



MuleSoft®

Anypoint Platform Development: Fundamentals

Student Manual

Mule runtime 4.2
November 16, 2019

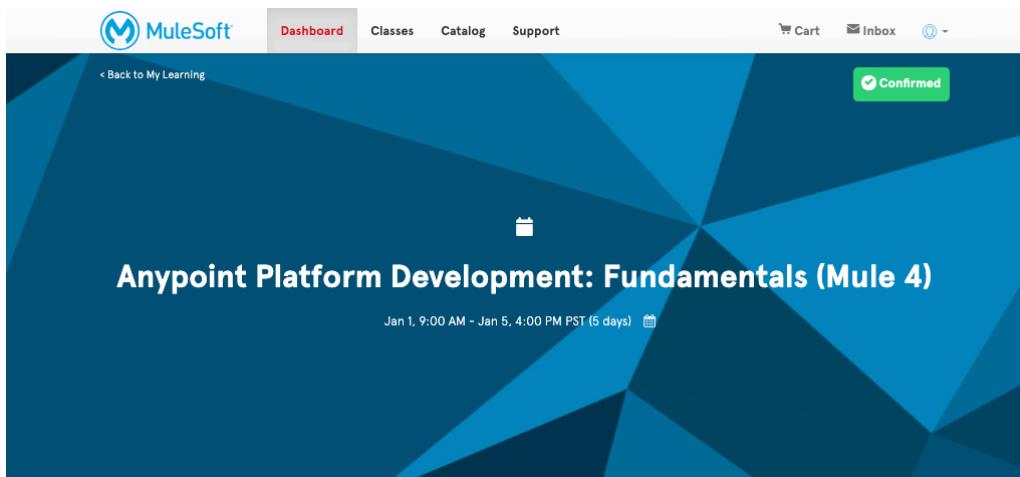
Table of Contents

INTRODUCING THE COURSE.....	5
Walkthrough: Set up your computer for class	6
PART 1: BUILDING APPLICATION NETWORKS WITH ANYPOINT PLATFORM ...	12
MODULE 1: INTRODUCING APPLICATION NETWORKS AND API-LED CONNECTIVITY	13
Walkthrough 1-1: Explore an API directory and an API portal.....	14
Walkthrough 1-2: Make calls to an API.....	20
MODULE 2: INTRODUCING ANYPOINT PLATFORM.....	31
Walkthrough 2-1: Explore Anypoint Platform and Anypoint Exchange.....	32
Walkthrough 2-2: Create a Mule application with Flow Designer.....	41
Walkthrough 2-3: Create an integration application with Flow Designer that consumes an API.....	51
MODULE 3: DESIGNING APIS	66
Walkthrough 3-1: Use API Designer to define an API with RAML.....	67
Walkthrough 3-2: Use the mocking service to test an API	73
Walkthrough 3-3: Add request and response details	77
Walkthrough 3-4: Add an API to Anypoint Exchange.....	91
Walkthrough 3-5: Share an API.....	103
MODULE 4: BUILDING APIS	111
Walkthrough 4-1: Create a Mule application with Anypoint Studio.....	112
Walkthrough 4-2: Connect to data (MySQL database)	118
Walkthrough 4-3: Transform data.....	130
Walkthrough 4-4: Create a RESTful interface for a Mule application	140
Walkthrough 4-5: Use Anypoint Studio to create a RESTful API interface from a RAML file.....	149
Walkthrough 4-6: Implement a RESTful web service.....	159
MODULE 5: DEPLOYING AND MANAGING APIS	163
Walkthrough 5-1: Deploy an application to CloudHub	164
Walkthrough 5-2: Create and deploy an API proxy.....	171
Walkthrough 5-3: Restrict API access with policies and SLAs.....	183
Walkthrough 5-4: Request and grant access to a managed API.....	191
Walkthrough 5-5: Add client ID enforcement to an API specification	200

PART 2: BUILDING APPLICATIONS WITH ANYPOINT STUDIO	208
MODULE 6: ACCESSING AND MODIFYING MULE EVENTS.....	209
Walkthrough 6-1: View event data.....	210
Walkthrough 6-2: Debug a Mule application.....	217
Walkthrough 6-3: Track event data as it moves in and out of a Mule application.....	225
Walkthrough 6-4: Set request and response data.....	232
Walkthrough 6-5: Get and set event data using DataWeave expressions	236
Walkthrough 6-6: Set and get variables.....	245
MODULE 7: STRUCTURING MULE APPLICATIONS.....	249
Walkthrough 7-1: Create and reference subflows and private flows	250
Walkthrough 7-2: Pass events between flows using the VM connector	254
Walkthrough 7-3: Encapsulate global elements in a separate configuration file.....	263
Walkthrough 7-4: Use property placeholders in connectors.....	273
Walkthrough 7-5: Create a well-organized Mule project	277
Walkthrough 7-6: Manage metadata for a project.....	287
MODULE 8: CONSUMING WEB SERVICES.....	295
Walkthrough 8-1: Consume a RESTful web service that has an API (and connector) in Exchange.	296
Walkthrough 8-2: Consume a RESTful web service	308
Walkthrough 8-3: Consume a SOAP web service	317
Walkthrough 8-4: Transform data from multiple services to a canonical format	329
MODULE 9: CONTROLLING EVENT FLOW.....	338
Walkthrough 9-1: Multicast an event	339
Walkthrough 9-2: Route events based on conditions.....	346
Walkthrough 9-3: Validate events.....	359
MODULE 10: HANDLING ERRORS	365
Walkthrough 10-1: Explore default error handling.....	366
Walkthrough 10-2: Handle errors at the application level.....	375
Walkthrough 10-3: Handle specific types of errors	384
Walkthrough 10-4: Handle errors at the flow level	389
Walkthrough 10-5: Handle errors at the processor level.....	398
Walkthrough 10-6: Map an error to a custom error type	405
Walkthrough 10-7: Review and integrate with APIkit error handlers	411
Walkthrough 10-8: Set a reconnection strategy for a connector	424

MODULE 11: WRITING DATAWEAVE TRANSFORMATIONS	426
Walkthrough 11-1: Create transformations with the Transform Message component	427
Walkthrough 11-2: Transform basic JSON, Java, and XML data structures	439
Walkthrough 11-3: Transform complex data structures with arrays	445
Walkthrough 11-4: Transform to and from XML with repeated elements	456
Walkthrough 11-5: Define and use variables and functions	464
Walkthrough 11-6: Coerce and format strings, numbers, and dates	470
Walkthrough 11-7: Define and use custom data types	477
Walkthrough 11-8: Use DataWeave functions	481
Walkthrough 11-9: Look up data by calling a flow.....	486
PART 3: BUILDING APPLICATIONS TO SYNCHRONIZE DATA	490
MODULE 12: TRIGGERING FLOWS	491
Walkthrough 12-1: Trigger a flow when a new file is added to a directory	492
Walkthrough 12-2: Trigger a flow when a new record is added to a database and use automatic watermarking	500
Walkthrough 12-3: Schedule a flow and use manual watermarking.....	511
Walkthrough 12-4: Publish and listen for JMS messages.....	525
MODULE 13: PROCESSING RECORDS	532
Walkthrough 13-1: Process items in a collection using the For Each scope	533
Walkthrough 13-2: Process records using the Batch Job scope	540
Walkthrough 13-3: Use filtering and aggregation in a batch step.....	547

Introducing the Course



In this module, you will:

- Learn about the course format.
- Download the course files.
- Make sure your computer is set up for class.
- Review the course outline.

Walkthrough: Set up your computer for class

In this walkthrough, you make sure your computer is set up correctly, so you can complete the class exercises. You will:

- Download the course files from the MuleSoft Training Learning Management System.
- Make sure you have JDK 1.8 and that it is included in your PATH environment variable.
- Make sure Anypoint Studio starts successfully.
- Install Advanced REST client (if you did not already).
- Make sure you have an active Anypoint Platform account.
- Make sure you have a Salesforce developer account and an API security token.

Download student files

1. In a web browser, navigate to <http://training.mulesoft.com>.
2. Click the My training account link.

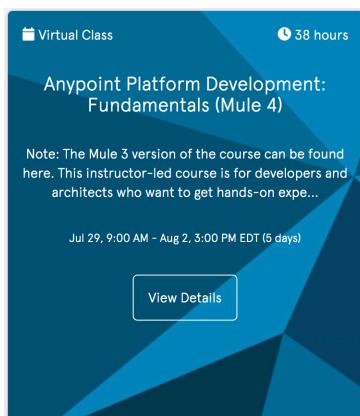
The screenshot shows the MuleSoft Training website. At the top, there is a navigation bar with links for Product, Solutions, Services, Resources, Company, Developers, Partners, Contact, a Free trial button, Login, and a search icon. Below the navigation bar, the page title is "My training account". The main content area has a dark background with a geometric pattern. It displays the text "Training and certification" and "Register for upcoming classes". There is also a breadcrumb trail: Training and certification > Training > Home.

3. Log in to your MuleSoft training account using the email that was used to register you for class.

Note: If you have never logged in before and do not have a password, click the [Forgot your password link](#), follow the instructions to obtain a password, and then log in.

4. On the My Learning page, locate the card for your class.

Note: If you do not see your event, locate the Current and Completed buttons under the My Learning tab, click the Completed button, and look for your event here.



- Click the event's View Details button.
- Locate the list of course materials on the right side of the page.

The screenshot shows the MuleSoft Learning Platform interface. At the top, there are navigation links: Dashboard, Classes, Catalog, Support, Cart, Inbox, and a user icon. A green button labeled 'Confirmed' is visible. The main title of the course is 'Anypoint Platform Development: Fundamentals (Mule 4)'. Below the title, it says 'Jan 1, 9:00 AM - Jan 5, 4:00 PM PST (5 days)'. The right sidebar contains sections for 'INSTRUCTORS' (empty), 'MATERIALS' (with four items: APDevFundamentals4.2 Student Files (ZIP), APDevFundamentals4.2 Student Manual (PDF), APDevFundamentals4.2 Student Slides (ZIP), and MCD - Level 1 Exam Datasheet (PDF)), and a note about the Mule 3 version of the course.

- Click the student files link to download the files.
- Click the student manual link to download the manual.
- Click the student slides link to download the slides.
- On your computer, locate the student files ZIP and expand it.
- Open the course snippets.txt file.

Note: Keep this file open. You will copy and paste text from it during class.

Make sure you have JDK 1.8

- On your computer, open Terminal (Mac) or Command Prompt (Windows) or some other command-line interface.
- Type java –version and press enter.

```
java –version
```

14. Look at the output and check if you have a supported JDK 1.8 installed.

```
$ java -version  
openjdk version "1.8.0_212"  
OpenJDK Runtime Environment (AdoptOpenJDK)(build 1.8.0_212-b03)  
OpenJDK 64-Bit Server VM (AdoptOpenJDK)(build 25.212-b03, mixed mode)
```

Note: If you have an older version of the JDK installed or have no version at all, go to <https://adoptopenjdk.net/> and download the correct version of JDK 1.8 for your operating system. Install and then confirm with java -version in a command-line interface again.

Note: JDK 1.9 or newer is NOT supported. Oracle JDK 1.8 is also supported although OpenJDK 1.8 is recommended.

Make sure you have Java in your PATH environment variable

15. In a command-line interface, type echo \$PATH (Mac) or %echo PATH% (Windows) and press enter to print the current value of your PATH environment variable.

- Mac: echo \$PATH
- Windows: echo %PATH%

16. After installing the correct JDK version, add or update an environment variable named JAVA_HOME that points to the installation location and then add JAVA_HOME/bin to your PATH environment variable.

Note: For instructions on how to set or change environment variables, see the following instructions for PATH: <http://docs.oracle.com/javase/tutorial/essential/environment/paths.html>.

17. Print your PATH environment variable once again and ensure it contains JAVA_HOME/bin.

Start Anypoint Studio

18. In your computer's file browser, navigate to where you installed Anypoint Studio and open it.

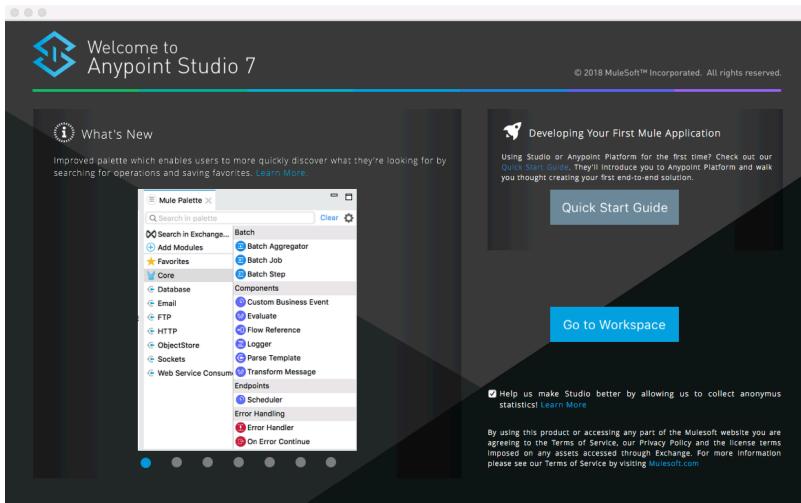
Note: If you do not have Anypoint Studio, you can download it from <https://www.mulesoft.com/lp/dl/studio>.

19. In the Workspace Launcher dialog box, look at the location of the default workspace; change the workspace location if you want.

20. Click OK to select the workspace; Anypoint Studio should open.

Note: If you cannot successfully start Anypoint Studio, make sure the JDK and Anypoint Studio are BOTH 64-bit or BOTH 32-bit and that you have enough available memory (at least 8GB available) to run Anypoint Studio.

21. If you get a Welcome Page, click the X on the tab to close it.



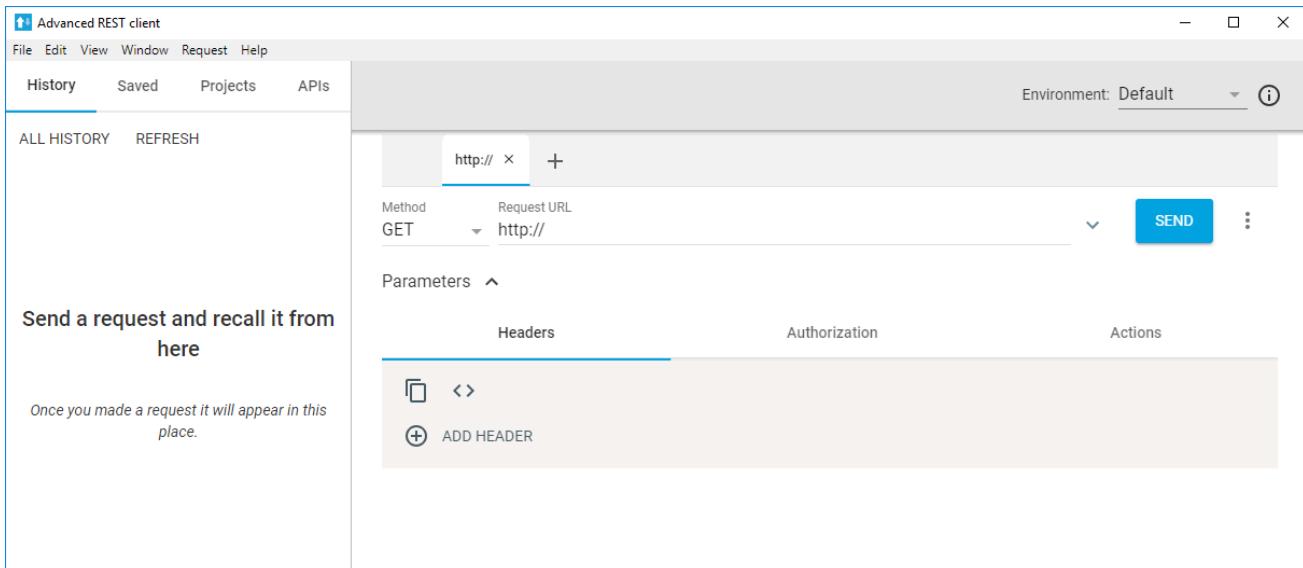
22. If you get an Updates Available pop-up in the lower-right corner of the application, click it and install the available updates.

Open Advanced REST Client

23. Open Advanced REST Client.

Note: If you do not have Advanced REST Client (or another REST API client) installed, download it now from <https://install.advancedrestclient.com/> and install it.

24. Leave Advanced REST client open; you will use it throughout class.



Make sure you have an active Anypoint Platform account

25. In a web browser, navigate to <http://anypoint.mulesoft.com/> and log in.

Note: If you do not have an account, sign up for a free, 30-day trial account now.

26. Click the menu button located in the upper-left in the main menu bar.



27. In the menu that appears, select Access Management.

Note: This will be called the main menu from now on.

The screenshot shows the Anypoint Platform main menu. On the left, there's a sidebar with icons for Design Center, Exchange, Management Center, Access Management (which is highlighted in blue), API Manager, Runtime Manager, and Data Gateway. The main content area has a blue banner at the top stating 'Available now: Unified, graph-based modeling of APIs to help you adopt API design best practices. →'. Below the banner, there are three cards: 'Management Center' (with a blue icon), 'Access Management' (with a blue icon), and 'API Manager' (with a blue icon). The 'Access Management' card has a tooltip: 'Manage Mule applications and APIs. Create, test, and reuse API fragments.'

28. In the left-side navigation, click the Runtime Manager link under Subscription.

29. Check your subscription level and if it is a trial account, make sure it is not expired.

Note: If your trial is expired or will expire during class, sign out and then sign up for a new trial account now.

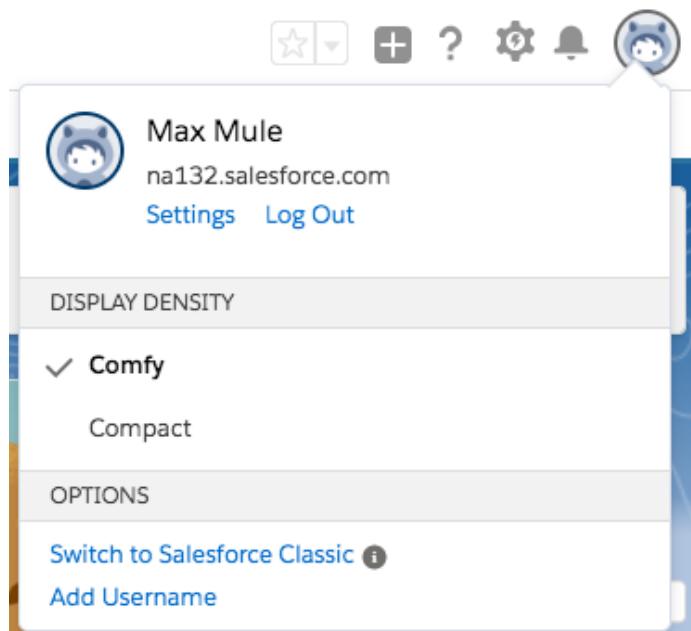
The screenshot shows the 'Access Management' page. On the left, there's a sidebar with 'ACCESS MANAGEMENT' (Organization, Users, Roles, Environments, External Identity, Audit Logs), 'SETTINGS' (Runtime Manager, Flow Designer), and 'SUBSCRIPTION' (Runtime Manager, Object Store). The main content area has two columns. The left column is titled 'Subscription Information' and shows: Organization Name: Training, Subscription Tier: Developer - Trial, and Expiration Date: Expires on 05/17/2019. The right column is titled 'Production' and shows environments for production applications. It includes sections for 'Sandbox' (vCores 0 / 0) and 'Static IP' (IPs 0 / 0). At the bottom, there's a note: 'Design environments for development of applications.' with a progress bar showing 0 / 1.

Make sure you have a Salesforce developer account and an API security token

30. In the same or another web browser tab, navigate to <http://salesforce.com> and log in to the Salesforce CRM.

Note: If you did not sign up for a free developer account yet, go to <http://developer.salesforce.com/> and sign up for one now. You will want to use a free developer account and not your company account (if you already have one) for class. You will use the API to add new fictitious accounts to it and will probably not want to add those to your real data.

31. In Salesforce, click your avatar at the top of the screen and select Settings.



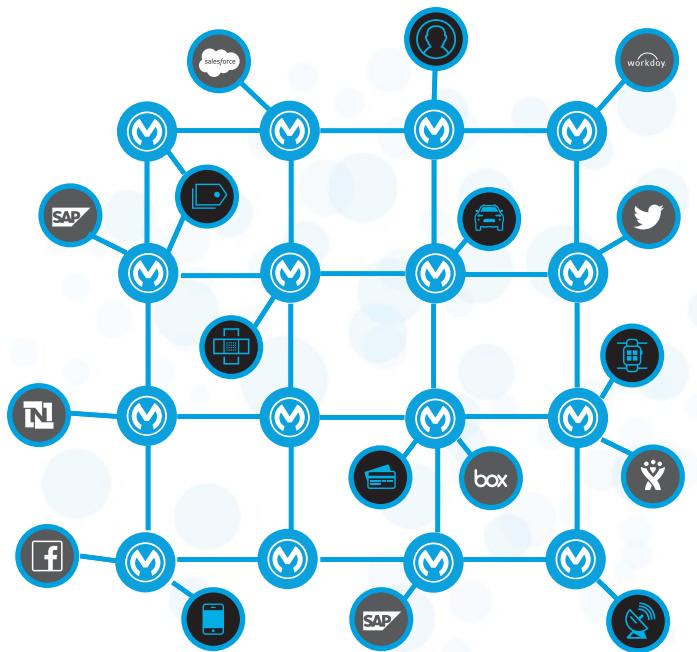
32. In the left-side navigation, select My Personal Information > Reset My Security Token.

33. If you did not already request a security token, click the Reset Security Token button.

Note: A security token will be sent to your email in a few minutes. You will need this token to make API calls to Salesforce from your Mule applications.

A screenshot of the "Reset My Security Token" page. The left sidebar shows "My Personal Information" with sub-options like "Advanced User Details", "Approver Settings", and "Change My Password". The main content area has a "Reset Security Token" button. A note explains that a security token is needed for untrusted IP addresses and APIs. A warning message states that after reset, the old token can't be used in API applications and desktop clients. A "Reset Security Token" button is at the bottom.

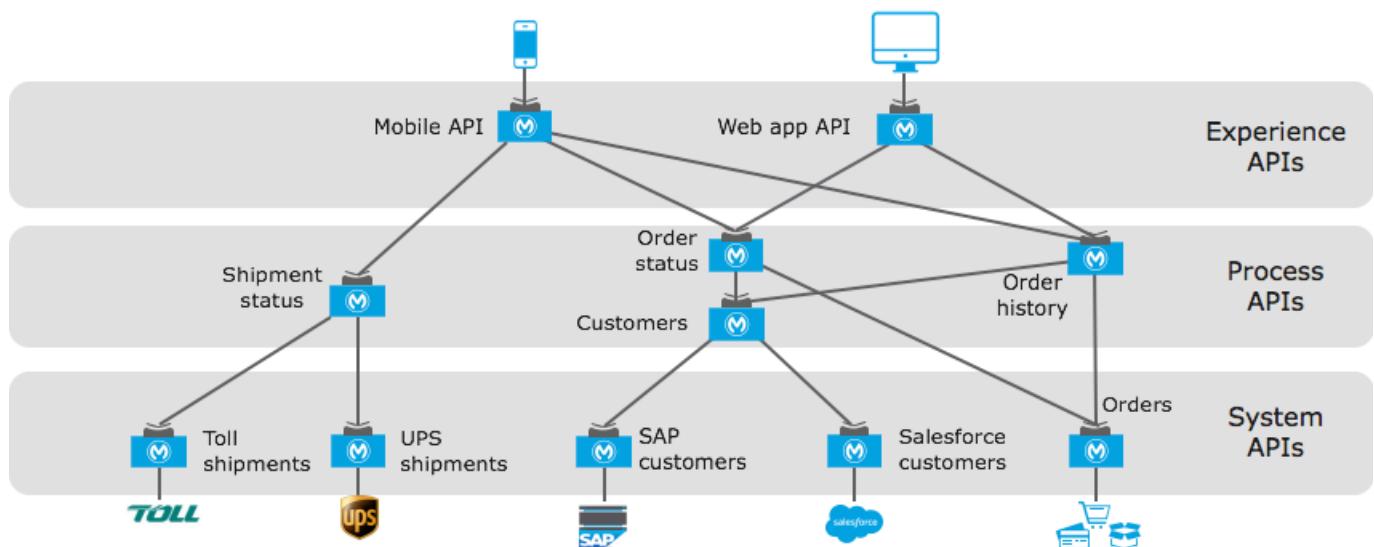
PART 1: Building Application Networks with Anypoint Platform



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

- Describe and explain the benefits of application networks & API-led connectivity.
- Use Anypoint Platform as a central repository for the discovery and reuse of assets.
- Use Anypoint Platform to build applications to consume assets and connect systems.
- Use Anypoint Platform to take an API through its complete development lifecycle.

Module 1: Introducing Application Networks and API-Led Connectivity



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

- Explain what an application network is and its benefits.
- Describe how to build an application network using API-led connectivity.
- Explain what web services and APIs are.
- Make calls to secure and unsecured APIs.

Walkthrough 1-1: Explore an API directory and an API portal

In this walkthrough, you locate and explore documentation about APIs. You will:

- Browse the ProgrammableWeb API directory.
- Explore the API reference for an API (like Twitter).
- Explore the API portal for an API to be used in the course.

The image shows two web pages side-by-side. The left page is the ProgrammableWeb API directory, featuring a search bar and a list of APIs. The right page is the MuleSoft API portal for the American Flights API, showing detailed documentation for the /flights/{ID} endpoint, including code examples and URI parameters.

Browse the ProgrammableWeb API directory

1. In a web browser, navigate to <http://www.programmableweb.com/>.
2. Click the API directory link.

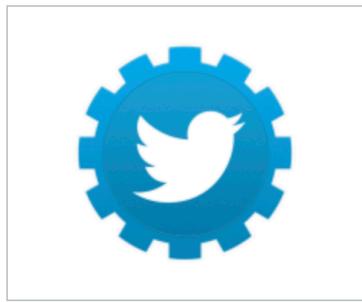
The image shows the ProgrammableWeb API directory homepage. It features a large green Android robot icon holding a smartphone. The phone screen displays a biometric login interface. Below the robot are several geometric shapes (hexagons and triangles) on a grid background. The top navigation bar includes links for API DIRECTORY and API NEWS. A search bar at the top right says "Search over 21,596 APIs and much more".

3. Browse the list of popular APIs.

Filter APIs			
By Category			<input type="checkbox"/> Include Deprecated APIs
API Name	Description	Category	Submitted
Google Maps	[This API is no longer available. Google Maps' services have been split into multiple APIs, including the Static Maps API ,	Mapping	12.05.2005
Twitter	[This API is no longer available. It has been split into multiple APIs, including the Twitter Ads API , Twitter Search Tweets...	Social	12.08.2006
YouTube	The Data API allows users to integrate their program with YouTube and allow it to perform many of the operations available on the website. It provides the capability to search for videos, retrieve...	Video	02.08.2006
Flickr	The Flickr API can be used to retrieve photos from the Flickr photo sharing service using a variety of feeds - public photos and videos, favorites, friends, group pools, discussions, and more. The...	Photos	09.04.2005
Facebook	[This API is no longer available. Its functions have been split among the following APIs: Facebook Ads ,	Social	08.16.2006
Amazon Product	What was formerly the ECS - eCommerce Service - has	eCommerce	12.02.2005

Explore the API reference for the Twitter API

4. Click the link for the Twitter API (or some other) API.
5. In the Specs section, click the API Portal / Home Page link.



Twitter API

[Social](#) [Blogging](#)

[This API is no longer available. It has been split into multiple APIs, including the [Twitter Ads API](#), [Twitter Search Tweets API](#), and [Twitter Direct Message API](#).]

This profile is maintained for historical, research, and reference purposes only.]

The Twitter micro-blogging service includes two RESTful APIs. The Twitter REST API methods allow developers to access core Twitter data. This includes update timelines, status data, and user information. The Search API methods give developers methods to interact with Twitter Search and trends data. The API presently supports the following data formats: XML, JSON, and the RSS and Atom syndication formats, with some methods only accepting a subset of these formats.

Summary	SDKs (73)	Articles (273)	How To (11)	Sample Source Code (25)	Libraries (38)	Developers (822)	Followers (2033)	Changelog (251)
-------------------------	---------------------------	--------------------------------	-----------------------------	---	--------------------------------	----------------------------------	----------------------------------	---------------------------------

- In the new browser tab that opens, click Tweets in the left-side navigation.
- In the left-side navigation, click Post, retrieve and engage with Tweets.

The screenshot shows the Twitter Developer API documentation. At the top, there's a purple header with links for 'Developer', 'Use cases', 'Products', 'Docs', 'More', 'Apply', a search icon, and 'Sign In'. Below the header, a search bar says 'Search all documentation...'. The main title 'Post, retrieve and engage with Tweets' is in large bold letters. Underneath it, there's a 'Basics' section, followed by 'Accounts and users' and 'Tweets'. The 'Tweets' section is expanded, showing 'Post, retrieve and engage with Tweets' as a sub-section. It lists several endpoints: 'Get Tweet timelines', 'Curate a collection of Tweets', 'Optimize Tweets with Cards', 'Search Tweets', 'Filter realtime Tweets', 'Sample realtime Tweets', and 'Get batch historical Tweets'. To the right, under 'Tweets', there's a list of POST methods: statuses/update, statuses/destroy/:id, and statuses/show/:id. Under 'Retweets', there's a list of GET methods: statuses/retweet/:id, statuses/unretweet/:id, statuses/retweets/:id, and statuses/retweets/_of_me. Under 'Likes (formerly favorites)', there's a list of POST and GET methods: favorites/create/:id, favorites/destroy/:id, and favorites/list.

- Browse the list of requests you can make to the API.
- Select the API Reference tab beneath the title of the page.
- Review the information for POST statuses/update including parameters, example request, and example response.

The screenshot continues from the previous one, showing the 'Example Request' section. It includes a code block for cURL:

```
curl -XPOST
--url 'https://api.twitter.com/1.1/statuses/update.json?status=hello'
--header 'Authorization: OAuth
oauth_consumer_key="oauth_customer_key",
oauth_nonce="generated_oauth_nonce",
oauth_signature="generated_oauth_signature",
oauth_signature_method="HMAC-SHA1",
oauth_timestamp="generated_timestamp",
oauth_token="oauth_token",
oauth_version="1.0"'
```

You may want to change the status from 'hello' to something different.

You can also use any other OAuth helper library you'd like such as twurl.

```
$ twurl -d 'status=Test tweet using the POST statuses/update endpoint' /1.1/statuses/update.json
```

Example Response

After you post successfully you should get back something that looks like this:

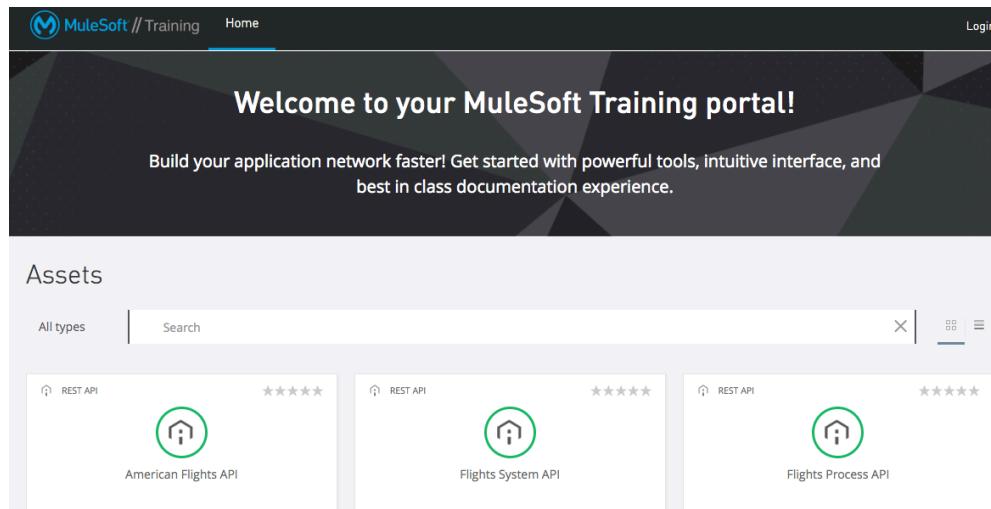
```
{
  "created_at": "Fri Jan 25 22:04:26 +0000 2019",
  "id": 1088920554520961000,
  "text": "Test tweet using the POST statuses/update endpoint"}
```

- Close the browser tab.

Explore an API portal for an API to be used in the course

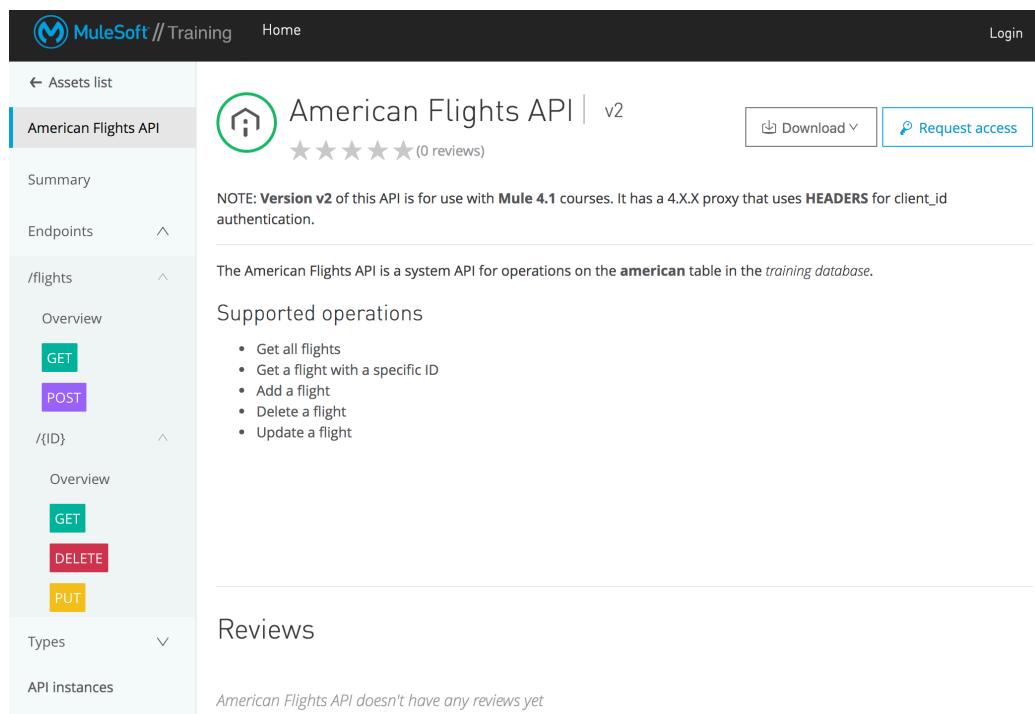
12. Return to or open the course snippets.txt file.
13. Copy the URL for the MuleSoft Training API portal.
14. Return to a browser window and navigate to that URL:

<https://anypoint.mulesoft.com/exchange/portals/muletraining/>.



The screenshot shows the MuleSoft Training portal's homepage. At the top, there's a navigation bar with the MuleSoft logo, a "Home" link, and a "Login" button. The main heading is "Welcome to your MuleSoft Training portal!". Below it, a sub-headline reads: "Build your application network faster! Get started with powerful tools, intuitive interface, and best in class documentation experience." The central area is titled "Assets" and contains a search bar. Three API cards are displayed: "American Flights API" (4 stars), "Flights System API" (4 stars), and "Flights Process API" (4 stars). Each card has a small icon of a house with a gear inside.

15. Click the American Flights API.
16. Click /flights in the API Summary on the left side.
17. Click /{ID} in the API Summary.
18. Browse the resources that are available for the API.



The screenshot shows the detailed view of the "American Flights API". On the left, a sidebar lists "Assets list", "American Flights API", "Summary", "Endpoints", and "flights". Under "Endpoints", "flights" is expanded, showing "Overview" with "GET" and "POST" methods, and "/{ID}" with "Overview", "GET", "DELETE", and "PUT" methods. The main content area shows the API title "American Flights API | v2" with a 0-star rating and 0 reviews. A note states: "NOTE: Version v2 of this API is for use with Mule 4.1 courses. It has a 4.XX proxy that uses HEADERS for client_id authentication." Below this, a summary says: "The American Flights API is a system API for operations on the **american** table in the *training* database." The "Supported operations" section lists: "Get all flights", "Get a flight with a specific ID", "Add a flight", "Delete a flight", and "Update a flight". The "Reviews" section notes: "American Flights API doesn't have any reviews yet".

