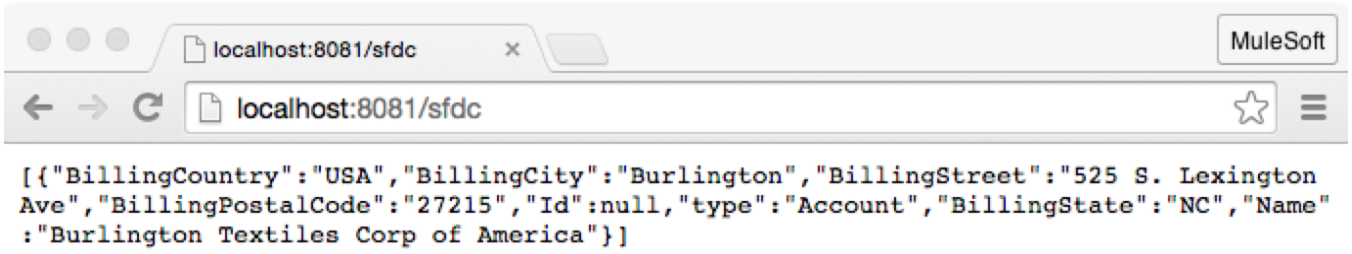
53. Click OK.
54. Examine the generated query.

Query Text:

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

Test the application

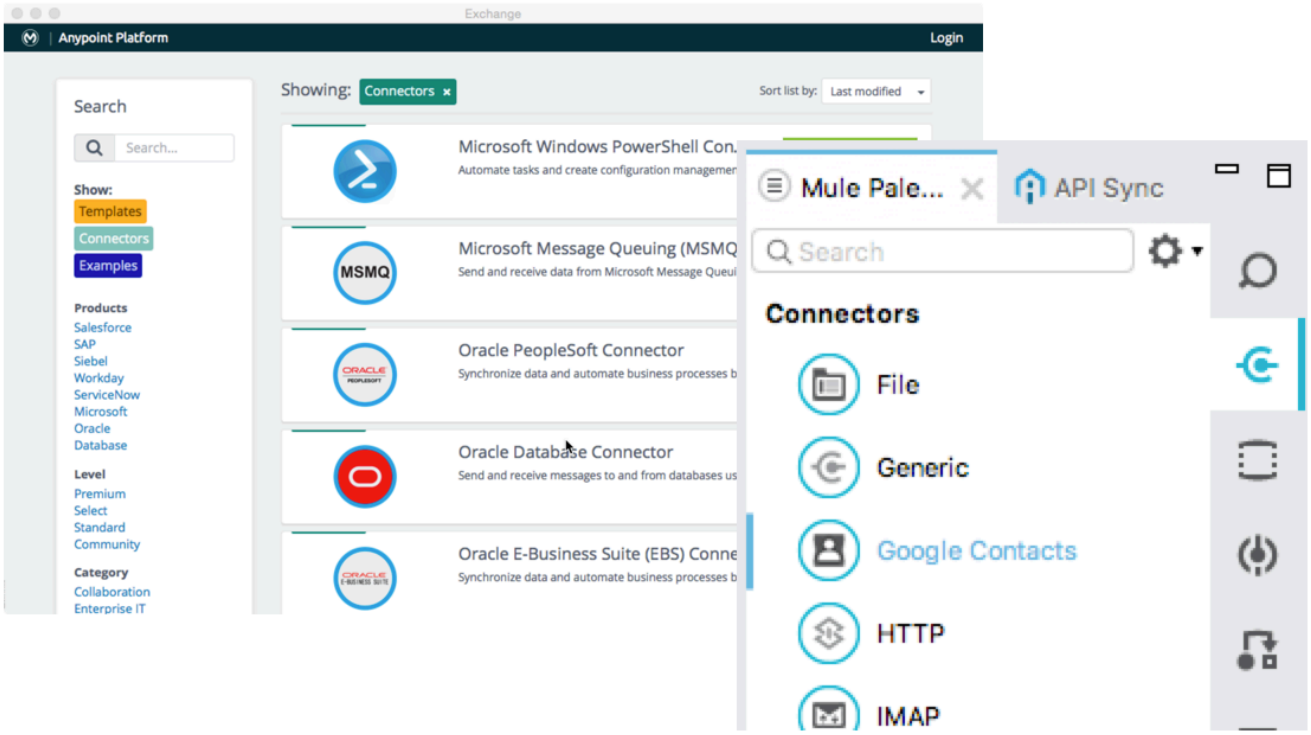
55. Save the file to redeploy the application.
56. Make another request to http://localhost:8081/sfdc; you should only the accounts with the specified postal code displayed.



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

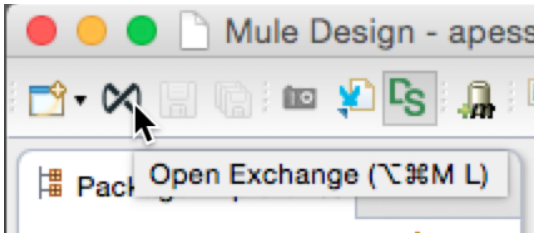
In this walkthrough, you will learn how to add a new connector to Anypoint Studio. You will:

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

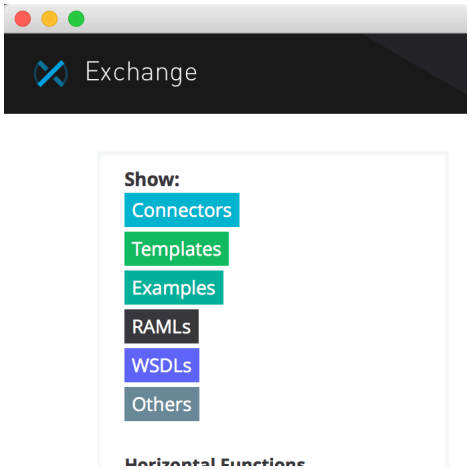


Browse the Anypoint Exchange from Anypoint Studio

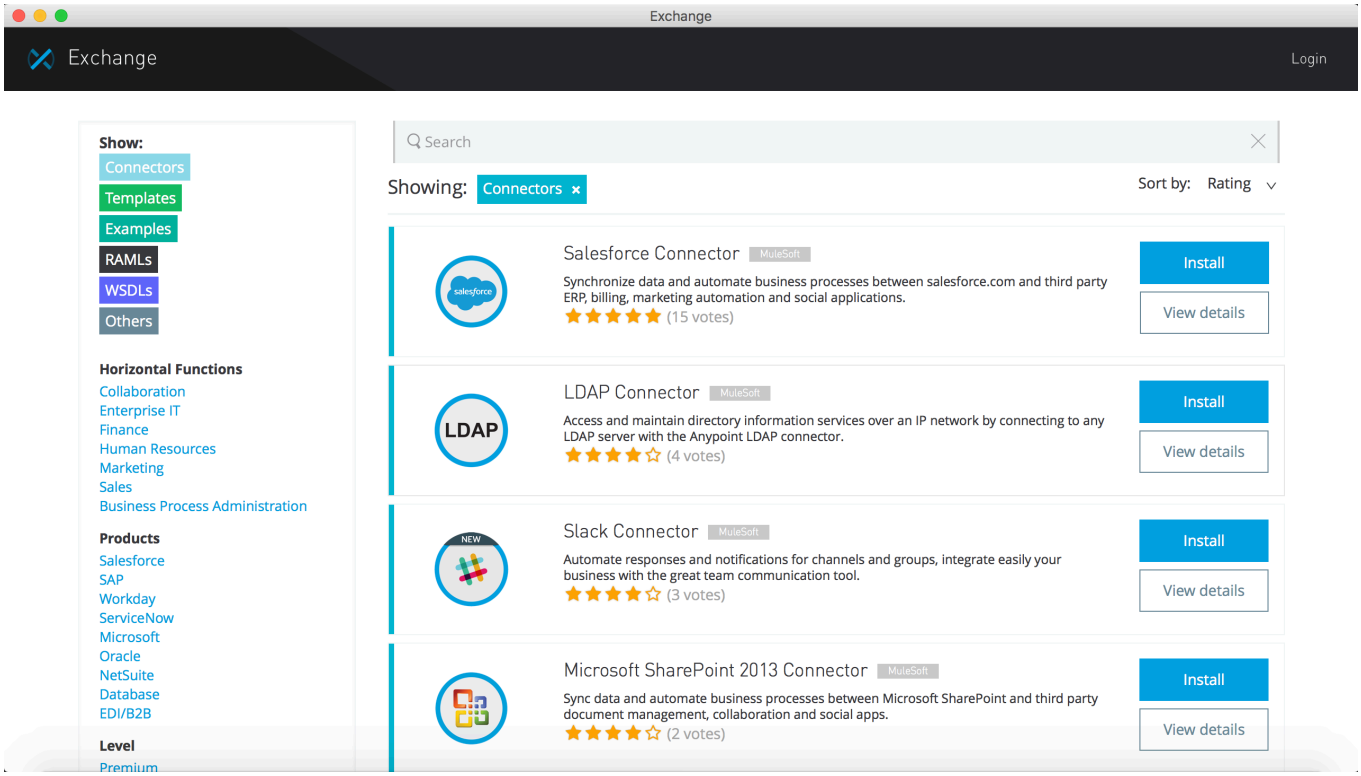
1. In Anypoint Studio, click the Open Exchange button; the Anypoint Exchange should open in a new window.



2. Click the Connectors button.

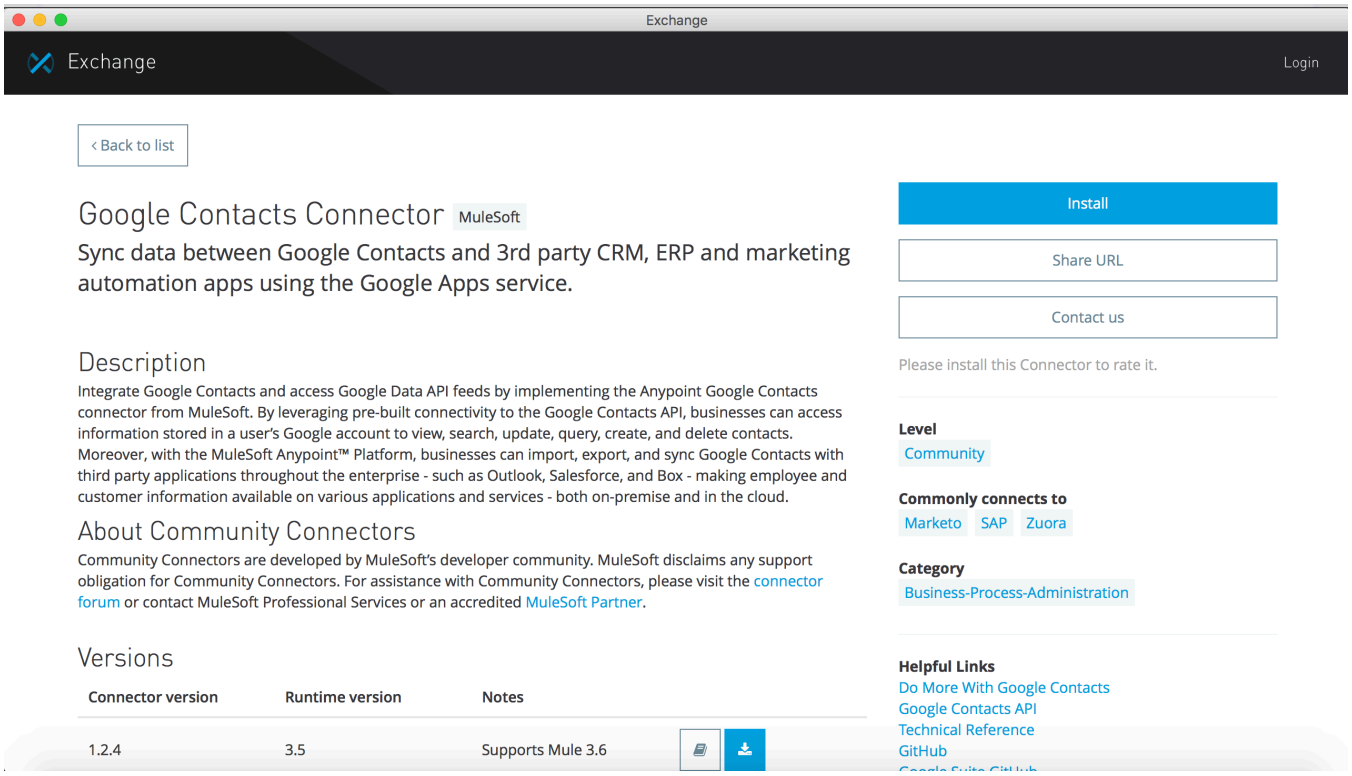


3. Browse the connectors, exploring the different levels and categories.



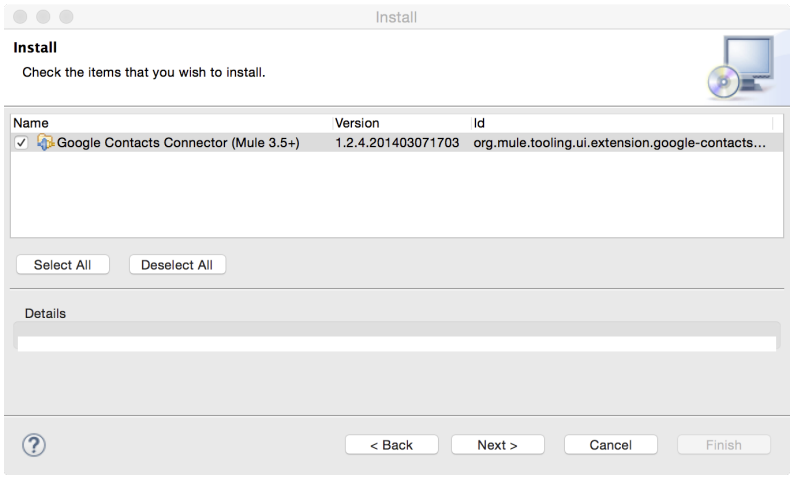
4. Locate the Google Contacts connector (or any other connector you are interested in).

5. Click the View details button and browse the connector's documentation.



Install the connector

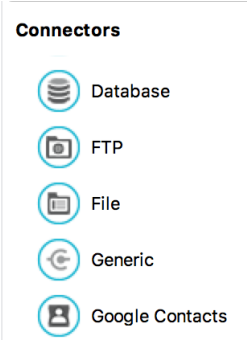
6. In the Anypoint Exchange, click the connector's Install button.
7. If you get a Terms of Service dialog box, enter your personal information and click Install.
8. Wait until an Install dialog box appears in Anypoint Studio.



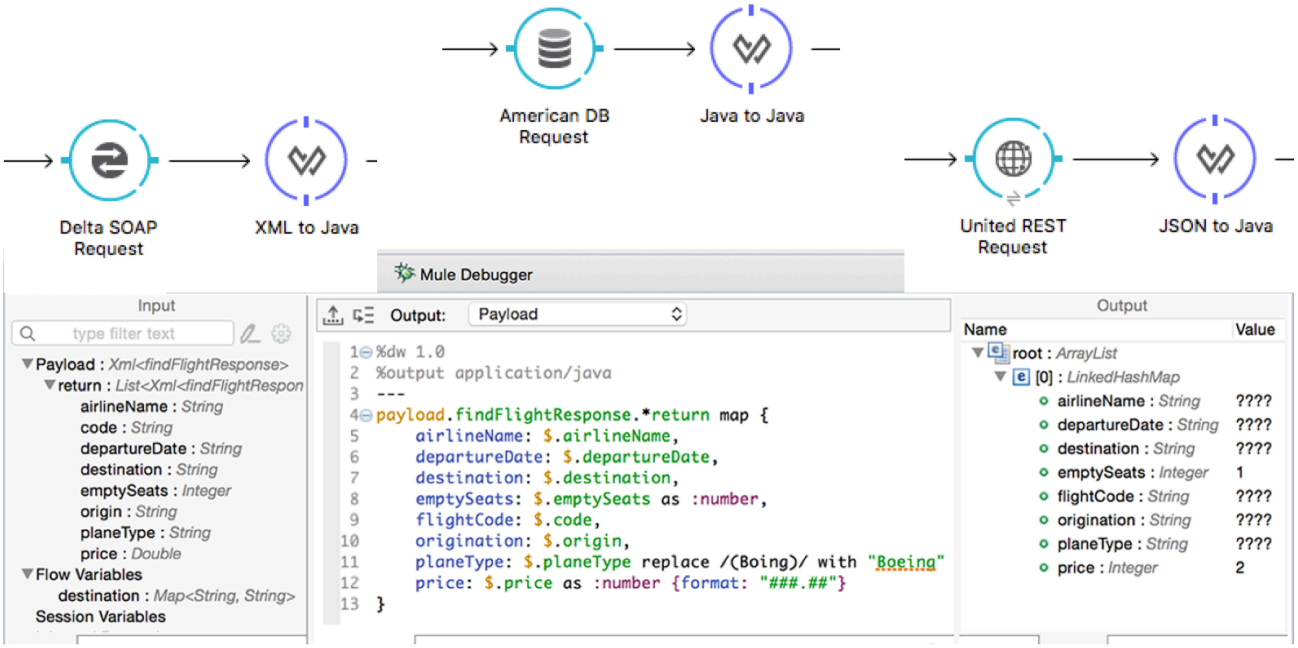
Note: If you do not see the connector listed, resize the Install dialog box.

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



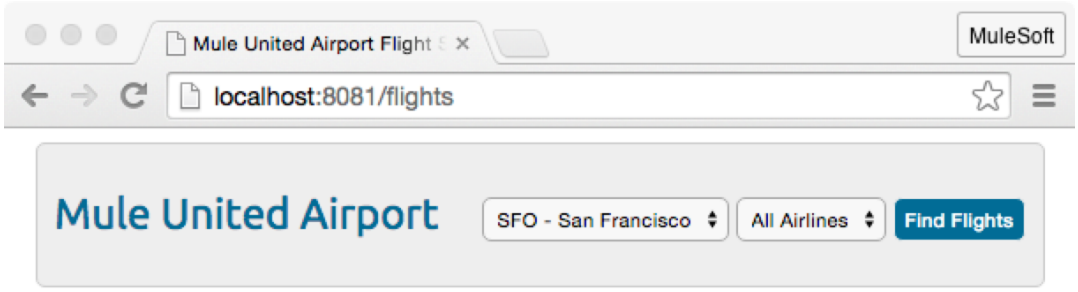
In this module, you will learn:

- 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

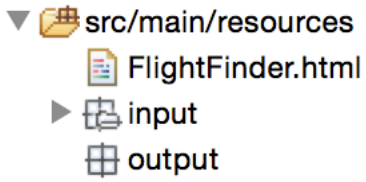
In this walkthrough, you will add an HTML form to your application that will eventually serve as a front end for the MUA integration application. You will:

- 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, APEssentials3.8_studentFiles_{date}.
2. Copy and paste (or drag) the resources/FlightFinder.html file into your project's src/main/resources folder.



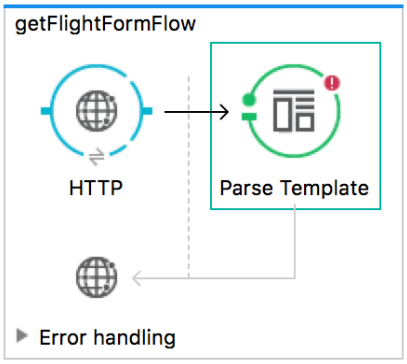
Note: You will examine the HTML code soon – after you load the form and see what it looks like.

Create a flow

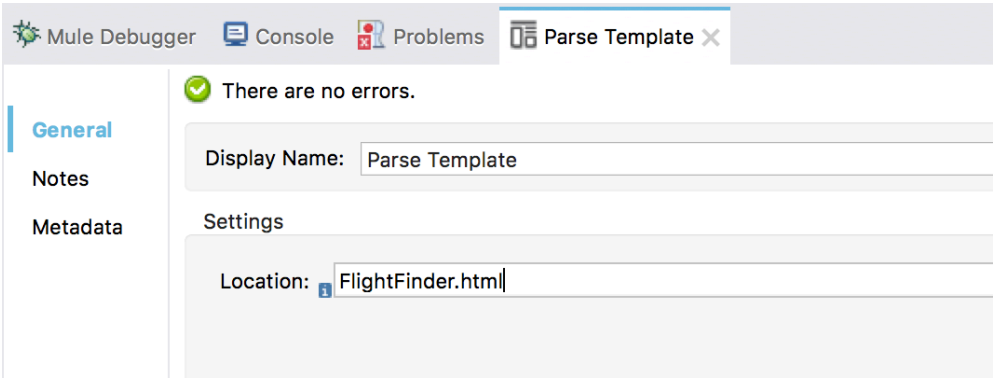
3. Return to apessentials.xml.
4. Drag out another HTTP connector to create a new flow at the top of the canvas and name it getFlightFormFlow.
5. In the HTTP Properties view, set the connector configuration to the existing HTTP_Listener_Configuration.
6. Set the path to /flights and set the allowed methods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

Set the payload to HTML and JavaScript

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



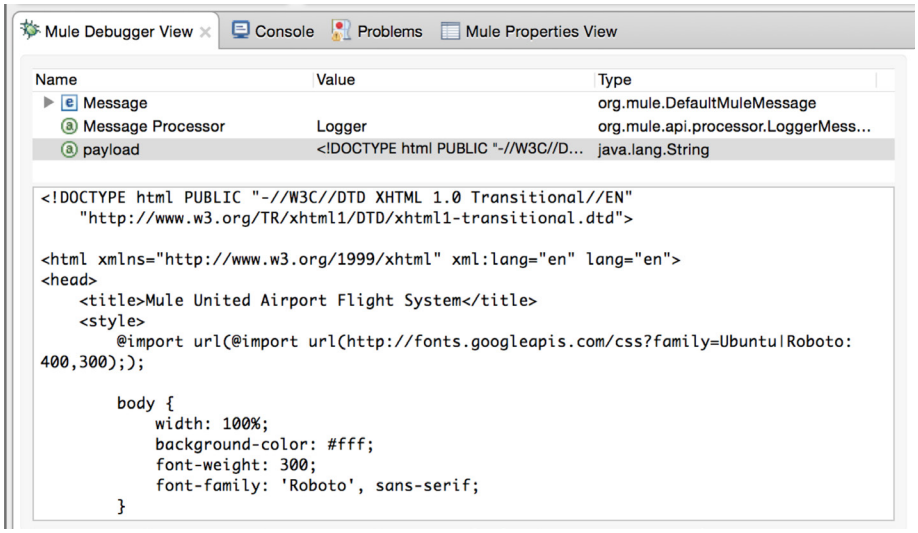
8. In the Properties view for the transformer, set the location to FlightFinder.html.



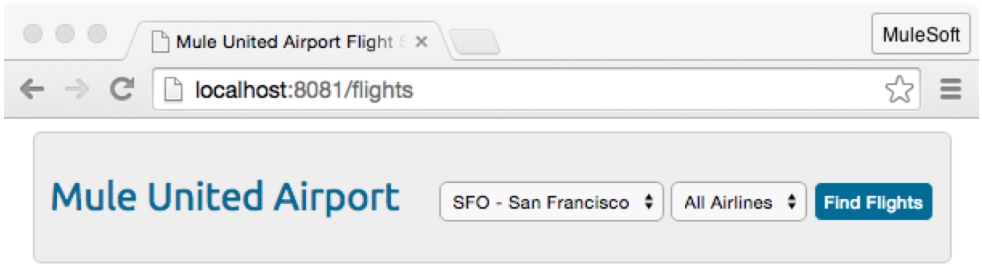
Debug the application

9. Add a breakpoint to the Parse Template transformer.
10. Add a Logger component after the transformer.
11. Add a breakpoint to the Logger.
12. Save the file and debug the application.
13. In a browser, make a request to http://localhost:8081/flights.

14. Step through the flow and watch the value of the payload change.



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



16. Click the Find Flights button; what happens?

Review the HTML form code

17. Open FlightFinder.html in Anypoint Studio.
18. 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>
```

19. 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);
```

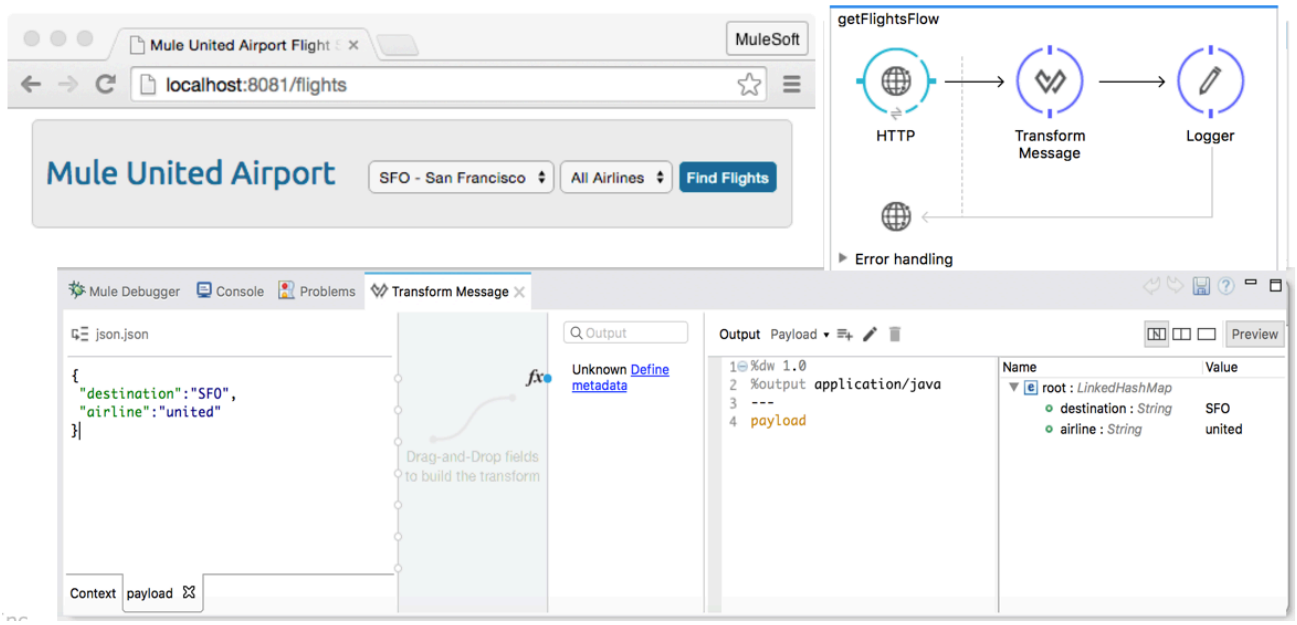
20. Locate in what format the data is that is posted.

```
 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.



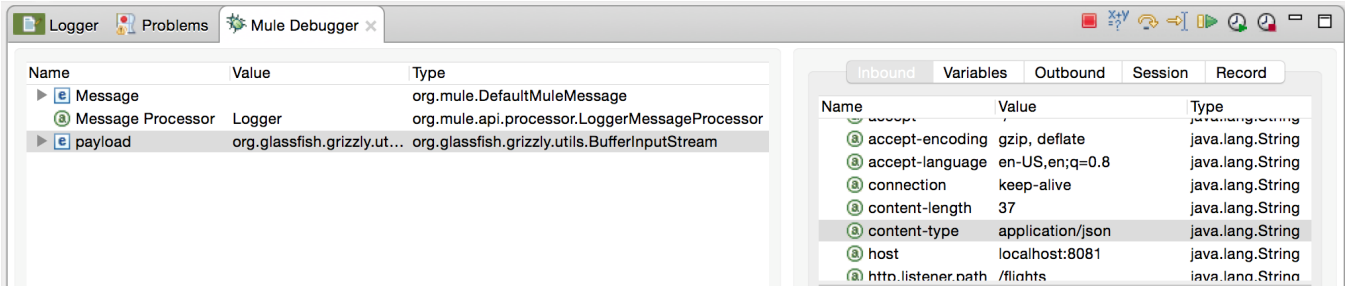
Create a new flow

1. Return to apessentials.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

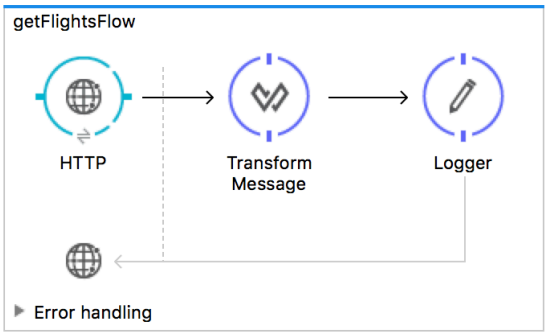
5. Add a Logger to the flow and add a breakpoint to it.
6. Save the file to redeploy the application in debug mode.
7. Make another request to http://localhost:8081/flights.
8. Step through or remove the breakpoints in the getFlightsFormFlow to get to the form in the browser window.

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.

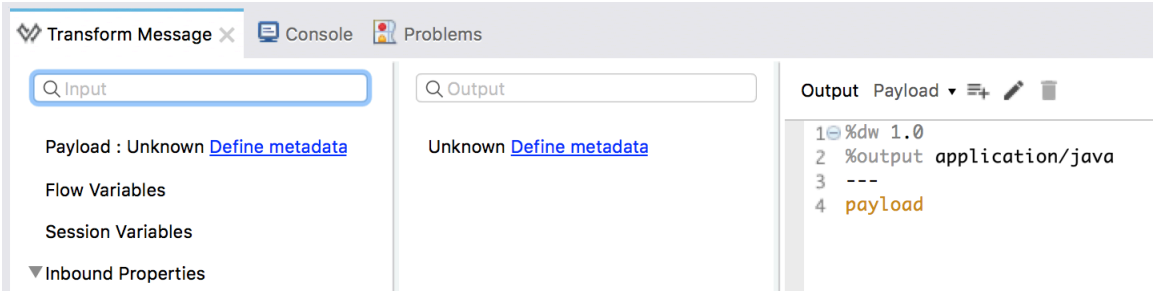


Add a DataWeave Transform Message component

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



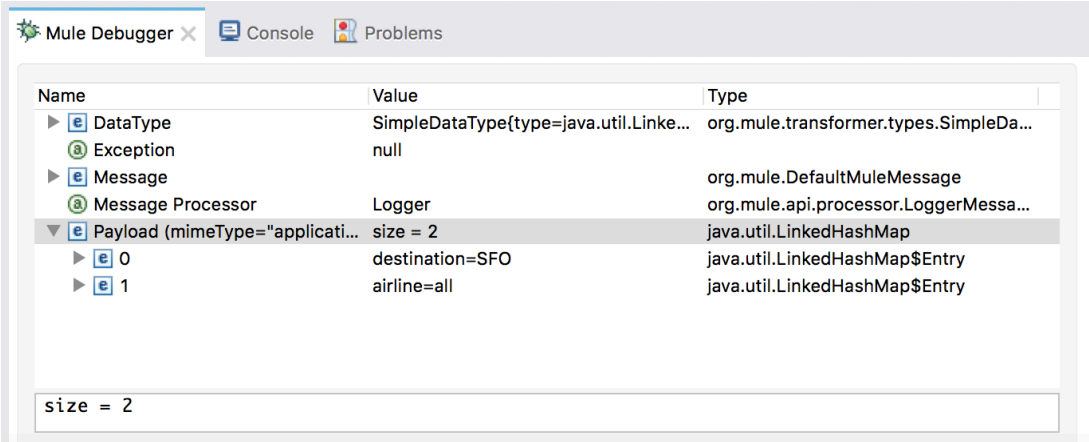
13. In the Transform Message Properties view, look at the Input, Transform, and Output sections.
14. In the Transform section, locate the drop-down menu at the top of the view that sets the output type; it should be set to Payload.
15. Beneath it, locate the directive that sets the output to application/java.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



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.

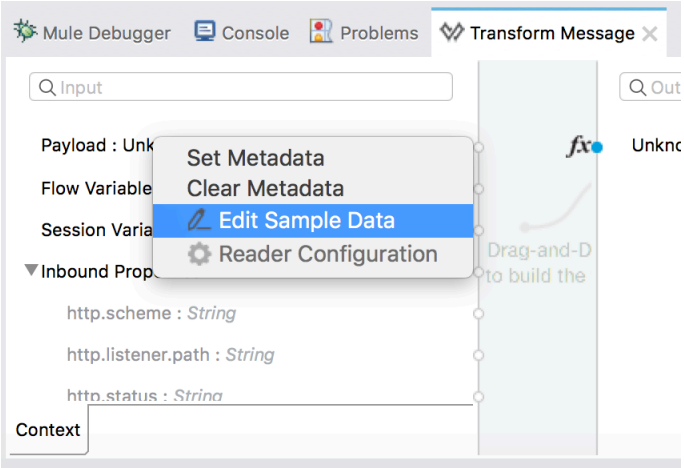
19. Make a request to http://localhost:8081/flights and submit the form again.
20. Step to the Logger in the getFlightsFlow; you should see the form data is now a LinkedHashMap.



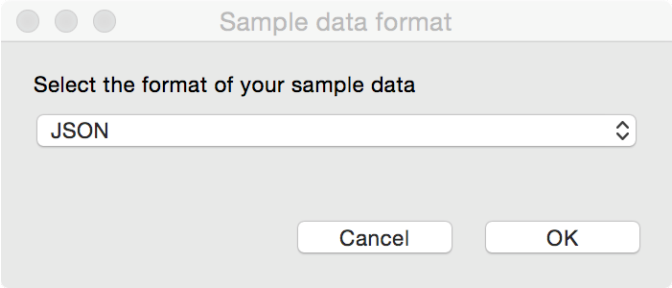
21. Click the Resume button.

Add sample data and preview

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



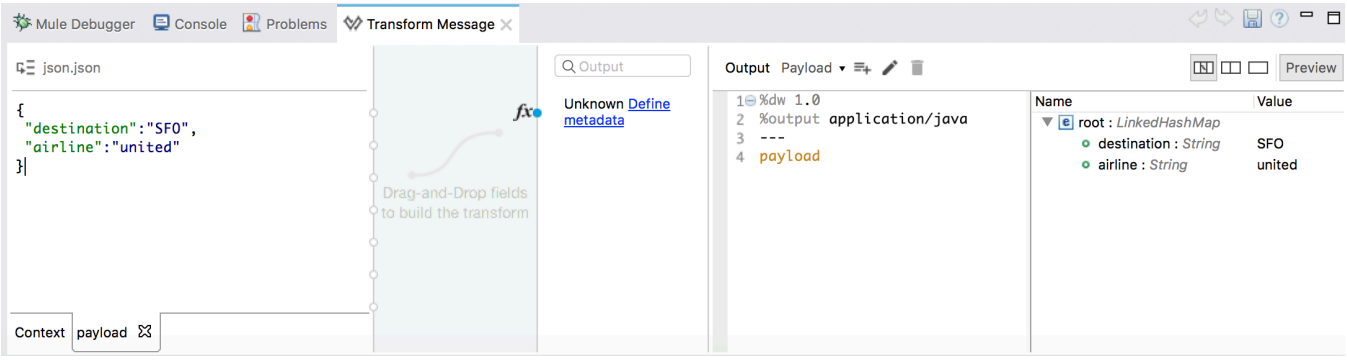
24. In the Sample data format dialog box, select JSON and click OK.



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

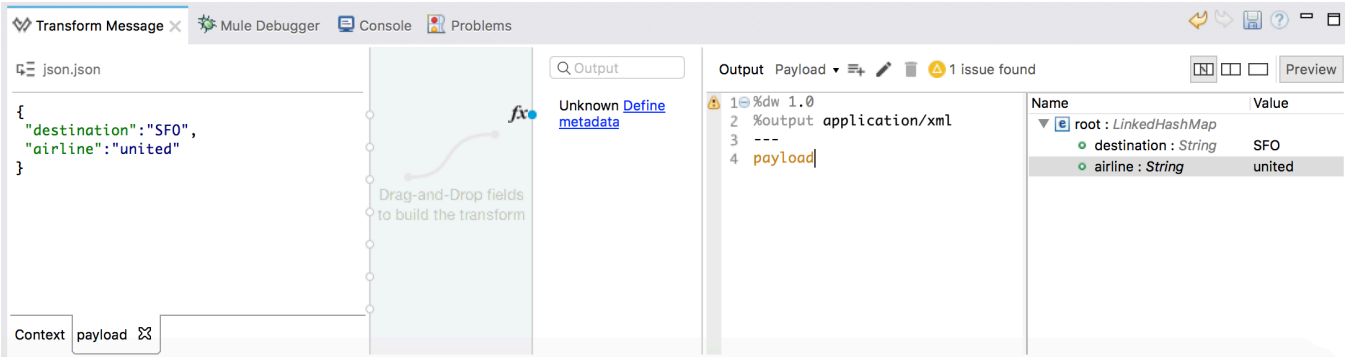
    Note: You can copy this value from the course snippets.txt file.

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

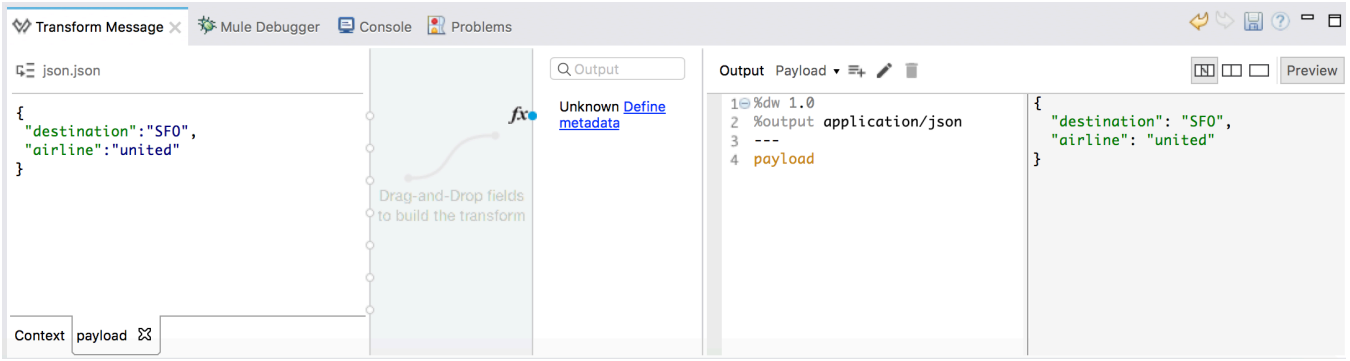


Change the output type

27. In the Transform section, change the output type from application/java to application/xml.
28. Look at the Output tab; you should get an error.

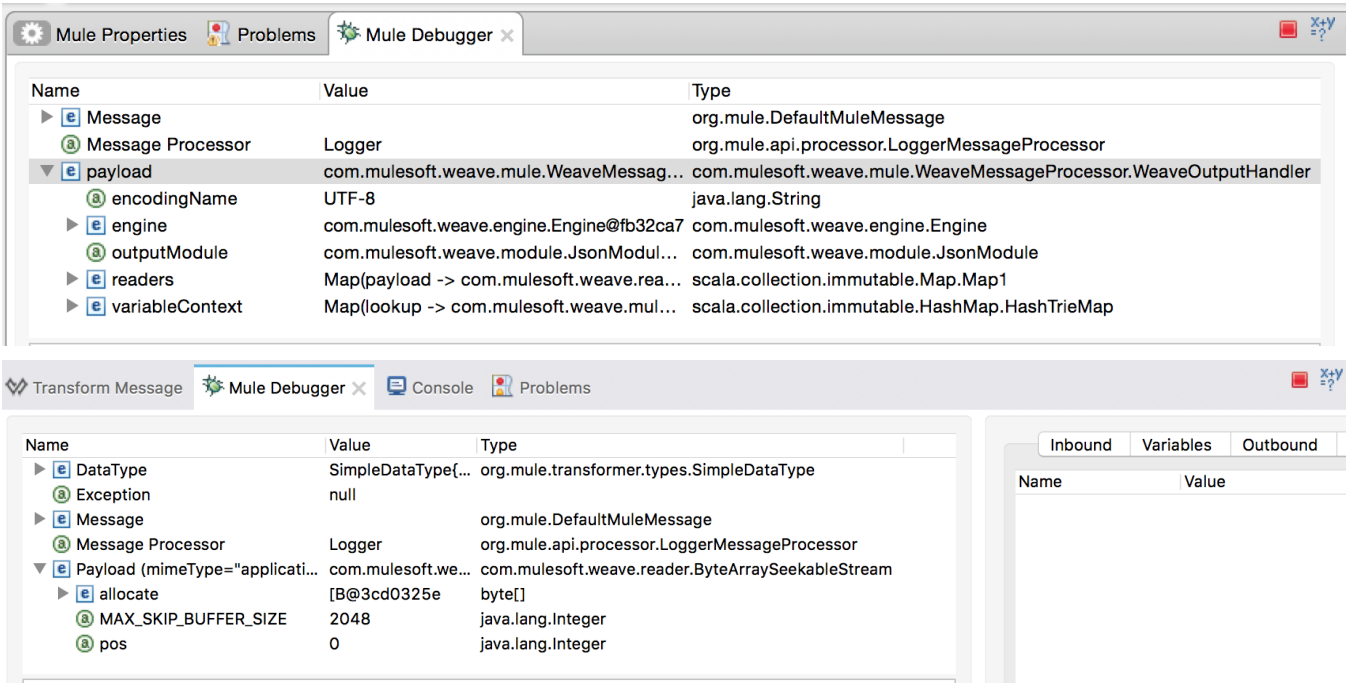


29. Change the output type to application/json; in the Preview tab, you should see sample JSON output.



Debug the application

30. Save the file to redeploy the application in debug mode.
31. Make a request to http://localhost:8081/flights and submit the form again.
32. Step to the Logger in the getFlightsFlow; you should see the form data is now of type WeaveOutputHandler.

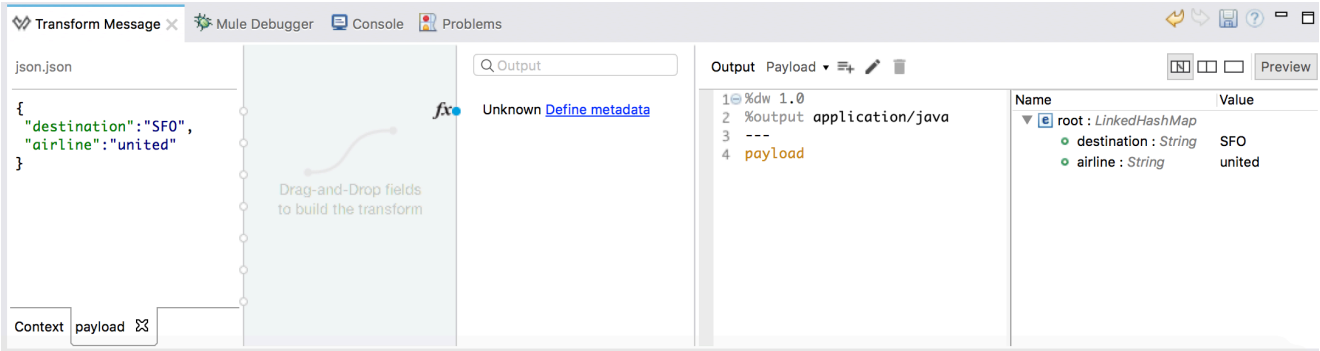


33. Click the Resume button.
34. Stop the Mule runtime.

Change the output type

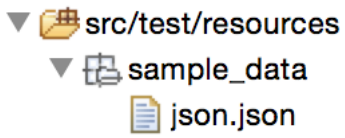
35. In the Transform section, change the output type from application/java.

36. Save the file.



Examine the code

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

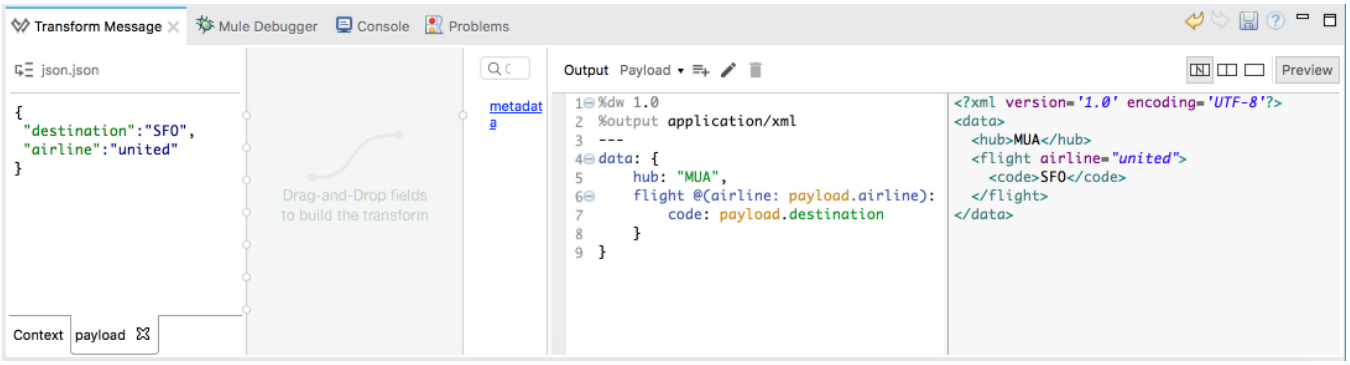


38. Switch to the Configuration XML view.
39. Locate the DataWeave code.
40. Switch back to the Message Flow view.

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

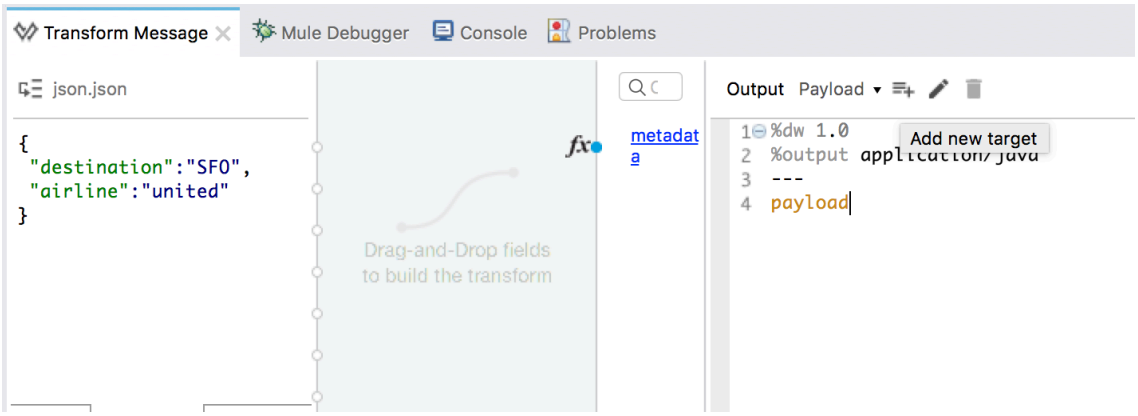
In this walkthrough, you will continue to work with the data posted from the HTML form. You will:

- 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.



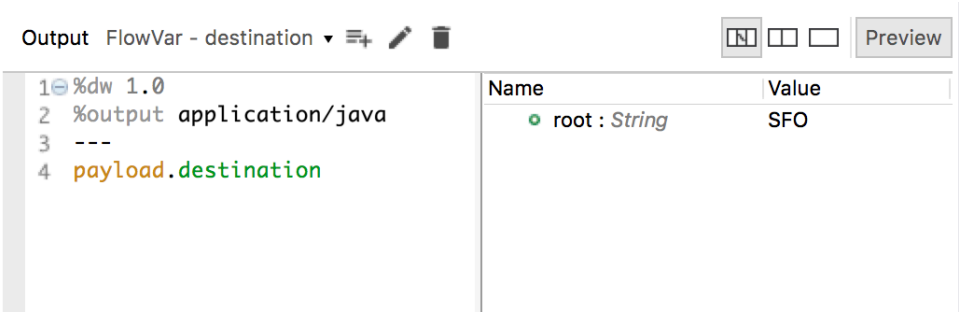
Create a second transformation to store destination in a flow variable

1. Return to getFlightsFlow in apessentials.xml.
2. Return to the Properties view for the Transform Message component.
3. In the Output section, click the Add new target button (right next to Payload) in the upper-right.



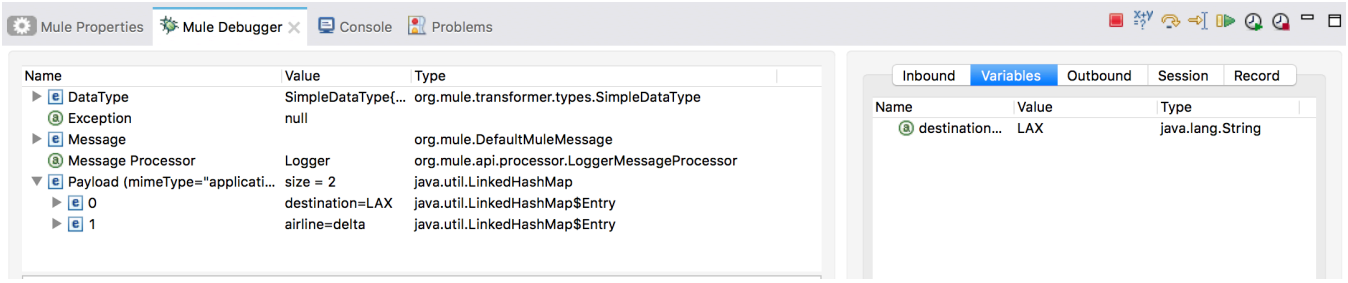
4. Select the Output type as Variable from the drop-down list.
5. Set the Variable name to destination.
6. Click OK.
7. Set the DataWeave expression to payload.destination.

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



Debug the application

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



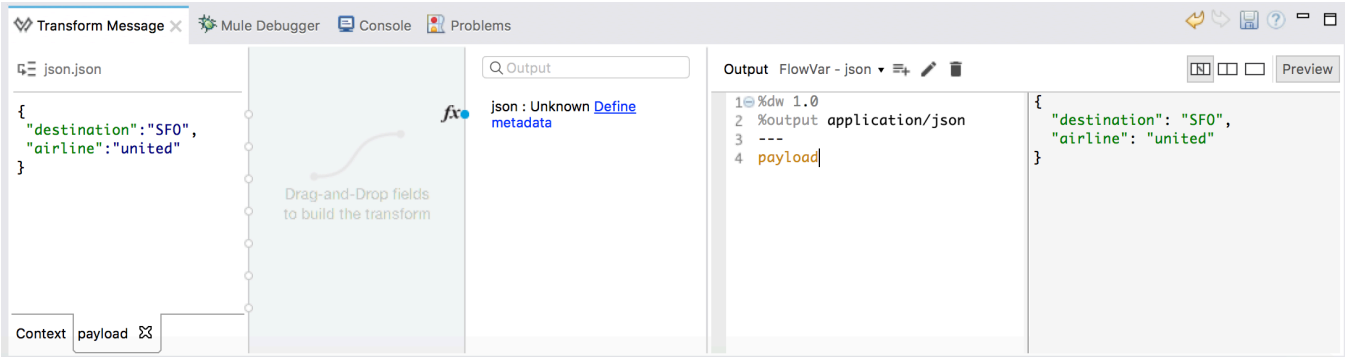
12. Click the Resume button.

    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.

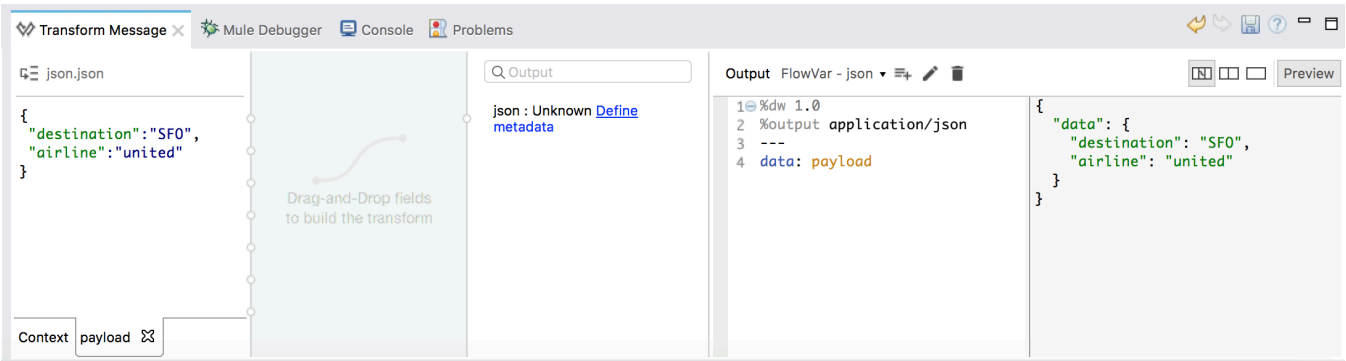
Create a third transformation to output data as JSON

13. In the Transform Message Properties view, click the Add new target button.
14. In the selection dialog window, leave the Output type set to Variable.
15. Set the name to json.
16. Click OK.
17. Set the DataWeave expression to payload.
18. Change the output type to application/json.

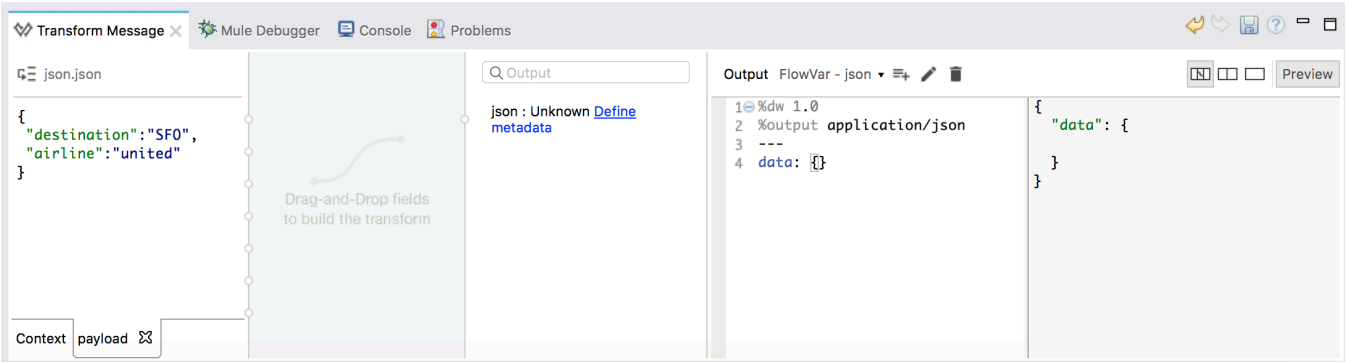
19. Look at the preview in the Output section.



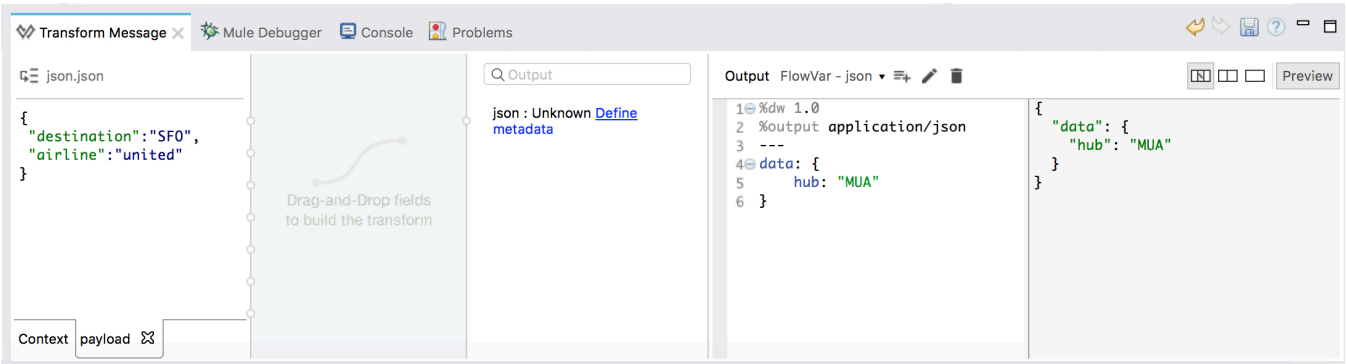
20. Change the DataWeave expression to data: payload.



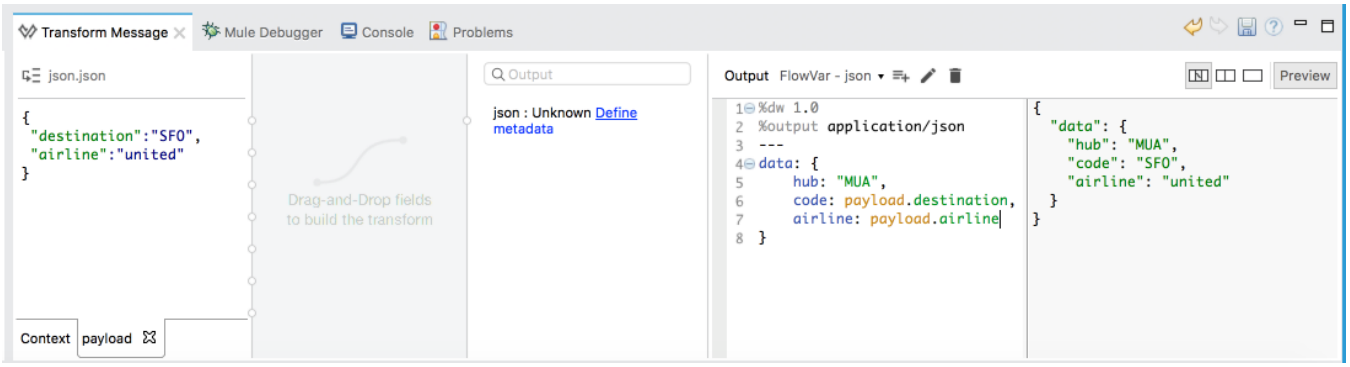
21. Change the DataWeave expression to data: {}.



22. Add a field called hub and set it to “MUA”.

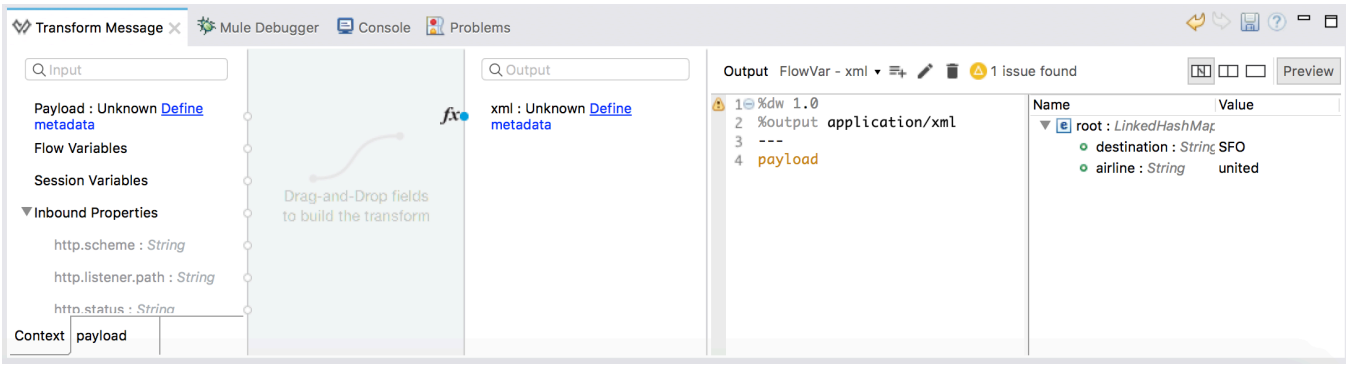


23. Add a field called code and set it to the destination property of the payload.
24. Add a field called airline and set it to the airline property of the payload.

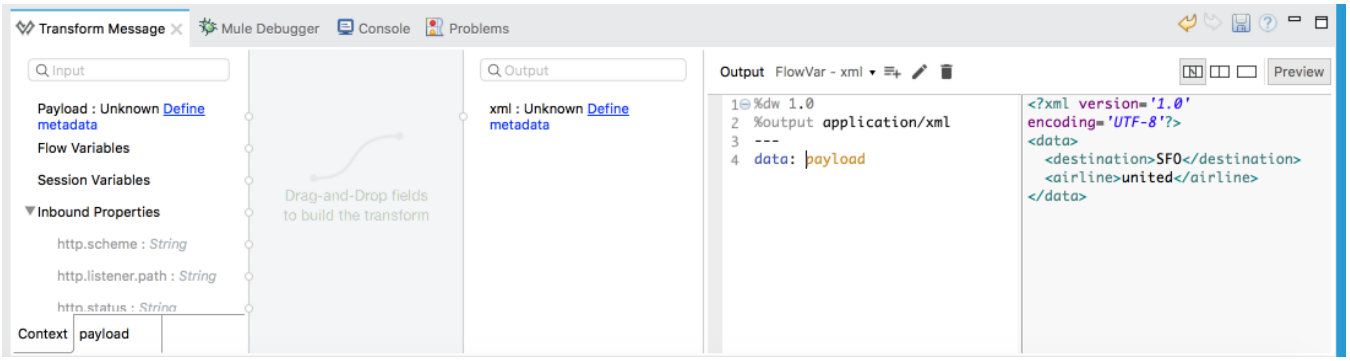


Create a fourth transformation to output data as XML

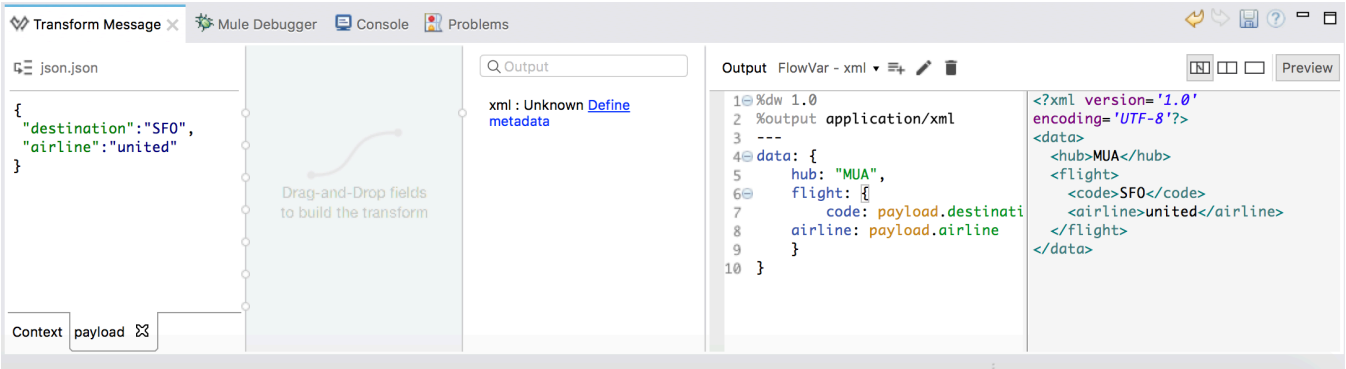
25. Click the Add new target button.
26. In the selection dialog window, leave the Output type set to Variable.
27. Set the name to xml.
28. Click OK.
29. Set the DataWeave expression to payload.
30. Change the output type to application/xml; you should get an issue in the Ouptut section.



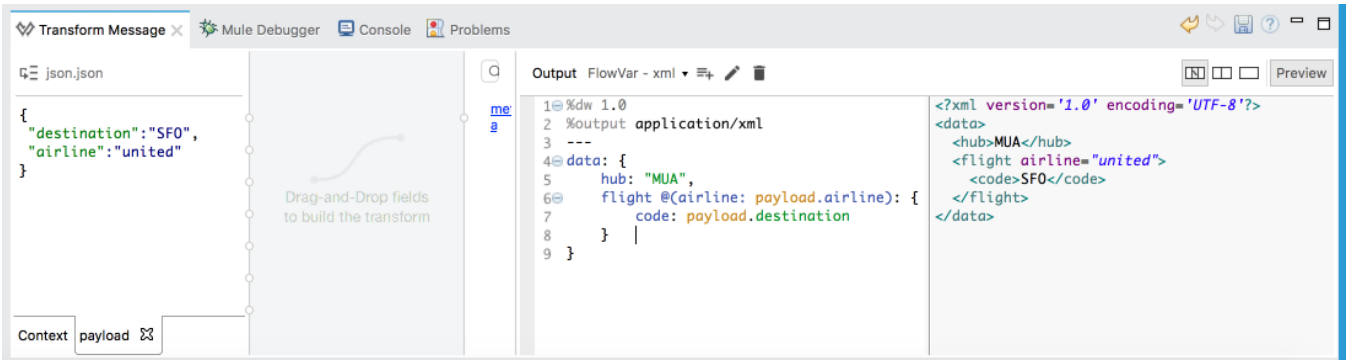
31. Change the DataWeave expression to data: payload.



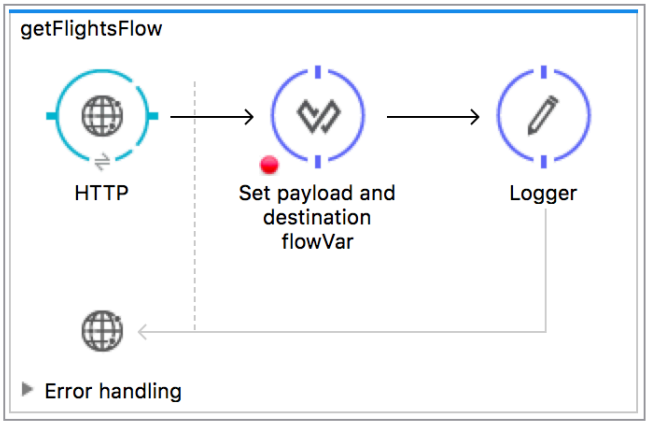
32. Click the drop-down arrow next to FlowVar – xml and select FlowVar - json and copy the DataWeave expression.
33. Switch back to FlowVar – xml and replace the DataWeave expression with the one you just copied.
34. Modify the expression so the code and airline properties are child elements of a new element called flight.



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

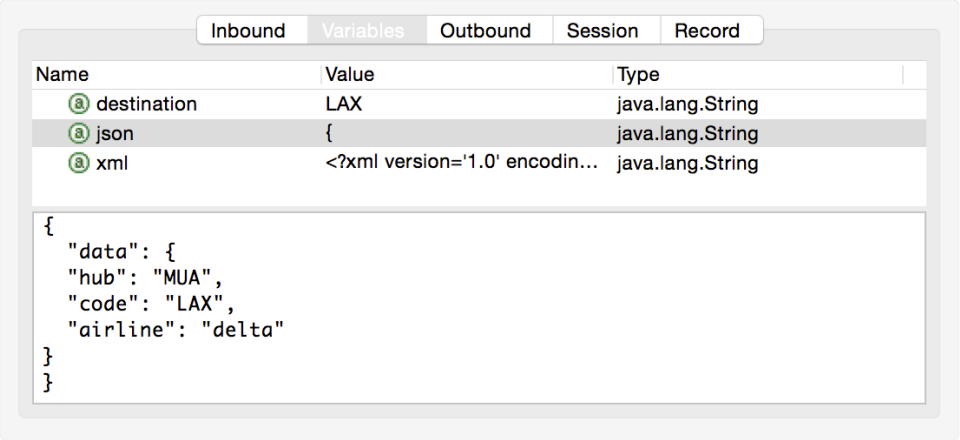


36. Change the name of the component to Set payload and destination flowVar.



Debug the application

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



40. Stop the Debugger.

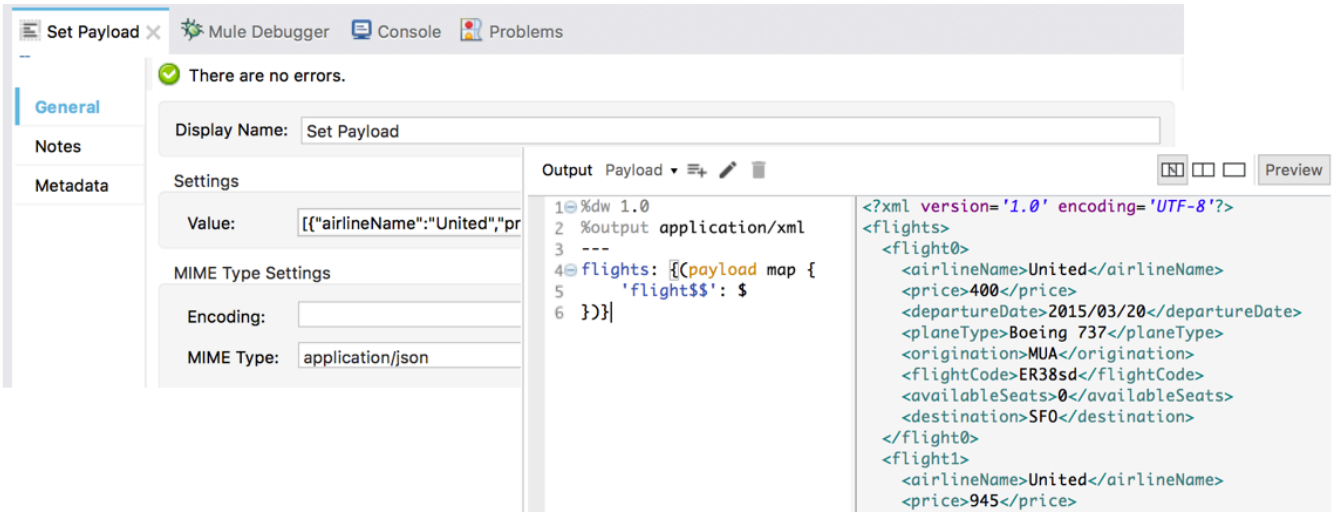
Note: The json and xml flow variables were created for learning purposes. If you want, you can delete them from the DataWeave transformer.

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.



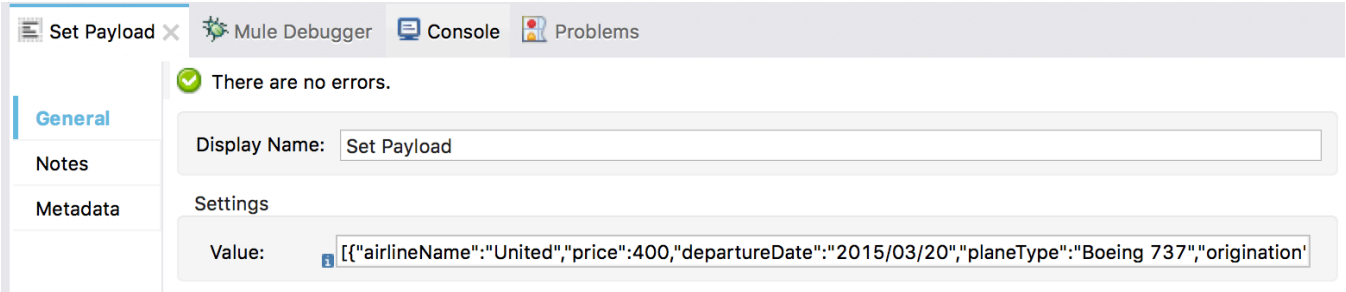
Create a new flow

1. Return to apessentials.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

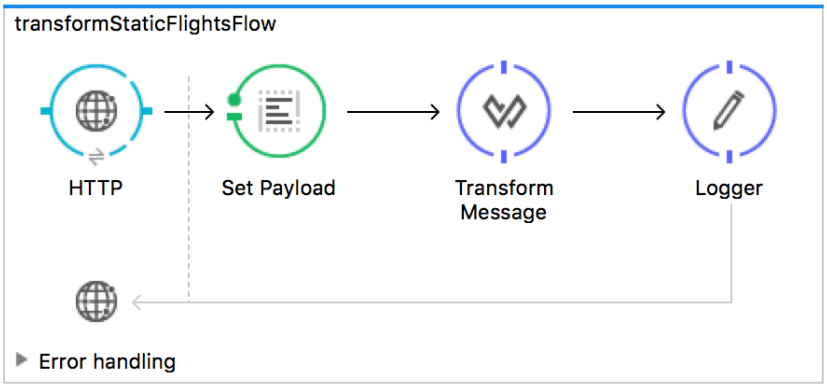
5. Add a Set Payload transformer to the flow.
6. Return to the course snippets.txt file and copy the value for the Example flights JSON.

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.

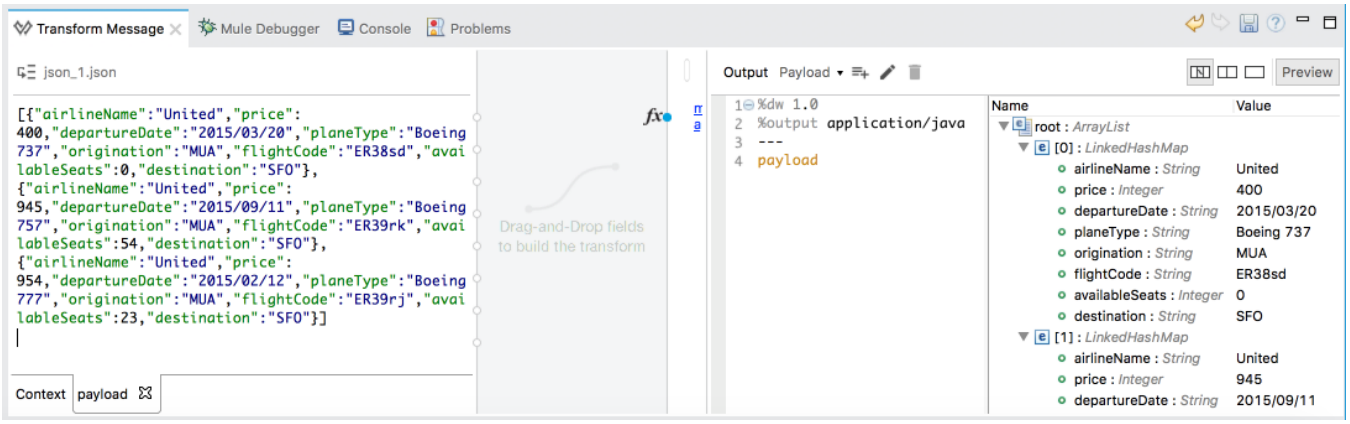


9. In the Transform Message Properties view, set the DataWeave expression to payload.
10. Look at the Preview tab in the Output section; you should not see any preview of the data.

Add sample data

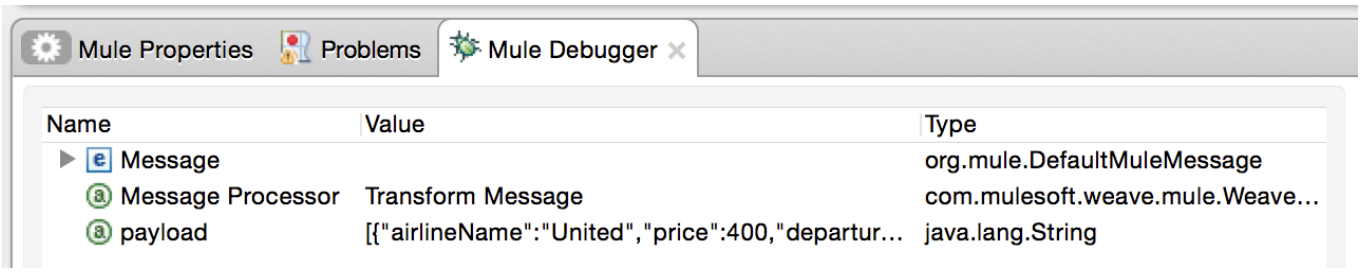
11. In the Input section, right click Payload: Unknown Define metadata.
12. Click Edit Sample Data.
13. In the Sample data format dialog box, select JSON and click OK.
14. In the new payload tab, replace the sample JSON with the JSON you used in the Set Payload transformer.

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.

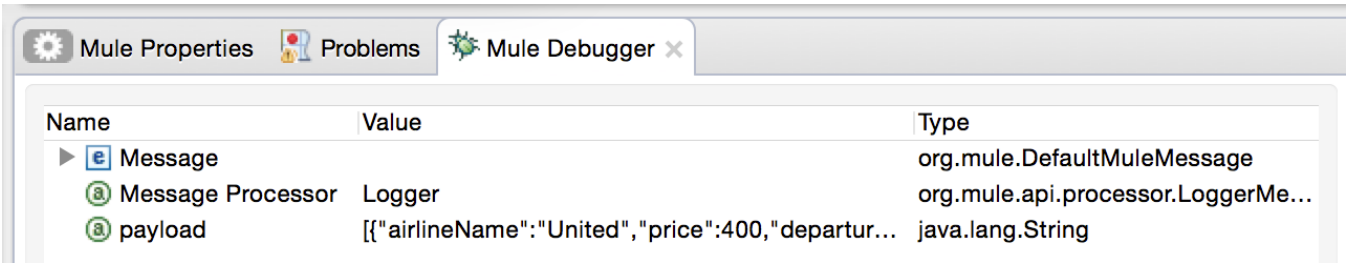


Debug the application

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.



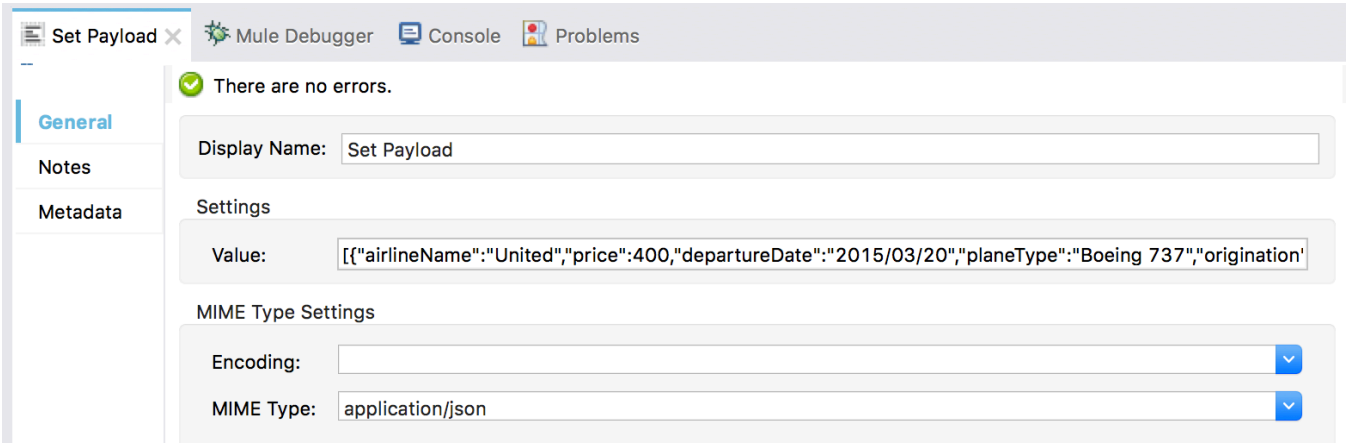
20. Step to the Logger; you should see the payload is still of type String.



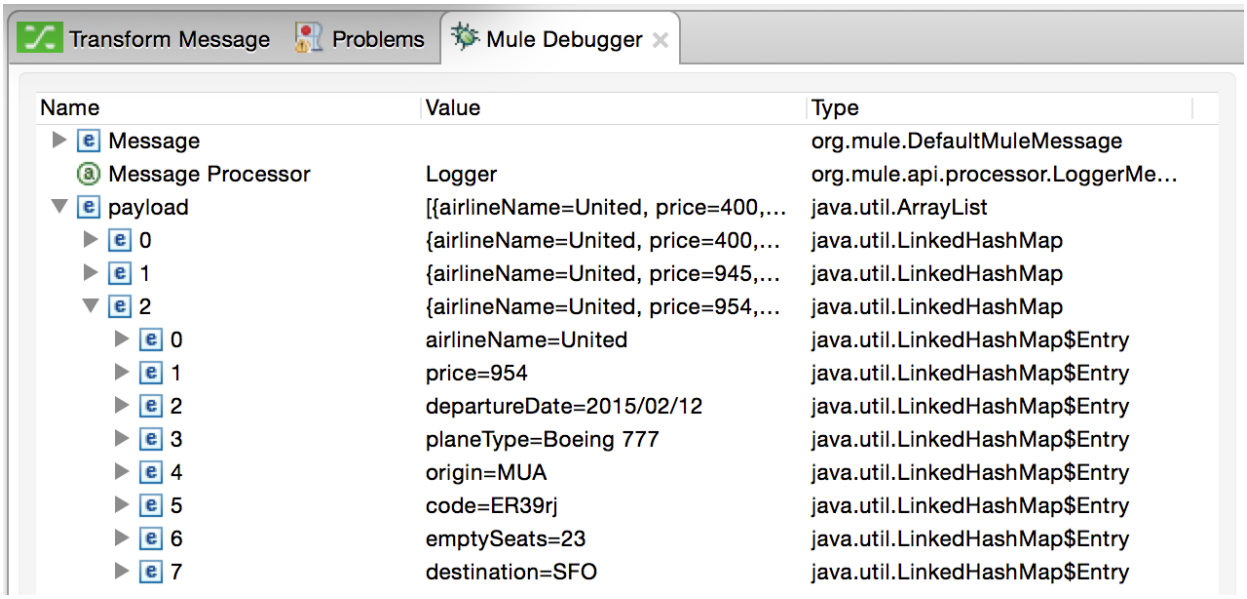
21. Click the Resume button.

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.

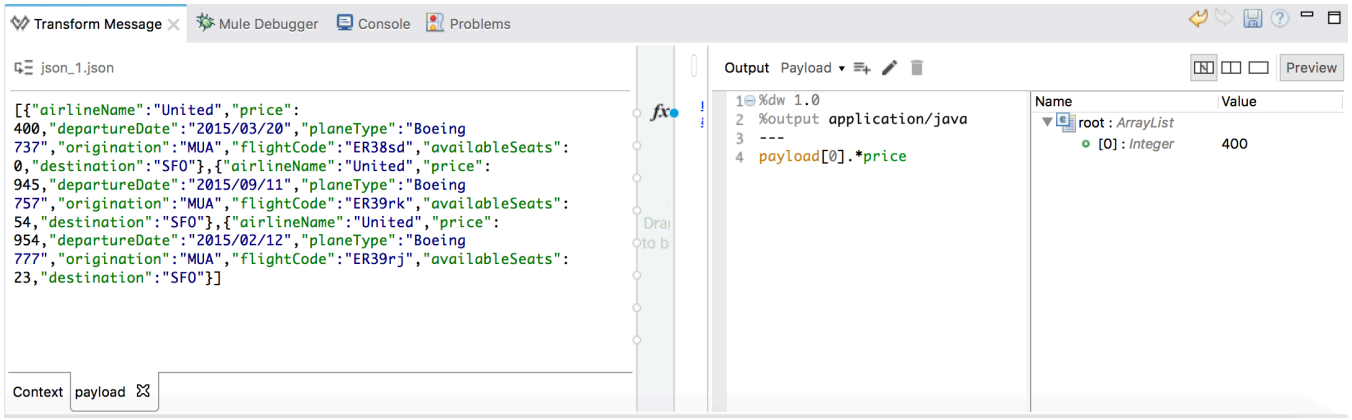


26. Click the Resume button; you should get a file download or garbled data depending upon if you are using a browser or some other tool to make your request.

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.

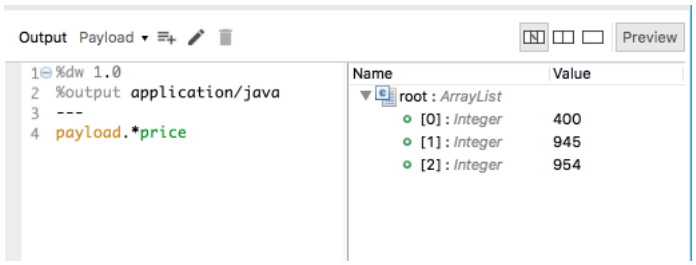


Return collections of values

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

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

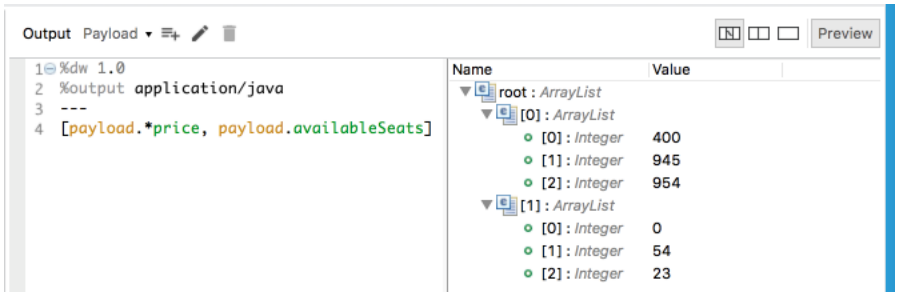
31. Look at the Preview tab; you should see an ArrayList with three values.



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

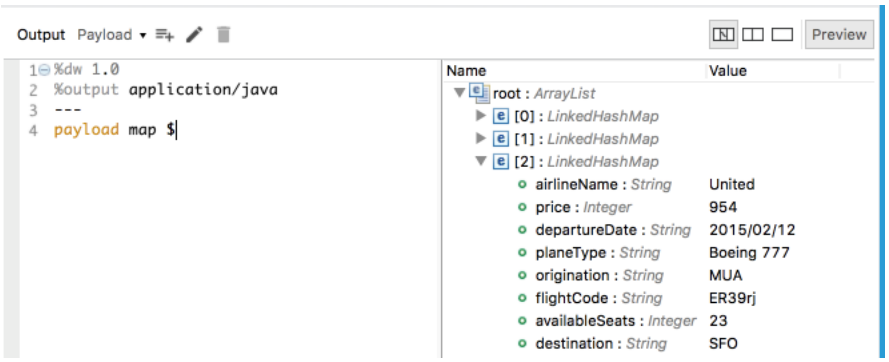
```
[payload.price, payload.availableSeats]
```

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

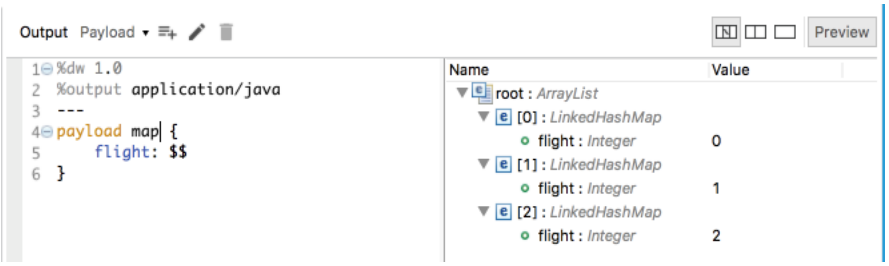


Use the map operator to return object collections with different data structures

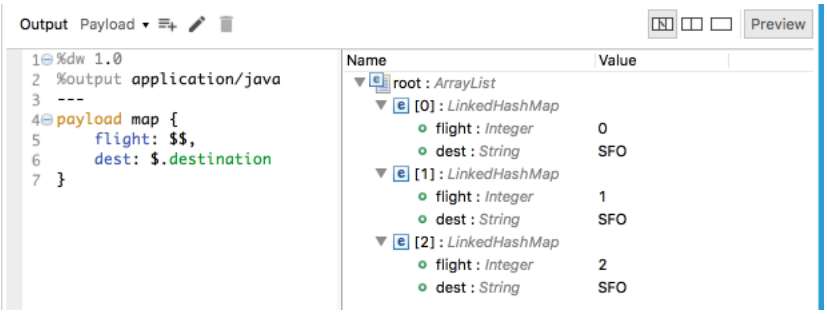
34. Change the DataWeave expression to payload map $, in the Preview tab, you should see three Java objects again.



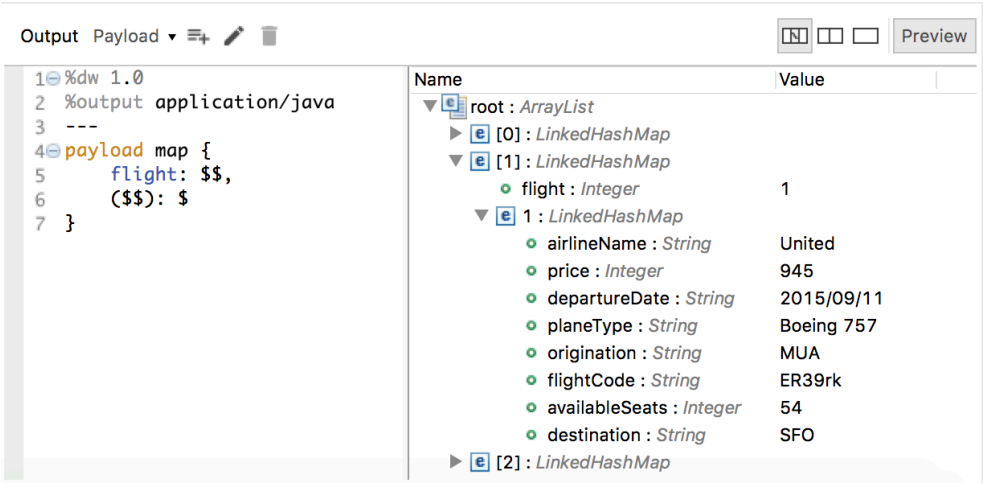
35. Change the DataWeave expression to set a property called flight for each object that is equal to its index value in the collection.



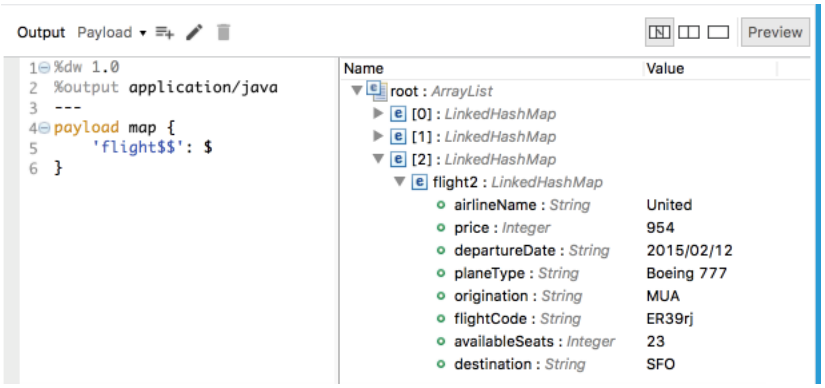
36. Change the DataWeave expression to create a property called dest for each object that is equal to the value of the destination field in the payload collection.



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



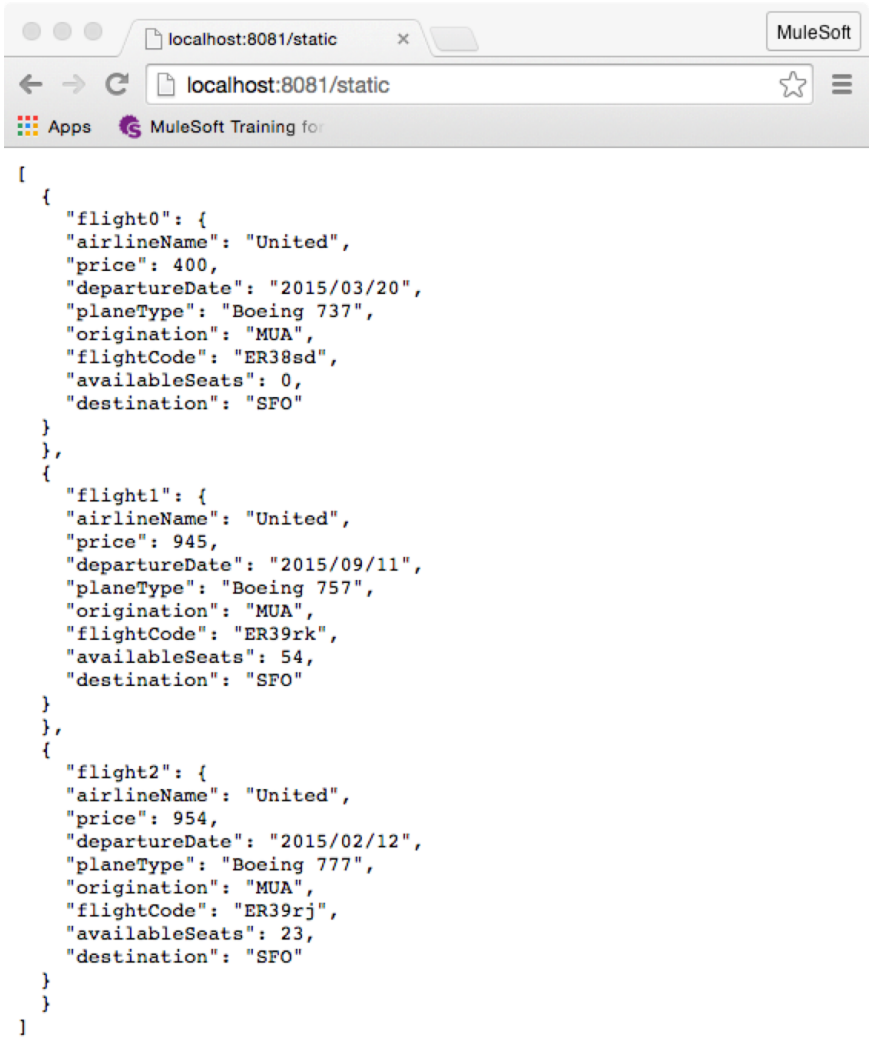
38. Change the DataWeave expression so the name of each object is flight+index and you get flight0, flight1, and flight2.



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.

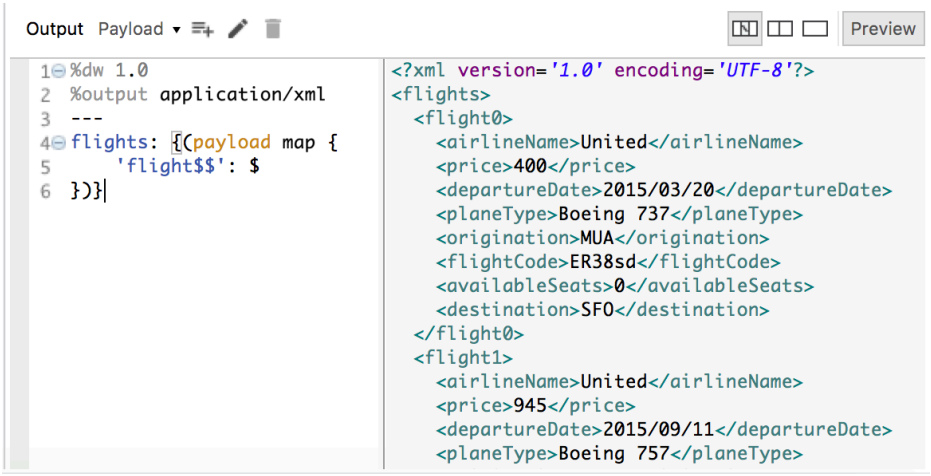


Change the expression to output valid XML and test the application

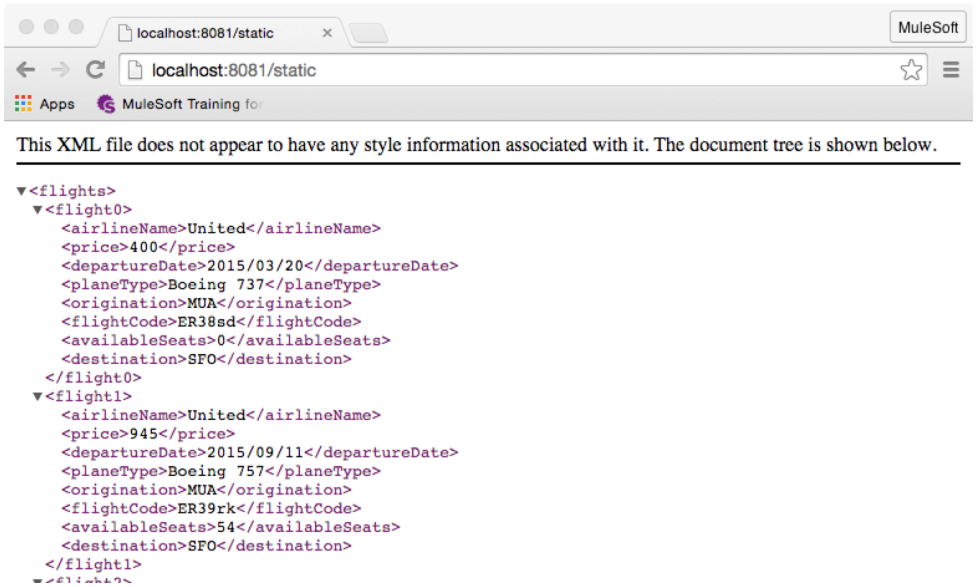
42. Change the DataWeave expression output type from application/json to application/xml; you should get an issue in the Ouput tab.
43. Change the DataWeave expression to create a root node called flights; you should still get an error in the Preview tab.

44. Change the expression to map each item in the input array to an XML element.

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



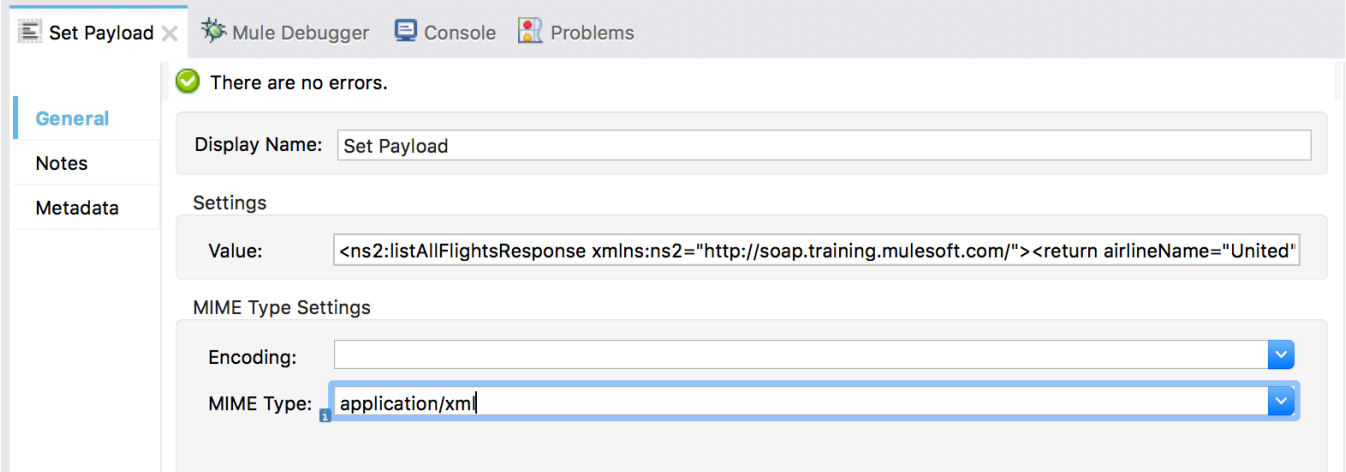
45. Save the file and run the application.
46. Make a request to http://localhost:8081/static; you should see the XML returned.



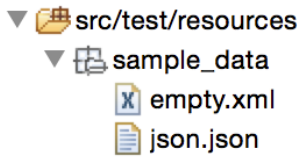
Add static XML payload data

47. Return to the course snippets.txt file and copy the value for the Example flights XML.
48. In the Set Payload Properties view, paste the copied value into the value field.

49. Change the MIME type to application/xml.



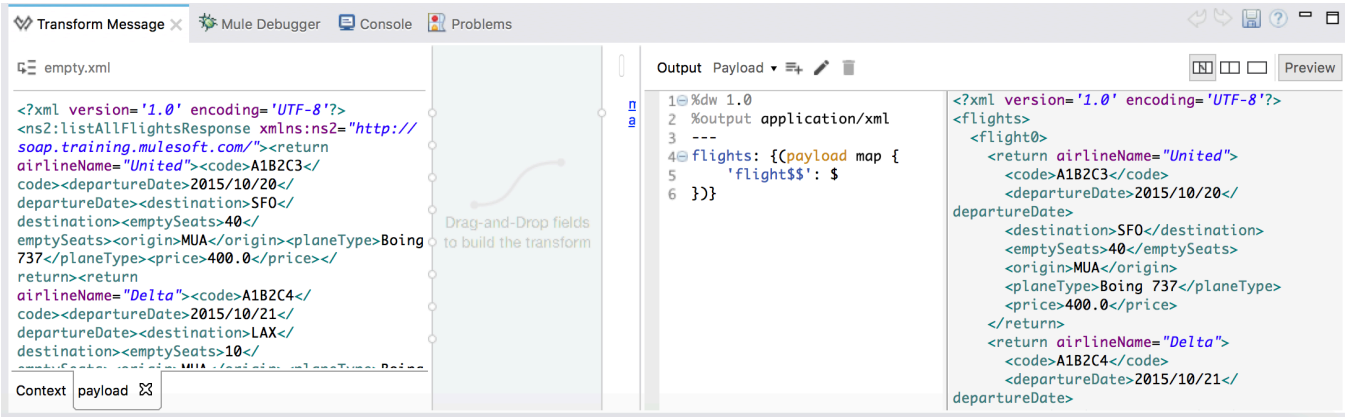
50. In the Input section of the Transform Message Properties view, click the x on the payload tab to delete the src/test/resources/sample_data/json_1.json file.
51. Right click Payload: Unknown Define Metadata and then select Edit Sample Data button.
52. In the Sample data format dialog box, select XML and click OK; a new sample data file empty.xml should be created.



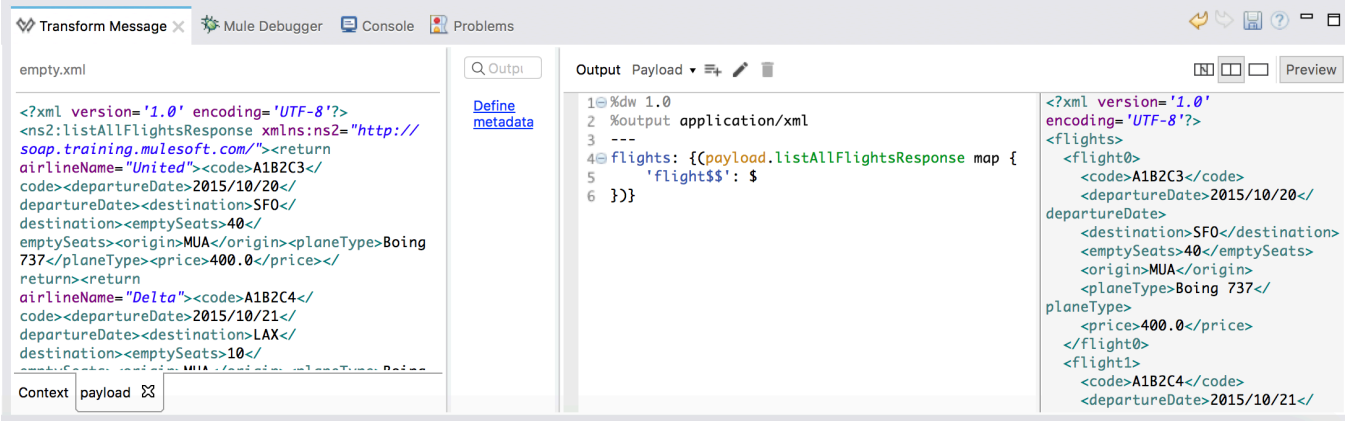
53. In the Transform Message Properties view, replace the sample payload data with the Example flights XML you copied.

Change the return structure of the XML

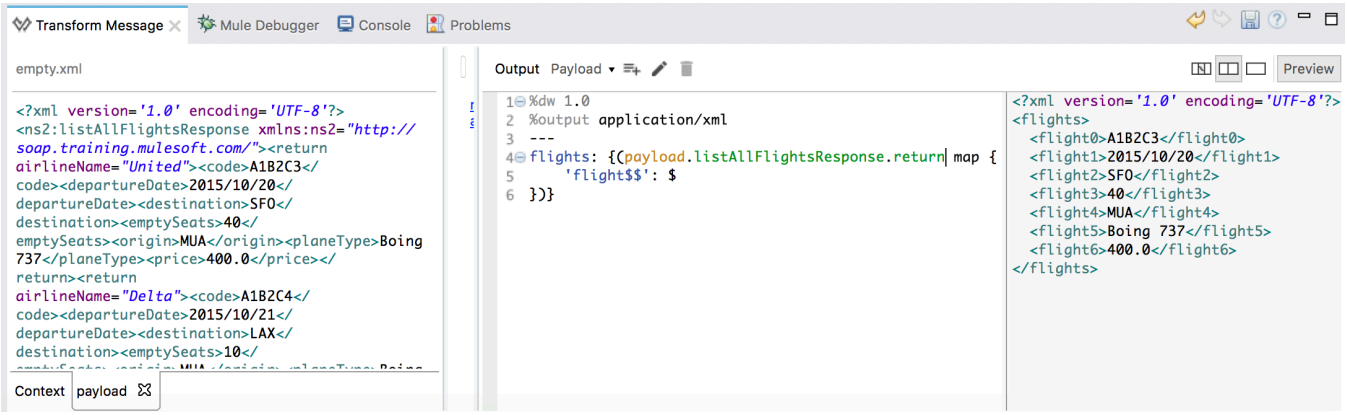
54. Look at the Preview tab; you should see all the flights are children of the flight0 element.



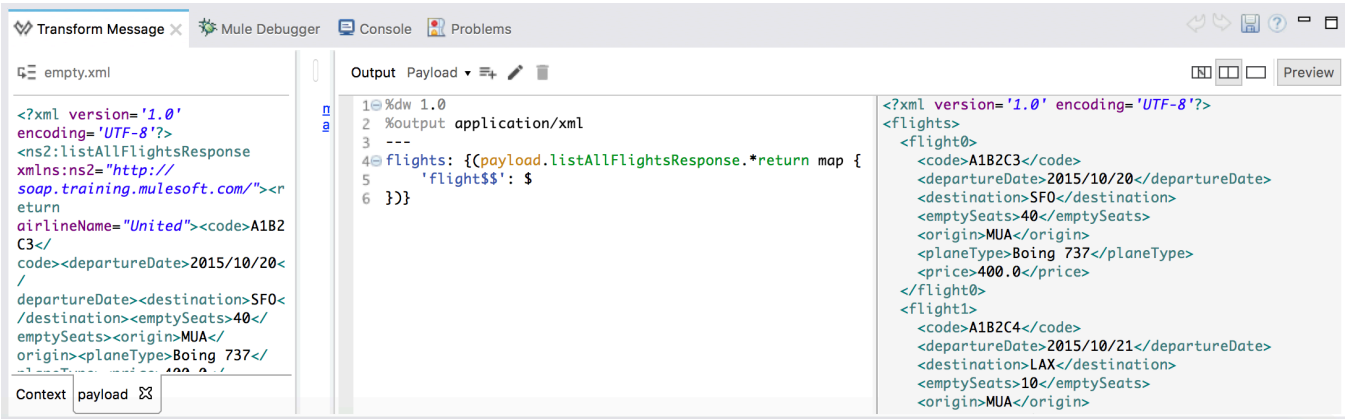
55. Change the DataWeave expression to loop over the payload.listAllFlightsResponse element; you should see the flights are now correctly transformed.



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



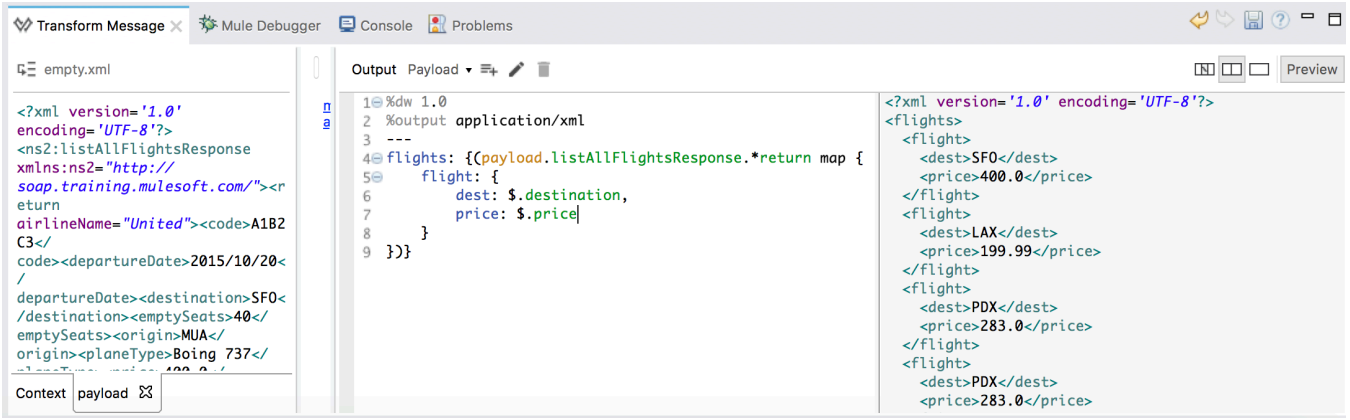
57. Change the DataWeave expression use * to loop over the XML repeated return elements.



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

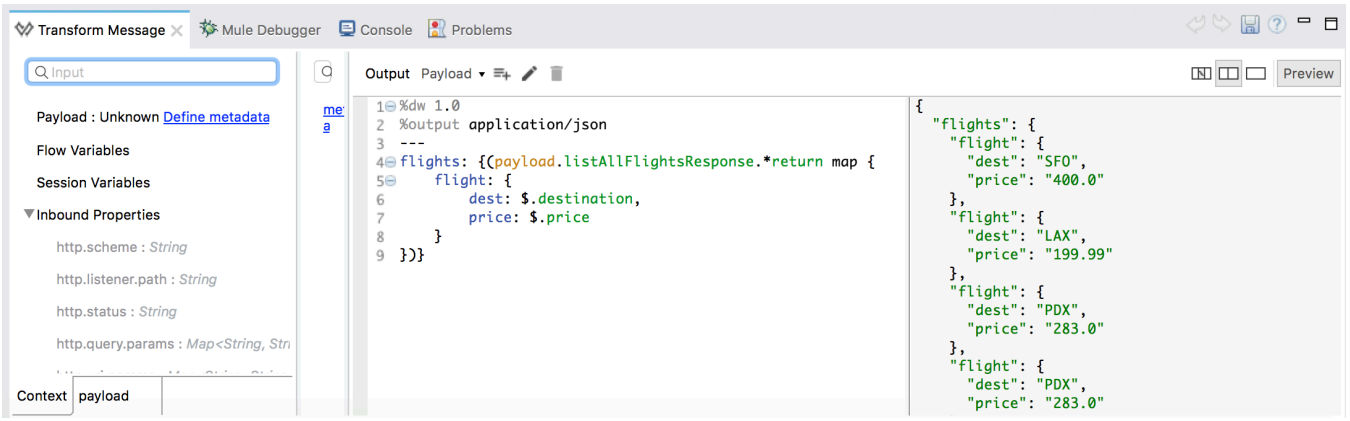
```
flights: {(payload.listAllFlightsResponse.*return map {
      flight: $
  }
)}
```

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



Change the output structure to JSON

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



Note: This output JSON has repeated object keys and is considered by most parsers to be invalid JSON.

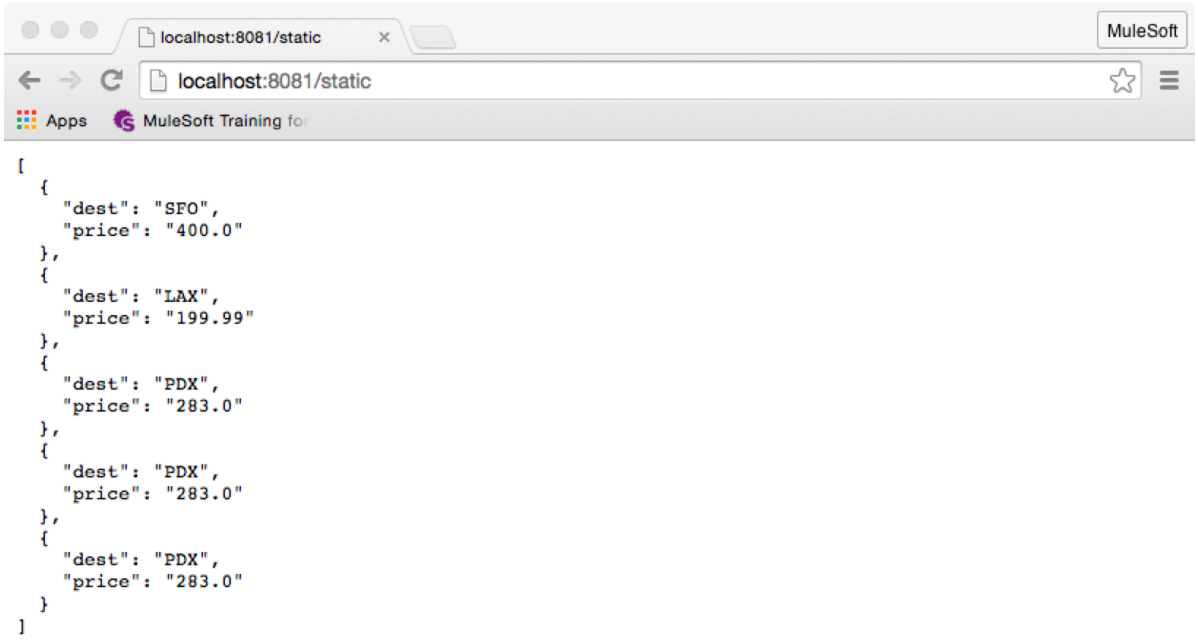
61. Remove the {(and)} around the payload map expression.

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

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

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

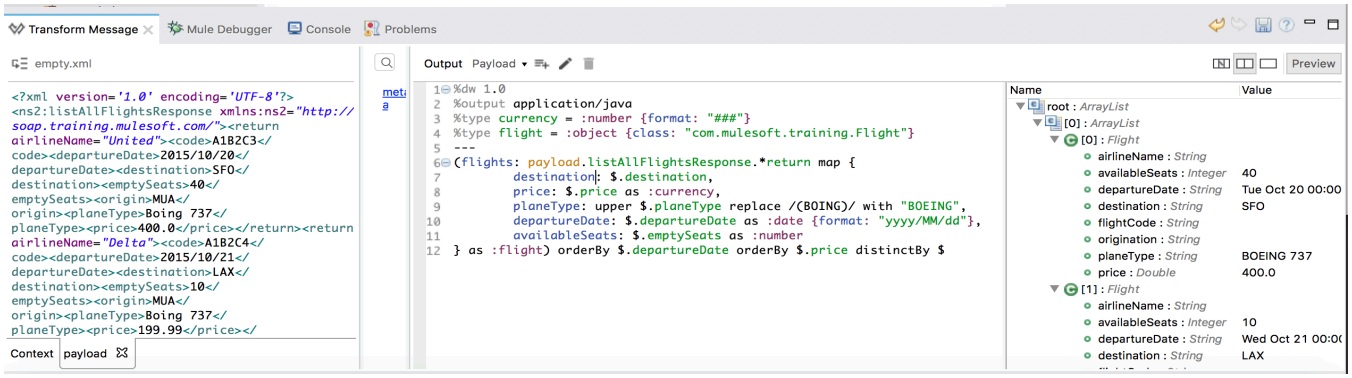
63. Save the file and run the application.
64. Make a request to http://localhost:8081/static and look at the JSON structure returned.



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.

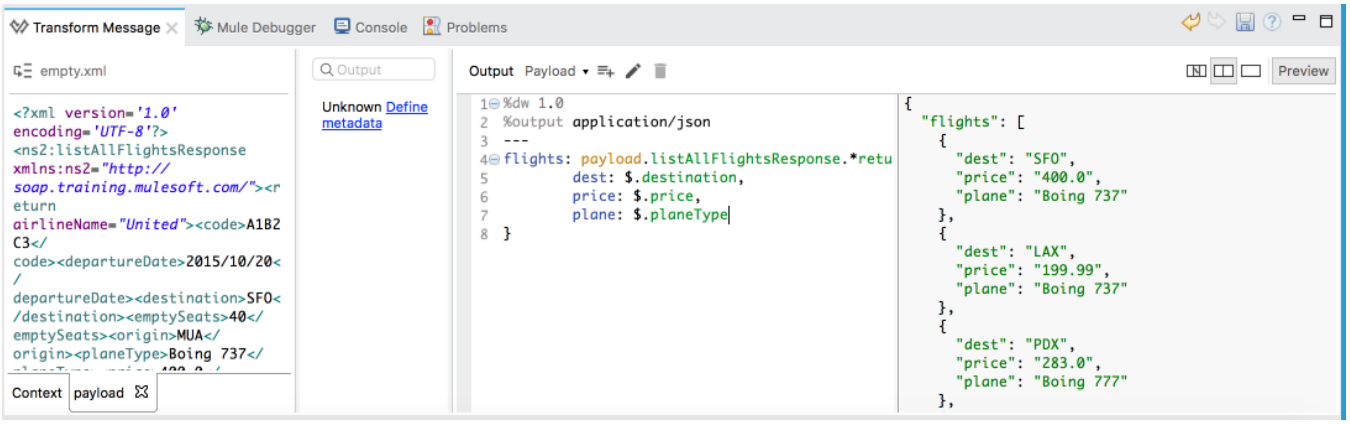


Format a string

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

   ```
   plane: upper $.planeType
   ```
4. Look at the Preview tab; you should see the uppercase plane fields.

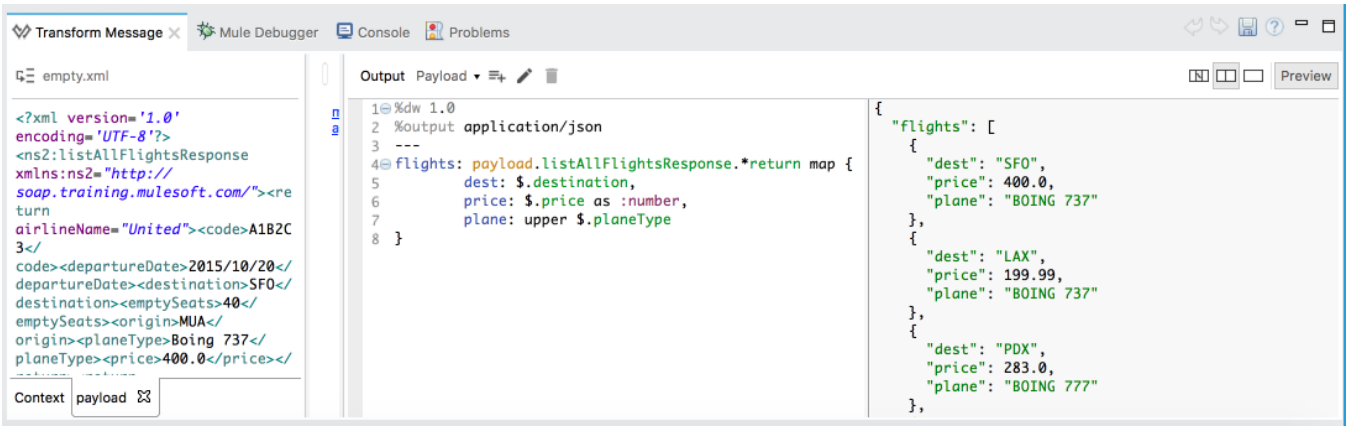
5. Look at the data type of the prices; you should see that they are strings (surrounded by quotation marks).



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.