19. Select the GET method for /flights.
20. Review the information about the GET method; you should see there is an optional query parameter called destination.

The screenshot shows the MuleSoft Anypoint Platform interface. On the left, the navigation sidebar lists 'Assets list', 'American Flights API', 'Summary', 'Endpoints', and 'Overview'. Under 'Endpoints', 'GET /flights' is selected. The main content area displays the 'American Flights API | v2' page. At the top, there is a star rating of 5 stars with '(0 reviews)'. Below this, a button for 'GET /flights' is shown. The 'Code examples' section has a 'Show' button. The 'Query parameters' section includes a 'destination' field, which is a String Enum with values SFO, LAX, and CLE. The 'Headers' section has a 'Hide' button. Below these, 'client_id' and 'client_secret' fields are listed as required strings. On the right side, there are buttons for 'Download', 'Request access', 'View code', and 'Mocking Service'. The 'Mocking Service' tab is selected, showing the URL <https://anypoint.mulesoft.com/mockng/api/v1/sources/exchange/assets/fd604b43-365c-42e8-810f-733a2b7f411f/american-flights-api/2.0.1/m/flights>. The 'Parameters' tab is active, showing the 'destination' parameter with a note to 'Fill in required parameters'. A 'Send' button is also present.

21. Scroll down and review the Headers and Response sections.

The screenshot shows the 'Headers' and 'Response' sections of the API documentation. The 'Headers' section contains two required string parameters: 'client_id' and 'client_secret'. The 'Response' section shows a 200 status code. Under 'Body', it specifies a media type of 'application/json'. The 'Examples' section provides a JSON example:

```

value
[{"ID": 1, "code": "ER38sd", "departureDate": "2017/07/26", "destination": "SFO", "emptySeats": 0, "origin": "CLE", "plane": {"totalSeats": 150, "type": "Boeing 737"}, "price": 400},
]
  
```

22. In the left-side navigation, select the DELETE method.

23. Locate information about the required ID URI parameter.

The screenshot shows the MuleSoft Anypoint Platform interface. On the left, the navigation sidebar lists 'Assets list', 'American Flights API' (selected), 'Summary', 'Endpoints' (expanded), '/flights' (expanded), and 'Types'. Under 'Endpoints', there are sections for 'Overview' (with 'GET' and 'POST' methods), '/{ID}' (with 'Overview', 'GET', 'DELETE', and 'PUT' methods), and 'API instances'. The main content area displays the 'American Flights API | v2' documentation. It features a green circular icon with a house symbol, a star rating of 5 stars (0 reviews), and a 'DELETE /flights/{ID}' endpoint. Below this are sections for 'Code examples', 'URI parameters' (with 'ID' and 'client_id' parameters), 'Headers', and 'Response' (status code 200). On the right, there are download, request access, and view code options, along with a URL: <https://anypoint.mulesoft.com/mockng/api/v1/sources/exchange/assets/fd604b43-365c-42e8-810f-733a2b7f411f/american-flights-api/2.0.1/m/flights/{ID}>. The 'Parameters' tab is selected, showing the 'ID*' parameter. A 'Send' button and a 'Fill in required parameters' link are also present.

24. Review the information for the other resources.

Walkthrough 1-2: Make calls to an API

In this walkthrough, you make calls to a RESTful API. You will:

- Use Advanced REST Client to make calls to an unsecured API (an implementation).
- Make GET, DELETE, POST, and PUT calls.
- Use Advanced REST Client to make calls to a secured API (an API proxy).
- Use the API console in an API portal to make calls to a managed API using a mocking service.
- Use the API console to make calls to an API proxy endpoint.

The screenshot shows the MuleSoft API Mocking Service interface. On the left, a sidebar lists recent API calls with icons indicating the method (GET, POST, PUT, DELETE) and timestamp. The main area displays the 'American Flights API | v2' endpoint. It includes a summary, endpoints (flights), and code examples for GET /flights. Below this, there are sections for destination (with enum values SFO, LAX, GLE), headers, and types (Client_Id, Client_Secret). A 'Mocking Service' tab is selected, showing the URL <https://mngmnt.mulesoft.com/mock/api/v1/mock/733a207f41famerican-flights-api/2.0.1/m/flight>. Parameters and Headers sections are also visible.

Use Advanced REST Client to make GET requests to retrieve data

1. Return to or open Advanced REST Client.
2. Make sure the method is set to GET.
3. Return to the course snippets.txt file.
4. Copy the URL for the American Flights web service:
<http://training4-american-ws.cloudhub.io/api/flights>.

Note: This is the URL for the API implementation, not the managed API proxy. The -ws stands for web service.

5. Return to Advanced REST Client and paste the URL in the text box that says Request URL, replacing any existing content.

The screenshot shows the Advanced REST Client interface. The 'Request URL' field contains the URL <http://training4-american-ws.cloudhub.io/api/flights>. Other fields visible include 'Method' (set to GET), 'Parameters' (dropdown menu), and a 'SEND' button.

6. Click the Send button; you should get a response.
7. Locate the return HTTP status code of 200.
8. Review the response body containing flights to SFO, LAX, and CLE.

200 OK 1099.00 ms DETAILS ▾

□ ↴ <> ⌂

```
[Array[11]
-0: {
  "ID": 1,
  "code": "rree0001",
  "price": 541,
  "departureDate": "2016-01-20T00:00:00",
  "origin": "MUA",
  "destination": "LAX",
  "emptySeats": 0,
  "plane": {
    "type": "Boeing 787",
    "totalSeats": 200
  }
},
-1: {
  "ID": 2,
  "code": "eefd0123",
```

9. Click the Toggle raw response view (or Toggle source view) button.

200 OK 1283.27 ms DETAILS ▾

□ ↴ <> ⌂ ⌂

```
[
{
  "ID": 1,
  "code": "rree0001",
  "price": 541,
  "departureDate": "2016-01-20T00:00:00",
  "origin": "MUA",
  "destination": "LAX",
  "emptySeats": 0,
  "plane": {
    "type": "Boeing 787",
    "totalSeats": 200
  }
},
```

10. Click the arrow to the right of the URL.
11. In the detailed editor area that appears, click the Add button.

12. Set the Parameter name to destination and the Parameter value to CLE.

The screenshot shows a REST client interface with the following details:

- Host: <http://training4-american-ws.cloudhub.io>
- Path: /api/flights
- Method: GET
- Query parameters:
 - Parameter name: destination
 - Parameter value: CLE
- Buttons: ADD, SEND, X

13. Click the Send button; you should get just flights to CLE returned.

14. Click the X next to the parameter to delete it.

15. Click the arrow to the right of the URL to collapse the detailed editor area.

16. Change the request URL to use a URI parameter to retrieve the flight with an ID of 3:

[http://training4-american-ws.cloudhub.io/api/flights/3.](http://training4-american-ws.cloudhub.io/api/flights/3)

17. Click the Send button; you should see only the flight with that ID returned.

The screenshot shows a REST client interface with the following details:

- Status: 200 OK
- Time: 1054.80 ms
- Buttons: Copy, Share, Refresh, Details, Stop
- Response body (JSON):

```
[Array[1]
  -0: {
    "ID": 3,
    "code": "ffee0192",
    "price": 300,
    "departureDate": "2016-01-20T00:00:00",
    "origin": "MUA",
    "destination": "LAX",
    "emptySeats": 0,
    "plane": {
      "type": "Boeing 777",
      "totalSeats": 300
    }
  }
],
```

Make DELETE requests to delete data

18. Change the method to DELETE.

19. Click the Send button; you should see a 200 response with the message Flight deleted (but not really).

Note: The database is not actually modified so that its data integrity can be retained for class.

Method Request URL

DELETE http://training4-american-ws.cloudhub.io/api/flights/3

SEND :
▼

Parameters ▼

200 OK 302.50 ms DETAILS ▼

□ ↻ ⌂

```
{ "message": "Flight deleted (but not really)" }
```

20. Remove the URI parameter from the request: <http://training4-american-ws.cloudhub.io/api/flights>.

21. Click the Send button; you should get a 405 response with the message Method not allowed.

Method Request URL

DELETE http://training4-american-ws.cloudhub.io/api/flights

SEND :
▼

Parameters ▼

405 Method Not Allowed 539.20 ms DETAILS ▼

□ ↻ ⌂

```
{ "message": "Method not allowed" }
```

Make a POST request to add data

22. Change the method to POST.

23. Click the Send button; you should get a 415 response with the message Unsupported media type.

Method Request URL

POST http://training4-american-ws.cloudhub.io/api/flights

SEND :
▼

Parameters ▼

415 Unsupported Media Type 63.50 ms DETAILS ▼

□ ↻ ⌂

```
{ "message": "Unsupported media type" }
```

24. Click the arrow next to Parameters beneath the request URL.
25. Click Add Header.
26. Click in the Header name field, type Co, and then select Content-Type.
27. Click in the Parameter value field and then select application/json.

Parameters ^

Headers	Body	Authorization	Actions
<input type="checkbox"/> <>			? -
Header name <input checked="" type="checkbox"/> Content-Type	Parameter value application/json		

28. Select the Body tab.
29. Return to the course snippets.txt file and copy the value for American Flights API post body.
30. Return to Advanced REST Client and paste the code in the body text area.

Method Request URL

POST <http://training4-american-ws.cloudhub.io/api/flights> [▼](#) [SEND](#) [⋮](#)

Parameters ^

Headers	Body	Authorization	Actions
<input type="checkbox"/> Body content type <input checked="" type="checkbox"/> application/json			
FORMAT JSON MINIFY JSON			

```
{
  "code": "GQ574",
  "price": 399,
  "departureDate": "2016/12/20",
  "origin": "ORD",
  "destination": "SFO",
  "emptySeats": 200,
  "plane": {"type": "Boeing 747", "totalSeats": 400}
}
```

31. Click the Send button; you should see a 201 Created response with the message Flight added (but not really).

201 Created 478.20 ms [DETAILS ▾](#)

<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/> <>	<input type="checkbox"/>

```
{
  "message": "Flight added (but not really)"
}
```

32. Return to the request body and remove the plane field and value from the request body.
33. Remove the comma after the emptySeats key/value pair.

Headers	Body	Authorization	Actions
	<p>Body content type application/json</p> <p>FORMAT JSON MINIFY JSON</p> <pre>{ "code": "GQ574", "price": 399, "departureDate": "2016/12/20", "origin": "ORD", "destination": "SFO", "emptySeats": 200 }</pre>		

34. Send the request; the message should still post successfully.
35. In the request body, remove the emptySeats key/value pair.
36. Delete the comma after the destination key/value pair.

Headers	Body	Authorization	Actions
	<p>Body content type application/json</p> <p>FORMAT JSON MINIFY JSON</p> <pre>{ "code": "GQ574", "price": 399, "departureDate": "2016/12/20", "origin": "ORD", "destination": "SFO" }</pre>		

37. Send the request; you should see a 400 Bad Request response with the message Bad request.

400 Bad Request	681.80 ms	DETAILS ▾
<p>Body content type application/json</p> <p>FORMAT JSON MINIFY JSON</p> <pre>{ "message": "Bad request" }</pre>		

Make a PUT request to update data

38. Change the method to PUT.
39. Add a flight ID of 3 to the URL.
40. Click the Send button; you should get a 400 Bad Request.

A screenshot of a browser developer tools network tab. The status bar at the top shows "400 Bad Request" and "303.20 ms". To the right is a "DETAILS" button with a dropdown arrow. Below the status bar are standard browser control icons: a refresh square, a back/forward triangle, a copy/cut/paste clipboard, and a search/magnifying glass. The main content area displays a JSON response:

```
{  "message": "Bad request"}
```

41. In the request body field, press Cmd+Z or Ctrl+Z until the emptySeats field is added back.
42. Send the request; you should get a 200 OK response with the message Flight updated (but not really).

A screenshot of a browser developer tools network tab. The status bar at the top shows "200 OK" and "574.90 ms". To the right is a "DETAILS" button with a dropdown arrow. Below the status bar are standard browser control icons. The main content area displays a JSON response:

```
{  "message": "Flight updated (but not really)"}
```

Make a request to a secured API

43. Remove the Content-Type header by clicking its Remove this header button.
44. Change the method to GET.
45. Change the request URL to <http://training4-american-api.cloudhub.io/flights/3>.

Note: The -ws in the URL has been changed to -api and the /api removed.

46. Click the Send button; you should get a 401 Unauthorized response with the message Invalid client id or secret.

A screenshot of a browser developer tools network tab. The status bar at the top shows "401 Unauthorized" and "290.20 ms". To the right is a "DETAILS" button with a dropdown arrow. Below the status bar are standard browser control icons. The main content area displays a JSON response:

```
{  "error": "Invalid client id or secret"}
```

47. Return to the course snippets.txt file and copy the value for the American Flights API client_id.
48. Return to Advanced REST Client and add a header called client_id.
49. Set client_id to the value you copied from the snippets.txt file.
50. Return to the course snippets.txt file and copy the value for the American Flights API client_secret.
51. Return to Advanced REST Client and add a second header called client_secret.
52. Set client_secret to the value you copied from the snippets.txt file.

Parameters ^

Headers	Authorization	Actions
<input type="checkbox"/> Content-Type <input checked="" type="checkbox"/> client_id <input checked="" type="checkbox"/> client_secret	Header name Parameter value application/json Header name Parameter value d1374b15c6864c3682ddbed2a247a826 Header name Parameter value 4a87fe7e2e43488c927372AEF981F066	? (edit) ? (edit) ? (edit)

53. Click the Send button; you should get data for flight 3 again.

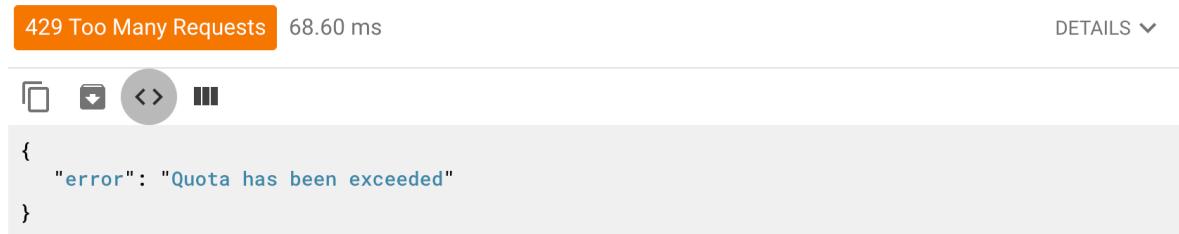
200 OK 881.60 ms DETAILS ▾

Headers
<input type="checkbox"/> Content-Type <input checked="" type="checkbox"/> client_id <input checked="" type="checkbox"/> client_secret

```
[Array[1]
-0: {
  "ID": 3,
  "code": "ffee0192",
  "price": 300,
  "departureDate": "2016-01-20T00:00:00",
  "origin": "MUA",
  "destination": "LAX",
  "emptySeats": 0,
  "plane": {
    "type": "Boeing 777",
    "totalSeats": 300
  }
},
]
```

54. Click the Send button several more times; you should get a 429 Too Many Requests response with the message Quota has been exceeded.

Note: For the self-study training class, the API service level agreement (SLA) for the application with your client ID and secret has been set to allow three API calls per minute while, for the instructor-led class, the SLA allows for a higher number to accommodate shared use.



A screenshot of a browser developer tools network tab. The status bar at the top shows "429 Too Many Requests" and "68.60 ms". To the right is a "DETAILS" button with a dropdown arrow. Below the status bar are icons for copy, refresh, and search. The main content area displays the JSON response body:

```
{  
  "error": "Quota has been exceeded"  
}
```

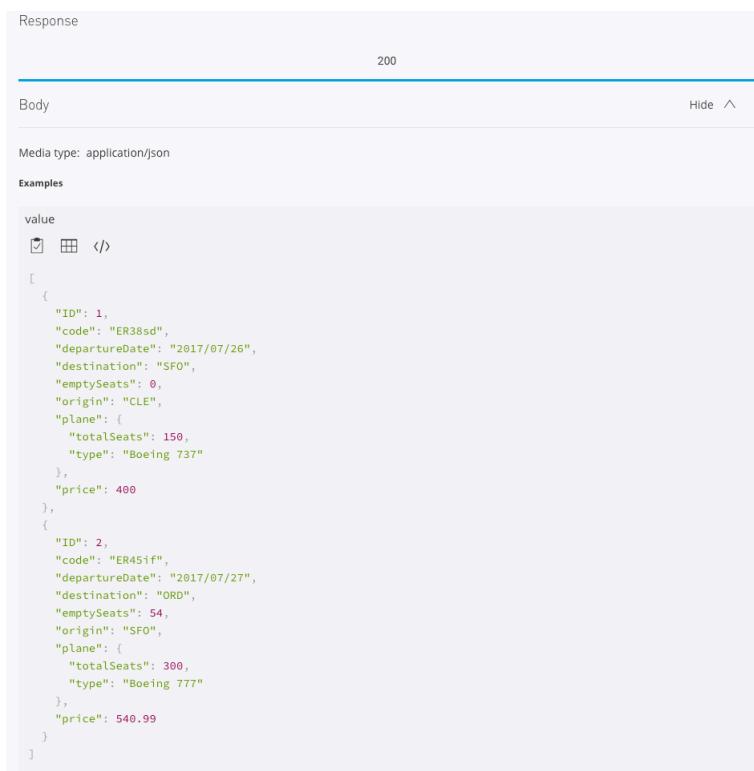
Use the API console in the API portal to make requests to the API using a mocking service

55. Return to the browser window with the American Flights API portal at
<https://anypoint.mulesoft.com/exchange/portals/muletraining>.

56. In the left-side navigation click the GET method for /flights.

57. Review the Headers and Response sections.

58. In the Response section, look at Examples.



A screenshot of the API console's Response section. At the top, it says "Response" and "200". Below that is a "Body" section with "Hide" and "▲" buttons. It shows the media type as "application/json". Under "Examples", there is a "value" section with a "Copy" icon and a "Raw" link. The JSON example shows two flight records:

```
[  
  {  
    "ID": 1,  
    "code": "ER38sd",  
    "departureDate": "2017/07/26",  
    "destination": "SFO",  
    "emptySeats": 0,  
    "origin": "CLE",  
    "plane": {  
      "totalSeats": 150,  
      "type": "Boeing 737"  
    },  
    "price": 400  
  },  
  {  
    "ID": 2,  
    "code": "ER45if",  
    "departureDate": "2017/07/27",  
    "destination": "ORD",  
    "emptySeats": 54,  
    "origin": "SFO",  
    "plane": {  
      "totalSeats": 300,  
      "type": "Boeing 777"  
    },  
    "price": 540.99  
  }]
```

59. In the API console located on the right side of the page, make sure the Mocking Service endpoint is selected.
60. Look at the endpoint URL that is displayed.

The screenshot shows the MuleSoft API Console interface. At the top, there are four buttons: 'Download' with a download icon, 'Request access' with a key icon, 'View code' with a code editor icon, and another 'View code' button. Below these is a dropdown menu labeled 'Mocking Service'. Underneath the dropdown is the endpoint URL: <https://anypoint.mulesoft.com/mockng/api/v1/sources/exch>.

61. Select the Show optional parameters checkbox.
62. Select LAX in the destination drop-down menu.
63. Select the Headers tab.
64. Enter any values for client_id and client_secret.
65. Click Send; you should get the example flights that are to SFO and ORD.

The screenshot shows the API response details. At the top, it says '200 OK' and '2411.93 ms'. Below that is a 'Details' dropdown. The main content area displays an array of two flight objects:

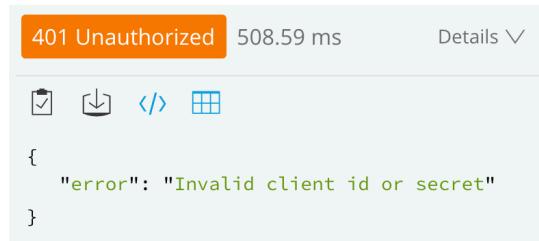
```
[Array[2]
-0: {
  "ID": 1,
  "code": "ER38sd",
  "price": 400,
  "departureDate": "2017/07/26",
  "origin": "CLE",
  "destination": "SFO",
  "emptySeats": 0,
  "plane": {
    "type": "Boeing 737",
    "totalSeats": 150
  }
}
-1: {
  "ID": 2,
  "code": "ER45if",
  "price": 540.99,
  "departureDate": "2017/07/27",
  "origin": "SFO",
  "destination": "ORD",
}
```

Make requests to the API using an API proxy endpoint

66. At the top of the API console, change the endpoint to Production – Rate limiting SLA based policy.
67. Look at the endpoint URL that is displayed.

The screenshot shows the MuleSoft API Console interface. At the top, there are four buttons: 'Download' with a download icon, 'Request access' with a key icon, 'View code' with a code editor icon, and another 'View code' button. Below these is a dropdown menu labeled 'Production - Rate limiting SLA based policy'. Underneath the dropdown is the endpoint URL: <http://training4-american-api.us-e1.cloudhub.io/flights?destination=LAX>.

68. Click Send; you should get a 401 Unauthorized response.



```
{  
  "error": "Invalid client id or secret"  
}
```

69. Copy and paste the client_id and client_secret values from the course snippets.txt file.

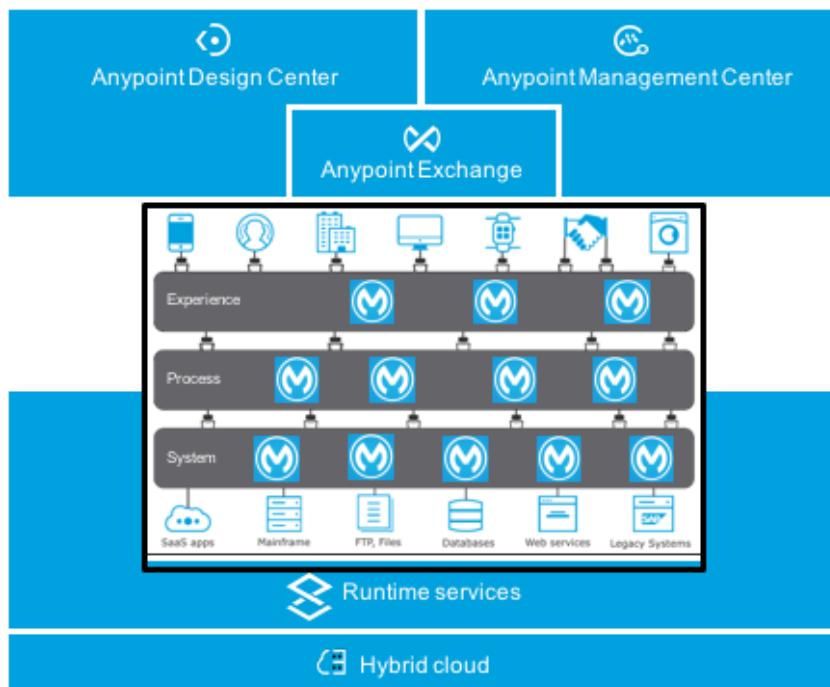
Parameters	Headers
<input checked="" type="checkbox"/> </>	
client_id*	
d1374b15c6864c3682ddbed2a247a826	
client_secret*	
4a87fe7e2e43488c927372AEF981F066	
+ Add header	

70. Click Send; you should get a 200 OK response with only flights to LAX.



```
[Array[3]  
-0: {  
  "ID": 1,  
  "code": "rree0001",  
  "price": 541,  
  "departureDate": "2016-01-20T00:00:00",  
  "origin": "MUA",  
  "destination": "LAX",  
  "emptySeats": 0,  
  "plane": {  
    "type": "Boeing 787",  
    "totalSeats": 200  
  }  
},  
-1: {  
  "ID": 3,  
  "code": "ffee0192",  
  "price": 300,  
  "departureDate": "2016-01-20T00:00:00",  
  "origin": "MUA",  
  "destination": "LAX",  
  "emptySeats": 0,  
  "plane": {  
    "type": "Boeing 787",  
    "totalSeats": 200  
  }  
},  
-2: {  
  "ID": 2,  
  "code": "ffff0000",  
  "price": 400,  
  "departureDate": "2016-01-20T00:00:00",  
  "origin": "MUA",  
  "destination": "LAX",  
  "emptySeats": 0,  
  "plane": {  
    "type": "Boeing 787",  
    "totalSeats": 200  
  }  
}]
```

Module 2: Introducing Anypoint Platform



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

- Describe the benefits of Anypoint Platform and MuleSoft's approach to be successful with it.
- Describe the role of each component in building application networks.
- Navigate Anypoint Platform.
- Locate APIs and other assets needed to build integrations and APIs in Anypoint Exchange.
- Build basic integrations to connect systems using Flow Designer.

Walkthrough 2-1: Explore Anypoint Platform and Anypoint Exchange

In this walkthrough, you get familiar Anypoint Platform. You will:

- Explore Anypoint Platform.
- Browse Anypoint Exchange.
- Review an API portal for a REST API in Exchange.
- Discover and make calls to the Training: American Flights API in the public Exchange.

The screenshot shows the Anypoint Exchange interface. The left sidebar has a 'REST APIs' tab selected. The main area displays a grid of REST API assets. One asset, 'Training: American Flights API', is highlighted with a green circle and a star rating of 5. Other visible assets include 'Appian API', 'PayPal Payments API', and several others with lower ratings. A search bar and a 'New asset' button are at the top right.

Return to Anypoint Platform

1. Return to Anypoint Platform at <https://anypoint.mulesoft.com> (not the public API portal you used last module!) in a web browser.

Note: If you closed the browser window or logged out, return to <https://anypoint.mulesoft.com> and log in.

2. Click the menu button located in the upper-left in the main menu bar.

3. In the menu that appears, select Anypoint Platform; this will return you to the home page.

Note: This will be called the main menu from now on.

The screenshot shows the Anypoint Platform interface. On the left, a vertical navigation bar lists several options: Design Center, Exchange, Management Center, Access Management, API Manager, Runtime Manager, Data Gateway, and MQ. The 'Management Center' option is currently selected. The main content area is divided into two main sections: 'Design Center' on the left and 'Management Center' on the right. The 'Design Center' section includes a 'Start designing' button and a 'Quick Start' section. The 'Management Center' section includes links to 'Access Management', 'API Manager', 'Runtime Manager', and 'Data Gateway'. A 'Recently Visited' sidebar on the left shows a link to 'Training: American Flights API'.

Explore Anypoint Platform

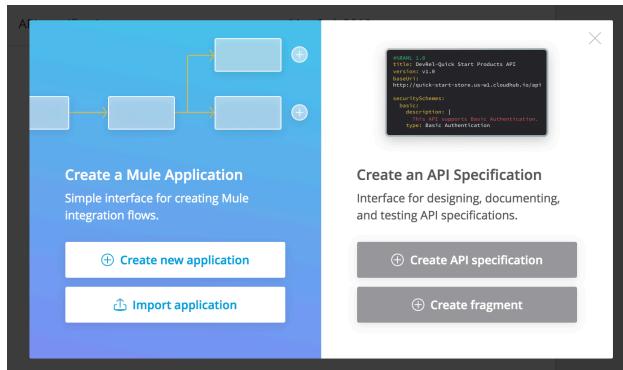
4. In the main menu, select Access Management.
5. In the left-side navigation, select Users.
6. In the left-side navigation, select Environments.

The screenshot shows the 'Access Management' interface. On the left, a sidebar provides navigation for 'ACCESS MANAGEMENT' (Organization, Users, Roles, Environments), 'SETTINGS' (Runtime Manager, Flow Designer), and 'SUBSCRIPTION' (Runtime Manager, MQ, Object Store). The main content area is titled 'Environments' and shows a table of existing environments:

Name	Type
Design	Design
Sandbox	Sandbox

A blue 'Add environment' button is located at the top of the table.

- In the main menu, select Design Center.
- Click the Create button and look at the options in the popup.



- Close the popup.
- In the main menu, select Runtime Manager.
- If you get a Choose Environment page, select Design.

The screenshot shows the Runtime Manager interface. At the top, there is a navigation bar with icons for Training, Help, and MM. Below it, a sidebar on the left has tabs for DESIGN, APPLICATIONS, SERVERS, ALERTS, VPCs, VPNs, RUNTIME FABRICS, and LOAD BALANCERS. The DESIGN tab is selected. The main area shows a placeholder icon and the message "There are no applications to show".

- In the main menu, select API Manager.

The screenshot shows the API Manager interface. At the top, there is a navigation bar with icons for Training, Help, and MM. Below it, a sidebar on the left has tabs for SANDBOX, API ADMINISTRATION, AUTOMATED POLICIES, CLIENT APPLICATIONS, CUSTOM POLICIES, and ANALYTICS. The SANDBOX tab is selected. The main area shows a message "No APIs to display. Get started by adding your first API." and a placeholder icon with the message "Select an API version to see more details".

Explore Anypoint Exchange

- In the main menu, select Exchange.

14. In the left-side navigation, select Provided by MuleSoft; you should see all the content in the public Exchange.

The screenshot shows the MuleSoft Exchange interface. The left sidebar has a dark background with white text. It lists several categories: All assets, Training (master), **Provided by MuleSoft**, My applications, Public portal ↗, and Settings. The main area has a light gray background with a header "Assets provided by MuleSoft". Below the header are two tabs: "All types" and "Search". On the right side of the header are "New asset" and other navigation icons. The main content area displays five asset cards arranged in two rows. Each card includes a small icon, the asset name, its rating (5 stars), and the organization it belongs to. The first row contains three cards: "SAP Connector - Mule 4" (MuleSoft Organization), "Salesforce Connector - Mule 4" (MuleSoft Organization), and "Amazon S3 Connector - Mule 4" (MuleSoft Organization). The second row contains two cards: "Custom" (Custom) and another "Custom" entry (Custom).

15. In the left-side navigation, select the name of your organization above Provided by MuleSoft (Training in the screenshots); you should now see only the content in your private Exchange, which is currently empty.

The screenshot shows the MuleSoft Exchange interface. The left sidebar has a dark background with white text. It lists several categories: All assets, **Training (master)**, Provided by MuleSoft, My applications, Public portal ↗, and Settings. The main area has a light gray background with a header "Training assets (master)". Below the header are two tabs: "All types" and "Search". On the right side of the header are "New asset" and other navigation icons. The main content area is currently empty, showing only the header and tabs.

16. In the left-side navigation, select Provided by MuleSoft.

17. In the types menu, select Connectors.

The screenshot shows the MuleSoft Exchange interface. On the left, there's a sidebar with navigation links: 'All assets', 'Training (master)', 'Provided by MuleSoft', 'My applications', 'Public portal ↗', and 'Settings'. The main area is titled 'Assets provided by MuleSoft' and has a sub-section 'Connectors'. A search bar is present. Below it, a message says 'Showing results for Connectors.' There are six connector cards displayed:

- SAP Connector - Mule 4 (MuleSoft Organization)
- Salesforce Connector - Mule 4 (MuleSoft Organization)
- Amazon S3 Connector - Mule 4 (MuleSoft Organization)
- (Placeholder card)
- (Placeholder card)
- (Placeholder card)

18. Select one of the connectors and review its information.

19. In the left-side navigation, click the Assets list link.

20. Search for the Salesforce Connector - Mule 4 connector and review its details.

Note: This Salesforce connector is used in the Development Fundamentals course.

The screenshot shows the details for the 'Salesforce Connector - Mule 4' connector. The left sidebar shows 'Assets list' and 'Salesforce Connector - Mule 4' is selected. The main content area displays the connector's icon, name, rating (4.5 stars from 6 reviews), and a 'Rate and review' button. Below this, a description states: 'MuleSoft's Salesforce Connector helps you to accelerate your [Salesforce integrations](#) across Sales Cloud, Service Cloud, Salesforce Platform, and Force.com. The connector gives you access to all Salesforce entities to enable automation of your business processes to help maximize your investments in services and solutions like enabling your sales teams, increasing revenue, and serving your customers better.' It also notes that this is a 'Connector' type asset from 'MuleSoft' organization, created by 'MuleSoft Organization' on May 3, 2019. The 'Associated Use Cases' section is partially visible. On the right, there are download and dependency snippets options, and a 'Versions' table:

Version	Runtime version
9.7.0	4.1.1
9.6.2	4.1.1

21. In the left-side navigation, click Assets list.

22. In the types menu, select Templates.

23. Remove salesforce from the search field and press Enter/Return.

Browse REST APIs in Anypoint Exchange

24. In the types menu, select REST APIs.

25. Browse the APIs.

The screenshot shows the MuleSoft Exchange interface. On the left, there's a sidebar with options like 'All assets', 'Training (master)', 'Provided by MuleSoft', 'My applications', 'Public portal', and 'Settings'. The main area is titled 'Assets provided by MuleSoft' and 'Showing results for REST APIs.' It displays five API cards:

- Appian API** (MuleSoft Organization)
- Training: American Flights API** (MuleSoft Organization)
- PayPal Payments API** (MuleSoft Organization)
- Flight API** (MuleSoft Organization)
- Flight Data API** (MuleSoft Organization)

Discover and review the API portal for the Training: American Flights API

26. Locate and click the Training: American Flights API.

A card for the 'Training: American Flights API' REST API. It has a green icon, a 5-star rating, and the title 'Training: American Flights API' from 'MuleSoft Organization'.

27. Review the API portal.

The screenshot shows the detailed view of the 'Training: American Flights API' page. The left sidebar lists sections like 'Assets list', 'Summary', 'Endpoints', 'Types', and 'API instances'. The main content area shows the API title 'Training: American Flights API | v2.0 Public', its description ('This example RAML 1.0 API is used in MuleSoft training courses.'), and its definition ('It defines the basic GET, POST, DELETE, and PUT operations for flights and uses data type and example fragments.') with a code snippet:

```
1  #RAML 1.0
2  version: v2.0
3  title: American Flights API
4
5  traits:
6    client-id-required:
7      headers:
8        client_id:
9          type: string
10       client_secret:
11         type: string
12
13  types:
14    AmericanFlight: !include exchange_modules/68ef9520-24e9-4cf2-b2f5-620025690913
15
16  /flights:
17    is [client-id-required]
18    get:
19      queryParameters:
20        destination:
21          required: false
22          enum:
23            - SFO
24            - LAX
25            - CLE
26
27  responses:
28    200:
```

The right side of the page shows details like 'Download', 'View code', 'Type: REST API', 'Organization: MuleSoft', 'Created by: MuleSoft Organization', 'Published on: May 2, 2019', 'Visibility: Public', and 'Asset versions for v2.0' (version 2.0.1 with 'Mocking Service' instance). There are also 'Tags' (training) and 'Dependencies' sections.

28. In the left-side navigation, expand and review the list of available resources.
29. Click the GET link for the /flights resource.
30. On the GET /flights page, review the information for the optional destination query parameter; you should see the API is similar to the one you explored in the public MuleSoft Training portal.

The screenshot shows the Exchange API console interface. The left sidebar has a tree view with 'Assets list' expanded, showing 'Training: American Flights API'. Under 'Endpoints', the '/flights' endpoint is selected, showing its details. The main panel displays the 'GET /flights' endpoint with a summary, code examples, and a 'Query parameters' section. The 'destination' parameter is defined as a String Enum with values SFO, LAX, and CLE. The right sidebar shows the 'Mocking Service' tab selected, displaying the URL <https://anypoint.mulesoft.com/mockng/api/v1/sources/exchange/assets/68ef9520-24e9-4cf2-b2f5-620025690913/training-american-flights-api/2.0.1/m/flights>.

Use the API console to make calls to the Training: American Flights API

31. In the API console, review the options for the instances you can test.

The screenshot shows the API console interface with the 'Mocking Service' tab selected. The left sidebar lists 'Mocking Service' and 'Rate limiting SLA based policy'. The main panel shows the selected 'Mocking Service' instance.

32. Select Mocking Service.

33. Select to Show the optional query parameters then select a destination in the drop-down menu.

The screenshot shows the 'Mocking Service' interface with the URL <https://anypoint.mulesoft.com/mockng/api/v1/sources/exchange/assets/68ef9520-24e9-4cf2-b2f5-620025690913/training-american-flights-api/2.0.1/m/flights>. The 'Parameters' tab is active. Under 'Query parameters', there is a checked checkbox 'Show optional parameters'. Below it, 'destination' is selected. A dropdown menu lists 'Select value', 'SFO', 'LAX', and 'CLE'. There is also a '+' button and a 'Selected' section.

34. Select the Headers tab.

35. Enter any values for client_id and client_secret.

36. Click Send; you should get the two example flights.

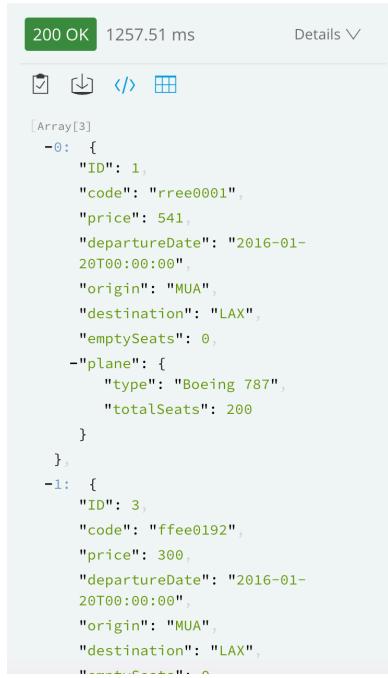
The screenshot shows the API response details. Status: 200 OK, Duration: 1036.12 ms, Details: ✓. The response body is a JSON array of two flight records:

```
[{"ID": 1, "code": "ER38sd", "price": 400, "departureDate": "2017/07/26", "origin": "CLE", "destination": "SFO", "emptySeats": 0, "plane": {"type": "Boeing 737", "totalSeats": 150}}, {"ID": 2, "code": "ER45if", "price": 540.99, "departureDate": "2017/07/27", "origin": "SFO", "destination": "ORD"}]
```

37. Change the API instance from Mocking Service to Rate limiting SLA based policy.

38. In the Headers tab, copy and paste the client_id and client_secret values from the course snippets.txt file

39. Click Send again; you should get results from the actual API implementation for the destination you selected.



200 OK | 1257.51 ms Details ▾

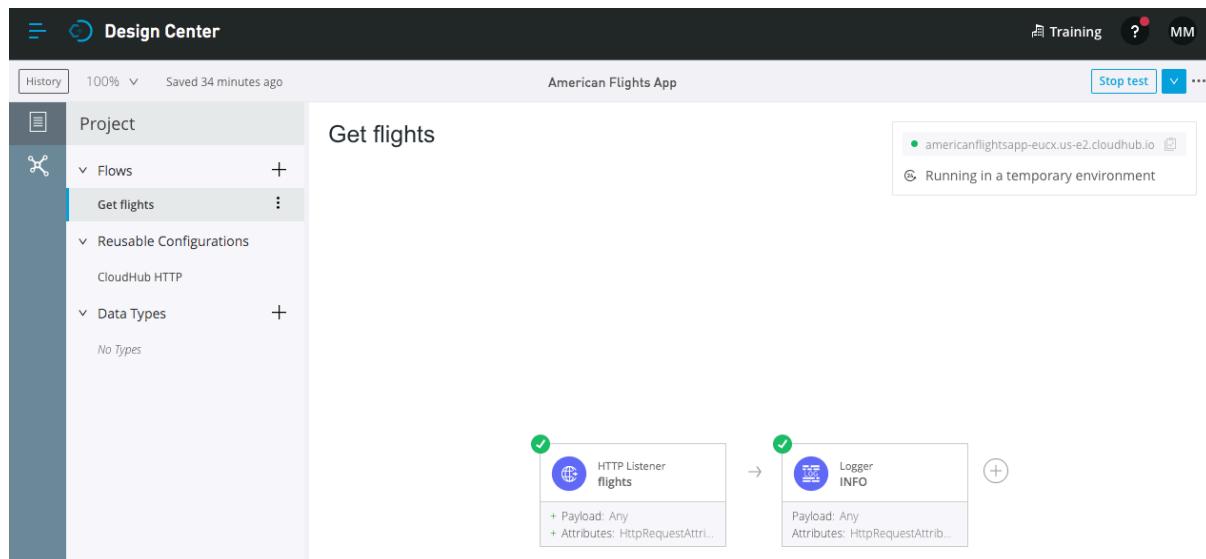
Array[3]

```
[{"ID": 1, "code": "rreee0001", "price": 541, "departureDate": "2016-01-20T00:00:00", "origin": "MUA", "destination": "LAX", "emptySeats": 0, "plane": {"type": "Boeing 787", "totalSeats": 200}}, {"ID": 3, "code": "ffee0192", "price": 300, "departureDate": "2016-01-20T00:00:00", "origin": "MUA", "destination": "LAX", "emptySeats": 0}], 0
```

Walkthrough 2-2: Create a Mule application with Flow Designer

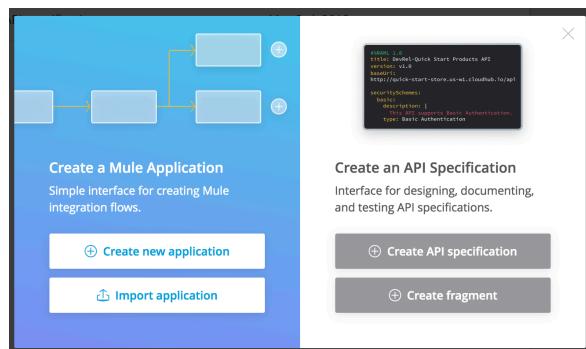
In this walkthrough, you build, run, and test a basic Mule application with Flow Designer. You will:

- Create a new Mule application project in Design Center.
- Create an HTTP trigger for a flow in the application.
- Add a Logger component.
- Run and test the application.
- View application information in Runtime Manager.



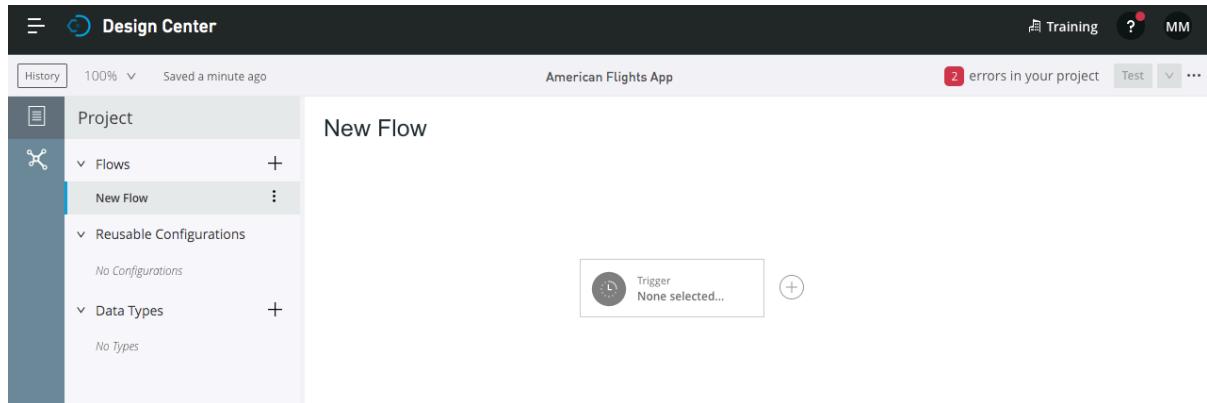
Create a Mule application project in Design Center

1. Return to Anypoint Platform.
2. In the main menu, select Design Center.
3. Click the Create button and select Create new application.

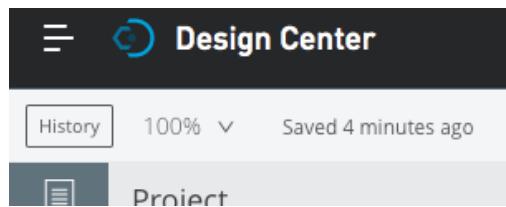


4. In the New Mule Application dialog box, set the project name to American Flights App and click Create.

5. In the Let's get started dialog box, select the Go straight to canvas link; Flow Designer should open.



6. Click Design Center in the upper-left corner; you should return to the Design Center.

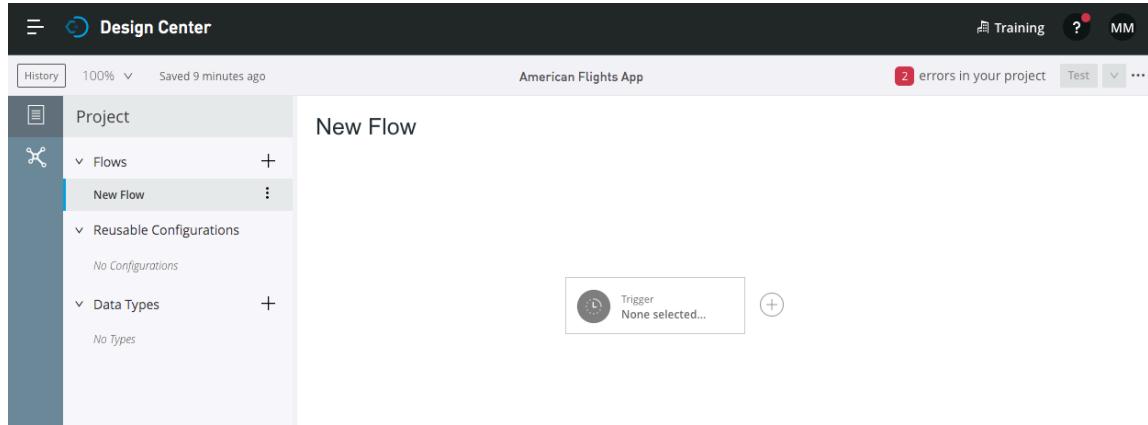


7. In the Design Center project list, click the row containing the American Flights App - but not the American Flights App link; you should see information about the project displayed on the right side of the page.

Name	Project Type	Last Update
American Flights App	Mule Application	November 10th, 2019

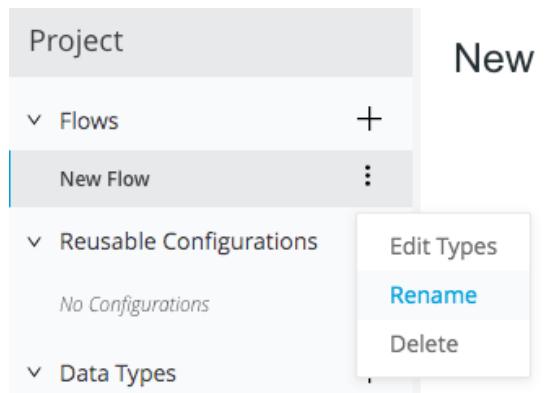
Details	
Name	American Flights App
Modified	November 10th, 2019
Created	November 10th, 2019
Created by	jag35411
Environment	d092d70f-d0d1-4c5b-87ac-46d03249fc2c
Status	Running
Deployment url	americanflightsapp-eucx.us-e2.cloudhub.io

8. Click the Open button or click the American Flights App link in the project list; the project should open in Flow Designer.

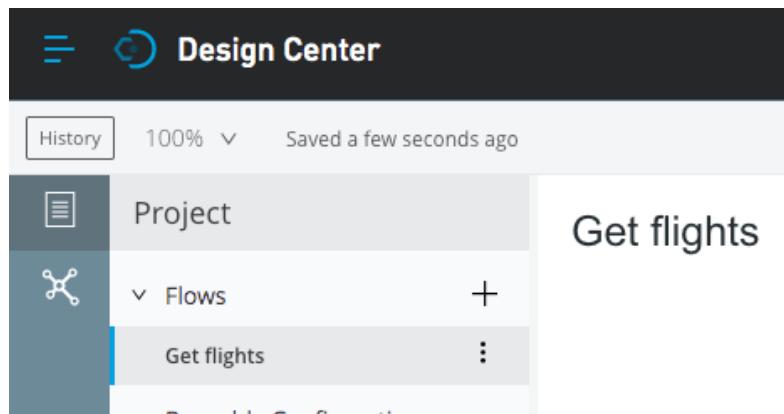


Rename the flow

9. Locate New Flow in the project explorer.
10. Click its option menu and select Rename.

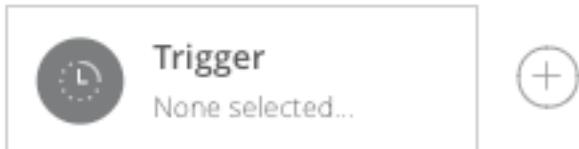


11. In the Rename Flow dialog box, set the name to Get flights and click OK.

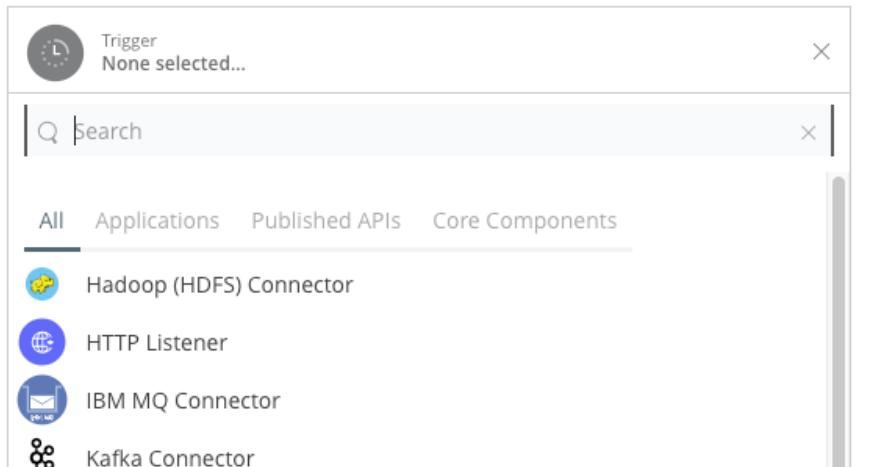


Create an HTTP trigger for a flow in the application

12. In Flow Designer, click the Trigger card.

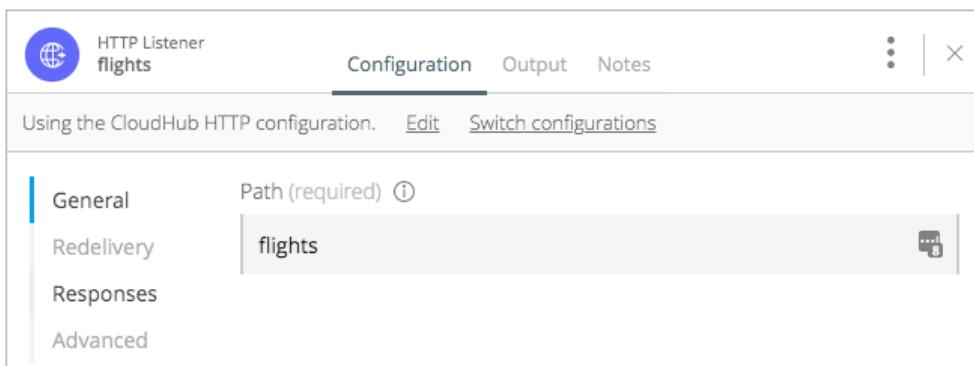


13. In the Trigger card, select HTTP Listener.



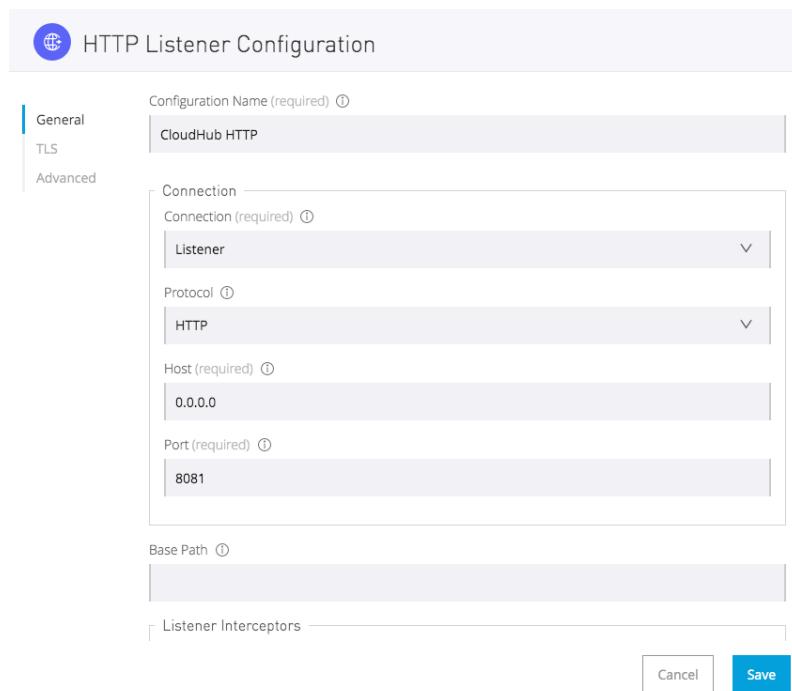
Note: Use your arrow keys or mouse scroll wheel if dragging the Triggers card's scroll bar does not work.

14. In the HTTP Listener dialog box, set the path to flights.

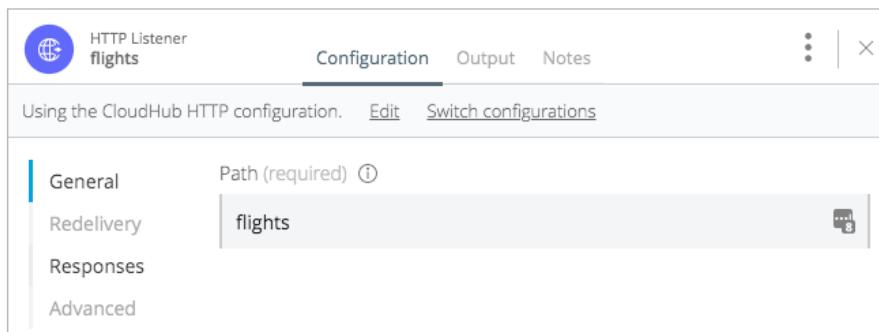


15. Click the Edit link for the CloudHub HTTP configuration.

16. In the HTTP Listener Configuration dialog box, review the information and click Cancel.

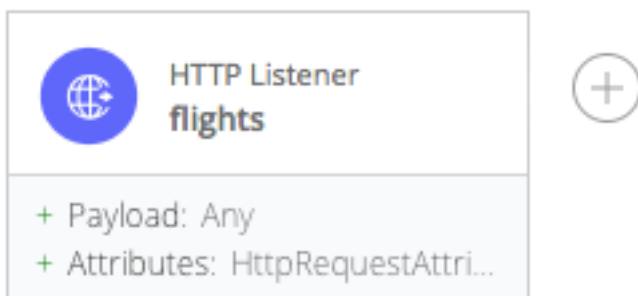


17. Close the HTTP Listener dialog box.

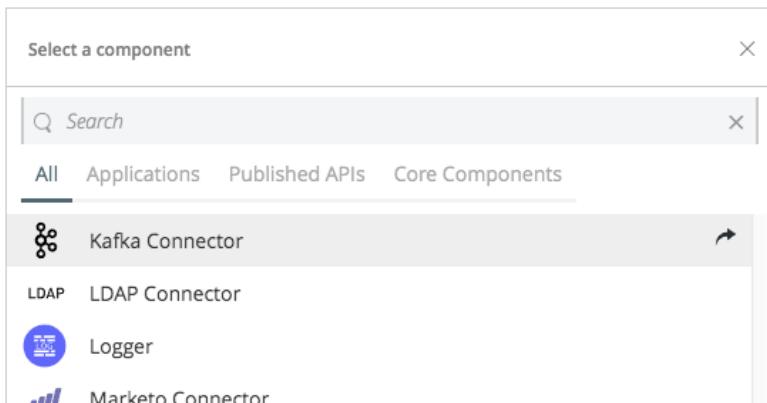


Add a Logger

18. Click the add button next to the HTTP Listener card.

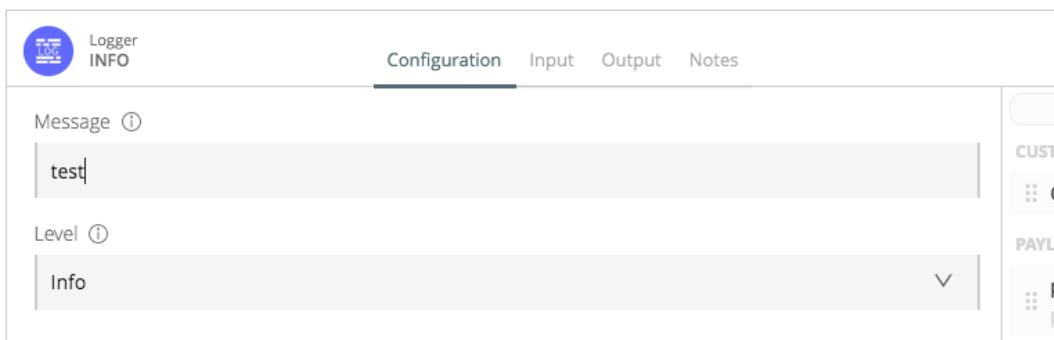


19. In the Select a component dialog box, select Logger.



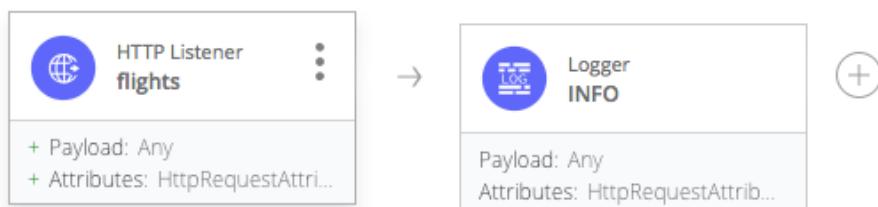
Note: Use your arrow keys or mouse scroll wheel if dragging the Triggers card's scroll bar does not work.

20. In the Logger dialog box, set the message to test.



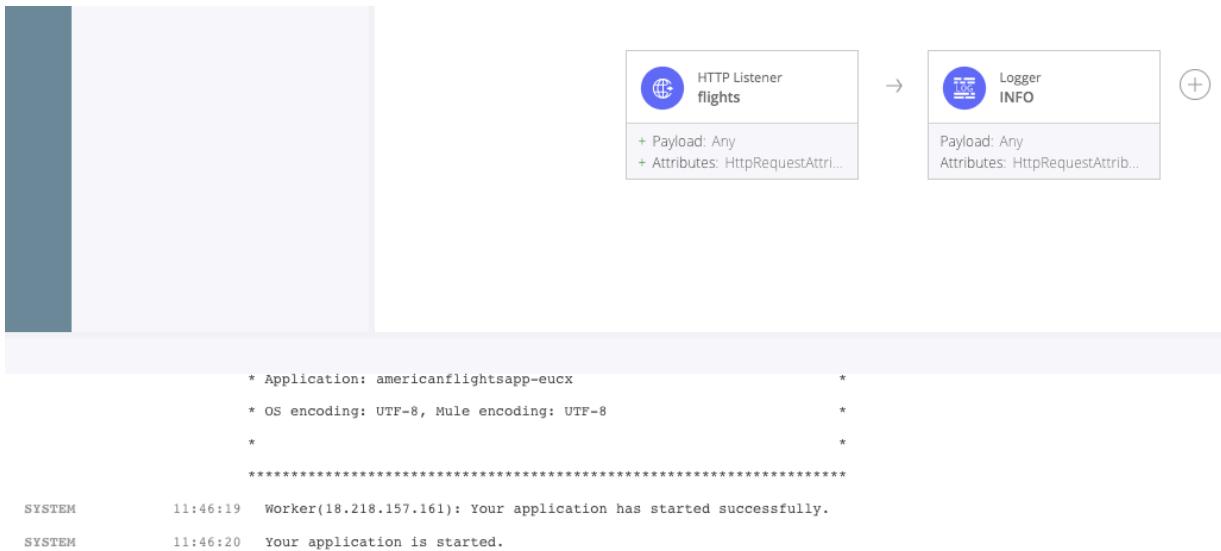
21. Close the card.

22. Notice that there are gray lines across the middle of both cards.

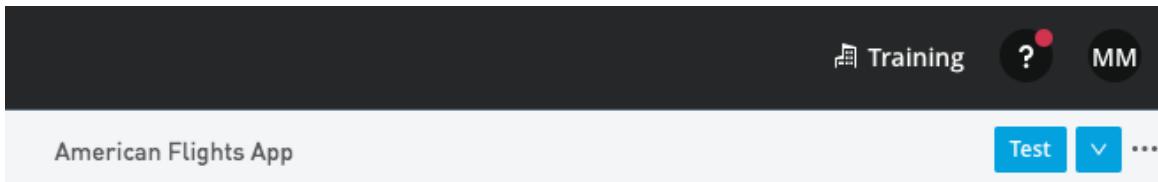


Run the application

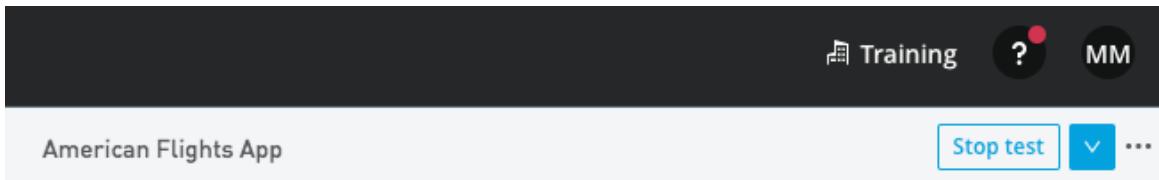
23. Click the Logs tab located in the lower-right corner of the window; you should see that your application is already started.



24. Click the Test button in the main menu bar.



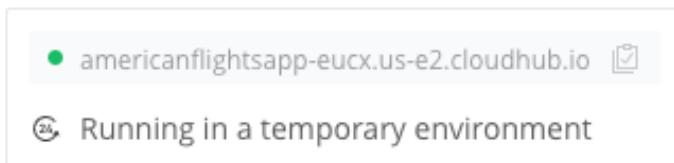
25. Ensure application status changes from Test to Preparing to test to Stop test.



Note: If your application fails to run, look at the messages in the Logs panel. If there is no message about incorrect syntax, try restarting the workspace by clicking the options menu in the application status area and selecting Restart workspace.

26. Near the upper-right corner, locate the box which contains the generated URL for the application.

Note: A four-letter suffix is added to the application name to guarantee that it is unique across all applications on CloudHub.



27. Click the icon to the right of the generated URL to copy its link.

Test the application

28. Return to Advanced REST Client, paste the copied link, prepend it with http://, and click Send; you should get a 404 Not Found status with the message No listener for endpoint: /.

A screenshot of the Advanced REST Client interface. At the top, there is a header with "Method: GET" and "Request URL: http://americanflightsapp-hxkm.us-e2.cloudhub.io". To the right of the URL is a "SEND" button and a more options menu. Below the header, there is a "Parameters" dropdown. The main area shows a response status of "404 Not Found" with a time of "1240.90 ms". There are several icons for interacting with the response (copy, refresh, etc.). Below the status, the message "No listener for endpoint: /" is displayed.

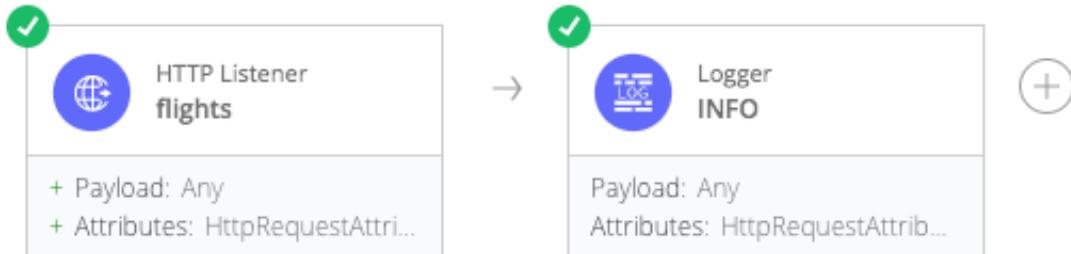
29. Add /flights to the path and click Send; you should get a 200 response with no body.

A screenshot of the Advanced REST Client interface. At the top, there is a header with "Method: GET" and "Request URL: http://americanflightsapp-hxkm.us-e2.cloudhub.io/flights". To the right of the URL is a "SEND" button and a more options menu. Below the header, there is a "Parameters" dropdown. The main area shows a response status of "200 OK" with a time of "4560.90 ms". There are several icons for interacting with the response (copy, refresh, etc.). Below the status, the message "No listener for endpoint: /" is displayed.

30. Click Send again to make a second request.

31. Return to Flow Designer.

32. Notice that there are now green check marks on both cards.

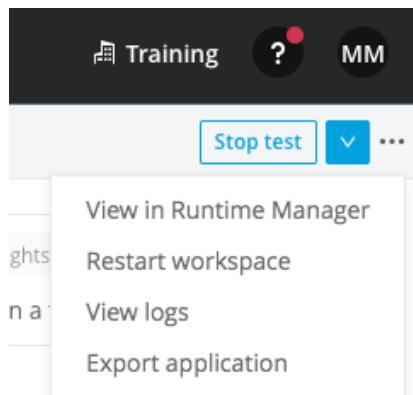


33. Look at the logs; you should see your Logger message displayed twice.

```
* OS encoding: UTF-8, Mule encoding: UTF-8
*
*****
INFO      15:21:04  Starting Bean: listener
INFO      15:39:08  event:21f5b6c0-03fa-11ea-9940-02a5c9ce0f56 test
INFO      15:39:14  event:28a21680-03fa-11ea-9940-02a5c9ce0f56 test
```

View the application in Runtime Manager

34. Click the options menu in the application status area and select View in Runtime Manager; Runtime Manager should open in a new tab.



35. In the new browser tab that opens with Runtime Manager, review the application log file; you should see your test log messages.

The screenshot shows the Runtime Manager interface. On the left, there's a navigation bar with options like DESIGN, Applications, Dashboard, Insight, Logs, Object Store, Queues, and Settings. The Logs option is currently selected. The main area shows an application named "americanflightsapp-eucx". Below it, there's a "Live Console" section with a message about log retention and a search bar. The logs pane lists two entries:

```
15:39:08.664 11/10/2019 Worker-0 [MuleRuntime].cpuLight.04: [americanflightsapp-eucx].get_flights.CPU_LITE @7fafaf INFO event:21f5b6c0-03fa-11ea-9940-02a5c9ce0f56 test
15:39:14.712 11/10/2019 Worker-0 [MuleRuntime].cpuLight.04: [americanflightsapp-eucx].get_flights.CPU_LITE @7fafaf INFO event:28a21680-03fa-11ea-9940-02a5c9ce0f56 test
```

The right side of the interface has a sidebar titled "Deployments" which shows a deployment entry for "11:45 - Deployment" with a "System Log" link and a "Worker-0" status indicator.

Note: If you are prompted to select an environment, select Design. If Runtime Manager still fails to launch in context, close the tab and select View in Runtime Manager again.

36. In the left-side navigation, click Settings.

37. Review the settings page and locate the following information for the application:

- To which environment it was deployed
- To what type of Mule runtime it was deployed
- To what size worker it was deployed

The screenshot shows the Runtime Manager interface with the "Settings" option selected in the left navigation. The main area shows the application "americanflightsapp-eucx". The "Runtime" tab is active, displaying the runtime version as "4.2.0", worker size as "0.2 vCores", and workers as "1". There are also sections for "Properties", "Insight", "Logging", and "Static IPs". Below the runtime section, there's a note about monitoring and a list of checkboxes for application restart, persistent queues, encryption, and object store usage.

Runtime	Properties	Insight	Logging	Static IPs
Runtime version 4.2.0	Worker size 0.2 vCores		Workers 1	

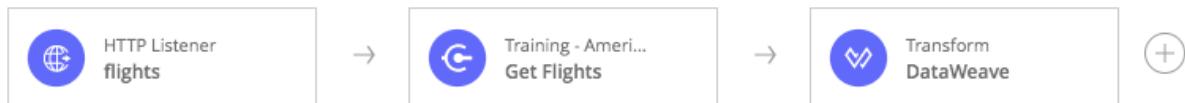
Monitoring note: *To use Monitoring and Visualizer with this version, you may need to enable the agent after deploying. Learn how*

Checkboxes (under Runtime):
 Automatically restart application when not responding
 Persistent queues | Encrypt persistent queues
 Use Object Store v2

Walkthrough 2-3: Create an integration application with Flow Designer that consumes an API

In this walkthrough, you build an integration application to consume an API from Anypoint Exchange. You will:

- Examine Mule event data for calls to an application.
- Use the Training: American Flights API in Anypoint Exchange to get all flights.
- Transform data returned from an API to another format.



Review Mule event data for the calls to the application

1. Return to the American Flights App in Flow Designer.
2. Click the down arrow in the upper-right corner of the Logs panel to close it.
3. Click the HTTP Listener card to expand it.
4. Select the Output tab.
5. Locate the Show drop-down menu that currently has Payload selected.

History
Nov 10, 2019 03:39pm
Nov 10, 2019 03:39pm

6. Locate your two calls to the application in the History panel; there should be no event payload for either call.
7. Change the Show drop-down menu to Attributes.

8. Review the attributes for the Mule event leaving the HTTP Listener processor.

```
{
  "headers": {
    "host": "americanflightsapp-eucx.us-e2.cloudhub.io",
    "x-real-ip": "204.14.236.152",
    "accept": "*/*",
    "client_id": "d846d01ea59640108779eaaa532d00ca",
    "client_secret": "F7C5ecA48EBA449399b4B140416849B0",
    "user-agent": "advanced-rest-client",
    "x-forwarded-for": "204.14.236.152",
    "x-forwarded-port": "80",
    "x-forwarded-proto": "http"
  },
  "clientCertificate": null,
  "method": "GET",
  "scheme": "http",
  "queryParams": {},
  "requestUri": "/flights",
  "queryString": "",
  "version": "HTTP/1.1",
  "maskedRequestPath": null,
  "listenerPath": "/flights",
  "relativePath": "/flights",
  "localAddress": "/172.25.31.98:8081",
  "uriParams": {},
  "rawRequestUri": "/flights",
  "rawRequestPath": "/flights",
  "remoteAddress": "/12.50.200.11.30140"
}
```

9. Close the card.

10. Expand the Logger card.

11. Select the Input tab.

12. Review the payload and attributes values for the two calls.

```
{
  "headers": {
    "host": "americanflightsapp-eucx.us-e2.cloudhub.io",
    "x-real-ip": "204.14.236.152",
    "accept": "*/*",
    "client_id": "d846d01ea59640108779eaaa532d00ca",
    "client_secret": "F7C5ecA48EBA449399b4B140416849B0",
    "user-agent": "advanced-rest-client",
    "x-forwarded-for": "204.14.236.152",
    "x-forwarded-port": "80",
    "x-forwarded-proto": "http"
  },
  "clientCertificate": null,
  "method": "GET",
  "scheme": "http",
  "queryParams": {},
  "requestUri": "/flights",
  "queryString": "",
  "version": "HTTP/1.1",
  "maskedRequestPath": null,
  "listenerPath": "/flights",
  "relativePath": "/flights",
  "localAddress": "/172.25.31.98:8081",
  "uriParams": {},
  "rawRequestUri": "/flights",
  "rawRequestPath": "/flights",
  "remoteAddress": "/12.50.200.11.30140"
}
```

13. Select the Output tab and review the payload and attributes values for the calls.

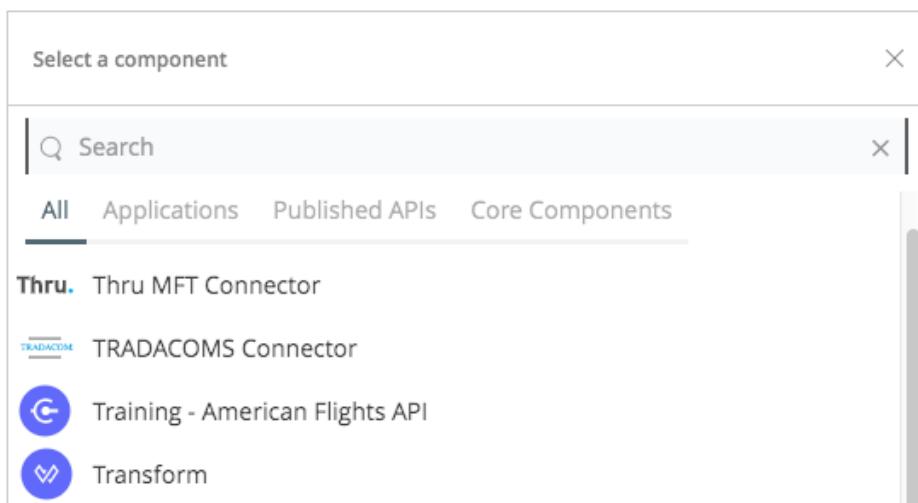
Delete a card

14. Click the options menu for the Logger card and select Delete.

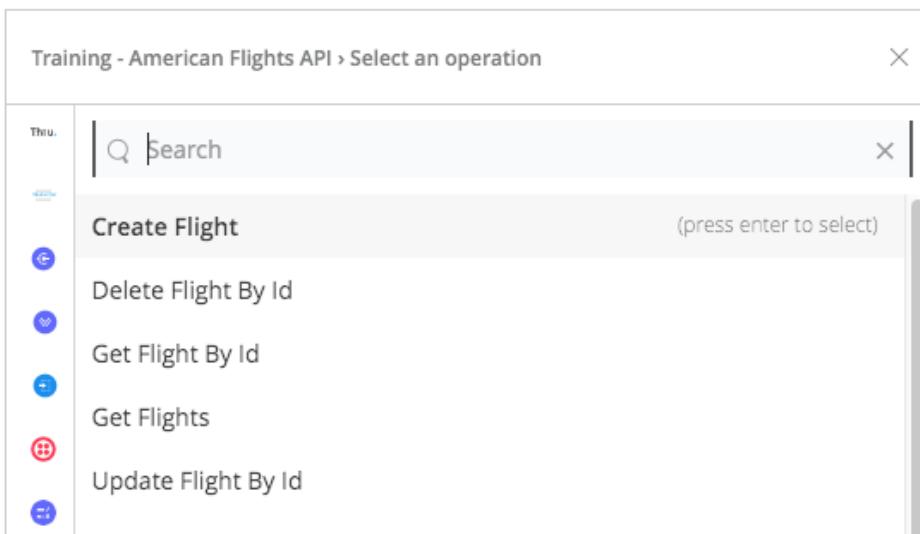


Use Training: American Flights API in Anypoint Exchange to get all flights

15. Click the Add button next to the HTTP Listener card.
16. In the Select a component dialog box, select the Training: American Flights API.