```
price: $.price as :number,
```



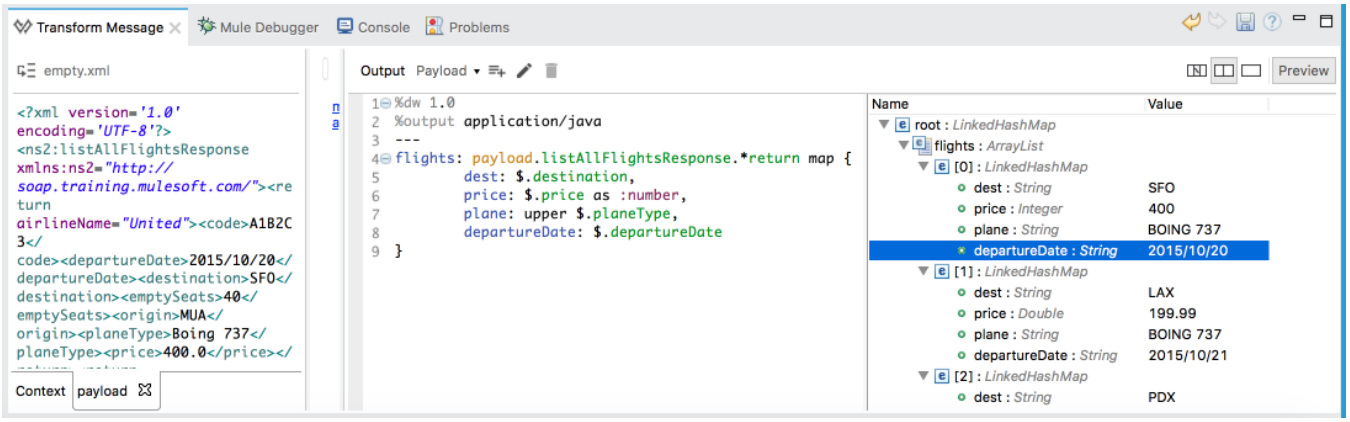
7. Add a departureDate field to the return object.

```
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
```

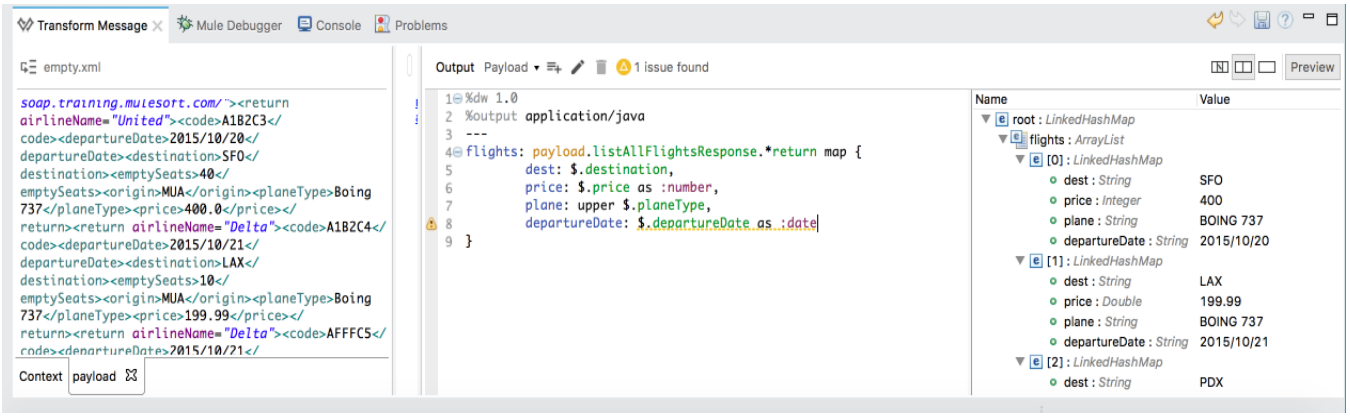
9. Look at the Preview tab; you should see the departureDate property is a String and price is an Integer or Double.



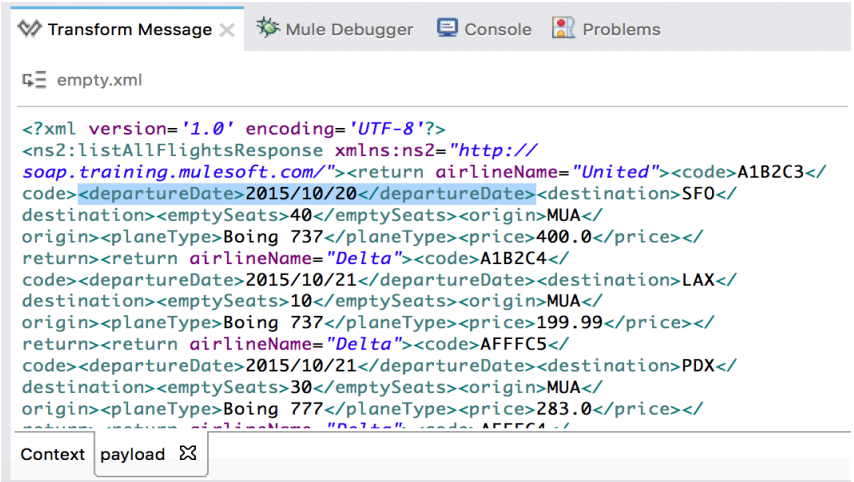
10. Change the DataWeave expression to use the as operator to convert departureDate to a date object.

```
departureDate: $.departureDate as:date
```

11. Look at the Output tab; you should get an issue in the transformation.



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

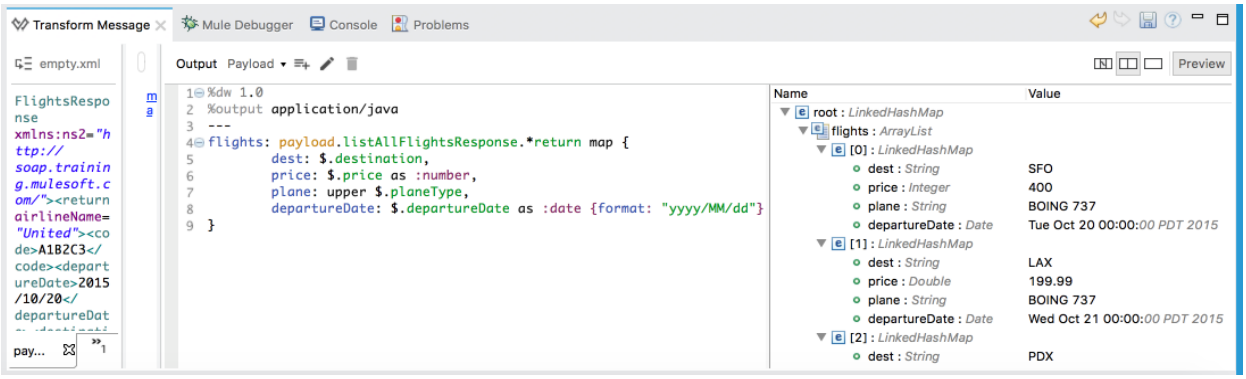


13. Use the format operator to specify the pattern of the input date strings.

```
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.

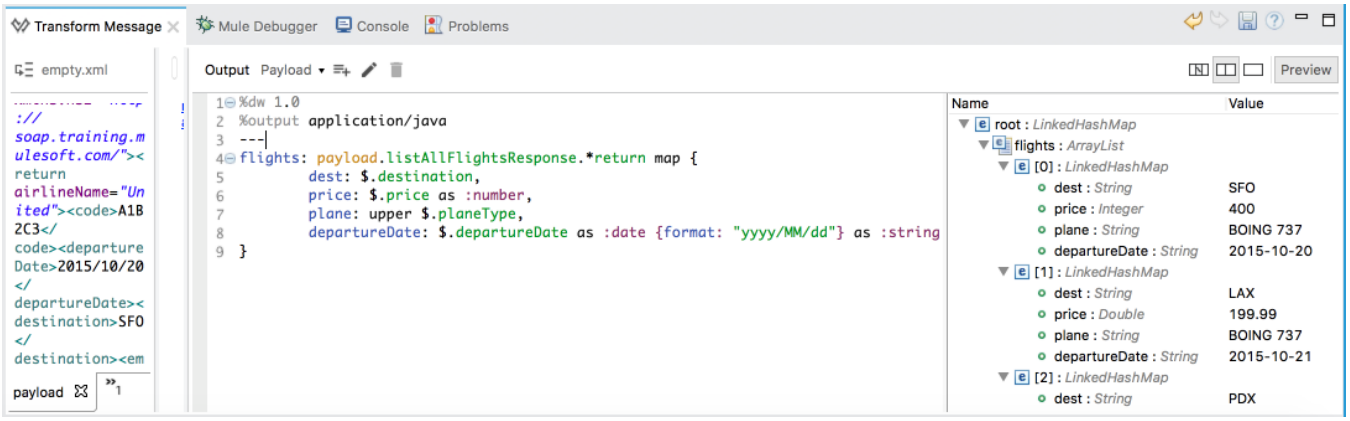


Format a date

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.



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

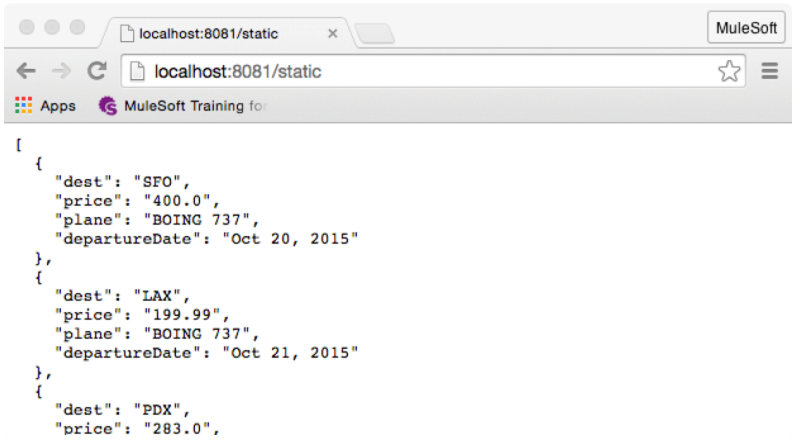
```
departureDate: $.departureDate as :date {format: "yyyy/MM/dd"}
as :string {format: "MMM dd, yyyy"}
```

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

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.



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.

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.
28. To get trailing zeros, convert the numbers to strings and format the strings.

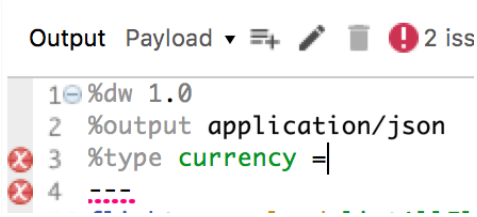
```
price: $.price as :number as :string {format: "#,###.00"},
```

29. Use the as operator again to convert the formatted strings back to numbers.

```
price: $.price as :number as :string {format: "#,###.00"} as :number,
```

Define a custom data type

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



31. Set it to a number formatted to have no digits after the decimal point.

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

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

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

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

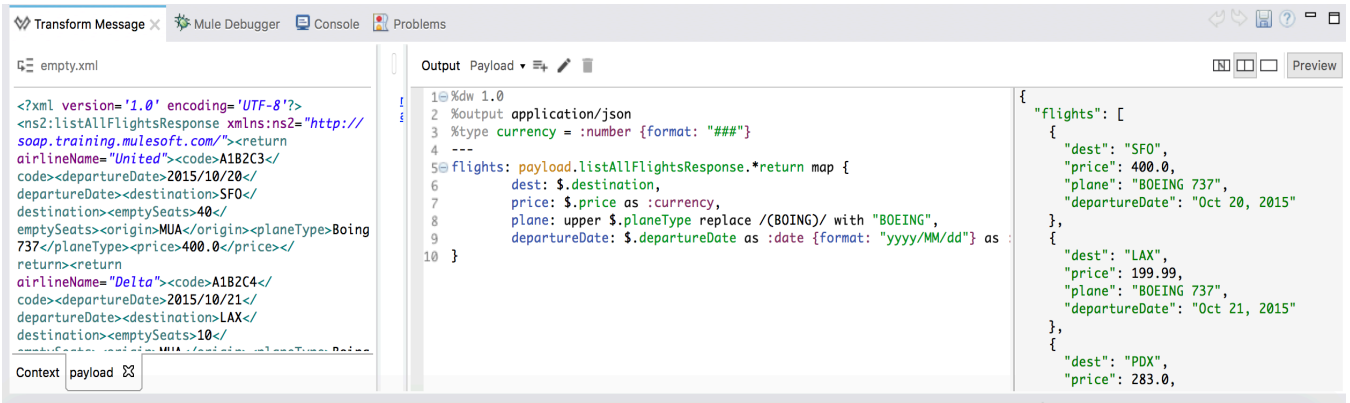
Note: There is a bug in Mule runtime 3.7.1 and the numbers are not formatted correctly.

Replace data values

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

```
plane: upper $.planeType replace /(BOING)/ with "BOEING",
```

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



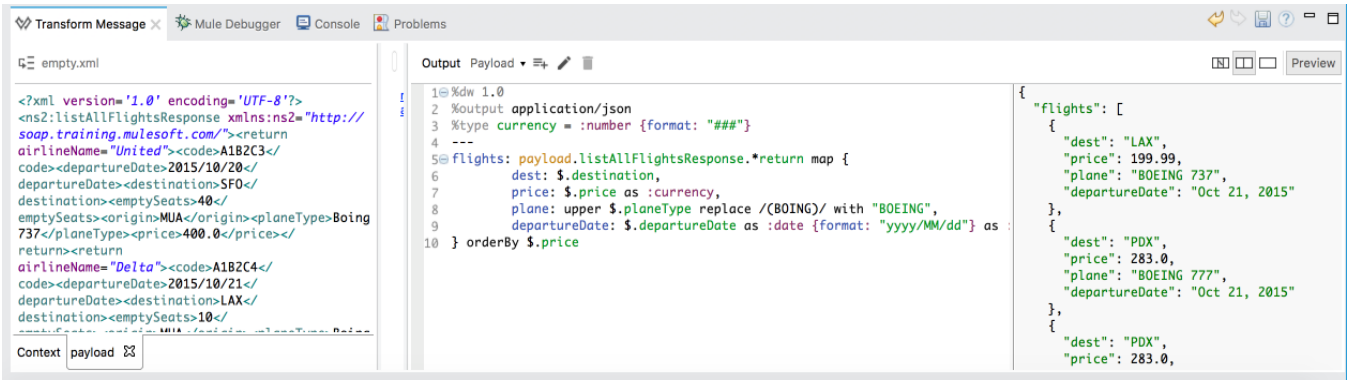
Order data

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

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

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

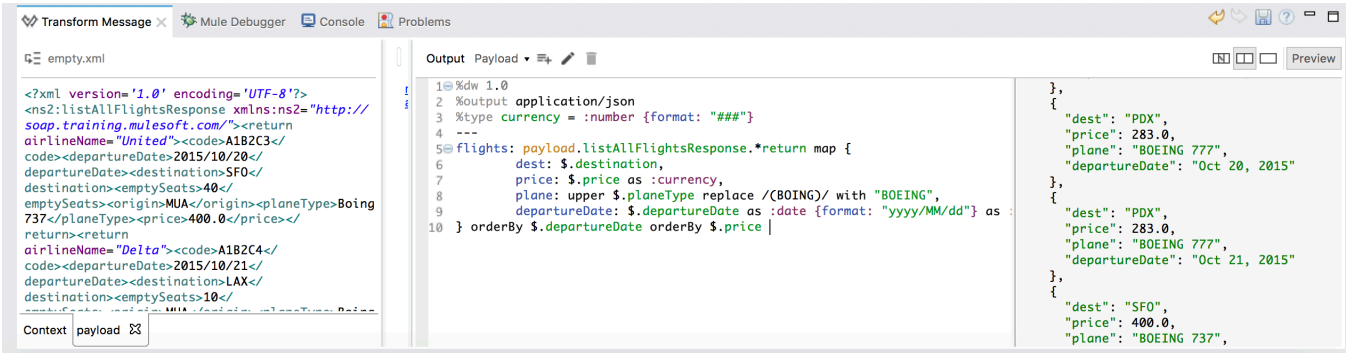
38. Look at the Preview tab or run the application; the flights should now be ordered by price.



39. Look at the PDX flights; they should not be ordered by departureDate.
40. Change the DataWeave expression to first sort by departureDate and then by price.

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

41. 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.

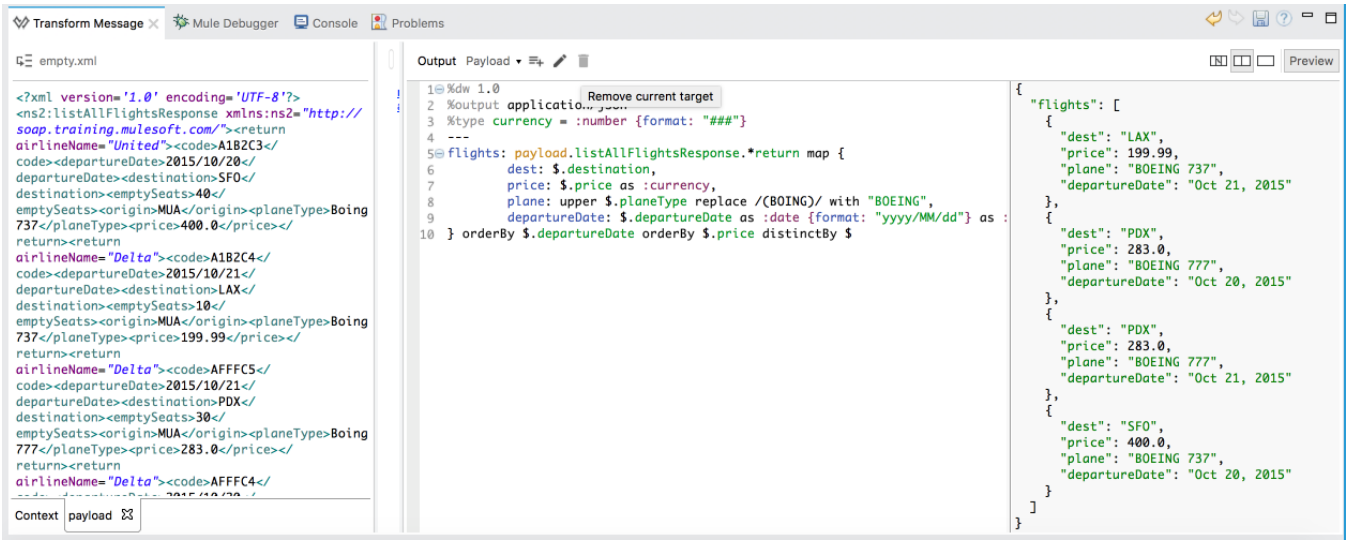


Remove duplicate data

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

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

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



Note: If you still get five flights, remove the currency type coercion from the price field; use price: $.price or price: $.price as :number {format: "###"} instead of price: $.price as :currency. This is a bug in Mule runtime 3.7.1.

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

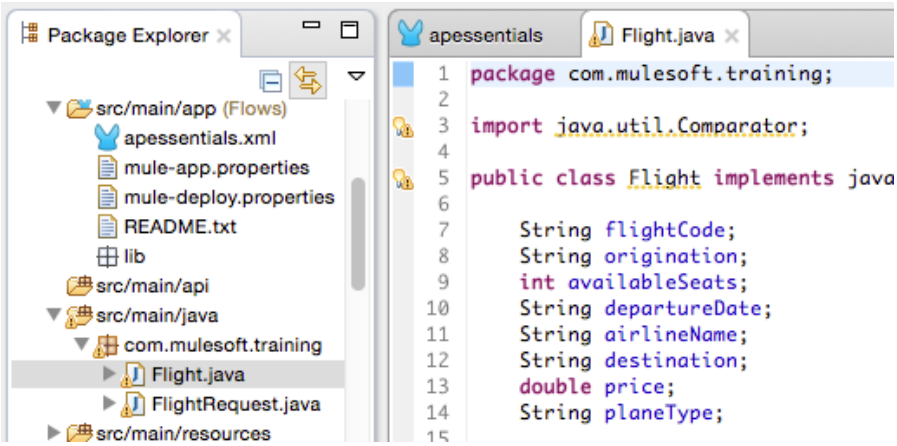
```
availableSeats: $.emptySeats as :number
```

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

Add Java classes file to the project

46. On your computer, navigate to the java folder in the student files folder for the course: APEssentials3.8_studentFiles_{date}/java.
47. Copy and paste (or drag) the com.mulesoft.training folder into your Anypoint Studio project's src/main/java folder.

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



Transform objects to POJOs

49. Return to transformStaticFlightsFlow in apessentials.xml.
50. Change the transformation output from json to java.

```
%output application/java
```

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

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

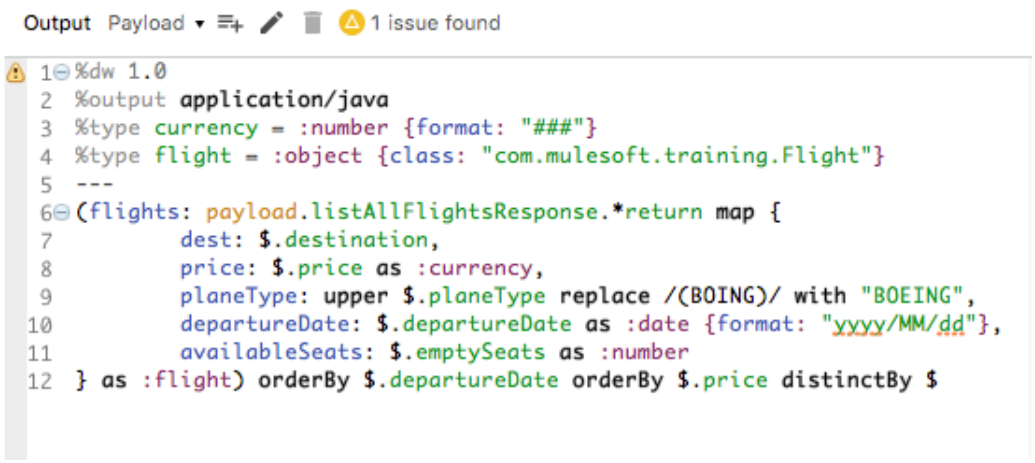
```
%type flight = :object {class: "com.mulesoft.training.Flight"}
```

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

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

Note: There is a bug in Mule 3.8.0 and operators are not applied after a custom type declaration.

54. In the Transform section, look at the warning you get: java.lang.NullPointerException.



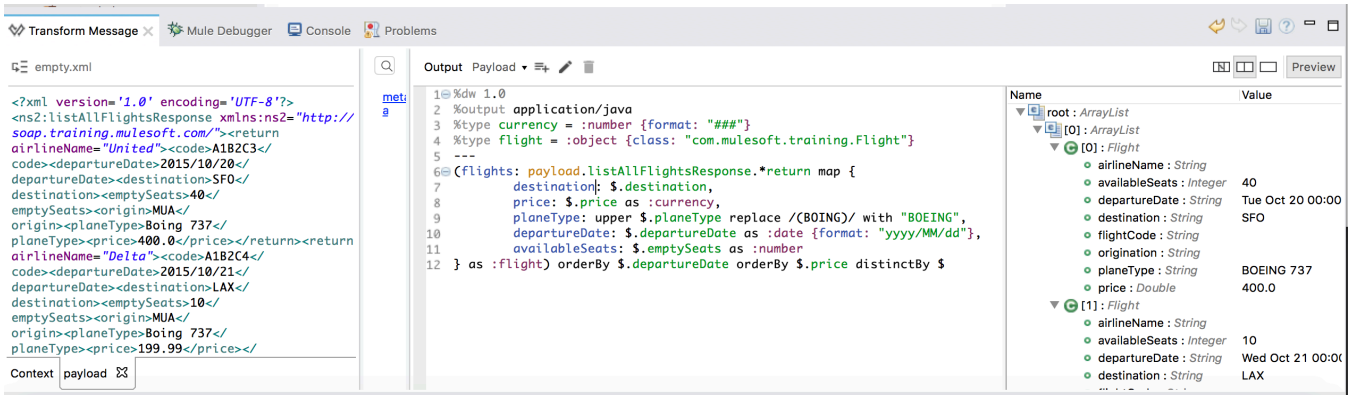
55. Change the name of the dest key to destination.

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

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

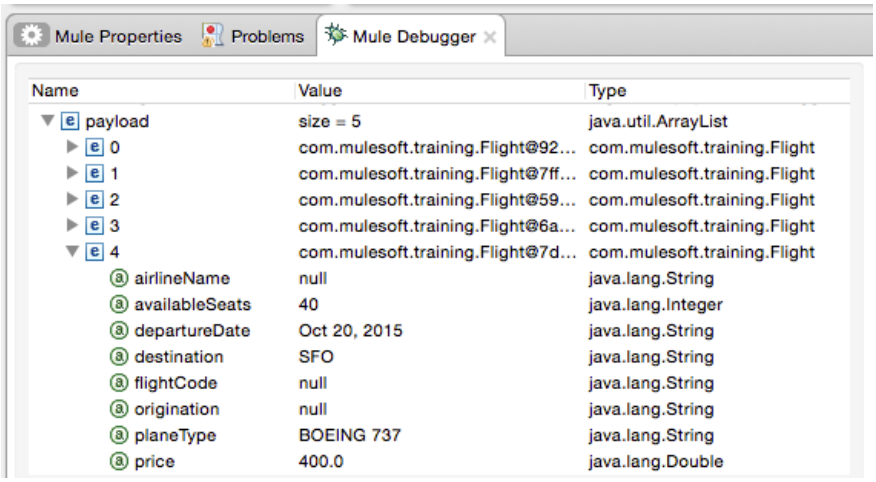
```
planeType: $.planeType,
```

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



58. Save the file and debug the application.
59. Make a request to http://localhost:8081/static.

60. Step to the Logger and drill-down into the payload; the objects should be of type Flight and several of the fields should have null values because no values were assigned to them.



61. Stop the debugger.

Filter data

62. Return to transformStaticFlightsFlow in apessentials.xml.
63. In the Preview tab, look at the values of the availableSeats properties.
64. Add a filter to the DataWeave expression that removes any objects that have availableSeats equal to 0.

```
(flights: payload.listAllFlightsResponse.*return map {
        …
} as :flight) orderBy $.departureDate orderBy $.price distinctBy $
filter ($.availableSeats !=0)
```

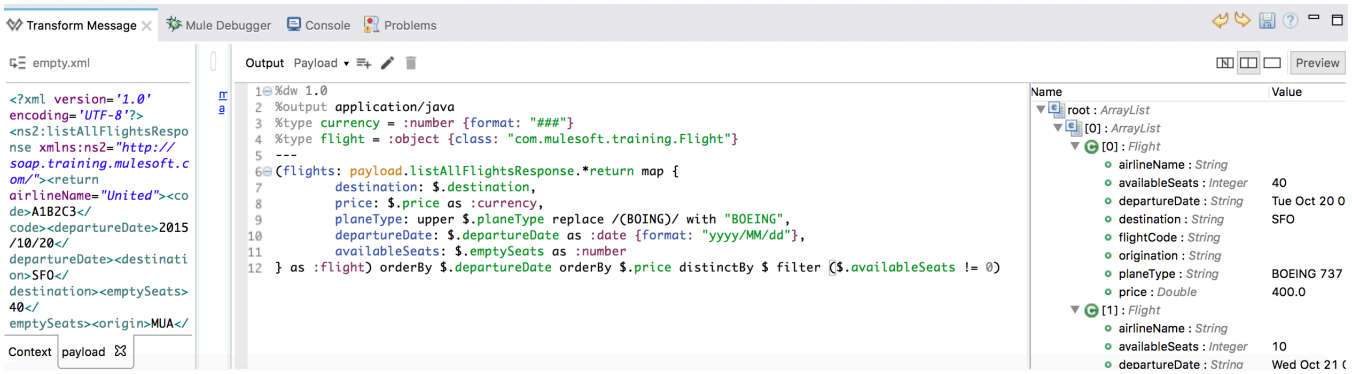
Note: There is a bug in Mule 3.8.0 and operators are not applied after a custom type declaration. To see the filtering mechanism work, update the expression as:

*payload.listAllFlightsResponse.*return map {*

…

} orderBy $.departureDate orderBy $.price distinctBy $ filter ($.availableSeats !=0)

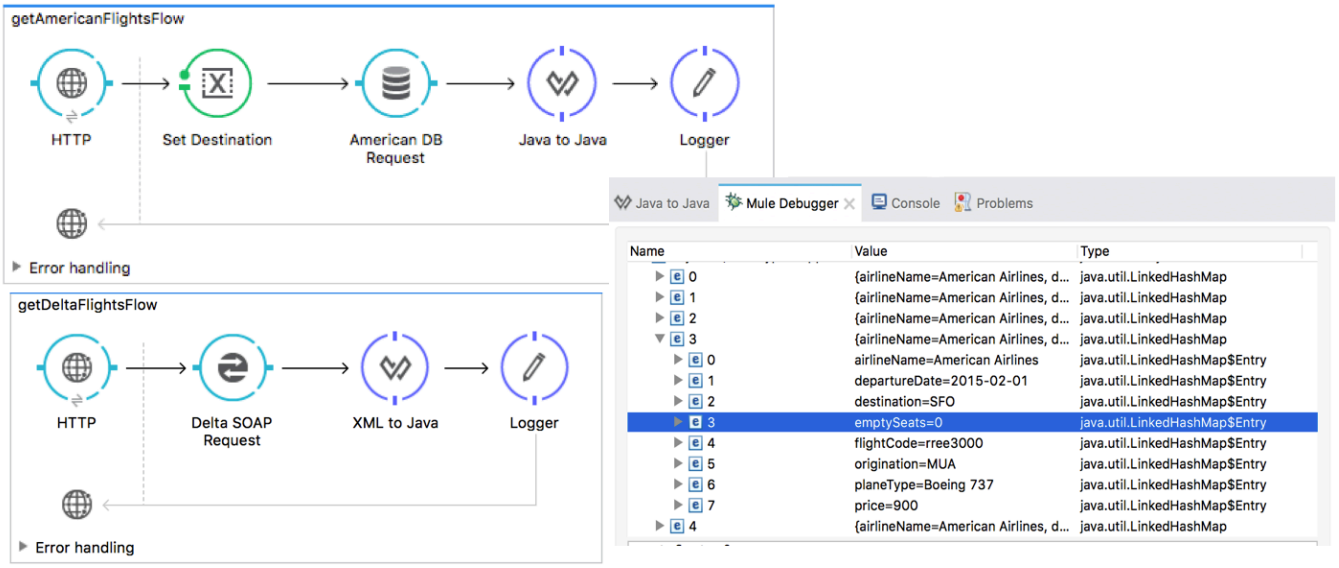
65. Look at the Preview tab; you should no longer see the flight that had no seats.



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

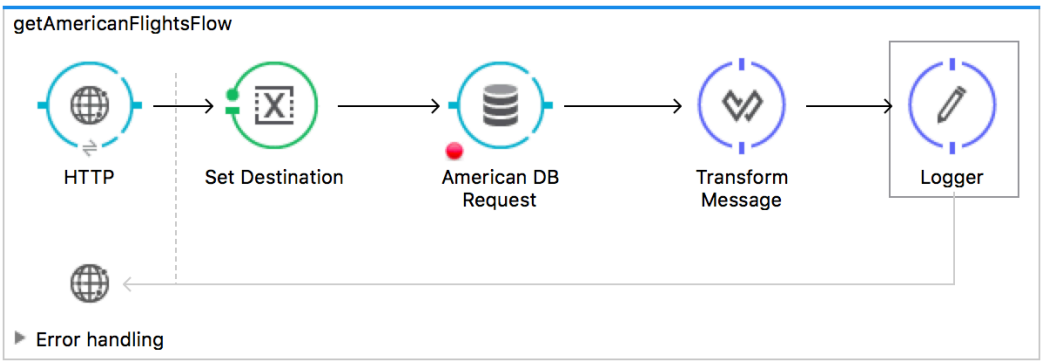
In this walkthrough, you will work with the American and Delta flight data. You will:

- 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.



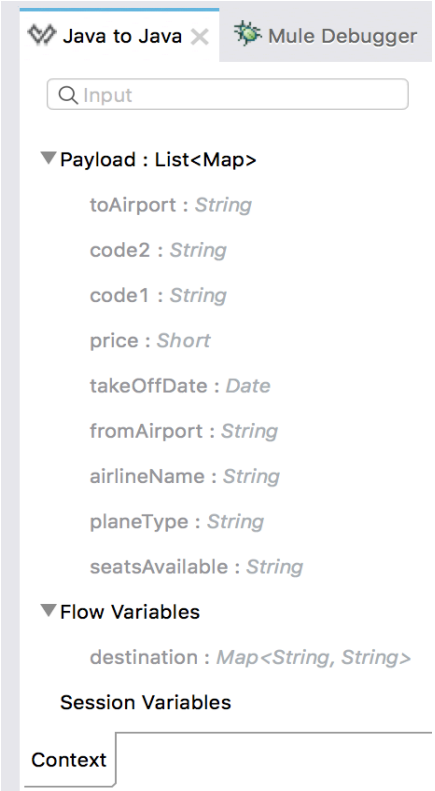
Add a DataWeave Transform Message component

1. Locate getAmericanFlightsFlow in apessentials.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.

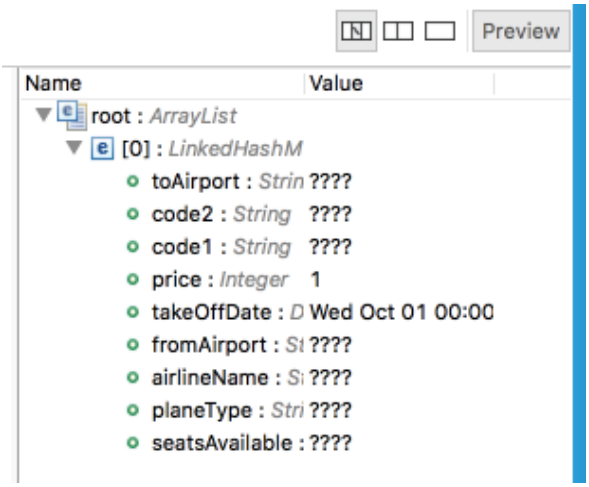
5. Look at the Input section of the Transform Message Properties view; the payload should automatically be set to a List<Map> with correct properties because the Database connector configuration is DataSense enabled by default.



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.

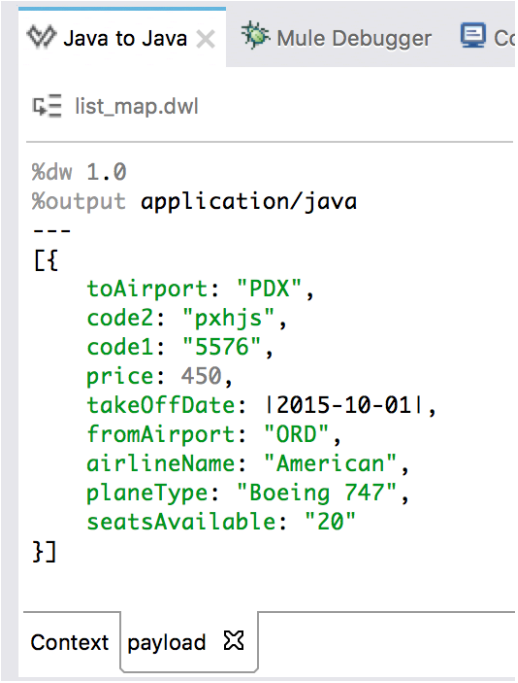
6. In the Transform section, leave the output set to Payload and of type application/java.
7. Delete the existing DataWeave expression (the curly braces) and set it to payload.