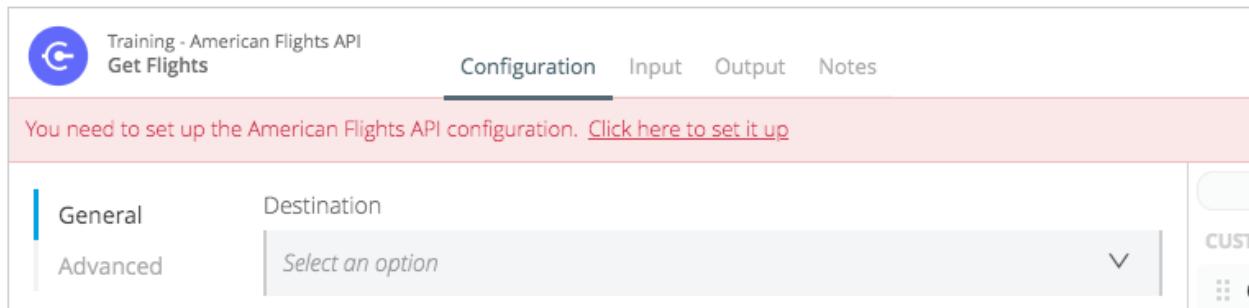


17. In the Training: American Flights API > Select an operation dialog box, select Get Flights.



Configure the American Flights API to use a secured endpoint

18. In the Get Flights dialog box, click the Click here to set it up link for the American Flights API configuration.



19. In the American Flights API Configuration dialog box, set the host, port, base path, and protocol values to the values listed in the course snippets.txt file and click Save:

American Flights API Configuration

Configuration Name (required)

Connection

Host (required)

Port (required)

Base Path

Protocol (required)

20. In the Get Flights dialog box, copy and paste the client_id and client_secret values from the course snippets.txt file.

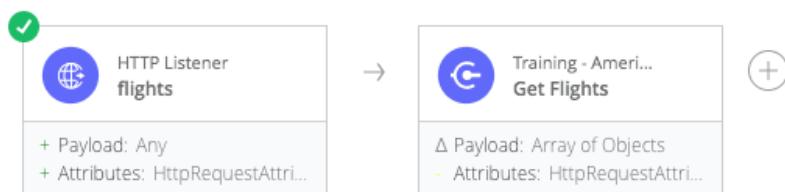
Client Id (required)

d1374b15c6864c3682ddbed2a247a826

Client Secret (required)

4a87fe7e2e43488c927372AEF981F066

21. Close the American Flights API card.



22. Click the Test button in the main menu bar.

23. Wait until the application is running.

Test the application

24. Return to Advanced REST Client and click Send; you should see flight data.

Method: GET Request URL: http://americanflightsapp-hxkm.us-e2.cloudhub.io/flights

Parameters: ▾

200 OK 4376.30 ms DETAILS ▾

[Array[11]]

```
[{"ID": 1, "code": "rree0001", "price": 541, "departureDate": "2016-01-20T00:00:00", "origin": "MUA", "destination": "LAX", "emptySeats": 0, "plane": {"type": "Boeing 787", "totalSeats": 200}}, {"ID": 2, "code": "eefd0123", "price": 541, "departureDate": "2016-01-20T00:00:00", "origin": "MUA", "destination": "LAX", "emptySeats": 0, "plane": {"type": "Boeing 787", "totalSeats": 200}}]
```

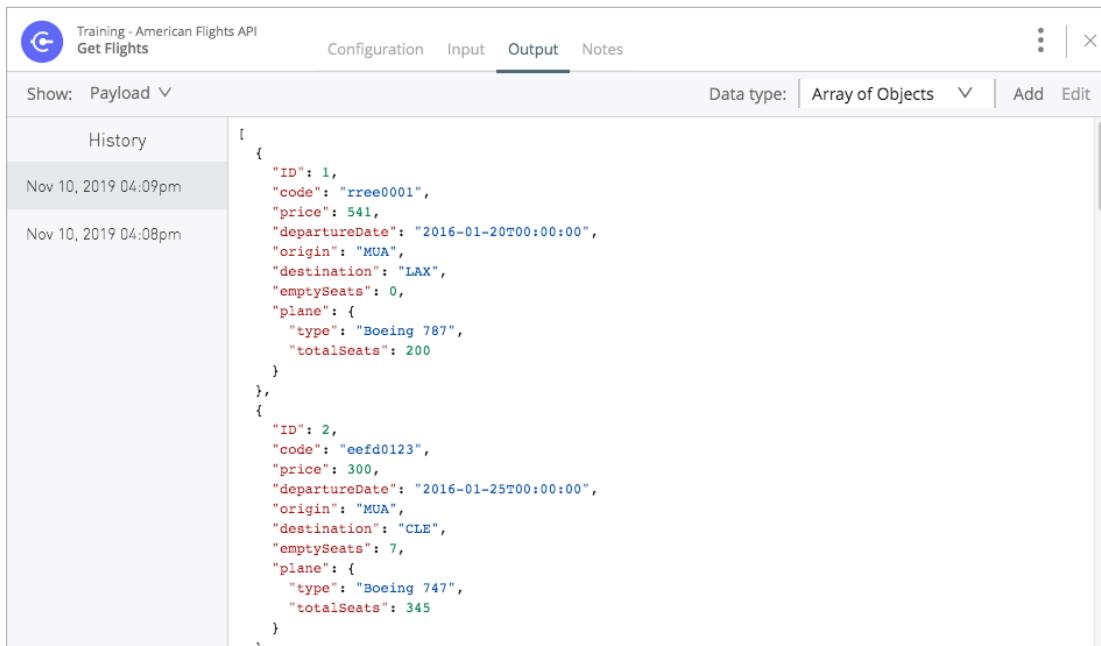
25. Click Send again to make a second request.

Review Mule event data

26. Return to Flow Designer and open the American Flights API card.

27. Select the Input tab and examine the Mule event data.

28. Select the Output tab; you should see payload data.



The screenshot shows the 'Training - American Flights API' card in the Mule Flow Designer. The 'Output' tab is selected. The payload is shown as an array of objects:

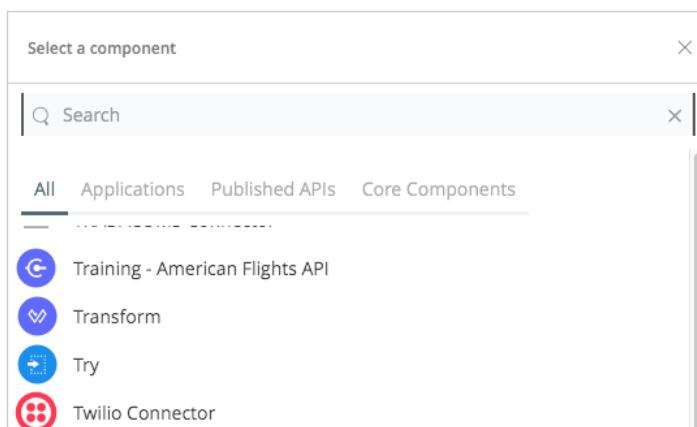
```
[{"ID": 1, "code": "rree0001", "price": 541, "departureDate": "2016-01-20T00:00:00", "origin": "MUA", "destination": "LAX", "emptySeats": 0, "plane": {"type": "Boeing 787", "totalSeats": 200}}, {"ID": 2, "code": "eefd0123", "price": 300, "departureDate": "2016-01-25T00:00:00", "origin": "MUA", "destination": "CLE", "emptySeats": 7, "plane": {"type": "Boeing 747", "totalSeats": 345}}]
```

29. Close the card.

Add and configure a component to transform the data

30. Click the add button in the flow.

31. In the Select a component dialog box, select Transform.



The screenshot shows the 'Select a component' dialog box. The 'Transform' component is selected. Other components listed include 'Training - American Flights API', 'Try', and 'Twilio Connector'.

32. In the Transform card, look at the Mule event structure in the input section.

33. In the output section, click the Create new Data Type button.

The screenshot shows the Mule DataWeave Transform card interface. At the top, there's a header with tabs: Configuration (which is selected), Input, Output, and Notes. Below the header, a message says "Create an output payload to see suggestions for mappings." The Input section on the left shows a search bar and a tree view of the Mule event structure:

- payload (Array<Object>)
 - plane (Object?)
 - code (String)
 - price (Number)
 - origin (String)
 - destination (String)
 - ID (Number?)
 - departureDate (String)
 - emptySeats (Number)
 - attributes (Void)
 - vars (Object)

A tooltip "Drag-and-Drop fields to build the transform" points to the tree view. The Output payload section in the center has a search bar and a tree view with the message "No data available". A button labeled "Create new Data Type" is visible. The Preview section on the right shows a small chart and the message "No data available, please perform some mappings and fill required sample data". At the bottom, there are tabs: Sample data, Script, and Mappings (which is selected).

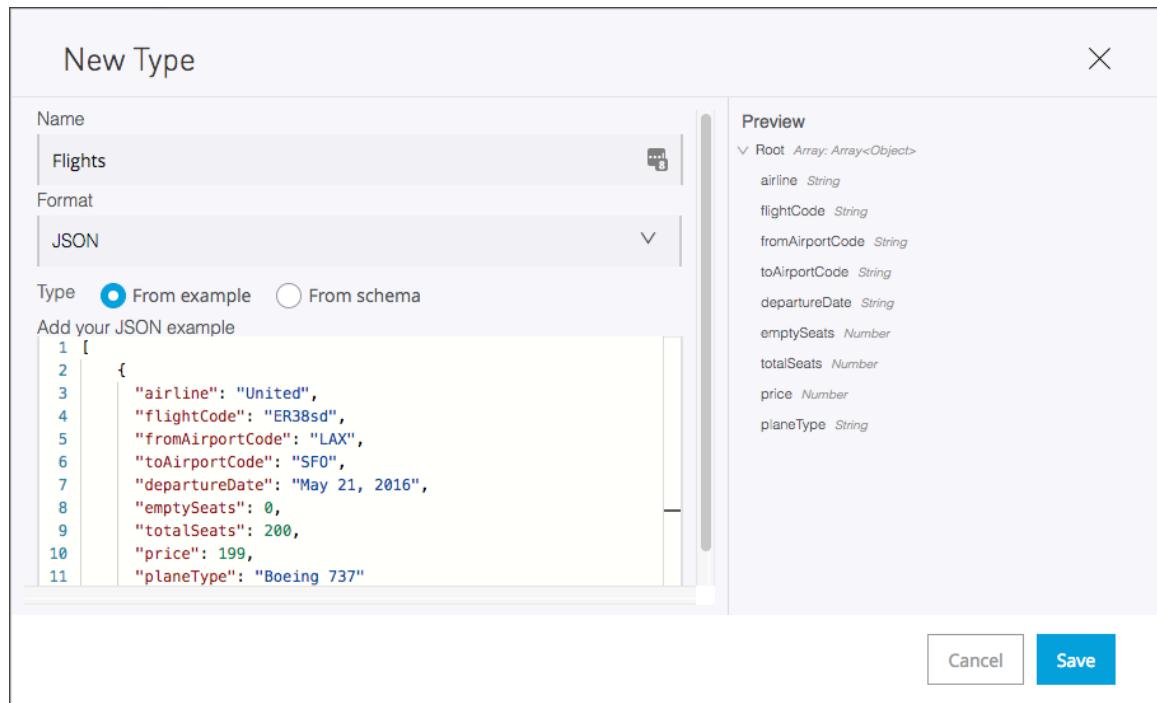
34. In the New Type dialog box, set the following values:

- Name: Flights
- Format: JSON
- Type: From example

35. In the computer's file explorer, return to the student files folder and locate the flights-example.json file in the resources/examples folder.

36. Open the file in a text editor and copy the code.

37. Return to Flow Designer and add your JSON example by pasting the code in the section replacing any existing code.



38. Click Save.

39. In the input section, expand the plane object.

Transform DataWeave

Configuration Input Output Notes

Mapping Suggestion available for this transformation

Preview

Input

Search

payload Array: Array<Object>

plane Object?

type String

totalSeats Number

code String

price Number

origin String

destination String

ID Number?

departureDate String

emptySeats Number

attributes Void

vars Object

Output payload

Search

payload Flights: Array<Object>

airline String

flightCode String

fromAirportCode String

toAirportCode String

departureDate String

emptySeats Number

totalSeats Number

price Number

planeType String

Drag-and-Drop fields to build the transform

Preview

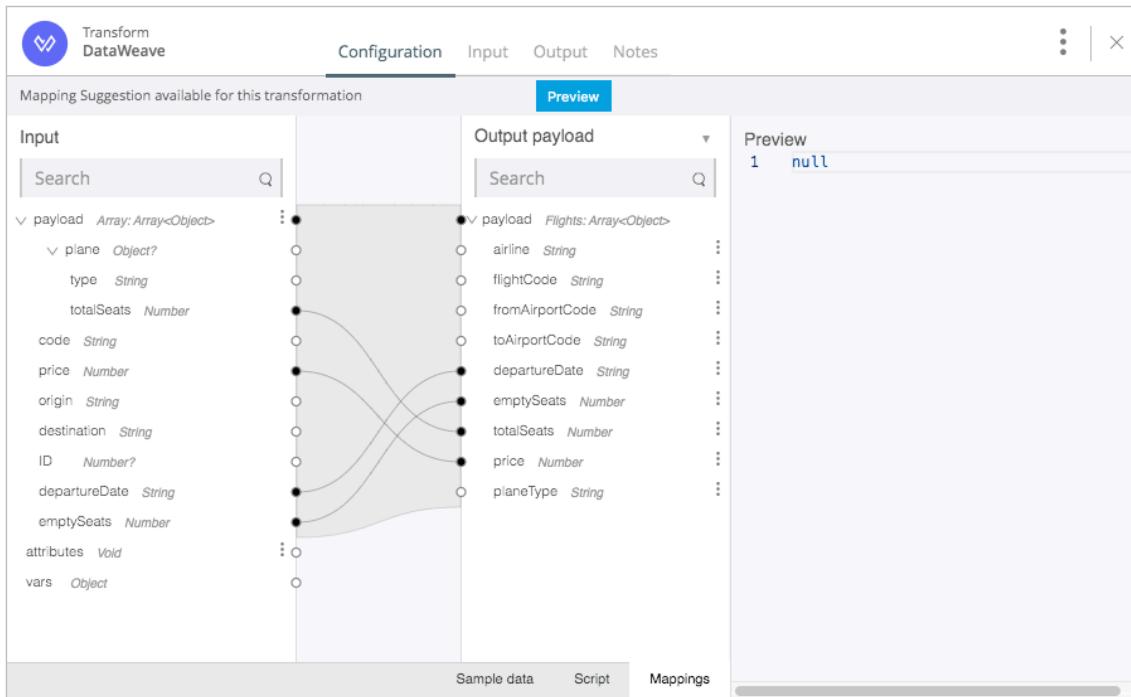
No data available, please perform some mappings and fill required sample data

Mappings

Create the transformation

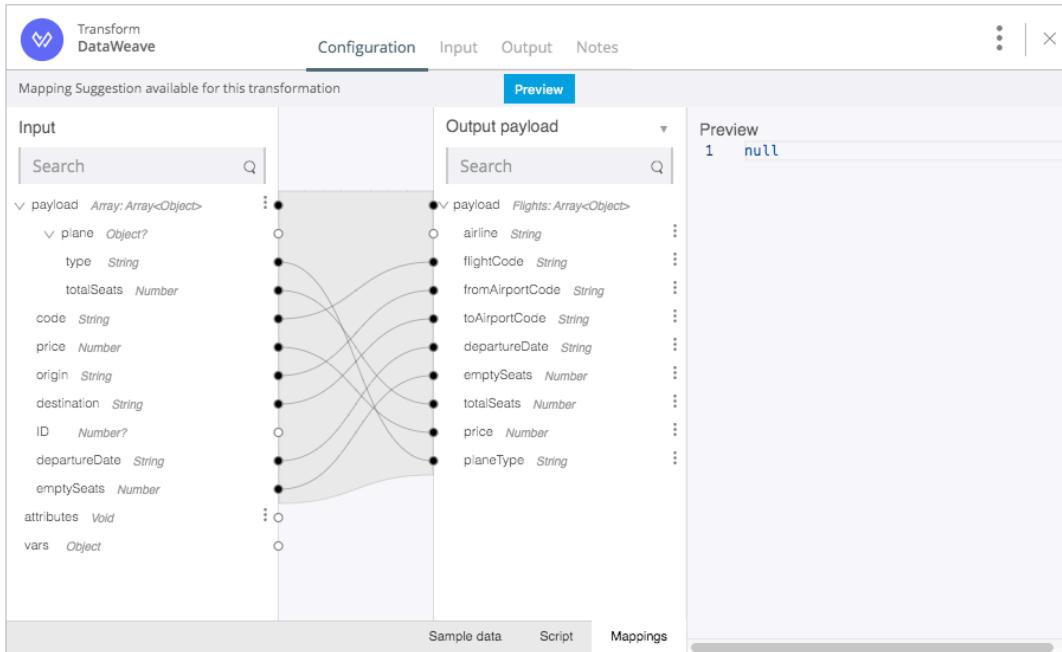
40. Map fields with the same names by dragging them from the input section and dropping them on the corresponding field in the output section.

- price to price
- departureDate to departureDate
- plane > totalSeats to totalSeats
- emptySeats to emptySeats

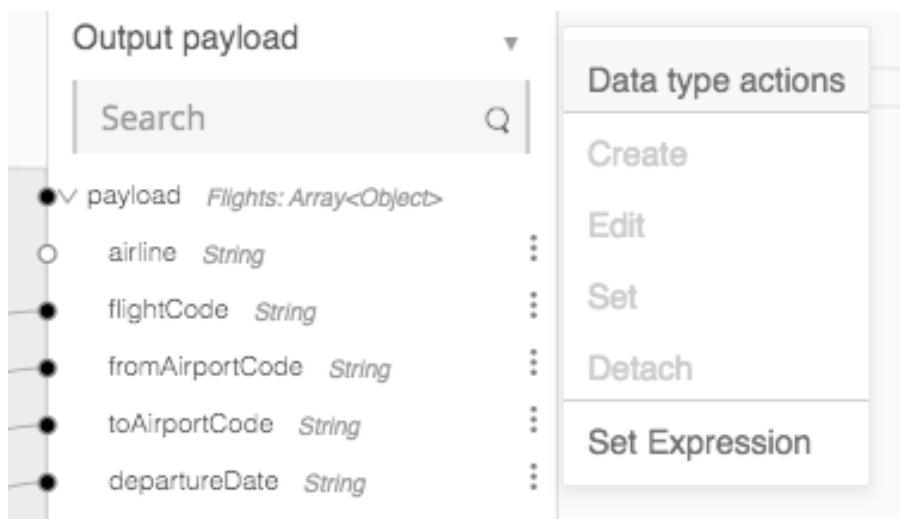


41. Map fields with different names by dragging them from the input section and dropping them on the corresponding field in the output section.

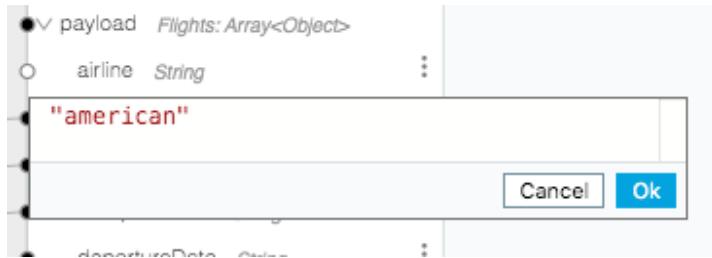
- plane > type to planeType
- code to flightCode
- origin to fromAirportCode
- destination to toAirportCode



42. In the output section, click the options menu for the airline field and select Set Expression.



43. Change the value from null to "american" and click OK.



44. Click the Script tab at the bottom of the card; you should see the DataWeave expression for the transformation.

Note: You learn to write DataWeave transformations later in the Development Fundamentals course.

The screenshot shows the DataWeave transformation configuration screen. The 'Script' tab is selected. The 'Input' section shows a search bar and a detailed tree view of the payload structure. The 'Transformation script' section contains the following DataWeave code:

```
1 %dw 2.0
2 output application/json
3 ---
4 (payload map (value0, index0) -> {
5   flightCode: value0.code,
6   fromAirportCode: value0.origin,
7   toAirportCode: value0.destination,
8   departureDate: value0.departureDate,
9   emptySeats: value0.emptySeats,
10  totalSeats: value0.plane.totalSeats,
11  price: value0.price,
12  planeType: value0.plane.type,
13  airline: "american"
14 })
```

The 'Preview' section on the right shows the resulting JSON output:

```
{"errorType": "class com.mulesoft.agent.exception.NoSuchApplicationException", "errorMessage": "No application deployed for applicationId: tooling-application-6efe12f0-03ff-11ea-9940-02a5c9ce0f56"}
```

Add sample data

45. Click the Sample data tab in the preview section.

46. In the computer's file explorer, return to the student files folder and locate the american-flights-example.json file in the resources/examples folder.

47. Open the file in a text editor and copy the code.

48. Return to Flow Designer and paste the code in the sample data for payload section.

The screenshot shows the Mule DataWeave Transform tool interface. The top navigation bar includes 'Transform DataWeave', 'Configuration' (selected), 'Input', 'Output', and 'Notes'. Below the navigation is a 'Mapping Suggestion available for this transformation' message and a 'Preview' button. The 'Input' section on the left lists various fields: payload (Array<Object>), plane (Object), type (String), totalSeats (Number), code (String), price (Number), origin (String), destination (String), ID (Number?), departureDate (String), emptySeats (Number), attributes (Void), and vars (Object). The 'Sample data for payload (application/json)' section in the center contains the following JSON code:

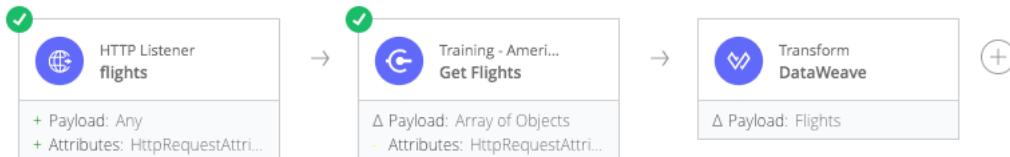
```
6 "origin": "MUA",
7 "destination": "SFO",
8 "emptySeats": 0,
9 "plane": {
10   "type": "Boeing 737",
11   "totalSeats": 150
12 },
13 },
14   "ID": 2,
15   "code": "ER45if",
16   "price": 345.99,
17   "departureDate": "2016/02/11",
18   "origin": "MUA",
19   "destination": "LAX",
20   "emptySeats": 52,
21   "plane": {
22     "type": "Boeing 777",
23     "totalSeats": 300
24   }
25 }
26 ]
```

The 'Preview' section on the right shows the transformed JSON output:

```
1 [
2   {
3     "flightCode": "ER38sd",
4     "fromAirportCode": "MUA",
5     "toAirportCode": "SFO",
6     "departureDate": "2016/03/20",
7     "emptySeats": 0,
8     "totalSeats": 150,
9     "price": 400.00,
10    "planeType": "Boeing 737",
11    "airline": "american"
12  },
13  {
14    "flightCode": "ER45if",
15    "fromAirportCode": "MUA",
16    "toAirportCode": "LAX",
17    "departureDate": "2016/02/11",
18    "emptySeats": 52,
19    "totalSeats": 300,
20    "price": 345.99,
21    "planeType": "Boeing 777",
22    "airline": "american"
23 }
```

49. Look at the preview section, you should see a sample response for the transformation.

50. Close the card.



Locate the data type and configuration definitions

51. Locate the connector configuration and the new Flights data type in the project explorer.

The Project Explorer sidebar shows the following structure:

- Project
- Flows
 - Get flights
- Reusable Configurations
 - American Flights API
 - CloudHub HTTP
- Data Types
 - Flights

Test the application

52. Test the project.
53. Return to Advanced REST Client and click Send to make another request to <http://americanflightsapp-xxxx.{region}.cloudbu.io/flights>; you should see all the flight data as JSON again but now with a different structure.

The screenshot shows the Advanced REST Client interface. At the top, there are fields for Method (GET), Request URL (<http://americanflightsapp-hxkm.us-e2.cloudbu.io/flights>), and a blue SEND button. Below the URL field is a dropdown labeled "Parameters". The main area displays the response details: a green "200 OK" status box with a "1462.70 ms" latency, and a "DETAILS" dropdown arrow. To the left of the response body are icons for copy, download, and refresh. The response body itself is a JSON array of flight objects:

```
[Array[11]
-0: {
  "flightCode": "rree0001",
  "fromAirportCode": "MUA",
  "toAirportCode": "LAX",
  "departureDate": "2016-01-20T00:00:00",
  "emptySeats": 0,
  "totalSeats": 200,
  "price": 541,
  "planeType": "Boeing 787",
  "airline": "american"
},
-1: {
  "flightCode": "eefd0123",
  "fromAirportCode": "MUA",
  "toAirportCode": "LAX",
  "departureDate": "2016-01-20T00:00:00",
  "emptySeats": 0,
  "totalSeats": 200,
  "price": 541,
  "planeType": "Boeing 787",
  "airline": "american"
}...]
```

Stop the application

54. Return to Runtime Manager.
55. In the left-side navigation, click Applications.

56. Select the row with your application; you should see information about the application displayed on the right side of the window.

The screenshot shows the MuleSoft Runtime Manager interface. On the left, there's a sidebar with 'DESIGN' selected, followed by 'Applications', 'Servers', 'Alerts', 'VPCs', and 'Load Balancers'. The main area has tabs for 'Deploy application' and 'Search Applications'. A table lists applications with columns: Name, Server, Status, and File. One row is selected: 'americanflightsapp-eucx' (Server: CloudHub, Status: Started, File: americanflightsapp-eucx.jar). To the right, a detailed view of the application is shown with a close button 'X'. It displays the application name, status (Started), server (CloudHub), file path (americanflightsapp-eucx.jar), and deployment details: Last Updated (2019-11-10 11:46:19AM), App url (americanflightsapp-eucx.us-e2.cloudhub.io), Runtime version (4.2.0), Worker size (0.2 vCores), and Workers (1). Buttons for 'Manage Application', 'Logs', and 'Insight' are at the bottom, along with a link to 'View Associated Alerts'.

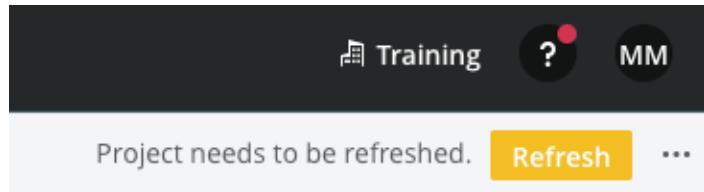
57. Click the drop-down menu button next to Started and select Stop; the status should change to Undeployed.

Note: You can deploy it again from Flow Designer when or if you work on the application again.

This screenshot shows the same Runtime Manager interface after the application status was changed. The application 'americanflightsapp-eucx' is now listed with a status of 'Undeployed' (radio button selected) instead of 'Started'. All other details (Server: CloudHub, File: americanflightsapp-eucx.jar, deployment info, and runtime settings) remain the same as in the previous screenshot.

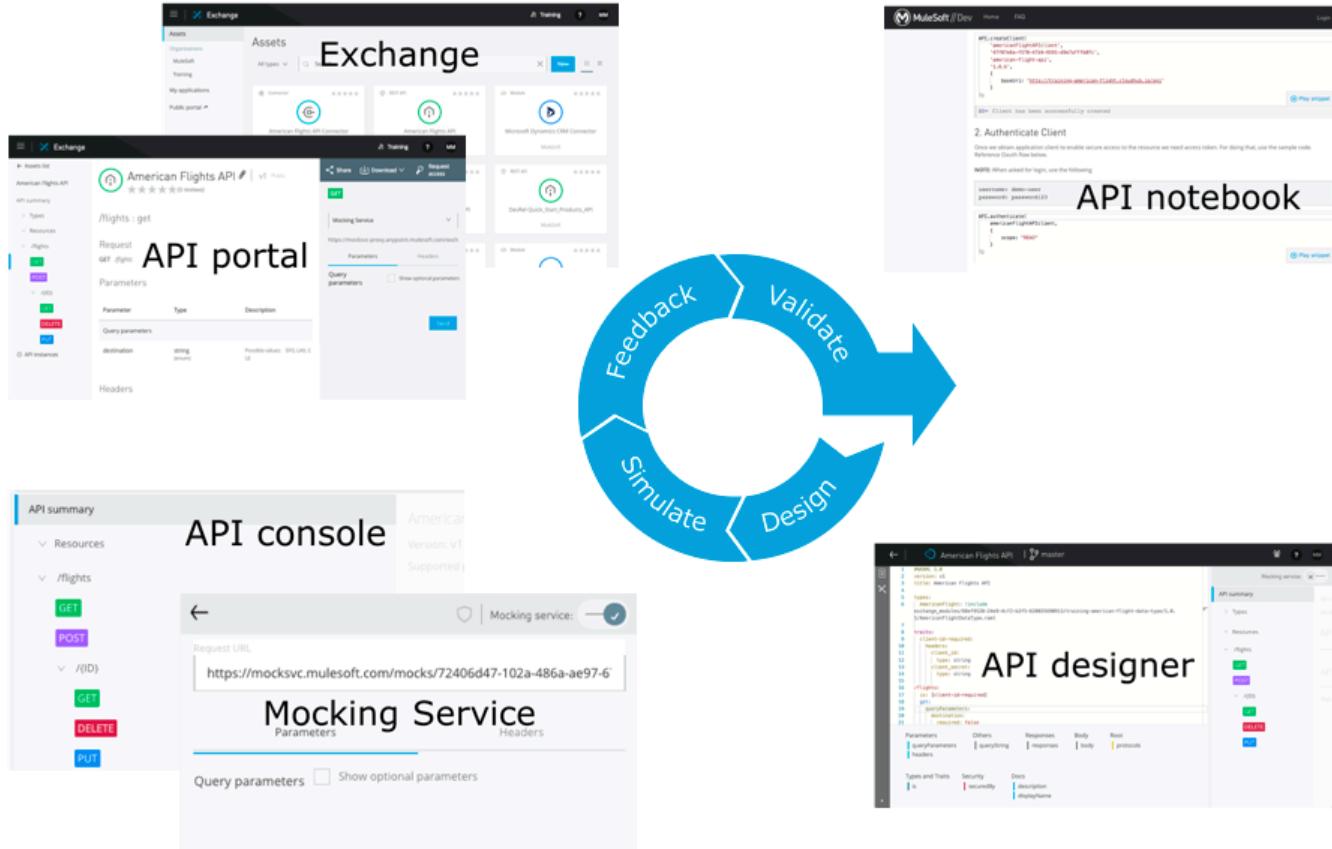
58. Close Runtime Manager.

59. Return to Flow Designer; you should see the application is not running and needs to be refreshed.



60. Return to Design Center.

Module 3: Designing APIs



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

- Define APIs with RAML, the Restful API Modeling Language.
- Mock APIs to test their design before they are built.
- Make APIs discoverable by adding them to the private Anypoint Exchange.
- Create public API portals for external developers.

Walkthrough 3-1: Use API Designer to define an API with RAML

In this walkthrough, you create an API definition with RAML using API Designer. You will:

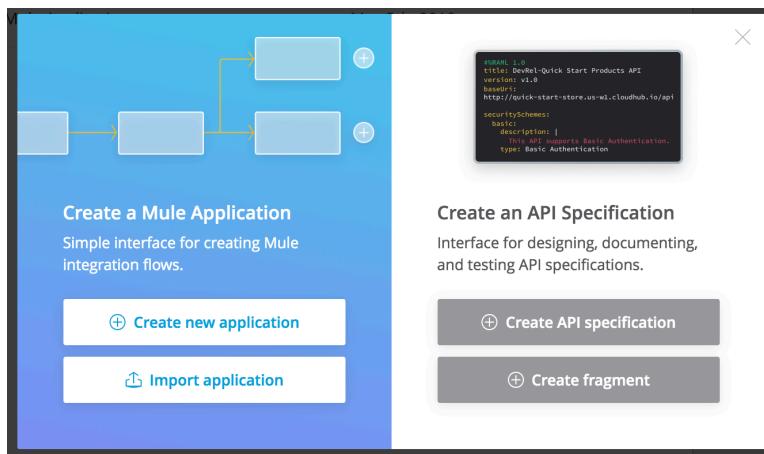
- Define resources and nested resources.
- Define get and post methods.
- Specify query parameters.
- Interact with an API using the API console.

The screenshot shows the Mule API Designer interface. On the left, the 'Design Center' sidebar lists files and exchange modules. In the center, a code editor displays a RAML 1.0 specification for the 'American Flights API'. The specification includes a root resource '/flights' with 'get', 'post', and '/{ID}:get' methods, and a query parameter 'destination' with enum values SFO, LAX, and CLE. To the right, the 'Mocking service' API console shows a 'Get' request for the '/flights' endpoint. Below the console, sections for 'Code examples', 'Query parameters', and a detailed view of the 'destination' parameter are visible.

```
#!/%RAML 1.0
title: American Flights API
/flights:
  get:
    queryParameters:
      destination:
        required: false
        enum:
          - SFO
          - LAX
          - CLE
  post:
    /{ID}:
      get:
```

Create a new Design Center project

1. Return to Design Center.
2. Click the Create button and select Create API specification.



3. In the new specification dialog box, set the project name to American Flights API.

4. Ensure Code editor is selected and click Create Specification; API Designer should open.

New Specification New Fragment

Name
American Flights API

Code editor A complete editing experience with interactive documentation.

Visual editor A visual interface for scaffolding API specifications.

Language RAML 1.0

Save to... Design Center

Cancel Create Specification

5. Review the three sections of API Designer: the file browser, the editor, and the API console.

Add a RAML resource

6. In the editor, place the cursor on a new line of code at the end of the file.
7. Add a resource called flights.

```
1  #%RAML 1.0
2  title: American Flights API
3
4  /flights:
```

View the API console

8. Look at the API console on the right side of the window; you should see summary information for the API.

Note: If you do not see the API console, click the arrow located in the upper-right corner of the web browser window.

The screenshot shows the MuleSoft API Designer interface. On the left, there's a sidebar with a 'Files' section containing 'american-flights-api.raml'. The main area is titled 'American Flights API/master' and contains the following RAML code:

```
1  #%RAML 1.0
2  title: American Flights API
3
4  /flights:
```

To the right, the API console displays the API title 'American Flights API' and a message 'Base URI not defined in the API file.' Below that, under 'API endpoints', it lists '/flights'.

Add RAML methods

9. In the editor, go to a new line of code and backspace so you are indented the same amount as the flights resource; look at the contents of the API Designer shelf.

Note: If you don't see the API Designer shelf, it is either minimized or there is an error in your code. To check if it is minimized, go to the bottom of the web browser window and look for an arrow. If you see the arrow, click it to display the shelf.

10. Indent by pressing the Tab key; the contents in the API Designer shelf should change.

The screenshot shows the MuleSoft API Designer interface with the API Designer shelf open. The shelf has three tabs: 'Docs', 'Methods', and 'Types and Traits'. Under 'Methods', the following options are listed: 'get', 'put', 'post', 'delete', 'options', 'head', and 'patch'. Under 'Types and Traits', the following options are listed: 'is' and 'type'.

11. Click the get method in the shelf.
12. Look at the API console; you should see a GET method for the flights resource.
13. In the editor, backspace so you are indented the same amount as the get method.
14. Click the post method in the shelf.
15. Look at the API console; you should see GET and POST methods for the flights resource.

```

1 %%RAML 1.0
2 title: American Flights API
3
4 /flights:
5   get:
6   post:
7
8
9
10
    
```

American Flights API

Base URI not defined in the API file.

API endpoints

/flights

GET POST

Add a nested RAML resource

16. In the editor, backspace and then go to a new line.
17. Make sure you are still under the flights resource (at the same indentation as the methods).
18. Add a nested resource for a flight with a particular ID.

`/{ID}:`

19. Add a get method to this resource.
20. Look at the API console; you should see the nested resource with a GET method.

```

1 %%RAML 1.0
2 title: American Flights API
3
4 /flights:
5   get:
6   post:
7
8   /{ID}:
9     get:
10
    
```

American Flights API

Base URI not defined in the API file.

API endpoints

/flights

GET POST

/flights/{ID}

GET

Add an optional query parameter

21. In the editor, indent under the /flights get method (not the /flights/{ID} get method).
22. In the shelf, click the queryParameters parameter.
23. Add a key named destination.

```
1  #%%RAML 1.0
2  title: American Flights API
3
4  /flights:
5    get:
6      queryParameters:
7        destination:|
8    post:
9
10 /{ID}:
11   get:
```

24. Indent under the destination query parameter and look at the possible parameters in the shelf.
25. In the shelf, click the required parameter.
26. In the shelf, click false.
27. Go to a new line of code; you should be at the same indent level as required.
28. In the shelf, click the enum parameter.
29. Set enum to a set of values including SFO, LAX, and CLE.

```
4  /flights:
5    get:
6      queryParameters:
7        destination:
8          required: false
9        enum:|
10       - SFO
11       - LAX
12       - CLE
```

Try to call an API method using the API console

30. In the API console, click the GET method for the /flights resource.

31. Review the information.

The screenshot shows a RAML API documentation interface. At the top, there's a header with a shield icon, the text "Mocking service:", and a close button. Below the header, there are two buttons: "Get" and "Try it". Under the "Get" button, there's a method selector "GET /flights" with a checked checkbox. To the right of the method, there's a "Code examples" section with a "Show" button and a dropdown arrow. Below this, there's a "Query parameters" section with a "Hide" button and a dropdown arrow. Under "Query parameters", there's a "destination" field described as a String Enum with values SFO, LAX, and CLE.

32. Click the Try it button; you should get a message that the Request URL is invalid; a URL to call to try the API needs to be added to the RAML definition.

The screenshot shows the "Try it" interface. At the top, there's a header with a shield icon, the text "Mocking service:", and a close button. Below the header, there are two buttons: "Get" and "Back to docs". Under the "Get" button, there's a "Request URL" input field containing the value "/flights". Below the input field, a red message says "The URL is invalid". There are tabs for "Parameters" and "Headers", with "Parameters" being the active tab. Under "Parameters", there's a "Query parameters" section with a checkbox labeled "Show optional parameters". At the bottom, there's a "Send" button and a red message "Fill in required parameters".

Walkthrough 3-2: Use the mocking service to test an API

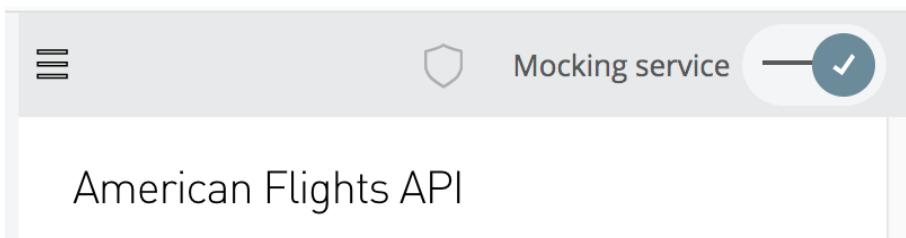
In this walkthrough, you test the API using the Anypoint Platform mocking service. You will:

- Turn on the mocking service.
- Use the API console to make calls to a mocked API.

The screenshot shows the Anypoint Platform interface. On the left, the 'Design Center' pane displays a file named 'american-flights-api.raml'. The code editor shows a RAML 1.0 definition for an 'American Flights API'. On the right, the 'Mocking service' tab of the API console is active. It shows a 'Body content type' dropdown set to 'Select value'. Below it, a 'Send' button is visible, followed by a response status of '200 OK' and a duration of '141.73 ms'. A 'Details' link is also present.

Turn on the mocking service

1. Return to API Designer.
2. Locate the Mocking service slider in the menu bar at the top of the API console.
3. Click the right side of the slider to turn it on.



4. Look at the baseUri added to the RAML definition in the editor.

```
1  #%RAML 1.0
2  baseUri: https://anypoint.mulesoft.com/mockng/api/v1/links/90898d12-2465-4c94-b5ce-75030cdd3dc7/ #
3  title: American Flights API
```

Test the /flights:get resource

5. In API console, click the Try it button for the flights GET method then click the Send button; you should get a 200 status code and an empty response.

The screenshot shows the MuleSoft API Mocking Service interface. At the top, there's a shield icon and the text "Mocking service: [checkmark]". Below that, a "Get" button and a "Back to docs" link. Under "Request URL", the URL "https://anypoint.mulesoft.com/mockng/api/v1/l" is entered. A "Parameters" tab is selected, showing a "Query parameters" section with a checkbox labeled "Show optional parameters" which is unchecked. Below this is a "Send" button. The response section shows a green "200 OK" status with "379.20 ms" latency. There are download and copy icons, and a "Details" link. The response body is shown as "{}".

6. Select the Show optional parameters checkbox.
7. In the destination drop-down menu, select SFO and click Send; you should get the same response.

Test the /flights/{ID} resource

8. Click the menu button located in the upper-left of the API console and select Summary to return to the resource list.

API endpoints

/flights
GET POST

/flights/{ID}
GET

9. Click the GET method for the /{ID} nested resource.

10. Click Try it; you should see a message that the Request URL is invalid.

The screenshot shows the MuleSoft Anypoint Mocking service interface. At the top, there's a navigation bar with a shield icon, the text "Mocking service:", and a toggle switch. Below the bar, the word "Get" is displayed next to a "Back to docs" link. A red error message "The URL is invalid" is prominently shown above a text input field containing the URL "https://anypoint.mulesoft.com/mockng/api/v1/l". Below the URL input, there are tabs for "Parameters" (which is selected) and "Headers". Under the "Parameters" tab, there's a section for "URI parameters" with a field labeled "ID*" containing the value "ID*".

11. In the ID text box, enter a value of 10.

12. Click the Send button.

13. Look at the response; you should get the same 200 status code and an empty response.

The screenshot shows the MuleSoft Anypoint Mocking service interface after the URL has been corrected. The "Request URL" field now contains "https://anypoint.mulesoft.com/mockng/api/v1/l". The "Parameters" tab is still selected, and the "ID*" field now contains the value "10". Below the parameters, there's a "Query parameters" section with a "+ Add query parameter" button. At the bottom left is a blue "Send" button. The response section shows a green box indicating "200 OK" and "882.25 ms". To the right of the status code is a "Details" link with a downward arrow. Below the status information, there are icons for download, copy, and refresh, followed by a JSON placeholder "[{}]".

Test the /flights:post resource

14. Use the menu button located in the upper-left of the API console to return to the resource list.
15. Click the POST method.
16. Click Try it.
17. Select the Body tab; it should not have any content.

Post [Back to docs](#)

Request URL

`https://anypoint.mulesoft.com/mockng/api/v1/l`

Parameters

Headers

Body

Body content type



Select value



18. Click the Send button.
19. Look at the response; you should get the same generic 200 status code response.

200 OK 101.20 ms [Details](#) ▾

{}

Walkthrough 3-3: Add request and response details

In this walkthrough, you add information about each of the methods to the API specification. You will:

- Use API fragments from Exchange.
- Add a data type and use it to define method requests and responses.
- Add example JSON requests and responses.
- Test an API and get example responses.

The screenshot shows the MuleSoft API Designer interface. On the left, the file browser displays 'Files' with sub-folders 'examples' and 'exchange_modules', and the file 'american-flights-api.raml' selected. The main area shows the RAML code for the 'flights' resource. On the right, the 'Mocking service' interface is open, showing a 'Post' request configuration. The 'Request URL' is set to <https://anypoint.mulesoft.com/mockng/api/v1/l>. The 'Body' tab is selected, showing a JSON example for a flight search. The JSON is as follows:

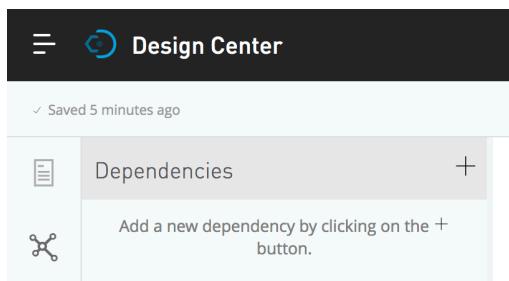
```
{  
  "code": "ER38sd",  
  "price": 400,  
  "departureDate": "2017/07/26",  
  "origin": "CLE",  
  "destination": "SFO",  
  "emptySeats": 0,  
  "plane": {  
    "type": "Boeing 737",  
    "totalSeats": 150  
  }  
}
```

Add data type and example fragments from Exchange

1. Return to API Designer.
2. In the file browser, click the Exchange dependencies button.

The screenshot shows the MuleSoft API Designer interface. The file browser on the left has 'Files' selected, showing the 'american-flights-api.raml' file. Below it, the 'Exchange dependencies' button is highlighted with a dark overlay, indicating it is selected or active.

- Click the Add button.

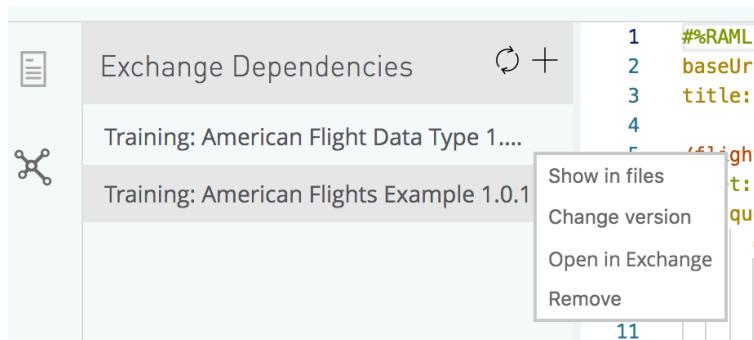


- In the Exchange Dependencies dialog box, select the Training: American Flight Data Type and the Training: American Flights Example.

Name	Version	Date modified	Rating	Created by
<input checked="" type="checkbox"/> Training: American Flight Data Type 🔗	1.0.1	Sep 10, 2019	★★★★★	MuleSoft Organiz...
<input checked="" type="checkbox"/> Training: American Flights Example 🔗	1.0.1	Sep 10, 2019	★★★★★	MuleSoft Organiz...
<input type="checkbox"/> Training: OAuth2.0 Security Scheme 🔗	1.0.1	Sep 10, 2019	★★★★★	MuleSoft Organiz...
<input type="checkbox"/> Training: Cacheable Trait 🔗	1.0.1	Sep 10, 2019	★★★★★	MuleSoft Organiz...
<input type="checkbox"/> Cloud Information Model 🔗	0.1.0	Oct 25, 2019	★★★★★	MuleSoft Organiz...

Cancel Add 2 dependencies

- Click the Add 2 dependencies button.
- In the dependencies list, click the Training: American Flight Data Type option menu and review the menu options.



- Click the Files button (above the Exchange dependencies button).
- Expand the exchange_modules section until you see AmericanFlightDataType.raml.
- Click AmericanFlightDataType.raml and review the code.

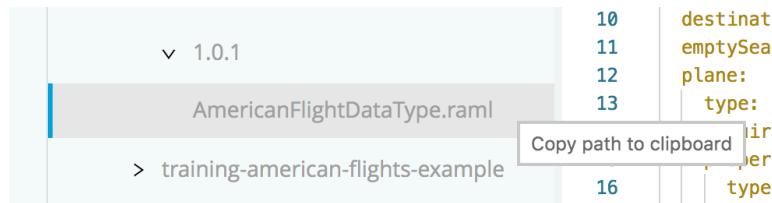
The screenshot shows the MuleSoft Anypoint Studio interface. At the top, there's a header bar with the title "Design Center". Below it, a message says "Saved 12 minutes ago". On the left, there's a sidebar with a "Files" section. Under "exchange_modules", there's a folder "68ef9520-24e9-4cf2-b2f5-620025690913" which contains a folder "training-american-flight-data-type" and a file "1.0.1/AmericanFlightDataType.raml". Other items in the sidebar include "training-american-flights-example" and "american-flights-api.raml". To the right of the sidebar is the main code editor area. The file "AmericanFlightDataType.raml" is open, showing the following RAML 1.0 code:

```

1  %%RAML 1.0 DataType
2
3  type: object
4  properties:
5    ID?: integer
6    code: string
7    price: number
8    departureDate: string
9    origin: string
10   destination: string
11   emptySeats: integer
12   plane:
13     type: object
14     required: false
15     properties:
16       type: string
17       totalSeats: integer
18

```

- In the file browser, click the options menu button next to AmericanFlightDataType.raml and select Copy path to clipboard.



Define an AmericanFlight data type for the API

- Return to american-flights-api.raml.
- Near the top of the code above the /flights resource, add a types element.
- Indent under types and add a type called AmericanFlight.
- Add the !include keyword and then paste the path you copied.

Note: You can also add the path by navigating through the exchange_modules folder in the shelf.

```

1  %%RAML 1.0
2  baseUri: https://anypoint.mulesoft.com/mockng/api/v1/links/e2ccb9f4-3851-44c1-a507-c2684c517b01/ #
3  title: American Flights API
4
5  types:
6    AmericanFlight: !include exchange_modules/68ef9520-24e9-4cf2-b2f5-620025690913/training-american-f
7
8  /flights:
9    get:

```

Specify the /flights:get method to return an array of AmericanFlight objects

15. Go to a new line of code at the end of the /flights get method and indent to the same level as queryParameters.

16. In the shelf, click responses > 200 > body > application/json > type > AmericanFlight.

Note: Enter the responses element manually if it doesn't appear in the shelf.

```
8  /flights:  
9    get:  
10   |   queryParameters:  
11   |   |   destination:  
12   |   |   |   required: false  
13   |   |   |   enum:  
14   |   |   |   |   - SFO  
15   |   |   |   |   - LAX  
16   |   |   |   |   - CLE  
17   |   responses:  
18   |   |   200:  
19   |   |   |   body:  
20   |   |   |   |   application/json:  
21   |   |   |   |   |   type: AmericanFlight
```

17. Set the type to be an array of AmericanFlight objects: AmericanFlight[].

```
17  |   responses:  
18  |   |   200:  
19  |   |   |   body:  
20  |   |   |   |   application/json:  
21  |   |   |   |   |   type: AmericanFlight[]  
~~
```

Add an example response for the /flights:get method

18. In the file browser, locate AmericanFlightsExample.raml in exchange_modules and review the code.

The screenshot shows the MuleSoft Anypoint Studio interface. On the left, the file browser displays a directory structure under 'exchange_modules'. It includes 'AmericanFlightData-Type.raml' and 'AmericanFlightsExample.raml'. On the right, the main editor window is titled 'American Flights A' and contains the following RAML code:

```
%RAML 1.0 NamedExample
value:
  -
    ID: 1
    code: ER38sd
    price: 400
    departureDate: 2017/07/26
    origin: CLE
    destination: SFO
    emptySeats: 0
    plane:
      type: Boeing 737
      totalSeats: 150
  -
    ID: 2
    code: ER45if
    price: 540.99
    departureDate: 2017/07/27
    origin: SFO
    destination: ORD
    emptySeats: 54
    plane:
      type: Boeing 777
      totalSeats: 300
```

19. In the file browser, click the options menu next to AmericanFlightsExample.raml and select Copy path to clipboard.
20. Return to american-flights-api.raml.
21. In the editor, go to a new line after the type declaration in the /flights:get 200 response (at the same indentation as type).
22. In the shelf, click examples.
23. Add a key called output.
24. Add the !include keyword and then paste the path you copied.

Note: You can also add the path by navigating through the exchange_modules folder in the shelf.

```
17  responses:
18  200:
19    body:
20      application/json:
21        type: AmericanFlight[]
22        examples:
23          output: !include exchange_modules/68ef9520-24e9-4cf2-b2f5-6200256909
24
25  post:
```

Review and test the /flights:get resource in API console

25. In API console, click the /flights:get method.

26. In the response information, look at the type information.

Array
ID
Integer
code
String Required
price
Number Required
departureDate
String Required
origin
String Required
destination
String Required

27. In the response information, ensure you see the example array of AmericanFlight objects.

Examples

output

 </>

```
[  
  {  
    "ID": 1,  
    "code": "ER38sd",  
    "departureDate":  
      "2017/07/26",  
    "destination": "SFO",  
    "emptySeats": 0,  
    "origin": "CLE",  
    "plane": {  
      "totalSeats": 150,  
      "type": "Boeing 737"  
    },  
    "price": 400  
  },  
  {  
    "ID": 2,  
    "code": "ER45if",  
    "departureDate":  
      "2017/07/27",  
    "destination": "ORD",  
    "emptySeats": 54,  
    "origin": "SFO",  
    "plane": {  
      "totalSeats": 300,  
      "type": "Boeing 777"  
    },  
  }]
```

28. Click the Try it button and click Send; you should now see the example response with two flights.

Specify the /{ID}:get method to return an AmericanFlight object

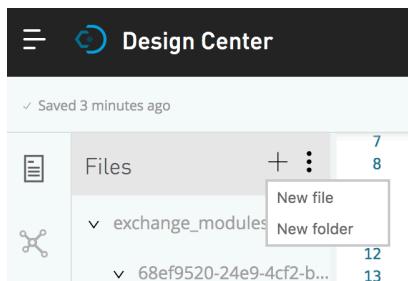
29. In the editor, indent under the /{ID} resource get method.

30. In the shelf, click responses > 200 > body > application/json > type > AmericanFlight.

```
27  [{  
28    "ID":  
29      get:  
30        responses:  
31          200:  
32            body:  
33              application/json:  
34                type: AmericanFlight
```

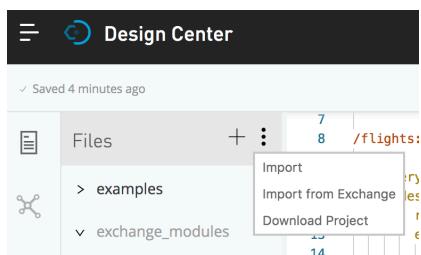
Define an example response for the /flights:get method in a new folder

31. In the file browser, click the add button and select New folder.



32. In the Add new folder dialog box, set the name to examples and click Create.

33. In the file browser, click the menu button and select Import.



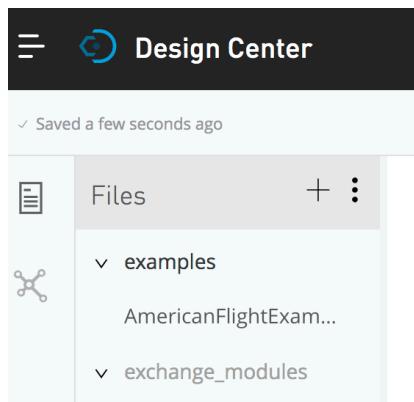
34. In the Import dialog box, leave File or ZIP selected and click the Choose file button.

35. Navigate to your student files and select the AmericanFlightExample.raml file in the resources/examples folder and click Open.

36. In the Import dialog box, click Import; you should see the new file in API Designer.

37. Review the code.

38. In the file browser, drag AmericanFlightExample.raml into the resources/examples folder.



Add an example response for the /{ID}:get method

39. Return to american-flights-api.raml.

40. In the editor, go to a new line after the type declaration in {ID}:/{get} (at the same indentation).

41. In the shelf, click examples.
42. Add a key called output.
43. Add an include statement and include the example in resources/examples/AmericanFlightExample.raml.

```

27   /{ID}:
28     get:
29       responses:
30         200:
31           body:
32             application/json:
33               type: AmericanFlight
34               examples:
35                 output: !include examples/AmericanFlightExample.raml

```

Review and test the /{ID}:get resource in API console

44. In API console, return to the /{ID}:get method; you should now see the response will be of type AmericanFlight.

Response

200

Body Hide ^

Media type: application/json

AmericanFlight

Examples

output

</>

```
{
  "ID": 1,
  "code": "ER38sd",
  "departureDate": "2017/07/26"
}
```

45. In the response information, ensure you see the example AmericanFlightExample data.

46. Click the Try it button, enter an ID, and click Send; you should now see the example flight data returned.

The screenshot shows the MuleSoft Anypoint Platform Mocking service interface. At the top, there's a shield icon and the text "Mocking service:" followed by a toggle switch with a checkmark. Below this, there are two tabs: "Parameters" (which is selected) and "Headers". Under "Parameters", there's a section for "URI parameters" with a field labeled "ID*" containing the value "10". Under "Query parameters", there's a button labeled "+ Add query parameter". Below these sections is a large blue "Send" button. After sending the request, the response is displayed: a green box indicating "200 OK" and "1327.27 ms". To the right of this, there's a "Details" link with a dropdown arrow. Below the status, there are four icons: a checkbox, a download arrow, a code editor icon, and a grid icon. The main content area shows a JSON response:

```
{  
    "ID": 1,  
    "code": "ER38sd",  
    "price": 400,  
    "departureDate": "2017/07/26",  
    "origin": "CLE",  
    "destination": "JFK",  
    "airline": "American",  
    "flightNumber": "38",  
    "gate": "A12",  
    "status": "On Time",  
    "estimatedArrival": "2017-07-26T15:45:00Z",  
    "estimatedDeparture": "2017-07-26T15:00:00Z",  
    "actualArrival": null,  
    "actualDeparture": null  
}
```

Specify the /flights:post method request to require an AmericanFlight object

47. In the editor, indent under the /flights post method.
48. In the shelf, click body > application/json > type > AmericanFlight.

```
25  |   post:  
26  |     body:  
27  |       application/json:  
28  |         type: AmericanFlight
```

Define an example request body for the /flights:post method

49. Return to AmericanFlightExample.raml and copy all the code.

50. In the file browser, click the add button next to the resources/examples folder and select New file.

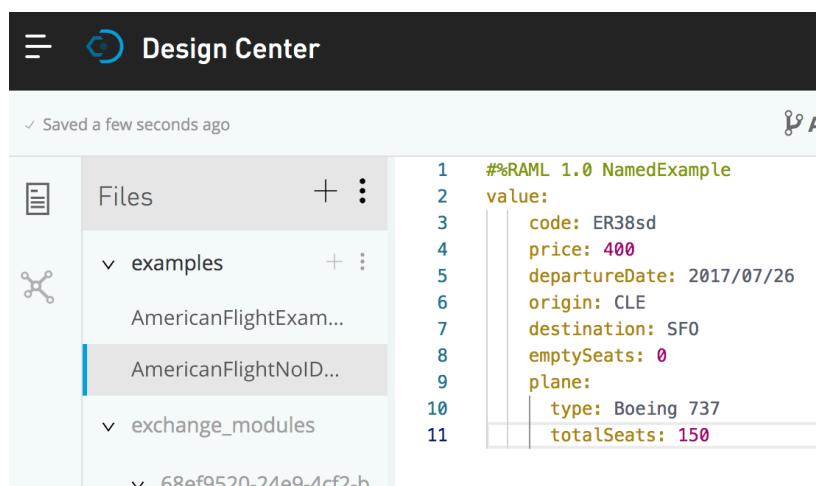
51. In the Add new file dialog box, set the following values:

- Version: RAML 1.0
- Type: Example
- File name: AmericanFlightNoIDExample.raml

52. Click Create.

53. Delete any code in the new file and then paste the code you copied.

54. Delete the line of code containing the ID.



The screenshot shows the MuleSoft Anypoint Studio interface. At the top, it says "Design Center". Below that, there's a message "Saved a few seconds ago". The left sidebar shows a tree view of files: "Files" (with a plus sign and three dots), "examples" (with a plus sign and three dots), and "examples/AmericanFlightExam..." (with a plus sign and three dots). Under "examples/AmericanFlightExam...", "AmericanFlightNoID..." is selected and highlighted with a blue bar. To the right of the tree view is the RAML code for "AmericanFlightNoID...". The code is as follows:

```
1  %%RAML 1.0 NamedExample
2  value:
3    code: ER38sd
4    price: 400
5    departureDate: 2017/07/26
6    origin: CLE
7    destination: SFO
8    emptySeats: 0
9    plane:
10   type: Boeing 737
11   totalSeats: 150
```

55. Return to american-flights-api.raml.

56. In the post method, go to a new line under type and add an examples element.

57. Add a key called input.

58. Add an include statement and include the example in

resources/examples/AmericanFlightNoIDExample.raml.

```
25  post:
26  body:
27  application/json:
28  type: AmericanFlight
29  examples:
30    input: !include examples/AmericanFlightNoIDExample.raml
```

Specify an example response for the /flights:post method

59. Go to a new line of code at the end of the /flights:post method and indent to the same level as body.

60. Add a 201 response of type application/json.

Note: Enter the responses element manually if it doesn't appear in the shelf.

```
25  post:
26    body:
27      application/json:
28        type: AmericanFlight
29        examples:
30          | input: !include examples/AmericanFlightNoIDExample.raml
31    responses:
32      201:
33        body:
34          application/json:
35
```

61. In the shelf, click example.

62. Indented under example, add a message property equal to the string: Flight added (but not really).

```
25  post:
26    body:
27      application/json:
28        type: AmericanFlight
29        examples:
30          | input: !include examples/AmericanFlightNoIDExample.raml
31    responses:
32      201:
33        body:
34          application/json:
35            example:
36              message: Flight added (but not really)
```

Review and test the /flights:post resource in API console

63. In API console, return to the /flights:post method.

64. Look at the request information; you should now see information about the body - that it is type AmericanFlight and it has an example.

Media type: application/json

AmericanFlight

Examples

input

 </>

```
{
  "code": "ER38sd",
  "departureDate": "2017/07/26",
  "destination": "SFO",
  "emptySeats": 0,
  "origin": "CLE",
  "plane": {
    "totalSeats": 150,
    "type": "Boeing 737"
  },
  "price": 400
}
```

65. Click the Try it button and select the Body tab again; you should now see the example request body.

Parameters Headers Body

Media type: application/json

Format JSON Minify JSON

```
{
  "code": "ER38sd",
  "departureDate": "2017/07/26",
  "destination": "SFO",
  "emptySeats": 0,
  "origin": "CLE",
  "plane": {
    "totalSeats": 150,
    "type": "Boeing 737"
  },
  "price": 400
}
```

Send

Note: Depending on your version of the API Designer, you may see double quotes around the values for the numeric properties of emptySeats, totalSeats, and price. If so, remove the double quotes around the 3 values.

66. Click the Send button; you should now get a 201 response with the example message.

The screenshot shows a REST API response. At the top, it says "201 Created" and "1161.64 ms". There are icons for copy, download, and refresh. Below that is a JSON response:

```
{  
  "message": "Flight added (but not  
  really)"  
}
```

67. In the body, remove the emptySeats property.

The screenshot shows a JSON editor interface. It has tabs for "Parameters", "Headers", and "Body", with "Body" selected. A checked checkbox indicates "Media type: application/json". Below that are "Format JSON" and "Minify JSON" buttons. The JSON body is shown in a code editor-like area:

```
{  
  "code": "ER38sd",  
  "departureDate": "2017/07/26",  
  "destination": "SFO",  
  "origin": "CLE",  
  "plane": {  
    "totalSeats": 150,  
    "type": "Boeing 737"  
  },  
  "price": 400  
}
```

68. Click Send again; you should get a 400 Bad Request response and a message that the emptySeats key is required.

The screenshot shows a REST API response. At the top, it says "400 Bad Request" and "709.40 ms". There are icons for copy, download, and refresh. Below that is an error message:

```
{  
  "code": "REQUEST_VALIDATION_ERROR",  
  "message": "Invalid schema for  
  content type application/json.  
  Errors: required key [emptySeats]  
  not found. "  
}
```

69. Add the emptySeats property back to the body.

Walkthrough 3-4: Add an API to Anypoint Exchange

In this walkthrough, you make an API discoverable by adding it to Anypoint Exchange. You will:

- Publish an API to Exchange from API Designer.
- Review an auto-generated API portal in Exchange and test the API.
- Add information about an API to its API portal.
- Create and publish a new API version to Exchange.

The screenshot shows the Anypoint Exchange interface with the following details:

- API Details:** American Flights API (v1)
- Description:** Add description
- Summary:** The American Flights API is a system API for operations on the **american** table in the **training** database.
- Supported operations:**
 - Get all flights
 - Get a flight with a specific ID
 - Add a flight
 - Delete a flight
 - Update a Flight
- Endpoints:** /flights
 - GET
 - POST
- Types:** /{ID}
- Other Details:** API Instances
- Reviews:** Be the first to review American Flights API
- Asset versions for v1:**

Version	Instances
1.0.1	Mocking Service
1.0.0	

[Add new version](#)
- Tags:** + Add a tag
- Dependencies:** Training: American Flights

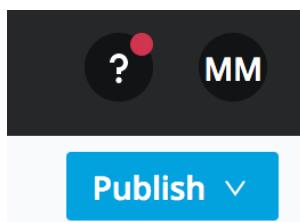
Remove the `baseUri` by turning off the mocking service

1. Return to API Designer.
2. Click the slider to turn off the mocking service; the RAML code should no longer have a `baseUri`.

```
1  #%RAML 1.0
2  title: American Flights API
3
```

Publish the API to Anypoint Exchange from API Designer

3. Click the Publish button.



4. Click the Publish to Exchange button.
5. In the Publish API specification to Exchange dialog box, leave these default values:
 - Name: American Flights API
 - Main file: american-flights-api.raml
 - Asset version: 1.0.0
6. Set API version to v1.
7. Click the Show advanced link.
8. Look at the ID values.
9. Click Publish.

Publish API specification to Exchange

Name (required)	Asset version ⓘ (required)
American Flights API	1.0.0
Main file (required)	API version ⓘ (required)
american-flights-api.raml	v1

Valid Specification

[Hide advanced ▾](#)

Group id

c66beee7-52a3-4c79-a7be-5b56308f2850

Asset id (required)

american-flights-api

[Cancel](#) [Publish](#)

10. In the Publish API specification to Exchange dialog box, click Done.
11. Look at the RAML file; you should see a version has been added.

```

1  #%RAML 1.0
2  version: v1
3  title: American Flights API

```

Locate your API in Anypoint Exchange

12. Return to Design Center.

13. In the main menu, select Exchange; you should see your American Flights API.

The screenshot shows the Exchange interface with the title bar "Exchange". On the left, there is a sidebar with navigation options: "All assets", "Training (master)", "Provided by MuleSoft", "My applications", "Public portal", and "Settings". The main area is titled "Assets" and contains a search bar with "Search" and a dropdown menu set to "All types". A card for the "American Flights API" is displayed, showing it is a REST API with a green icon, five stars, and the developer "Max Mule".

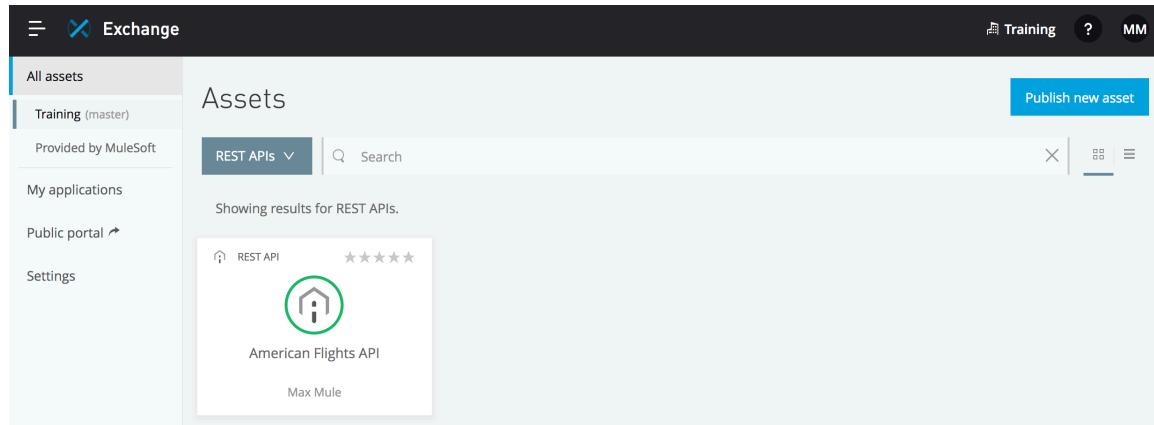
14. In the types drop-down menu, select REST APIs; you should still see your API.

The screenshot shows the Exchange interface with the title bar "Exchange". The sidebar remains the same. The main area is titled "Assets" and has a dropdown menu set to "REST APIs". Below it, a message says "Showing results for REST APIs." A card for the "American Flights API" is shown, identical to the one in the previous screenshot.

15. In the left-side navigation, select Provided by MuleSoft; you should not see your American Flights API in the public Exchange (just the Training: American Flights API).