8. 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.

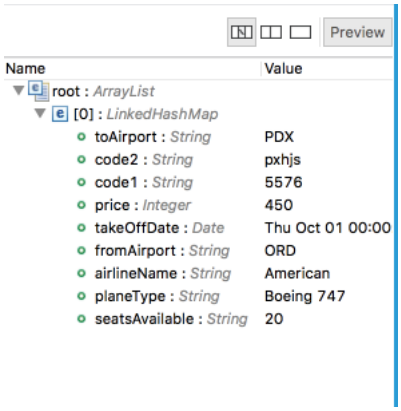


Add sample data and preview

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.

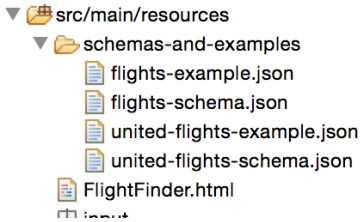


12. Look at the Preview tab in the Output section; you should see sample values.

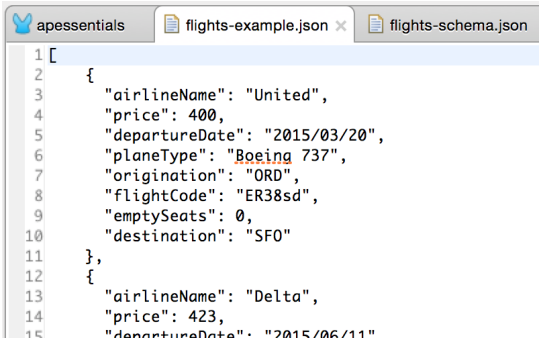


Look at the flights schema and example files

13. On your computer, navigate to the resources folder in the student files folder for the course: APEssentials3.8_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.



Change the output structure

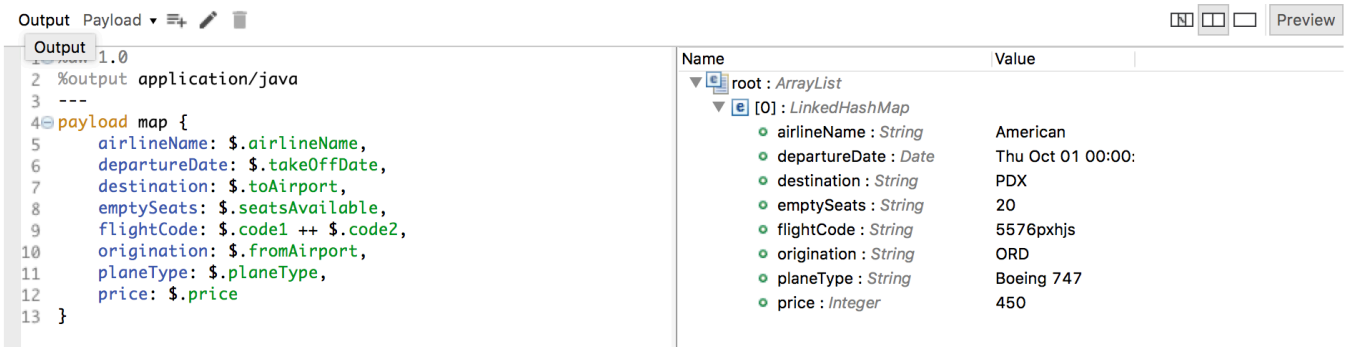
16. Return to getAmericanFlightsFlow and change the DataWeave expression to payload map { }.

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

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

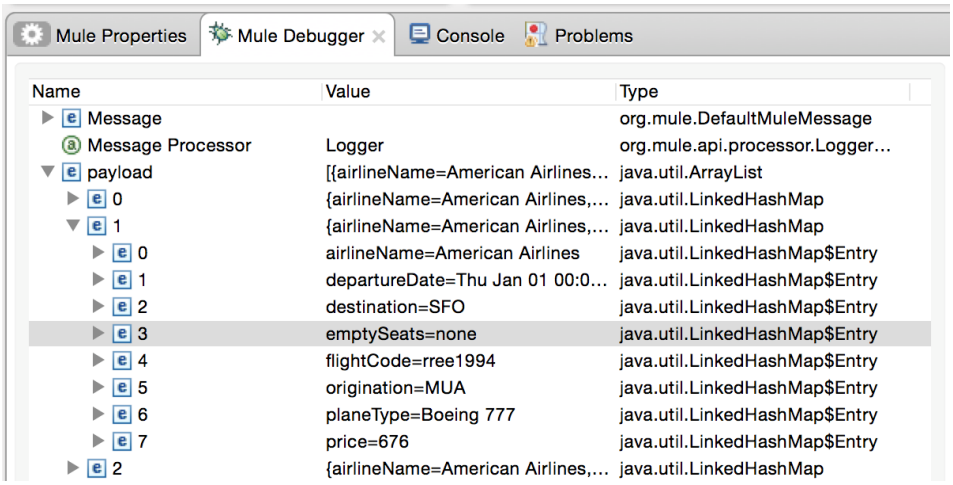
Note: You can also copy this code from the course snippets.txt file.



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.



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.

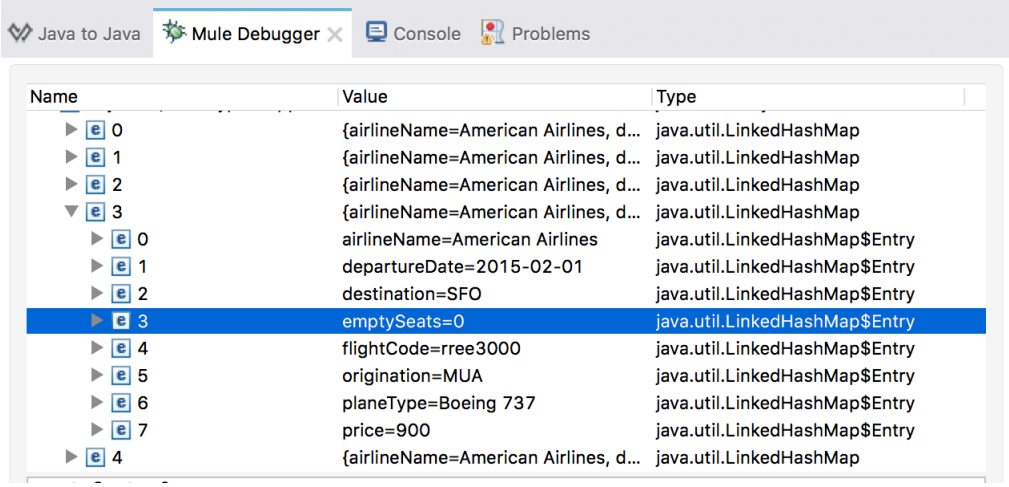
25. In the DataWeave expression, add logic to set the emptySeats field to a number unless it is equal to "none" in which case set it to 0.

```
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.

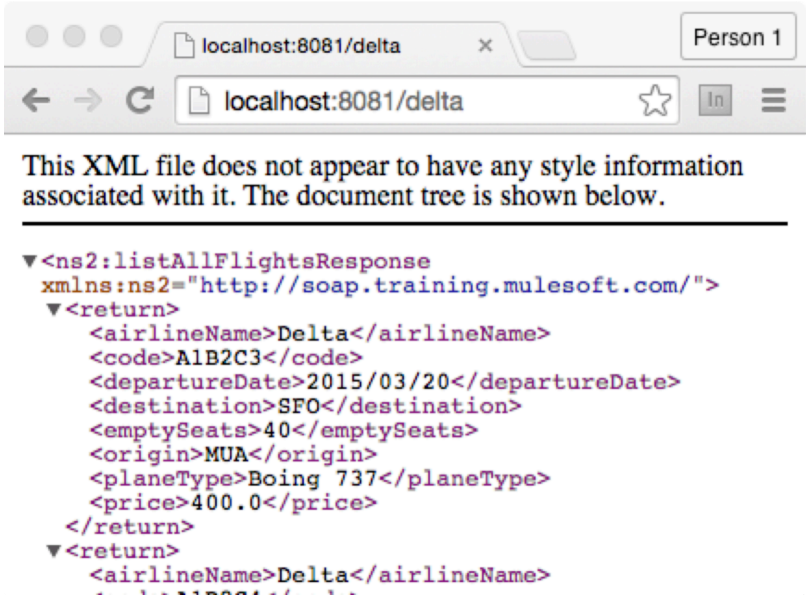
29. Drill-down into the data for the first and second flights in the payload and look at the data types and values and make sure emptySeats and price are set correctly.



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.

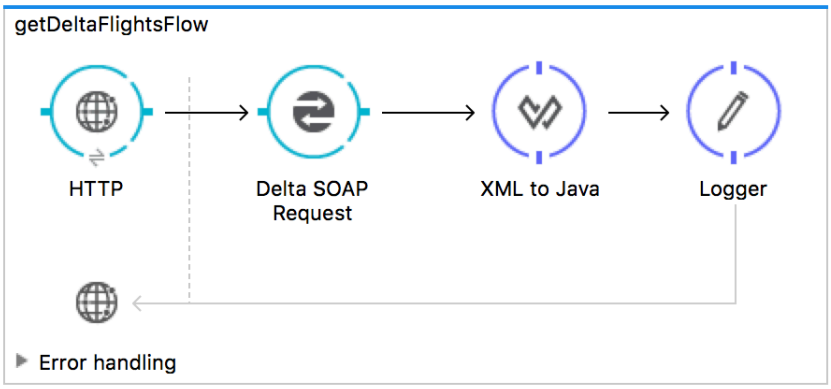


Review at the flights schema and example files

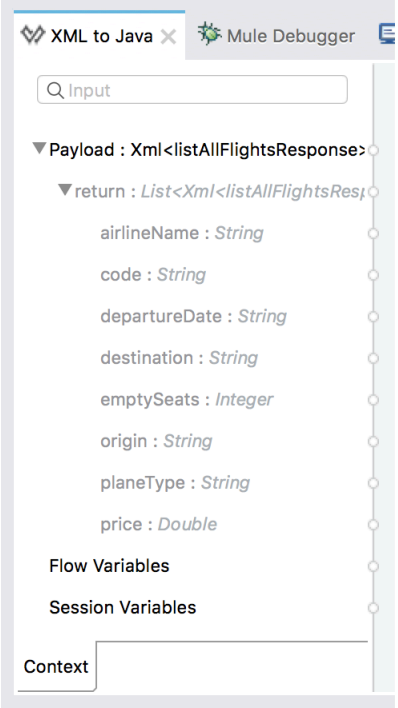
34. Return to or open the flights-schema.json and flights-example.json files and examine the code.

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.

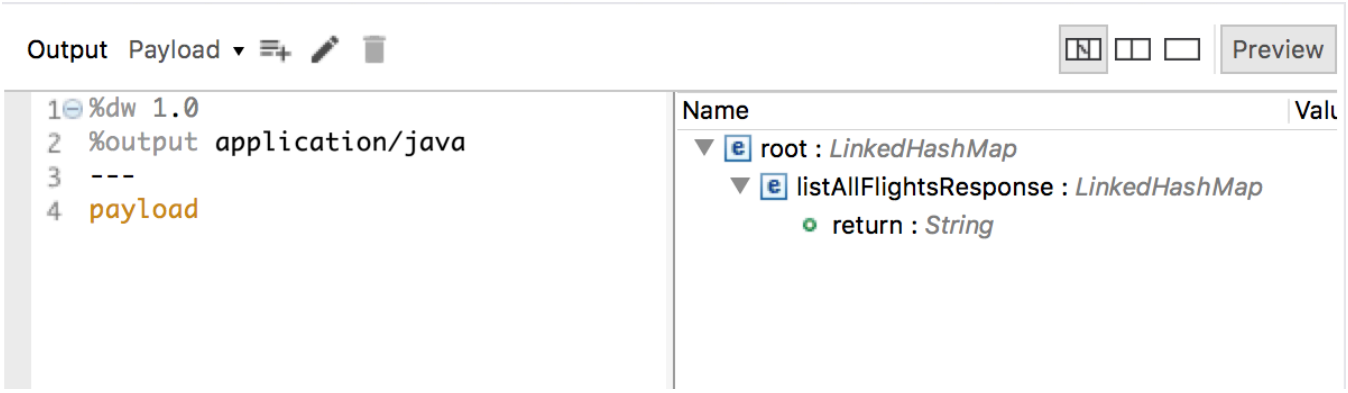


39. Look at the Input section of the XML to Java Properties view; the payload should automatically be set to a Xml<listAllFlightsResponse> with correct properties because the Web Service Consumer connector configuration is DataSense enabled by default.



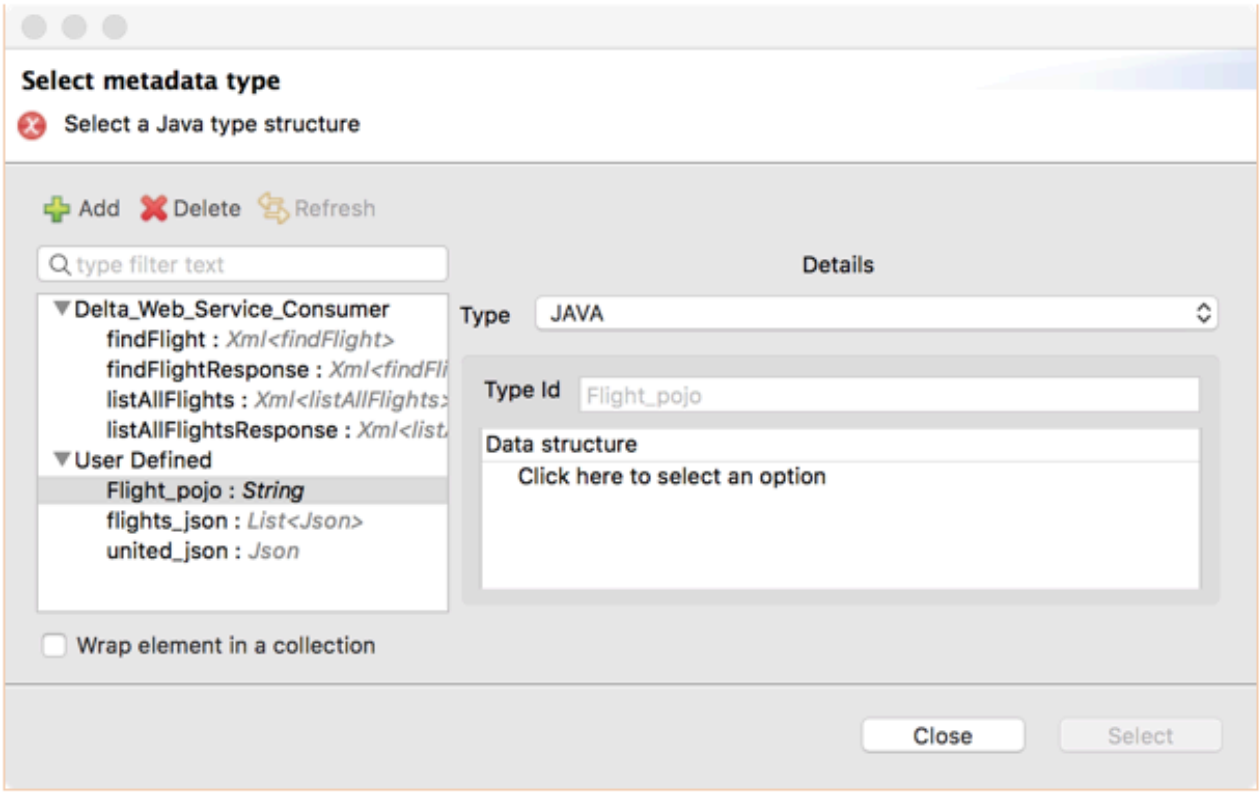
40. In the Transform section, leave the output set to Payload and of type application/java.
41. Delete the existing DataWeave expression (the curly braces) and set it to payload.

42. In the Preview tab, notice that the listAllFlightsResponse returns a LinkedHashMap.

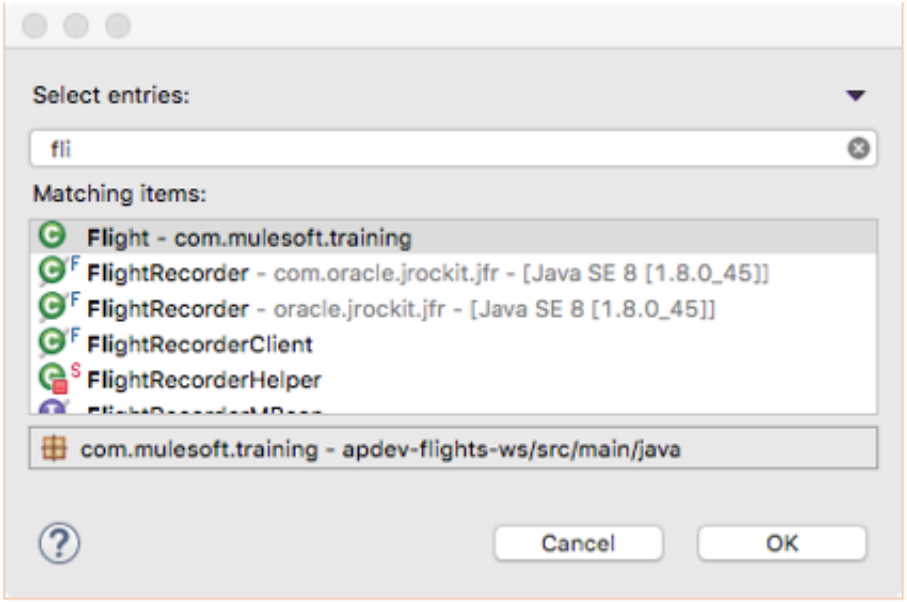


Change the output structure

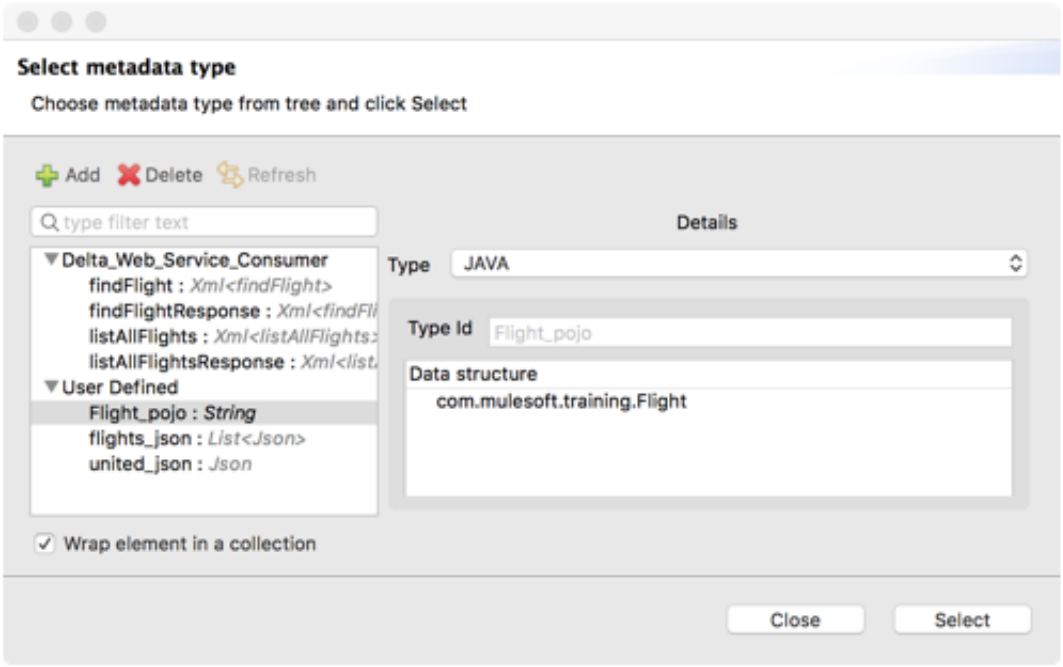
43. In the Output section, select Define Metadata link.
44. In the Select metadata type dialog box, click the Add button.
45. In the Create new type dialog box, set the name to Flight_pojo and click Create type.
46. In the Select metadata type dialog box, change the type to JAVA.
47. In the Data structure section, click the Click here to select an option link.



48. In the drop-down menu that appears, select Java object.
49. In the dialog box that opens, type fli and then in the list of classes that appears, select Flight – com.mulesoft.training.com.



50. Click OK.
51. In the Select metadata type dialog box, select Wrap element in a collection in the lower-left corner.

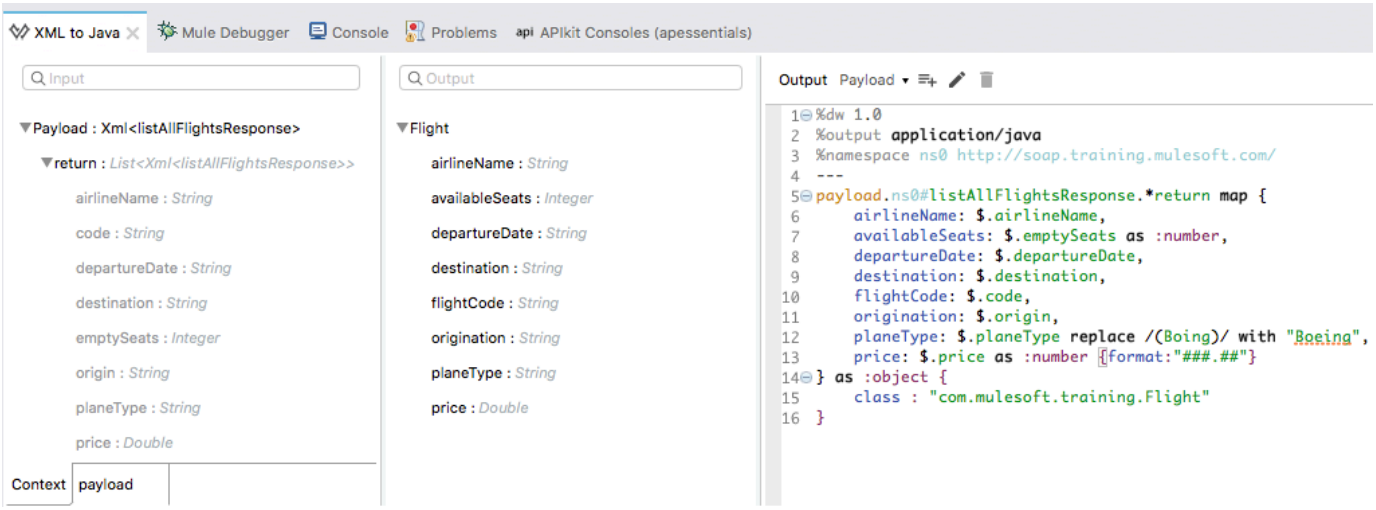


52. Click Select.
53. Use the graphical editor to map the fields from the Input to the Output appropriately.

54. Modify the DataWeave expression to 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 APEssentials3.8_snippets.txt file.



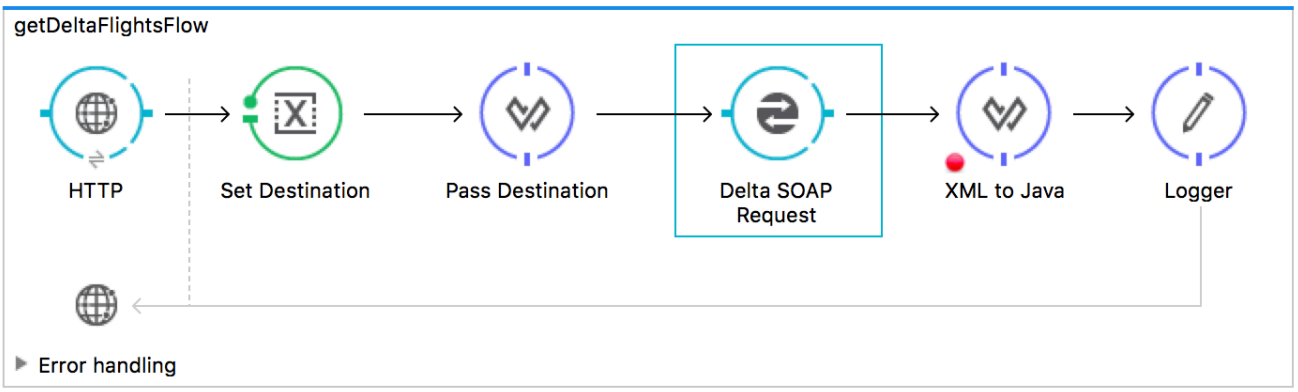
Debug the application

55. Save the file and debug the application.
56. Make a request to http://localhost:8081/delta.
57. Watch the payload value and step through to the Logger.
58. Drill-down into the data and make sure it all mapped correctly.
59. Stop the Mule Debugger.

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

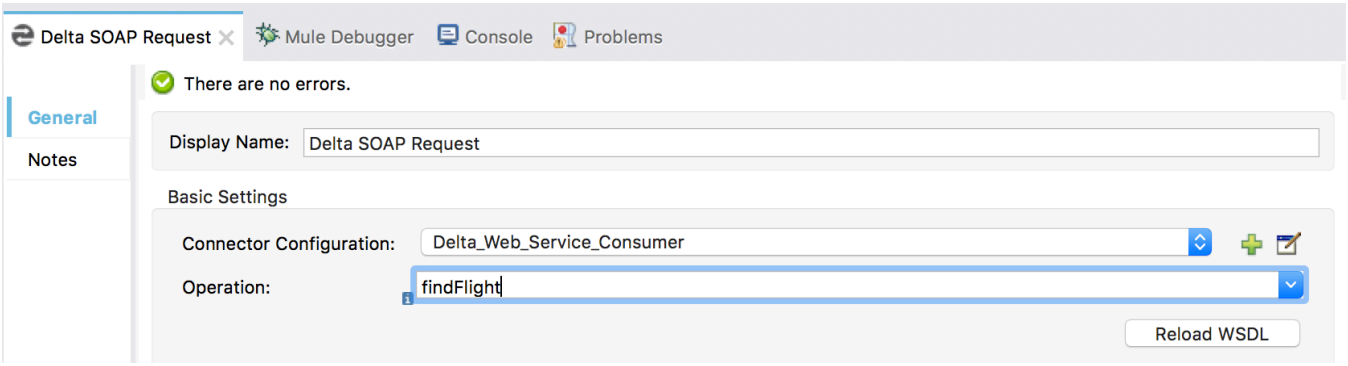
In this walkthrough, you will continue to work with the Delta flight data. You will:

- 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.

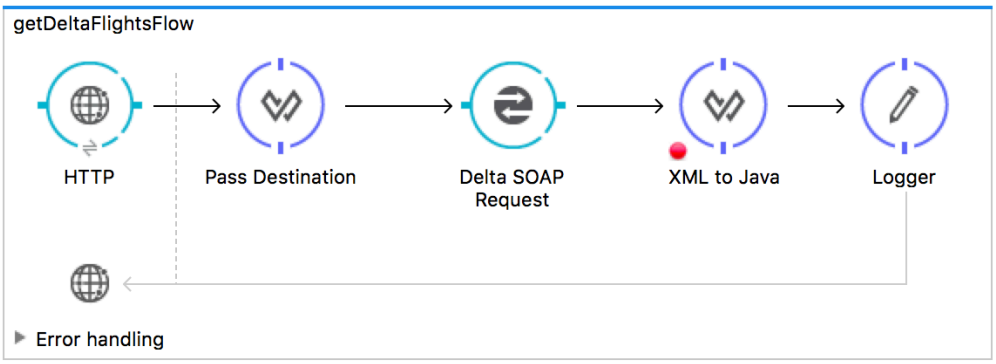
5. Make a request to http://localhost:8081/delta.
6. Return to Anypoint Studio and step through the flow; you should get an error.
7. Look at the console; you should a SoapFaultException because you are calling an operation that requires parameters but you have not passed it any.

```
ERROR 2015-07-08 19:05:22,005 [[apessentials-mod04].HTTP_Listener_Configuration.worker.01] org.mule.exception.DefaultMessagingExce
ptionStrategy:
********************************************************************************
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
********************************************************************************
```

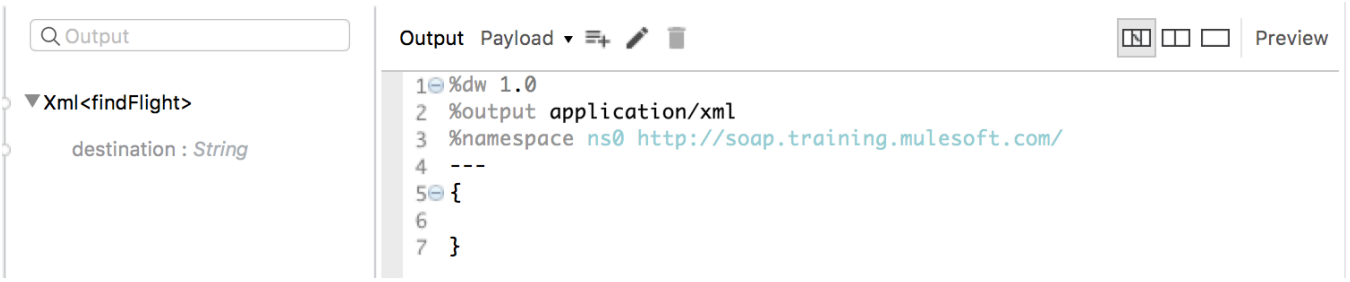
8. Stop the debugger.

Use DataWeave to pass parameters to the web service

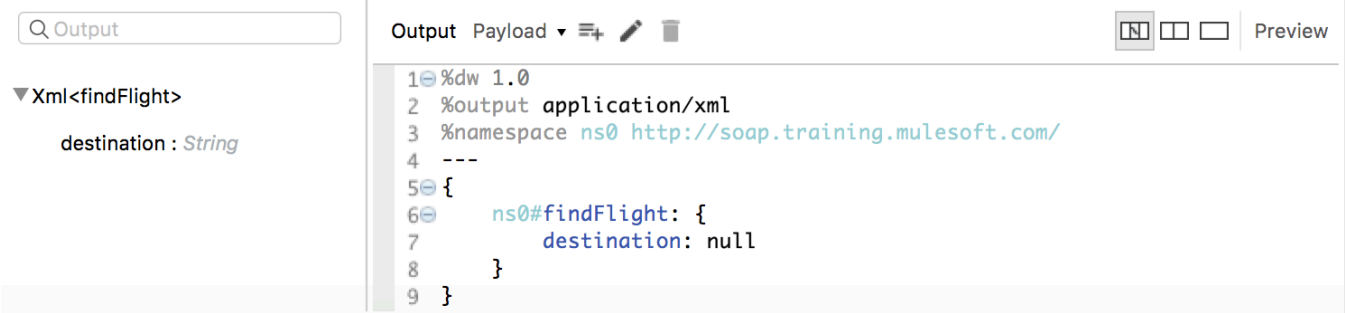
9. Add a Transform Message component to the left of the Delta SOAP Request endpoint.
10. Change its name to Pass Destination.



11. Double click on the destination: String in the Output tab inside the Pass Destination Properties view.



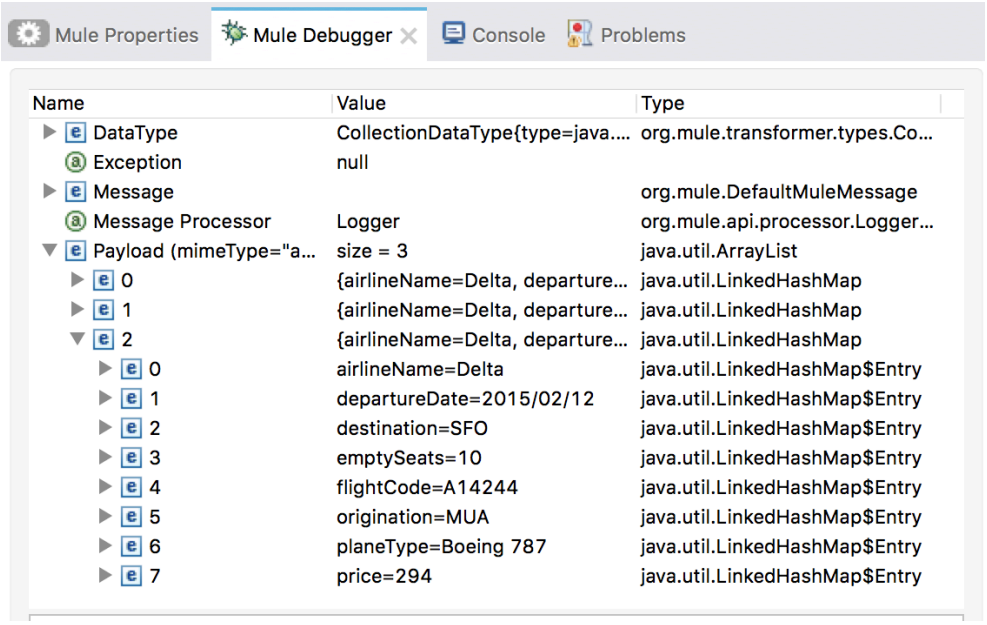
12. Observe the expression that gets added inside the Output transformation.



13. Set the destination to SFO and look at the preview.

Debug the application

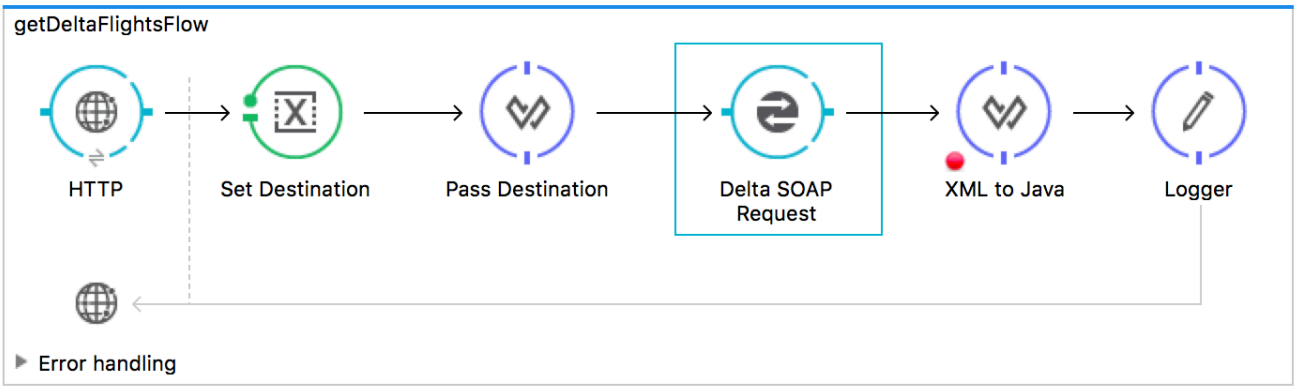
14. Save the file and debug the application.
15. Make a request to http://localhost:8081/delta.
16. In the Mule Debugger, step through to the Logger; you should now see only the SFO flights.



17. Step through the rest of the flow.

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

18. Select the Set Destination transformer in getUnitedFlightsFlow and from the main menu, select Edit > Copy (or press Cmd+C/Ctrl+C).
19. 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.
20. Move the transformer before the Pass Destination component.
21. In the Variable Properties view, change the display name to Set Destination and review the expression.



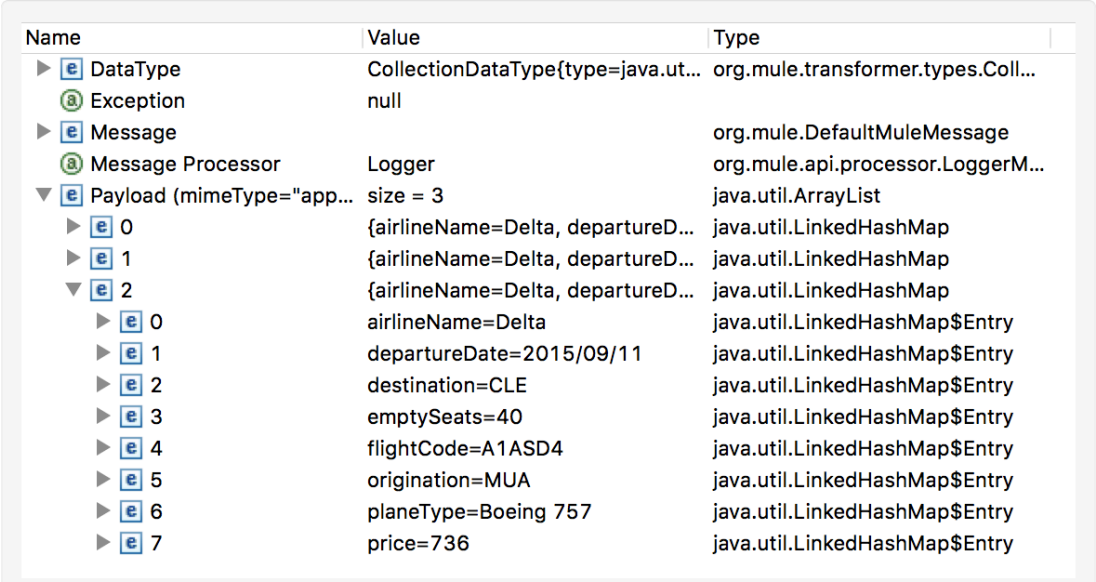
Modify the input argument to use a dynamic destination from a query parameter

22. In the Pass Destination Properties view, replace the literal of SFO with a reference to the destination flow variable.

Debug the application

23. Save the file and redeploy the application.

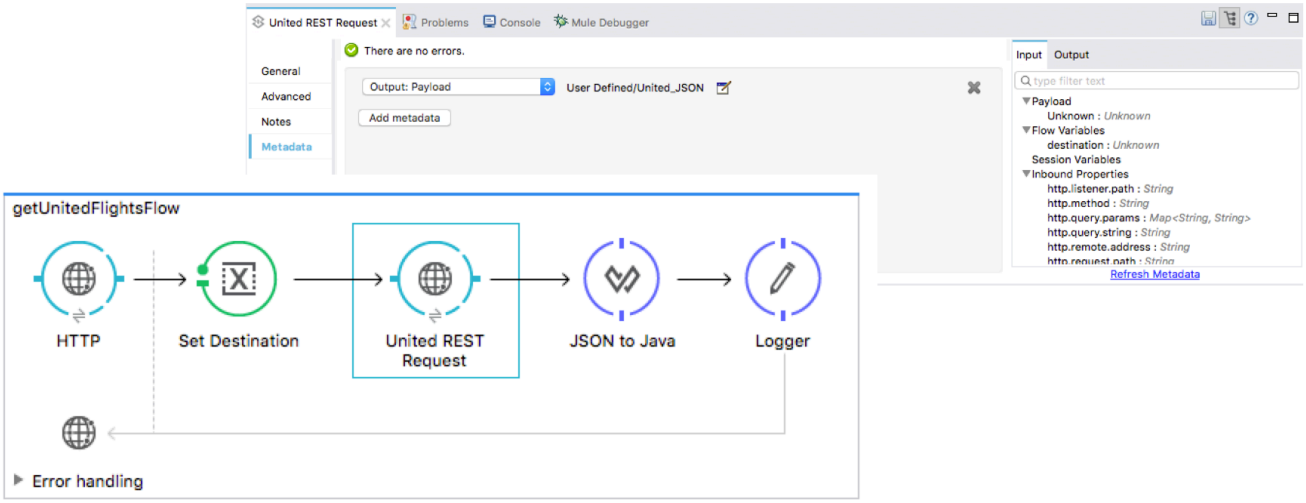
24. Make a request to http://localhost:8081/delta and step through to the Logger; you should still only see flights to SFO.
25. Make another request to http://localhost:8081/delta?code=CLE and step through to the Logger;; you should now only see flights to CLE.



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

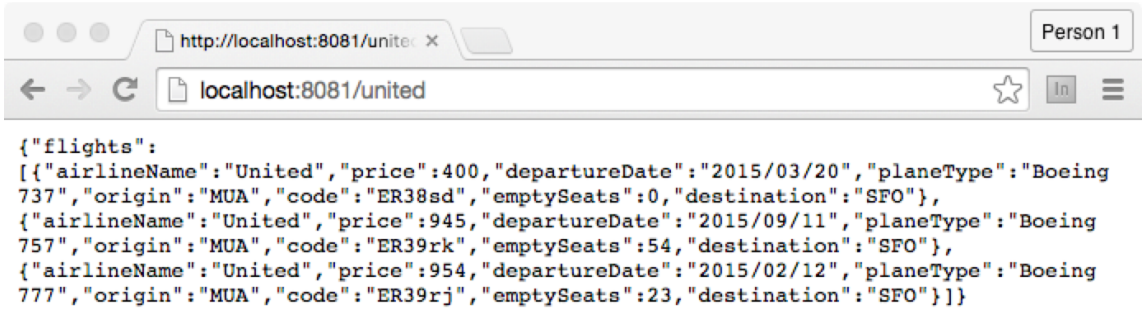
In this walkthrough, you will work with the United flight data. You will:

- 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

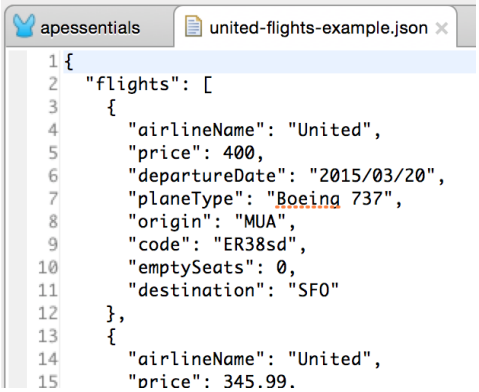
1. Run the application and make a request to http://locahost:8081/united.
2. Look at the names of the object keys.



Review at the flights schema and example files

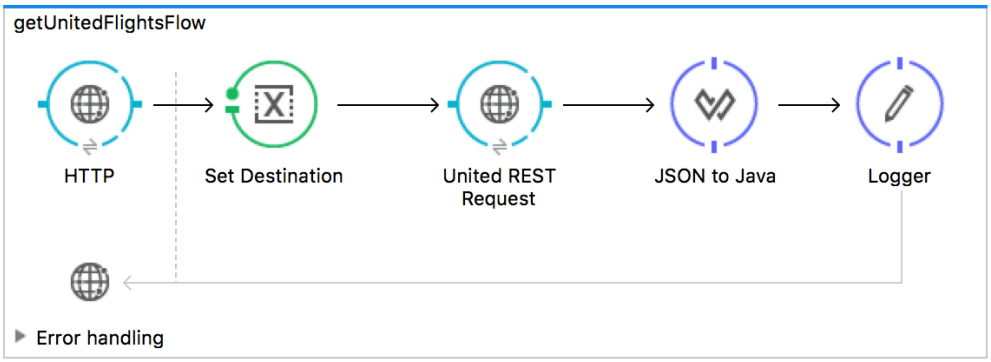
3. Open the united-flights-example.json file located in the src/main/resources/schemas-and-examples folder and examine the code.

4. Open the united-flights-schema.json file located in the src/main/resources/schemas-and-examples folder and examine the code.



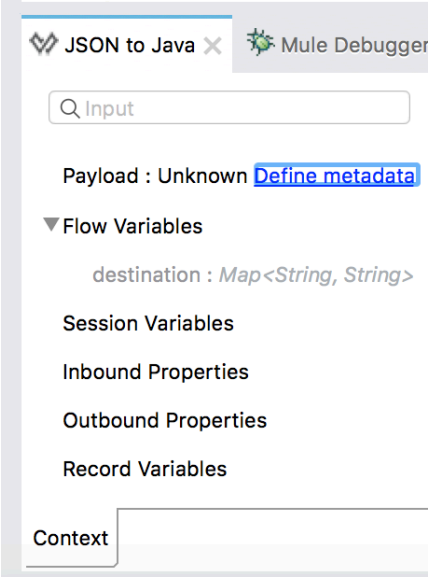
Add a DataWeave Transform Message component

5. Locate getUnitedFlightsFlow.
6. Add a Transform Message component after the United REST Request.
7. Change its name to JSON to Java.
8. Add a Logger after the component.



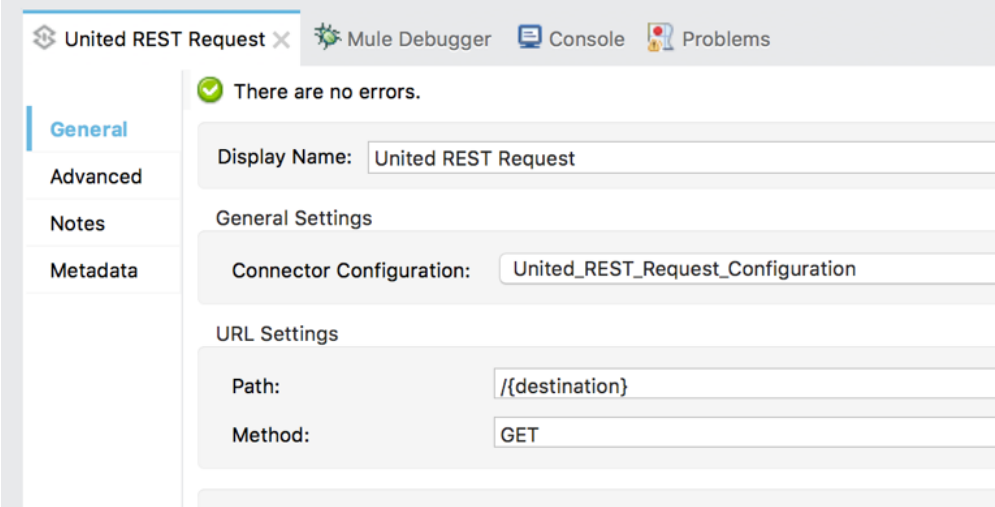
9. In the Transform Message Properties view, leave the output set to Payload and of type application/java.

10. Look at the Input section of the Transform Message Properties view; the payload is of type unknown.



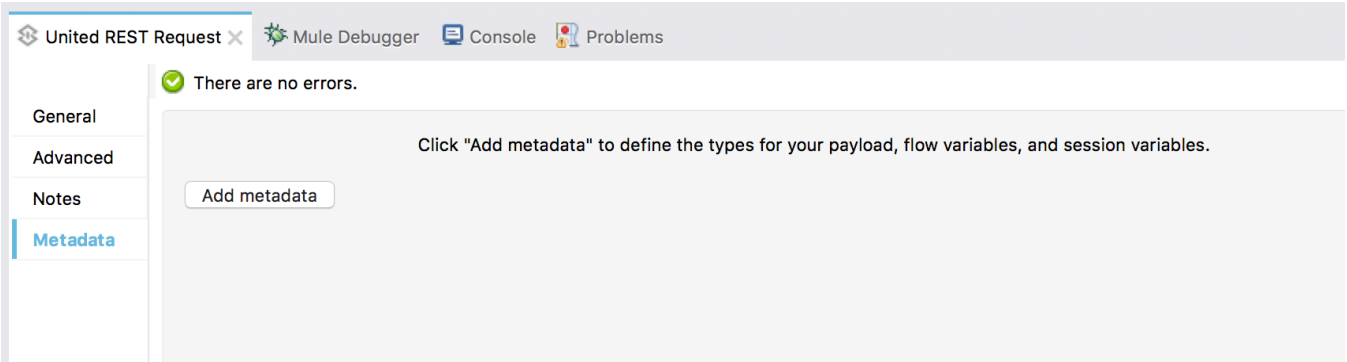
Add metadata to the HTTP Request endpoint

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.



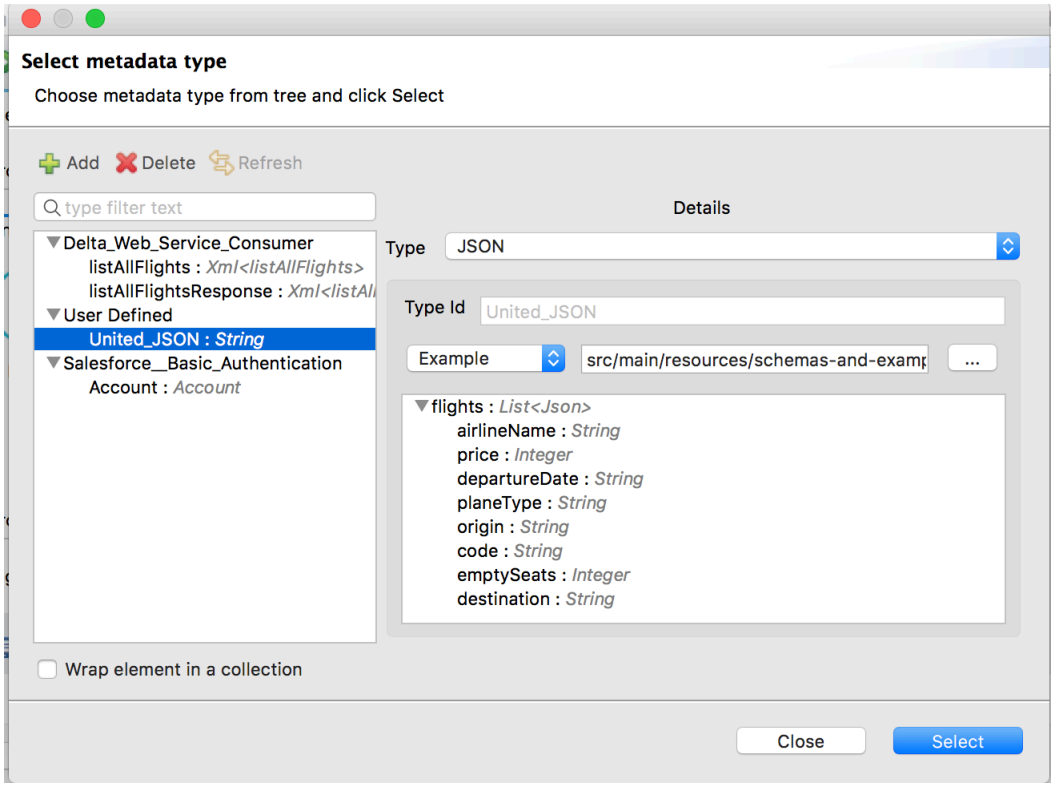
12. Click the Metadata link in the left-side navigation.

13. Click the Add metadata button.

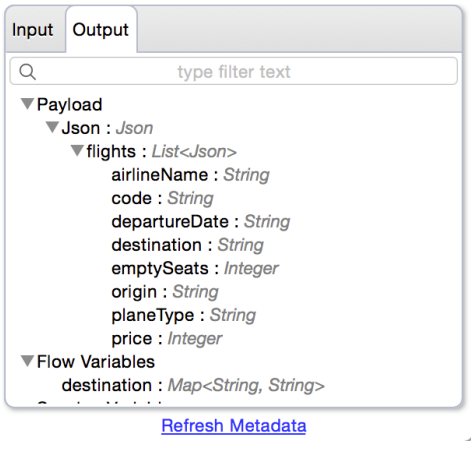


14. In the drop-down menu that appears, select Output: Payload.
15. Click the Edit button located to the right of the drop-down menu.
16. In the Select metadata type dialog box, click the Add button.
17. Set the type id to United_JSON.
18. Click OK.
19. Set the Type to JSON.
20. In the drop-down menu (with the value of Schema by default), select Example.
21. Click the browse (…) button.
22. In the Open dialog box browse to src/main/resources/schemas-and-examples/united-flights-example.json and click Open.

23. In the Define Type dialog box, click Select.



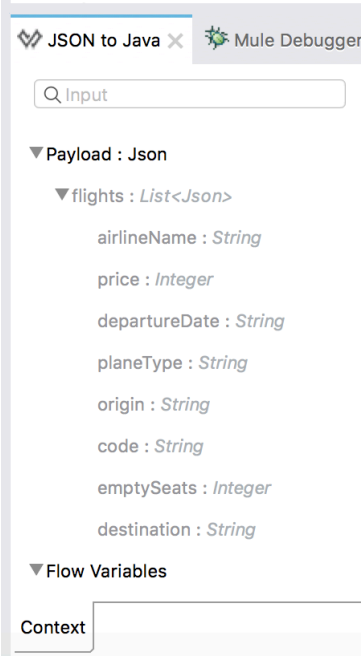
24. Look at the Output tab in the Metadata section again; you should now see the payload is of type JSON with a flights property that is a list of objects with specific fields.



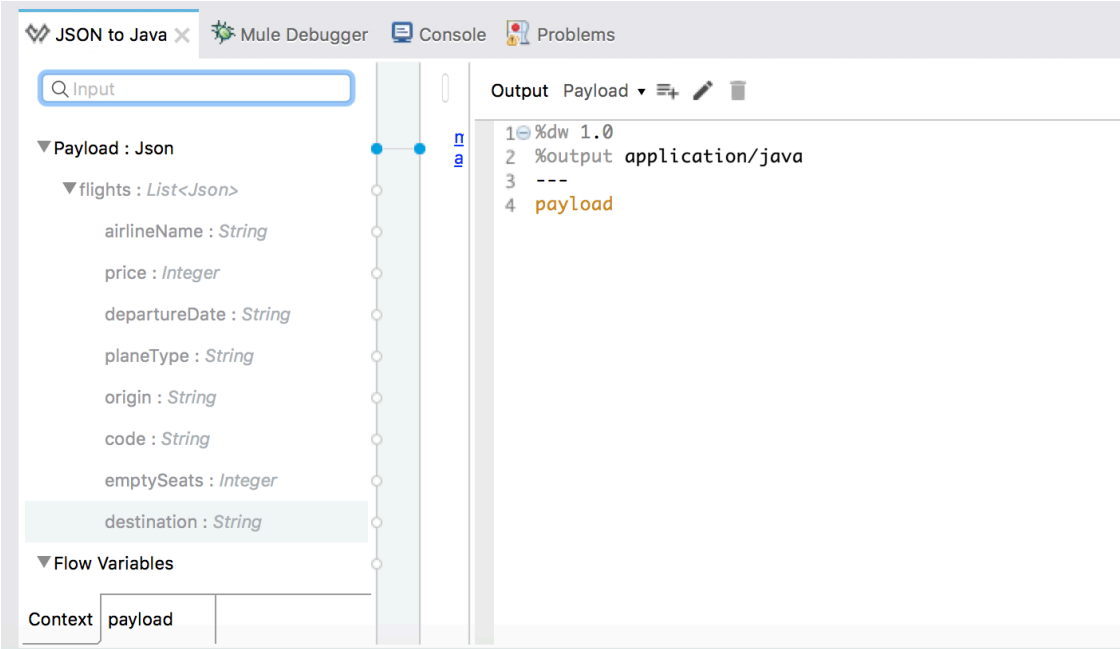
Transform the JSON to a collection of Java objects

25. Return to the JSON to Java component Properties view.
26. Look at the Input section again; the payload should now be of type Json.

27. Expand the flights List; you should see the the field names of each object.



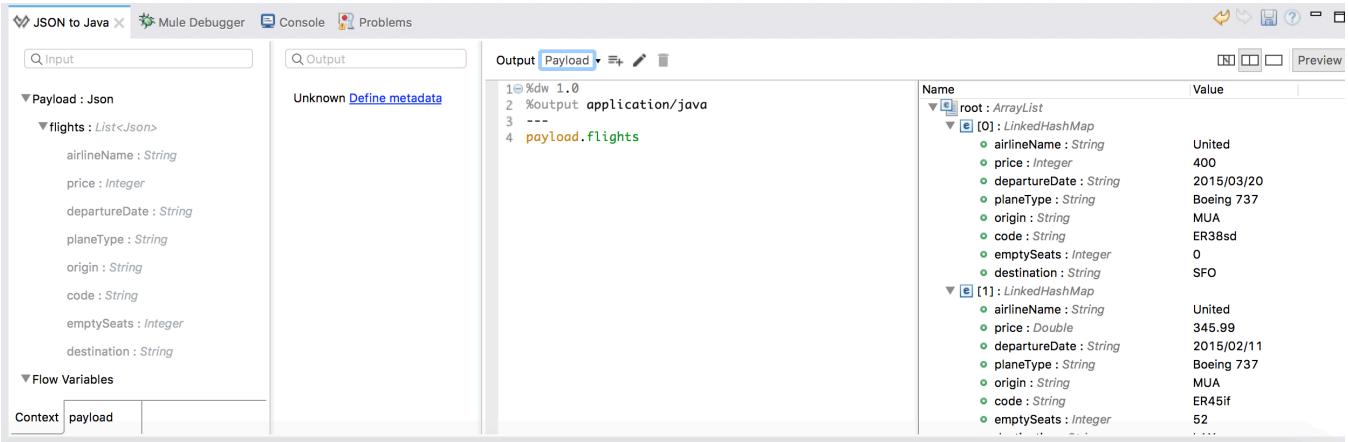
28. Delete the existing DataWeave expression (the curly braces) and set it to payload.



29. Look at the Output section; the output structure should be set to a Map.

Change the output structure

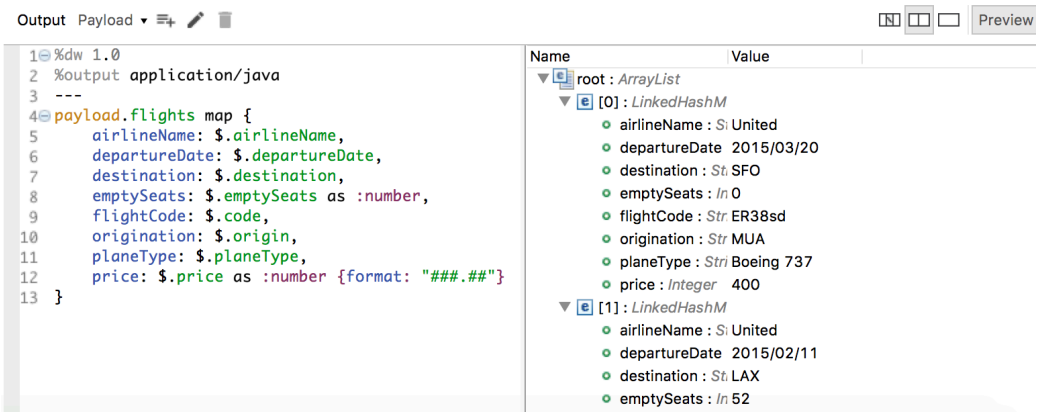
30. Change the DataWeave expression to payload.flights; the output structure should now be a List of Map objects.



31. Modify the DataWeave expression to use the map operator.
32. In the transformation function, set the following mappings:

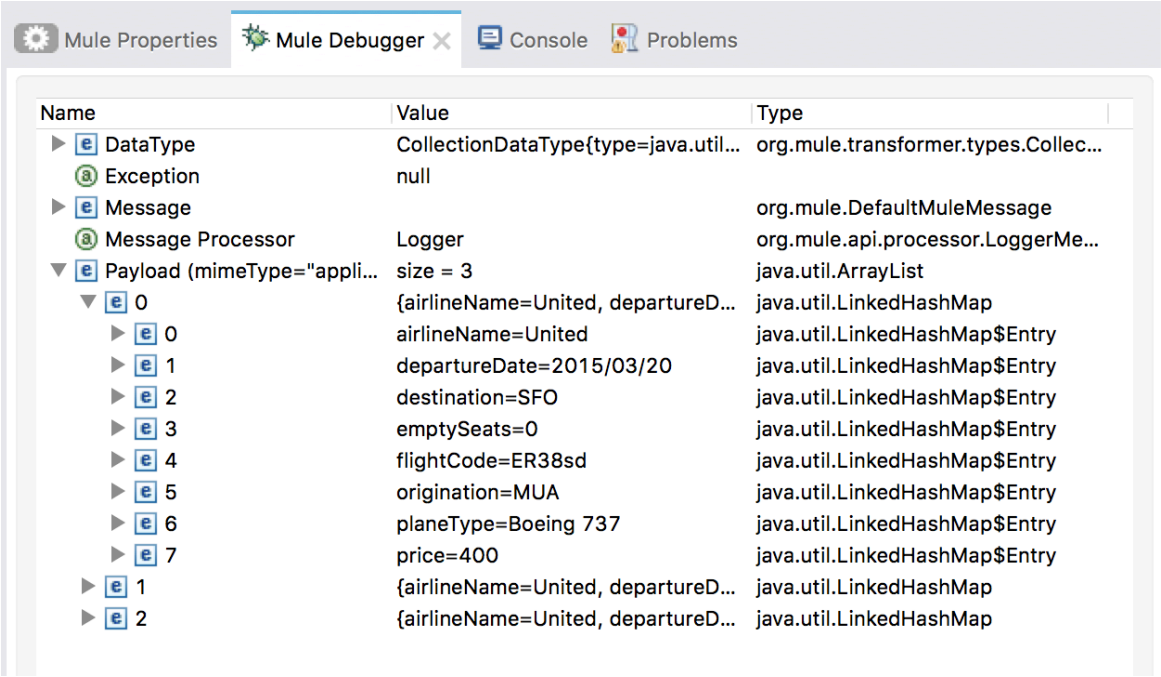
- airlineName < airlineName
- departureDate < departureDate
- destination < destination
- emptySeats < emptySeats as a number
- flightCode < code
- origination < origin
- planeType < planeType
- price < price and format it as a number with the pattern “###.##”

Note: If you want, you can copy these lines of code from the Delta DataWeave expression.



Debug the application

33. Save the file and debug the application.
34. Make a request to http://localhost:8081/united.
35. Watch the payload value and step through to the Logger.
36. Drill-down into the data and make sure it all mapped correctly.
37. Stop the Mule Debugger.



Module 6: Refactoring Mule Applications

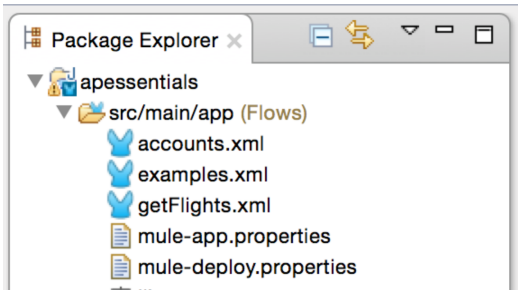
In this module, you will learn:

- 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

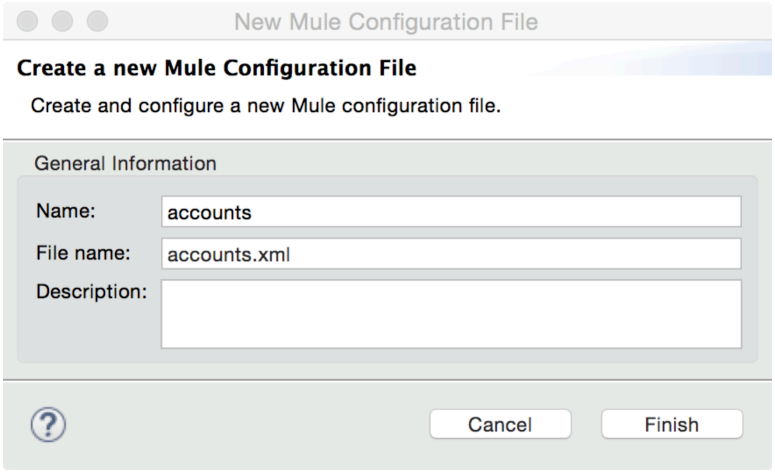
In this walkthrough, you will refactor your monolithic apessentials application. You will:

- 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

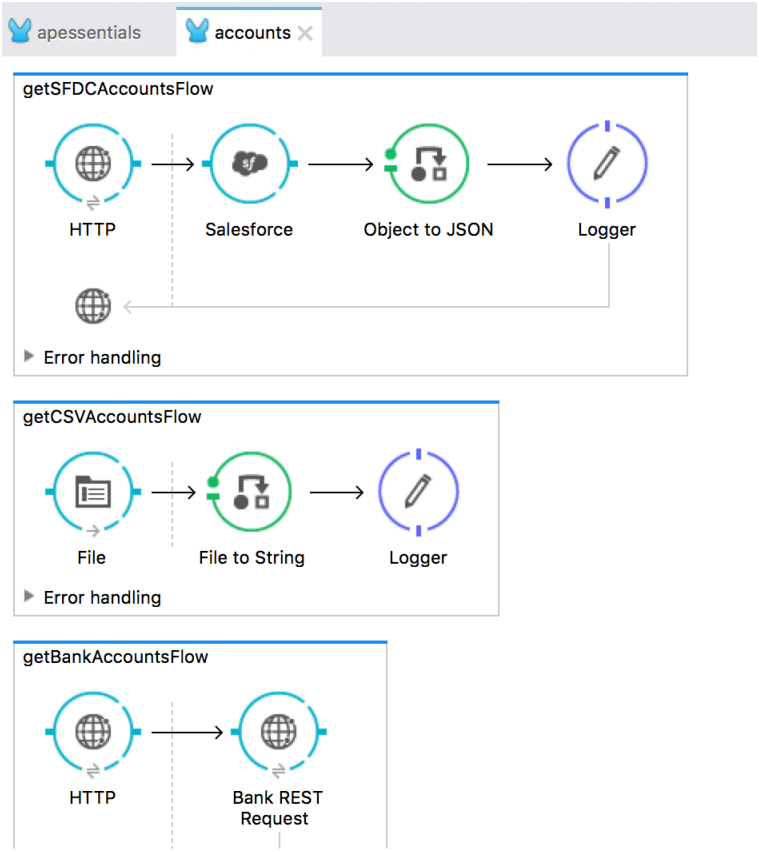
1. Right-click the apessentials project in the Package Explorer and select New > Mule Configuration File.
2. Set the name to accounts and click Finish.



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.

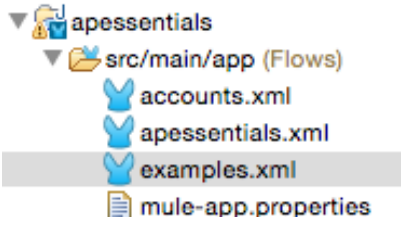
8. Switch to the Message Flow view.



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.

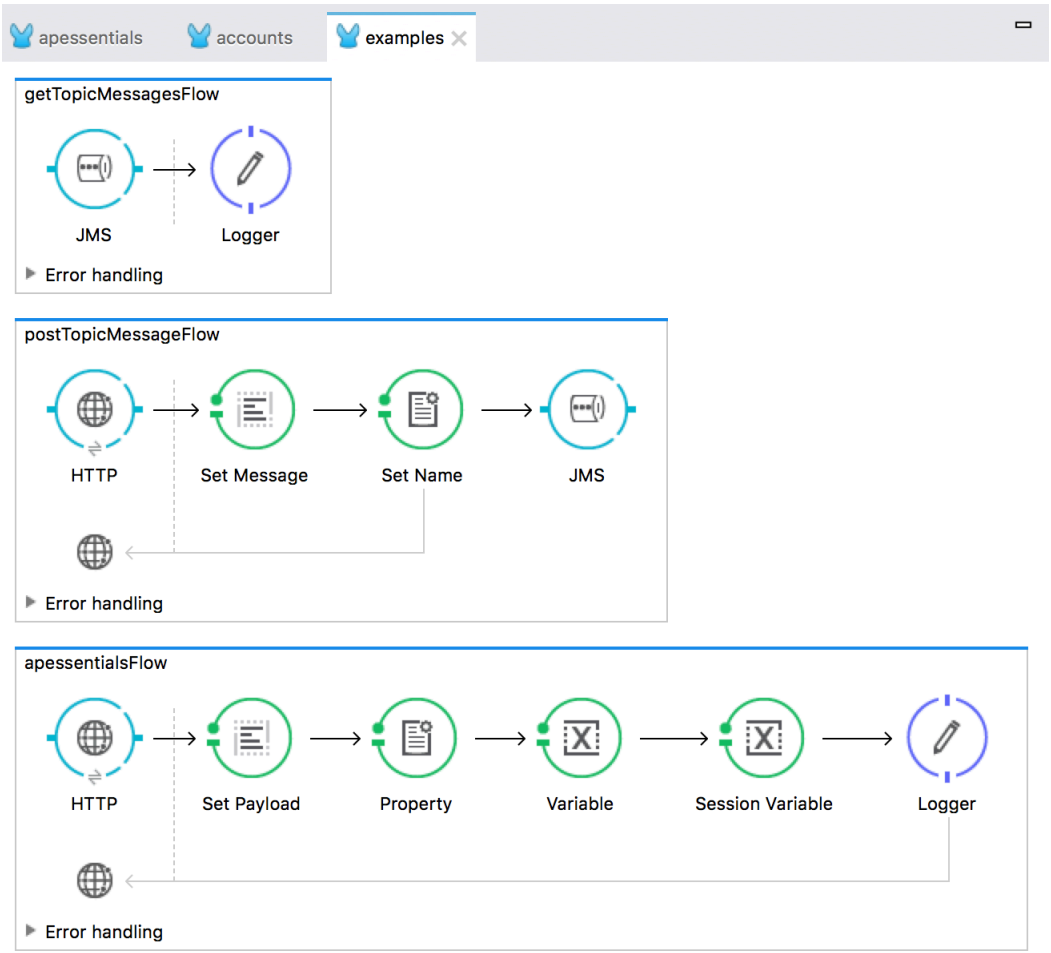


12. In examples.xml, switch to the Configuration XML view.
13. Place some empty lines between the start and end mule tags.
14. Return to apessentials.xml and select and cut apessentialsFlow, getTopicMessagesFlow, and postTopicMessageFlow.

15. Return to examples.xml and paste the flows inside the mule tags.

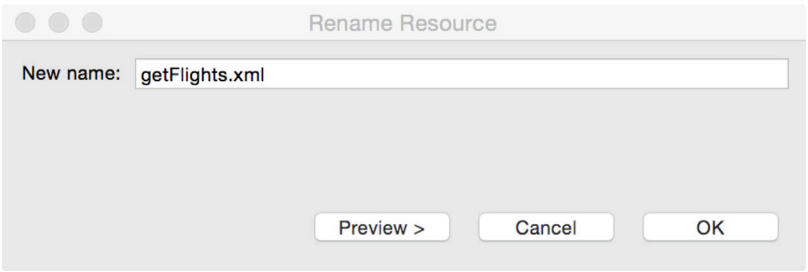
    Note: If these flows are not adjacent, you will need to repeat the process several times.

16. Switch to the Message Flow view.
17. Save all the files.

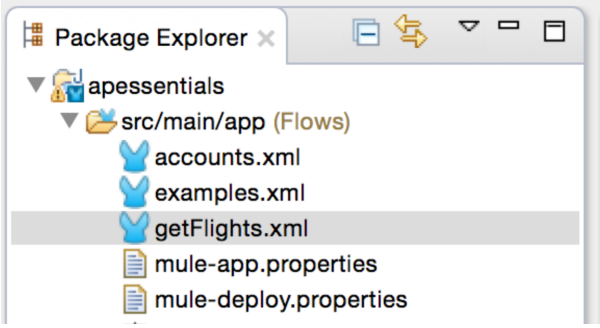


Rename the apessentials configuration file

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.

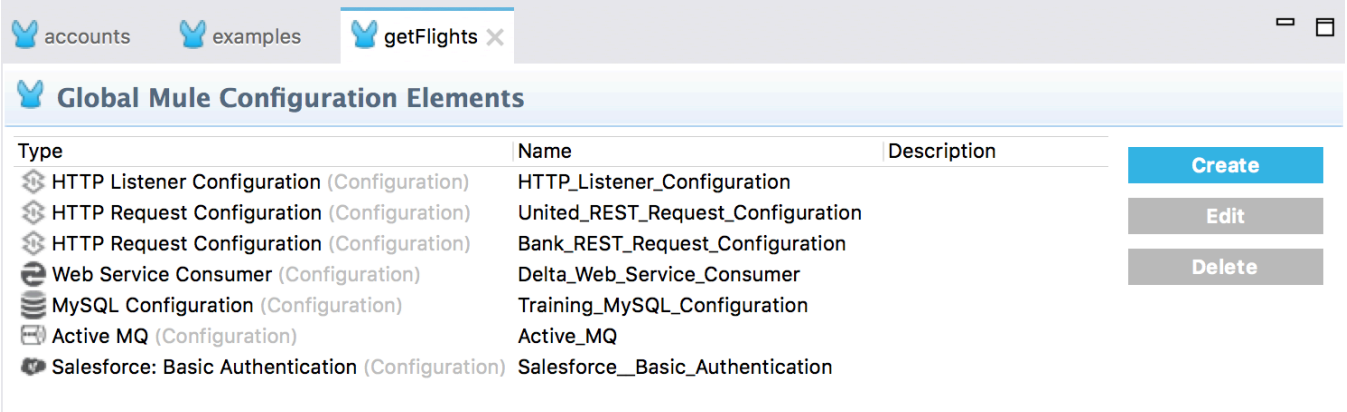


20. Double-click getFlights.xml in the Package Explorer to reopen it.



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.



Test the application

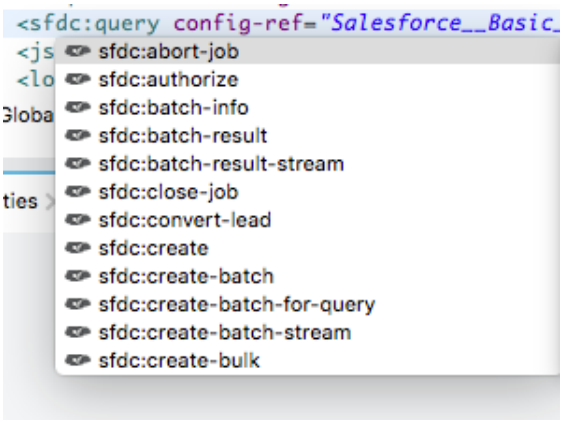
23. Run the application; you should get an error that the prefix for sfdc:query is not bound.

```
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.

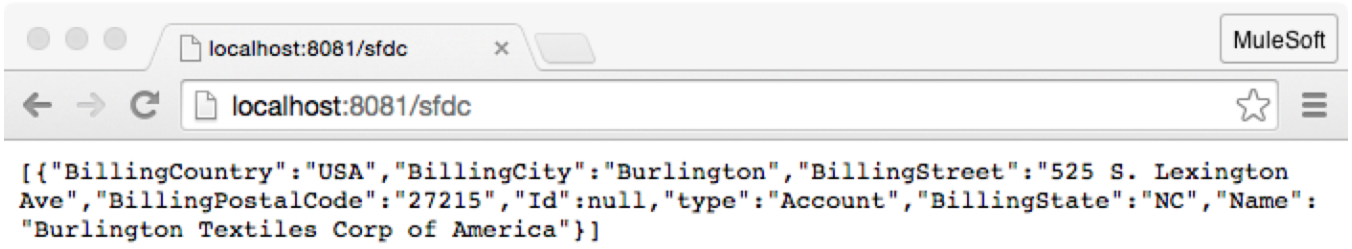


27. In the autocomplete pop-up, select sfdc:query and press Enter.
28. Delete the new config-ref attribute that was added.
29. Locate the new xmlns and schemaLocation for sfdc.

```
3  <mule xmlns:sfdc="http://www.mulesoft.org/schema/mule/sfdc" xmlns:file="
4      xmlns:http="http://www.mulesoft.org/schema/mule/http" xmlns:json="ht
5      xmlns="http://www.mulesoft.org/schema/mule/core" xmlns:doc="http://w
6      xmlns:spring="http://www.springframework.org/schema/beans" version="
7      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
8      xsi:schemaLocation="http://www.mulesoft.org/schema/mule/sfdc http://
9  http://www.mulesoft.org/schema/mule/file http://www.mulesoft.org/schema/
```

Test the application

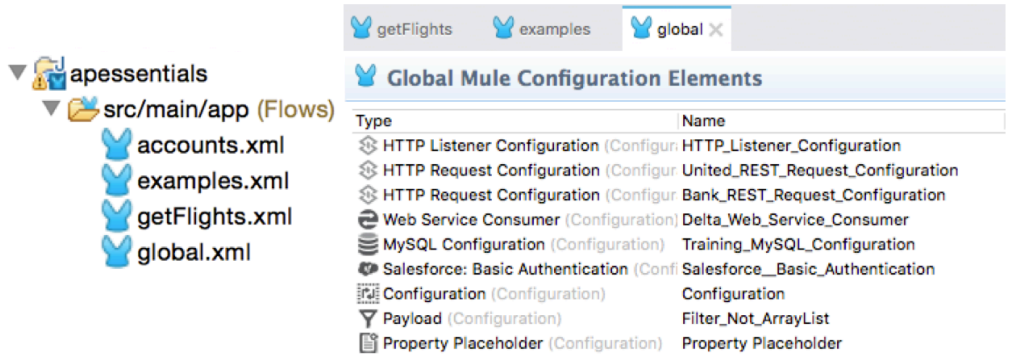
30. Save all the files and run the application.
31. Make a request to http://localhost:8081/sfdc; you should now get account results.