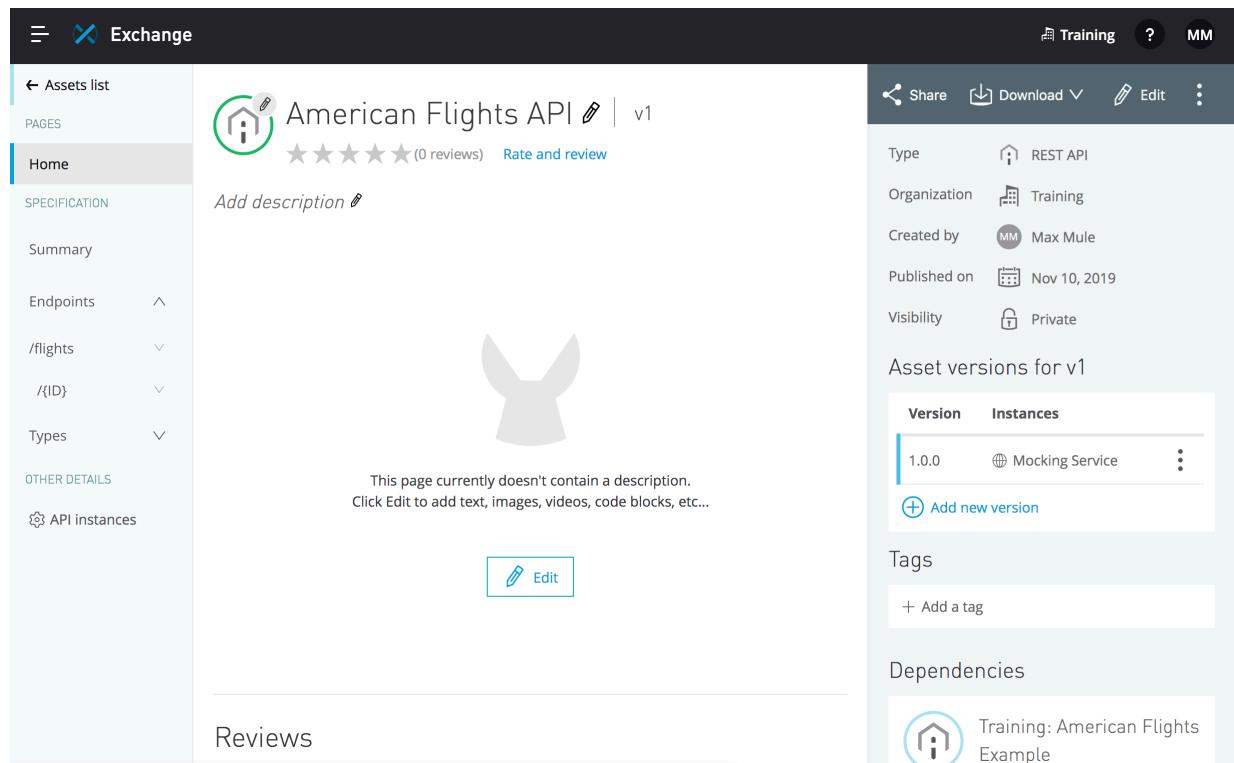
The screenshot shows the Exchange interface with the title bar "Exchange". The sidebar now has a selected option "Provided by MuleSoft". The main area is titled "Assets provided by MuleSoft" and contains a dropdown menu set to "REST APIs". Below it, a message says "Showing results for REST APIs." A grid of cards for various MuleSoft APIs is shown, including the "Appian API", "Omnichannel Experience API | RAML Definition", "Order Fulfillment API | RAML Definition", and "Shopping Cart API | RAML Definition", all from the "MuleSoft Organization". The "American Flights API" is not visible in this list.

16. In the left-side navigation, select the name of your organization (Training in the screenshots); you should see your American Flights API in your private Exchange.



Review the auto-generated API portal

17. Click the American Flights API.
18. Review the page; you should see that as the creator of this API, you can edit, review, share, download, and add tags to this version.



19. Locate the API dependencies in the lower-right corner.

20. Locate the API version (v1) next to the name of the API at the top of the page.

The screenshot shows the MuleSoft Exchange interface. On the left, there's a navigation sidebar with 'Assets list' and 'PAGES' sections, and a 'SPECIFICATION' section containing 'Home' and 'Add description'. The main area displays the 'American Flights API' with a green icon, a rating of 5 stars (0 reviews), and a 'Rate and review' button. The API version 'v1' is shown to the right of the API name.

21. Locate the asset versions for v1; you should see one version (1.0.0) of this API specification (v1) has been published and there is one API instance for it and that uses the mocking service.

The screenshot shows a table titled 'Asset versions for v1'. It has two columns: 'Version' and 'Instances'. One row is listed: '1.0.0' with 'Mocking Service' under 'Instances' and three vertical dots for more options.

Version	Instances
1.0.0	Mocking Service

22. In the left-side navigation, select API instances; you should see information for the API instance generated from the API specification using the mocking service.

The screenshot shows the 'API instances' page. The left sidebar includes 'Assets list', 'PAGES', 'Home', 'SPECIFICATION' (with 'Summary', 'Endpoints', '/flights', '/{ID}', 'Types', and 'OTHER DETAILS' sections), and 'API instances' (which is selected). The main area shows a table with columns 'Instances', 'Environment', 'URL', and 'Visibility'. One instance is listed: 'Mocking Service' with URL 'https://anypoint.mulesoft.com/mockng/api/v1/sources/exchange/assets/942dc757-7e9d-417f-a9f5-9e906c0c173c/american-flights-api/1.0.0/m' and 'Public' visibility. There's also a '+ Add new instance' button.

Instances	Environment	URL	Visibility
Mocking Service		https://anypoint.mulesoft.com/mockng/api/v1/sources/exchange/assets/942dc757-7e9d-417f-a9f5-9e906c0c173c/american-flights-api/1.0.0/m	Public

23. In the left-side navigation, expand Types.

24. Select AmericanFlight and review the information.

Note: Depending on your version of the API Designer, you may see more than one AmericanFlight data type.

American Flights API | v1

★★★★★ (0 reviews) Rate and review

AmericanFlight

Examples

input

```
code: ER38sd
price: 400
departureDate: 2017/07/26
origin: CLE
destination: SFO
emptySeats: 0
plane:
  type: Boeing 737
  totalSeats: 150
```

Test the API in its API portal in Exchange

25. In the left-side navigation, select the /flights GET method; you should now see the API console on the right side of the page.
26. In the API console, select to show optional query parameters.
27. Click the destination drop-down and select a value.

GET /flights

Code examples Show ▾

Query parameters Hide ▾

destination

String Enum

Enum values:

- SFO
- LAX
- CLE

Response

Mocking Service

https://anypoint.mulesoft.com/mockng/api/v1/sources/exchange/assets/942dc757-7e9d-417f-a9f5-9e906c0173c/american-flights-api/1.0.0/m/flights?destination=LAX

Parameters Headers

destination LAX

Show optional parameters

Add query parameter

Send

28. Click Send; you should get a 200 response and the example data displayed – just as you did in API console in API Designer.

200 OK 58.91 ms

Array[2]

```
-0: {
    "ID": 1,
    "code": "ER38sd",
    "price": 400,
    "departureDate": "2017/07/26",
    "origin": "CLE",
    "destination": "SFO",
    "emptySeats": 0,
    -"plane": {
        "type": "Boeing 737",
        "totalSeats": 150
    }
},
-1: {
    "ID": 2,
    "code": "ER45if",
    "price": 540.99
}
```

Add information about the API

29. In the left-side navigation, select Home.
30. Click the Edit button for the API near the center.



This page currently doesn't contain a description.
Click Edit to add text, images, videos, code blocks, etc...



31. Return to the course snippets.txt file and copy the text for the American Flights API description text.
32. Return to the editor in Anypoint Exchange and paste the content.
33. Select the words american table and click the strong button (the B).

The American Flights API is a system API for operations on the **american table** in the training database.
 Supported operations
 Get all flights
 Get a flight with a specific ID
 Add a flight

34. Select the words training database and click the emphasis button (the I).

The American Flights API is a system API for operations on the **american table** in the _training database_.
 Supported operations
 Get all flights
 Get a flight with a specific ID
 Add a flight

35. Select the words Supported operations and select Heading 4 from the heading drop-down menu (the H).

The American Flights API is a system API for operations on the **american table** in the _training database_.
 #### Supported operations
 Get all flights
 Get a flight with a specific ID
 Add a flight

36. Select Get all flights and click the bulleted list button.
37. Repeat for the other operations.

The American Flights API is a system API for operations on the **american table** in the _training database_.
 #### Supported operations

- Get all flights
- Get a flight with a specific ID
- Add a flight

38. Select the Visual tab in the editor toolbar and view the rendered markdown.

The screenshot shows the Exchange API editor interface. On the left, there's a sidebar with navigation links like 'Assets list', 'PAGES', 'Home' (which is selected), '+ Add new page', 'SPECIFICATION', 'Summary', 'OTHER DETAILS', '+ Add terms and conditions', and 'API instances'. The main content area has a title 'American Flights API | v1' with a house icon. Below it is a text input field 'Add description' with a pencil icon. A toolbar above the text area contains icons for bold, italic, code, list, and other document operations. To the right of the toolbar are buttons for 'Markdown' and 'Visual' (which is highlighted). The visual content displays the rendered markdown of the API description. It includes a summary paragraph about the American Flights API being a system API for operations on the 'american table' in the 'training database'. Below this is a section titled 'Supported operations' with a bulleted list: 'Get all flights', 'Get a flight with a specific ID', and 'Add a flight'. At the bottom right are 'Discard changes' and 'Save as draft' buttons. The URL 'https://anvpoint.mulesoft.com/exchange/' is visible at the bottom left.

39. Click the Save as draft button; you should now see buttons to discard the draft, exit the draft, or publish it.

This screenshot shows the Exchange API editor after saving the API as a draft. The top bar now indicates 'This is a draft that has not been published yet.' Below the title, there are 'Discard draft' and 'Exit draft' buttons. The right side of the screen shows the API details: Type (REST API), Organization (Training), Created by (Max Mule), Published on (Nov 10, 2019), and Visibility (Private). There's also a section for 'Asset versions for v1' with tabs for 'Version' and 'Instances'.

40. Click the Publish button; you should now see the new information about the API in the API portal.

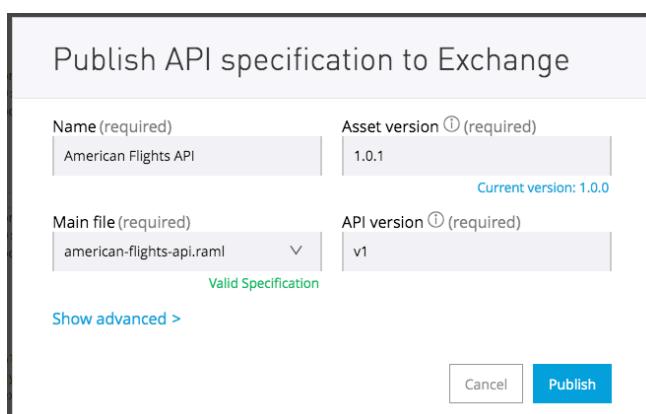
This screenshot shows the Exchange API editor after publishing the API. The top bar now indicates 'Training' and has 'Share', 'Download', 'Edit', and a more options menu. The right side of the screen shows the published API details: Type (REST API), Organization (Training), Created by (Max Mule), Published on (Nov 10, 2019), and Visibility (Private). There's also a section for 'Asset versions for v1' with tabs for 'Version' and 'Instances'.

Modify the API and publish the new version to Exchange

41. Return to your American Flights API in API Designer.
42. Return to the course snippets.txt file and copy the American Flights API - /{ID} DELETE and PUT methods.
43. Return to american-flights-api.raml and paste the code after the {ID}/get method.
44. Fix the indentation if necessary.
45. Review the code.

```
38  [{ID}:
39    get:
40      responses:
41        200:
42          body:
43            application/json:
44              type: AmericanFlight
45              examples:
46                output: !include examples/AmericanFlightExample.raml
47    delete:
48      responses:
49        200:
50          body:
51            application/json:
52              example:
53                message: Flight deleted (but not really)
54    put:
55      body:
56        application/json:
57        type: AmericanFlight
58        examples:
59        input: !include examples/AmericanFlightNoIDExample.raml
60      responses:
61        200:
62          body:
63            application/json:
64              example:
65                message: Flight updated (but not really)
```

46. Click the Publish button.
47. Click the Publish to Exchange button
48. In the Publish API specification to Exchange dialog box, look at the asset version.



49. Click Publish.
50. In the Publish API specification to Exchange dialog box, click Exchange; Exchange should open in a new browser tab.
51. Locate the asset versions listed for the API; you should see both asset versions of the API listed with an associated API instance using the mocking service for the latest version.

Asset versions for v1		
Version	Instances	
1.0.1	Mocking Service	⋮
1.0.0		⋮

52. Click the Edit button for the API near the upper-right corner.



53. Add two new operations that delete a flight and update a flight.

The American Flights API is a system API for operations on the `american` table in the `_training database`.

Supported operations

- Get all flights
- Get a flight with a specific ID
- Add a flight
- Delete a flight
- Update a flight

54. Click Save as draft and then Publish.

The screenshot shows the MuleSoft Exchange interface for the "American Flights API".

Left Sidebar: Assets list, American Flights API selected.

Top Bar: Exchange, Training, MM.

Asset Details:

- Name:** American Flights API | v1
- Rating:** ★★★★☆ (0 reviews) | Rate and review
- Description:** The American Flights API is a system API for operations on the **american** table in the **training** database.
- Supported operations:**
 - Get all flights
 - Get a flight with a specific ID
 - Add a flight
 - Delete a flight
 - Update a flight
- Endpoints:** /flights, /(ID)
- Types:**
- API instances:**

Right Panel:

- Type:** REST API
- Organization:** Training
- Created by:** Max Mule
- Published on:** May 6, 2019
- Visibility:** Private
- Asset versions for v1:**

Version	Instances
1.0.1	Mocking Service
1.0.0	

[Add new version](#)
- Tags:** + Add a tag
- Dependencies:**
 - Training: American Flights Example 1.0.1 (API Spec Fragment)
 - Training: American Flight Data Type 1.0.1 (API Spec Fragment)

Walkthrough 3-5: Share an API

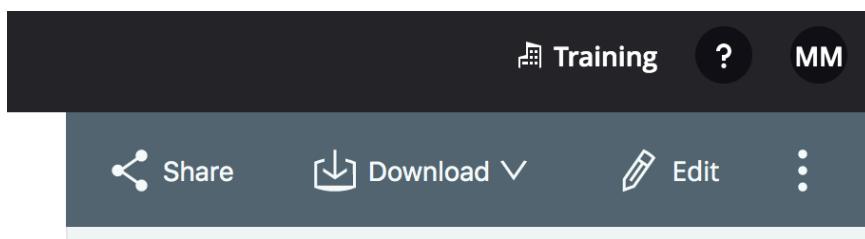
In this walkthrough, you share an API with both internal and external developers to locate, learn about, and try out the API. You will:

- Share an API within an organization using the private Exchange.
- Create a public API portal.
- Customize a public portal.
- Explore a public portal as an external developer.

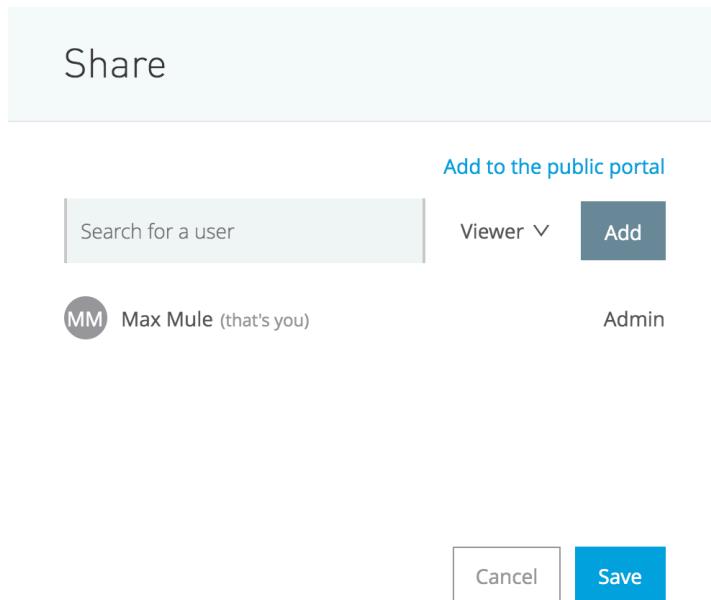
The screenshot shows the MuleSoft Training portal homepage. At the top, there's a navigation bar with the MuleSoft logo, the text "MuleSoft // Training", a "Home" link, and a "Login" button. Below the header, a large banner features the text "Welcome to your MuleSoft Training portal!" and a subtext: "Build your application network faster! Get started with powerful tools, intuitive interface, and best in class documentation experience." In the main content area, there's a section titled "Assets" with a search bar and a dropdown menu. A card for the "American Flights API" is displayed, showing it's a REST API with a 5-star rating and a house icon.

Share the API in the private Exchange with others

1. Return to your American Flights API in Exchange.
2. Click Share.



3. In the Share dialog box, you should see that you are an Admin for this API.



4. Open the Viewer drop-down menu located next to the user search field; you should see that you can add additional users to be of type Viewer, Contributor, or Admin.

Note: In the future, you will also be able to share an asset with an entire business group.

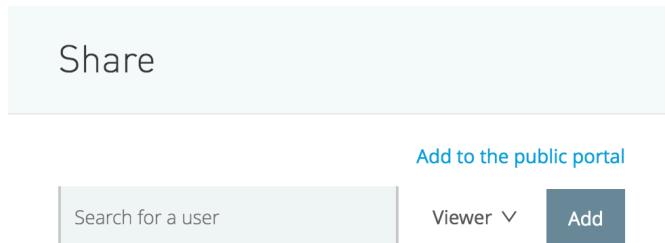
5. Click Cancel.
6. In the left-side navigation, click Assets list.
7. In the left-side navigation, select the name of your organization.
8. Click your American Flights API.

Note: This is how users you share the API with will find the API.

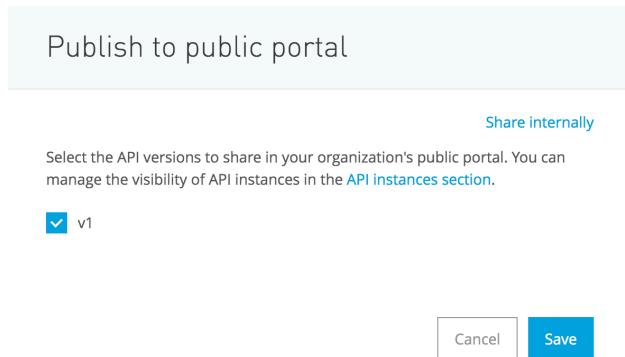
The screenshot shows the 'Exchange' interface with the 'Assets' list open. On the left, a sidebar lists 'All assets', 'Training (master)', 'Provided by MuleSoft', 'My applications', 'Public portal', and 'Settings'. The main area displays 'Training assets (master)' with a 'REST API' asset titled 'American Flights API' by 'Max Mule'. A 'Publish new asset' button is visible at the top right. The interface has a dark theme with light-colored cards for assets.

Create a public API portal

9. Click the Share button for the API again.
10. In the Share dialog box, click Add to the public portal.



11. In the Publish to public portal dialog box, select to share v1 of the API.
12. Click Save.



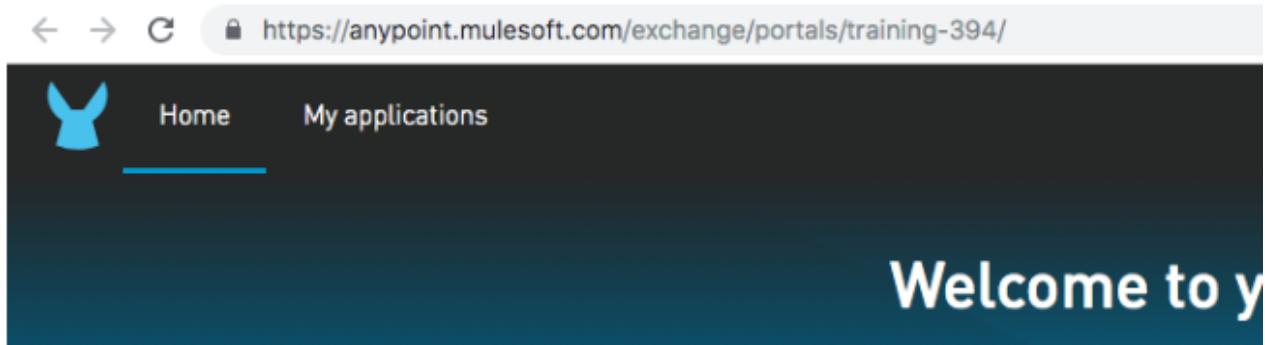
Explore the API in the public portal

13. In the left-side navigation, click Assets list.
14. In the left-side navigation, select Public Portal.

A screenshot of the MuleSoft Exchange interface. The top navigation bar includes 'Exchange', 'Training', a help icon, and a 'MM' icon. On the left is a sidebar with 'All assets' (selected), 'Training (master)', 'Provided by MuleSoft', 'My applications', 'Public portal ▾' (selected), and 'Settings'. The main area shows 'Training assets (master)' with a search bar and a 'Publish new asset' button. A card for the 'American Flights API' is displayed, showing it's a REST API with a green star rating and a house icon. The card also lists 'Max Mule'.

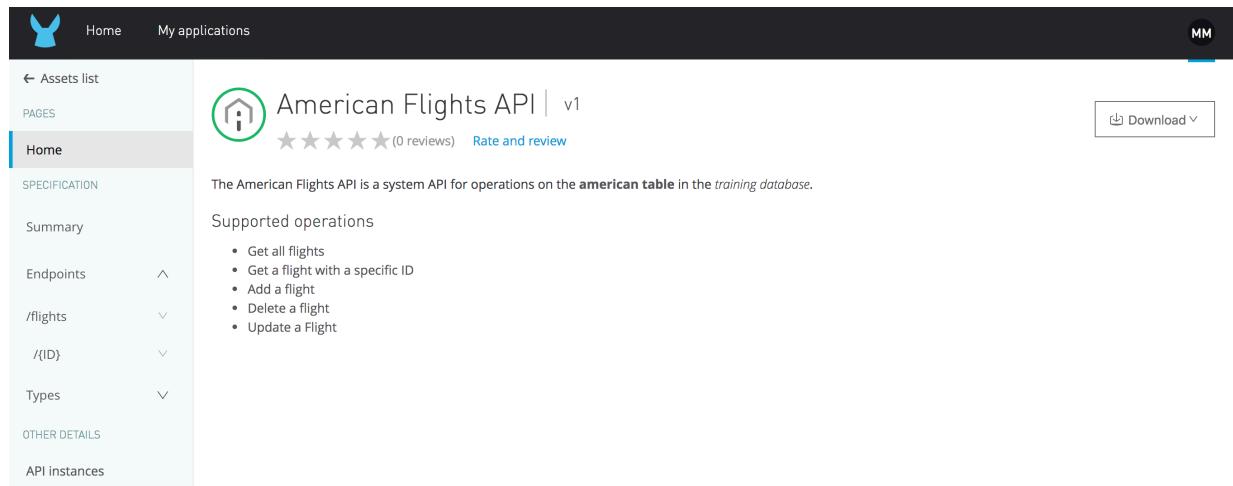
15. Review the public portal that opens in a new browser tab.

16. Look at the URL for the public portal.



The screenshot shows a web browser window with the URL <https://anypoint.mulesoft.com/exchange/portals/training-394/>. The page has a dark blue header with a blue icon on the left, followed by 'Home' and 'My applications'. Below the header, the text 'Welcome to y' is visible. The browser's address bar and navigation buttons are also shown.

17. Click your American Flights API and review the auto-generated public API portal.



The screenshot shows a detailed view of the American Flights API documentation. On the left, a sidebar lists 'Assets list', 'PAGES', 'SPECIFICATION' (which is expanded to show 'Home', 'Summary', 'Endpoints', '/flights', '/{ID}', 'Types', and 'OTHER DETAILS'), and 'API instances'. The main content area shows the 'American Flights API | v1' page, which includes a 'Rate and review' button and a 'Download' button. It describes the API as a system API for operations on the `american` table in the `training` database. A section titled 'Supported operations' lists the following methods:

- Get all flights
- Get a flight with a specific ID
- Add a flight
- Delete a flight
- Update a Flight

18. In the left-side navigation, click the GET method for the flights resource.

19. Review the response information.

20. In the API console, click Send; you should get a 200 response with example flights returned from the mocking service.

The screenshot shows the MuleSoft Anypoint Platform interface. On the left, there's a navigation sidebar with options like 'Assets list', 'Home', 'SPECIFICATION', 'Summary', 'Endpoints' (expanded), 'Overview' (selected), 'GET /flights' (selected), 'POST /{ID}', 'Types', and 'OTHER DETAILS'. The main area displays the 'American Flights API | v1' with a green house icon. It shows a 5-star rating (0 reviews) and a 'Rate and review' button. Below this is a 'GET /flights' button with a checked checkbox. To the right, there's a 'Mocking Service' section with a URL: <https://anypoint.mulesoft.com/mockng/api/v1/sources/exchange/assets/942dc757-7e9d-417f-a9f5-9e906c0c173c/american-flights-api/1.0.1/m/flights>. It includes sections for 'Parameters', 'Query parameters' (with an 'Add query parameter' button), and a 'Send' button. A successful response is shown: '200 OK' with a time of '157.31 ms' and a link to 'Details'. The response body is displayed as JSON:

```
[{"id": 1, "code": "ER38sd", "price": 400, "departDate": "2017/07/26"}]
```

Customize the public portal

21. In the left-side navigation, click Assets list.
22. Click the Customize button.

The screenshot shows the MuleSoft Anypoint Platform developer portal. At the top, there's a navigation bar with 'Home' and 'My applications' on the left, and 'Customize' (with a pencil icon) and 'MM' on the right. The main content area has a dark blue gradient background. It features a large white text 'Welcome to your developer portal!' and a smaller text below it: 'Build your application network faster! Get started with powerful tools, intuitive interface, and best in class documentation experience.'

23. In the Title field, change the text to Welcome to your MuleSoft Training portal!

The screenshot shows the MuleSoft Studio interface. On the left, there's a preview window displaying a dark-themed landing page with a blue header bar. The header bar contains the MuleSoft logo, a 'Home' button, and a 'My applications' link. Below the header is a top bar with a trash icon, 'Discard changes', and a checkmark icon for 'Done editing'. The main content area features a title 'Welcome to your MuleSoft Training portal!' and a subtitle 'Build your application network faster! Get started with powerful tools, intuitive interface, and best in class documentation experience.' To the right of the preview is a sidebar titled 'Assets' which lists an 'American Flights API' item. The right side of the interface is a configuration panel for the 'Top bar' and 'Welcome section'. Under 'Top bar', there are fields for 'Logo' (with a 'Choose file' button) and 'Favicon' (with a 'Choose file' button). Under 'Background color', a color swatch is set to #262728. Under 'Text color' and 'Text color (active)', color swatches are set to #FFFFFF and #00A2DF respectively. In the 'Welcome section', there are fields for 'Title' (containing 'Welcome to your MuleSoft Training portal!') and 'Subtitle'.

24. In the logo field, click Choose file.

25. In the file browser dialog box, navigate to the student files and locate the MuleSoft_training_logo.png file in the resources folder and click Open.

26. Locate the new logo in the preview.

27. In the hero image field, click Choose file.

28. In the file browser dialog box, navigate to the student files and locate the banner.jpg file in the resources folder and click Open.

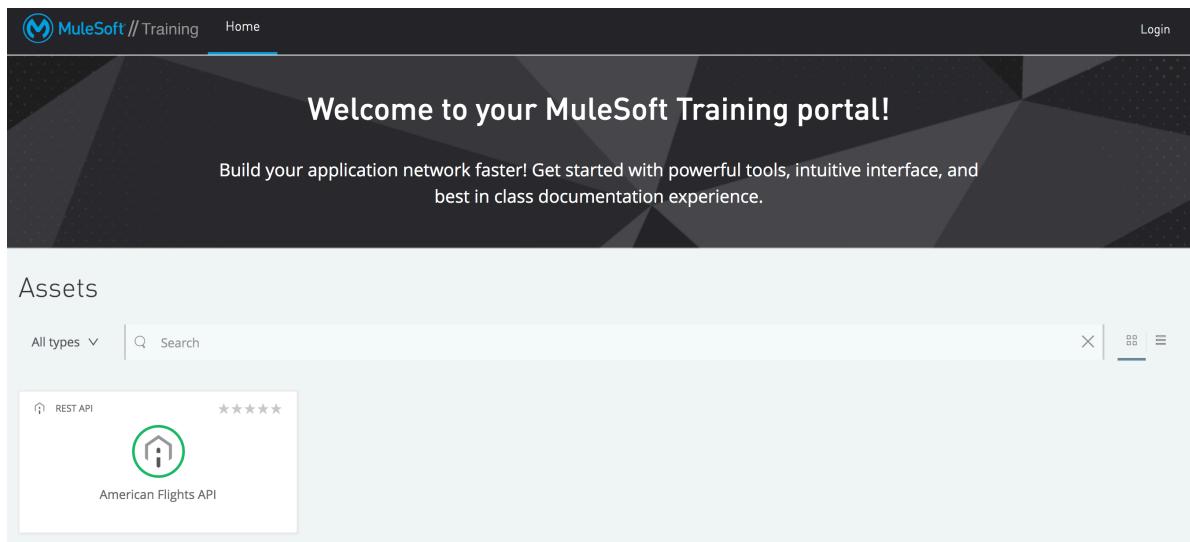
29. Review the new logo and banner in the preview.

This screenshot shows the same MuleSoft Studio interface after the changes from step 29 have been applied. The preview window now displays the updated landing page. The 'Top bar' configuration has been updated to use the 'MuleSoft_training_logo.png' file for the logo. The 'Welcome section' configuration remains the same as in the previous screenshot. The 'Assets' sidebar still shows the 'American Flights API' item. The right-hand configuration panel is identical to the one in the previous screenshot, reflecting the completed changes.

30. Change any colors that you want.
31. Click the Done editing button.
32. In the Publish changes dialog box, click Yes, publish; you should see your customized public portal.

Explore the public portal as an external developer

33. In the browser, copy the URL for the public portal.
34. Open a new private or incognito window in your browser.
35. Navigate to the portal URL you copied; you should see the public portal (without the customize button).



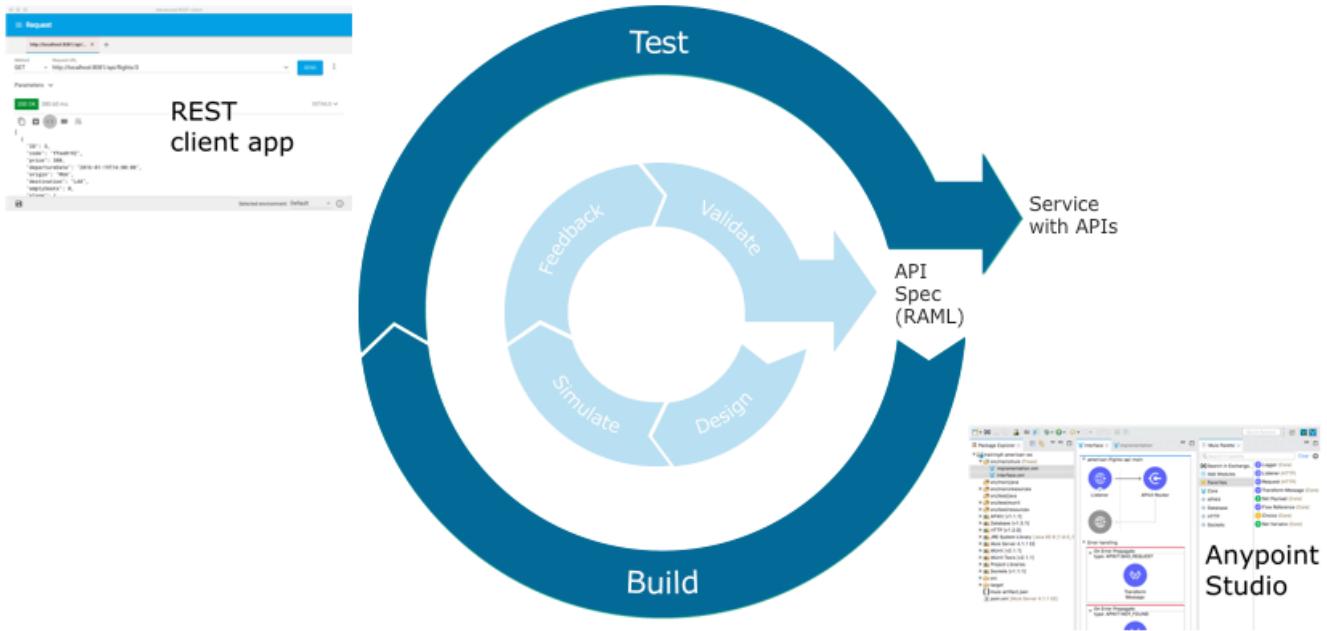
36. Click the American Flights API.
37. Explore the API portal.

38. Make a call to one of the resource methods.

Note: As an anonymous user, you can make calls to an API instance that uses the mocking service but not managed APIs.

The screenshot shows the MuleSoft Anypoint Platform interface. On the left, there's a sidebar with navigation links like 'Assets list', 'PAGES', 'Home', 'SPECIFICATION', 'Summary', 'Endpoints', 'Overview' (which is selected), 'Types', and 'OTHER DETAILS'. Under 'Overview', there are buttons for 'GET', 'DELETE', and 'PUT'. The main content area displays the 'American Flights API | v1' endpoint. It shows a 'GET /flights/{ID}' method. Below it, there are sections for 'Code examples', 'URI parameters', and 'ID' (with 'String Required'). The 'Response' section shows a 200 status code. Under 'Body', it says 'Media type: application/json' and shows a sample response: { "ID": 1, "code": "ER38sd", "price": 400 }. On the right side, there are buttons for 'Download', 'View code', and 'Mocking Service' (which is expanded, showing the URL https://anypoint.mulesoft.com/mockng/api/v1/sources/exchange/assets/942dc757-7e9d-417f-a9f5-9e906c0c173c/american-flights-api/1.0.1/m/flights/10). There are also tabs for 'Parameters' and 'Headers', and a 'Send' button. At the bottom, it shows '200 OK' status, '273.61 ms' duration, and a 'Details' link.

Module 4: Building APIs



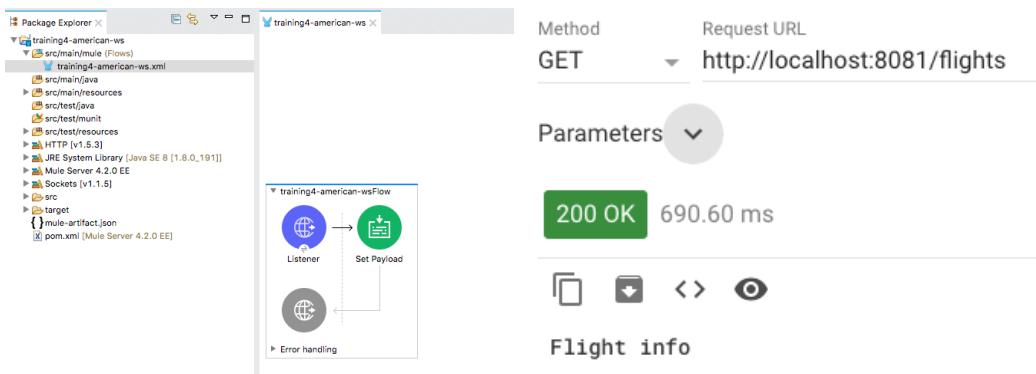
At the end of this module, you should be able to:

- Use Anypoint Studio to build, run, and test Mule applications.
- Use a connector to connect to databases.
- Use the graphical DataWeave editor to transform data.
- Create RESTful interfaces for applications from RAML files.
- Connect API interfaces to API implementations.

Walkthrough 4-1: Create a Mule application with Anypoint Studio

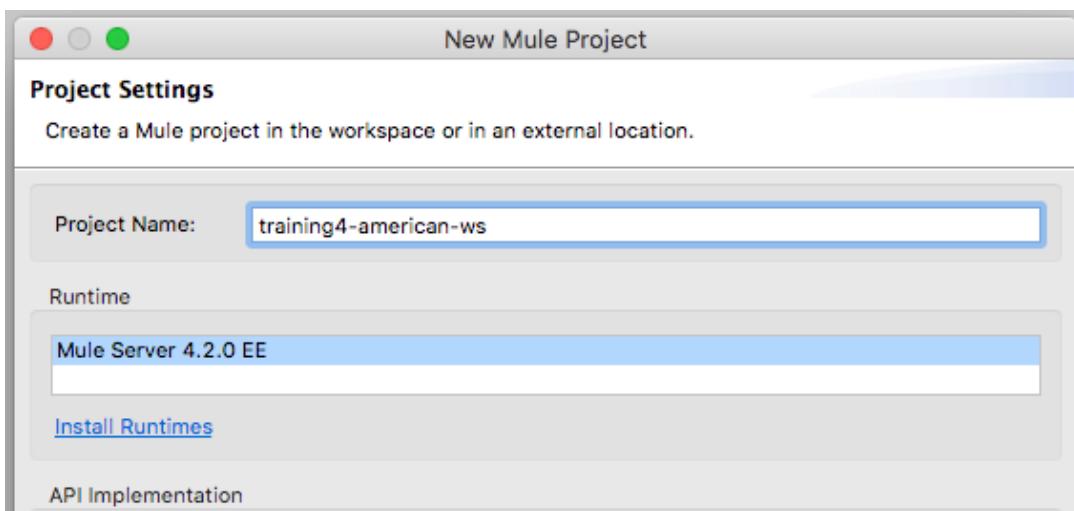
In this walkthrough, you build a Mule application. You will:

- Create a new Mule project with Anypoint Studio.
- Add a connector to receive requests at an endpoint.
- Set the event payload.
- Run a Mule application using the embedded Mule runtime.
- Make an HTTP request to the endpoint using Advanced REST Client.