Walkthrough 6-2: Encapsulate global elements in a separate configuration file

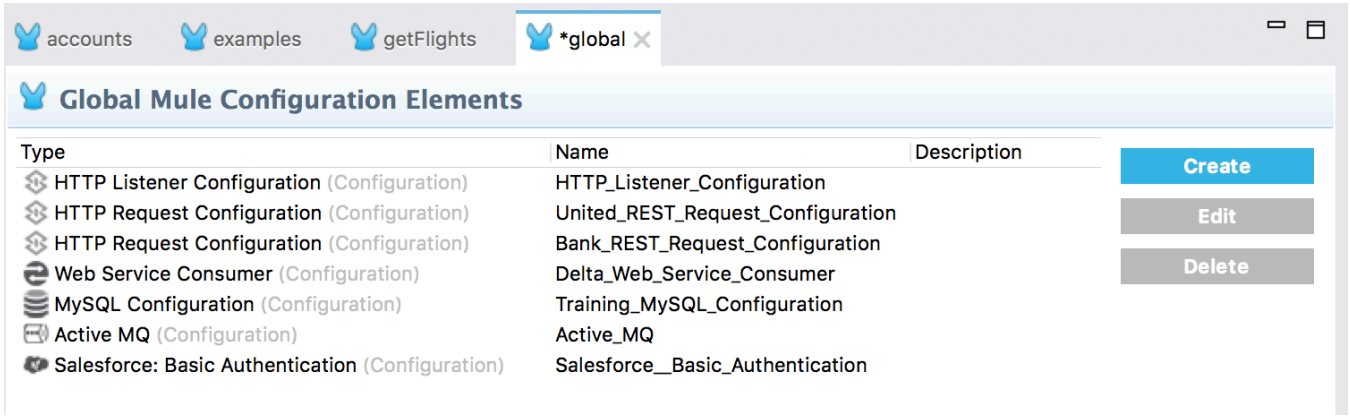
In this walkthrough, you will separate global elements into a new configuration file. You will:

- 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.

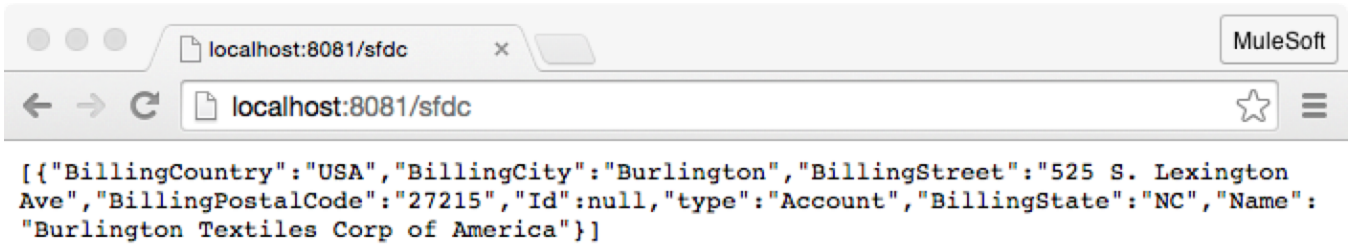
10. Select and delete the eight global elements defined before the flows.

    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

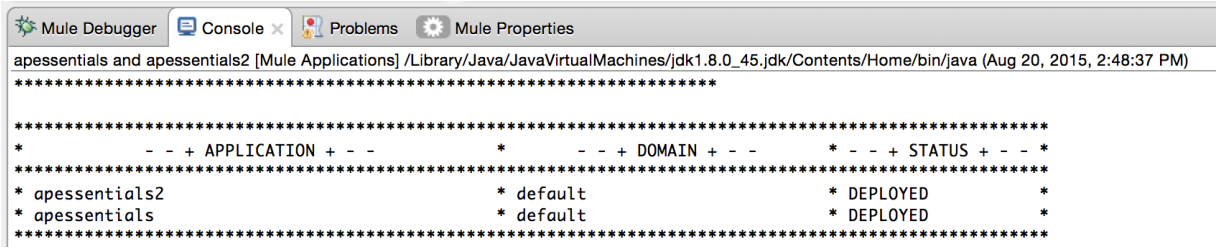
14. Run the application.
15. Make a request to http://localhost:8081/sfdc; you should still get account results.



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.

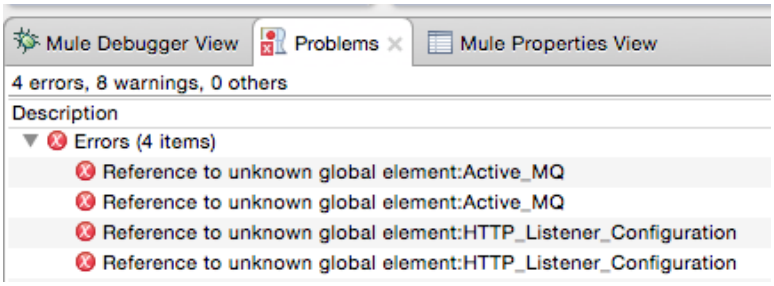


Create a new project and application

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

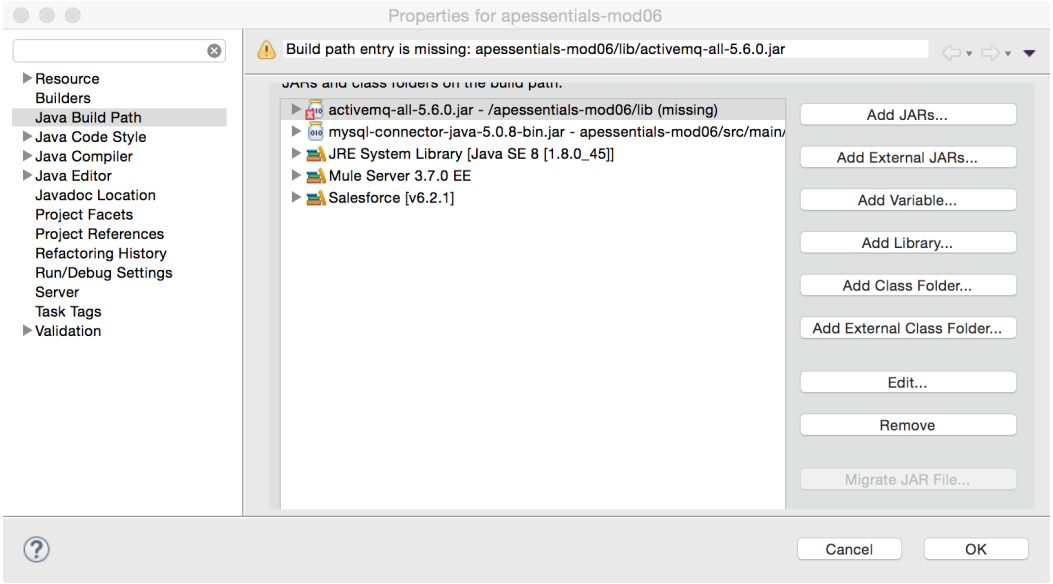
5. Drag examples.xml from the apessentials project to the src/main/app folder of the apessentials2 project; you should get unknown global element errors.



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.

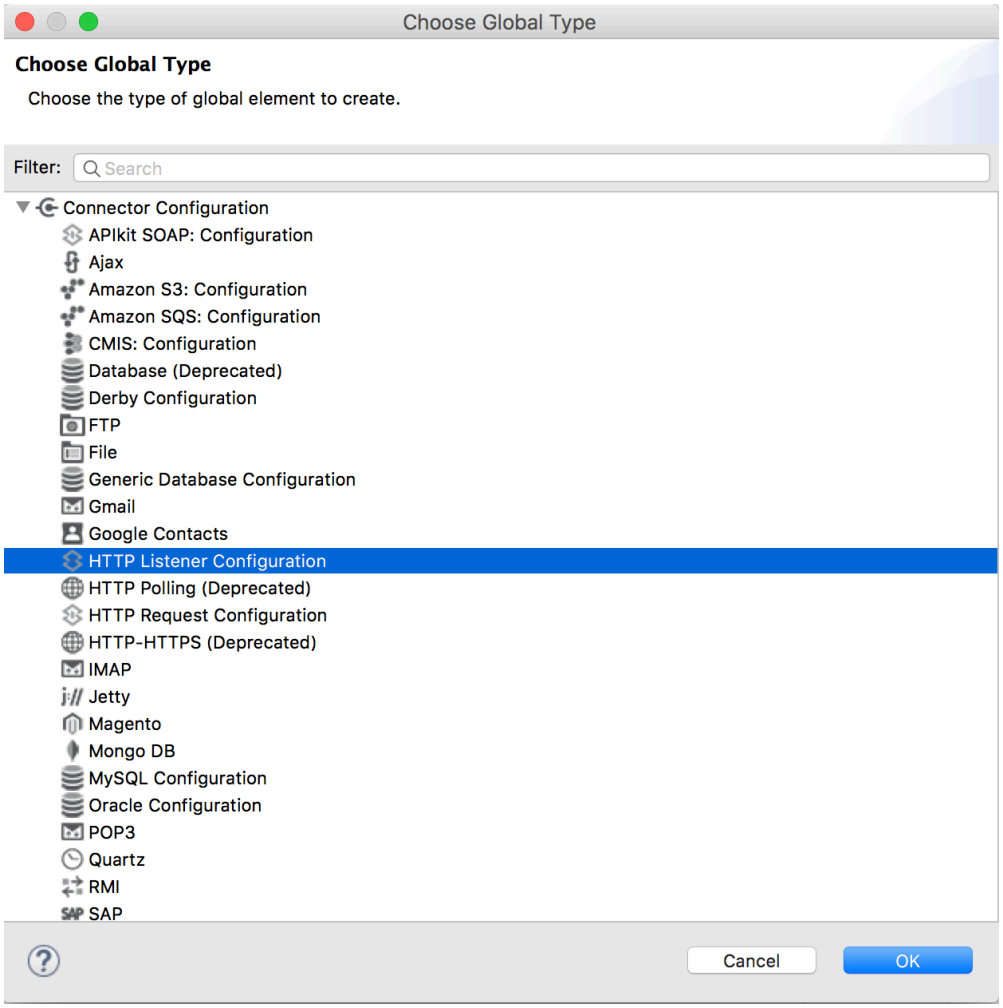


16. Click OK.

Create new global elements

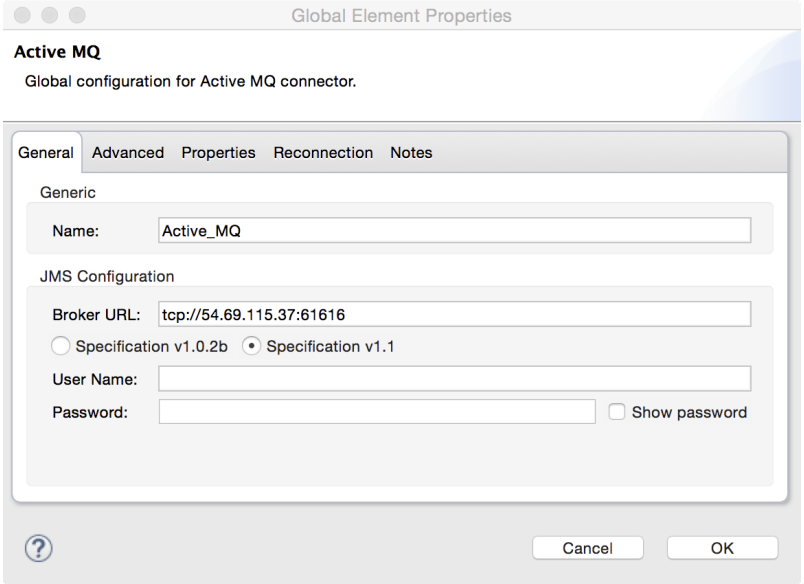
17. Right-click the apessentials2 project and select New > Mule Configuration File.
18. In the New Mule Configuration File dialog box, set the name to global and click Finish.
19. In global.xml, switch to the Global Elements view.
20. Click the Create button.

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.

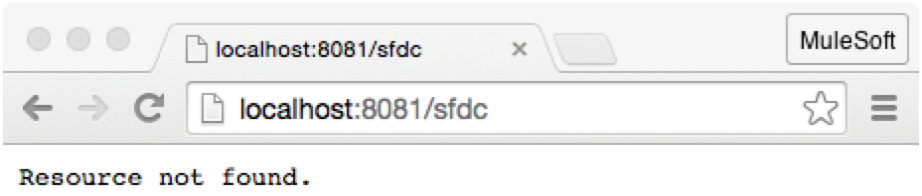


Note: You could also copy and paste these definitions from the apessentials global.xml file.

26. Save all the files.

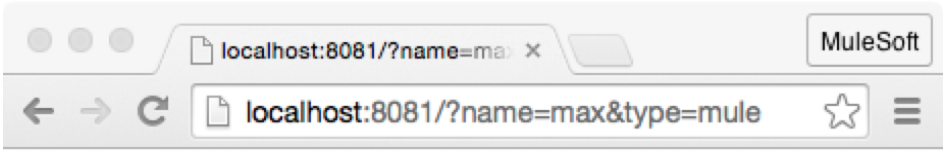
Run the new application

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.




```
Resource not found.
```

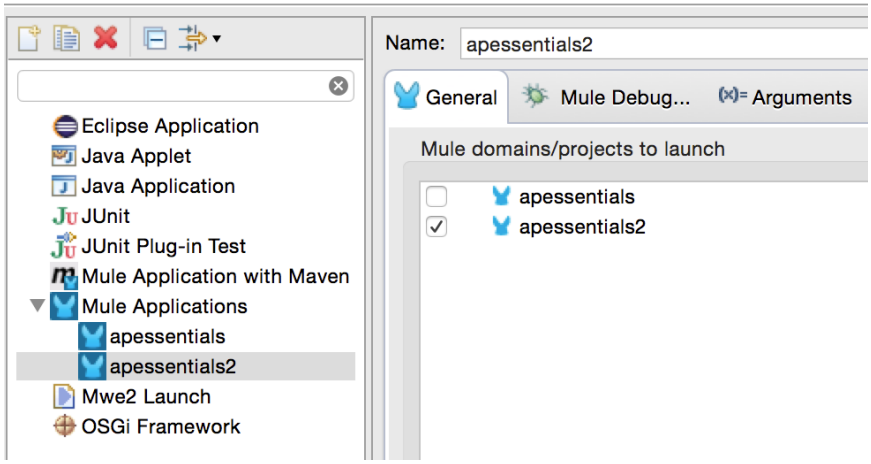
29. Make a request to http://localhost:8081/?name=max&type=mule using your own query parameter values; you should get HELLO WORLD.



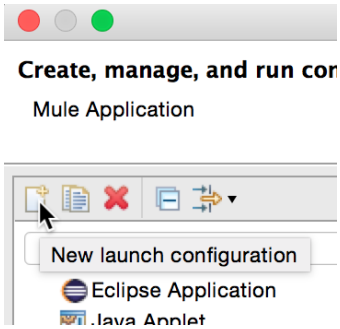

```
HELLO WORLD
```

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.

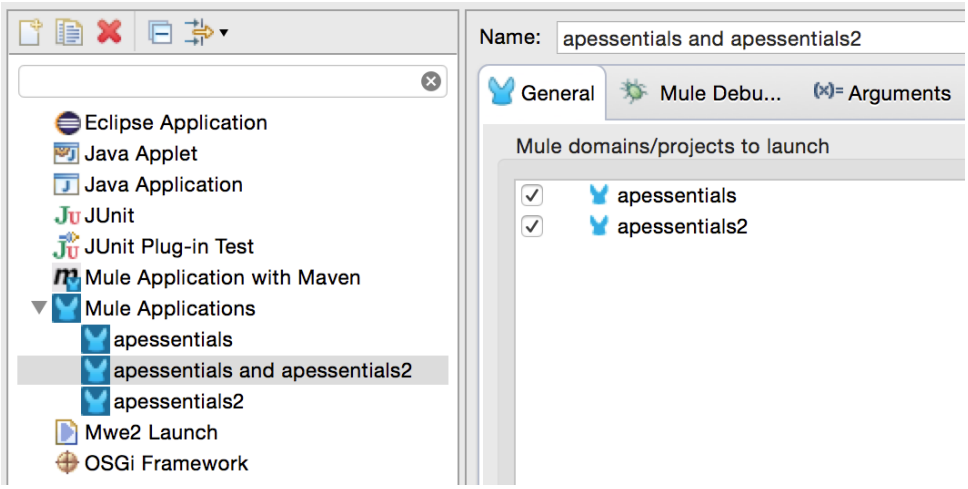


34. Click the New Launch Configuration button.



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.

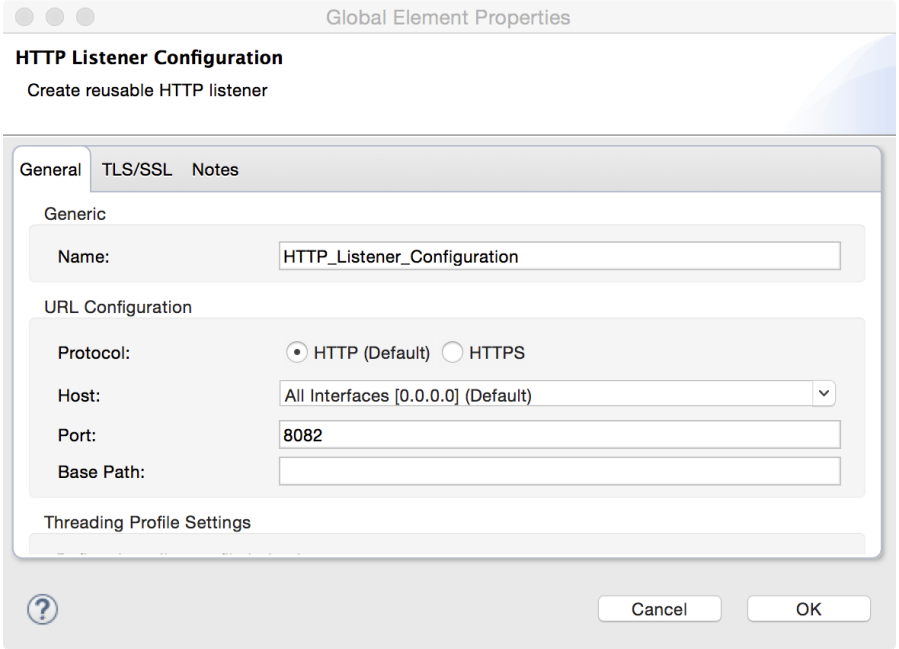
```
**********************************************************************************************************
*            - - + APPLICATION + - -            *       - - + DOMAIN + - -       * - - + STATUS + - - *
**********************************************************************************************************
* 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.DefaultM
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.

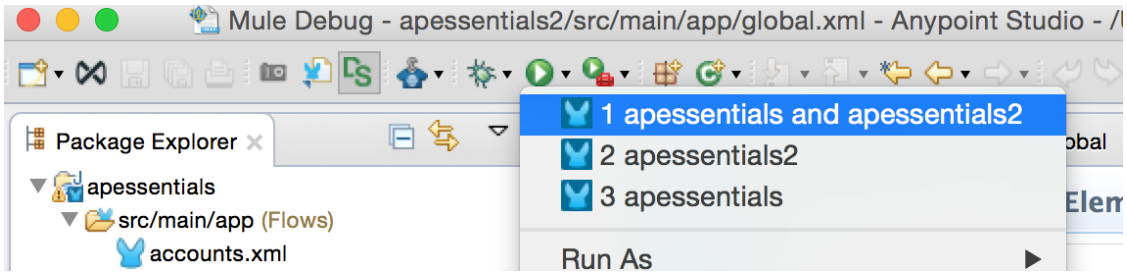


43. Save the file; both applications should now be successfully deployed – apessentials was already deployed.

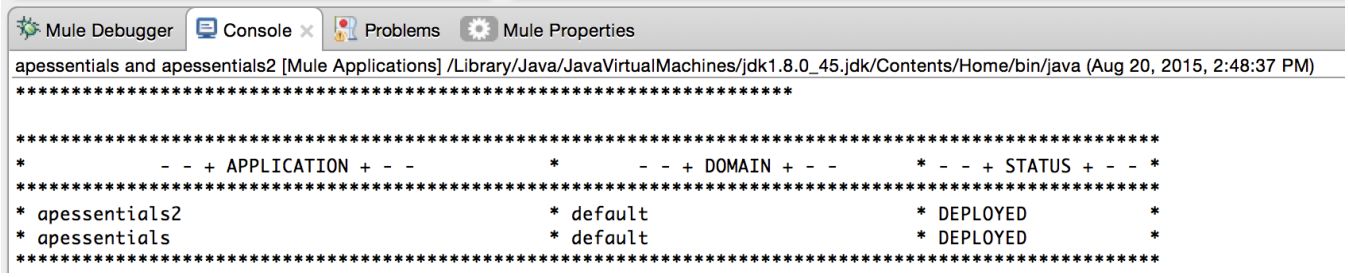
```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+ Started app 'apessentials2'                                          +
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

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.

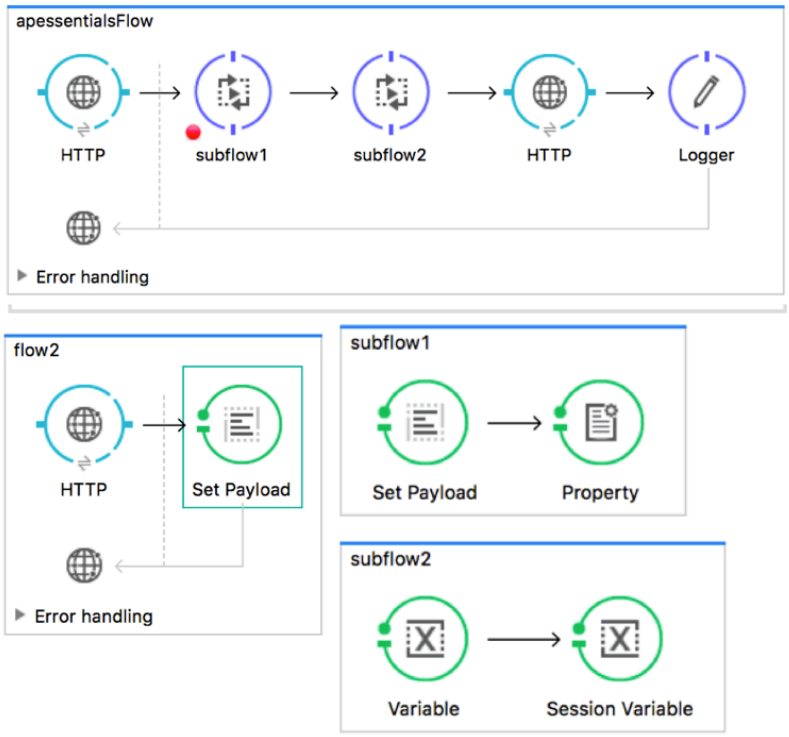


49. Stop the Mule runtime.

Walkthrough 6-4: Create and reference flows and subflows

In this walkthrough, you will work with the apessentialsFlow in examples.xml. You will:

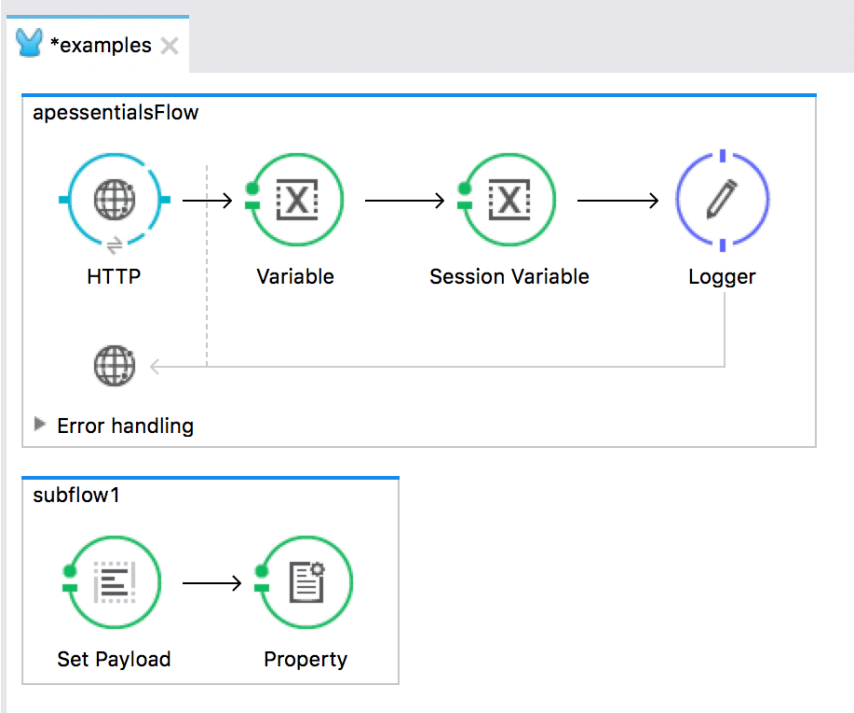
- Extract processors into separate subflows and flows.
- Use the Flow Reference component to reference other flows.
- Create and reference synchronous and asynchronous flows.
- Explore variable persistence through flows, subflows, and across transport barriers.



Create a subflow

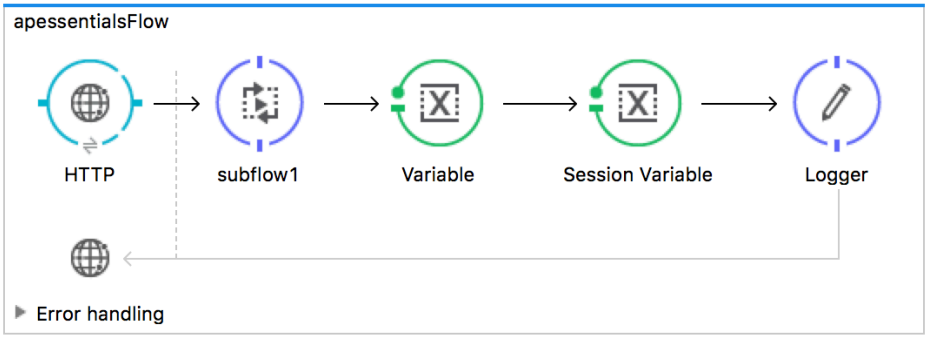
1. Open examples.xml in apessentials2.
2. Drag a Sub Flow scope element to the canvas.
3. Select the Set Payload and Property transformers in the apessentialsFlow and drag them into the subflow.

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.

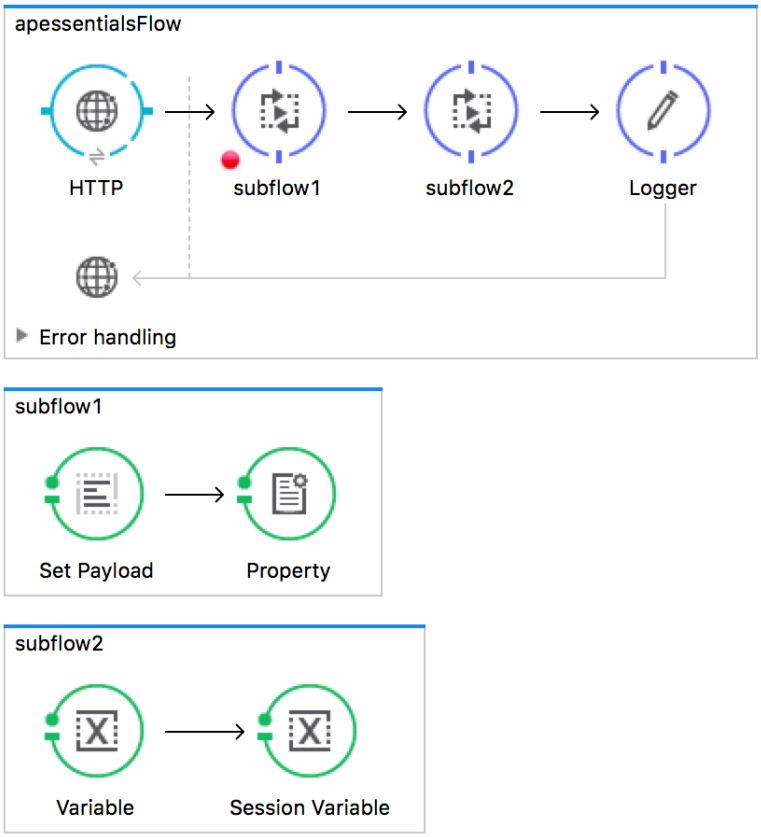


7. Add a breakpoint to the subflow1 Flow Reference.

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.

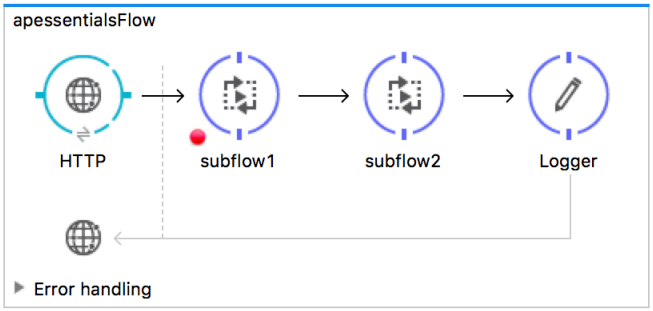


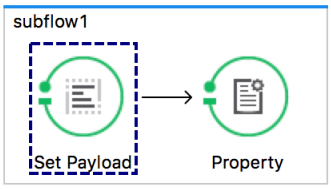
13. Save the file.

Debug the application

14. Click the Debug button and select apessentials2 to just debug apessentials2.
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.

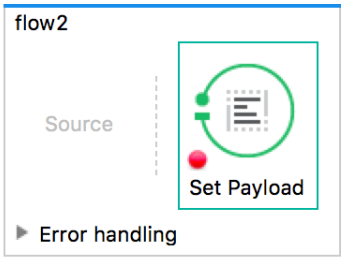




17. Make another request to http://localhost:8082/?name=max&type=mule using your own query parameter values.
18. Step through the application again, this time watching the values of the inbound properties, variables, outbound properties, and session variables.

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.

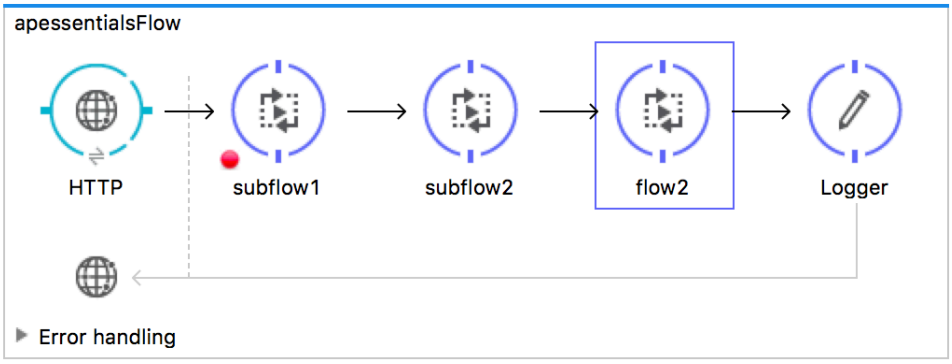


23. In the Set Payload Properties view, set the value to a string, like flow2.

Reference a flow

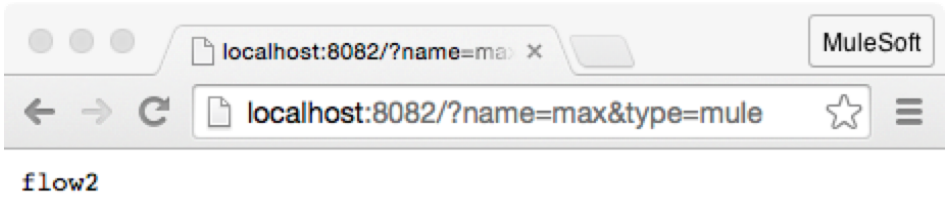
24. Drag a Flow Reference component from the palette and drop it into the apessentialsFlow between the subflow2 Flow Reference and the Logger.

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.

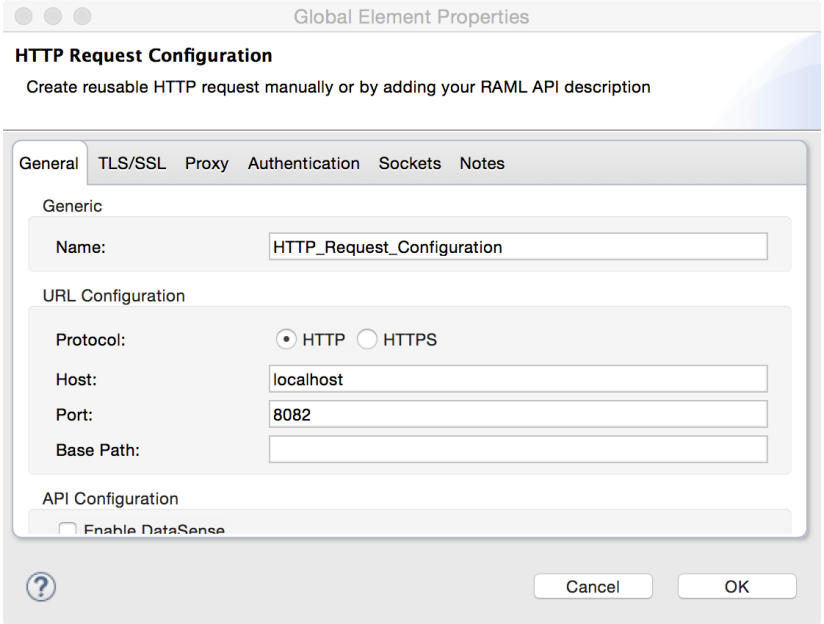


29. Make another request to http://localhost:8082/?name=max&type=mule using your own query parameter values.
30. Step through the application again, this time watching the values of the inbound properties, variables, outbound properties, and session variables.

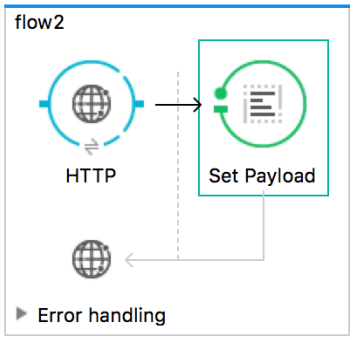
Create a transport barrier for the second flow

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.

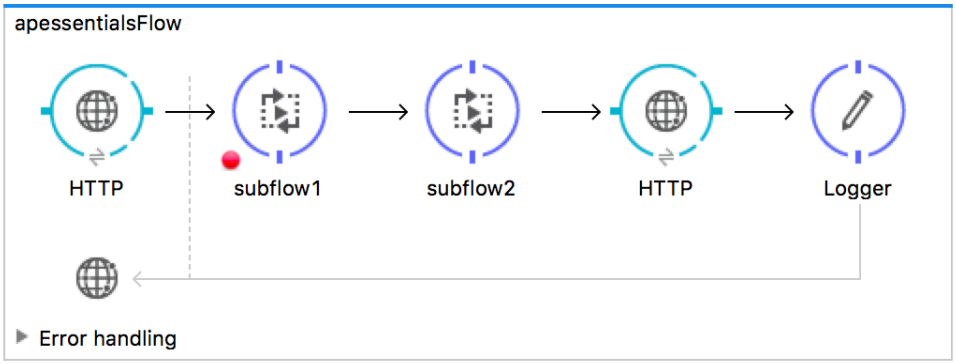


35. In examples.xml, drag an HTTP connector into the Source section of flow2.
36. In the HTTP Properties view, set the connector configuration to the existing HTTP_Listener_Configuration.
37. Set the path to /flow2 and the allowed methods to GET.



38. Add a second HTTP connector after the flow2 reference in apessentialsFlow.
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.



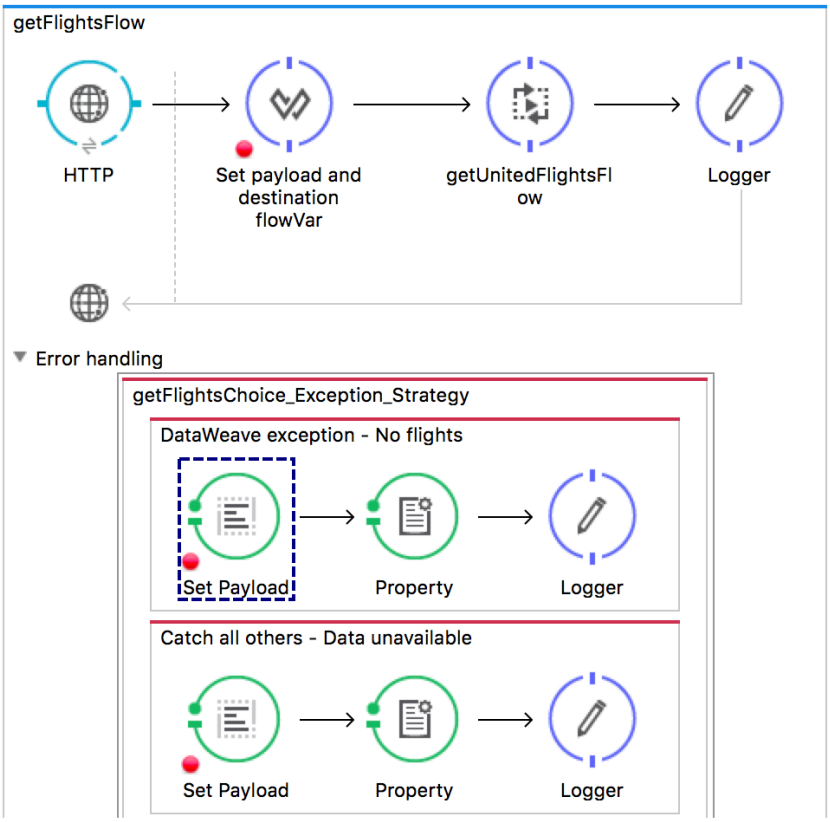
Debug the application

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



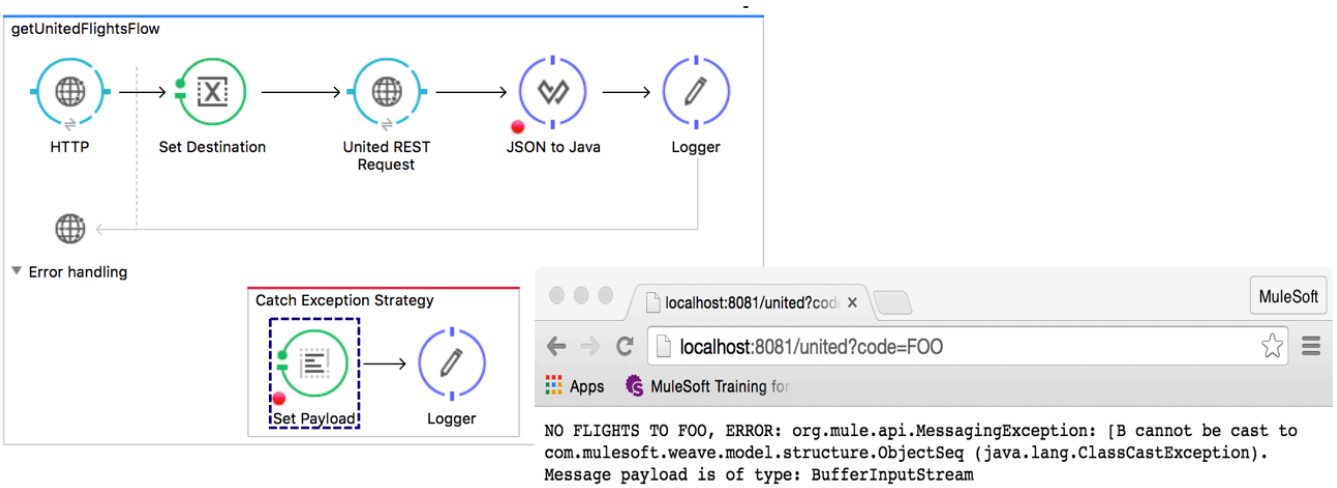
In this module, you will learn:

- 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:

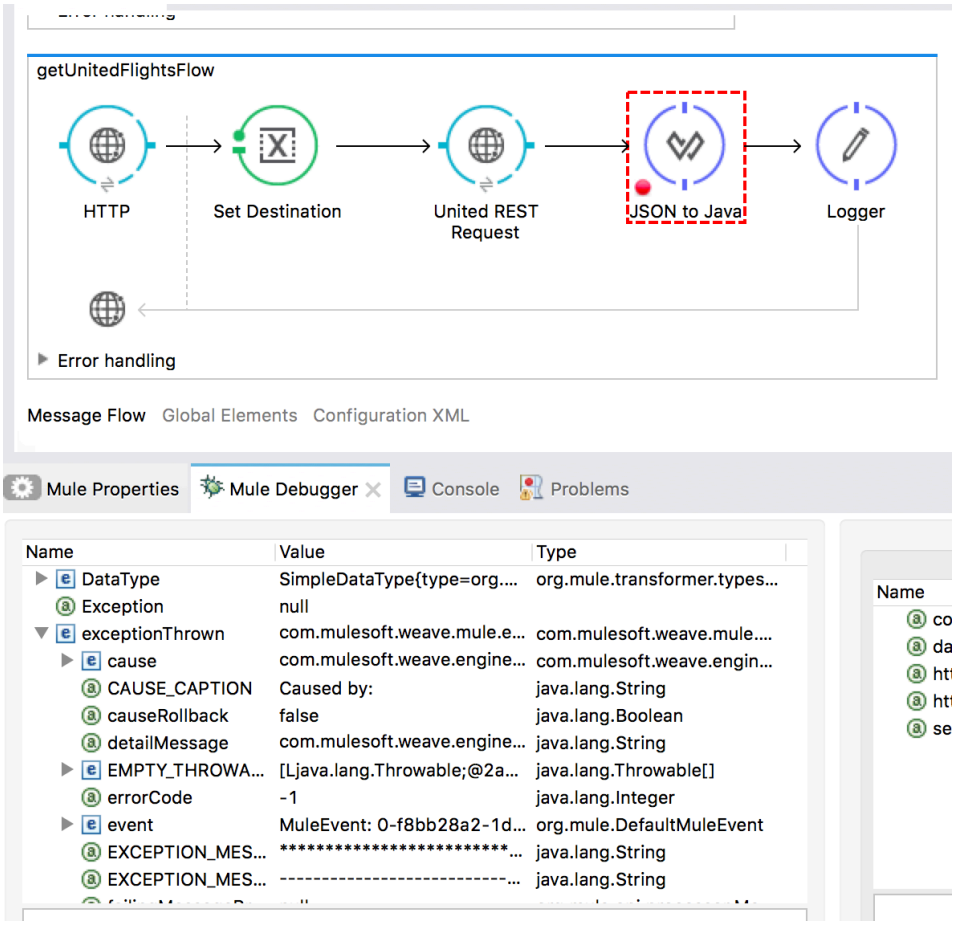
- Add a Catch Exception Strategy to a flow.
- Catch the exception and send an error message back to the requester.
- Reference the exception object inside an exception handler.
- Create and catch a web service request error.



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.

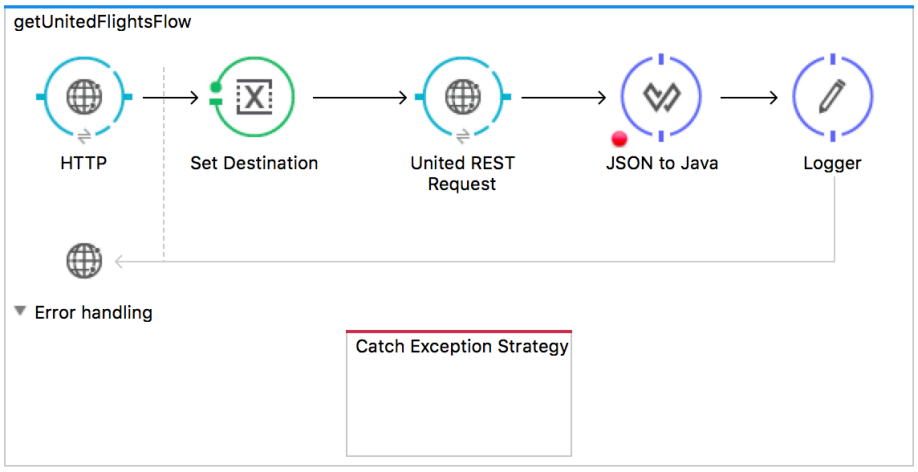


6. Click the Resume button.

Add a catch exception strategy

7. Return to getUnitedFlightsFlow and click the arrow to expand the Error handling section.

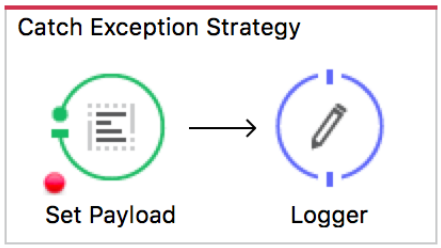
8. Drag and drop a Catch Exception Strategy from the Error Handling section of the palette into the Error handling section of the flow.



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]
```

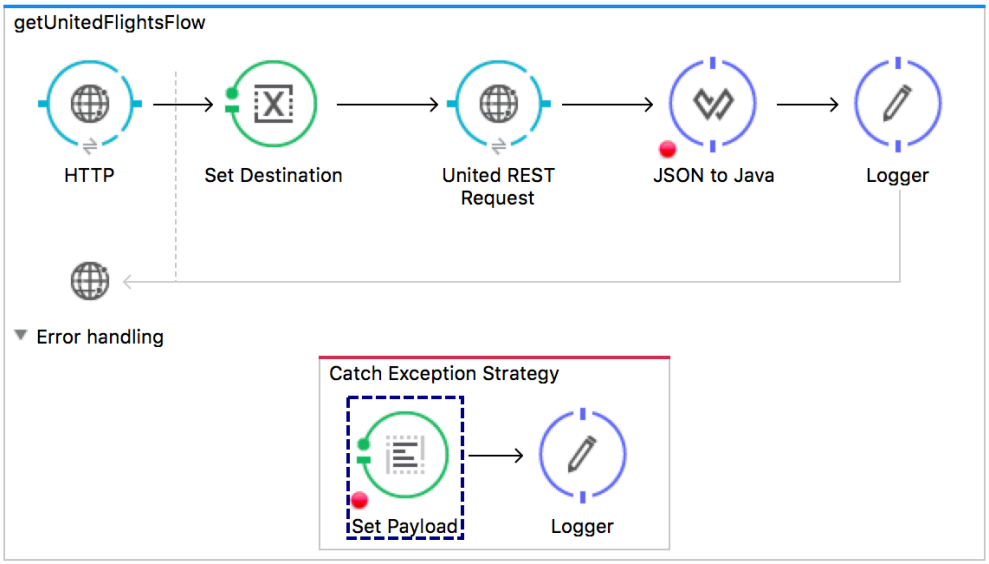
11. Add a breakpoint to the transformer.
12. Add a Logger after the transformer.



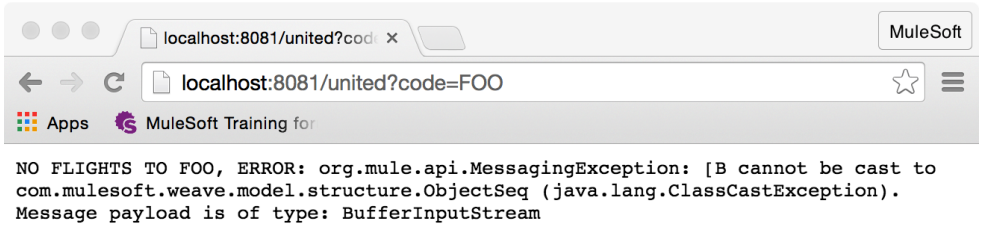
Test the application

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 /{destinatio}.

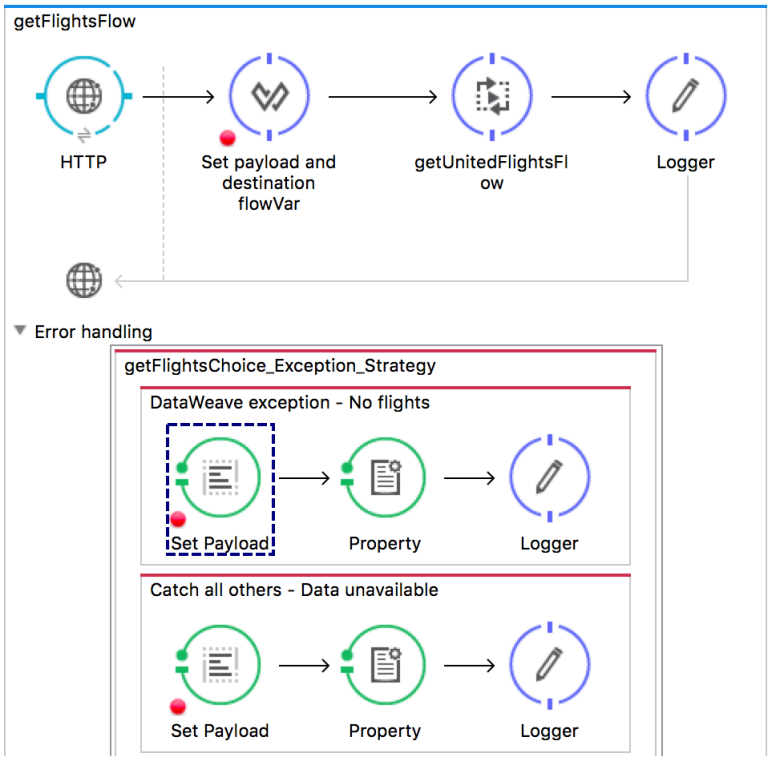
Test the application

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.

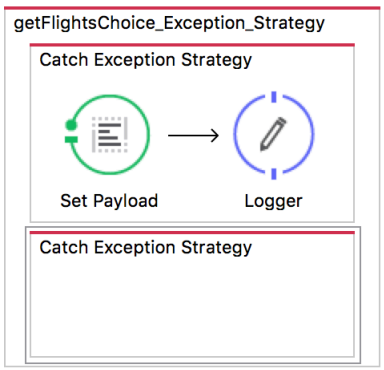


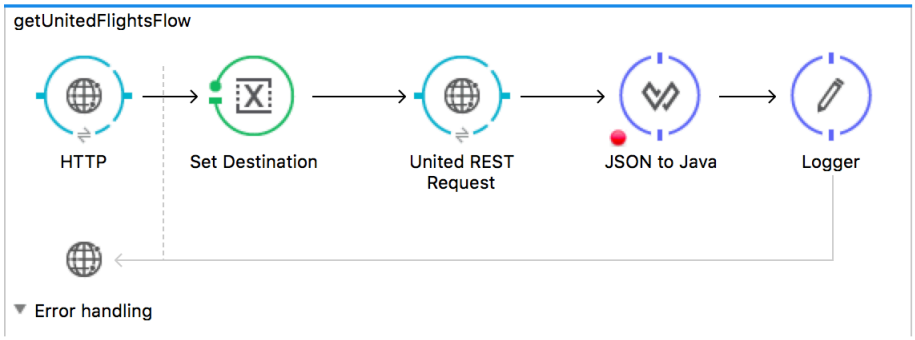
Add a choice exception strategy

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.





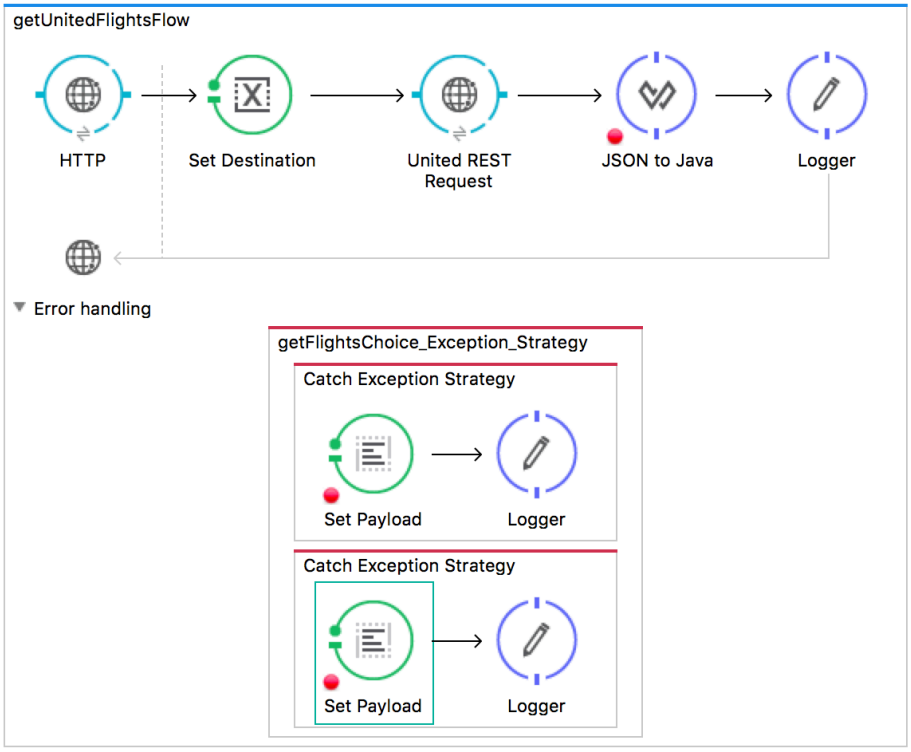
4. Drag the Choice Exception Strategy and drop it in the Error handling section of getUnitedFlightsFlow.

   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:

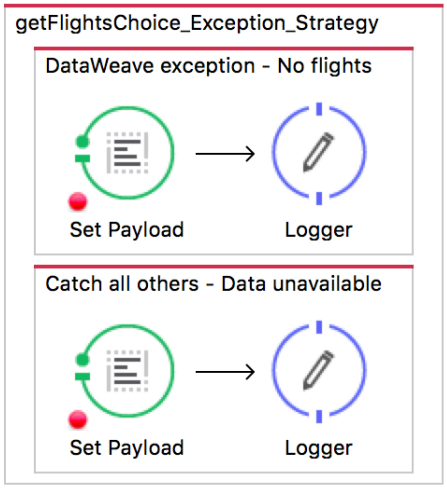
```
DATA IS UNAVAILABLE. TRY LATER. ERROR: #[exception]
```

7. Add a breakpoint to the transformer.



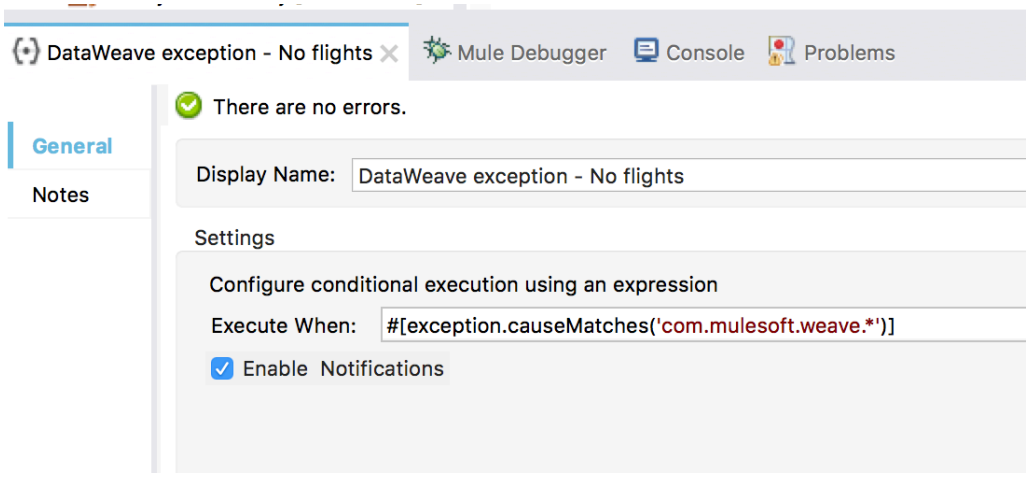
Configure the choice exception strategy

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.*')]
```

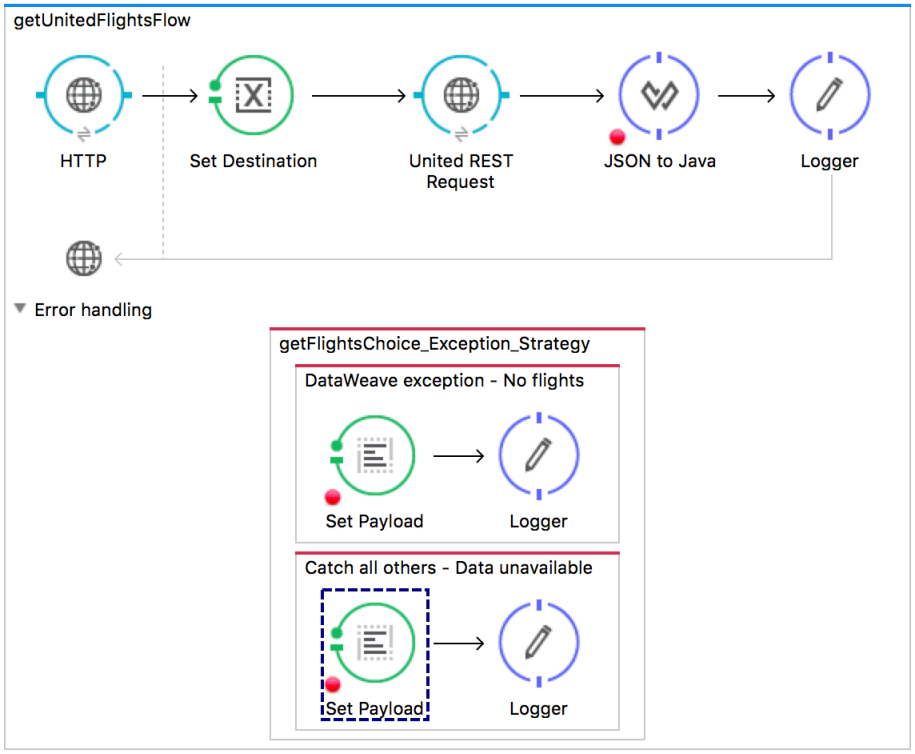


11. In the Properties view for the second Catch Exception Strategy, do not set any execute when expression.

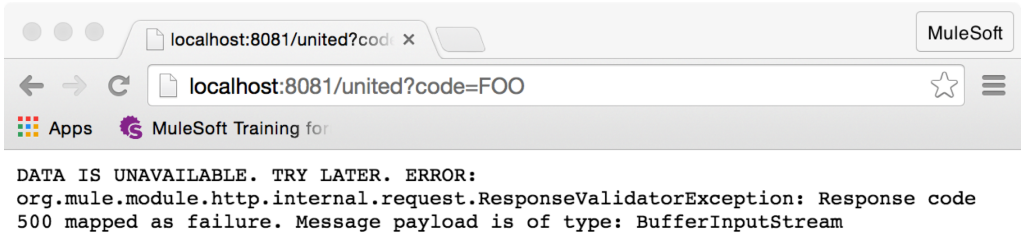
Test the application

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.



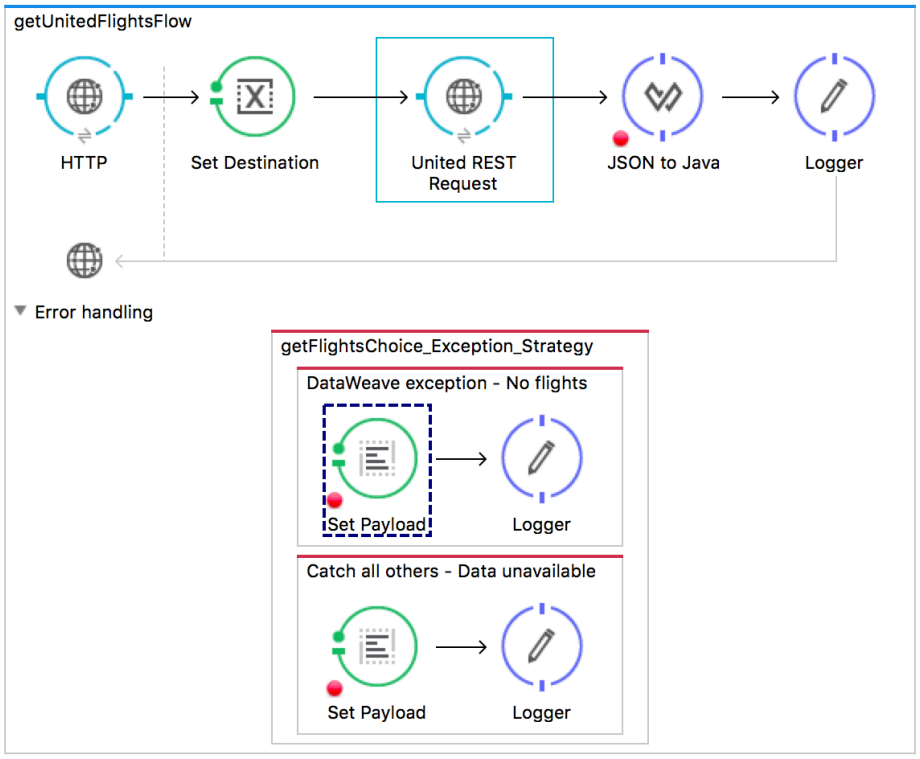
Fix the RESTful web service request error

16. Navigate to the United REST Request endpoint in getUnitedFlightsFlow.
17. Change the path back to /{destination}.

Test the application

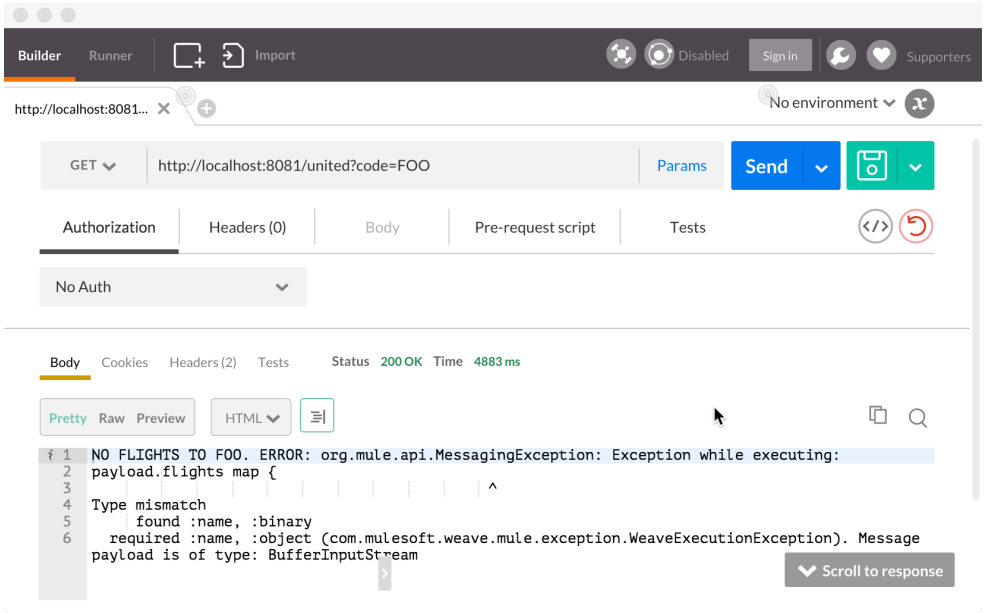
18. Save the file and debug the application.
19. Make a request to http://localhost:8081/united?code=FOO.

20. Step through the application; the exception should be handled by the first Catch Exception Strategy.



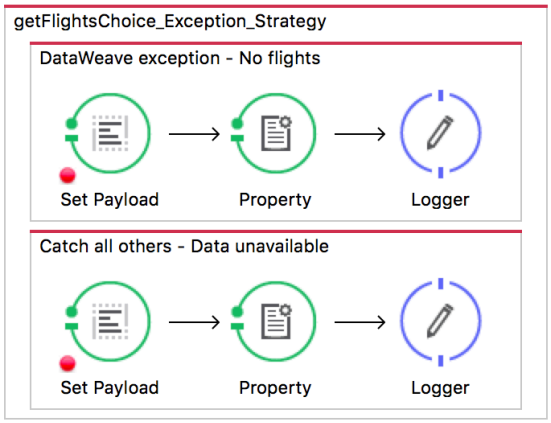
21. Step to the end of the application; you should see the first error message.

22. If you are using Postman, locate the http status code value.

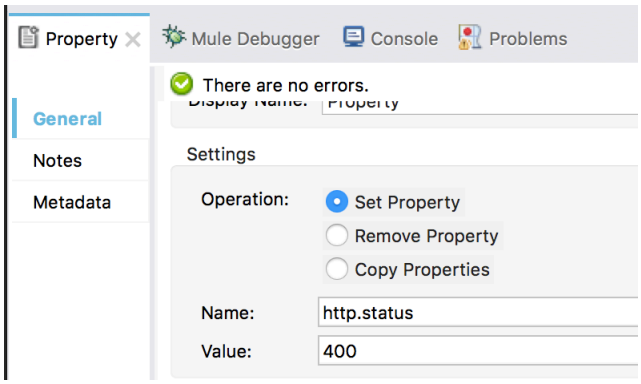


Set http status codes

23. Add a Property transformer before the Logger in both Catch Exception Strategies.



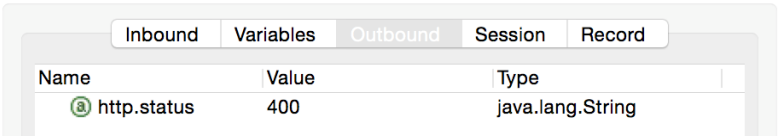
24. In the first Property transformer Properties view, set the property http.status to 400.



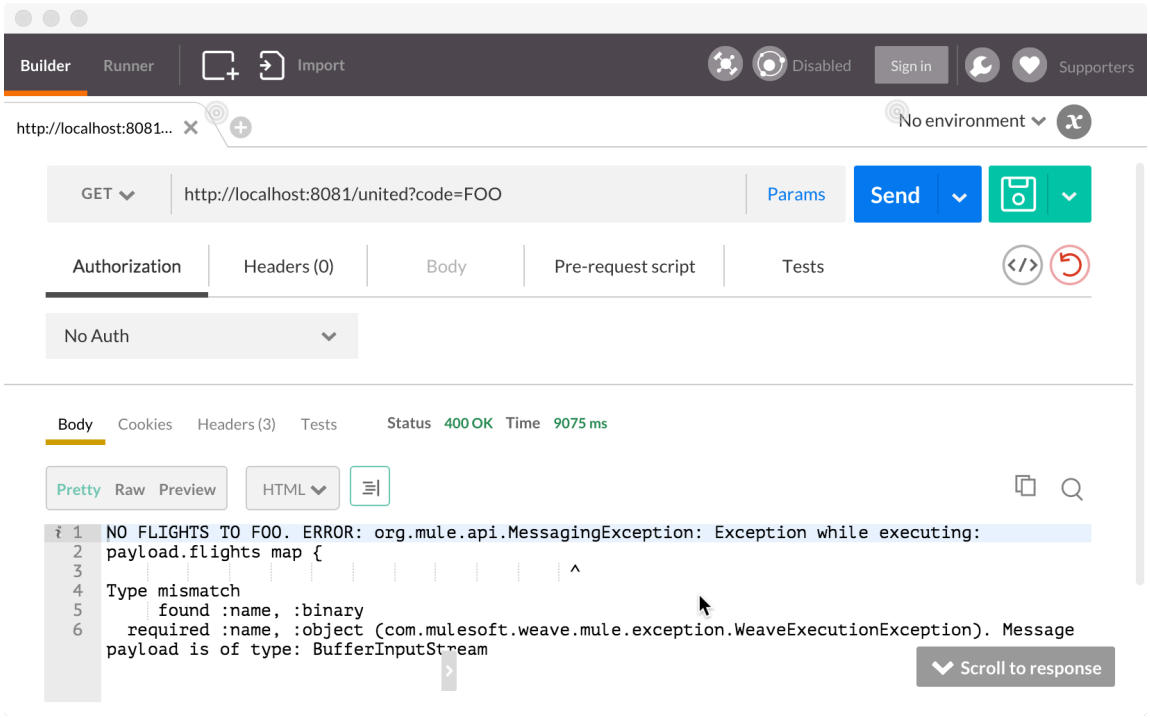
25. In the second Property transformer Properties view, set the property http.status to 500.

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.



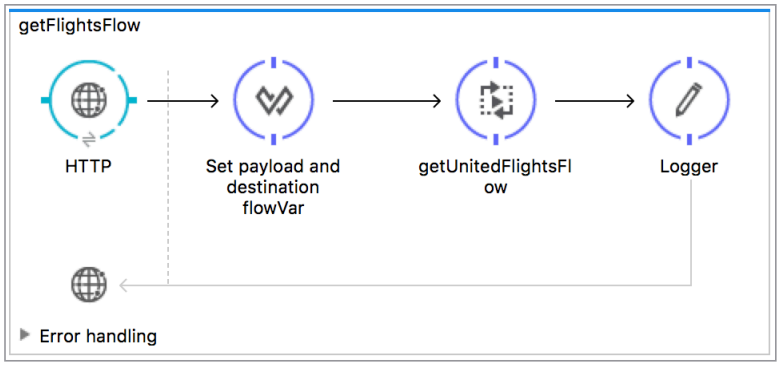
29. Step to the end of the application; if you are using Postman, you should see the new http status code value.



Call the United flow from another flow

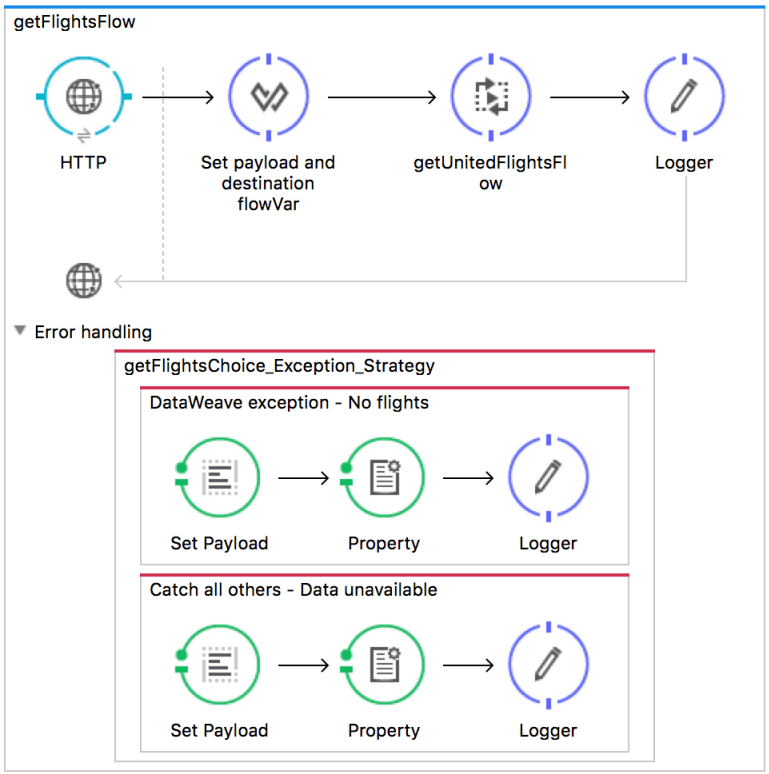
30. Locate getFlightsFlow.
31. Add a Flow Reference before the Logger.

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.

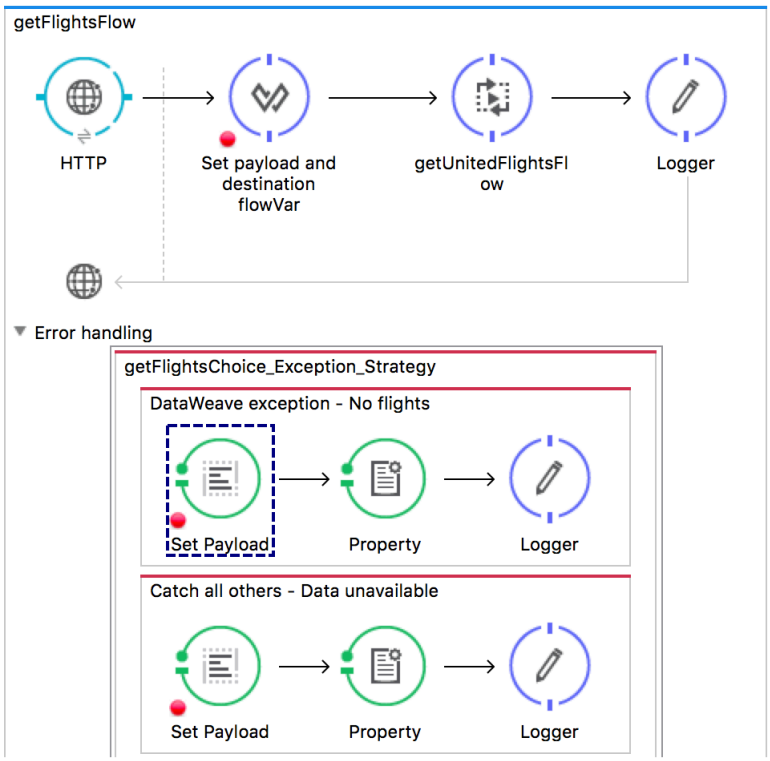


Test the application

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.

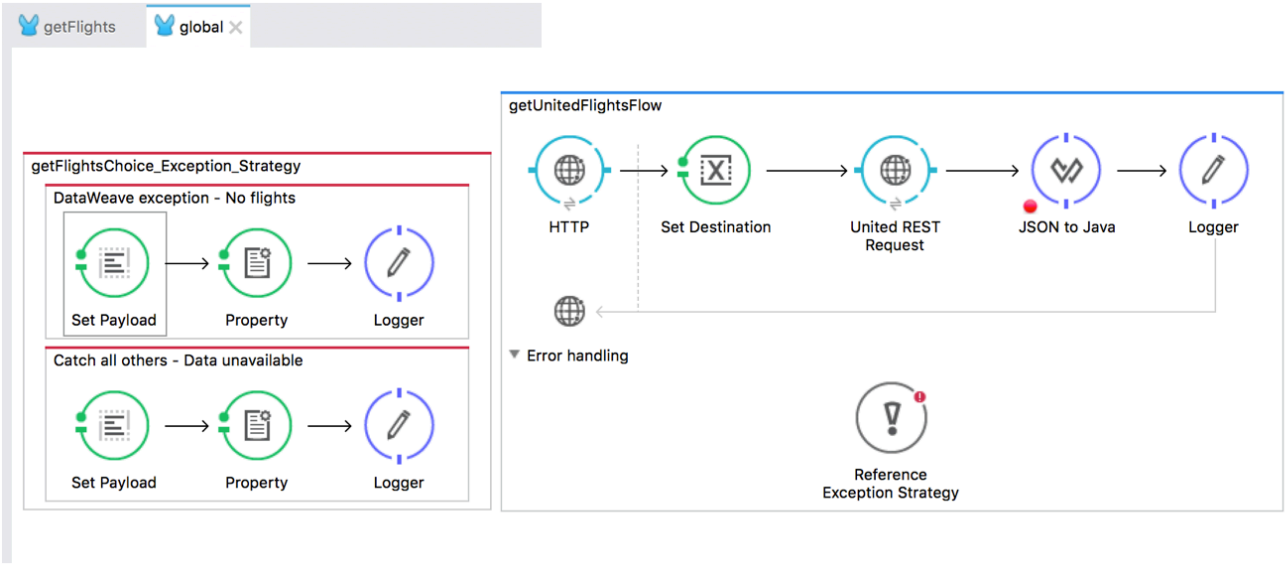
    Note: You will have to click the Resume button to move into the exception strategy after the exception is thrown by the United flow.



Walkthrough 7-3: Create and use global exception handlers

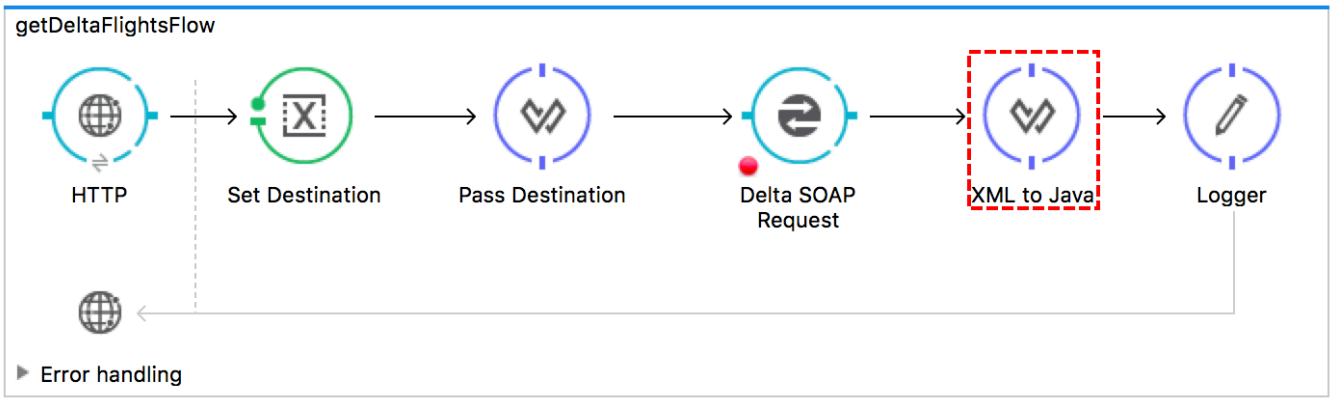
In this walkthrough, you will create a global exception handler in the MUA application. You will:

- 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.



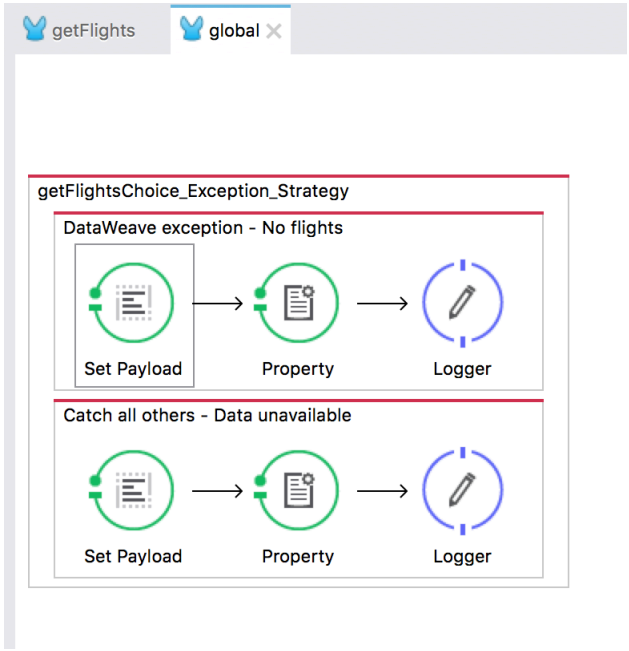
6. Click the Resume button.

Create a global exception handler

7. In getFlights.xml, switch to the Configuration XML view.
8. Locate the getFlightsChoice_Exception_Strategy and select it and cut it.

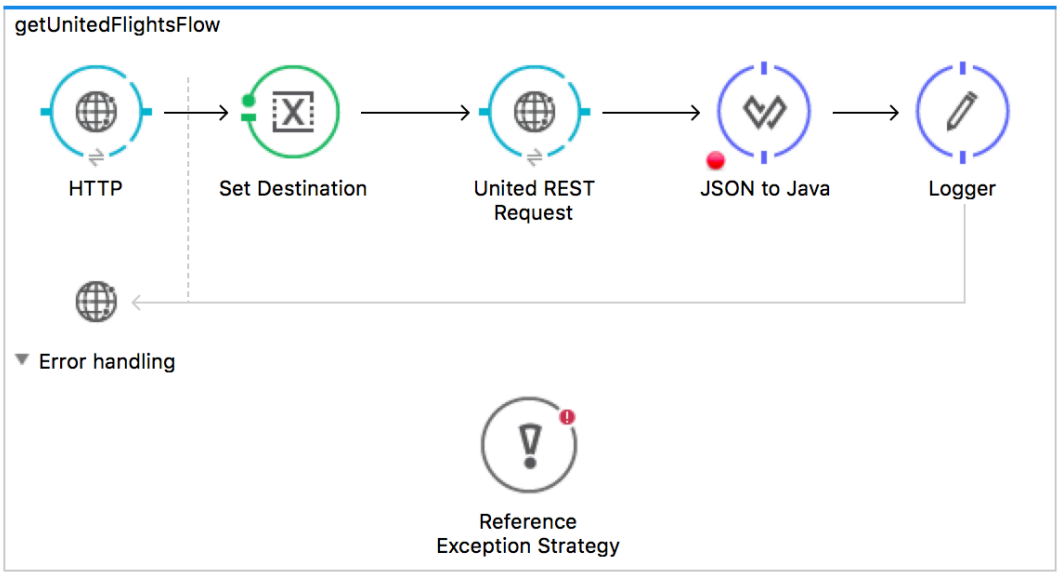
```
        <flow-ref name="getUnitedFlightsFlow" doc:name="getUnitedFlightsFlow"
        <logger level="INFO" doc:name="Logger" />
        <choice-exception-strategy doc:name="getFlightsChoice_Exception_Strate
            <catch-exception-strategy
                when="#[exception.causeMatches('com.mulesoft.weave.*')]" doc:n
                <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 unavai
                <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. Paste the Choice Exception Strategy after the global configuration elements inside the start and end mule tags.
11. Switch to the Message Flow view; you should see the Choice Exception Strategy.

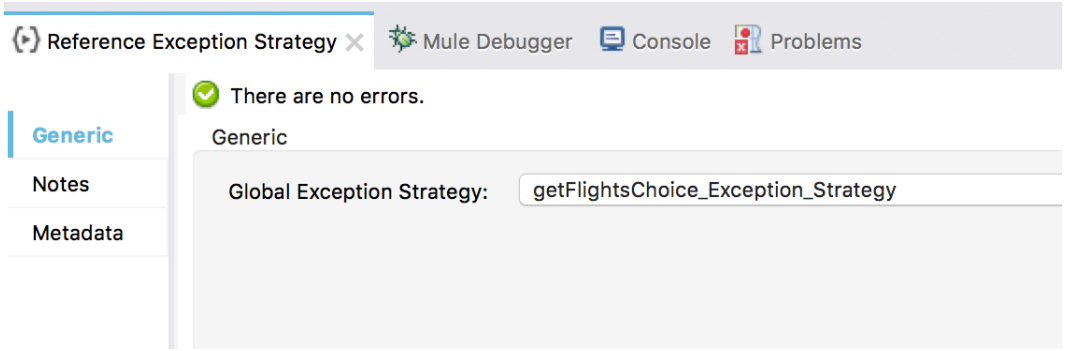


Use the global exception handler

12. Return to getFlights.xml and switch to the Message Flow view.
13. Locate the getUnitedFlightsFlow and expand its Error handling section.
14. Drag a Reference Exception Strategy from the palette and drop it in the Error handling section.



15. In the Properties view, set the global exception strategy to getFlightsChoice_Exception_Strategy.

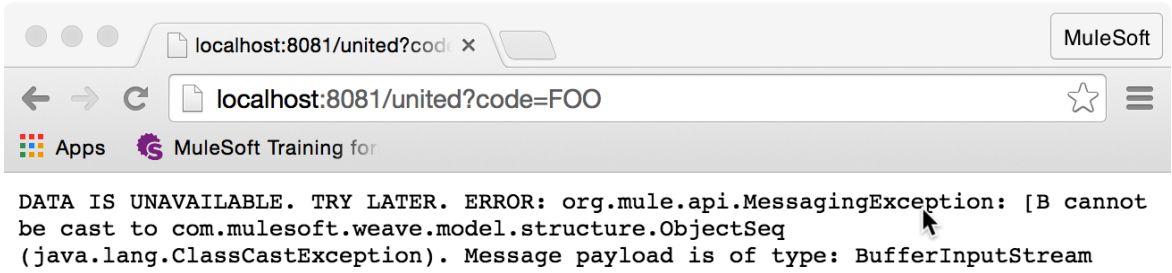


16. Drag a Reference Exception Strategy to the error handling section of getDeltaFlightsFlow.
17. In the Properties view, set the global exception strategy to getFlightsChoice_Exception_Strategy.

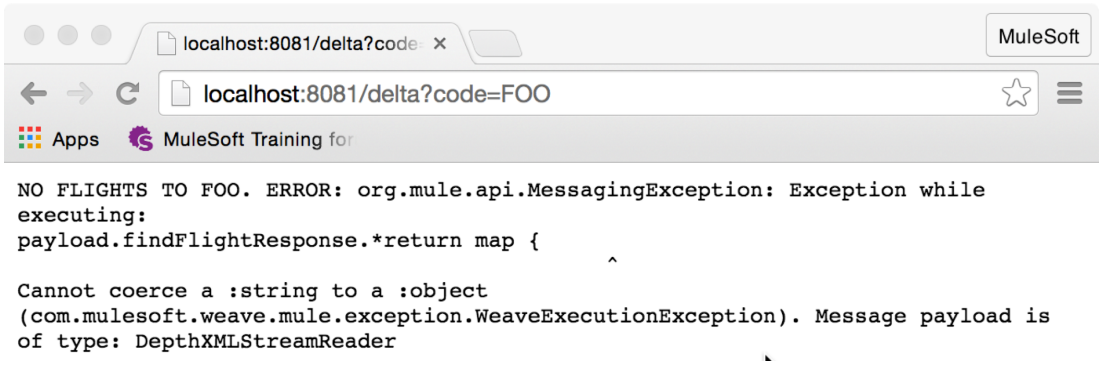
    Note: You could add a Reference Exception Strategy to the rest of the flows, but instead, you will create a global default exception strategy in the next walkthrough.

Test the application

18. Save all the files and run the application.
19. Make a request to http://localhost:8081/united?code=FOO; you should see the data unavailable error message displayed.



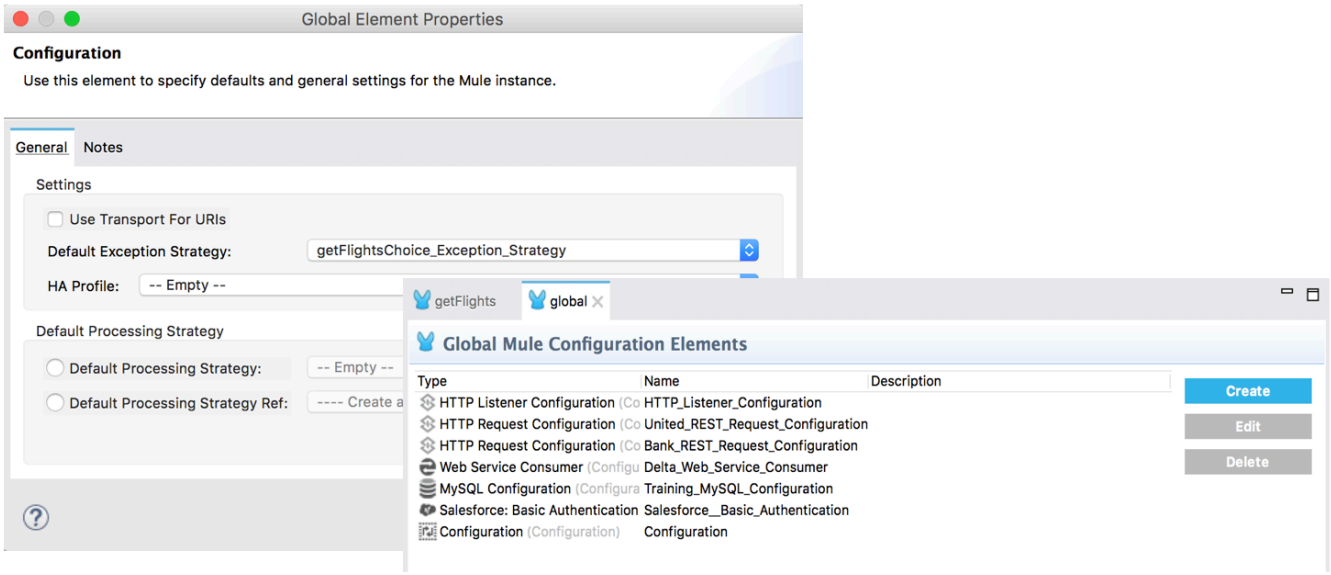
20. Make a request to http://localhost:8081/delta?code=FOO; you should see the data unavailable error message displayed.



Walkthrough 7-4: Specify a global default exception strategy

In this walkthrough, you will change the default exception handling for the application. You will:

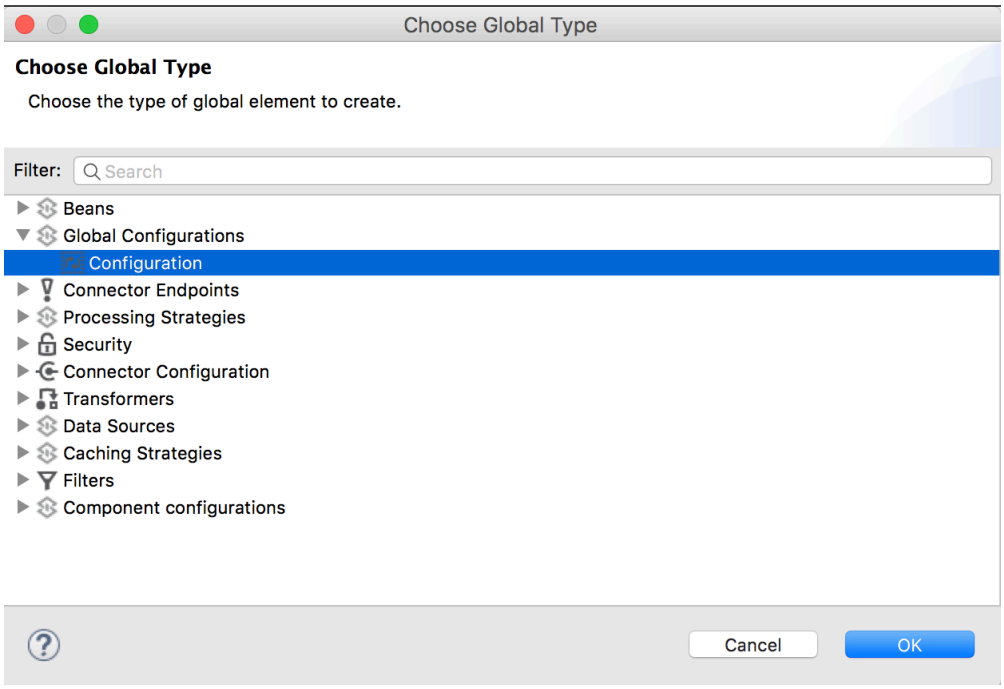
- Create a global configuration element in the global.xml file.
- Specify a default exception strategy in the global configuration element.
- Remove the existing exception handling strategies.
- Use the default exception handling strategy.



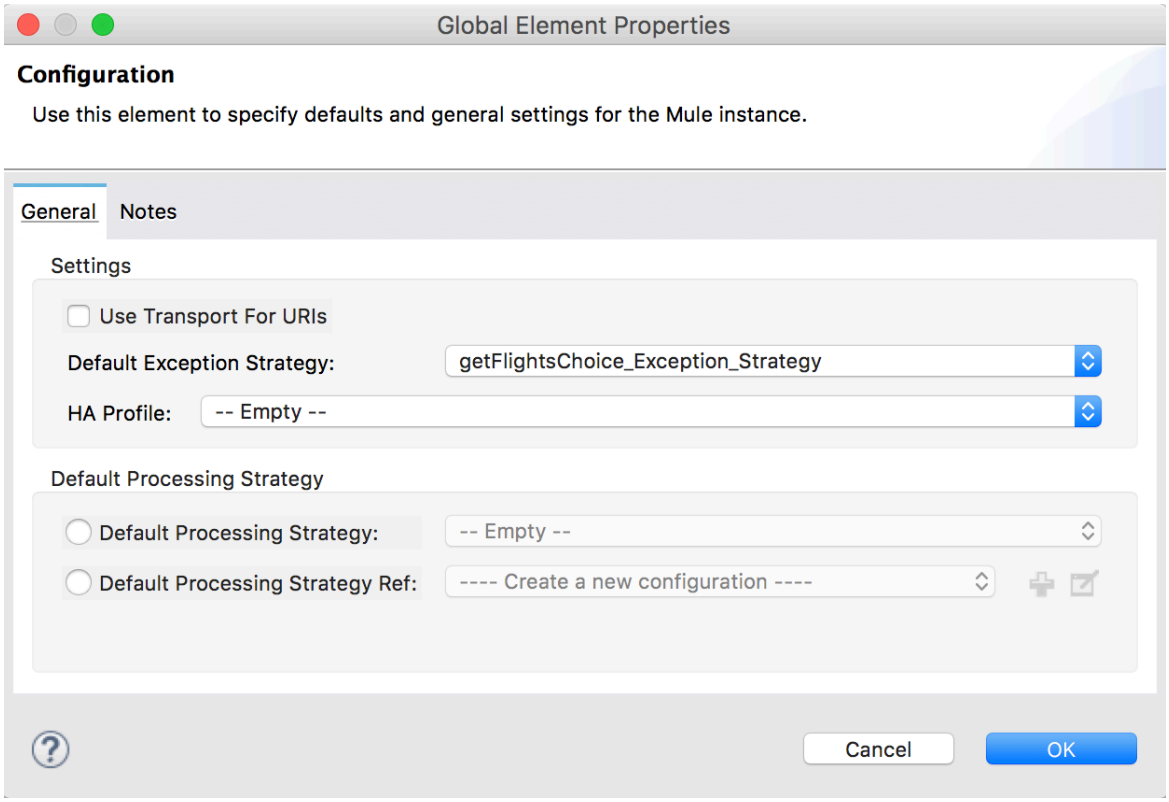
Specify a global default exception strategy

1. Return to global.xml.
2. Switch to the Global Elements view.
3. Click the Create button.

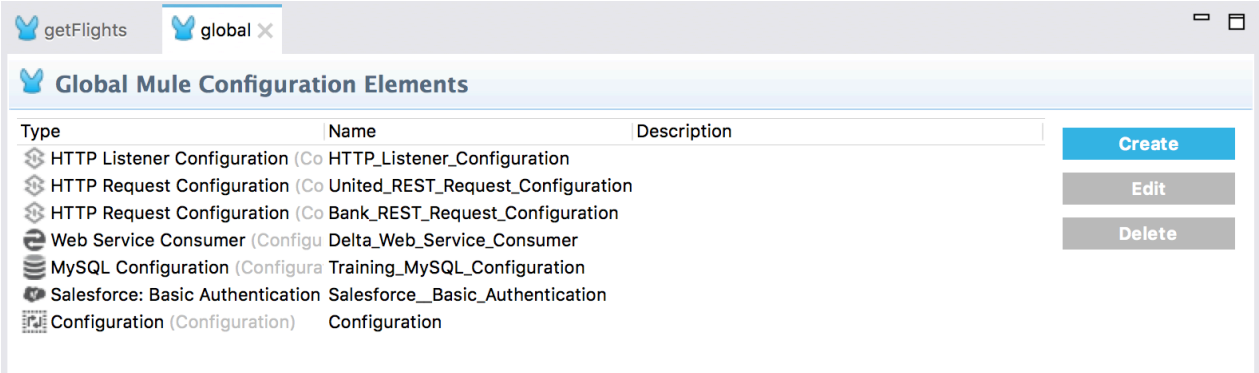
4. In the Choose Global Type dialog box, select Global Configurations > Configuration and click OK.



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.

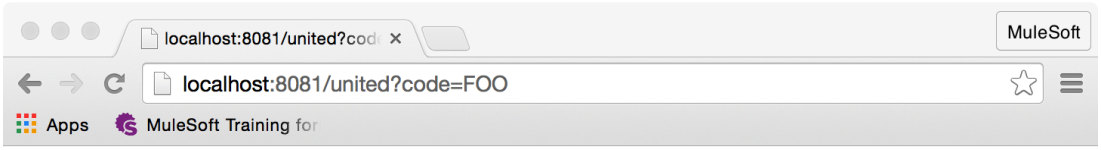


Remove the existing exception strategy references

7. Return to getFlights.xml.
8. Delete the Reference Exception Strategy in getUnitedFlightsFlow.
9. Delete the Reference Exception Strategy in getDeltaFlightsFlow.

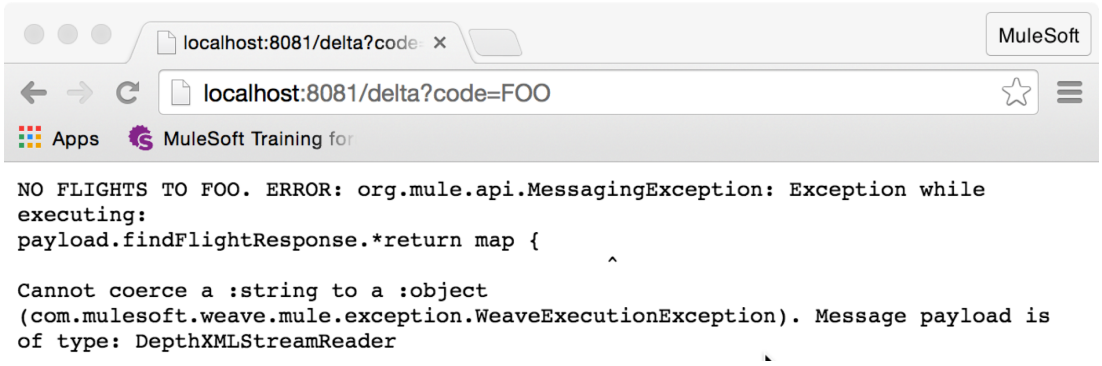
Test the application

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.



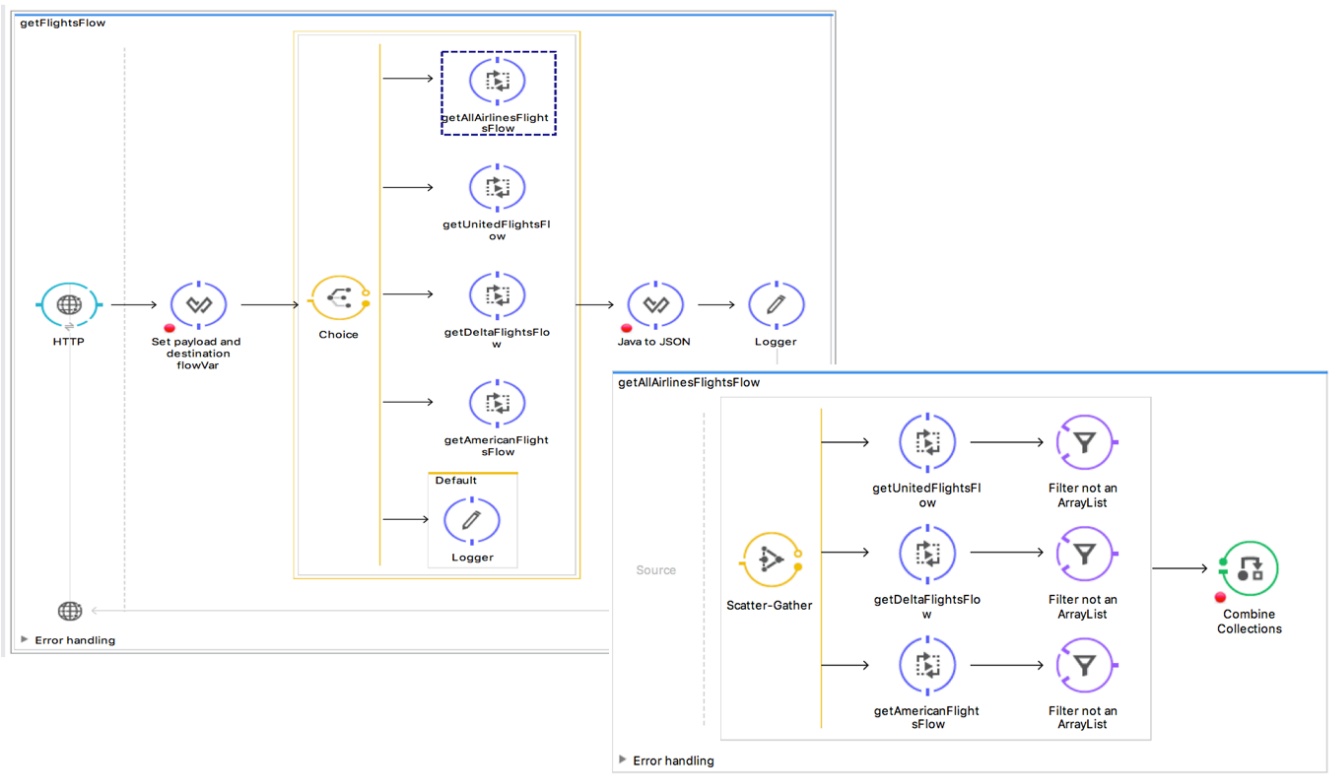

```
NO FLIGHTS TO FOO. ERROR: org.mule.api.MessagingException: Exception while executing:
payload.flights map {
                                                ^

Type mismatch
     found :name, :binary
  required :name, :object (com.mulesoft.weave.mule.exception.WeaveExecutionException).
Message payload is of type: BufferInputStream
```

12. Make a request to http://localhost:8081/delta?code=FOO; you should still see the no flights error message displayed.



13. Stop the Mule runtime.

Module 8: Controlling Message Flow



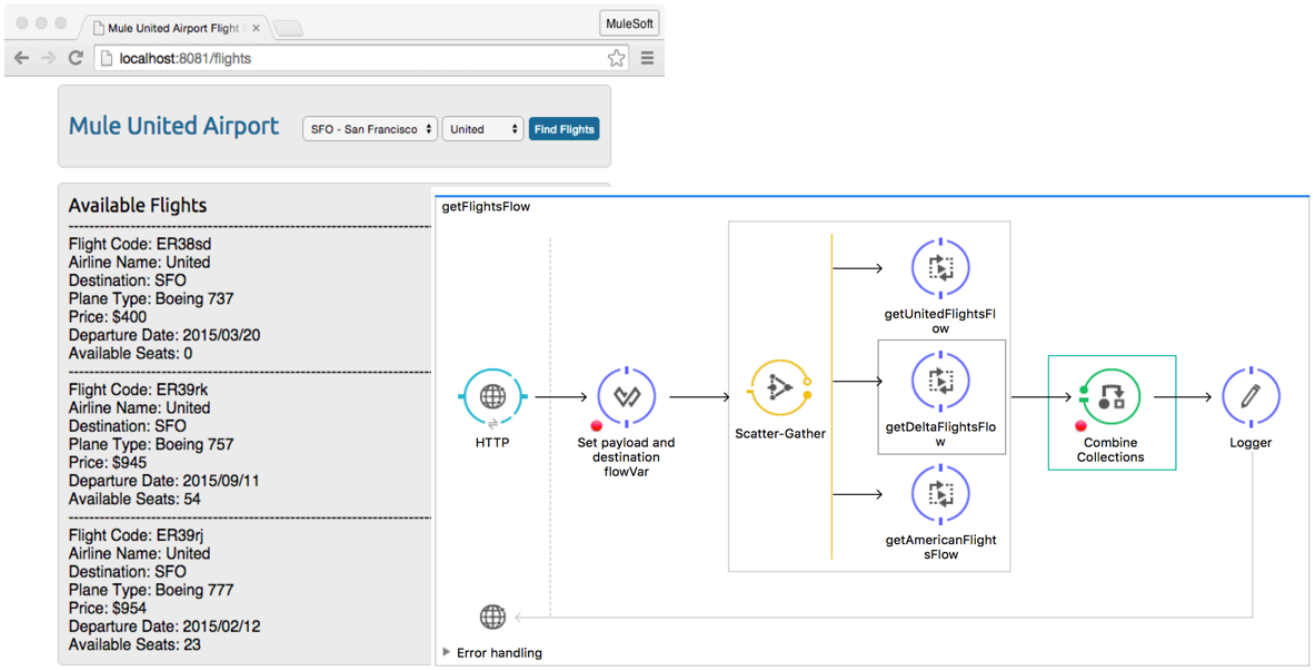
In this module, you will learn:

- 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:

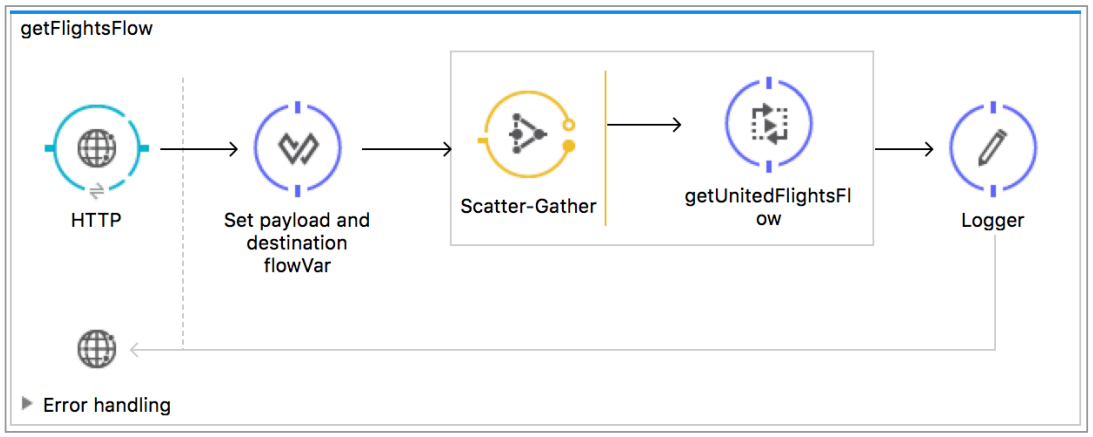
- Use a Scatter-Gather router to concurrently call all three flight services.
- Use a Combine Collections transformer to combine a collection of three ArrayLists of objects into one collection.
- Use DataWeave to sort the flights and return them as JSON to the form.



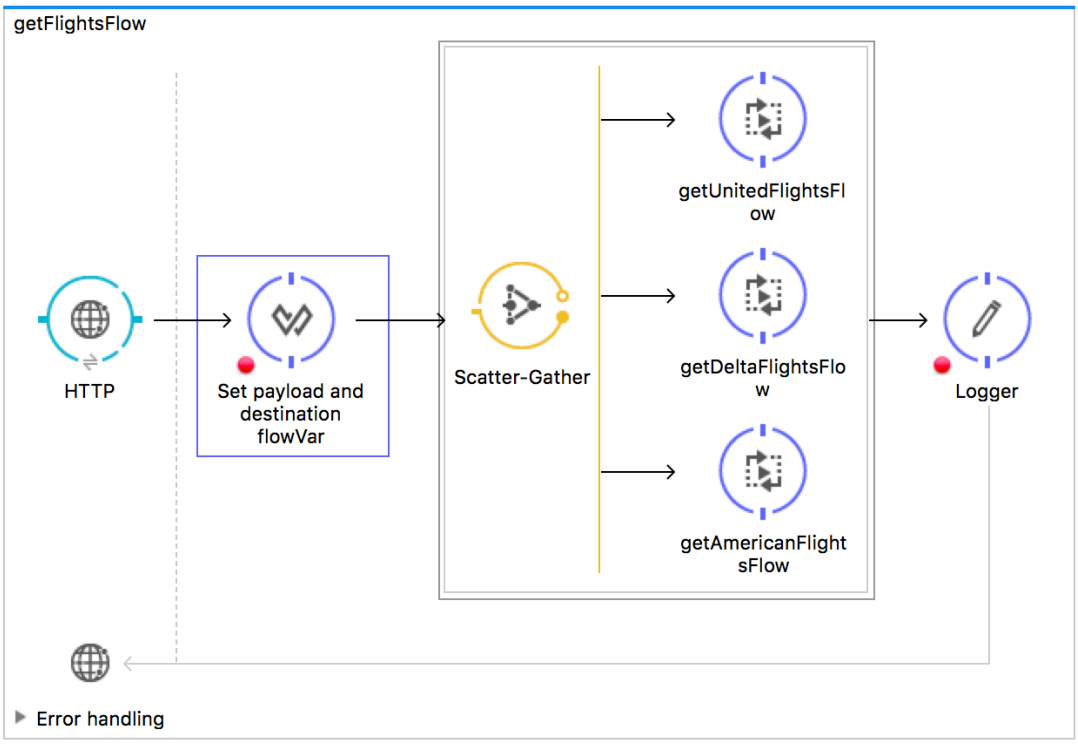
Add a router to call all three airline services

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.

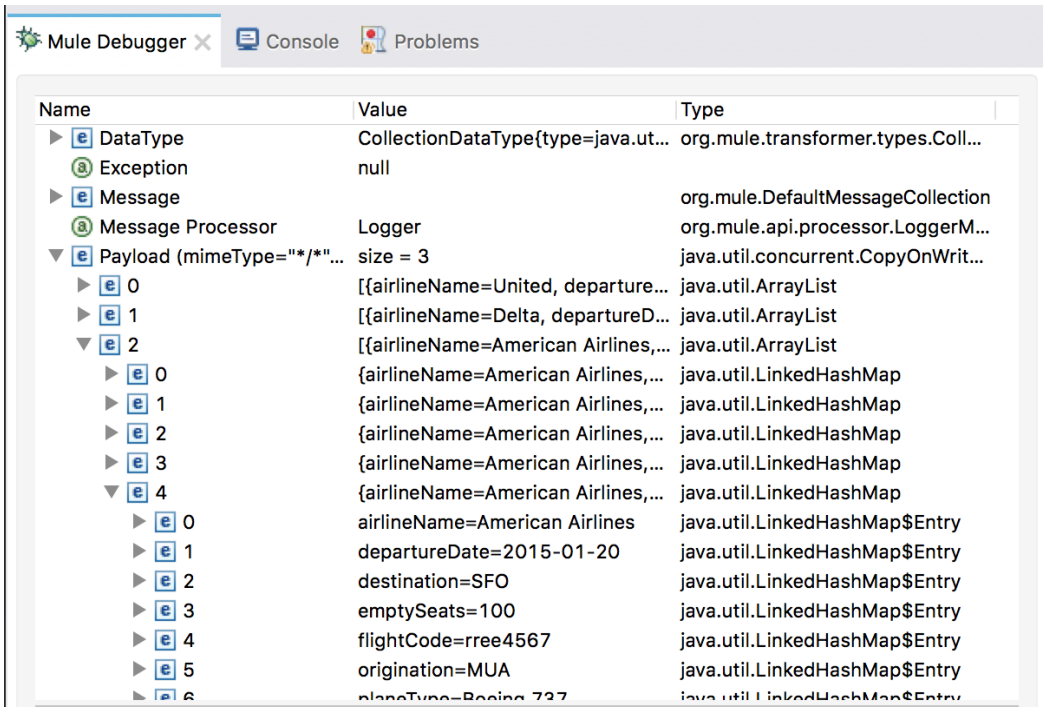


4. Add a second Flow Reference component to the Scatter-Gather router.
5. In the Properties view for the Flow Reference, set the flow name to getDeltaFlightsFlow.
6. Add a third Flow Reference component to the Scatter-Gather router.
7. In the Properties view for the Flow Reference, set the flow name 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.

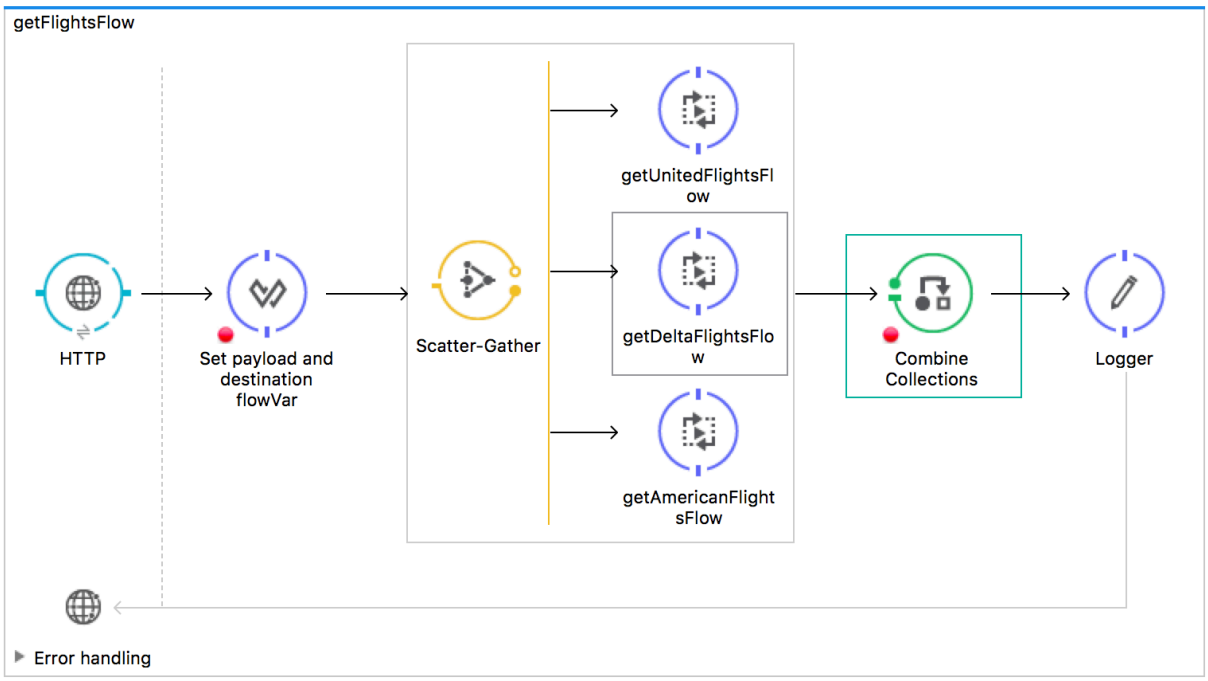


Note: If you step through to the end of the application, ignore any errors you get.

15. Click the Resume button.

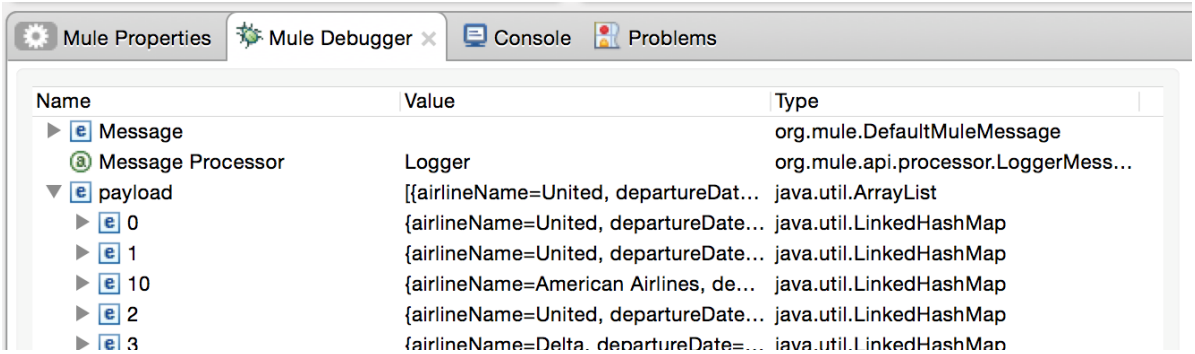
Flatten the results

16. Add a Combine Collections transformer before the Logger.

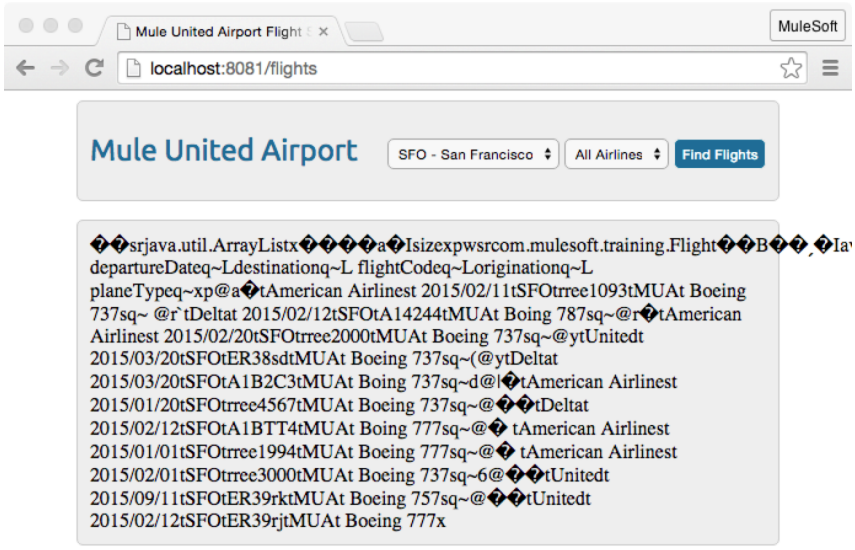


Test the application

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.



20. Step through to the end of the application; you should see a jumble of data returned.



Review HTML form code to see what data format it expects

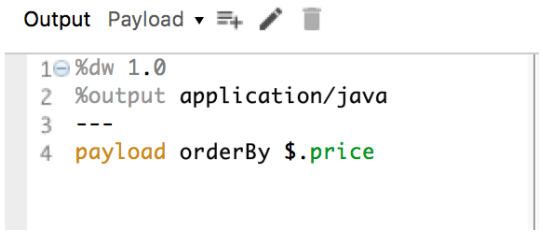
21. Open FlightFinder.html in src/main/resources and locate in what format it expects the data to be sent back.

```
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 + "<br>";
```

Return the data as JSON and sort by price

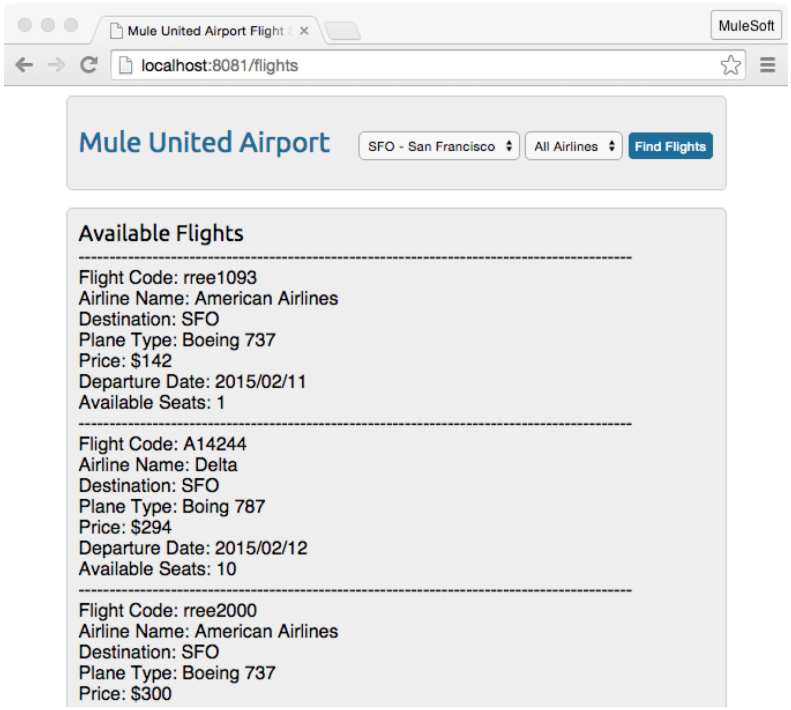
22. In getFlights.xml, add a Transform Message component after the Combine Collections transformer.
23. Change its name to Java to JSON
24. In the Java to JSON Properties view, change the output type to application/json.

25. Write a DataWeave expression to transform the payload and order by price.



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.



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.

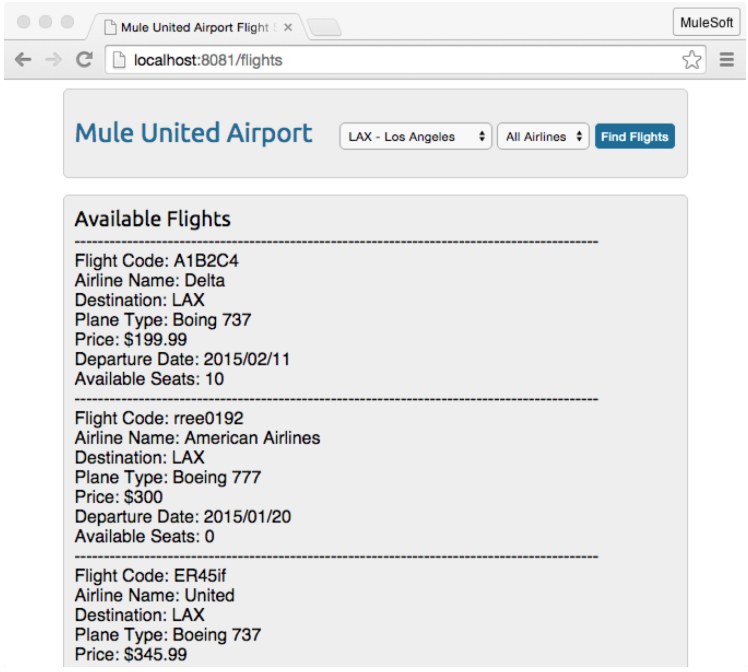
```
#[(flowVars.destination ==empty &&
message.inboundProperties.'http.query.params'.code == empty) ? 'SFO' :
(flowVars.destination != empty ? flowVars.destination :
message.inboundProperties.'http.query.params'.code)]
```

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.

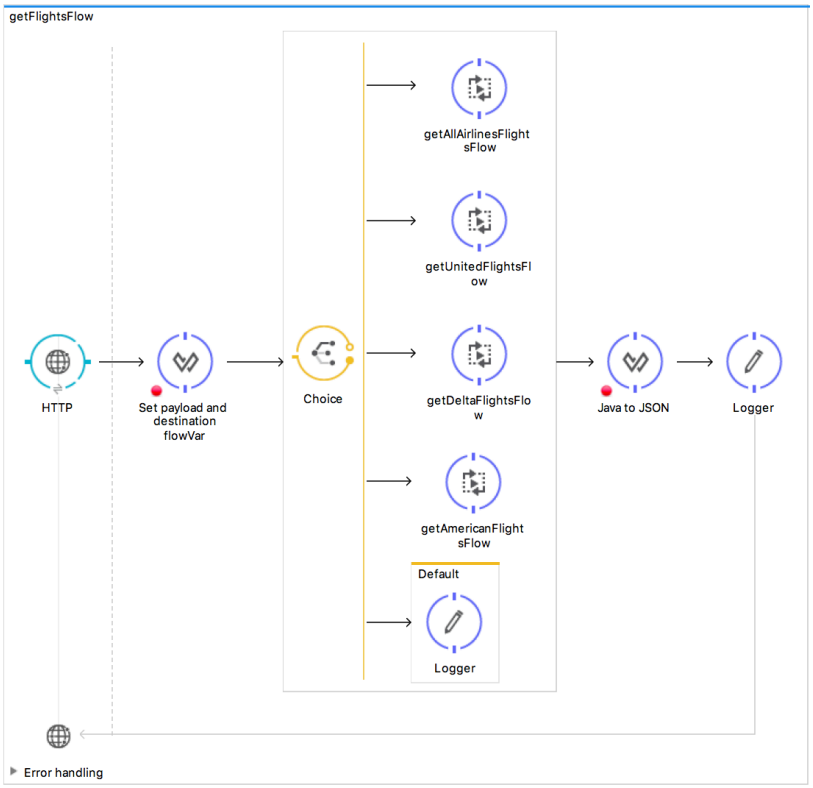


35. Submit the form for a destination of PDF or FOO; you should get an error in the console but no return error message to the form.
36. Submit the form for SFO, LAX, CLE, or PDX and a different airline; you should still see flights for all three airlines returned.

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

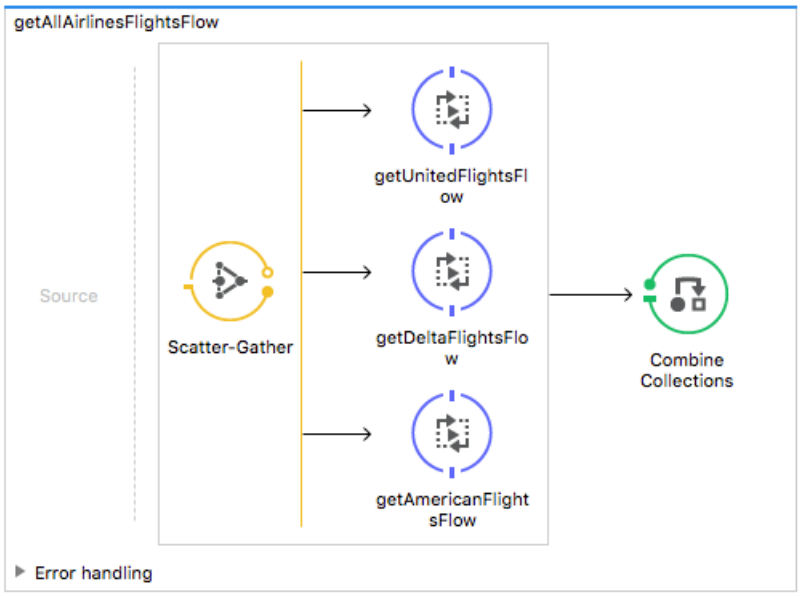
1. In Anypoint Studio, return to FlightFinder.html.
2. Locate the select box that sets the airline and see what values are set and returned.

```
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.
7. Leave the target Mule configuration set to current and click OK.

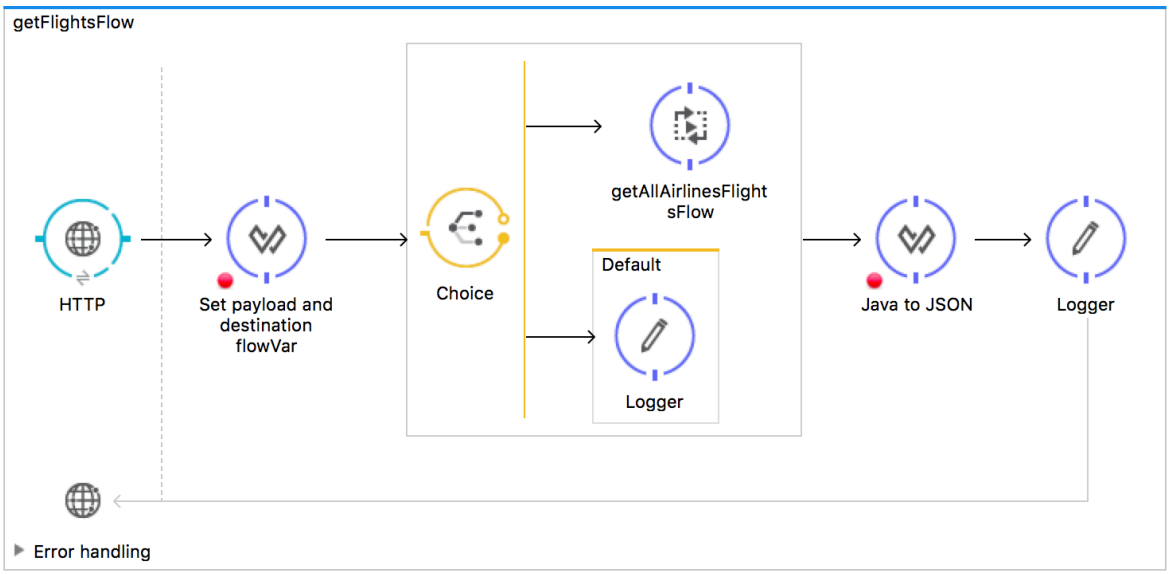


8. In getFlightsFlow, double-click the new Flow Reference.
9. Change its display name to getAllAirlinesFlightsFlow.
10. Move getAllAirlinesFlightsFlow beneath getFlightsFlow.
11. Add a breakpoint to the Combine Collections transformer.

Add a Choice router

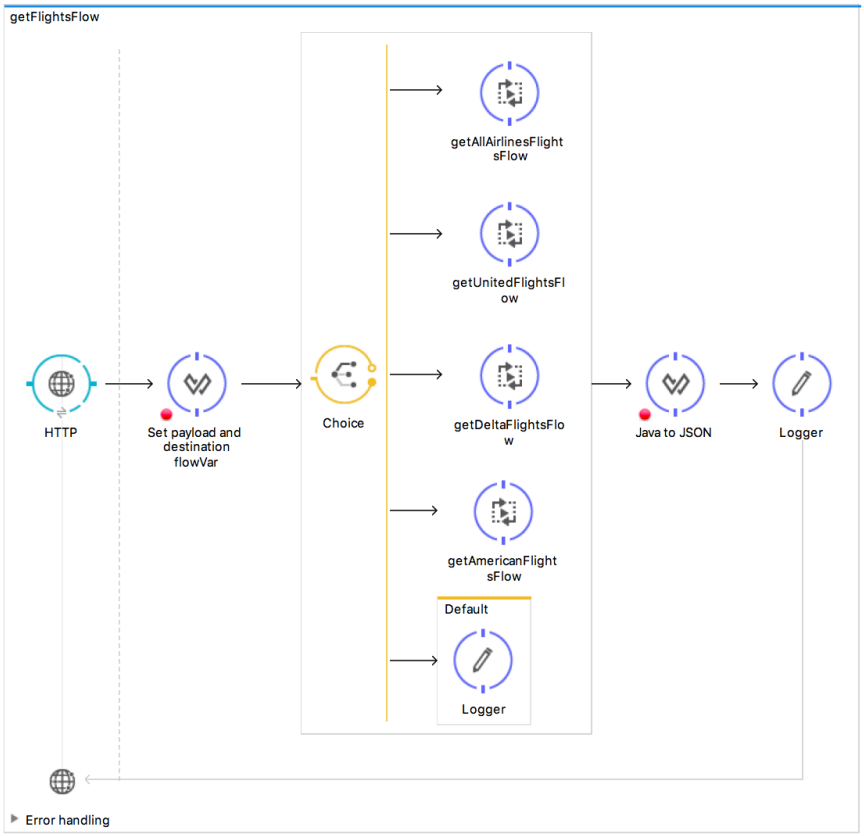
12. In getFlightsFlow, add a Choice flow control element between the Set payload and destination flowVar component and the getAllAirlinesFlightsFlow flow reference.
13. Drag the getAllAirlinesFlightsFlow flow reference into the router.

14. Add a new Logger component to the default branch.



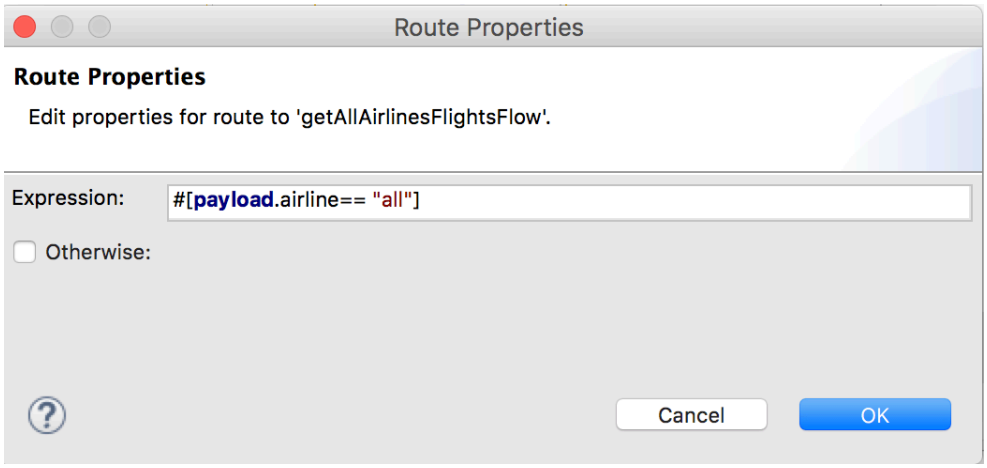
15. Add three additional Flow Reference components to the Choice router to create a total of five branches.
16. In the Properties view for the first new flow reference, set the flow name to getUnitedFlightsFlow.
17. In the Properties view for the second flow reference, set the flow name to getDeltaFlightsFlow.

18. In the Properties view for the third flow reference, set the flow name to getAmericanFlightsFlow.



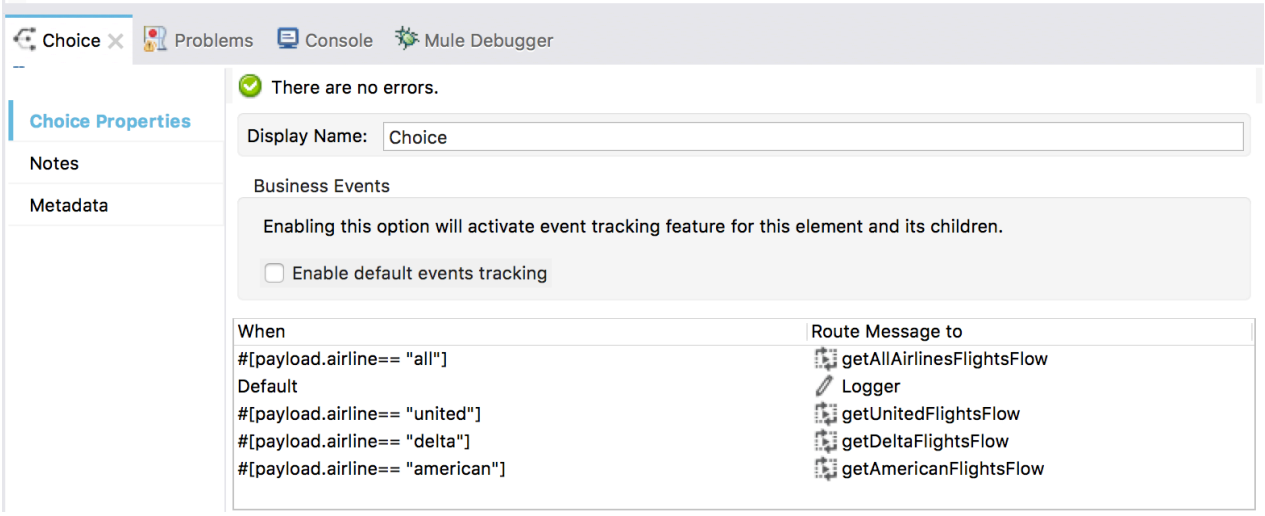
Configure the Choice router

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.

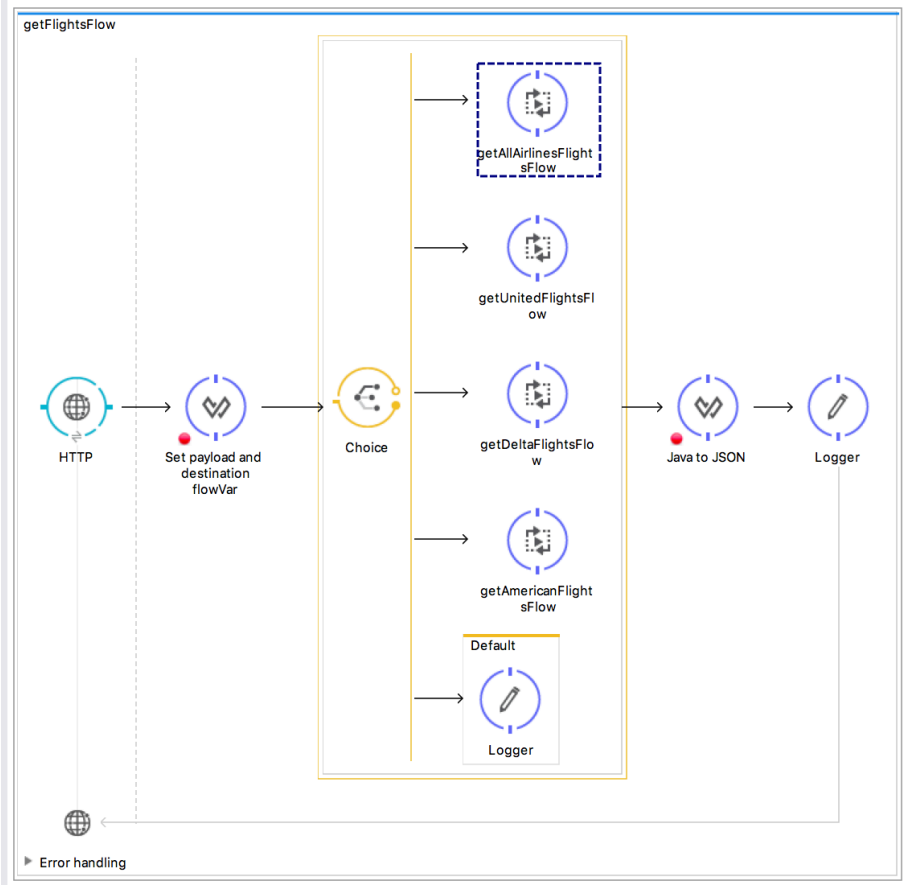
23. Set a similar expression for the American route, routing to it when payload.airline is equal to american.



Debug the application

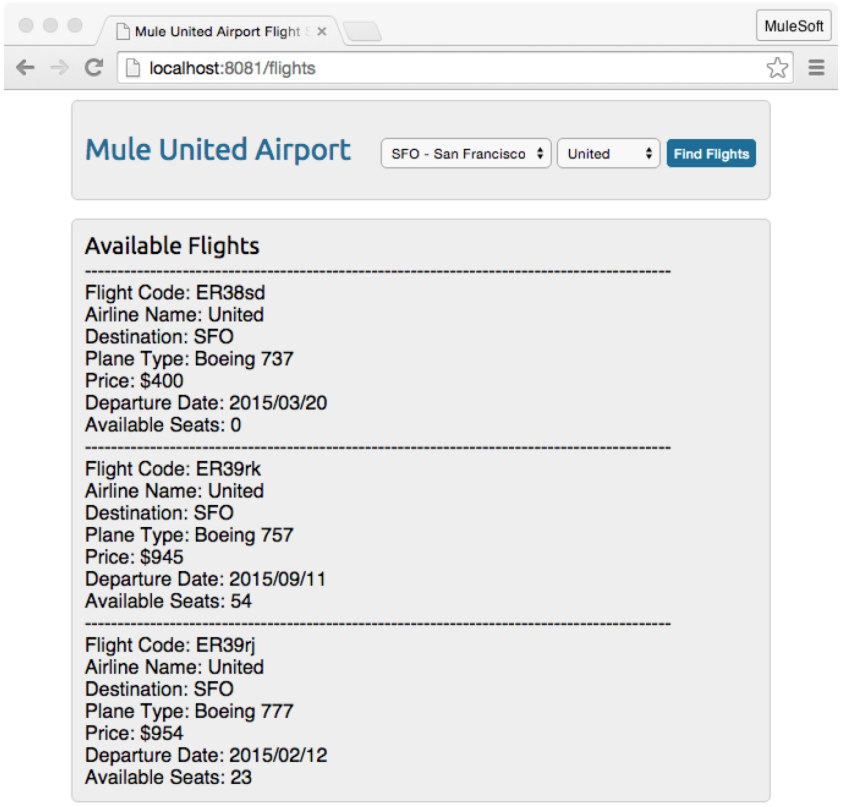
24. Save the file and debug the application.
25. Make a request to http://localhost:8081/flights.
26. On the form page, leave SFO and All Airlines selected and click Find Flights.

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.



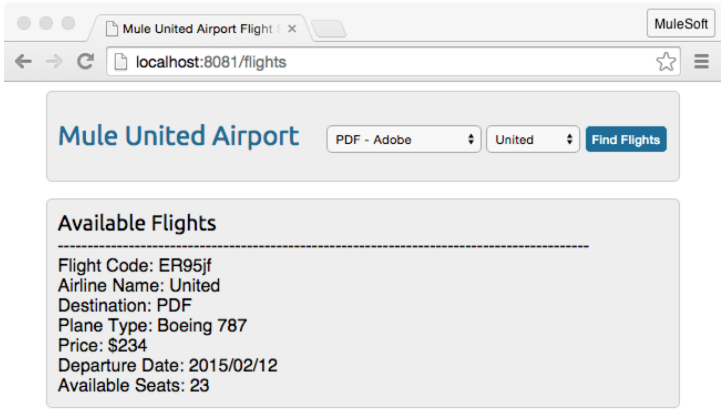
28. Click the Resume button until flight data for all airlines is returned back to the form page.
29. On the form page, select SFO and United and click Find Flights.
30. Return to the Mule Debugger view and step through the application; you should see the Choice router pass the message to the United branch.

31. Click the Resume button until flight data is returned back to the form page; you should see only United flights.

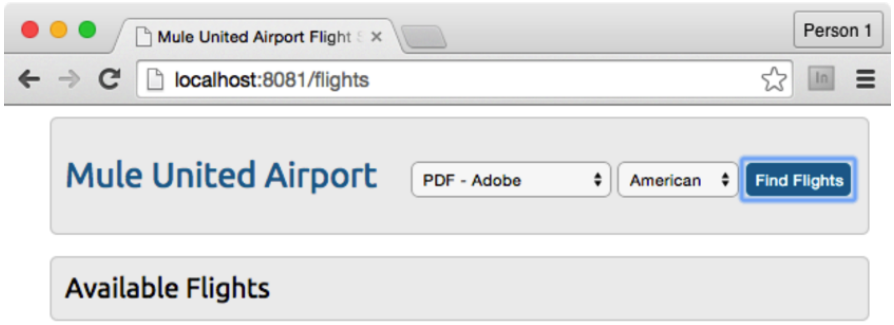


Test the application

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.



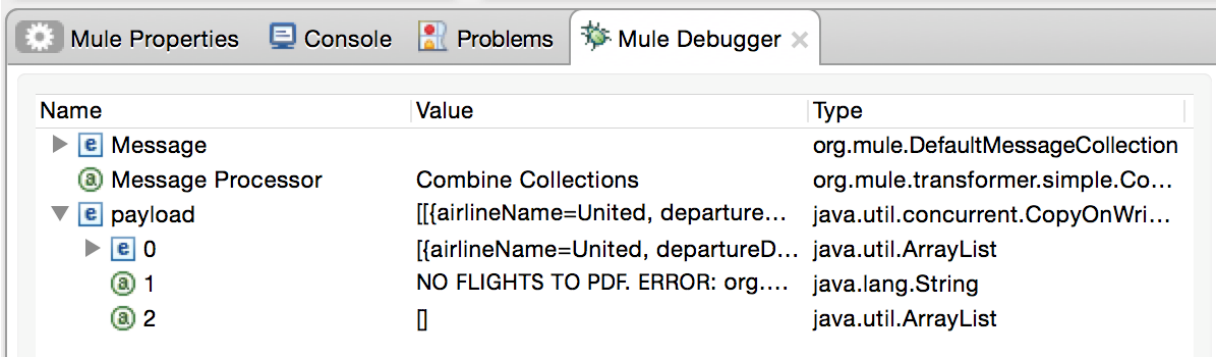
36. Test the PDF location with Delta; you should get an error in the console.
37. Test the PDF location with American; you should get no error, but no results.



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.



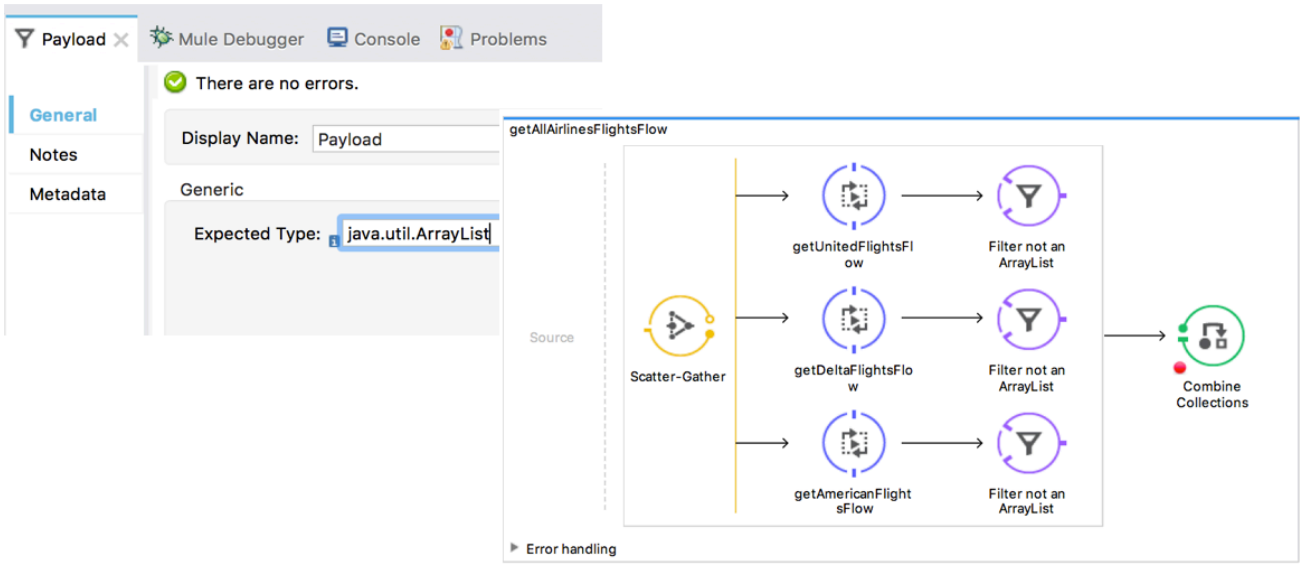
42. Step to the Java to JSON component; you should see the payload still contains the error message in addition to the valid flight results to PDF.
43. Step to the end of the application; you should not get the United results to PDF.

    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

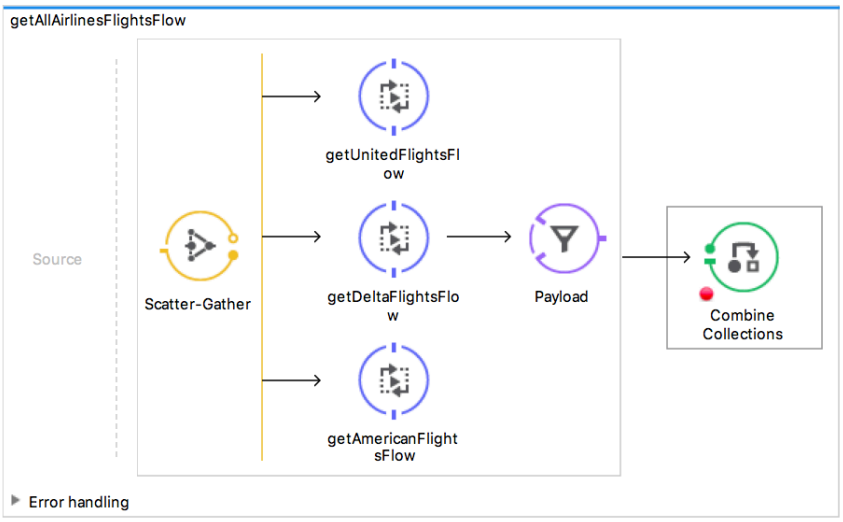
In this walkthrough, you will continue to work with the airline flight results. You will:

- 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.

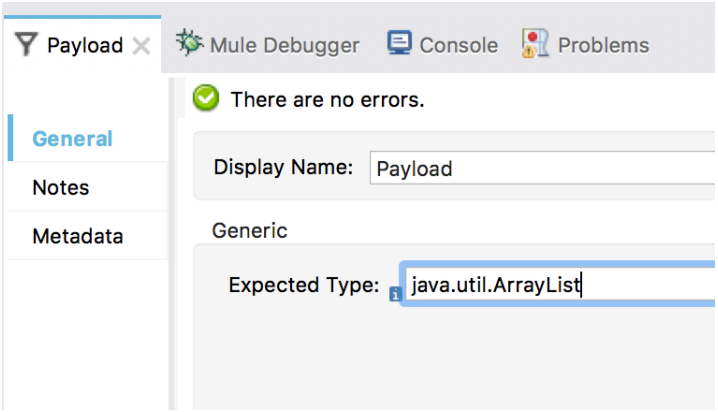


Add a filter

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.

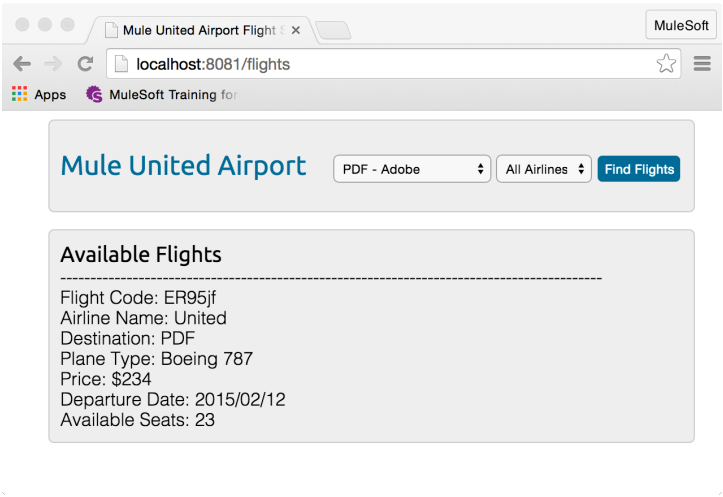


3. In the Payload Properties view, set the expected type to java.util.ArrayList.



Test the application

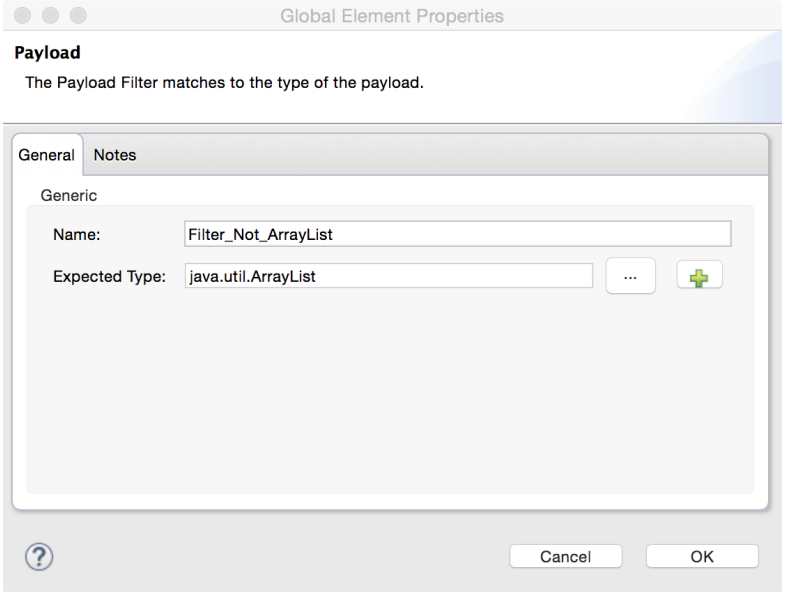
4. Save the file and debug the application.
5. Make a request to http://localhost:8081/flights and submit the form with PDF and All Airlines.
6. Step through the Java to JSON component; this time you should see the payload only contains the ArrayList and not the error string.
7. Step to the end of the application; you should see the United results to PDF returned to the form.



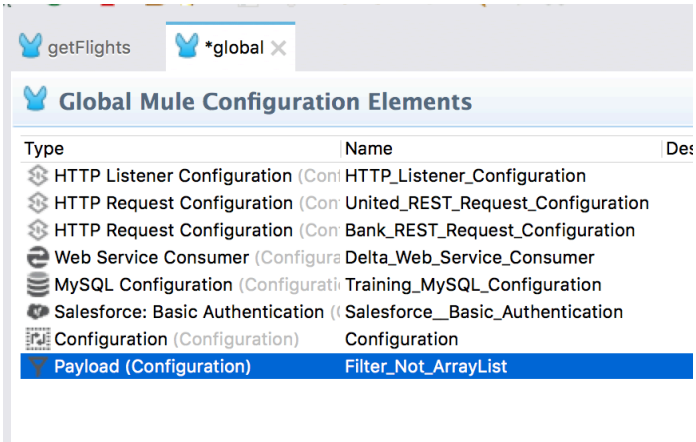
Create a global filter

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.



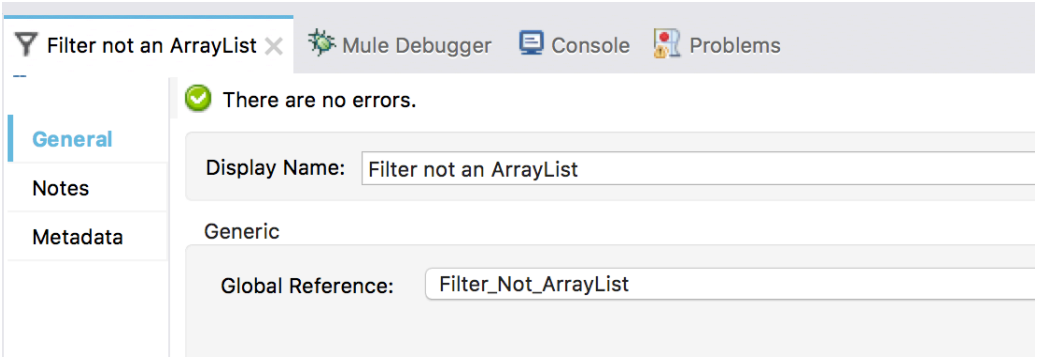
14. See your new global filter in the list.



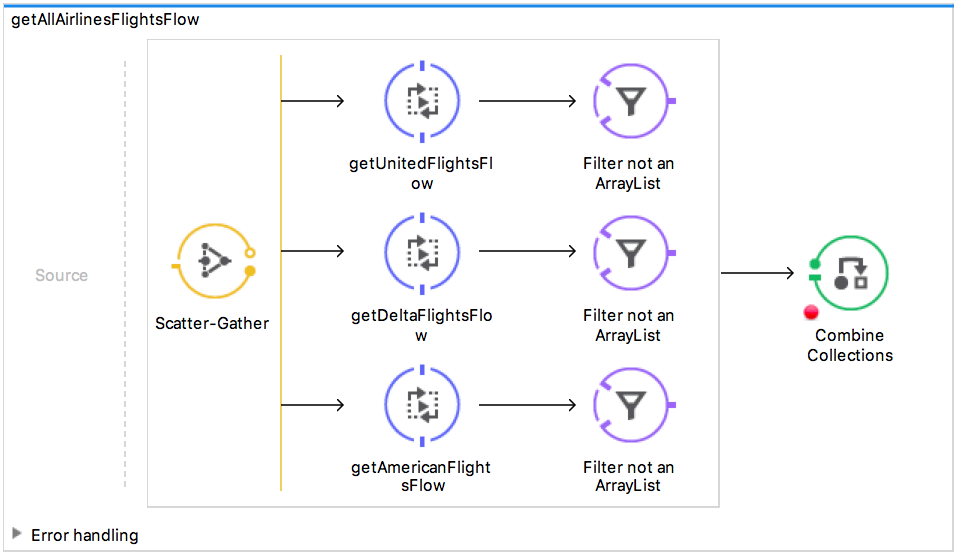
Use the global filter

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.