Create a Mule project

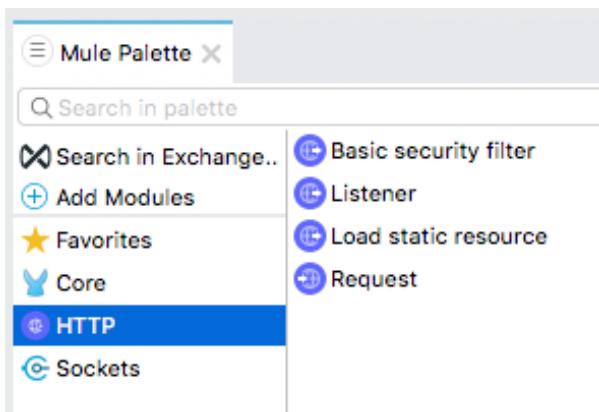
1. Open Anypoint Studio.
2. Select File > New > Mule Project.
3. In the New Mule Project dialog box, set the Project Name to training4-american-ws.
4. Ensure the Runtime is set to the latest version of Mule.



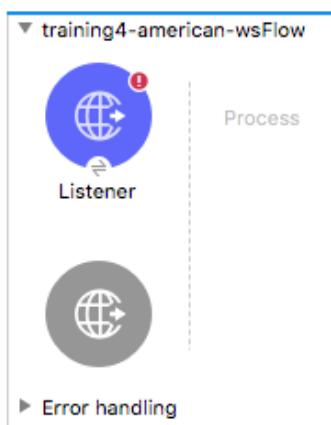
5. Click Finish.

Create an HTTP connector endpoint to receive requests

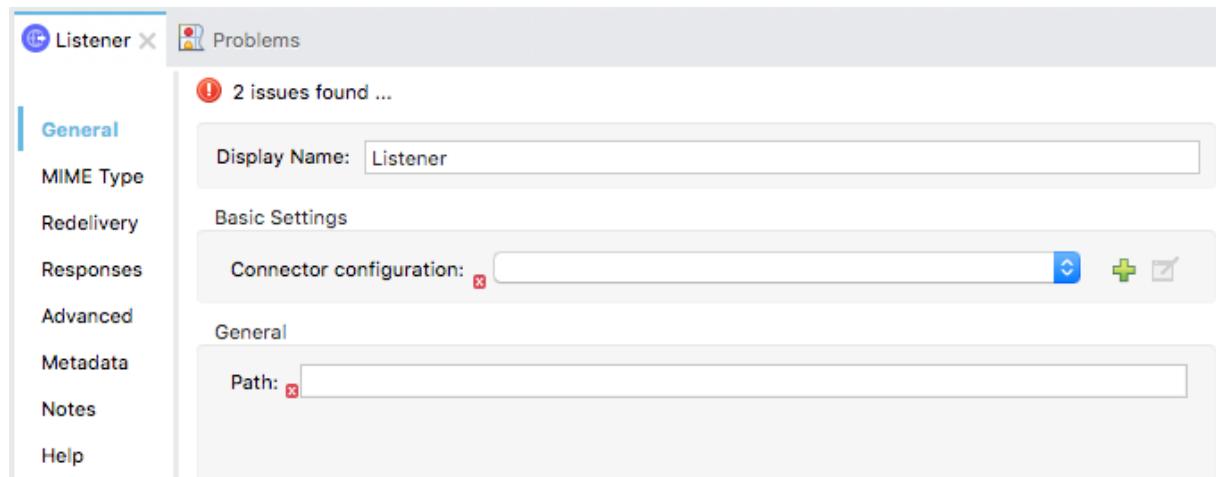
6. In the Mule Palette, select the HTTP module.



7. Drag the Listener operation from the Mule Palette to the canvas.

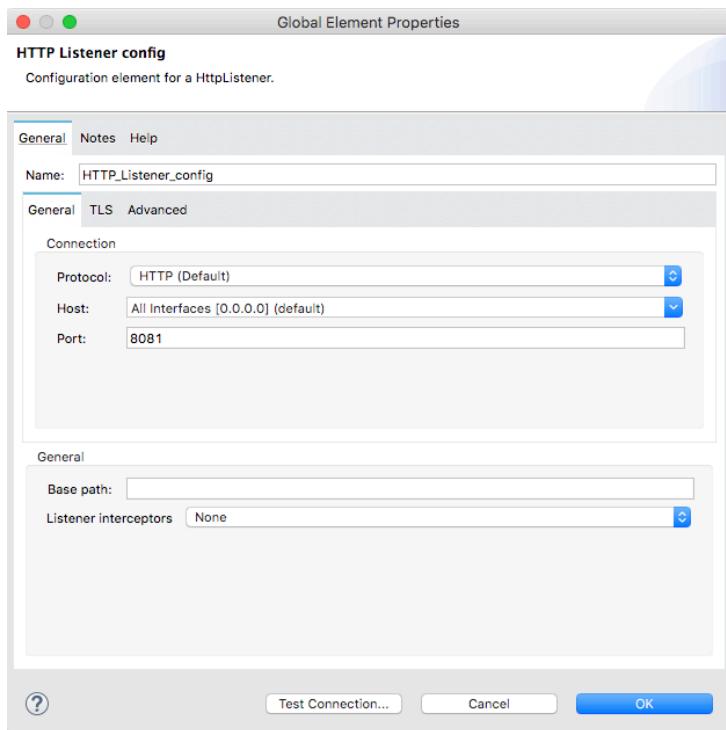


8. Double-click the HTTP Listener endpoint.
9. In the Listener properties view that opens at the bottom of the window, click the Add button next to connector configuration.



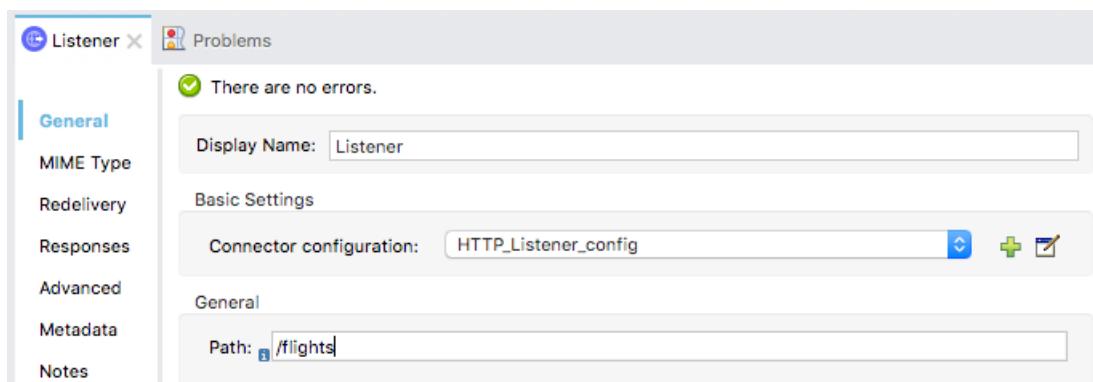
10. In the Global Element Properties dialog box, verify the following default values are present.

- Host: 0.0.0.0
- Port: 8081

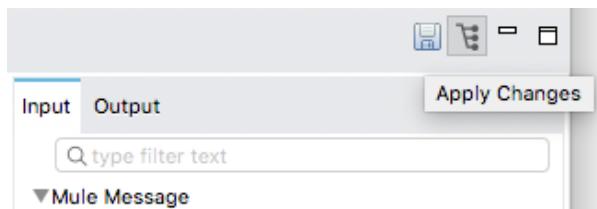


11. Click OK.

12. In the Listener properties view, set the path to /flights.

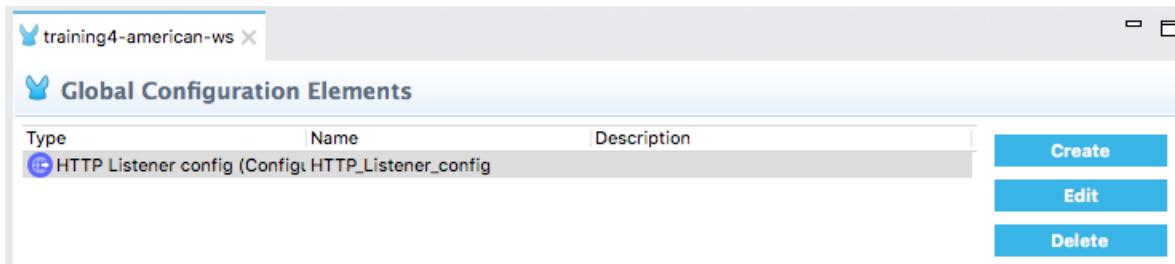


13. Click the Apply Changes button to save the file.



Review the HTTP Listener global element

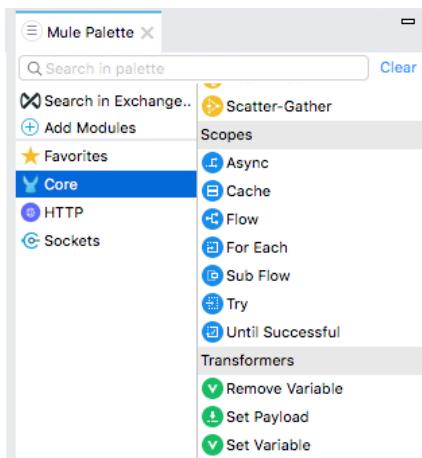
14. Select the Global Elements tab at the bottom of the canvas.
15. Double-click the HTTP Listener config.



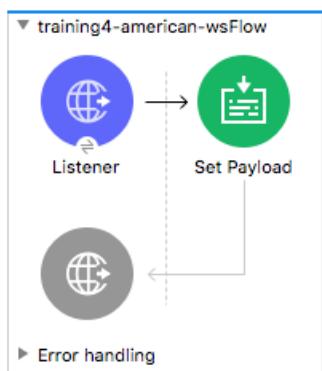
16. Review the information in the Global Element Properties dialog box and click Cancel.
17. Select the Message Flow tab to return to the canvas.

Display data

18. In the Mule Palette, select Core.
19. Scroll down in the right-side of the Mule Palette and locate the Transformers section.



20. Drag the Set Payload transformer from the Mule Palette into the process section of the flow.

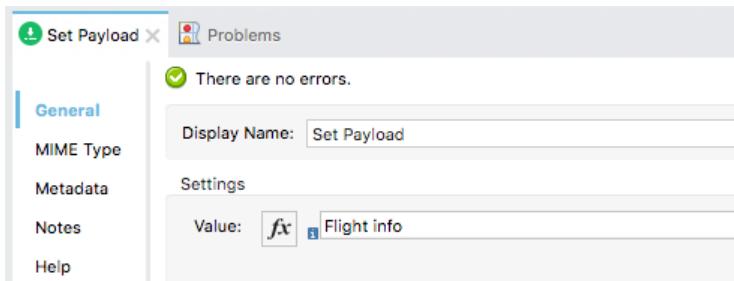


Configure the Set Payload transformer

21. In the Set Payload properties view, click the Switch to literal mode button for the value field.



22. Set the value field to Flight info.



23. Select the Configuration XML tab at the bottom of the canvas and examine the corresponding XML.

```
<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns:http="http://www.mulesoft.org/schema/mule/http" xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:doc="http://www.mulesoft.org/schema/mule/documentation"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.mulesoft.org/schema/mule/http http://www.mulesoft.org/schema/mule/http/current/mule-
      http-listener-config.xsd">
    <http:listener-config name="HTTP_Listener_config" doc:name="HTTP Listener config" doc:id="2a-
      8d-46e9-8c66-15c0aca06c65">
        <http:listener-connection host="0.0.0.0" port="8881" />
    </http:listener-config>
    <flow name="training4-american-wsFlow" doc:id="1400d159-c683-491c-b269-2fa7075a2375" >
        <http:listener doc:name="Listener" doc:id="500e5a4d-9367-46e9-8c66-15c0aca06c65" config-
            <set-payload value="Flight info" doc:name="Set Payload" doc:id="25506e33-312b-4a6f-9be4-
              </set-payload>
            </http:listener>
        </flow>
    </mule>
```

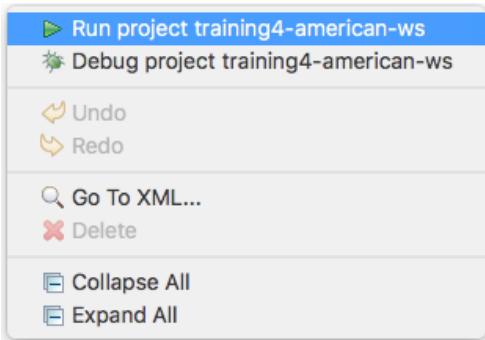
Message Flow Global Elements Configuration XML

24. Select the Message Flow tab to return to the canvas.

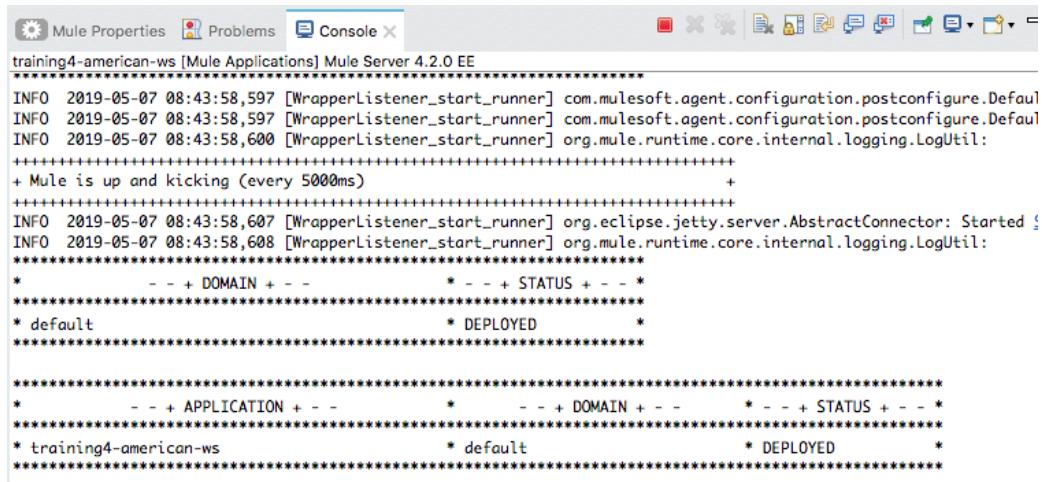
25. Click the Save button or press Cmd+S or Ctrl+S.

Run the application

26. Right-click in the canvas and select Run project training4-american-ws.



27. Watch the Console view; it should display information letting you know that both the Mule runtime and the training4-american-ws application started.



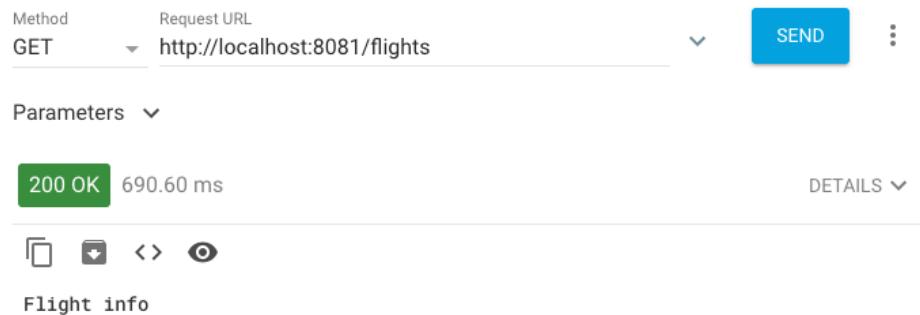
```
Mule Properties Problems Console
training4-american-ws [Mule Applications] Mule Server 4.2.0 EE
INFO 2019-05-07 08:43:58,597 [WrapperListener_start_runner] com.mulesoft.agent.configuration.postconfigure.DefaultConfigurationPostConfigure
INFO 2019-05-07 08:43:58,597 [WrapperListener_start_runner] com.mulesoft.agent.configuration.postconfigure.DefaultConfigurationPostConfigure
INFO 2019-05-07 08:43:58,600 [WrapperListener_start_runner] org.mule.runtime.core.internal.logging.LogUtil:
+-----+
+ Mule is up and kicking (every 5000ms) +
+-----+
INFO 2019-05-07 08:43:58,607 [WrapperListener_start_runner] org.eclipse.jetty.server.AbstractConnector: Started $Connector@53333333
INFO 2019-05-07 08:43:58,608 [WrapperListener_start_runner] org.mule.runtime.core.internal.logging.LogUtil:
* - - + DOMAIN + - - * - - + STATUS + - - *
*-----*-----*-----*
* default * DEPLOYED *-----*
*-----*-----*-----*
* - - + APPLICATION + - - * - - + DOMAIN + - - * - - + STATUS + - - *
*-----*-----*-----*-----*-----*-----*
* training4-american-ws * default * DEPLOYED *-----*
*-----*-----*-----*
```

Test the application

28. Return to Advanced REST Client.

29. Make sure the method is set to GET and that no headers or body are set for the request.

30. Make a GET request to <http://localhost:8081/flights>; you should see Flight info displayed.



Method: GET Request URL: http://localhost:8081/flights

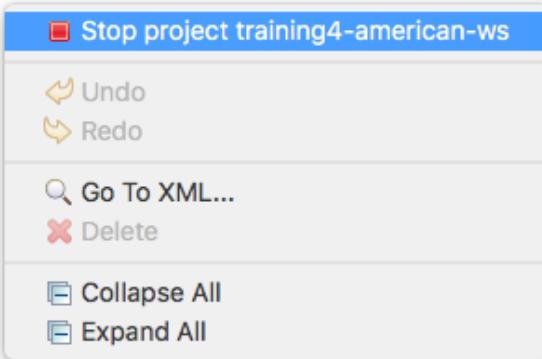
Parameters: ▾

200 OK 690.60 ms DETAILS ▾

Flight info

31. Return to Anypoint Studio.

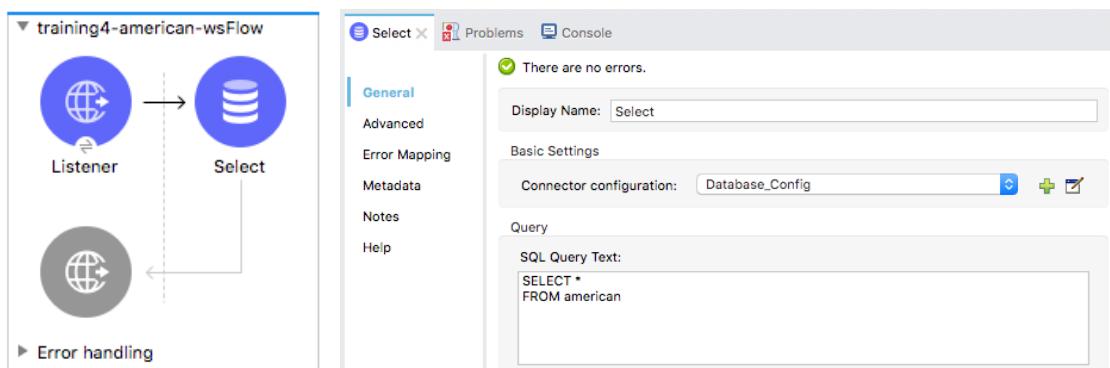
32. Right-click in the canvas and select Stop project training4-american-ws.



Walkthrough 4-2: Connect to data (MySQL database)

In this walkthrough, you connect to a database and retrieve data from a table that contains flight information. You will:

- Add a Database Select operation.
- Configure a Database connector that connects to a MySQL database (or optionally an in-memory Derby database if you do not have access to port 3306).
- Configure the Database Select operation to use that Database connector.
- Write a query to select data from a table in the database.



Locate database information

1. Return to the course snippets.txt file and locate the MySQL and Derby database information.

```
* MySQL database
db:
    host: "mudb.learn.mulesoft.com"
    port: "3306"
    user: "mule"
    password: "mule"
    database: "training"

American table: american
Account table: accounts
Account list URL: http://mu.learn.mulesoft.com/accounts/show
or if using mulesoft-training-services.jar application:
http://localhost:9090/accounts/show.html

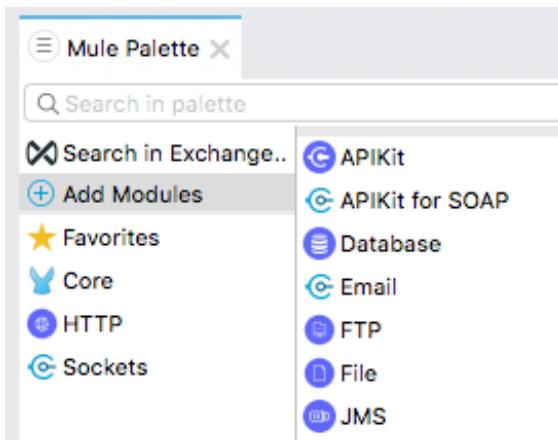
* MySQL database as URL and driver name
URL: jdbc:mysql://mudb.learn.mulesoft.com:3306/training?user=mule&password=mule
Driver class name: com.mysql.jdbc.Driver

* Derby database
URL: jdbc:derby://localhost:1527/memory:training
Driver class name: org.apache.derby.jdbc.ClientDriver
```

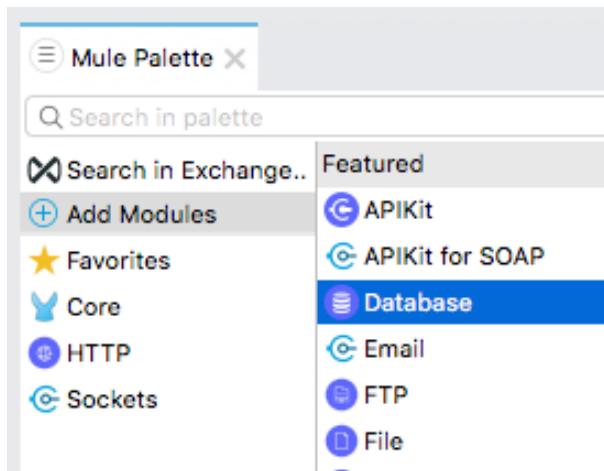
Note: The database information you see may be different than what is shown here; the values in the snippets file differ for instructor-led and self-study training classes.

Add a Database connector endpoint

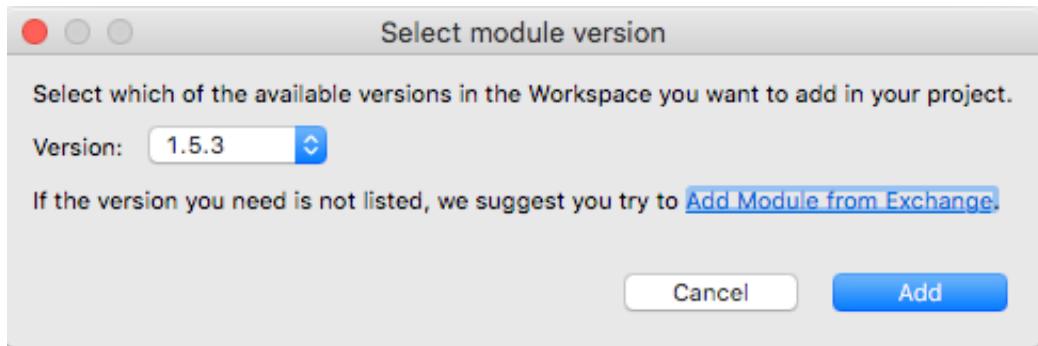
2. Return to Anypoint Studio.
3. Right-click the Set Payload message processor and select Delete.
4. In the Mule Palette, select Add Modules.



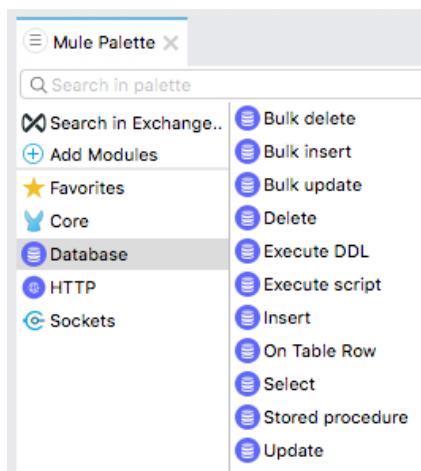
5. Select the Database connector in the right side of the Mule Palette and drag and drop it into the left side.



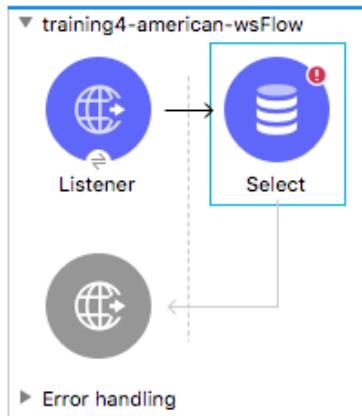
6. If you get a Select module version dialog box, select the latest version and click Add.



7. Locate the new Database connector in the Mule Palette.

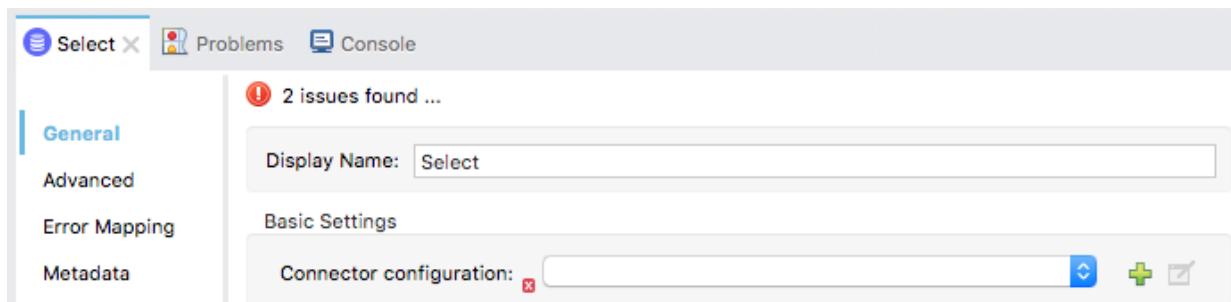


8. Drag and drop the Select operation in the process section of the flow.



Option 1: Configure a MySQL Database connector (if you have access to port 3306)

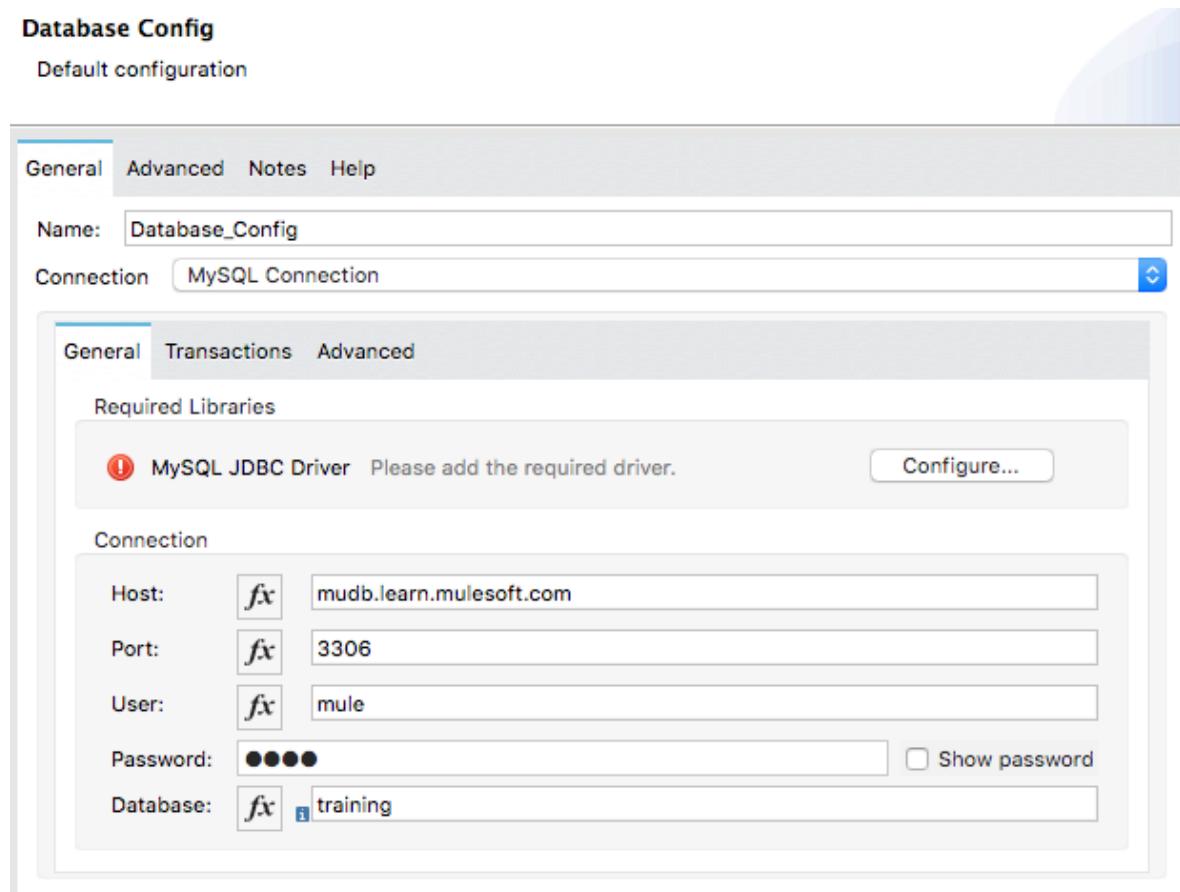
9. In the Select properties view, click the Add button next to connector configuration.



10. In the Global Element Properties dialog box, set the Connection to MySQL Connection.

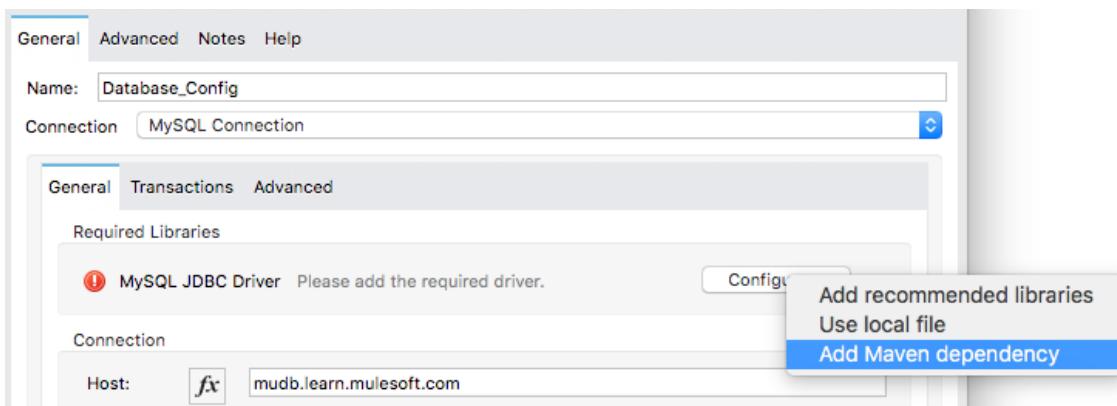


11. Set the host, port, user, password, and database values to the values listed in the course snippets.txt file.

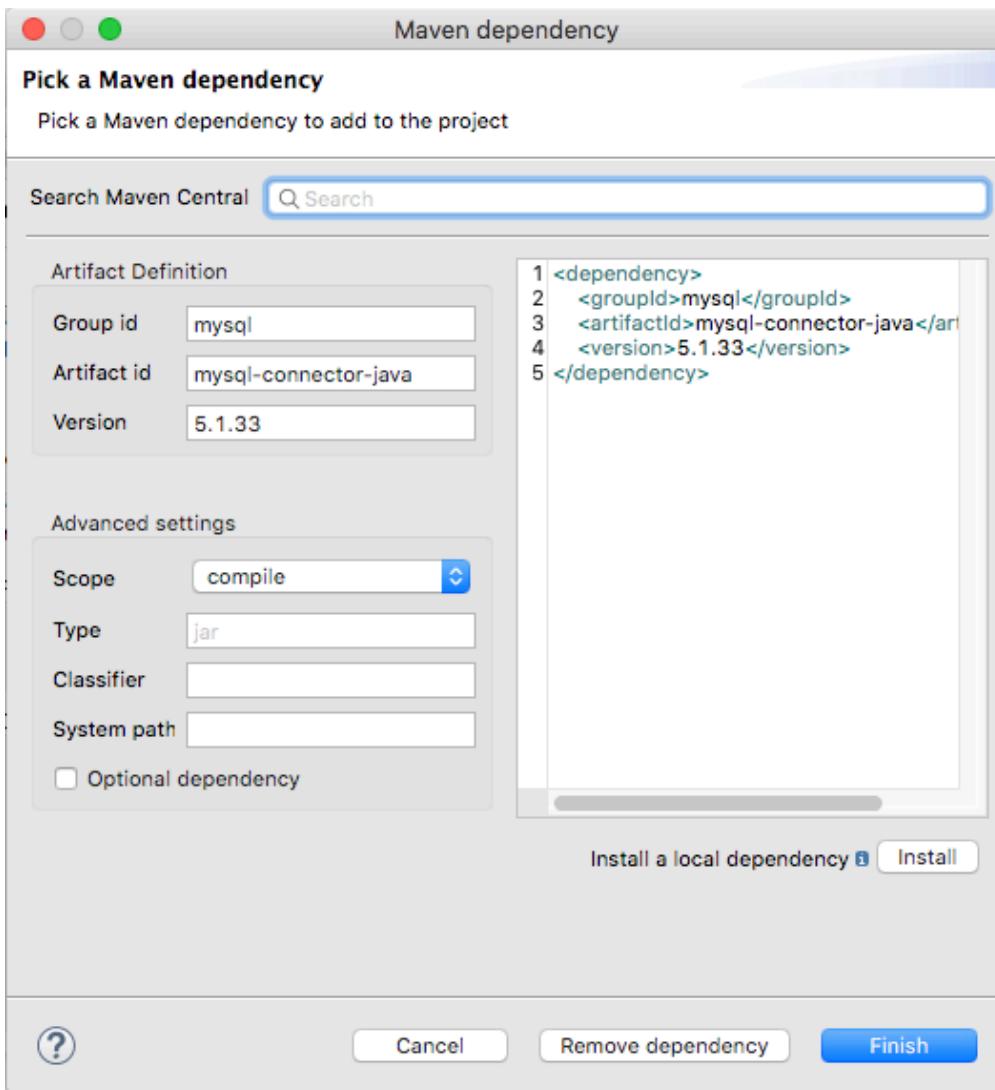


12. Click the Configure button next to MySQL JDBC Driver.

13. In the configure drop-down menu, select Add Maven dependency.

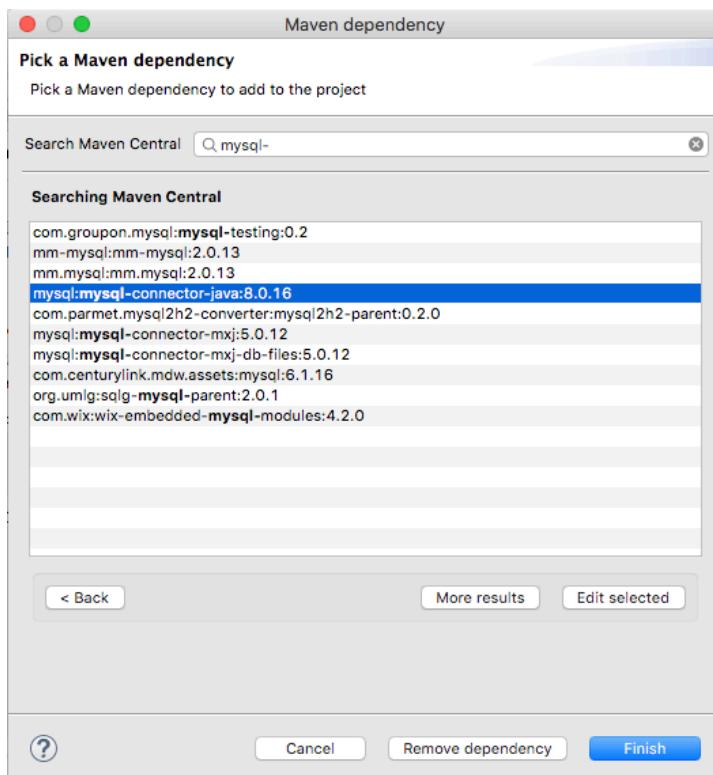


14. In the Maven dependency dialog box, locate the Search Maven Central text field.



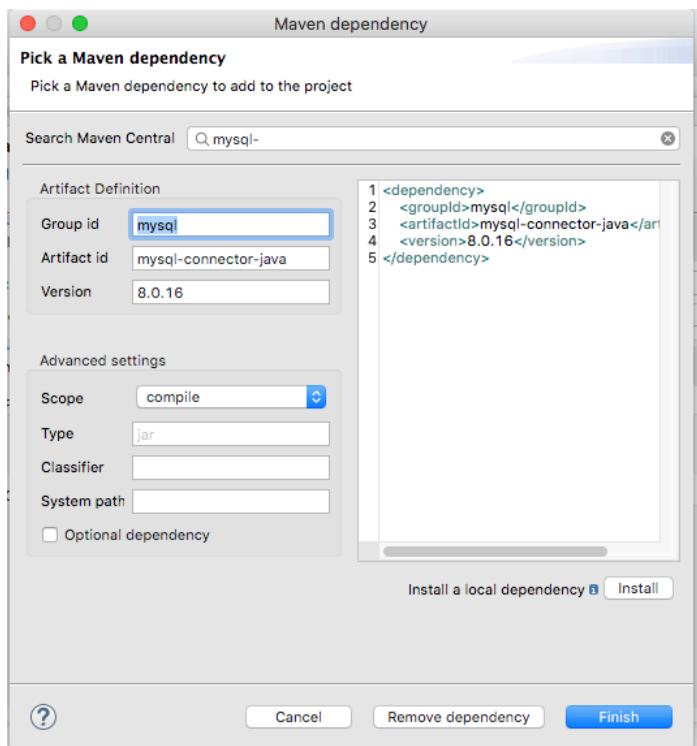
15. Enter mysql- in the Search Maven Central text field.