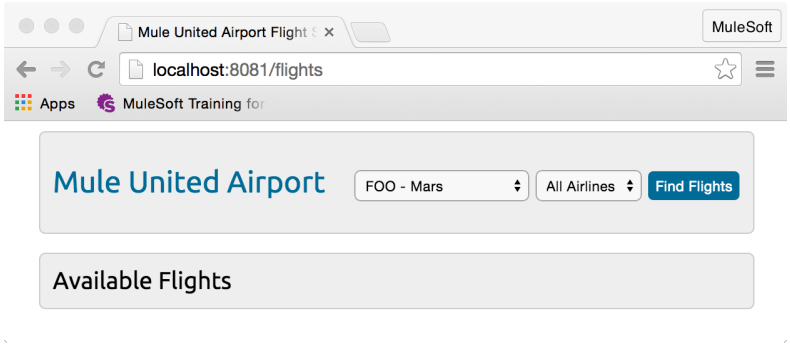
18. Add a Filter Reference after getAmericanFlightsFlow.
19. In the Filter Reference Properties view, set the display name to Filter not an ArrayList and set the global reference to Filter_Not_ArrayList.
20. Copy and paste the Filter Reference in the Scatter-Gather.
21. Drag it after getUnitedFlightsFlow 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

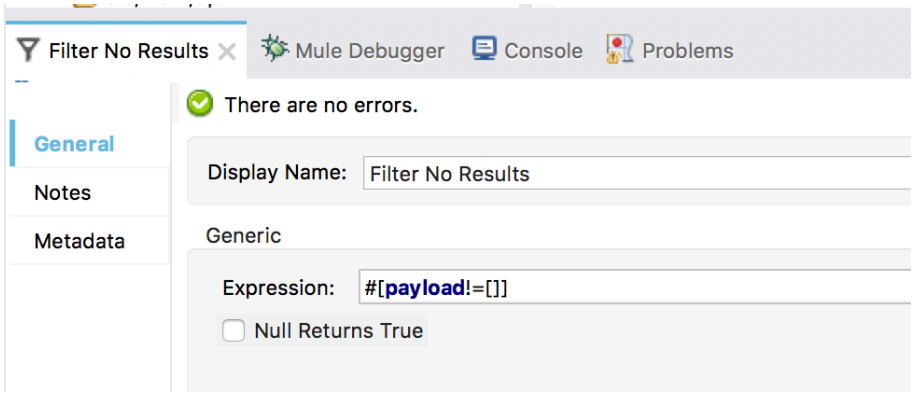
25. In Anypoint Studio, return to or open FlightFinder.html.
26. Locate the code that sets the text if there is no flights are returned.

```
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

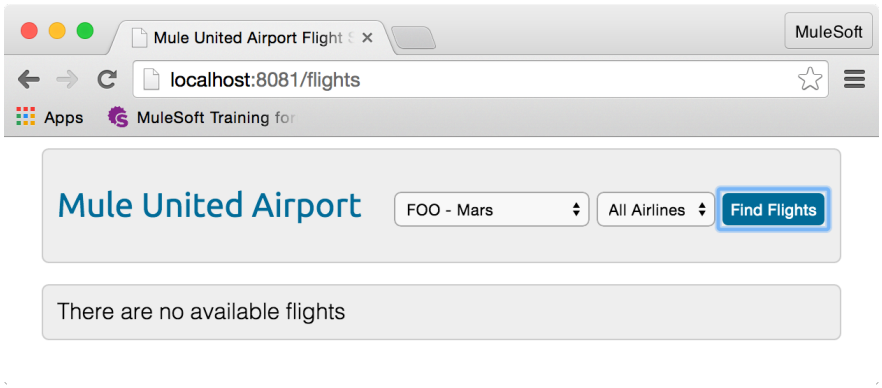
27. In the getFlightsFlow, add an Expression filter after the Choice router.
28. In the Expression Properties view, set the display name to Filter No Results.
29. Set the expression to see if the payload is equal to an empty Array.

```
#[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.



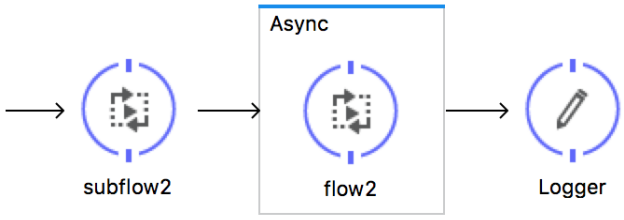
32. Test the PDF location with American; you should get the no available flights message.

    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

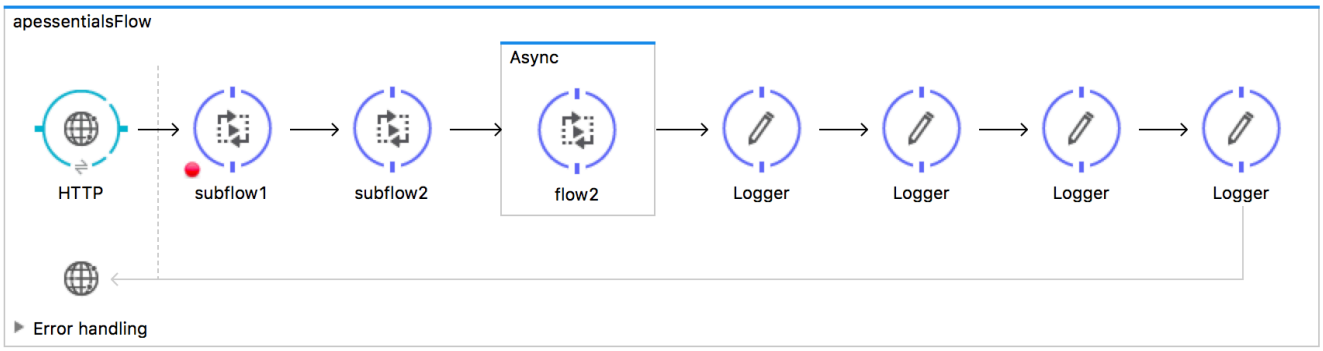
In this walkthrough, you will work in the apessentials2 project and create and pass messages to an asynchronous flow. You will:

- Use the Async scope element to create an asynchronous flow.
- Use the Mule Debugger to watch messages flow through both flows.



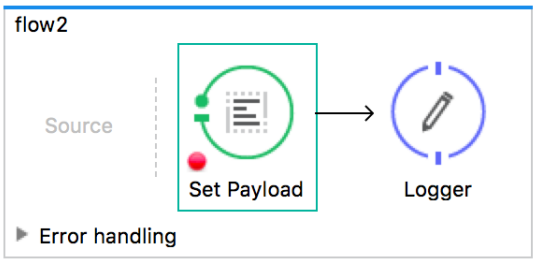
Create an asynchronous flow

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.

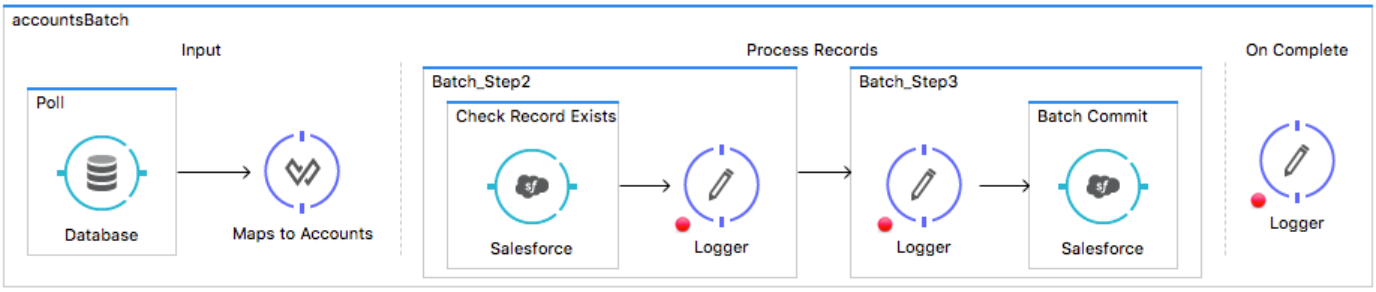
9. Make sure there is a breakpoint on the Set Payload in flow2.



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



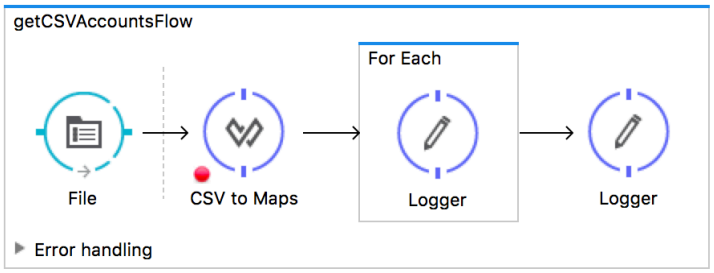
In this module, you will learn:

- 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

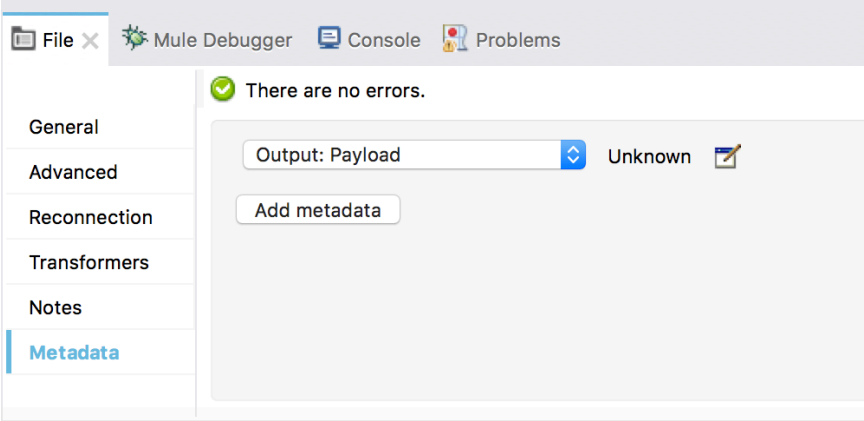
1. Open accounts.xml.
2. In the Package Explorer, expand the src/main/resources/input folder.
3. Right-click accounts.csv.backup and select Refactor > Rename.
4. In the Rename Resource dialog box, set the new name to accounts.csv and click OK.
5. In getCSVAccountsFlow, delete the File-to-String transformer.
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.

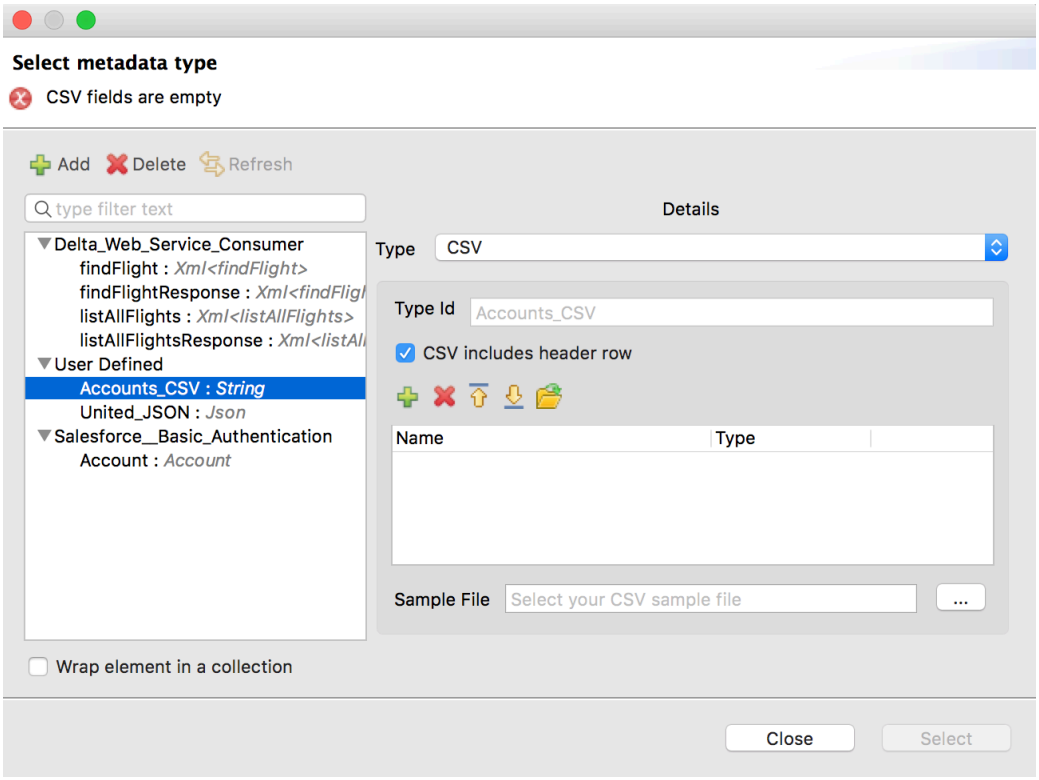
Add File endpoint metadata

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.

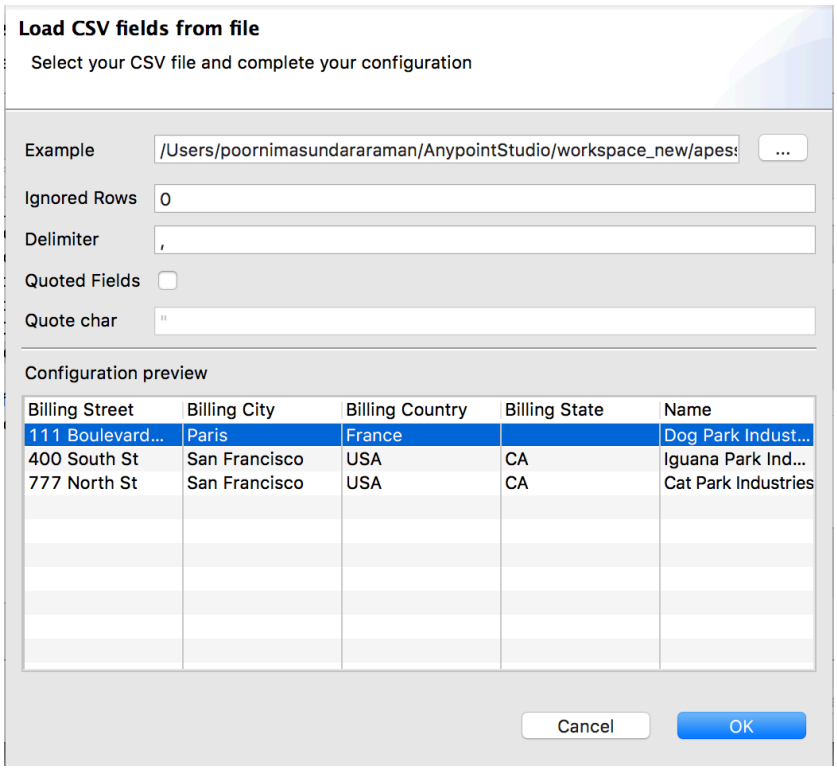


10. Click the Edit button next to the drop-down menu.
11. In the Define Type dialog box, select Create new type.
12. Set the Type Id to Accounts_CSV.
13. Click Create type.
14. Change the Type to CSV.
15. Make sure CSV includes header row is checked.
16. Click the Load from example button (the folder icon).

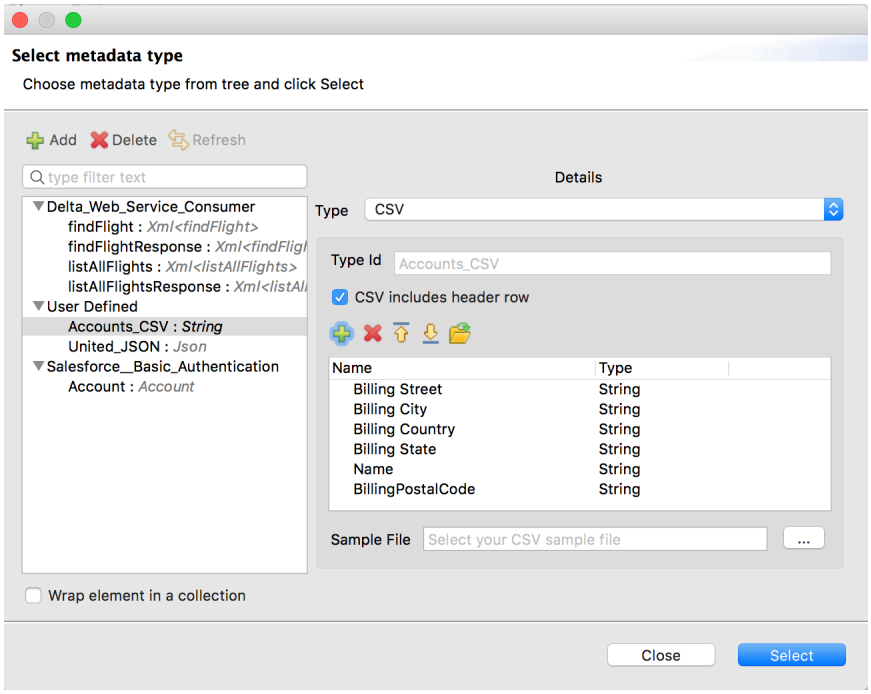


17. In the Load CSV fields from file dialog box, click the Browse button next to Example.

18. In the Select CSV example dialog box, browse to the project's src/main/resources/input folder, select accounts.csv, and click Open.
19. In the Load CSV fields from file dialog box, click OK.

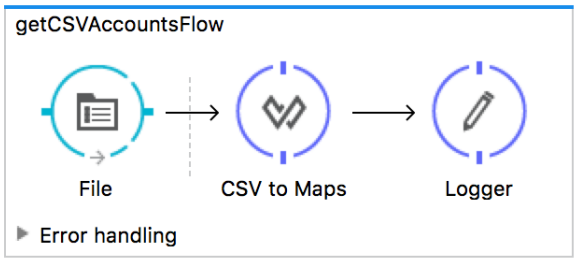


20. In the Select metadata type dialog box, click the Select button.

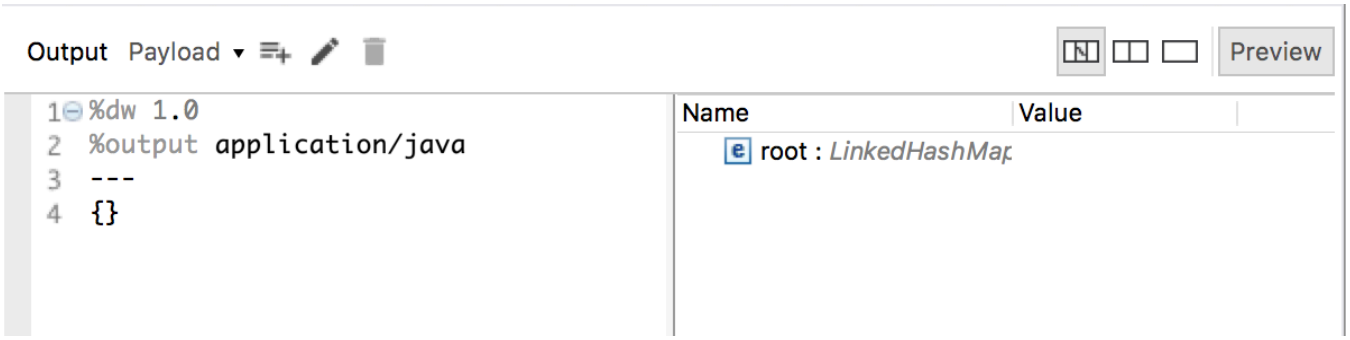


Use DataWeave to convert a CSV file to a collection of objects

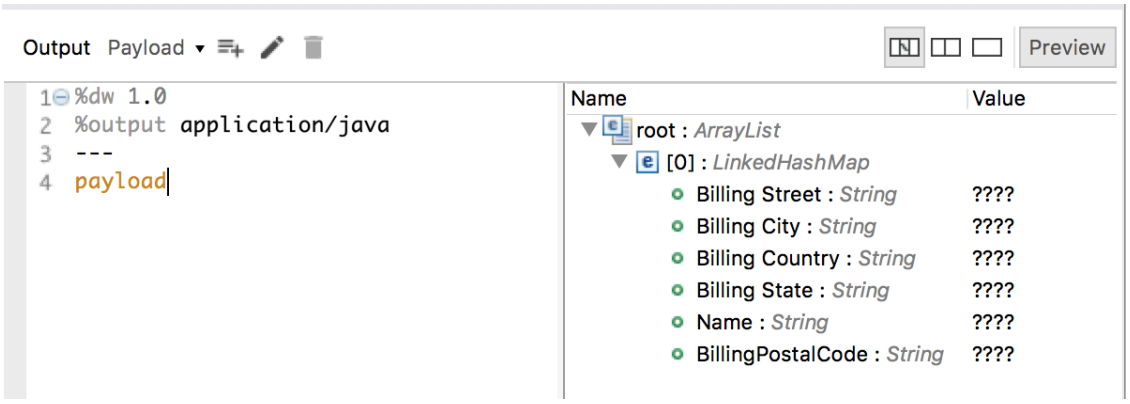
21. Add a Transform Message component between the File and the Logger.
22. Change the name of the component to CSV to Maps.



23. In the Input section of the Transform Message Properties view, make sure the Payload is specified as a List<Csv> with appropriate fields.
24. Right click on the Payload in the Input section and click Edit Sample Data.
25. Look at the Transform section, the output should be set to application/java.
26. Look at the Preview tab in the Output section, the Payload should be set to a Map.

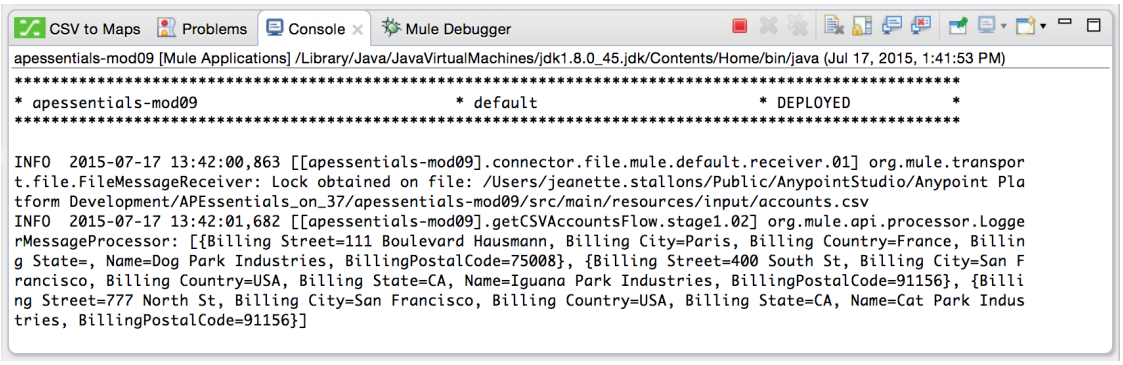


27. Write a DataWeave expression to transform the payload without making any changes to its structure or values.
28. Look at the preview.



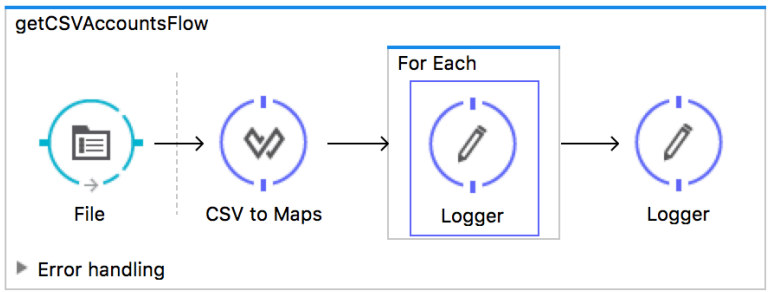
29. Run the application.

30. Return to the console in Anypoint Studio; you should see the values for all the accounts in the CSV file displayed.



Add a For Each scope element

31. Drag a For Each scope element form the palette and drop it after the Transform Message component.
32. Add a Logger to the For Each scope.



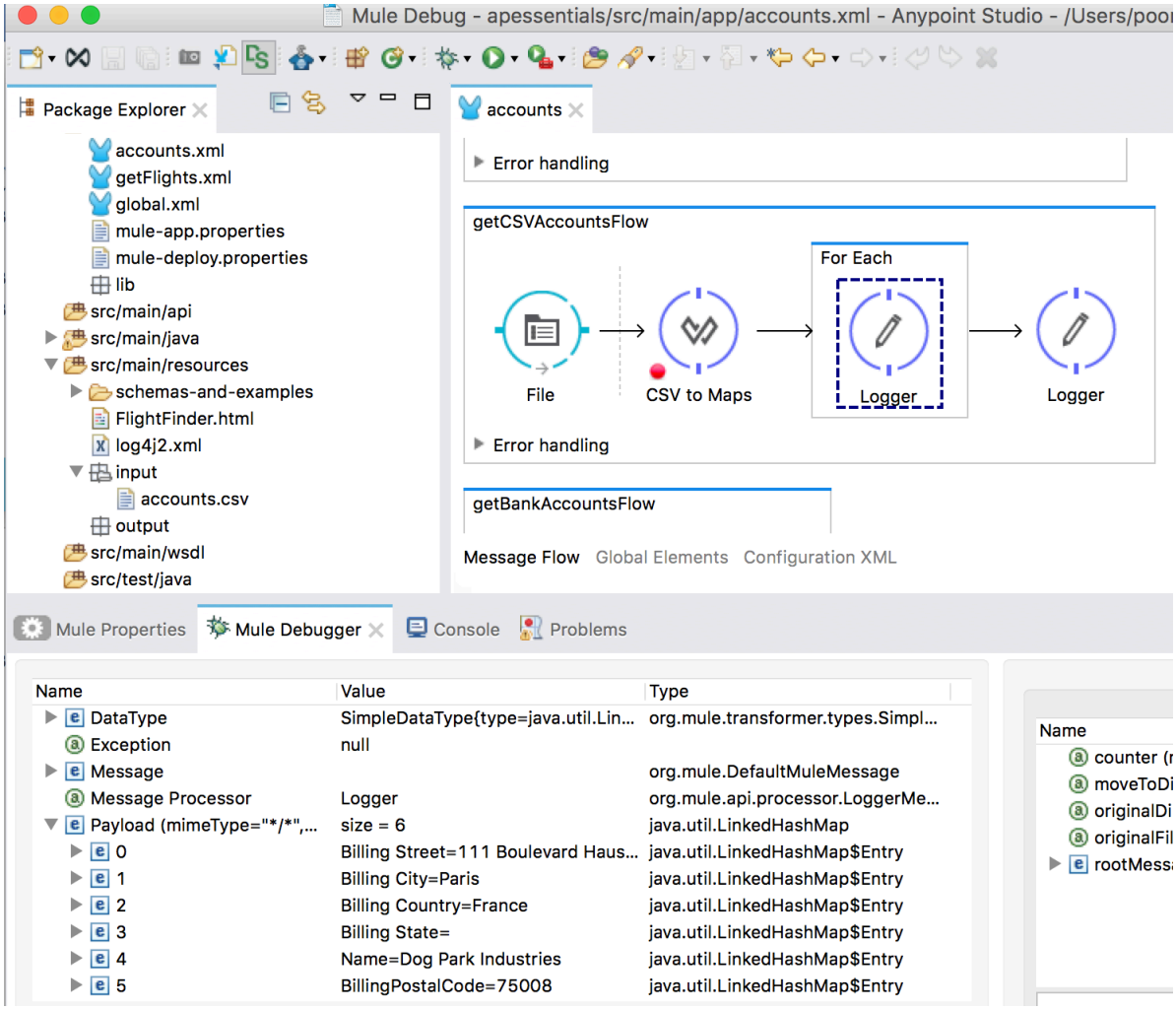
Process each element

33. In the Logger Properties view, set the message to #[payload].

Debug the application

34. Add a breakpoint to the Transform Message component.
35. Save the file and debug the application.
36. 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.
37. In the Mule Debugger view, watch the payload value.
38. Step to the For Each scope; the payload should be an ArrayList of LInkedHashMaps.

39. Step through the For Each; the payload should be a LinkedHashMap.



40. Watch the console as you step through; you should see each record displayed.

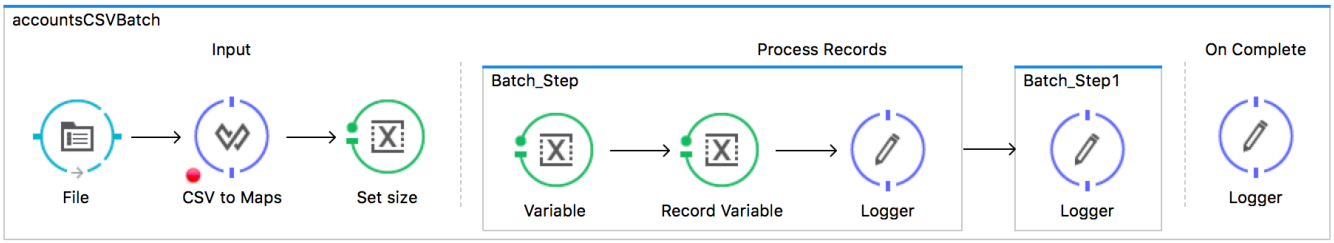
```
INFO  2015-07-17 13:47:29,712 [[apessentials-mod09].connector.file.mule.default.receiver.01] org.mule.transpor
t.file.FileMessageReceiver: Lock obtained on file: /Users/jeanette.stallons/Public/AnypointStudio/Anypoint Pla
tform Development/APEssentials_on_37/apessentials-mod09/src/main/resources/input/accounts.csv
INFO  2015-07-17 13:48:44,456 [[apessentials-mod09].getCSVAccountsFlow.stage1.02] org.mule.api.processor.Logge
rMessageProcessor: {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.Logge
rMessageProcessor: {Billing Street=400 South St, Billing City=San Francisco, Billing Country=USA, Billing Stat
e=CA, Name=Iguana Park Industries, BillingPostalCode=91156}
INFO  2015-07-17 13:48:50,220 [[apessentials-mod09].getCSVAccountsFlow.stage1.02] org.mule.api.processor.Logge
rMessageProcessor: {Billing Street=777 North St, Billing City=San Francisco, Billing Country=USA, Billing Stat
e=CA, Name=Cat Park Industries, BillingPostalCode=91156}
```

41. 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

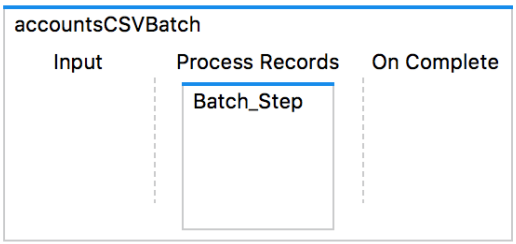
In this walkthrough, you will create a batch job to process records in a CSV file. You will:

- 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

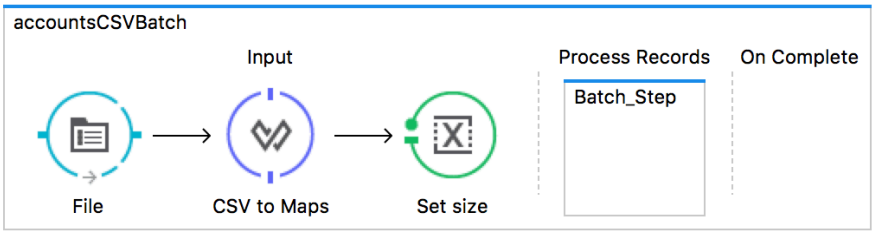
1. Return to accounts.xml.
2. Drag a Batch scope element from the palette and drop it above the getCSVAccountsFlow.
3. Change the batch job name to accountsCSVBatch.



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.

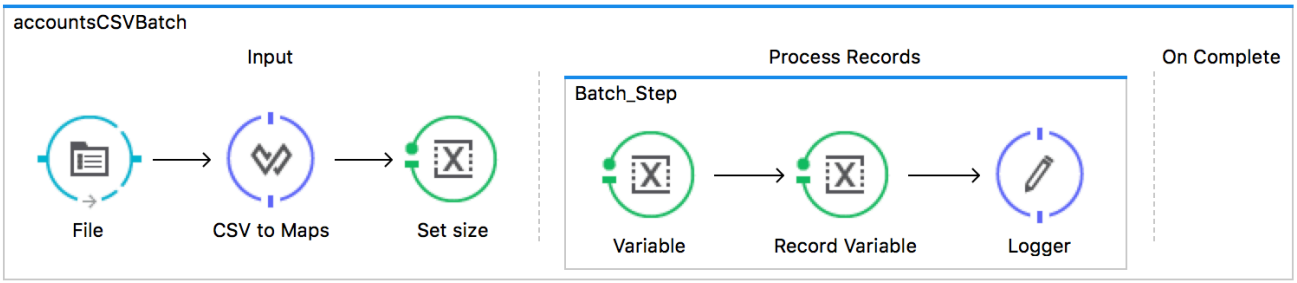
6. Rename the elements to remove Copy_of_ from their names.
7. Add a Variable transformer after the transformer in the input phase.
8. In the Variable Properties view, change the display name to Set size and select Set Variable.



9. Set the name to size and the value to #[payload.size()].

Add processors to a batch step in the process records phase

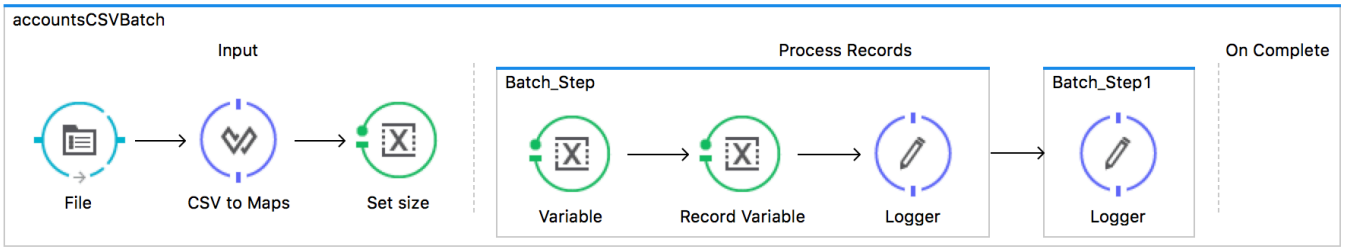
10. Add a Variable transformer to the batch step in the process records phase.
11. In the Properties view, select Set Variable and set the name to fname and the value to #[payload.Name].
12. Add a Record Variable transformer to the batch step.
13. In the Properties view, select Set Record Variable and set the name to rname and the value to #[payload.Name].
14. Add a Logger component to the batch step.



Create a second batch step

15. Drag a Batch Step scope element from the palette and drop it in the process records phase after the first batch step.

16. Add a Logger to the second batch step.



Add processors to the on complete phase

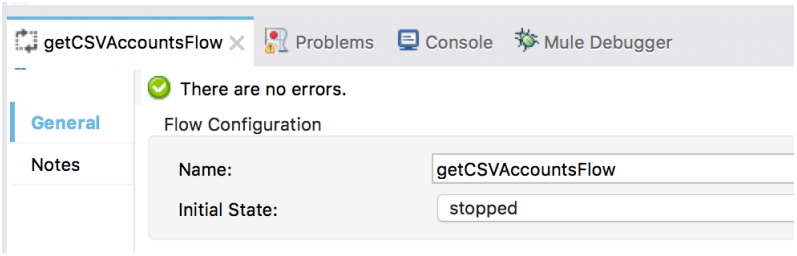
17. Add a Logger component to the on complete phase.
18. In the Logger Properties view, set the message to display the number of processed records and failed records.

    ```
    #['\n\nProcessed: ' + payload.processedRecords + ' Failed: ' + payload.failedRecords]
    ```

    Note: If you want to add line breaks, add one or more '\n' to your MEL expression – meaning it must be inside the #[].

Stop the getCSVAccountsFlow from running

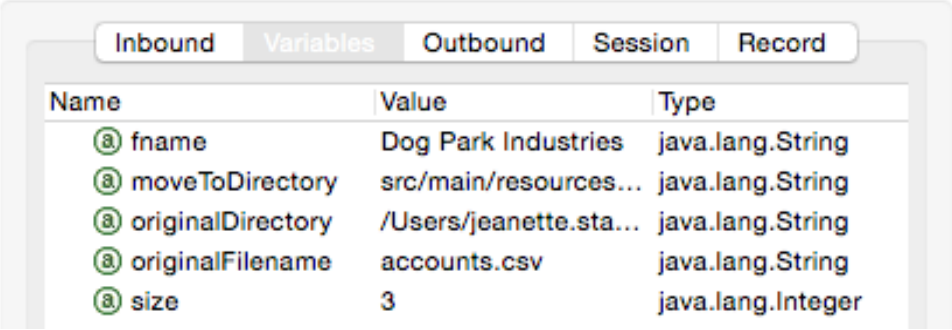
19. Double-click getCSVAccountsFlow.
20. In the Properties view, set the initial state to stopped.



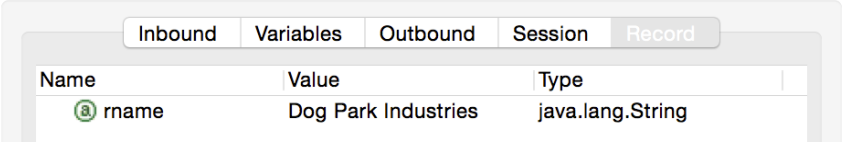
Debug the application

21. Add a breakpoint to the CSV to Maps component in accountCSVBatch.
22. Save the file to redeploy the application in debug mode.
23. After the application starts, drag the accounts.csv file from the output folder to the input folder.
24. In the Mule Debugger view, watch the payload value.
25. Step into the process records phase.
26. Click the Variables tab; you should see the size flow variable.

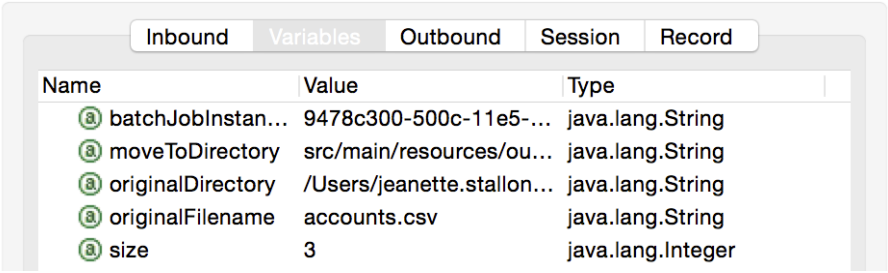
27. Step to the Logger in the first batch step; you should see the fname flow variable.



28. Click the Record tab; you should see the rname flow variable.

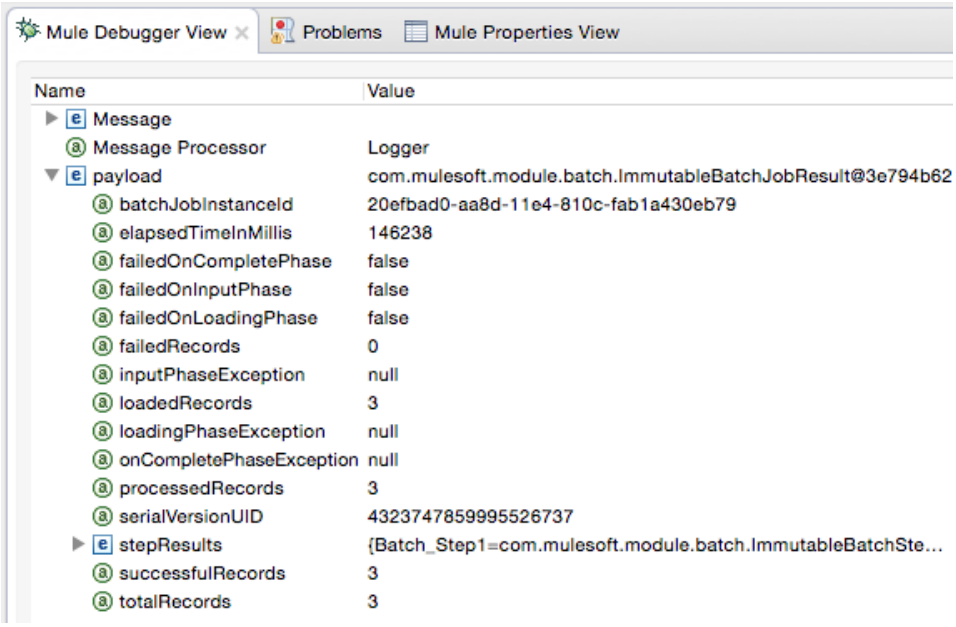


29. Click the Variables tab.
30. Step through the rest of the records in the first batch step and watch the payload, the fname flow variable, and the rname record variable.
31. Step into the second batch step and look at the payload, the flow variables, and the record variables; you should see the rname record variable and size flow variable but not the fname flow variable.



32. Step through the rest of the records in the second batch step.

33. Step into the on complete phase; you should see the payload is an ImmutableBatchJobResult.



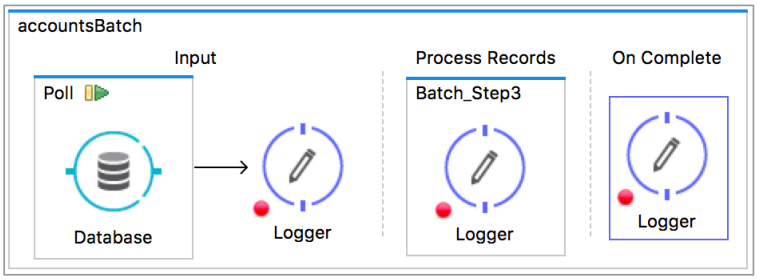
34. Step to the end of the application; you should see the number of process records phase and failed records in the console.

```
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

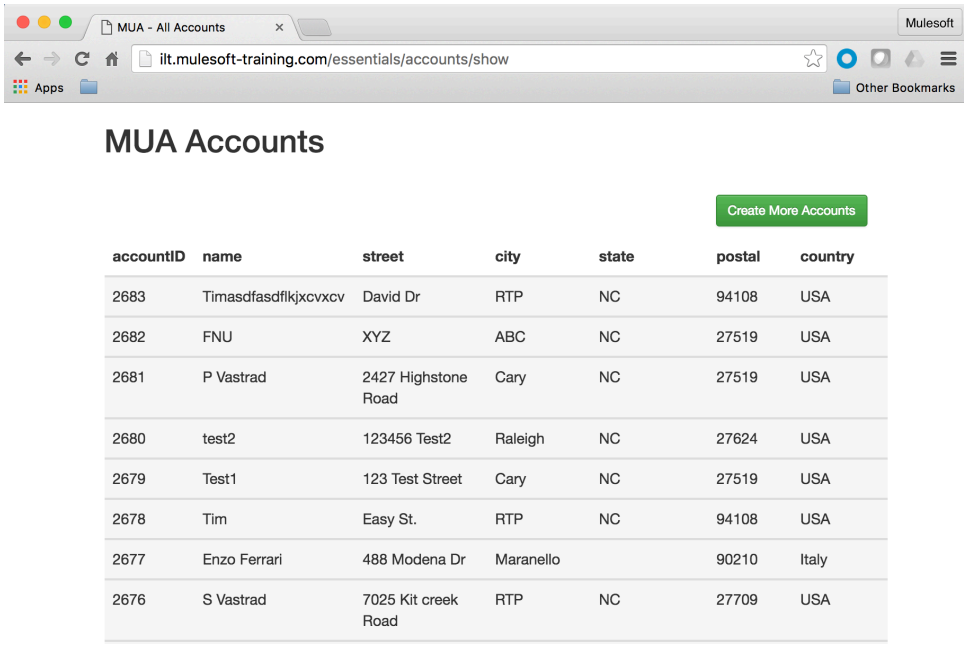
In this walkthrough, you will create a batch job that retrieves records from a legacy database. You will:

- 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.



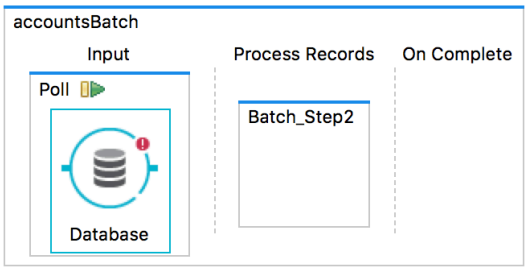
accountID	name	street	city	state	postal	country
2683	Timasdfasdflkjxcvxcv	David Dr	RTP	NC	94108	USA
2682	FNU	XYZ	ABC	NC	27519	USA
2681	P Vastrad	2427 Highstone Road	Cary	NC	27519	USA
2680	test2	123456 Test2	Raleigh	NC	27624	USA
2679	Test1	123 Test Street	Cary	NC	27519	USA
2678	Tim	Easy St.	RTP	NC	94108	USA
2677	Enzo Ferrari	488 Modena Dr	Maranello		90210	Italy
2676	S Vastrad	7025 Kit creek Road	RTP	NC	27709	USA

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

5. Return to accounts.xml.
6. Drag a Batch scope element onto the canvas from the palette to create a new batch job.
7. Add a Poll scope element to the input phase.
8. In the Properties view, select the fixed frequency scheduler.
9. Set the frequency to 30 and the time unit to seconds.
10. Add a Database connector to the poll.



11. Set the connector configuration to the existing Training_MySQL_Configuration global element.
12. Set the operation to Select.
13. 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

14. Add a Logger component to the Process Records phase.
15. Display the record – the message payload – and other text or information you wish.

```
#['\n\nRECORD: ' + payload]
```

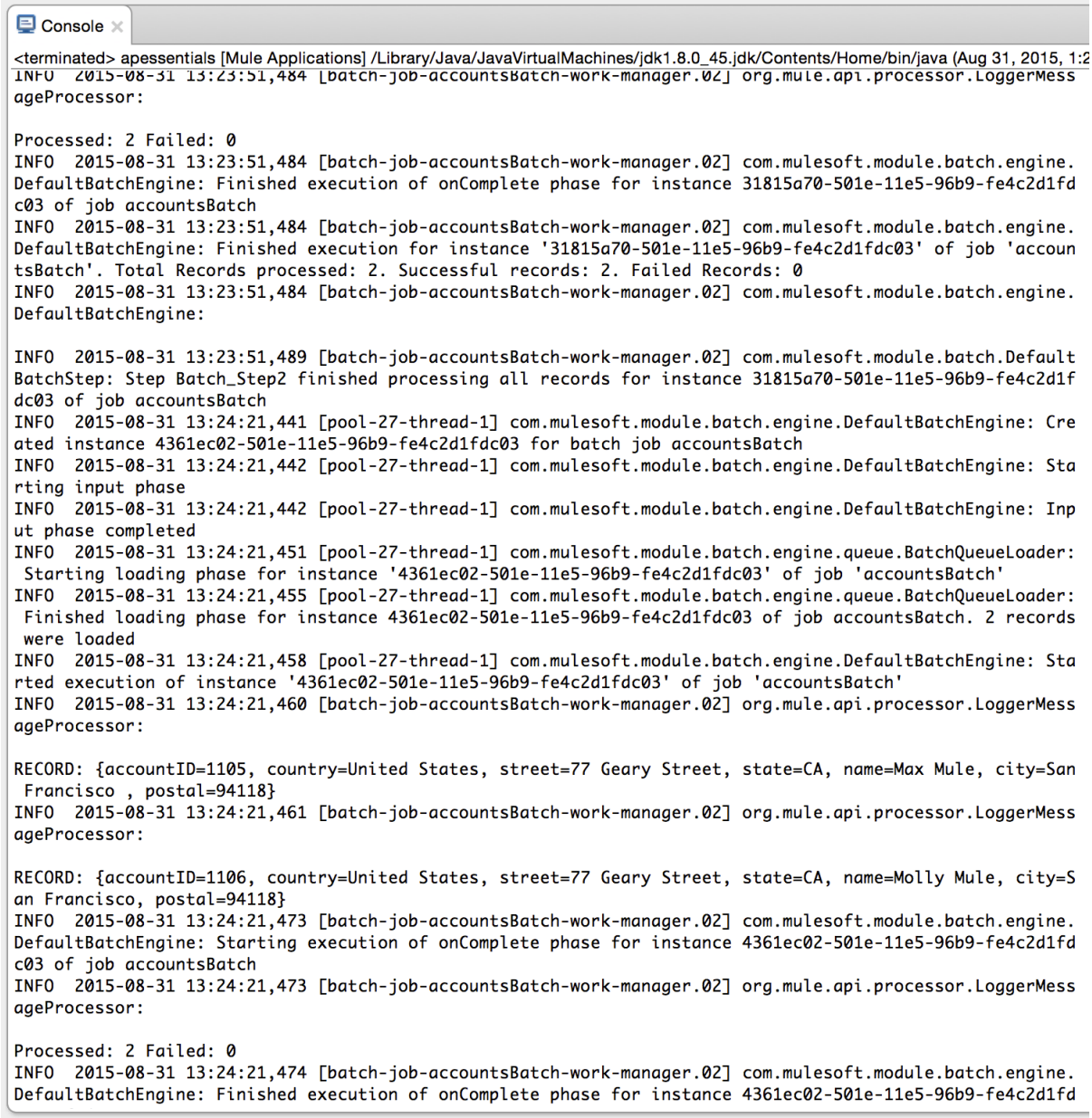
Log processed records in on complete phase

16. Add a Logger component to the on complete phase.
17. Set the Logger message to display the number of processed records and failed records.

```
#['\n\nProcessed: ' + payload.processedRecords + ' Failed: ' +
payload.failedRecords]
```

Test the application

18. Run the application and watch the console; you should see records displayed every 30 seconds.



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:

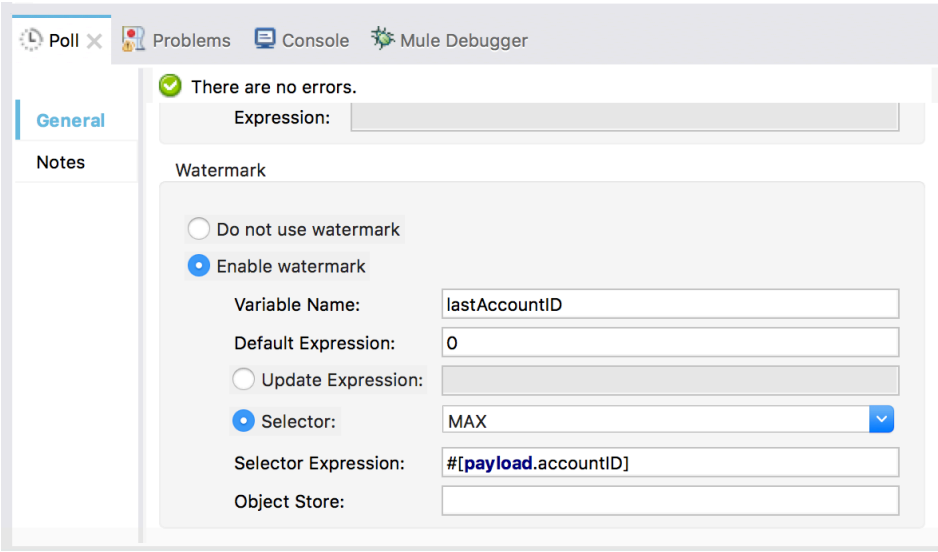
- Modify the Poll to use a watermark to keep track of the last record returned from the database.
- Modify the database query to use the watermark.
- Clear application data.

Parameterized query:

```
SELECT *
FROM accounts
WHERE postal = '94108' AND accountID > #[flowVars.lastAccountID]
```

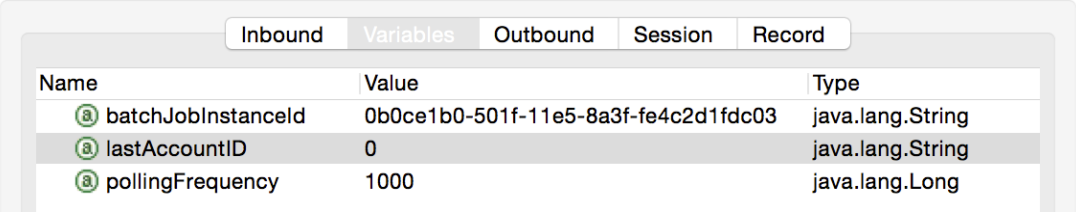
Use a watermark to keep track of the last record

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]

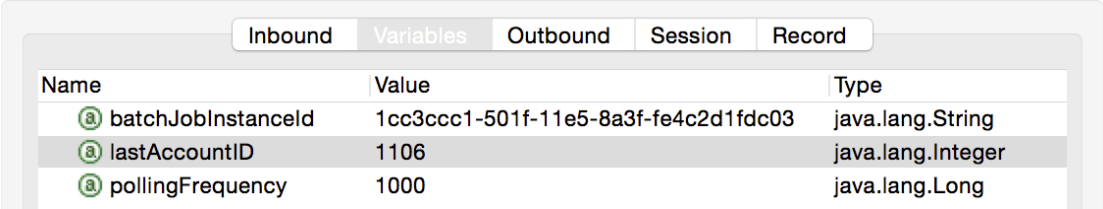


Debug and examine the watermark value

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.



7. Run the application 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.



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

9. 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

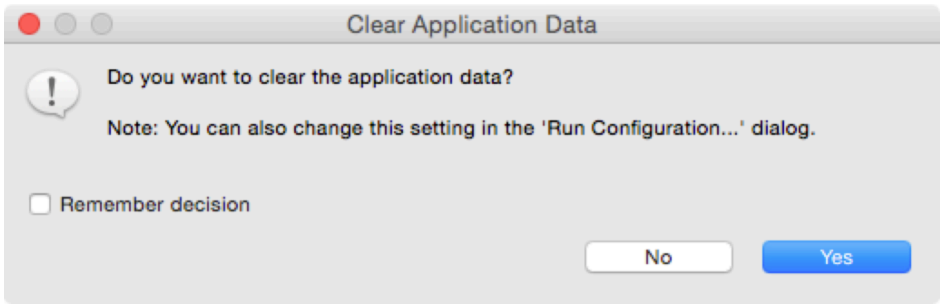
10. Run the application and look at the console; you should see that no records are retrieved at all this time.

```
Processed: 0 Failed: 0
INFO  2015-08-31 13:33:19,804 [pool-27-thread-1] com.mulesoft.module.batch.engine.DefaultBatchEngine: Fin
ished execution of onComplete phase for instance 843f092c-501f-11e5-802f-fe4c2d1fdc03 of job accountsBatc
h
```

Note: By default, the watermark is stored in a persistent object store so its value is retained between different executions of the application.

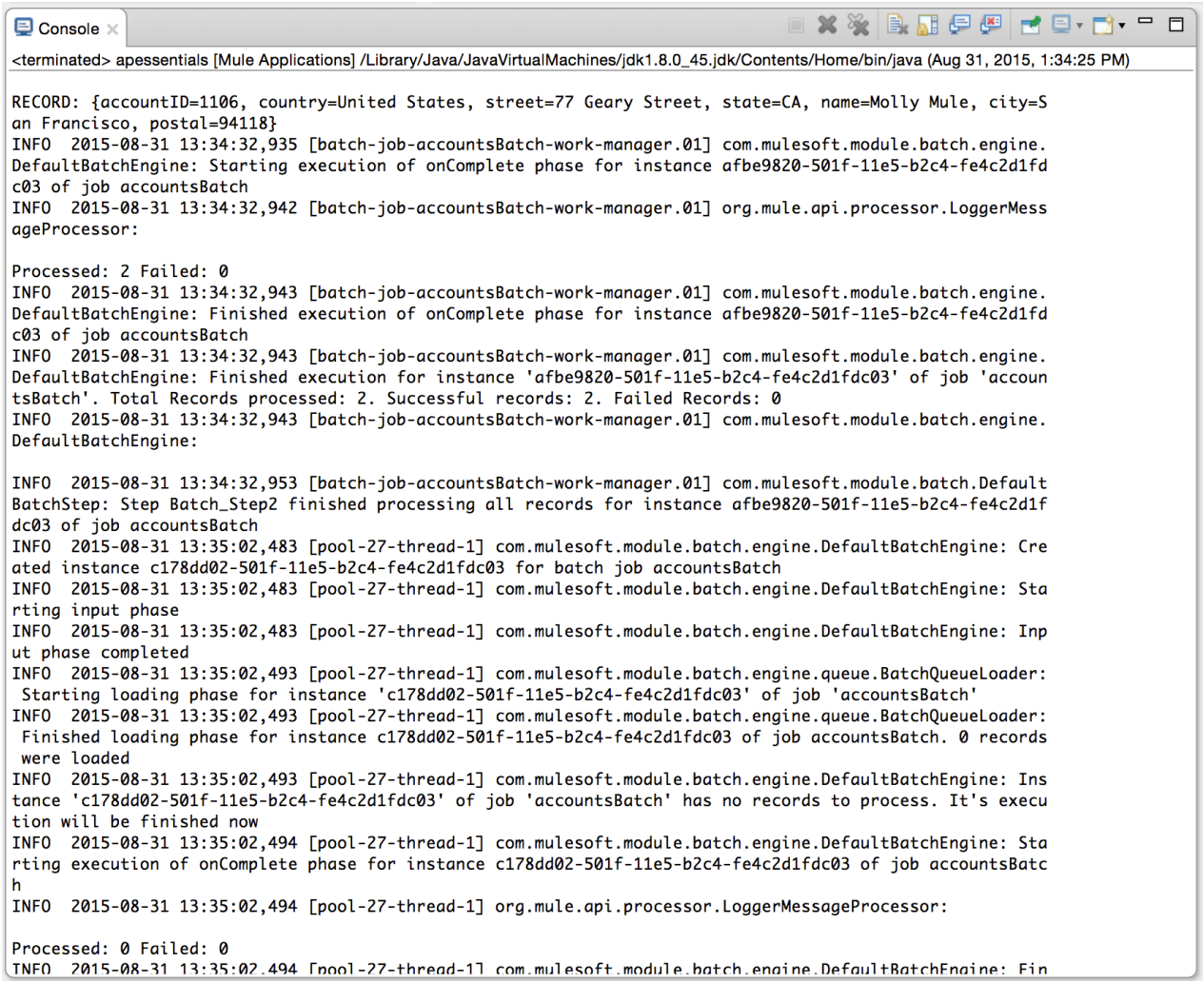
Clear application data

11. Select Run > Run Configurations.
12. Make sure your apessentials project is selected and then on the General tab, change Clear Application Data from Never to Prompt.
13. Run or debug your application again; you should be prompted to clear the application data.
14. Click Yes.



15. Look at the console; you should see the latest matching records retrieved from the legacy database again – but this time, only once.

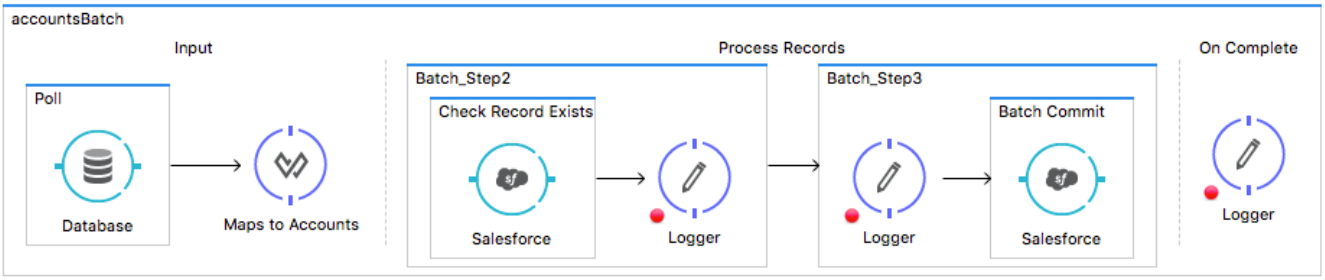
16. Watch the console and see that all subsequent polling events retrieve no records.



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:

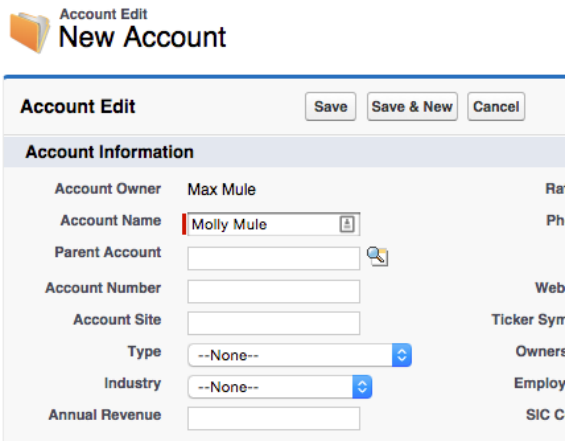
- Add a first batch step with a Message Enricher scope element that checks if a record already exists in Salesforce (an account with the same Name) and stores the result in a record variable and retains the original payload.
- Modify the second batch step to use a filter that only allows new records (records that don't already exist) to be processed.
- (Optional) Add the record(s) to Salesforce.



Add a Salesforce account manually

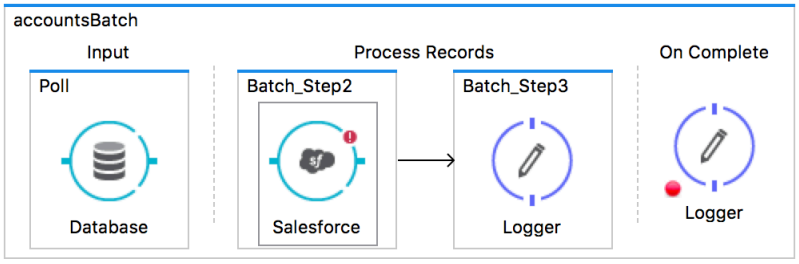
1. In a browser, go to http://login.salesforce.com/ and log in with your Salesforce Developer account.
2. Click the Accounts link in the main menu bar.
3. In the view drop-down, select All Accounts with Postal Code and click the Go button.
4. Click the New Account button.

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.



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.



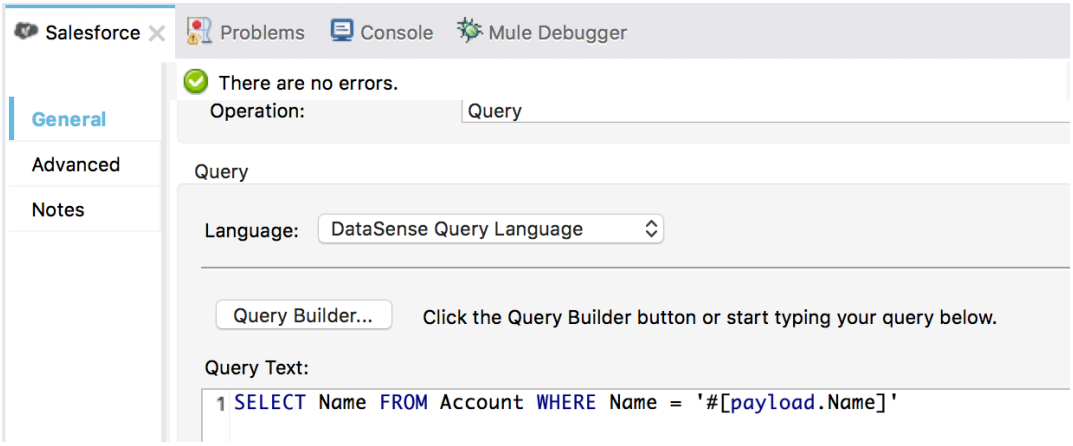
10. Configure the Salesforce connector endpoint to use the existing Salesforce connector configuration.
11. Set the operation to Query.
12. Click the Query Builder button.
13. Set the type to Account (Account).
14. Select a field of Name; it does not matter what you select, you just want to see if any 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]
    ```

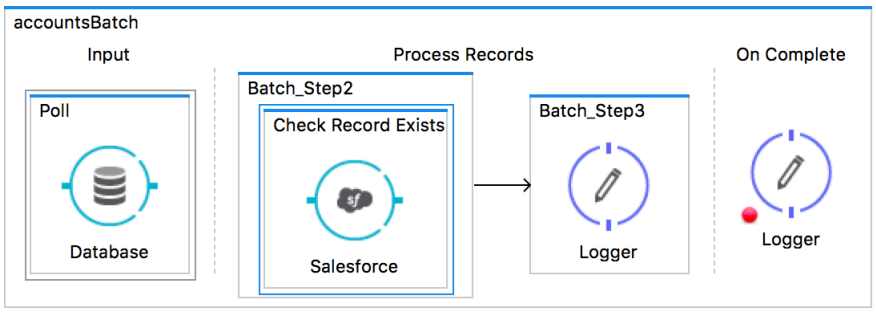
    Note: Right now, you can use payload.name or payload.Name because the payload is a caseSensitiveHashMap. Later this walkthrough, however, you will transform the Map to an Account object with a Name property and will have to refer to this as payload.Name.

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.

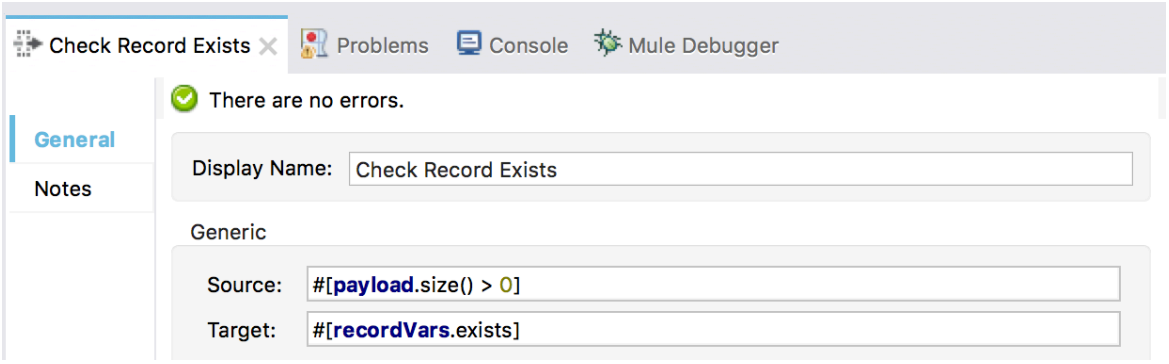


21. Set the source to an expression that checks to see if any records were returned, i.e. if there is a payload.

    ```
    #[payload.size() > 0]
    ```

22. Set the target to assign the result of the source expression to a record variable called exists.

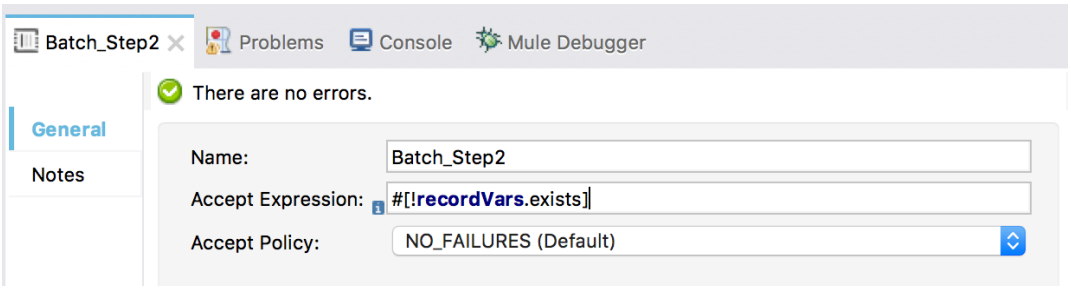
```
#[recordVars.exists]
```



Set a filter for the insertion step

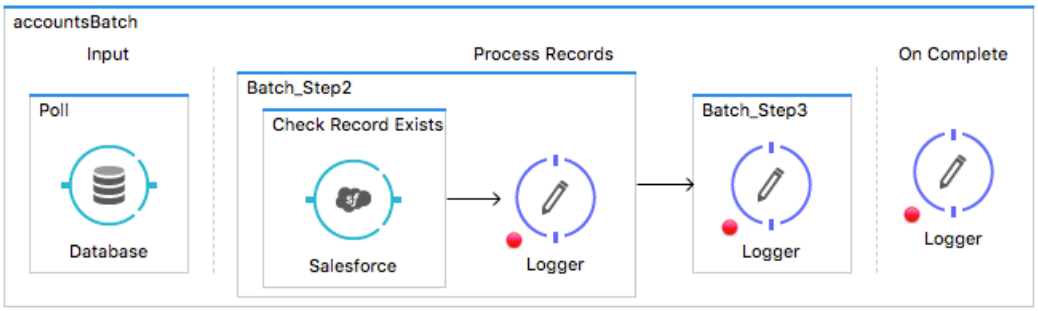
23. Double-click the second batch step that will/would insert the record into Salesforce.
24. In the Batch Step Properties view, set the Accept Expression so that only records that have a record variable exists set to false are processed.

```
#[!recordVars.exists]
```



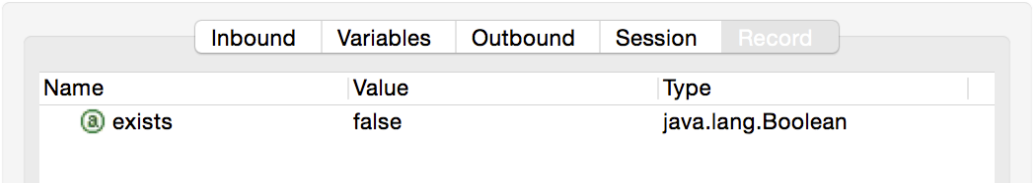
Test the application

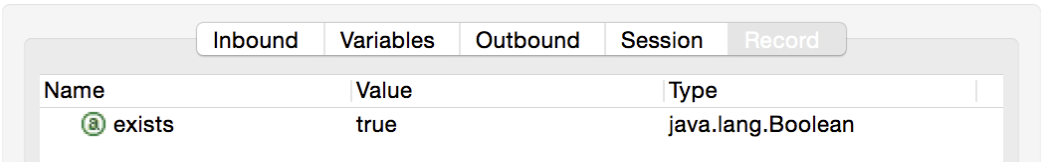
25. Add a breakpoint to the Message Enricher.
26. Add a Logger after the Message Enricher in the first batch step and add a breakpoint to it.



27. Add a breakpoint to the Logger in the second batch step.

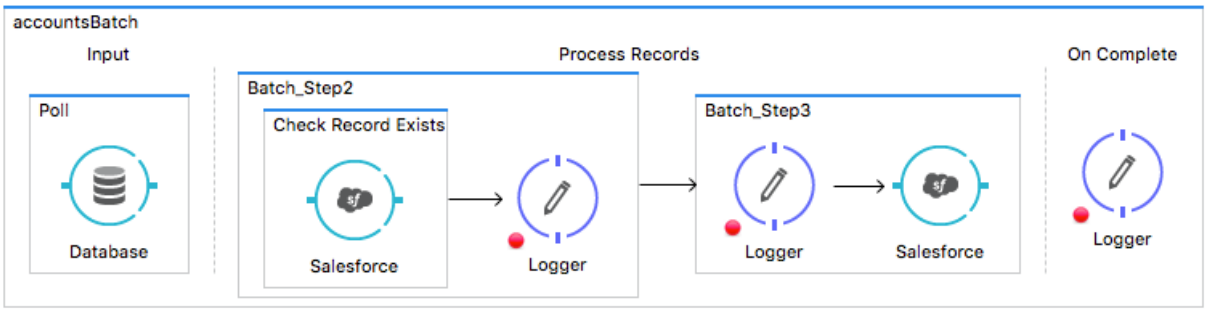
28. Debug the application and clear the application data.
29. Step through the application and watch the record variables; you should see exists set to false for records with names that don't exist in Salesforce and true for those that do.



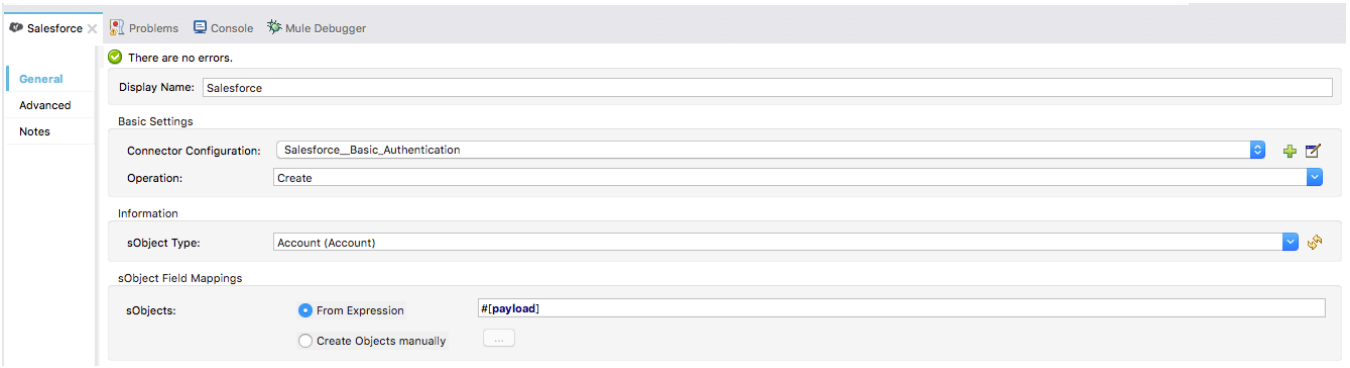


Commit new account records to Salesforce (optional)

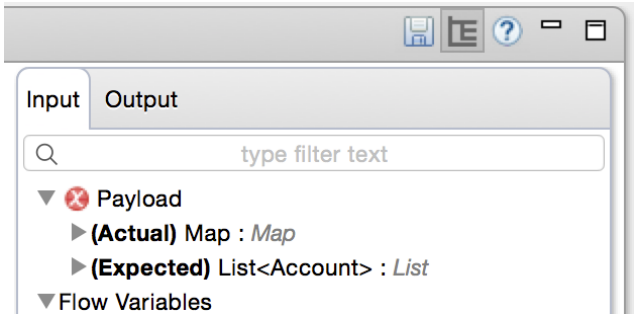
30. Add a Salesforce connector to the second batch step in the processing phase.



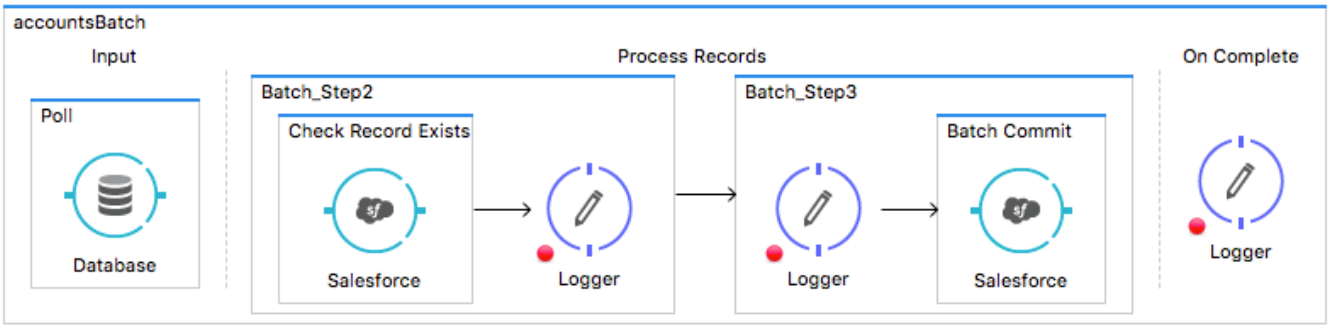
31. Configure the Salesforce connector endpoint to use the existing Salesforce connector configuration.
32. Set the operation to Create.
33. Set the sObject Type to Account (Account).
34. Leave the sObject Field Mappings to From Message #[payload].



35. 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.



36. Drag out a Batch Commit scope and drop it in the second batch step.
37. Add the Salesforce endpoint to it.



38. In the Batch Commit Properties view, set the commit size to 100.

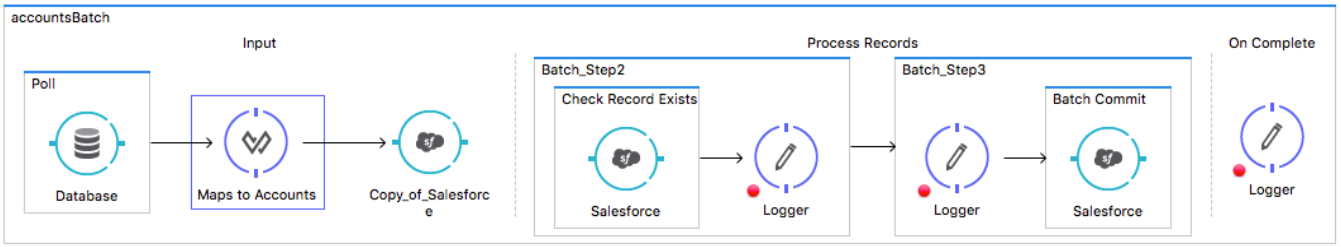
Transform the record data

39. Copy the Salesforce endpoint in the second batch step and paste it after the poll in the input phase.

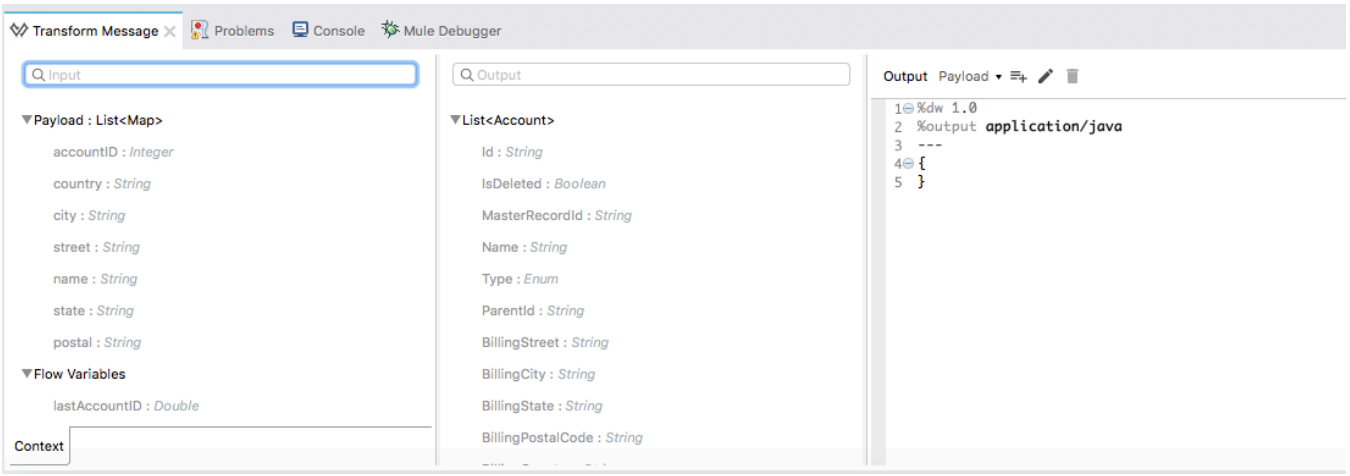
    Note: You are temporarily adding this Salesforce endpoint after the Database endpoint so DataSense will work to create the mappings. You could also add the DataWeave transformation to the process records phase, but this would add additional overhead.

40. Add a Transform Message component between the Poll scope and the Salesforce endpoint.

41. Change the name of Transform Message component to Maps to Accounts.



42. 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.



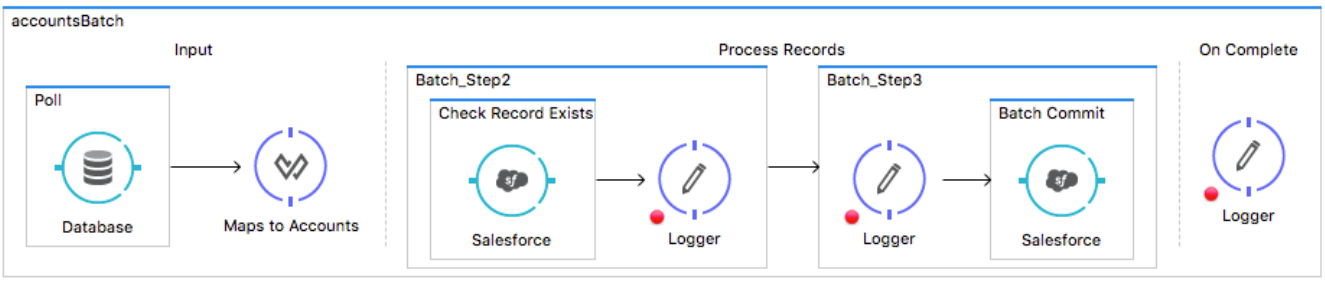
43. Double click on the following fields from the List<Account> properties listed in the Output to form a DataWeave expression: Name, BillingStreet, BillingCity, BillingState, BillingPostalCode, and BillingCountry fields.

44. Change the expression to use the use the map operator to map the payload.

45. Replace the placeholder values with references to the corresponding property in the input.

```
Output  Payload
%dw 1.0
%output application/java
---
payload map {
    Name: $.name,
    BillingStreet: $.street,
    BillingCity: $.city,
    BillingState: $.state,
    BillingPostalCode: $.postal,
    BillingCountry: $.country
}
```

46. Delete the Salesforce endpoint in the input phase.



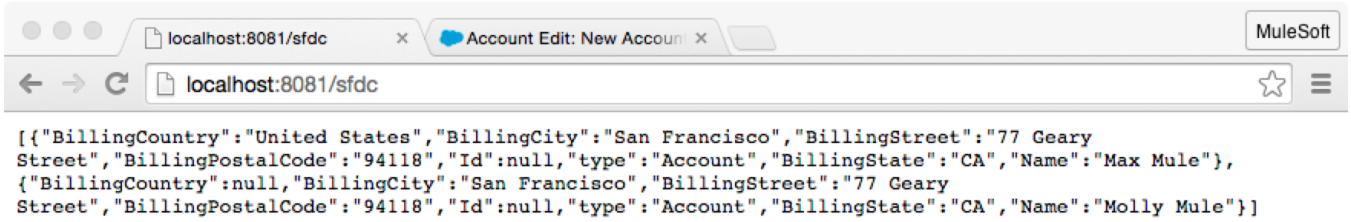
47. Save the file.

Test the application

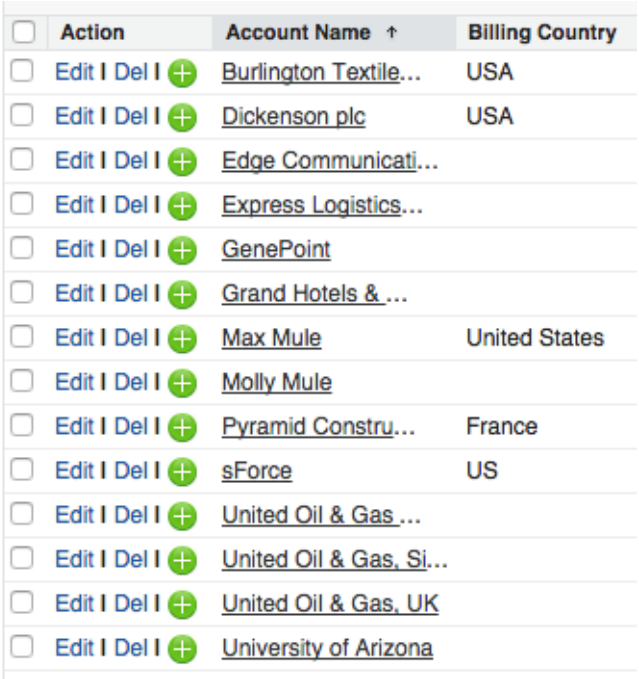
48. Locate the Salesforce endpoint in getSFDCAccountsFlow.
49. Modify the query so it selects accounts with the postal code you have been using this module.

```
Query Text:
SELECT BillingCity,BillingCountry,BillingPostalCode,BillingState,BillingStreet,Name
FROM Account
WHERE BillingPostalCode = '94108'
```

50. Run the application and clear the application data.
51. Check the console and make sure you see your new records processed.
52. Make a request to http://localhost:8081/sfdc; you should see your new records from the legacy MySQL database now in the Salesforce database.

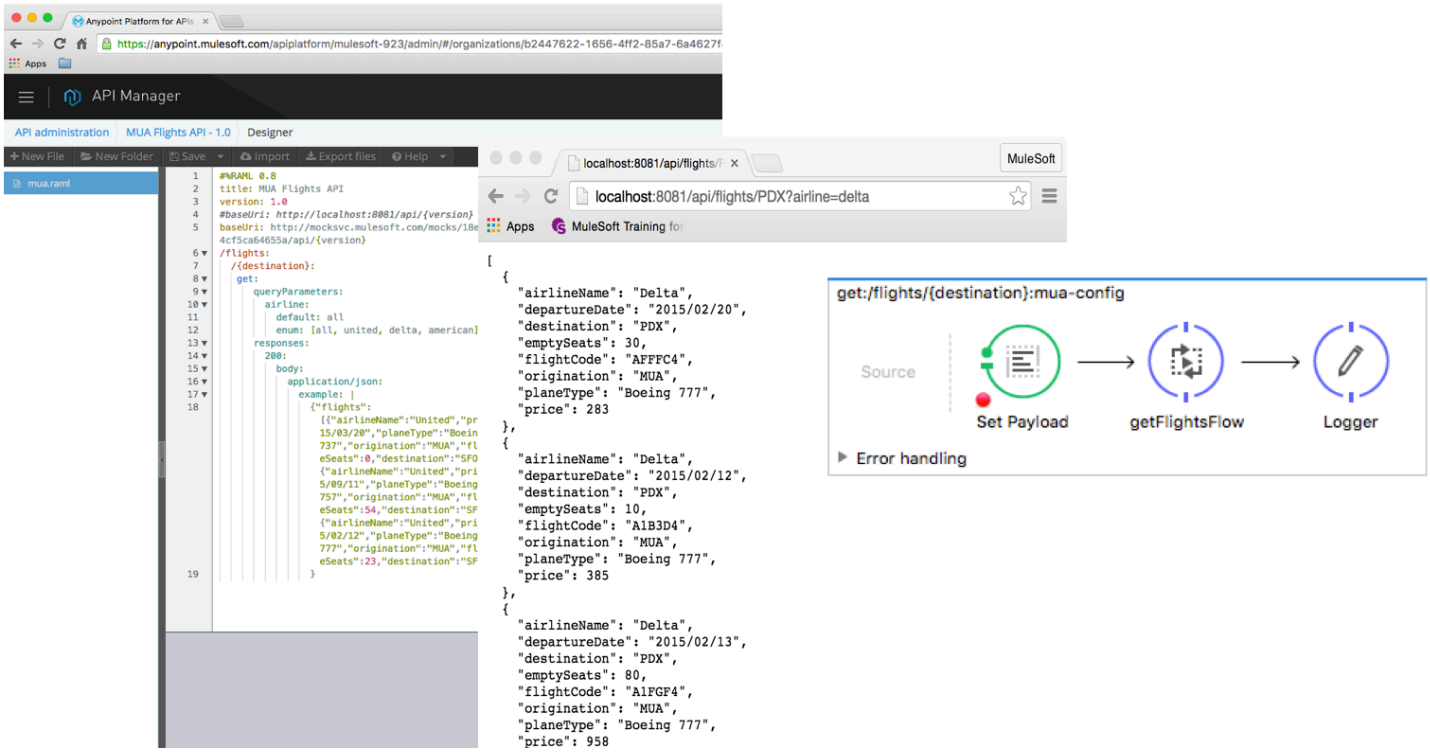


53. Run the application again; no records should be processed.
54. Return to salesforce.com and locate your new record(s); they should have been inserted only once.



Action	Account Name ↑	Billing Country
Edit | Del |	Burlington Textile...	USA
Edit | Del |	Dickenson plc	USA
Edit | Del |	Edge Communicati...	
Edit | Del |	Express Logistics...	
Edit | Del |	GenePoint	
Edit | Del |	Grand Hotels & ...	
Edit | Del |	Max Mule	United States
Edit | Del |	Molly Mule	
Edit | Del |	Pyramid Constru...	France
Edit | Del |	sForce	US
Edit | Del |	United Oil & Gas ...	
Edit | Del |	United Oil & Gas, Si...	
Edit | Del |	United Oil & Gas, UK	
Edit | Del |	University of Arizona	

Module 10: Building RESTful Interfaces with RAML and APIkit



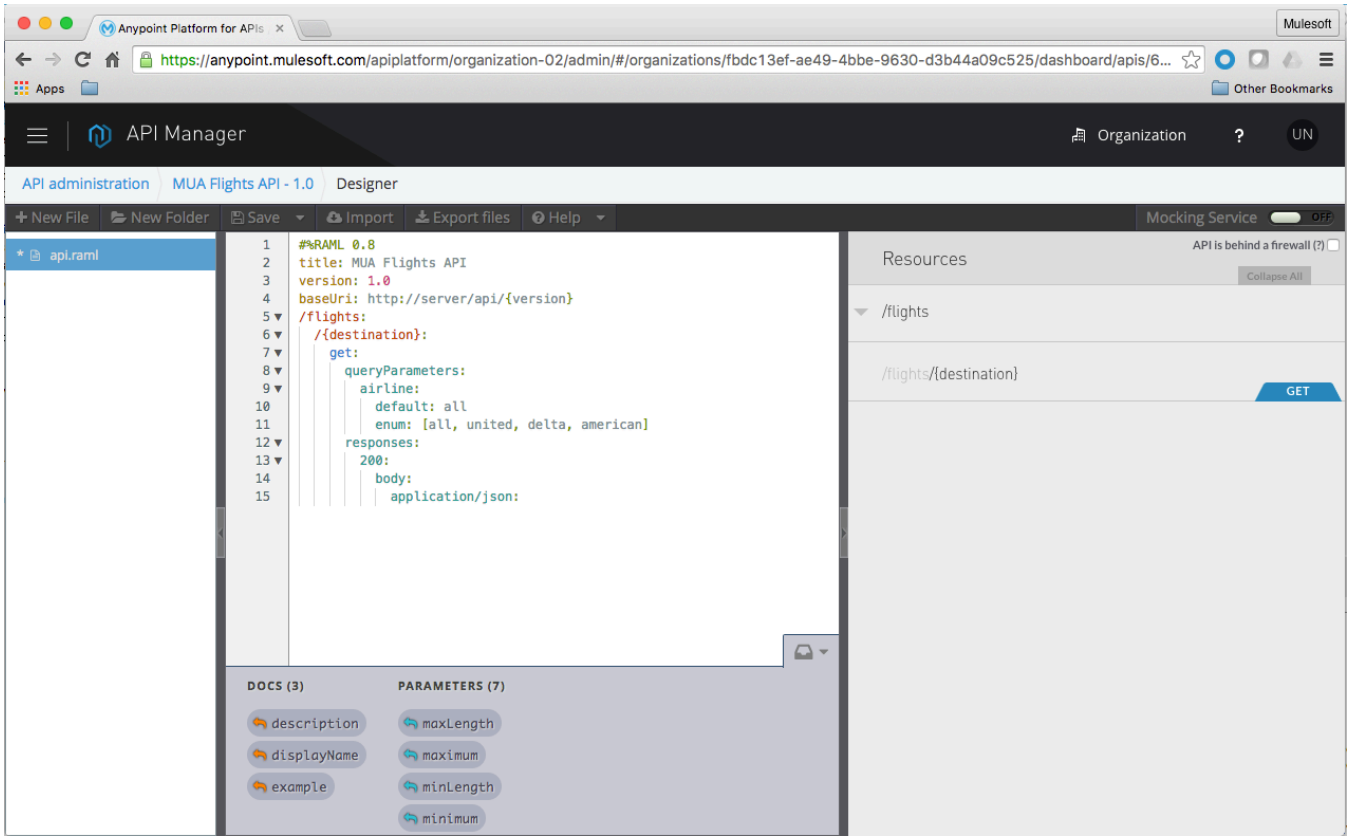
In this module, you will learn:

- 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

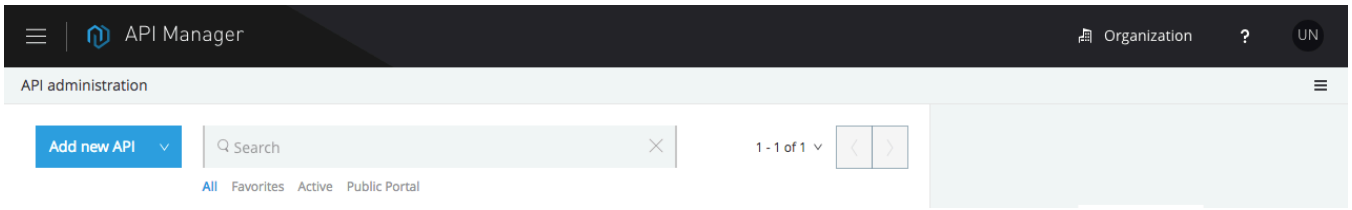
In this walkthrough, you will create an API definition for MUA with RAML. You will:

- 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.



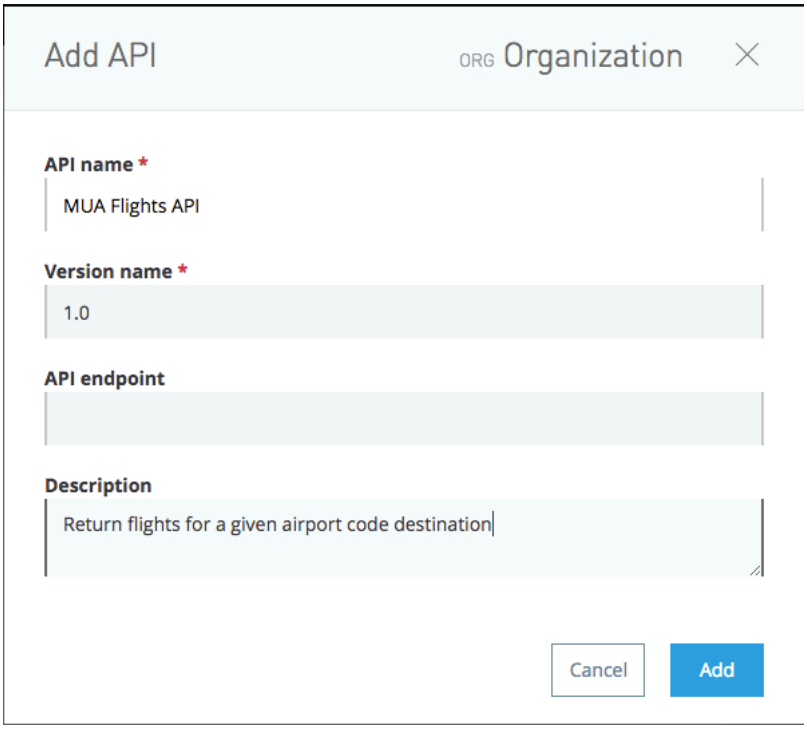
Add a new API

1. In a browser, go to http://anypoint.mulesoft.com and log in.
2. Click the upper left menu icon and select API Manager.
3. Click the Add new API button.

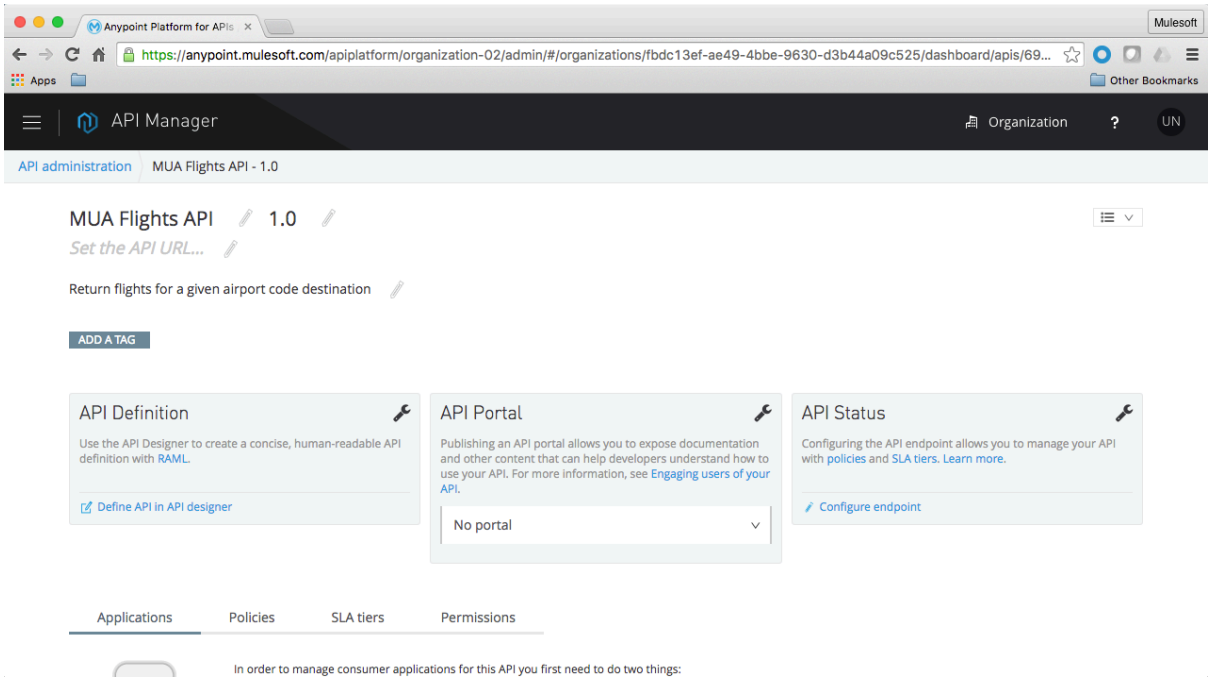


4. In the Add API dialog box, enter the following and then click Add.

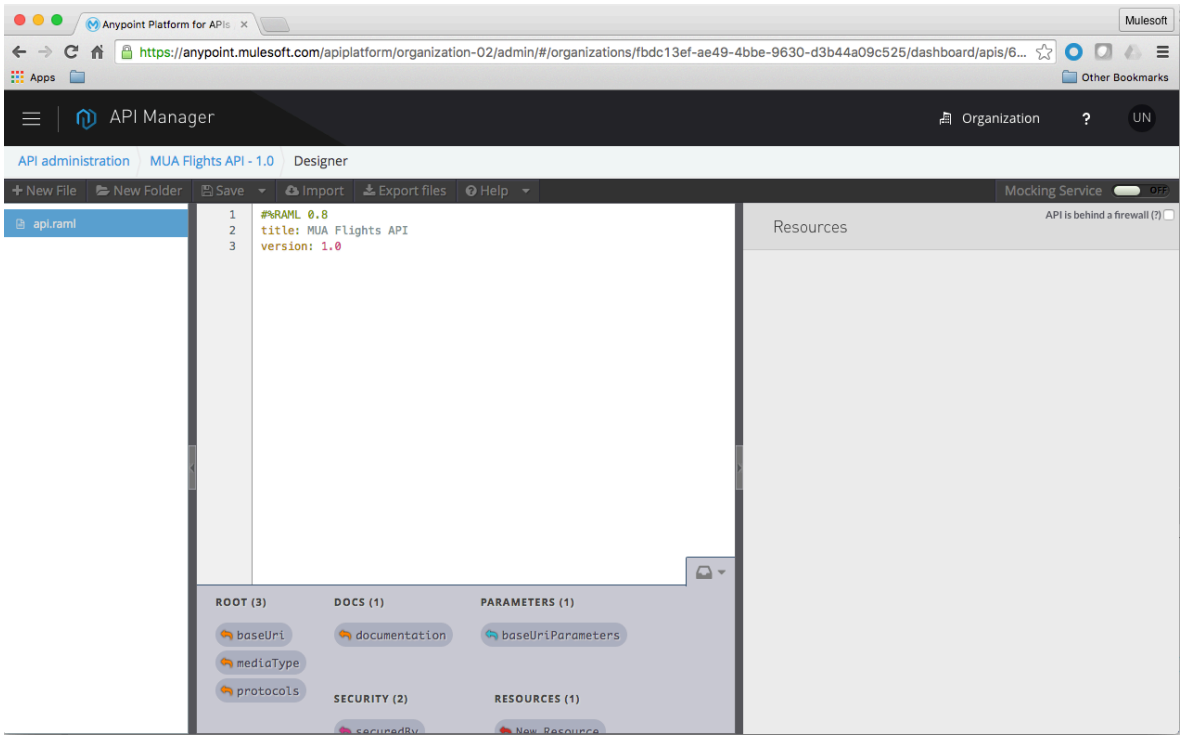
   - API name: MUA Flights API
   - Version name: 1.0
   - Description: Returns flights for a given airport code destination



5. Take a look at the different sections and links for the API in the API administration.



6. Click the Define API in API designer link in the API Definition section; the API Designer should open.



Note: To change the background color from black to white, press Ctrl+Shift+T.

Add RAML root metatdata

7. Place the cursor on a new line of code at the end of the file.
8. 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

9. Go to a new line of code and add a resource called flights.

   ```
   /flights:
   ```

Add a nested RAML resource

10. Indent by pressing the Enter key and then the Tab key.

11. Add a nested resource to return flights for a particular destination.

```
/flights:
  /{destination}:
```

Add a RAML method

12. 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.
13. Indent by pressing the Enter key and then the Tab key.
14. Scroll down in the API Designer shelf and locate and click responses.

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.

15. Indent and type 200:.

```
responses:
      200:
```

16. Indent and type b and then press Enter to add a body parameter (or add it from the shelf).

17. Indent and type a and then press Enter to add application/json (or add it from the shelf).

```
#%RAML 0.8
title: MUA Flights API
version: 1.0
baseUri: http://server/api/{version}

/flights:
  /{destination}:
    get:
      responses:
        200:
          body:
            application/json:
```

Note: This is the minimal RAML file needed for Anypoint Studio and APIkit to generate the RESTful interface.

Add a query parameter

18. Go to a new line of code after get:.
19. Press the q key and then the Enter key to add queryParameters.
20. Indent and add a query parameter called airline.
21. Indent and add a default property and set it to all.
22. Return and add an enum property and set it to an array with values all, united, delta, and american.

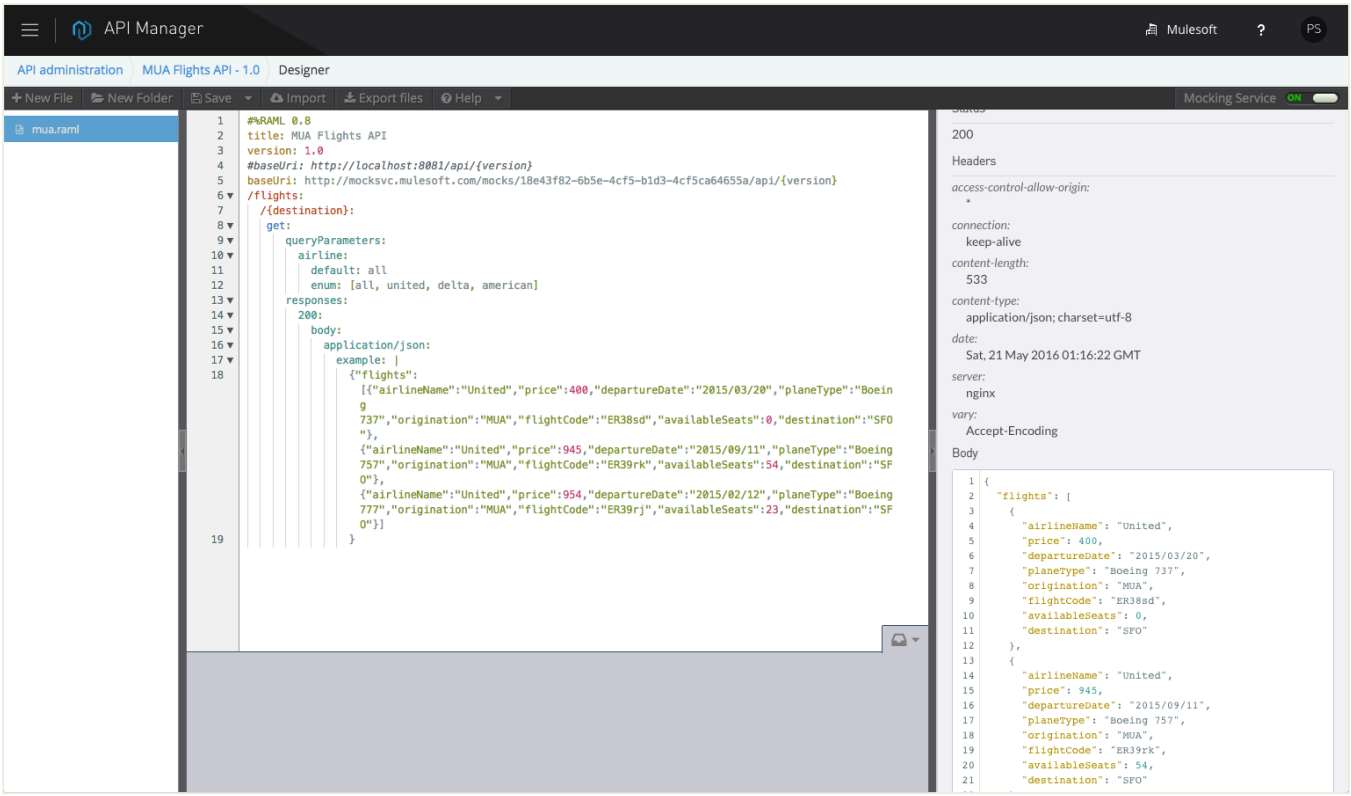
```
/flights:
  /{destination}:
    get:
      queryParameters:
        airline:
          default: all
          enum:  [all, united, delta, american]
      responses:
        200:
```

23. 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.

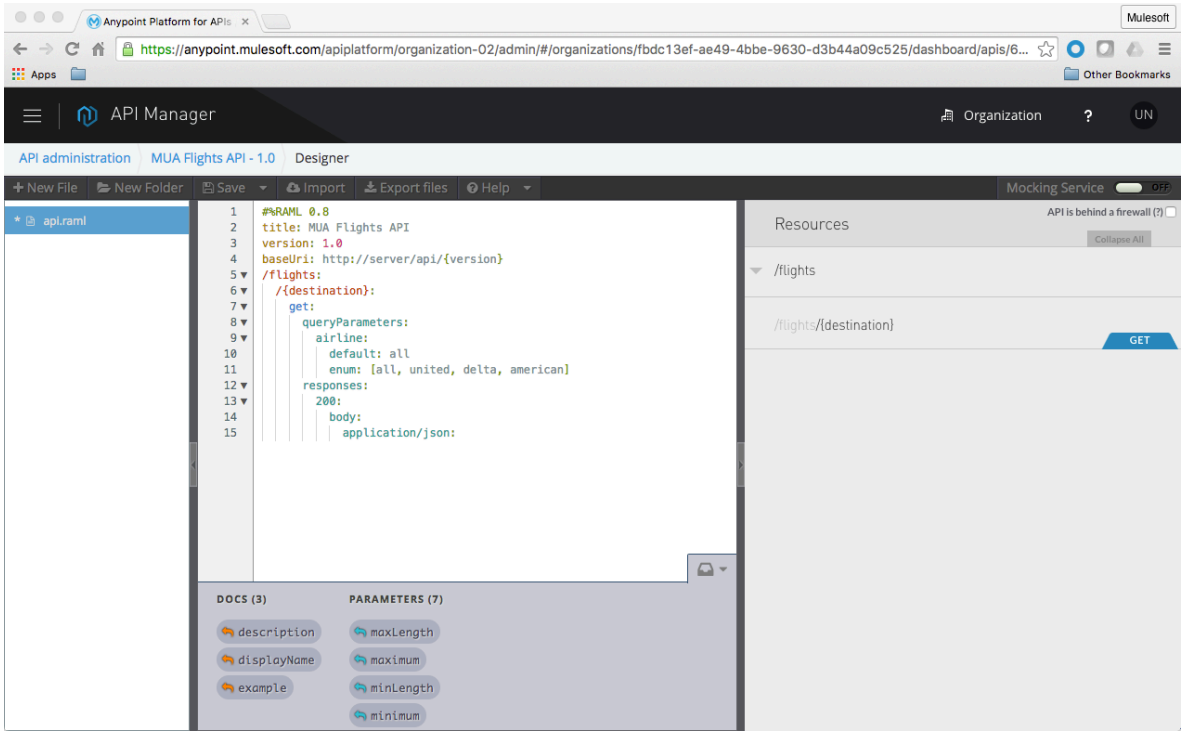


Use the API Designer console

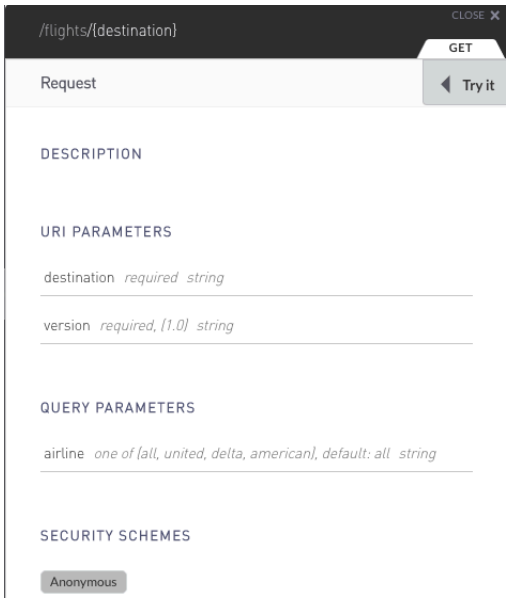
1. Return to your MUA Flights API in API Designer.

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.

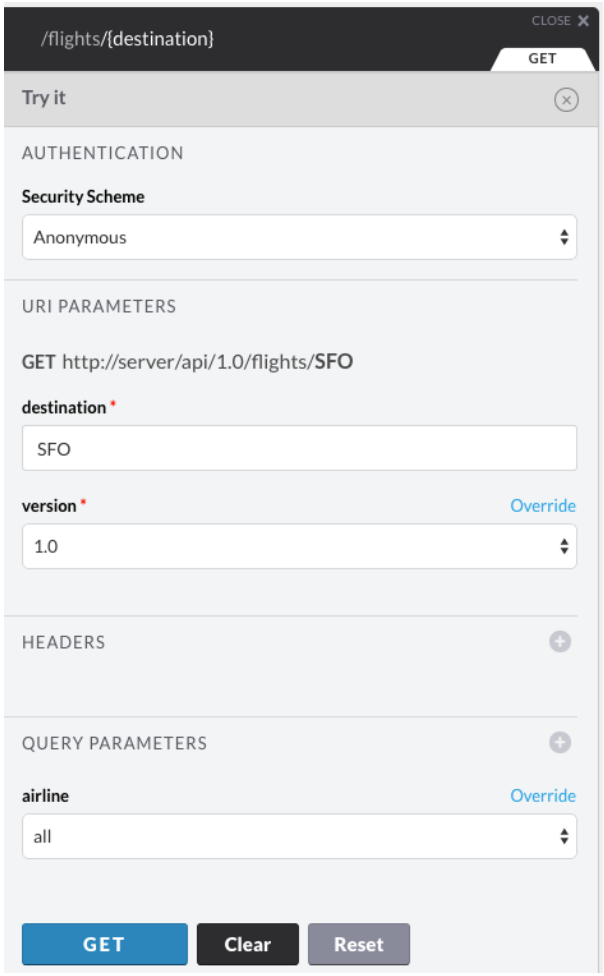


3. Click the GET button for the /flights/{destination} resource; you should see the request information.

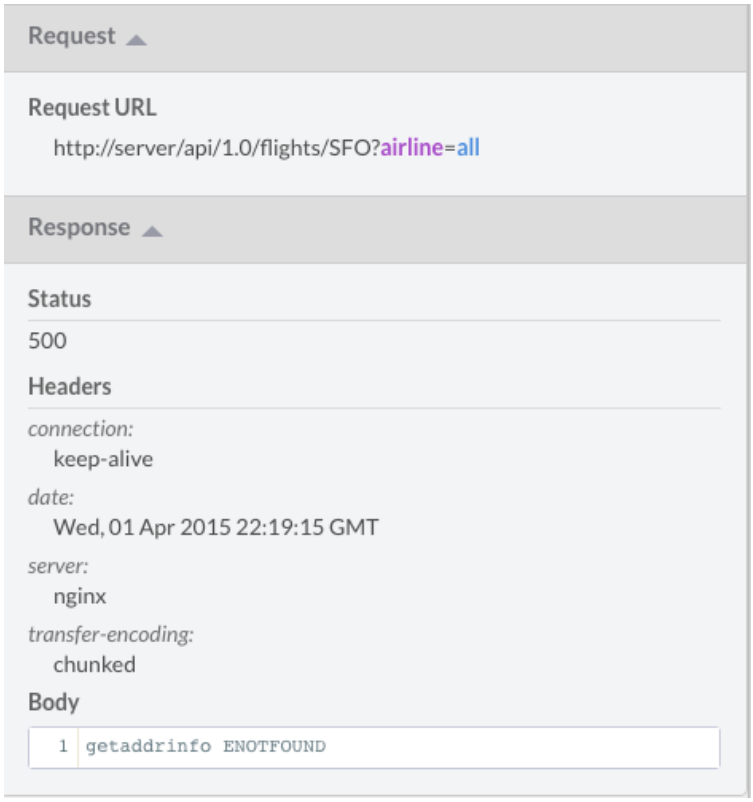


4. Scroll down to the Responses section and see the specification for a 200 status code.
5. Scroll back up and click the Try it button.

6. Enter a destination of SFO and click the GET button.

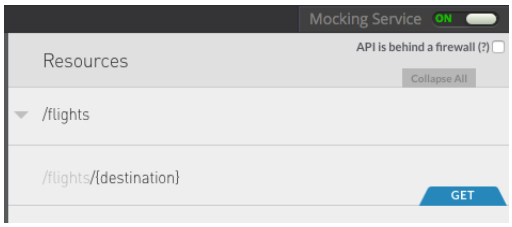


7. Scroll down and look at the response; you should get a 500 status code because the baseUri is not found.



Use the mocking service

8. Locate the Mocking Service slider in the menu bar above the console.
9. Slide it to on.

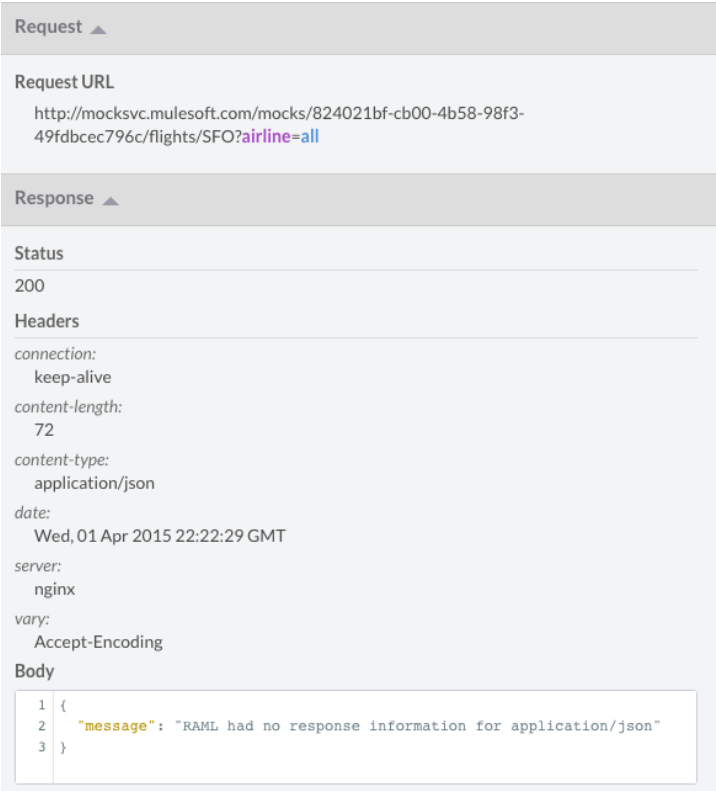


10. Look at the new baseUri in the editor.

```
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
```

11. 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.

12. Scroll down and look at the body; you should get a general RAML message placeholder.



13. Close the /flights/{destination} window.

Add examples

14. Return to the course snippets.txt file and copy the Example flights 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"}]
```

15. Return to the API Designer editor and indent from the application/json element and add an example element.
16. Add a space and then a | after the example element.

17. Indent and paste the JSON array you just copied.

```
body:
  application/json:
    example: |
      [{"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

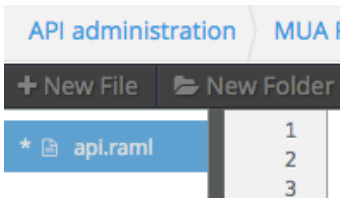
18. Return to the API console.
19. Click the GET button for the /flights/{destination} resource again.
20. Click Try it, enter a destination, and click GET.
21. Look at the response body; you should now get your example data.

Body

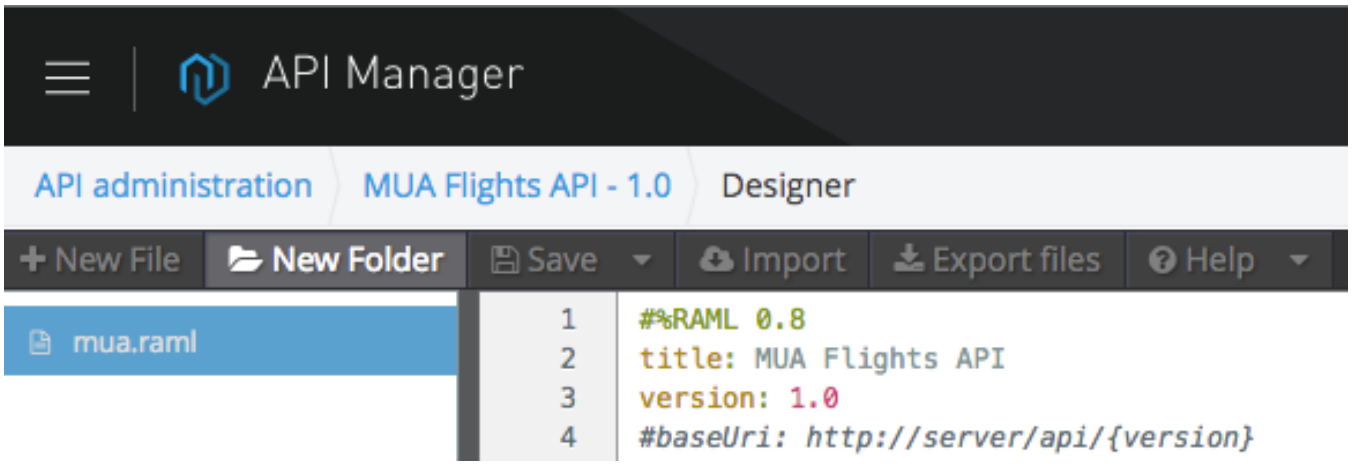

```
[
  {
    "airlineName": "United",
    "price": 400,
    "departureDate": "2015/03/20",
    "planeType": "Boeing 737",
    "origination": "MUA",
    "flightCode": "ER38sd",
    "availableSeats": 0,
    "destination": "SFO"
  },
  {
    "airlineName": "United",
    "price": 945,
    "departureDate": "2015/09/11",
    "planeType": "Boeing 757",
    "origination": "MUA",
    "flightCode": "ER39rk",
```


Save and export the RAML file

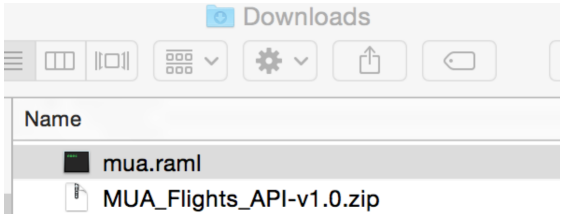
22. Slide the Mocking Service slider to off.
23. Save the file.
24. Right-click api.raml in the left pane and select Rename.



25. Change the name to mua.raml, by clicking on the gear icon by api.raml and click OK.
26. Click on Save.
27. Click the Export files button to download the RAML file.



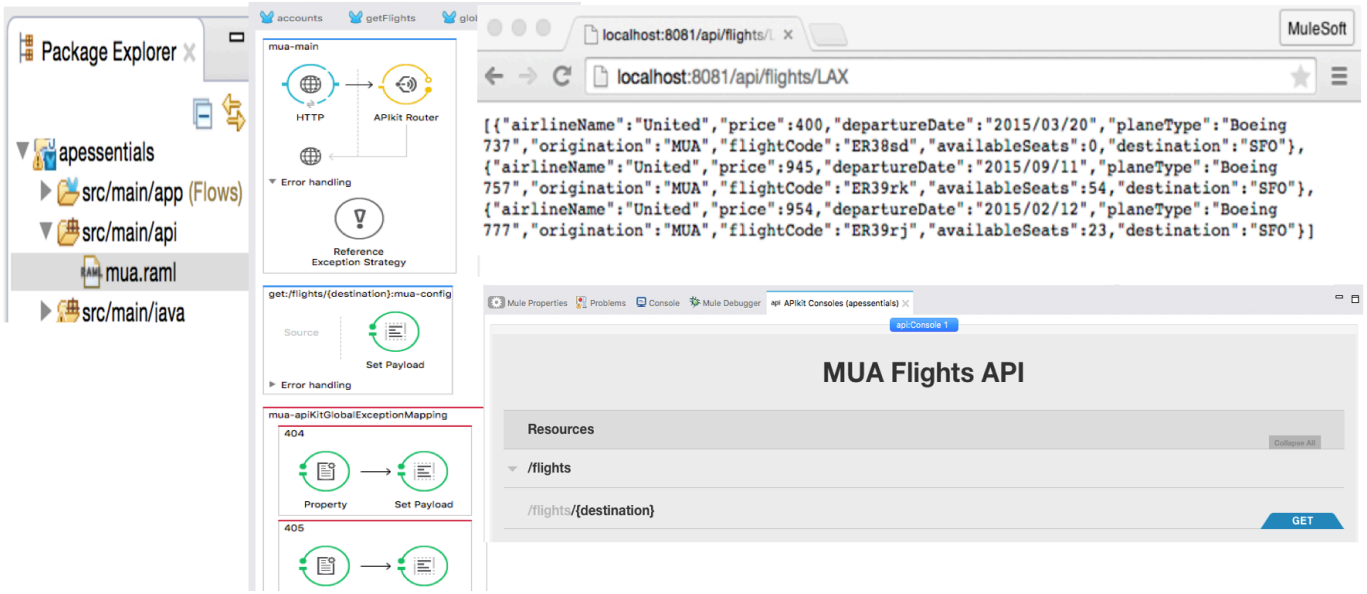
28. Locate the MUA_Flights_API-v1.0.zip file that is downloaded.
29. Expand the ZIP and locate the mua.raml file it contains.



Walkthrough 10-3: Use Anypoint Studio to create a RESTful API interface from a RAML file

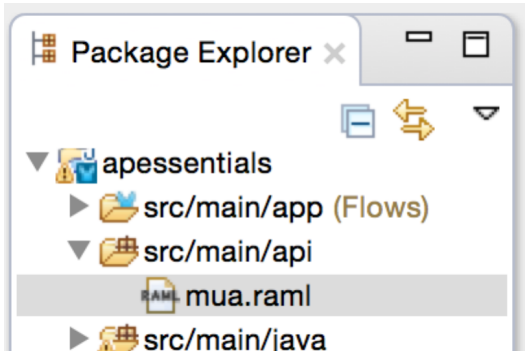
In this walkthrough, you will generate a RESTful interface from the RAML file. You will:

- 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.



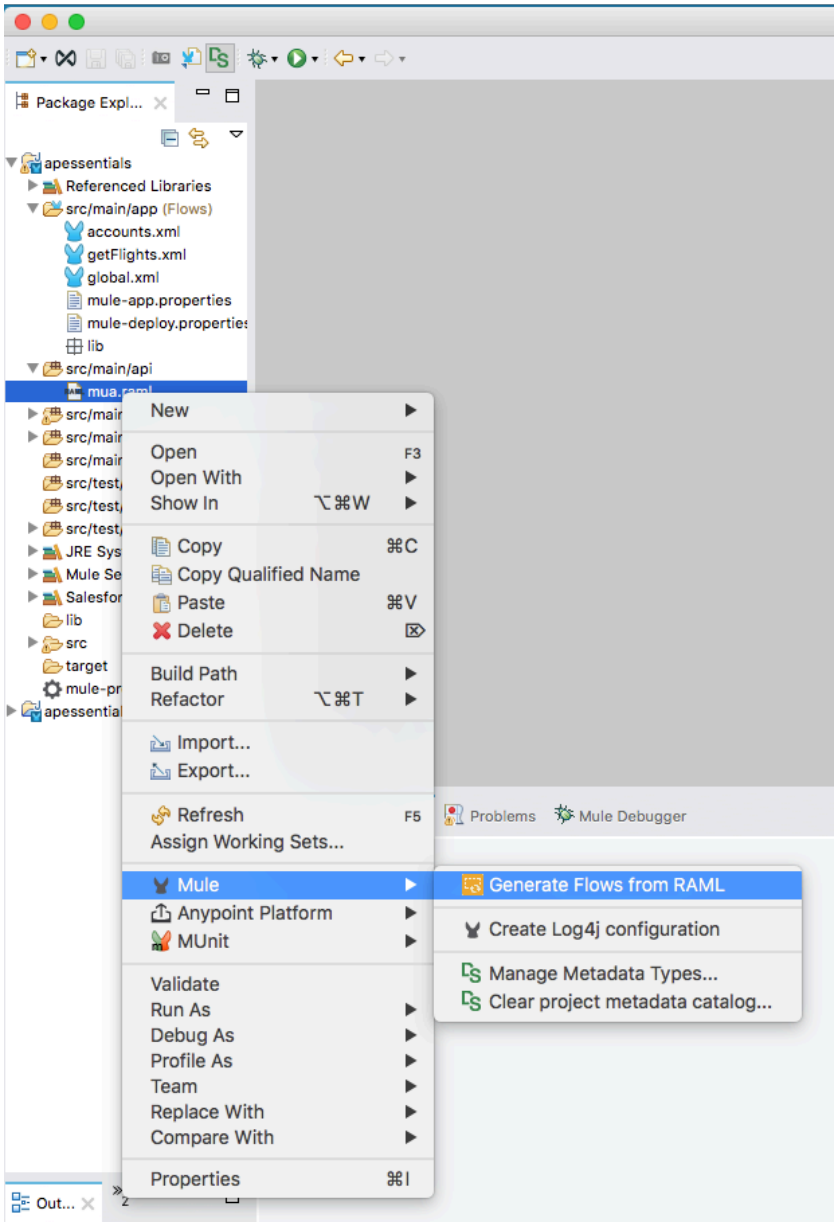
Add the RAML file

1. Drag the mua.raml file into the src/main/api folder in your apessentials project.



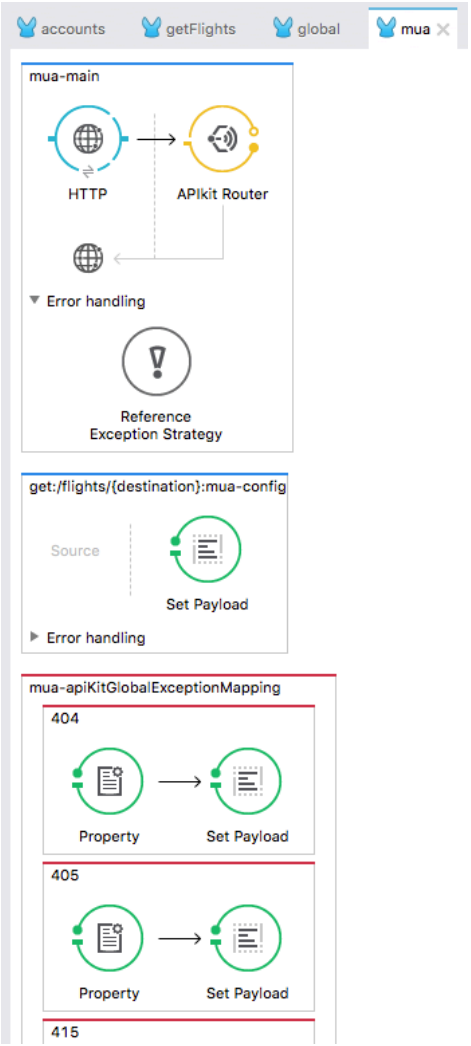
Create a RESTful interface

2. Right-click the RAML file and select Mule > Generate Flows from RAML.



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.



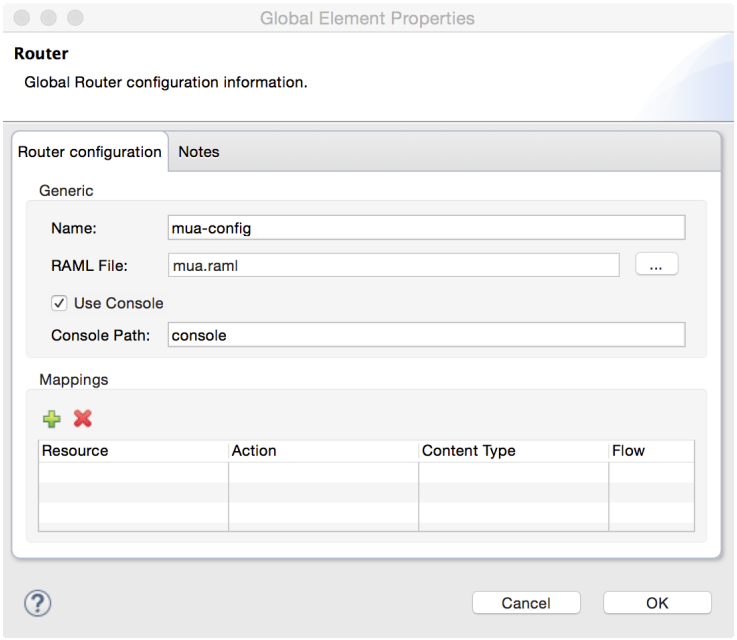
Modify the HTTP endpoint

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

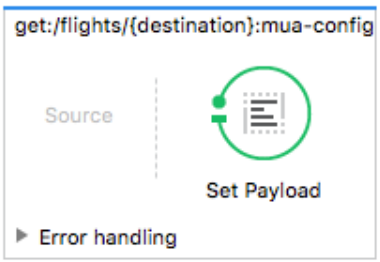
12. Double-click the APIkit Router.
13. In the APIkit Router Properties view, see the router configuration is set to mua-config.
14. Click the Edit button next to Connector Configuration.

15. In the Global Element Properties dialog box, look at the values and then click OK.

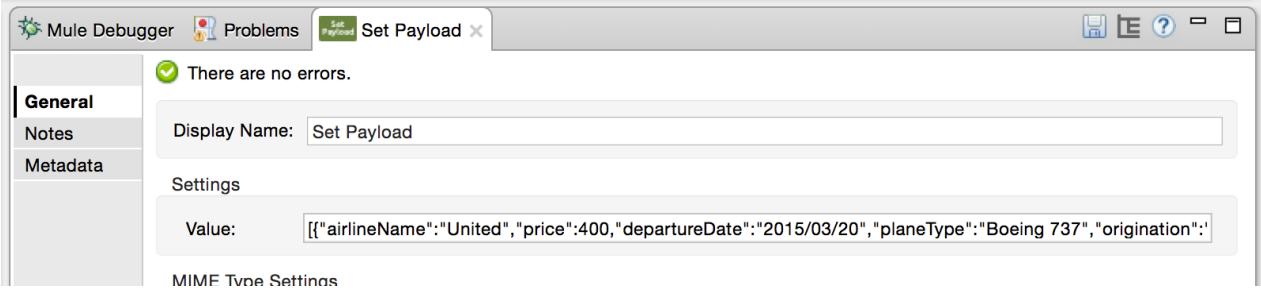


Look at the generated resource flow

16. Look at the name of the other flow created: get:/flights/{destination}:mua-config.



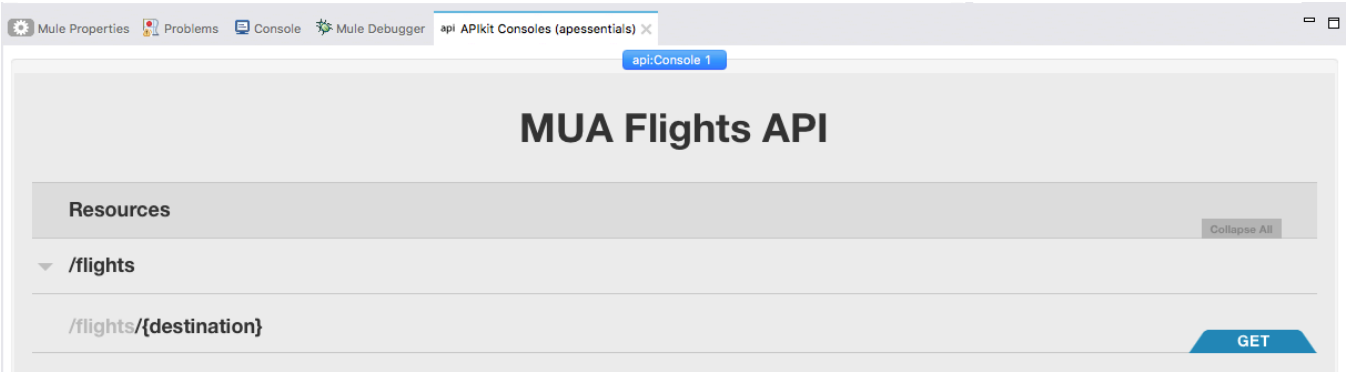
17. Double-click the Set Payload transformer and look at the value.



18. Switch to the Configuration XML view and look at the generated code.

Test the web service in the APIkit Consoles view

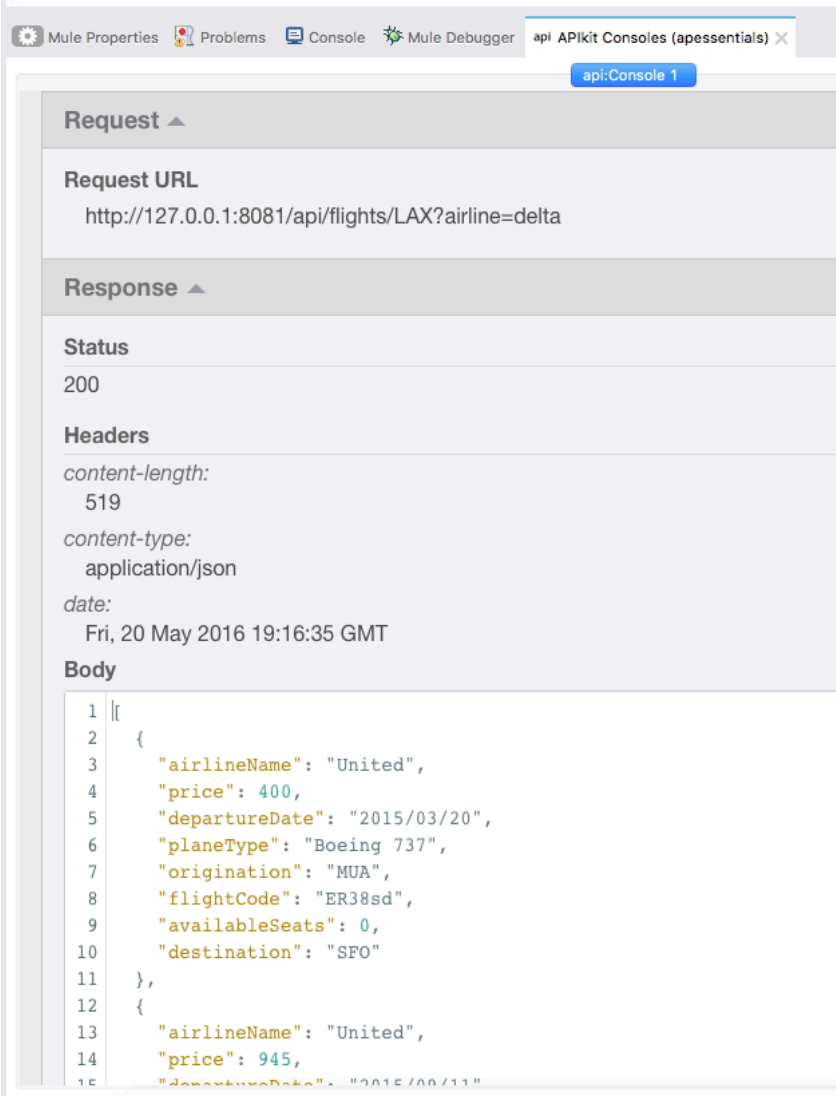
19. Save the file and run the application.
20. Look at the APIkit Consoles view that opens.



Note: If the API is not displayed in the APIkit Console, change your HTTP Listener Configuration host from 0.0.0.0 to localhost.

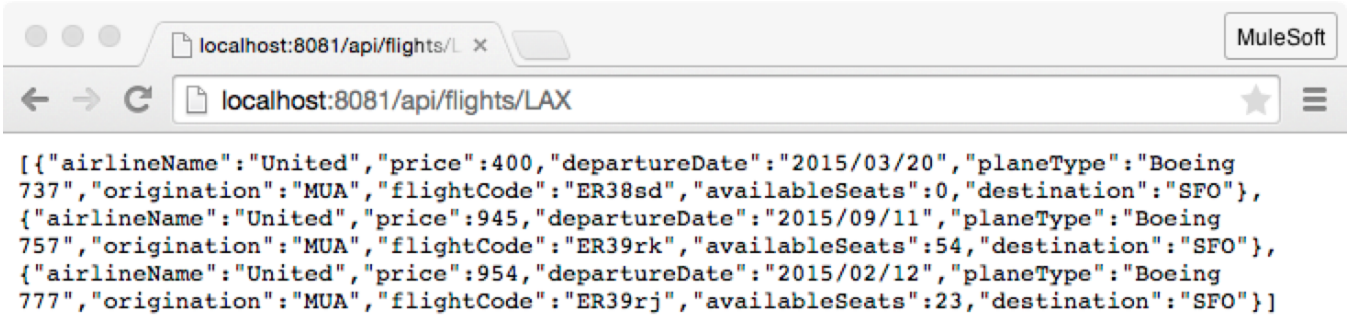
21. Click the GET button for /flights/{destination}.
22. Click the Try It button.
23. Enter a destination of LAX (or any other value).
24. Click in the airline query parameter field and select one of the enumerated values, like delta.

25. Click the GET button; you should see the example data displayed.



Test the web service in a browser

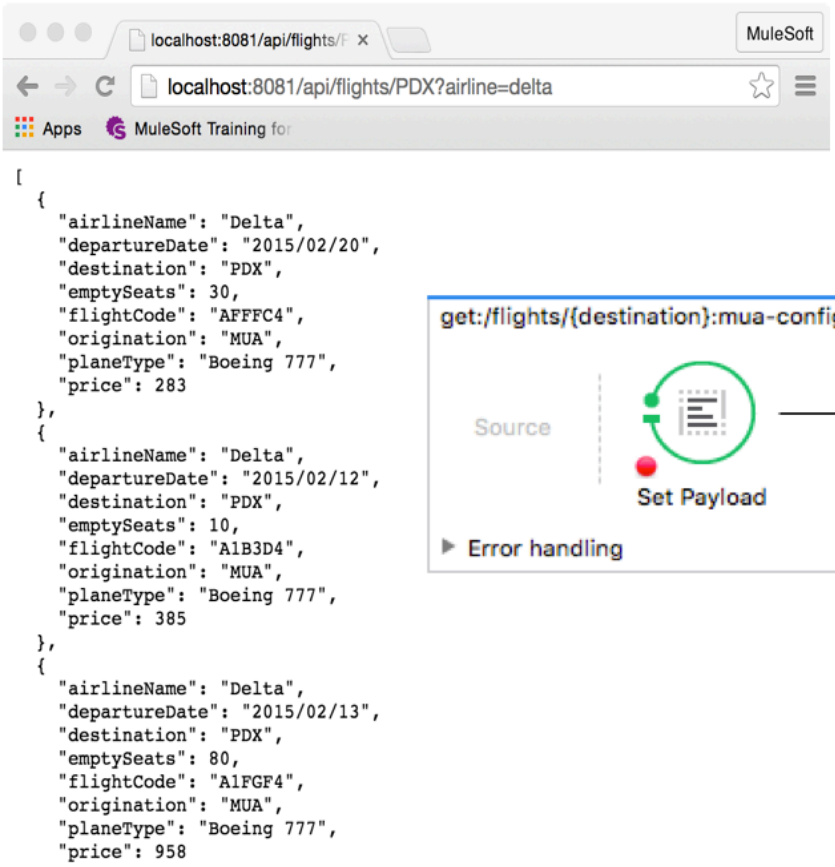
26. Go to a browser, and make a request to http://localhost:8081/api/flights/LAX (or any other destination); you should see the example data returned.

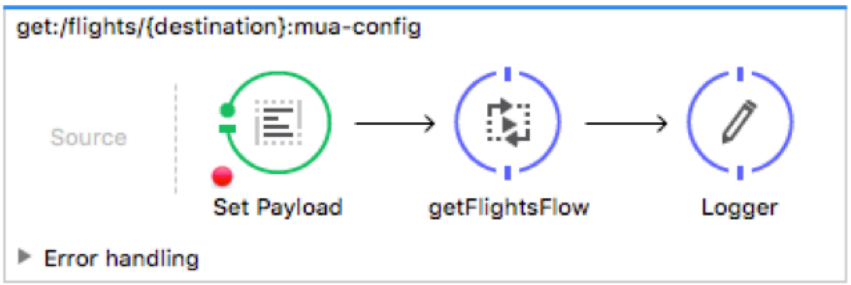


Walkthrough 10-4: Use Anypoint Studio to implement a RESTful web service

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.

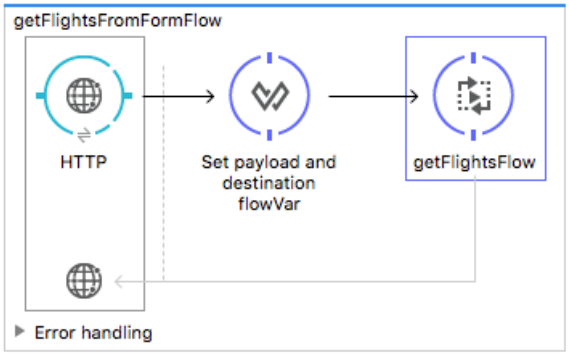




Modify the existing flows

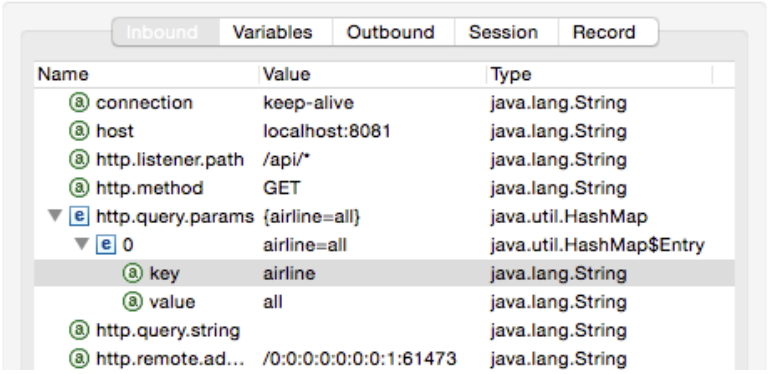
1. Return to getFlights.xml.
2. Drag out a Flow scope element from the palette and drop it at the top of the canvas.
3. Give it a display name of getFlightsFromFormFlow.
4. Drag the HTTP endpoint from getFlightsFlow and drop it in the source section of getFlightsFromFormFlow.

5. Drag the Set payload and destination flowVar component from getFlightsFlow and drop it in the process section of getFlightsFromFormFlow.
6. Drag out a Flow Reference component from the palette and drop it in the process section of getFlightsFromFormFlow.
7. In the Flow Reference Properties view, set the flow name to getFlightsFlow.

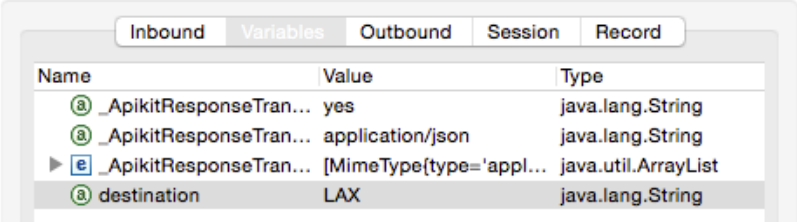


Determine how to reference the web service destination and airline values

8. Return to mua.xml.
9. Add a Logger to the get:/flights/{destination} flow.
10. Add a breakpoint to the Set Payload transformer in the get:/flights/{destination} flow.
11. Save all the files and debug the application.
12. Make a request to http://localhost:8081/api/flights/LAX.
13. In the Mule Debugger view, look at the inbound properties; you should see a query parameter called airline with a value of all.



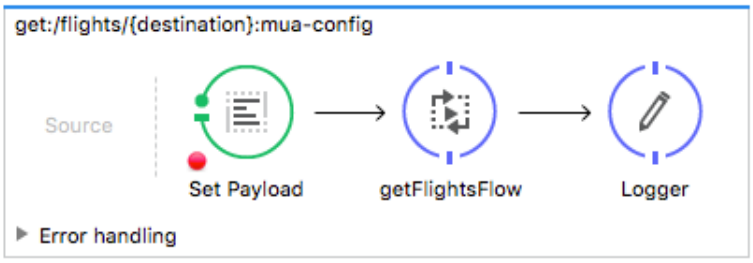
14. Click the Variables tab; you should see a variable called destination with a value of LAX.



15. Click the Resume button.

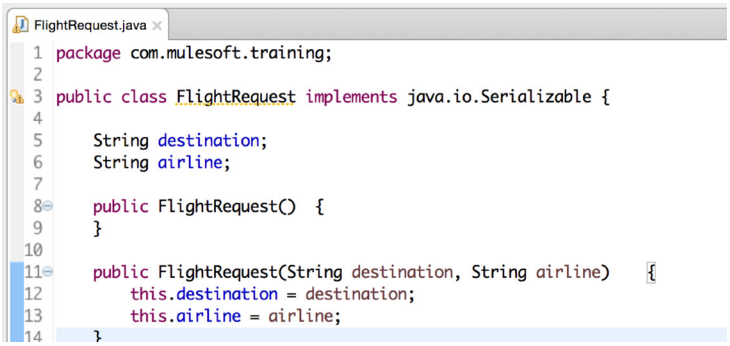
Call the backend flow

16. Return to the get:/flights/{destination} flow.
17. Add a Flow Reference component after the Set Payload transformer.
18. In the Flow Reference Properties view, set the flow name to getFlightsFlow.



Review the FlightRequest Java class

19. Open the com.mulesoft.training.FlightRequest.java class located in the project's src/main/java folder and examine the code.



Set the payload to a Java object with flight request data

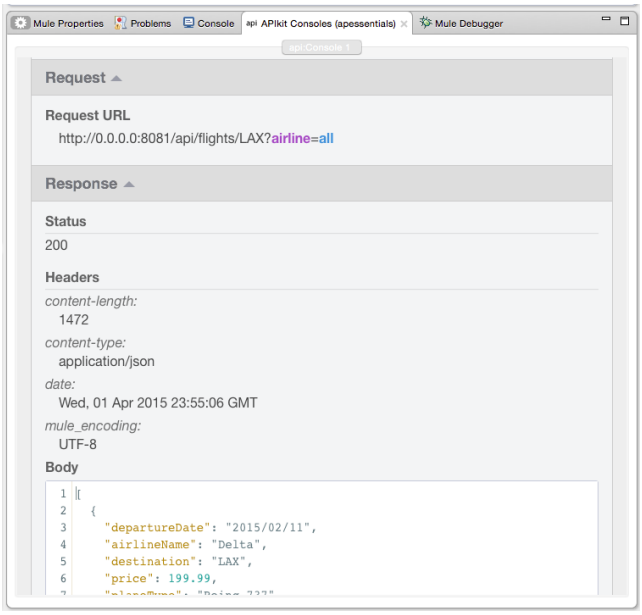
20. Double-click the Set Payload transformer in the get:/flights/{destination} flow.

21. 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 form the course snippets.txt file.

Test the web service in the APIkit Consoles view

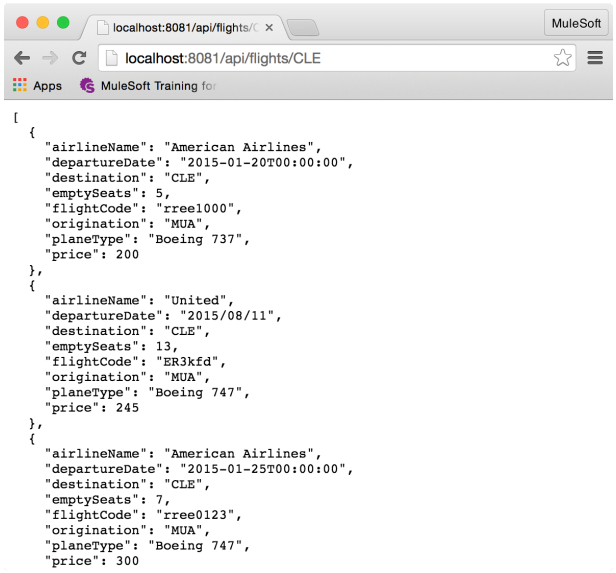
22. Save the files and run the application.
23. In the APIkit Consoles view, click the GET button for /flights/{destination}.
24. Click the Try It button.
25. Enter a destination of LAX (or any other value) and click the GET button; you should now see the real data (for LAX) displayed.



26. Scroll up in the /flights/{destination} resource window.
27. Set the airline query parameter to united.
28. Click the GET button; you see should now see only the flights for LAX on United.

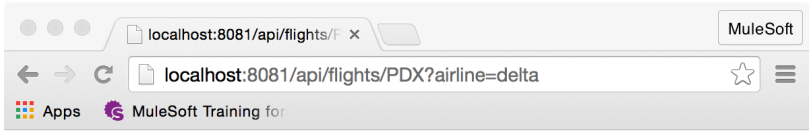
Test the web service in a browser

29. Make a request to http://localhost:8081/api/flights/CLE; you should see data for all the flights to CLE returned.



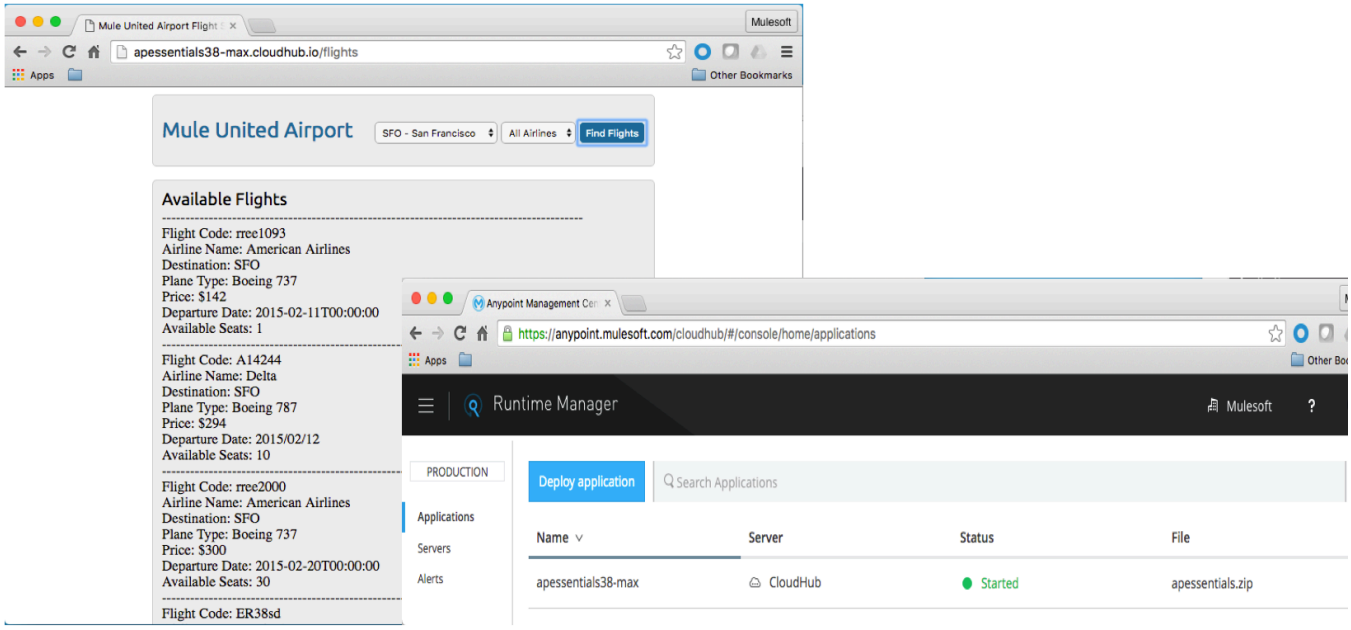

```
[
  {
    "airlineName": "American Airlines",
    "departureDate": "2015-01-20T00:00:00",
    "destination": "CLE",
    "emptySeats": 5,
    "flightCode": "rree1000",
    "origination": "MUA",
    "planeType": "Boeing 737",
    "price": 200
  },
  {
    "airlineName": "United",
    "departureDate": "2015/08/11",
    "destination": "CLE",
    "emptySeats": 13,
    "flightCode": "ER3kfd",
    "origination": "MUA",
    "planeType": "Boeing 747",
    "price": 245
  },
  {
    "airlineName": "American Airlines",
    "departureDate": "2015-01-25T00:00:00",
    "destination": "CLE",
    "emptySeats": 7,
    "flightCode": "rree0123",
    "origination": "MUA",
    "planeType": "Boeing 747",
    "price": 300
```


30. 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.




```
[
  {
    "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



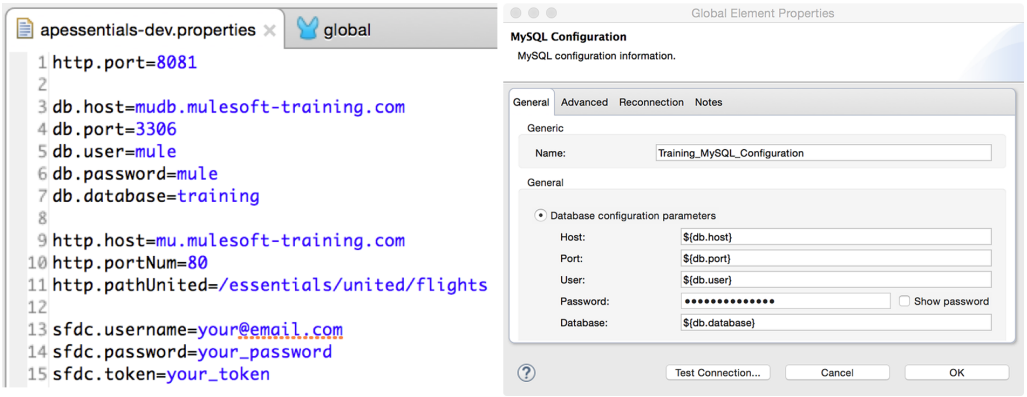
In this module, you will learn:

- 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

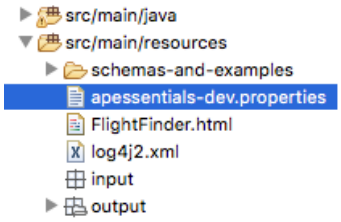
In this walkthrough, you will introduce properties into your Mule application. You will:

- 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

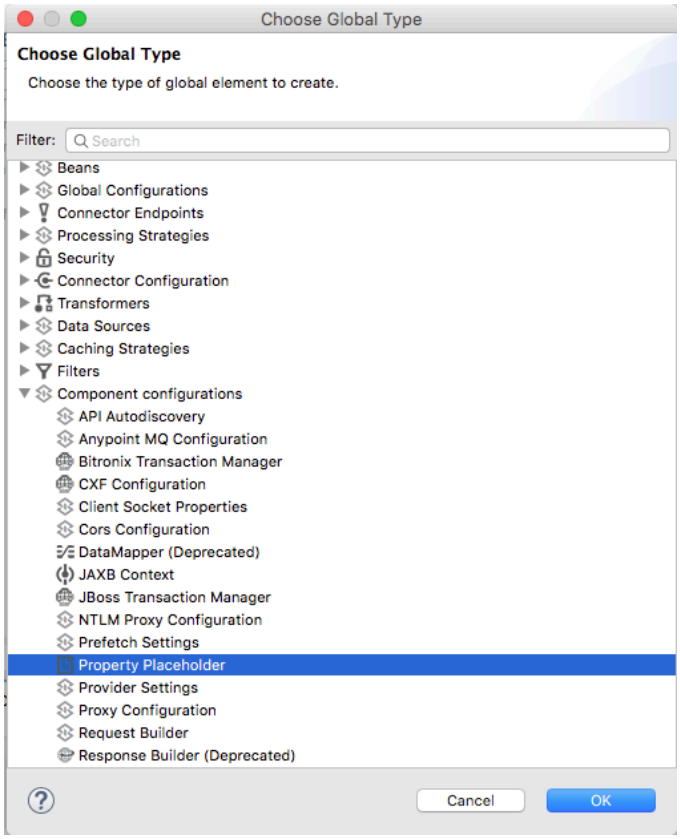
1. Right-click the src/main/resources folder in the Package Explorer and select New > File.
2. Name the file apessentials-dev.properties and click Finish.



Create a Properties Placeholder global element

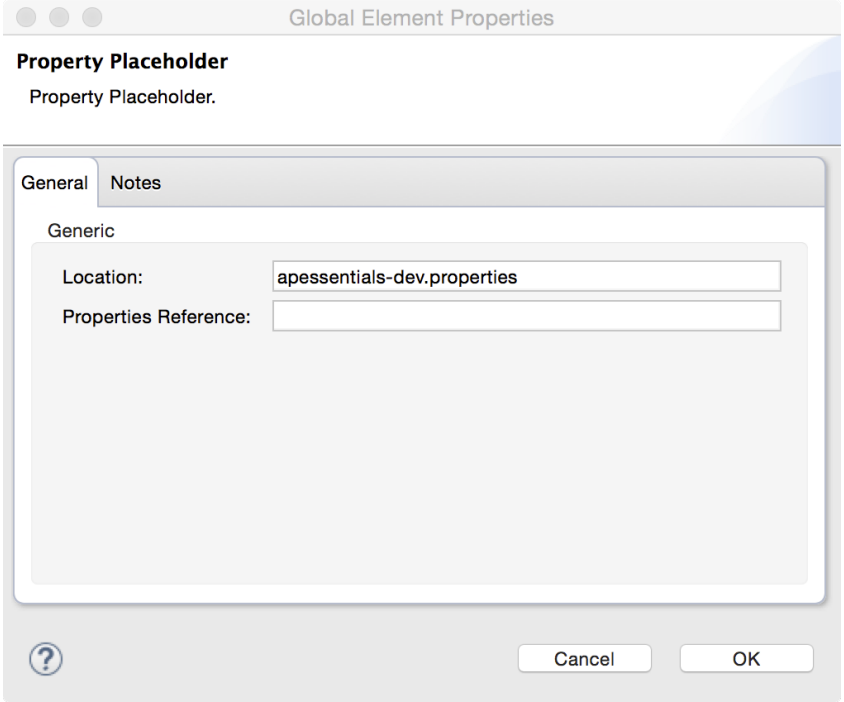
3. Open global.xml.
4. Navigate to the Global Elements view and click Create.

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.



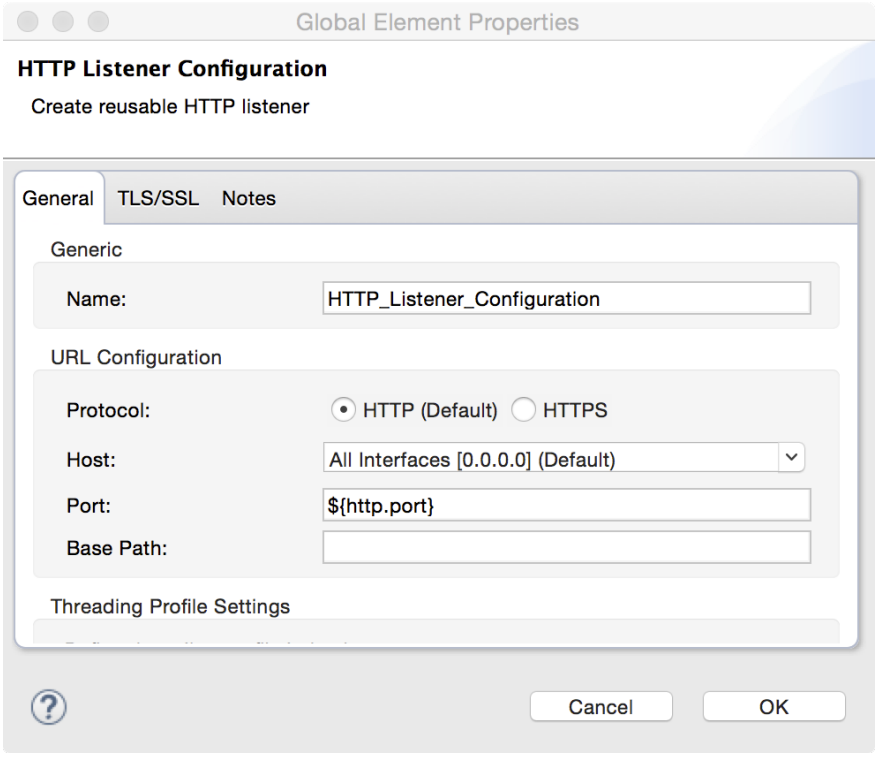
Parameterize the HTTP Listener port

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.

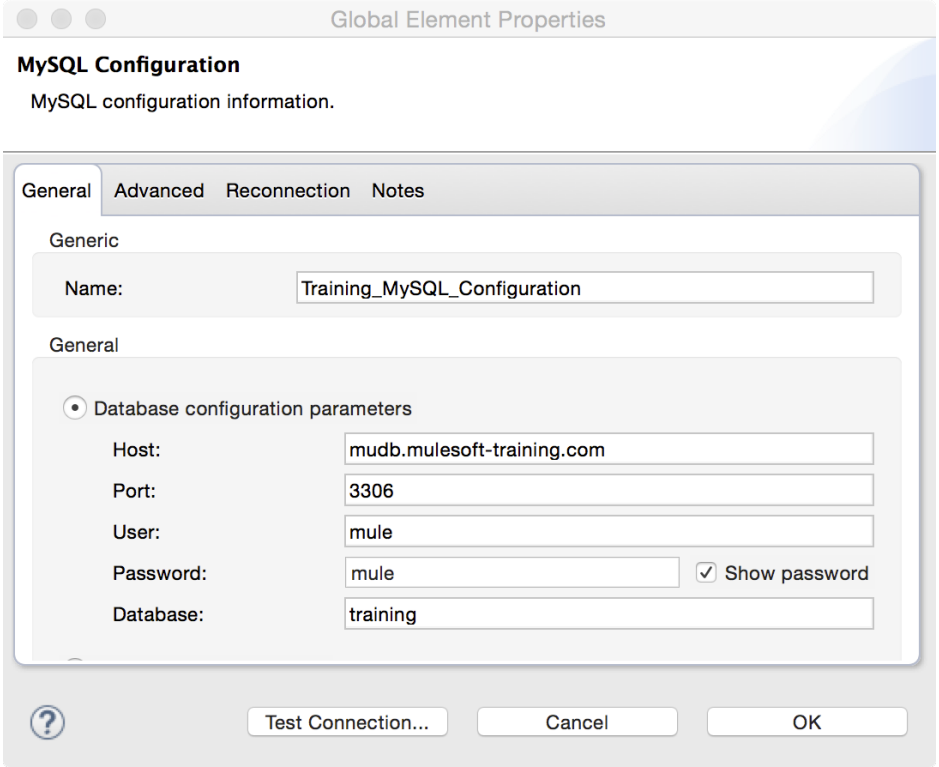


Test the application

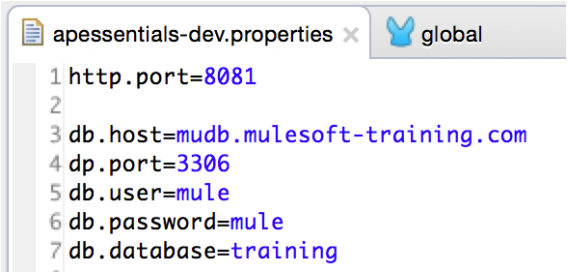
13. Save all the files and run the application.
14. Make a request to http://localhost:8081/api/flights/SFO in the APIkit Consoles view or a browser and confirm it still works.

Parameterize database credentials

15. Return to global.xml.
16. Double-click the MySQL Configuration global element to edit it.

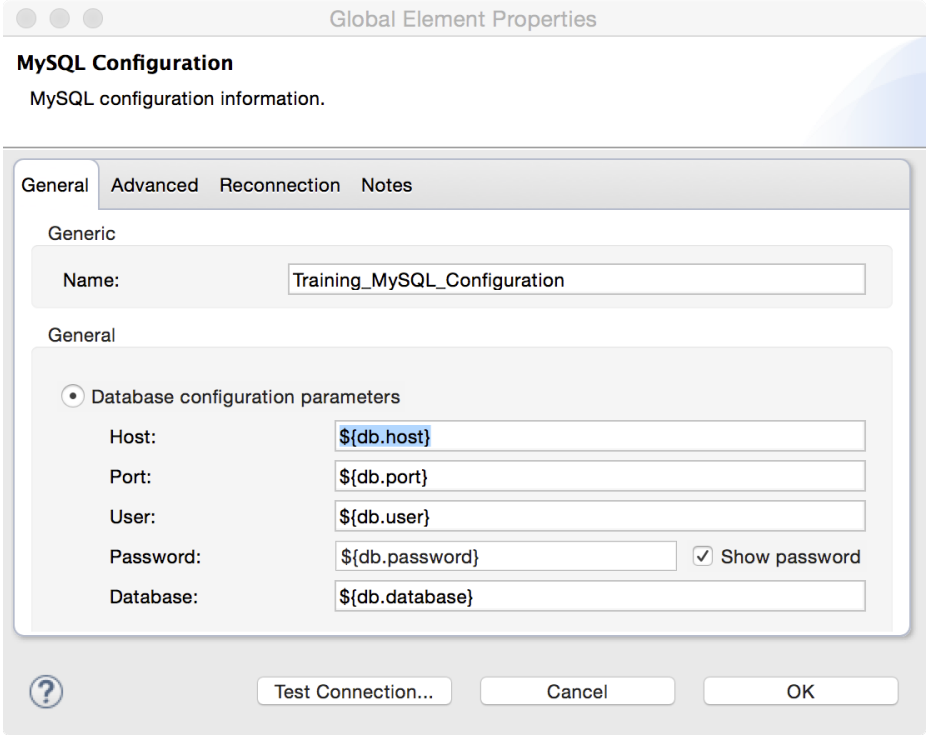


17. Copy the host value and then click OK.
18. Return to your properties file and create a property called db.host and paste the value you copied.
19. Create additional properties for the database port, user, password, and database values.



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

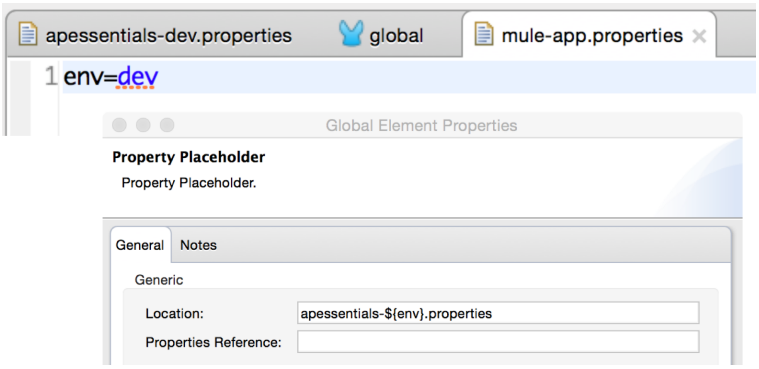
24. Save the file and run the application.
25. Make a request to http://localhost:8081/api/flights/SFO in the APIkit Consoles view or a browser and confirm it still works and returns American flights.

Note: If you have time, also parameterize your Salesforce credentials and your HTTP Request properties.

Walkthrough 11-2: Dynamically specify property files

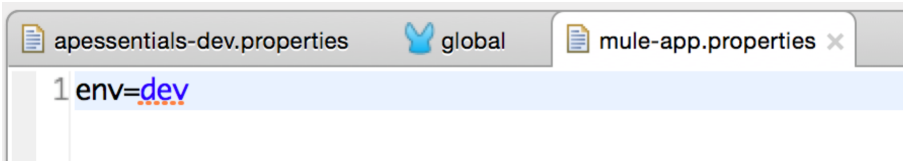
In this walkthrough, you will set the property file to use dynamically. You will:

- Define an environment property value in mule-app.properties.
- Use the environment property in the Property Placeholder.



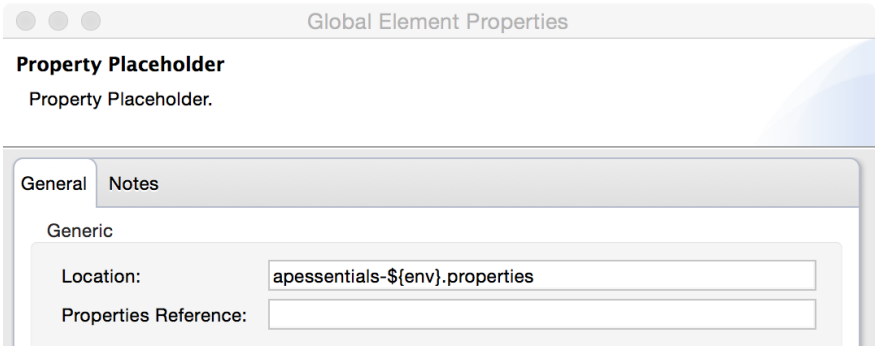
Define an environment property value in mule-app.properties

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.



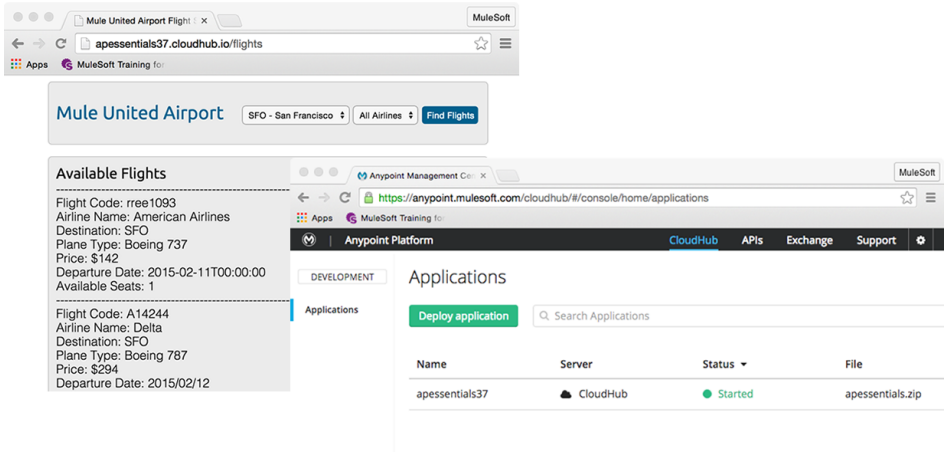
Test the application

7. Save all the files to redeploy the application.
8. Make a request to http://localhost:8081/api/flights/SFO in the APIkit Consoles view or a browser and confirm it still works and returns flights.

Walkthrough 11-3: (Optional) Deploy an application to the cloud

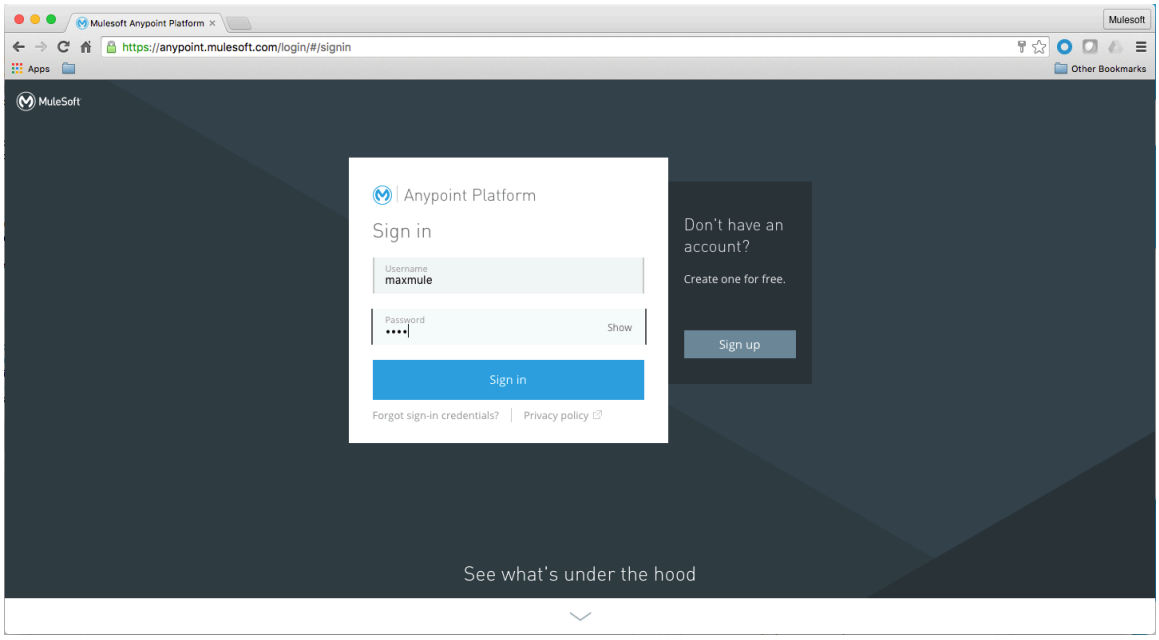
In this walkthrough, you will deploy your apessentials application to the cloud. You will:

- Deploy an application to CloudHub from Anypoint Studio.
- Run the application on its new, hosted domain.
- View application data in CloudHub.

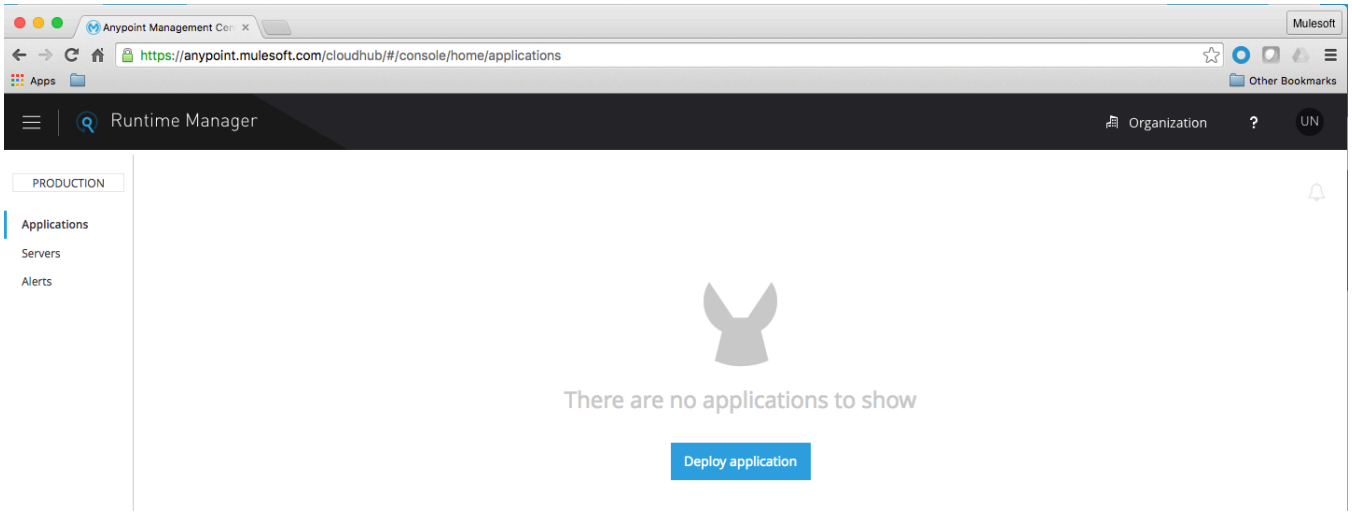


Access CloudHub

1. In a browser, go to http://anypoint.mulesoft.com and log in.



2. Click the upper left menu icon and select Runtime Manager.
3. Look at the applications; for a new account, there will be no applications listed.

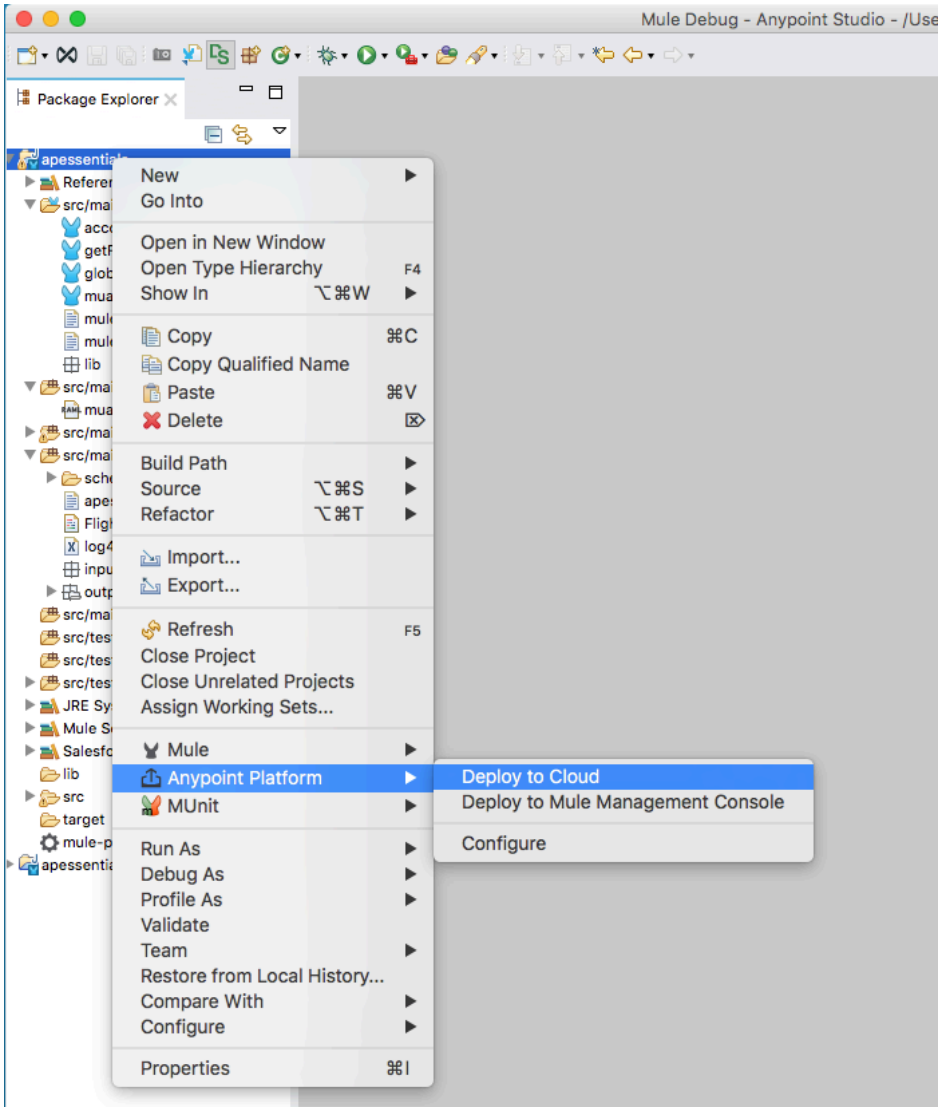


Deploy the application to CloudHub

4. Return to Anypoint Studio.
5. Right click on the apessentials project.

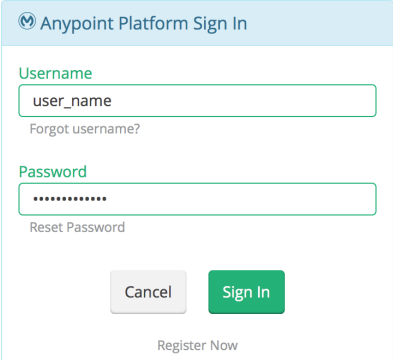
6. Select Anypoint Platform > Deploy to Cloud.

   Note: You can also right-click the project in the Package Explorer and select CloudHub > Deploy to CloudHub.



7. Click Next.

8. In the Anypoint Platform Sign In dialog box, enter your Anypoint Platform credentials.



9. Verify that you are logged into Anypoint Platform and you can see the Deploying Application window.
10. Notice that environment (Production/Dev) to deploy the application is given in left hand corner.

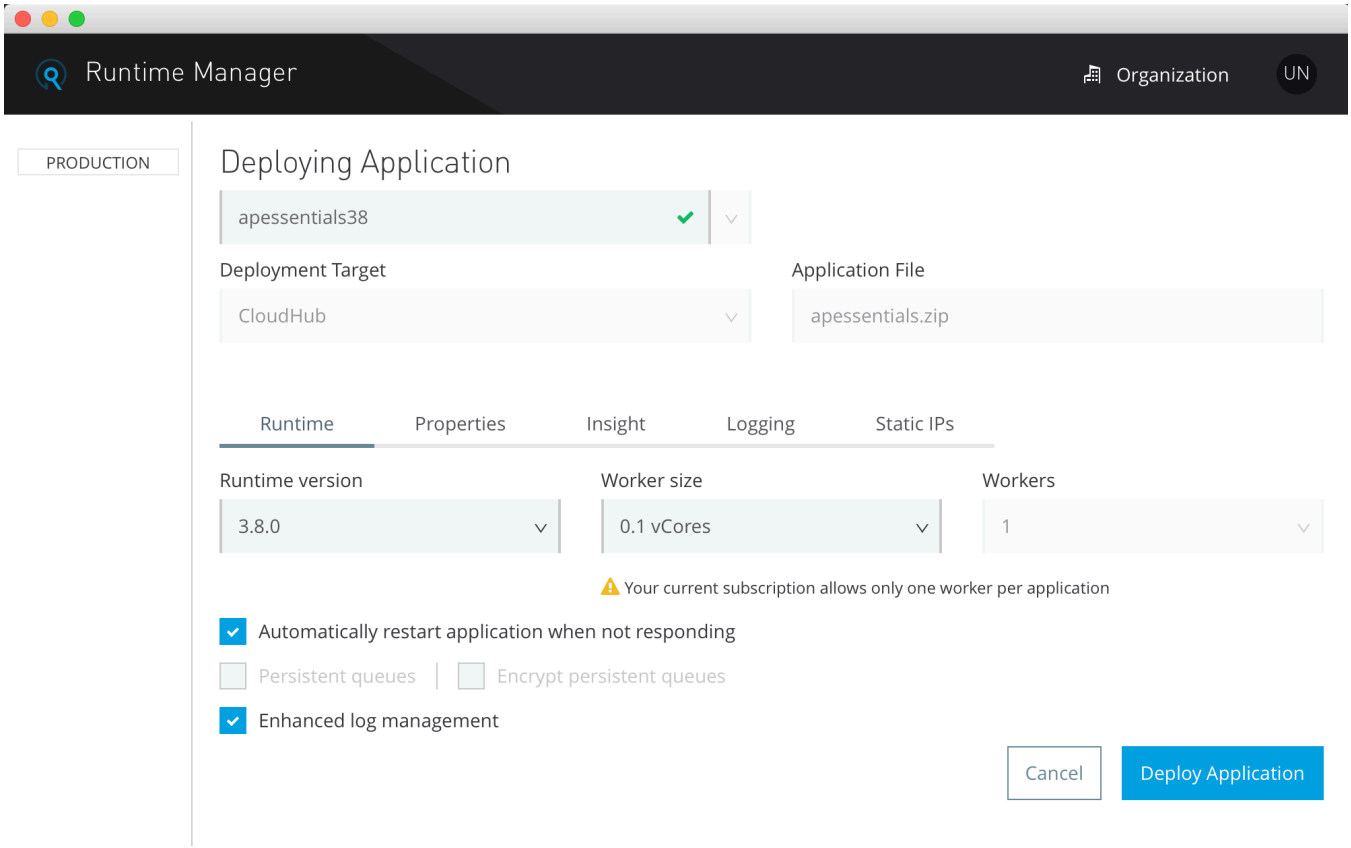
    Note: You can click on the environment name to switch environments or sometimes you might not have more than one option.

11. Enter a valid application name – this will be a part of 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, apessentials38-{your_lastname}. The availability and validity of the domain is instantly checked and you will get a green check mark if it is available.

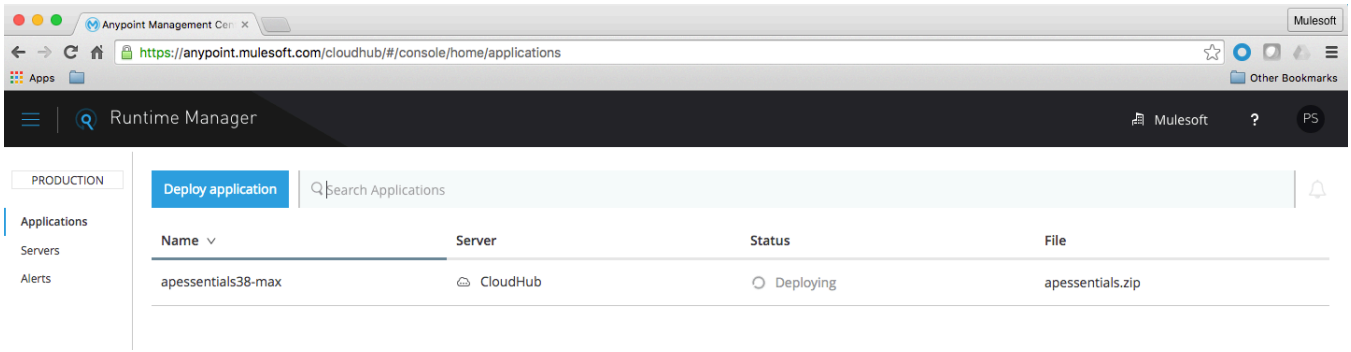
12. Set the Runtime version to the version your project is using, like 3.8.0
13. Look at the Properties tab; you should see the env variable set to dev.
14. Set the worker size to 0.1vCore.

15. Click Deploy Application.



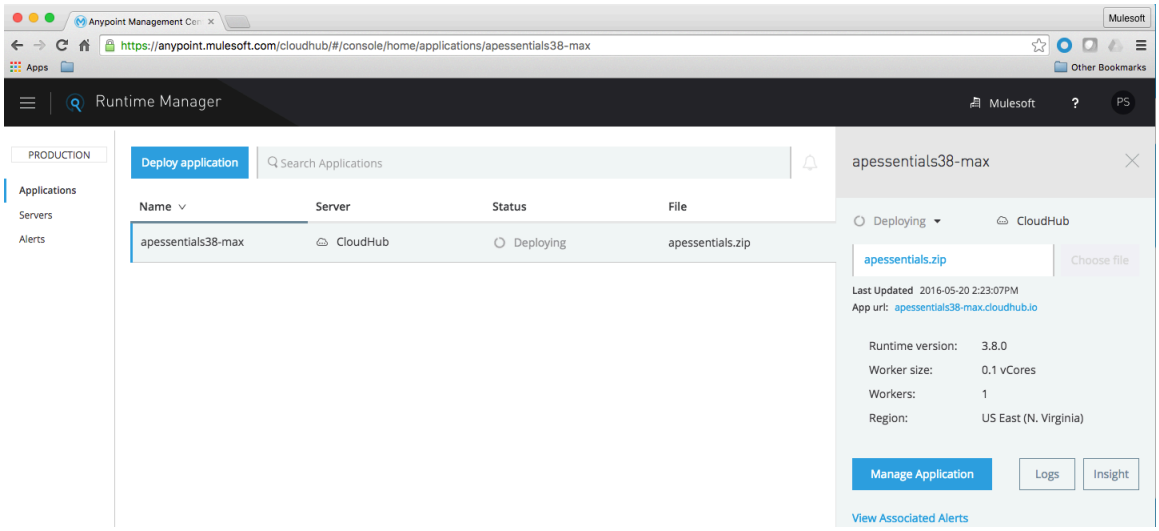
Monitor the application deployment to CloudHub

16. Return to CloudHub on Anypoint Platform and see your application listed; the status should be set to Deploying.



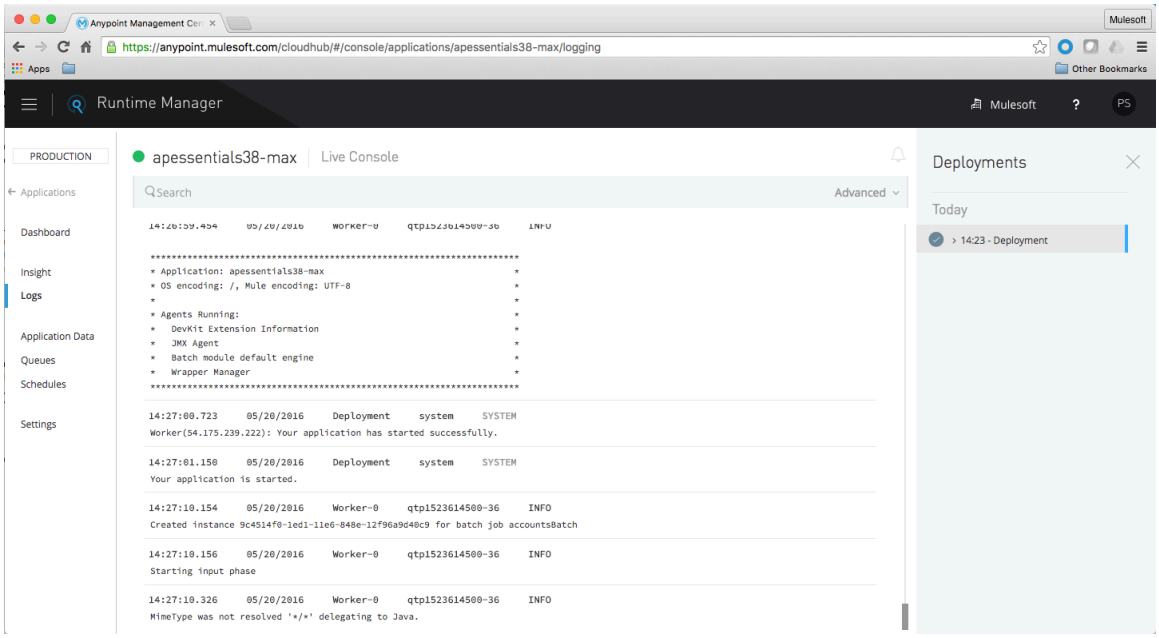
17. Click anywhere in the application row.

18. Click the Manage Application button that appears on the right side.



19. Click Logs in the left-side menu.
20. Watch the logs and wait until the application successfully deploys (or fails to deploy).

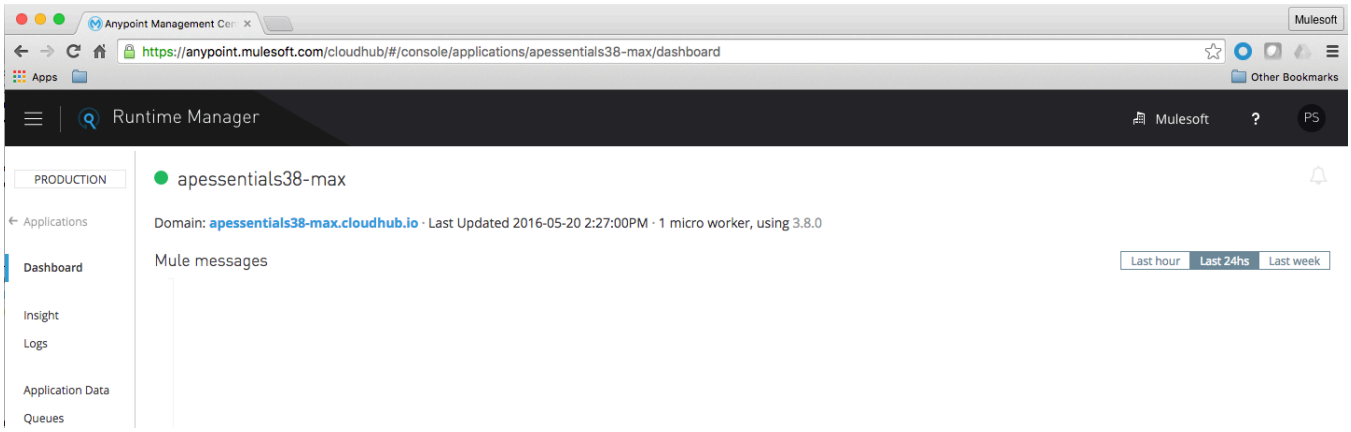
    Note: If your application did not successfully deploy, examine the log file to figure out why the application did not deploy. If you had errors when deploying, troubleshoot them, fix them in Anypoint Studio, and then redeploy the application to CloudHub.



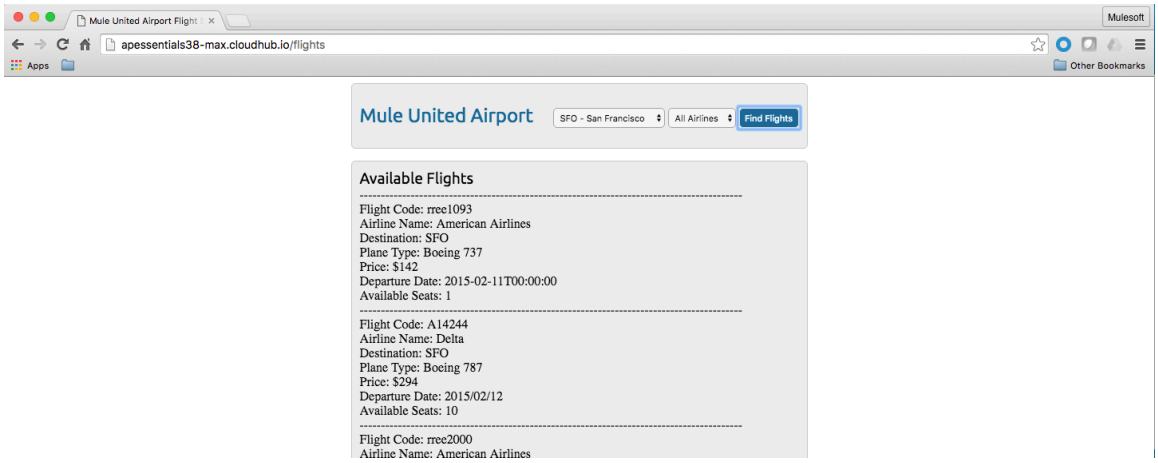
Test the application in the cloud

21. Click Dashboard in the left-side menu.

22. Click the domain link for the application, apessentials-*lastname*.cloudhub.io.



23. In the new browser tab that opens, add /flights to the URL path: http://apessentials38-*lastname*.cloudhub.io/flights; your application should work as before but now it is running on CloudHub.

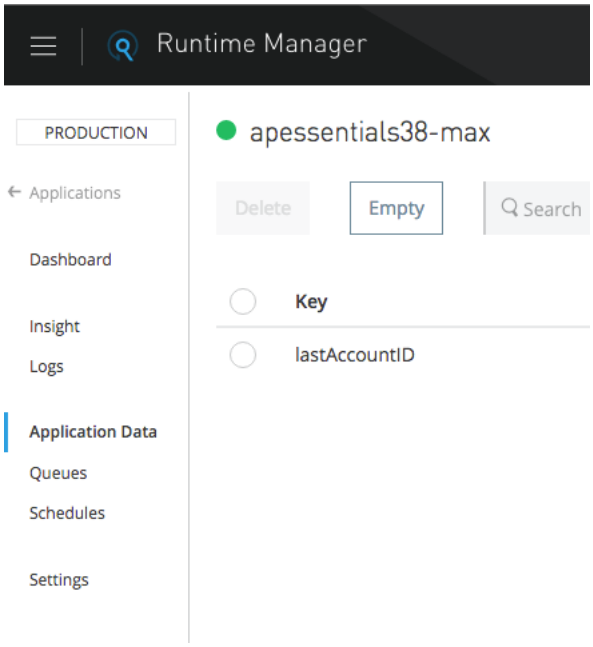


Note: You can also test your other endpoints: http://apessentials-lastname/sfdc and http://apessentials-lastname/api/flights/SFO.

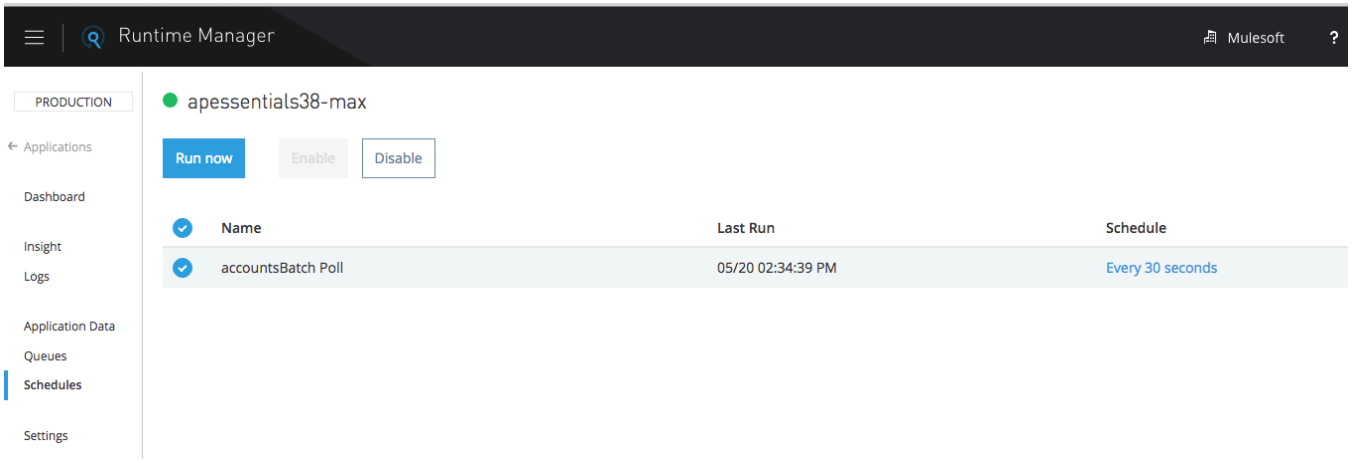
Explore application data on CloudHub

24. Return to Anypoint Platform.
25. Navigate to Runtime Manager by clicking on the upper left menu icon.
26. Select the Application row and click on Manage Application.

27. Click Application Data in the left-side menu; you should see your watermark variable.



28. Click Schedules in the left-side menu; you should see the application poll and its current schedule.
29. Select the accountsBatch poll and click the Disable button.

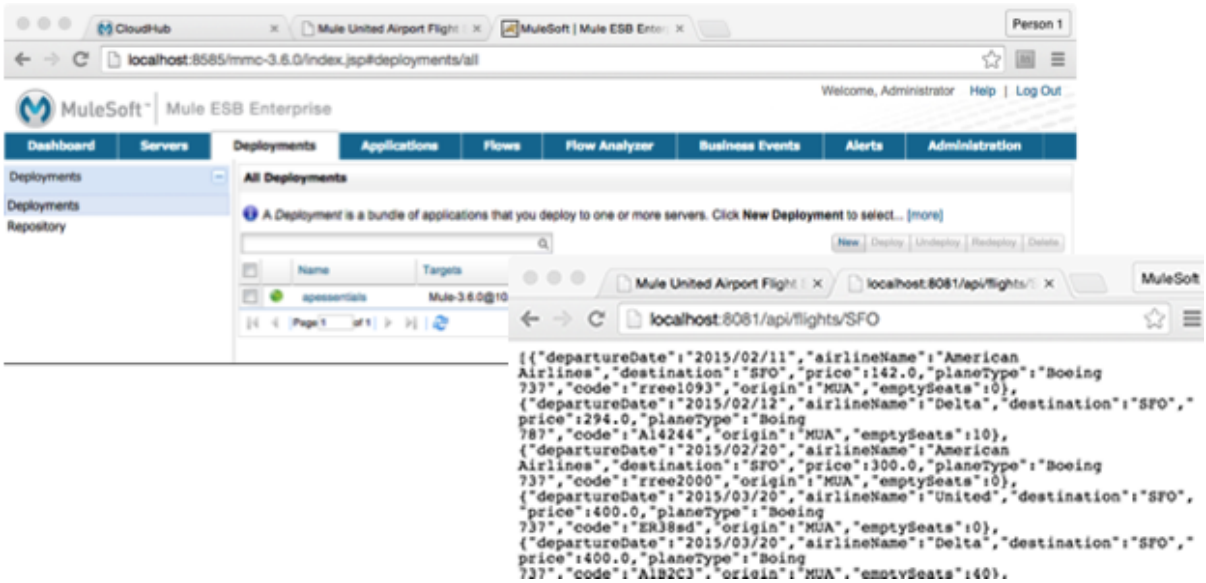


Walkthrough 11-4: (Optional) Deploy an application on-prem

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 standalone 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.

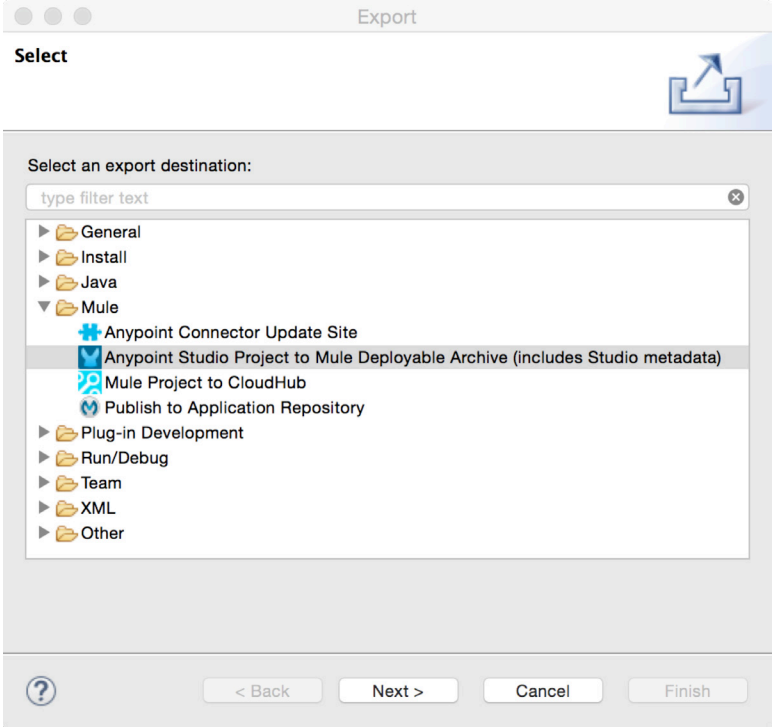


Package the application

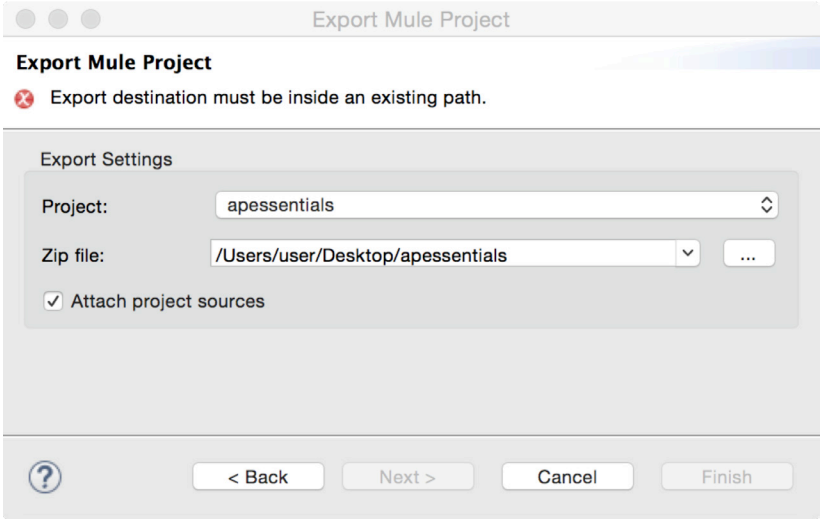
1. Return to Anypoint Studio and run the application to make sure it is working.
2. Click the red Terminate button to stop the embedded Mule runtime; YOU MUST DO THIS.
3. Select File > Export.

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.

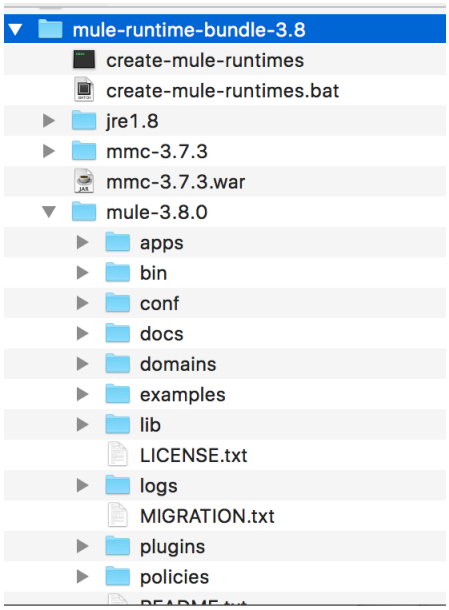


5. In the Export Mule Project dialog box, set the project to apessentials.
6. Browse to a location (like your desktop) and save the file as apessentials.



Set the environment property in the Mule wrapper.conf file

7. Locate the mmc-runtime-bundle-3.8.zip you downloaded from http://mulesoft-training.com/mule-runtime-bundle-3.8.zip.
8. Unzip the file into some directory.

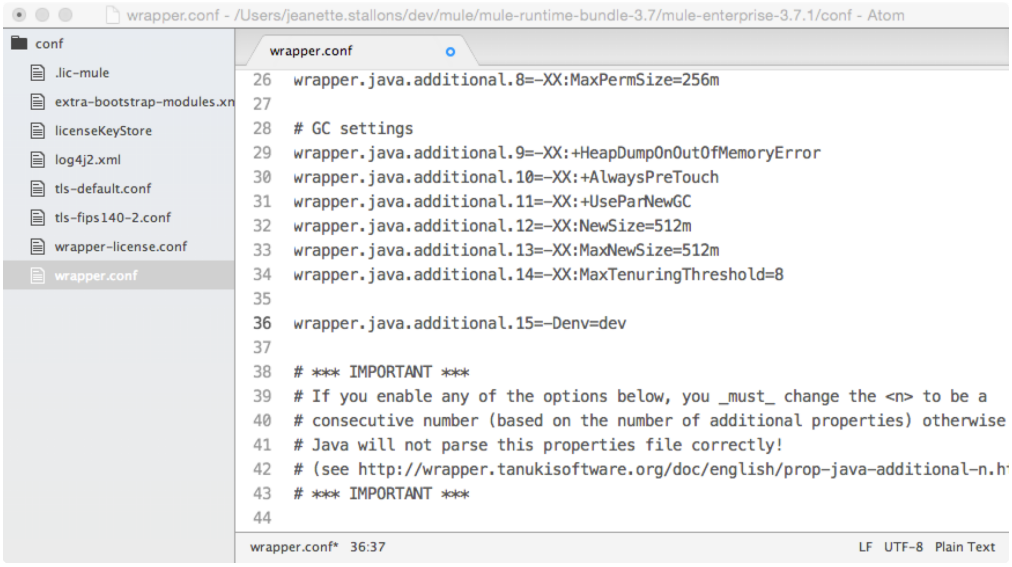


9. Navigate to the mule-3.8.X/conf folder.
10. Open wrapper.conf in a text editor.
11. Locate the last numbered wrapper.java.additional instruction near the top of the file.

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
    ```

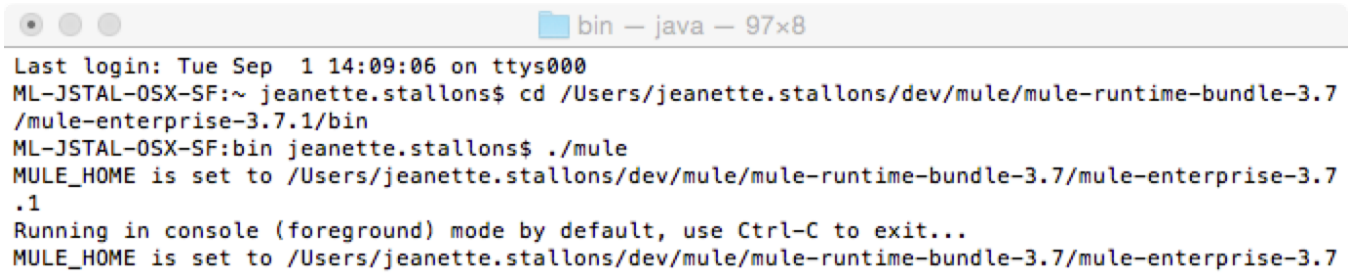
    Note: Be sure to use the correct number. This number may be different than shown here for your version of the runtime.



13. Save the file.

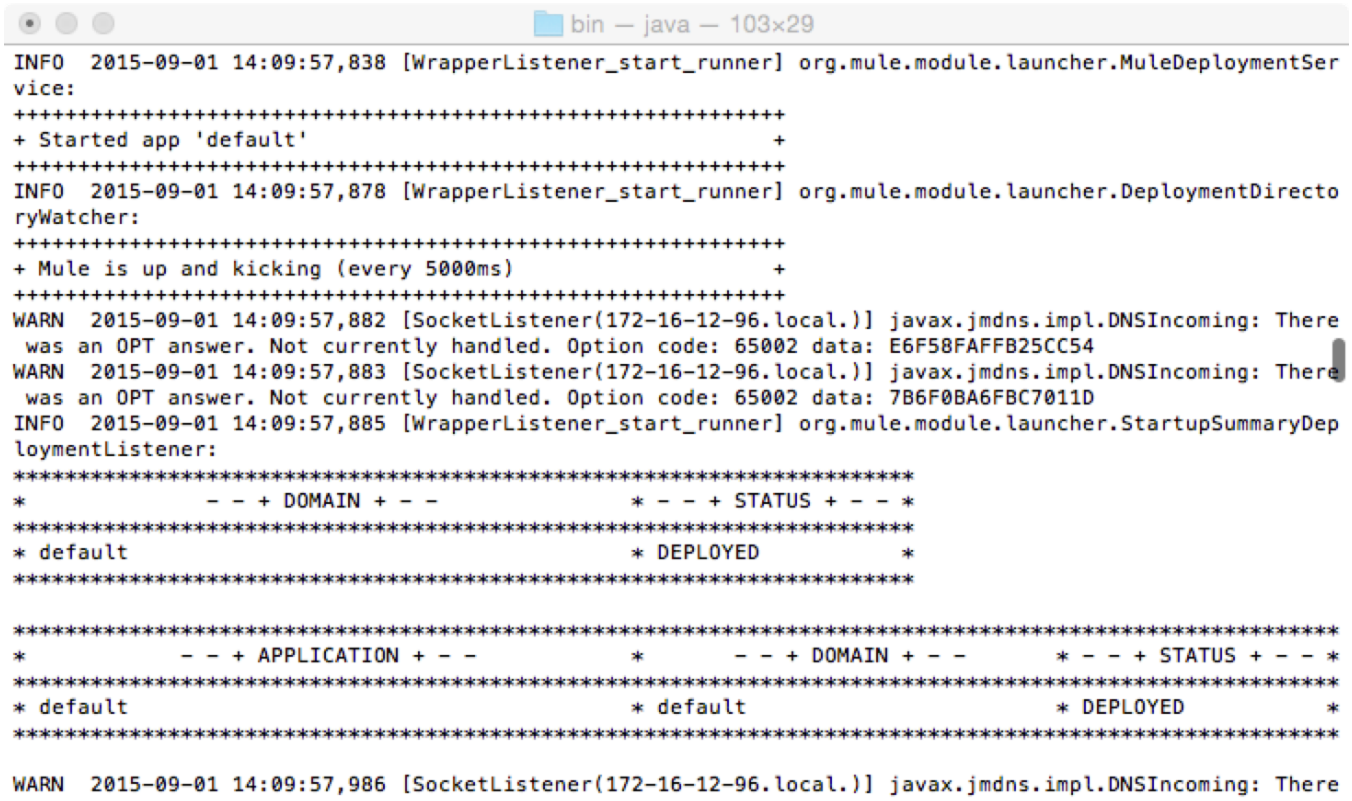
Start the Mule runtime

14. On your computer, open a Terminal or Command window.
15. Change to the /mule-runtime-bundle-3.8/mule-3.8.X/bin directory.
16. Run the mule script: mule (Unix) or mule.bat (Windows).



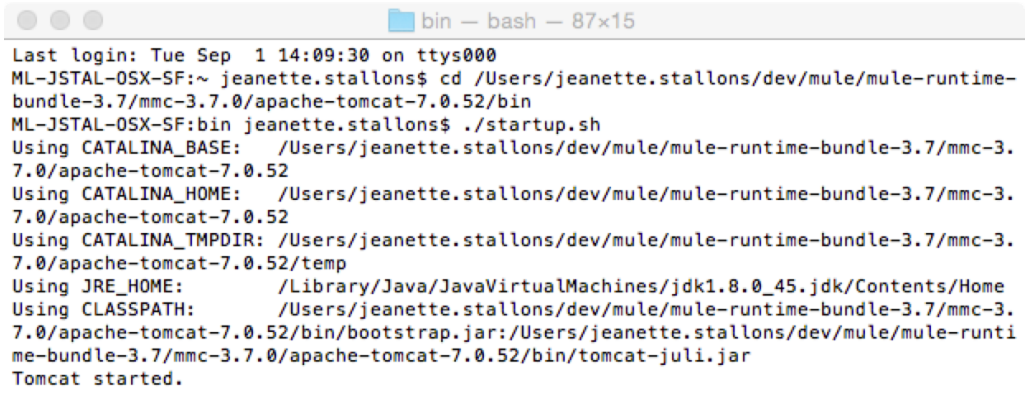
17. Wait until you see a message that Mule is up and kicking.

    Note: You may need to scroll up to see this if you get a lot of DNS warnings after it.



Start MMC

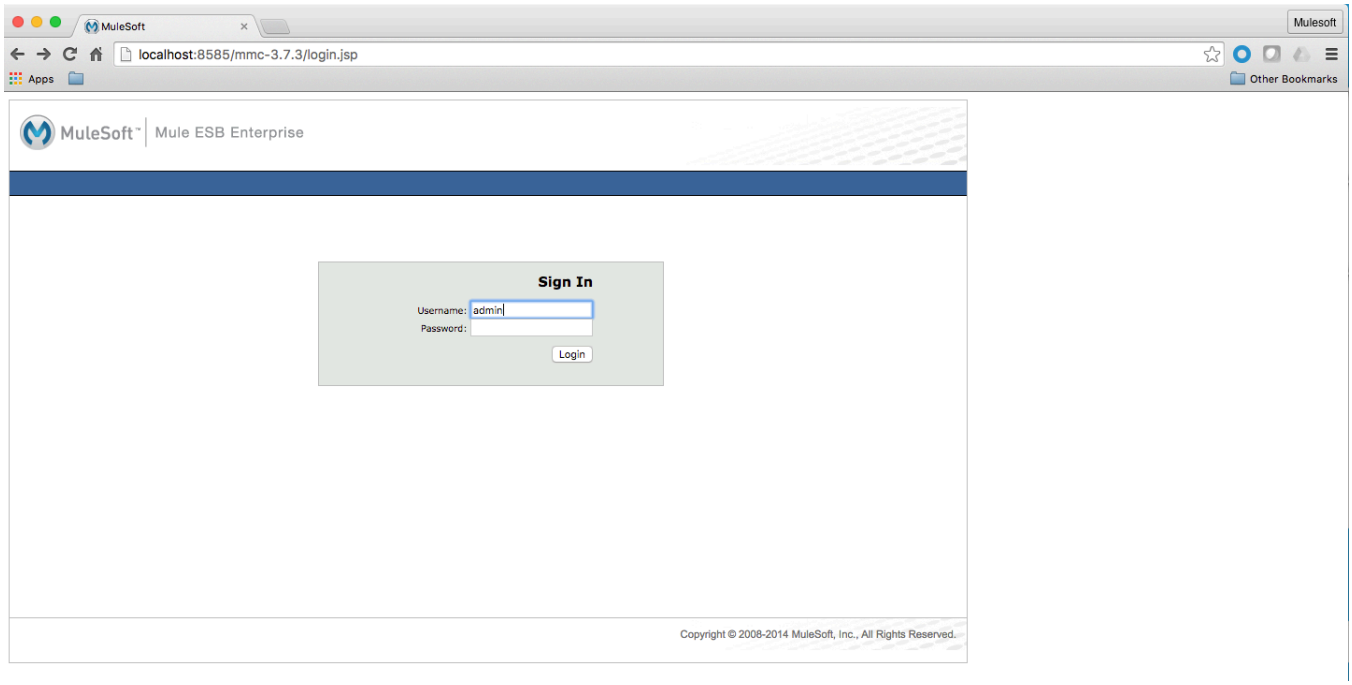
18. On your computer, open a second Terminal or Command window.
19. Change to the /mule-runtime-bundle-3.8/mmc-3.7.X/apache-tomcat-7.0.X/bin directory.
20. Run the startup script: ./startup.sh (Unix) or startup.bat (Windows).
21. Wait until you see a message that Tomcat started in your console.



22. In a browser window, navigate to http://localhost:8585/mmc-3.7.3.

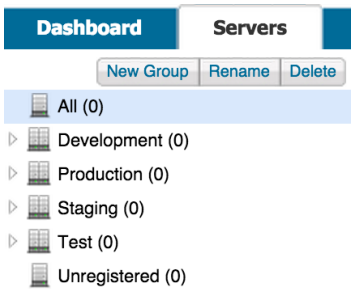
    Note: Change the URL as necessary for a different version of the runtime.

23. Enter a username and password of admin and click Login.



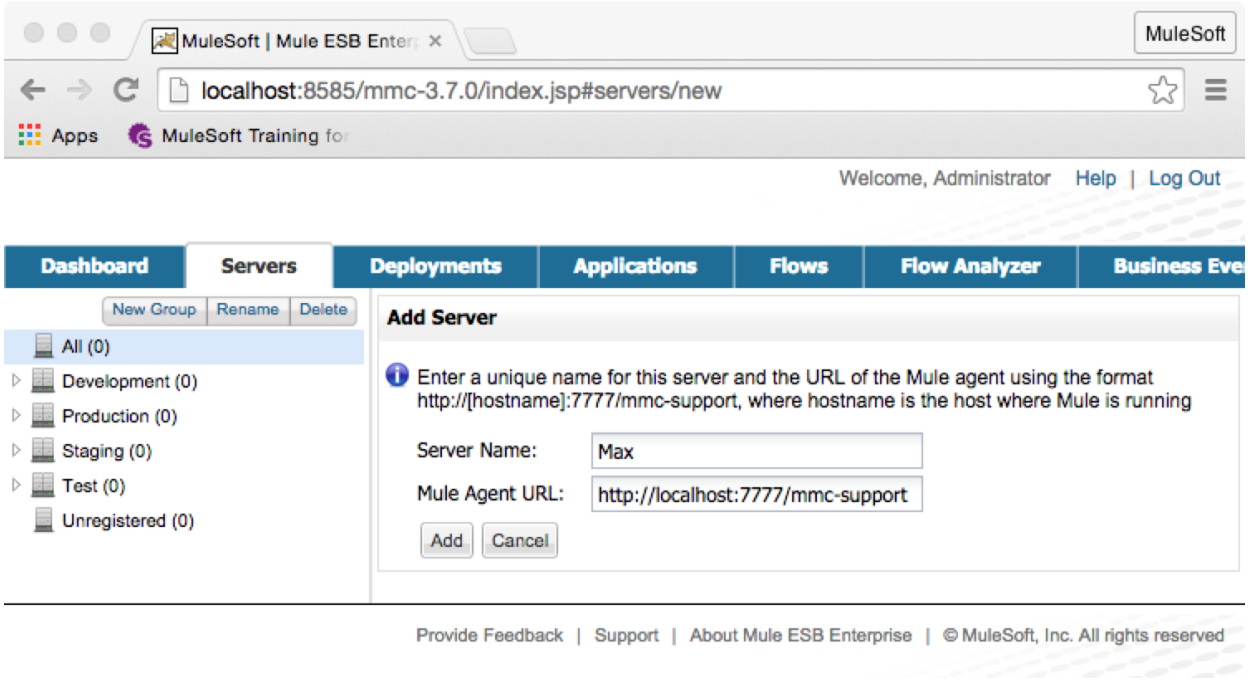
Register the Mule runtime in MMC

24. In MMC, click the Servers tab.
25. Look at the groups on the left; they should all show 0 registered servers.



26. Click the Add drop-down menu and select New Server.
27. Enter a server name of Max (or some other value).
28. Leave the Mule Agent URL as http://localhost:7777/mmc-support

29. Click Add.

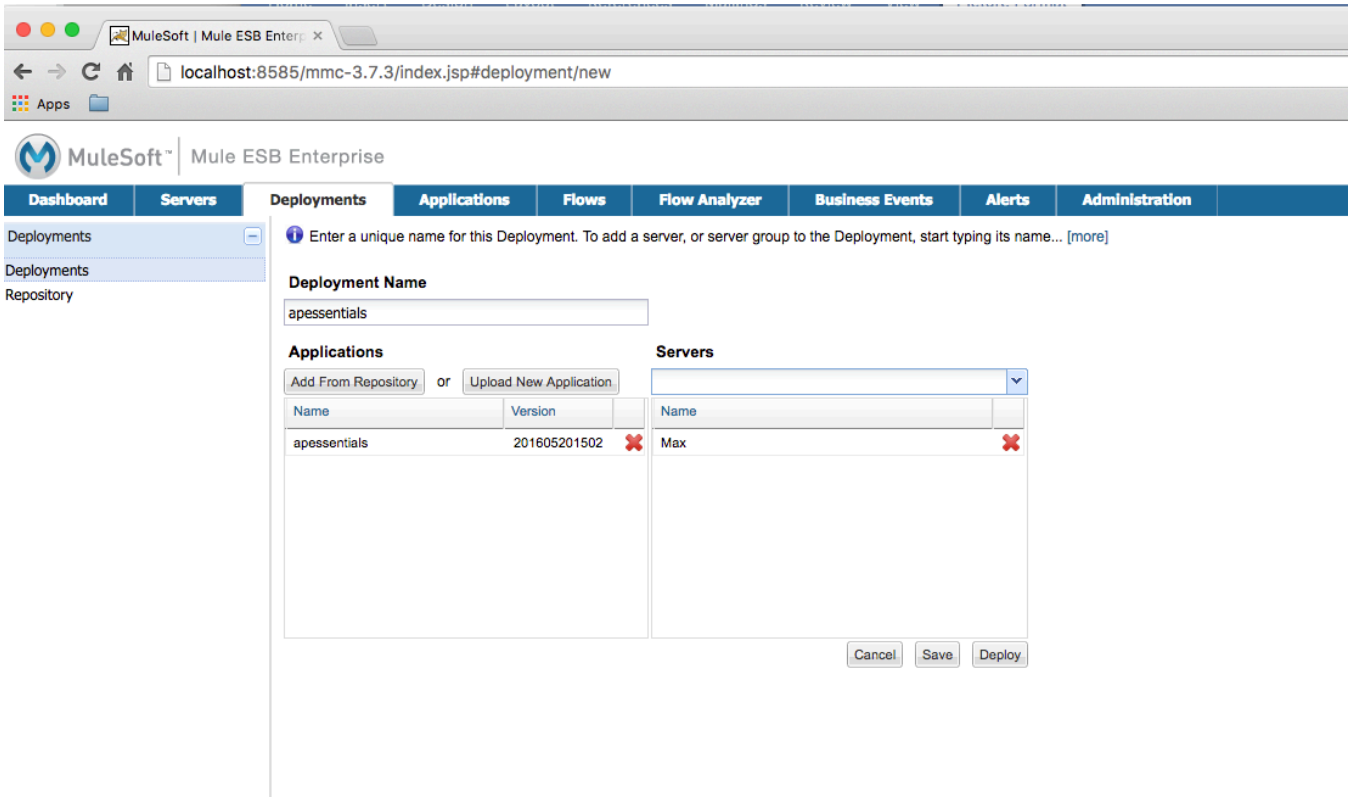


30. Wait for the server to register.

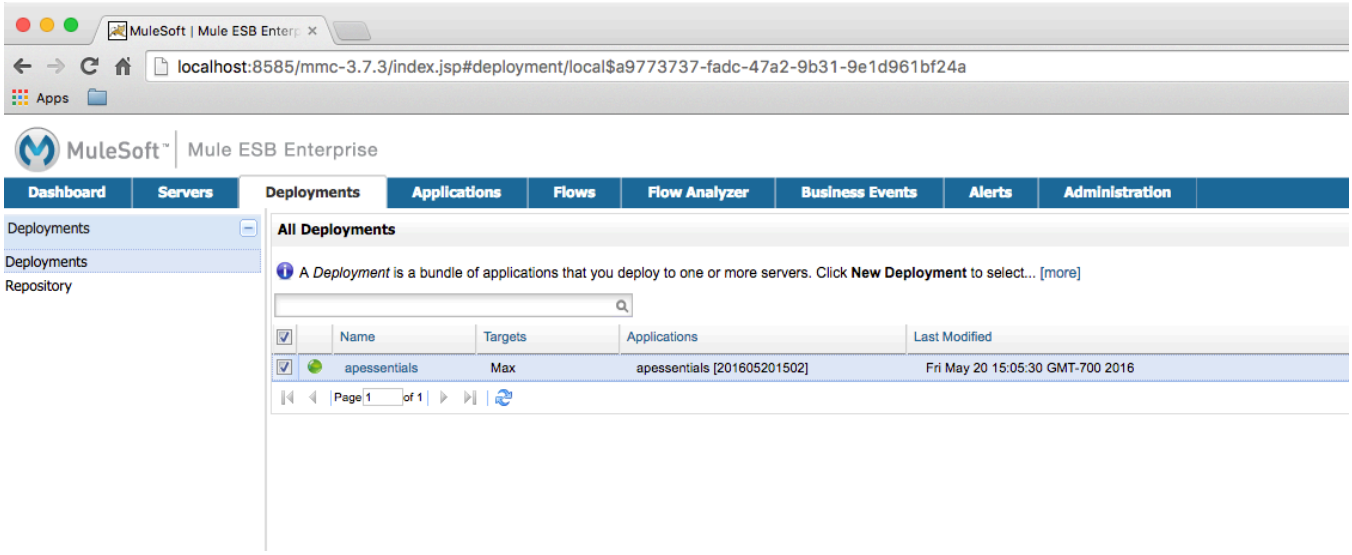
Deploy an application to the Mule runtime

31. Click the Deployments tab.
32. Click the New button.
33. Enter a deployment name of apessentials.
34. Select the server you just registered, Max.
35. Click the Upload New Application button.
36. Click the Browse button, browse to the zip file you created earlier, and click Open.

37. Click the Add button; you should see your archive listed.



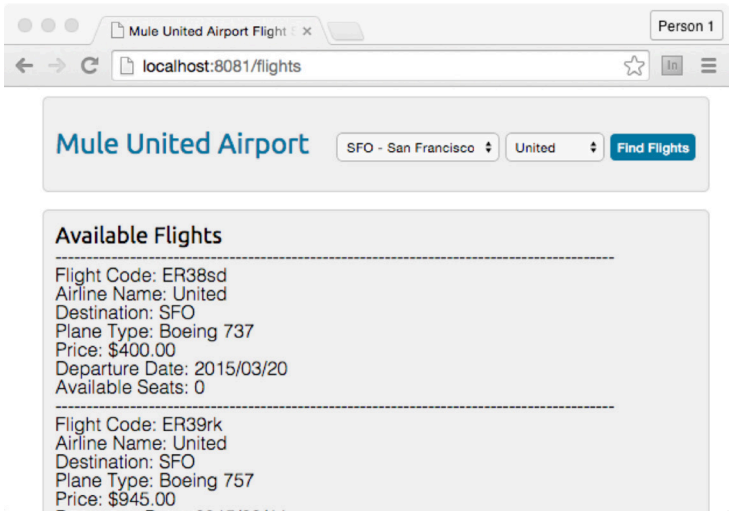
38. Click the Deploy button in the lower-right corner; you should see your application appear in the deployment list with a green circle next to it.



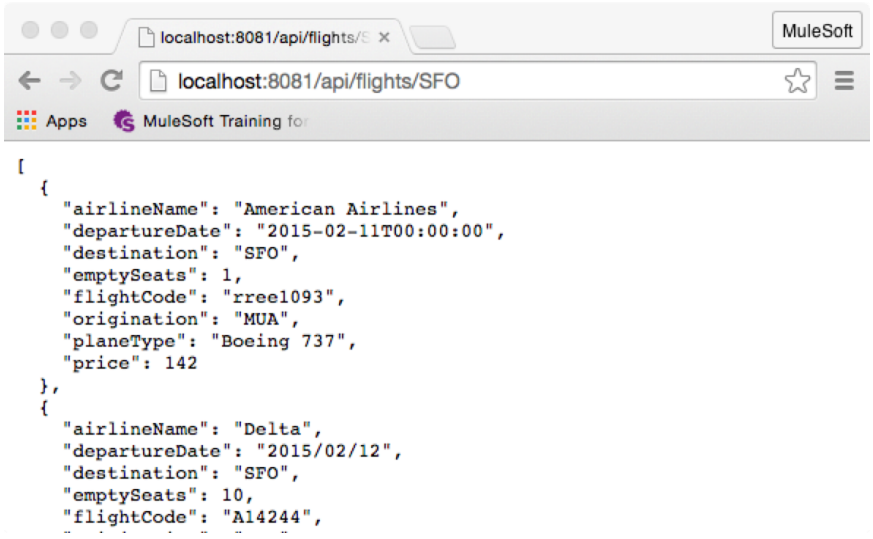
Note: If there is a red circle next to the application name, it was not deployed. Examine the log files to figure out why it failed. Return to the mule-runtime-bundle-3.8/mule-3.8.X folder and open the logs folder. Open the mule-app-apessentials-{version}.log file and locate the error. Fix the errors, create a new deployable archive if necessary, and then redeploy the app in MMC.

Test the application

39. 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.



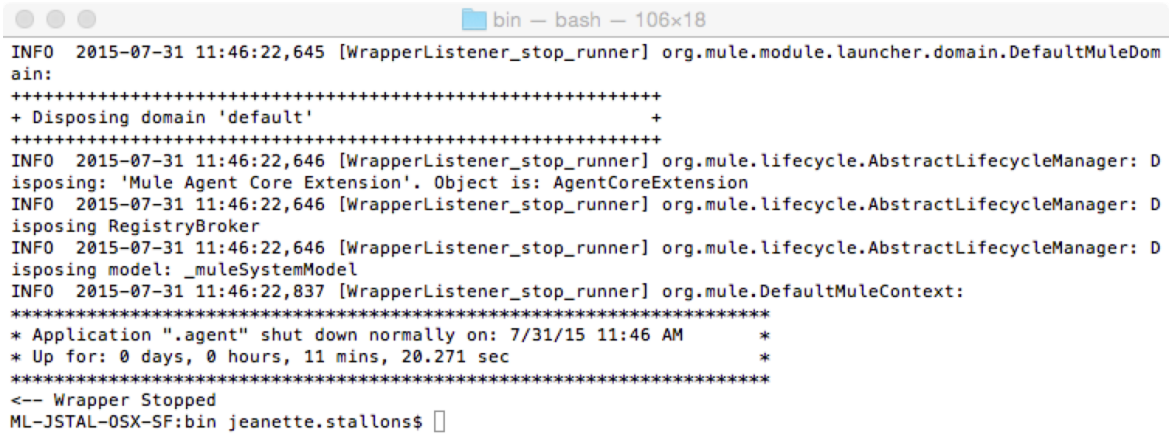
40. Make a request to http://localhost:8081/api/flights/SFO; you should get flight results.



Stop MMC and the Mule runtime

41. Return to the MMC Terminal/Command window.
42. Run the shutdown script: ./shutdown.sh (Unix) or shutdown.bat (Windows).
43. Close the Terminal/Command window.
44. Return to the Mule Terminal/Command window.
45. Stop the server by pressing Ctrl+C.

46. Wait until the runtime stops.



47. Close the Terminal/Command window.