16. Select mysql:mysql-connector-java in the results that are displayed.



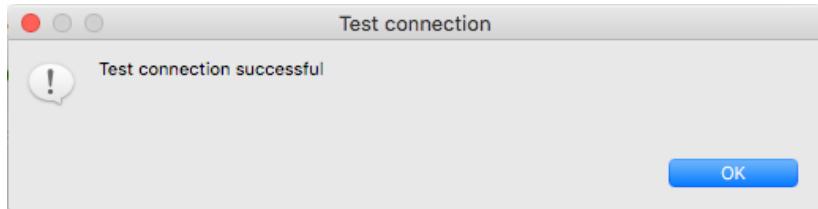
17. Click Edit selected.

18. Click Finish.



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

Note: Make sure the connection succeeds before proceeding.



Note: If the connectivity test fails, make sure you are not behind a firewall restricting access to port 3306. If you cannot access port 3306, use the instructions in the next section for option 2.

20. Click OK to close the Test connection dialog box.
21. Click OK to close the Global Element Properties dialog box.

Option 2: Configure a Derby Database connector (if no access to port 3306)

22. In a command-line interface, use the cd command to navigate to the folder containing the jars folder of the student files.
23. Run the mulesoft-training-services.jar file.

```
java -jar mulesoft-training-services-X.X.X.jar
```

Note: Replace X.X.X with the version of the JAR file, for example 1.7.0.

Note: The application uses ports 1527, 9090, 9091, and 61616. If any of these ports are already in use, you can change them when you start the application as shown in the following code.

```
java -jar mulesoft-training-services-X.X.X.jar --database.port=1530 --  
ws.port=9092 --activemq.broker.url=tcp://localhost:61617 --  
server.port=9193
```

24. Look at the output and make sure all the services started.

```
(\_/)      M U L E S O F T      T R A I N I N G      S E R V I C E S
/ \      *** Version 1.7.0 ***

Starting resources:
- American database started
- American flights database ready
- Message Broker started
- Order web service started
- Delta flights web service started
- United flights web service started
- JMS API published
- American flights API published
- Banking REST API published
- Accounts REST API published

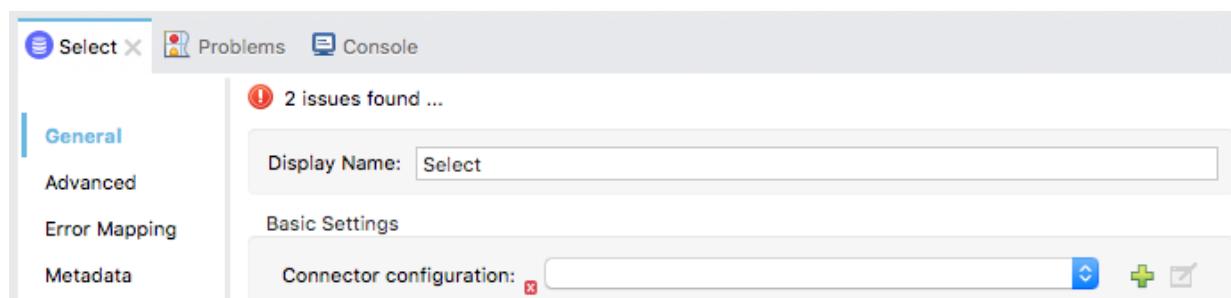
Available resources:
- Welcome page          : http://localhost:9090
- American database URL : jdbc:derby:///localhost:1527/memory:training
- American REST API     : http://localhost:9090/essentials/american/flights
- American REST API RAML: http://localhost:9090/essentials/american/flights-api.raml
- United REST service   : http://localhost:9090/essentials/united/flights
- Delta SOAP service    : http://localhost:9191/essentials/delta
- Delta SOAP WSDL       : http://localhost:9191/essentials/delta?wsdl
- Accounts API          : http://localhost:9090/essentials/accounts/api
- Accounts form         : http://localhost:9090/essentials/accounts/show.html
- JMS broker URL        : tcp://localhost:61616
- JMS topic name        : ap essentials
- JMS Form               : http://localhost:9090/essentials/jmsform.html
- Banking API            : http://localhost:9090/api/...
- Banking API RAML      : http://localhost:9090/api/banking-api.raml

Press CTRL-C to terminate this application...
```

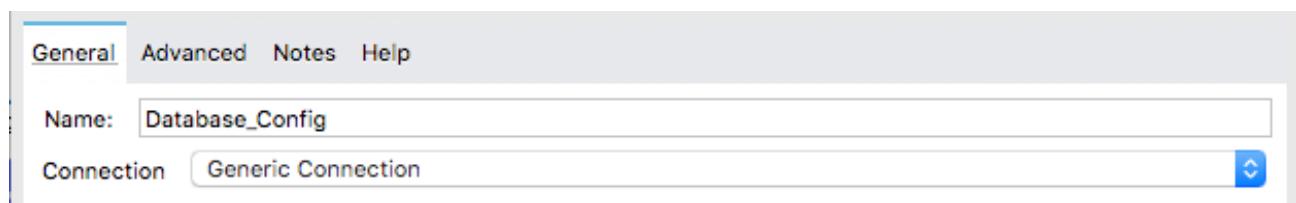
Note: When you want to stop the application, return to this window and press Ctrl+C.

25. Return to Anypoint Studio.

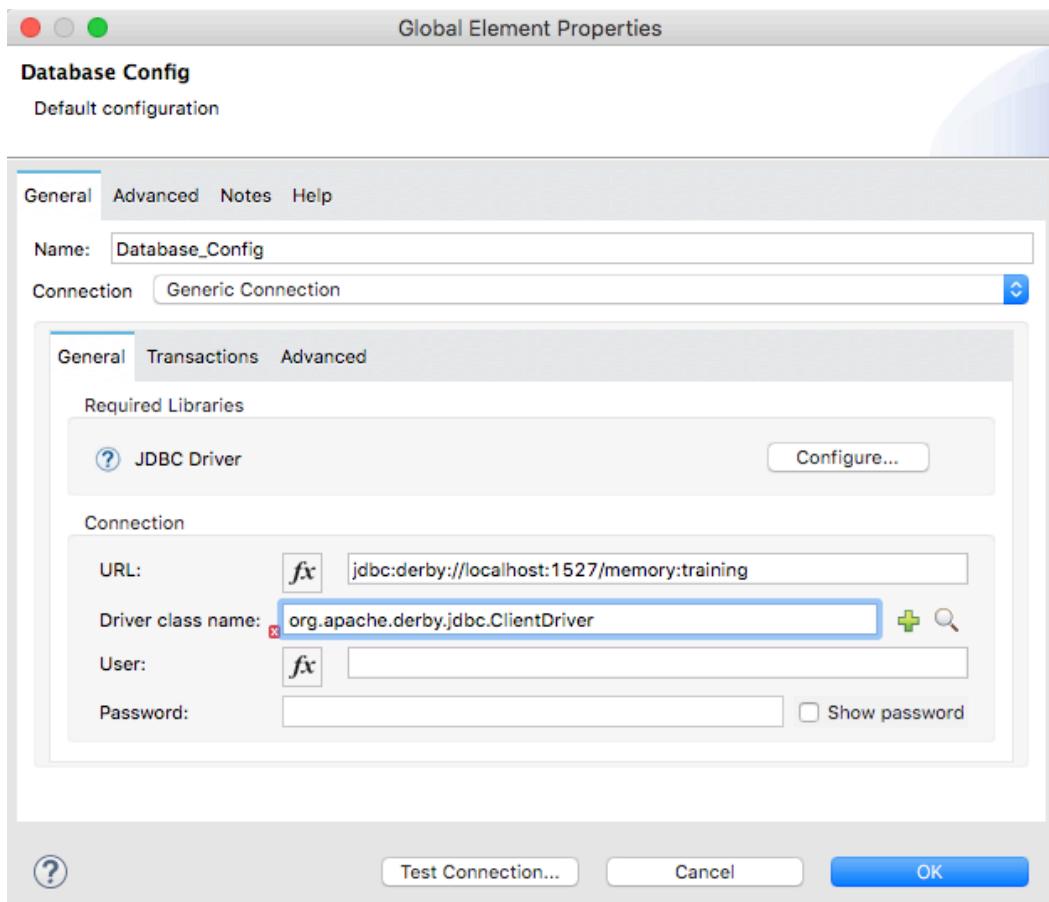
26. In the Select properties view, click the Add button next to connector configuration.



27. In the Global Element Properties dialog box, set the Connection to Generic Connection.



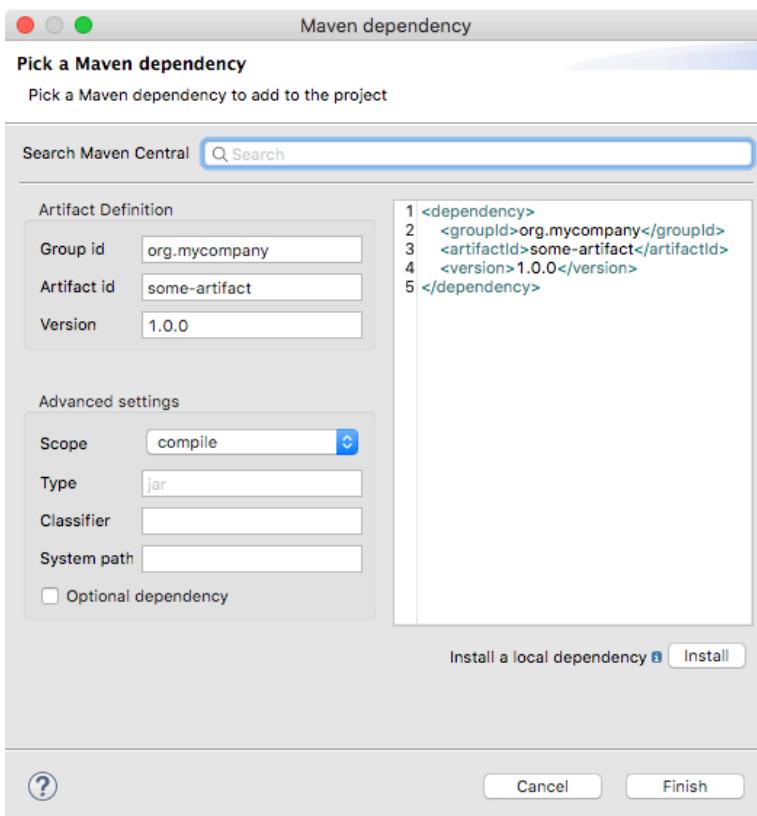
28. Set the URL and driver class name values to the values listed in the course snippets.txt file.



29. Click the Configure button next to JDBC Driver.

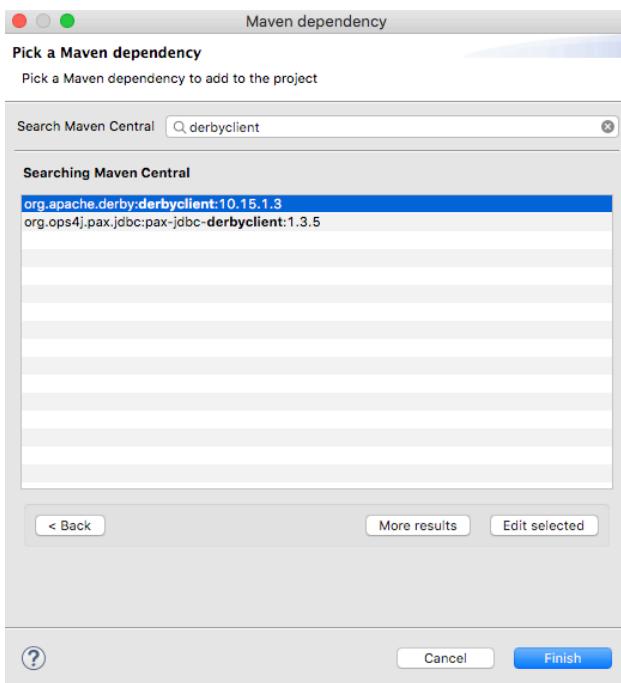
30. In the configure drop-down menu, select Add Maven dependency.

31. In the Maven dependency dialog box, locate the Search Maven Central text field.



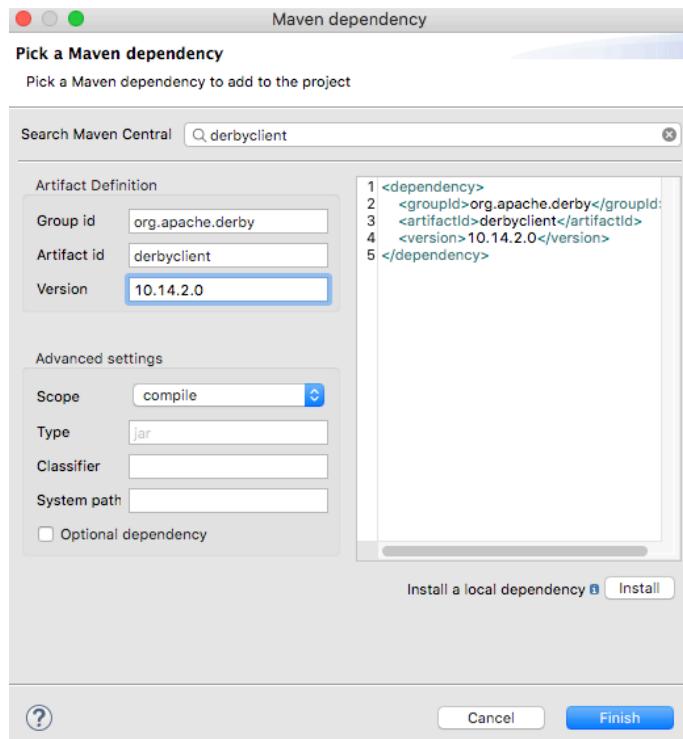
32. Enter derbyclient in the Search Maven Central text field.

33. Select org.apache.derby:derbyclient in the results that are displayed.



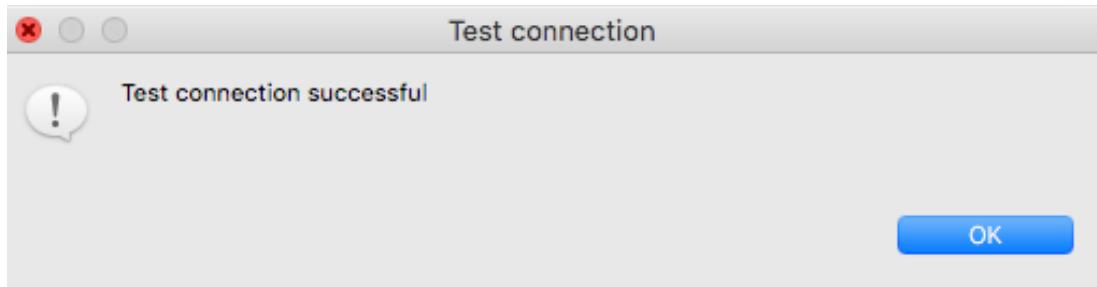
34. Click Edit selected.
35. Enter 10.14.2.0 in the Version text field.

Note: Version 10.14.2.0 is the latest Derby client compatible with Java 8.



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

Note: Make sure the connection succeeds before proceeding.

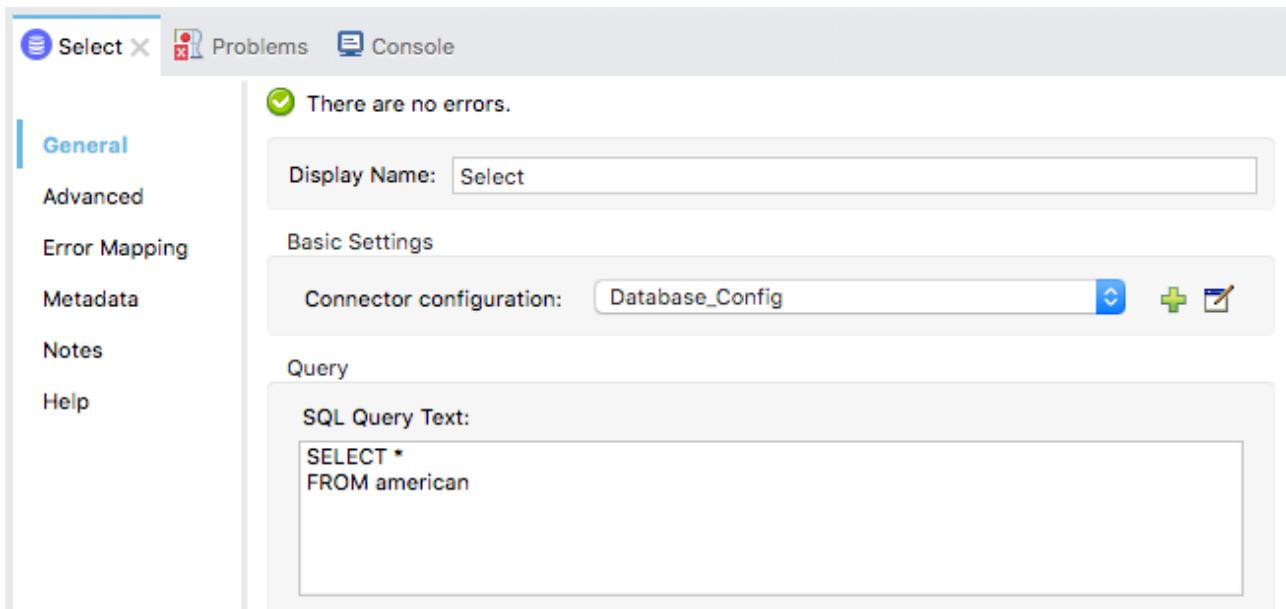


38. Click OK to close the Test connection dialog box.
39. Click OK to close the Global Element Properties dialog box.

Write a query to return all flights

40. In the Select properties view, add a query to select all records from the american table.

```
SELECT *  
FROM american
```



Test the application

41. Run the project.
42. In the Save changes dialog box, click Save.
43. Watch the console and wait for the application to start.
44. Once the application has started, return to Advanced REST Client.
45. In Advanced REST Client, make another request to <http://localhost:8081/flights>; you should get a 500 Server Error with an invalid data message.

Method: GET Request URL: http://localhost:8081/flights SEND Request parameters DETAILS

500 Server Error 5059.59 ms

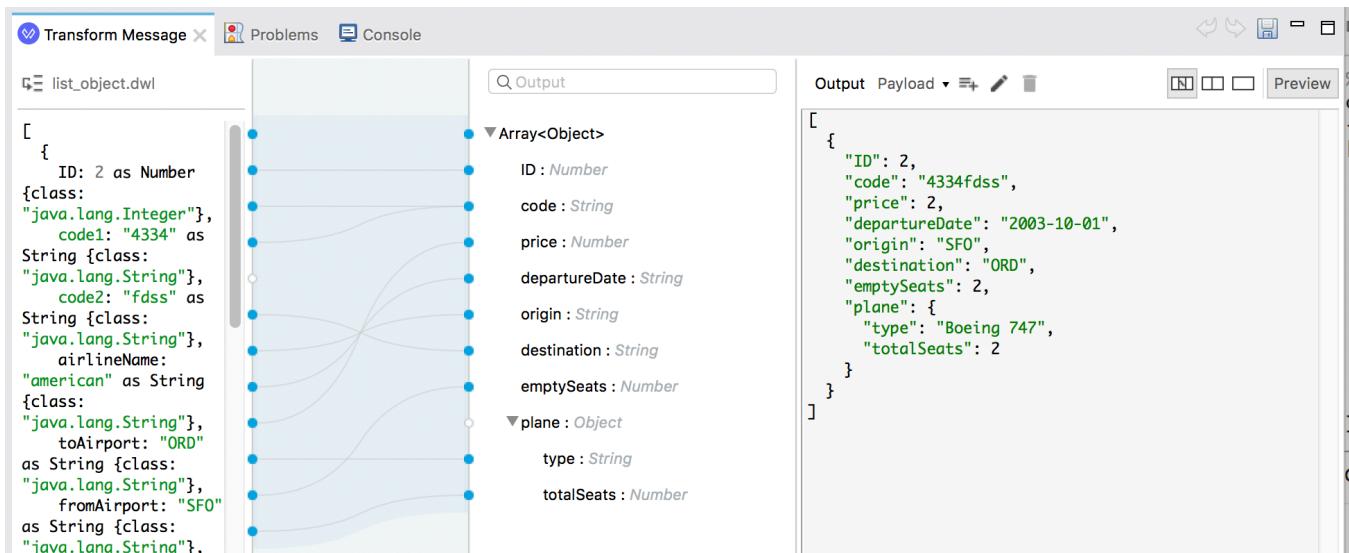
COPY SAVE SOURCE VIEW

Attempted to send invalid data through http response.

Walkthrough 4-3: Transform data

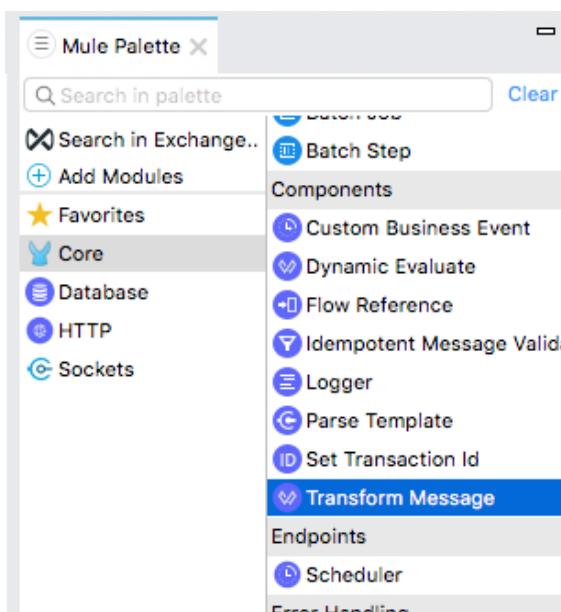
In this walkthrough, you transform and display the flight data into JSON. You will:

- Use the Transform Message component.
- Use the DataWeave visual mapper to change the response to a different JSON structure.

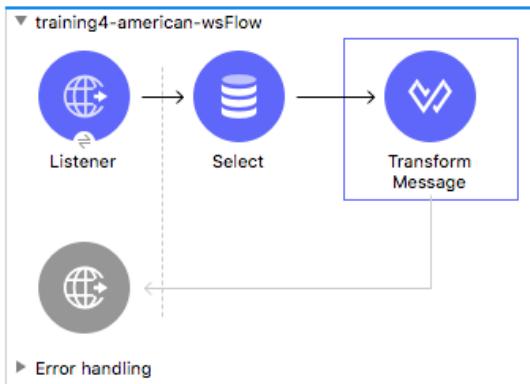


Add a Transform Message component

1. Return to Anypoint Studio.
2. In the Mule Palette, select Core and locate the Transform Message component in the Components section.



3. Drag the Transform Message component and drop it after the Select processor.



Review metadata for the transformation input

4. In the Transform Message properties view, look at the input section and review the payload metadata.

Note: If you are using the local Derby database, the properties will be uppercase instead.

Input Fields	Output Preview
ID : Number	1 %dw 2.0
code1 : String	2 output application/java
code2 : String?	3 ---
airlineName : String?	4 {
toAirport : String?	5 }

Return the payload as JSON

5. In the Transform Message properties view, change the output type from application/java to application/json and change the {} to payload.

```

Output Payload ▾ + ⚏ ⚗ | [ ] [ ] [ ] [ ]
1 %dw 2.0
2 output application/json
3 ---
4 payload
      
```

Test the application

6. Save the file to redeploy the project.
7. In Advanced REST Client, send the same request; you should get a 200 response and the American flight data represented as JSON.

Method Request URL
GET ▾ <http://localhost:8081/flights>

Parameters ▾

200 OK 1869.10 ms

Copy Download Share Refresh

```
[Array[11]
-0: {
  "planeType": "Boeing 787",
  "code2": "0001",
  "takeOffDate": "2016-01-19T19:00:00",
  "code1": "ree",
  "fromAirport": "MUA",
  "price": 541,
  "seatsAvailable": 0,
  "toAirport": "LAX",
  "ID": 1,
  "airlineName": "American Airlines",
  "totalSeats": 200
},
-1: {
  "planeType": "Boeing 747",
  "code2": "0123"
}]
```

Note: If you are using the local Derby database, the properties will be uppercase instead.

Review the data structure to be returned by the American flights API

8. Return to your American Flights API in Exchange.

9. Look at the example data returned for the /flights GET method.



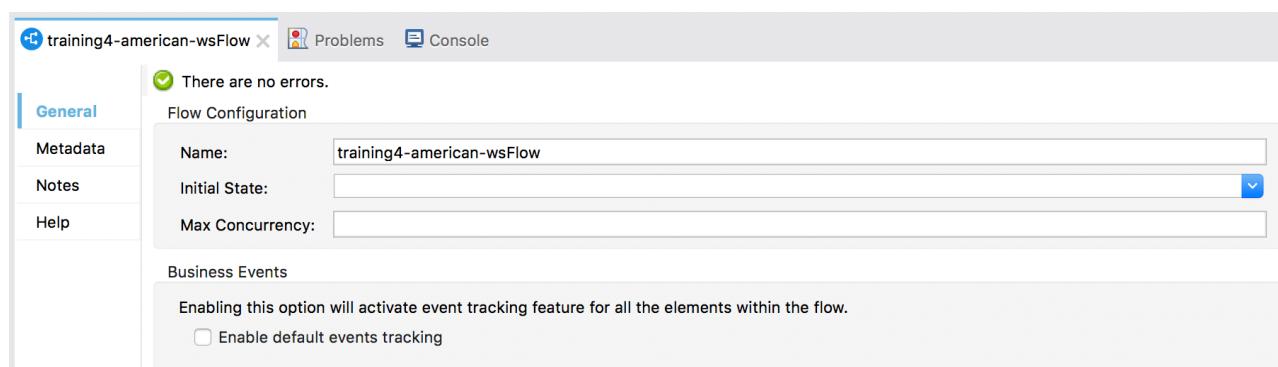
```
200 OK 545.81 ms Details ✓

[Array[2]
-0: {
  "ID": 1,
  "code": "ER38sd",
  "price": 400,
  "departureDate": "2017/07/26",
  "origin": "CLE",
  "destination": "SFO",
  "emptySeats": 0,
  "plane": {
    "type": "Boeing 737",
    "totalSeats": 150
  }
},
-1: {
  "ID": 2,
  "code": "ER45if",
  "price": 540.99
}]
```

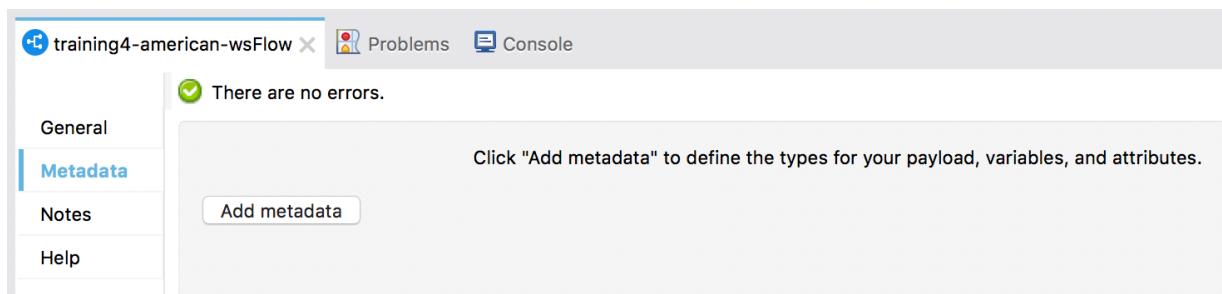
10. Notice that the structure of the JSON being returned by the Mule application does not match this example JSON format.

Define metadata for the data structure to be returned by the American flights API

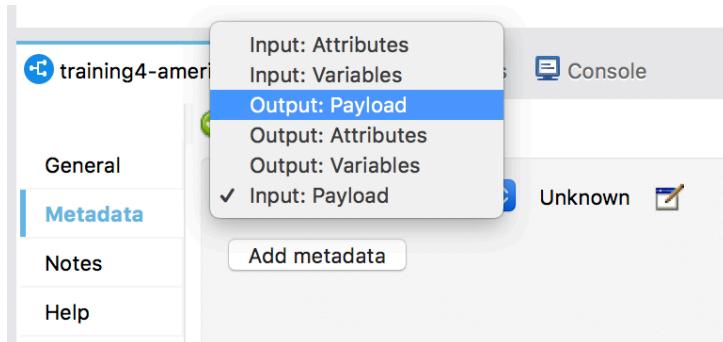
11. Return to Anypoint Studio.
12. In the canvas, click training4-american-wsFlow name.
13. In the training4-american-wsFlow properties view, select the Metadata tab.



14. Click the Add metadata button.



15. In the drop-down menu, select Output: Payload.



16. Click the Edit button.

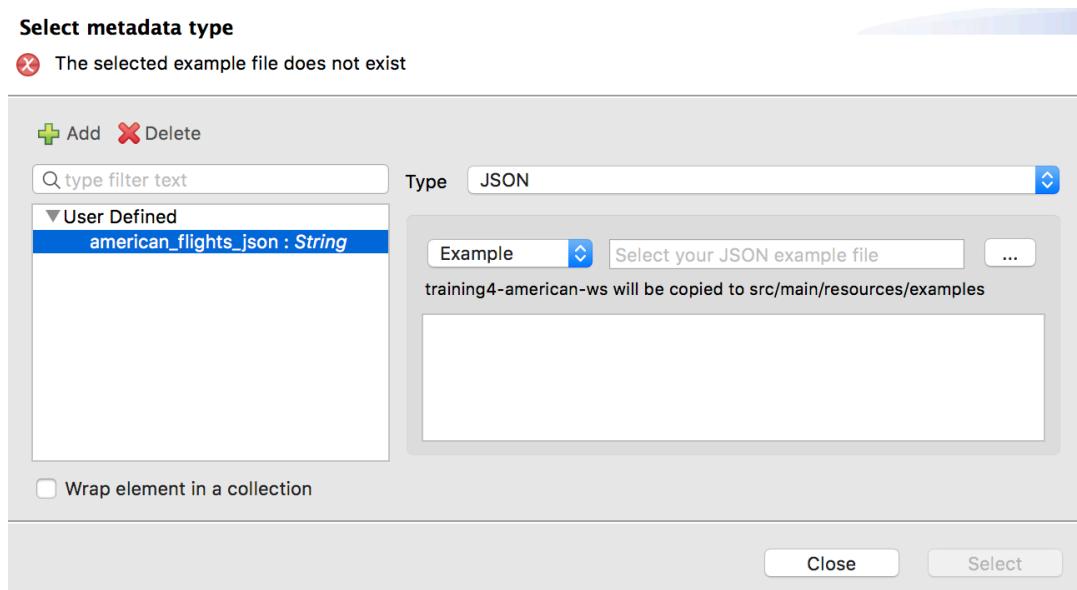
17. In the Select metadata type dialog box, click the Add button.

18. In the Create new type dialog box, set the type id to american_flights_json.

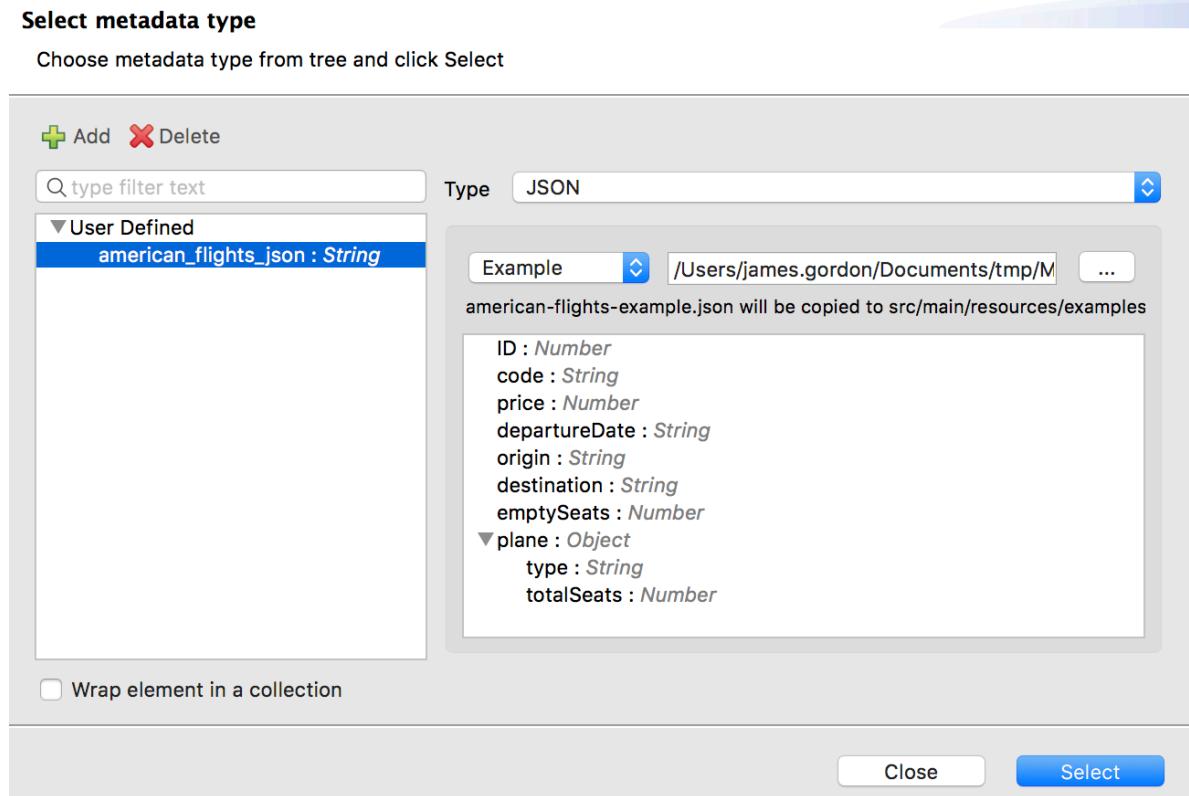
19. Click Create type.

20. Back in the Select metadata type dialog box, set the first type to JSON.

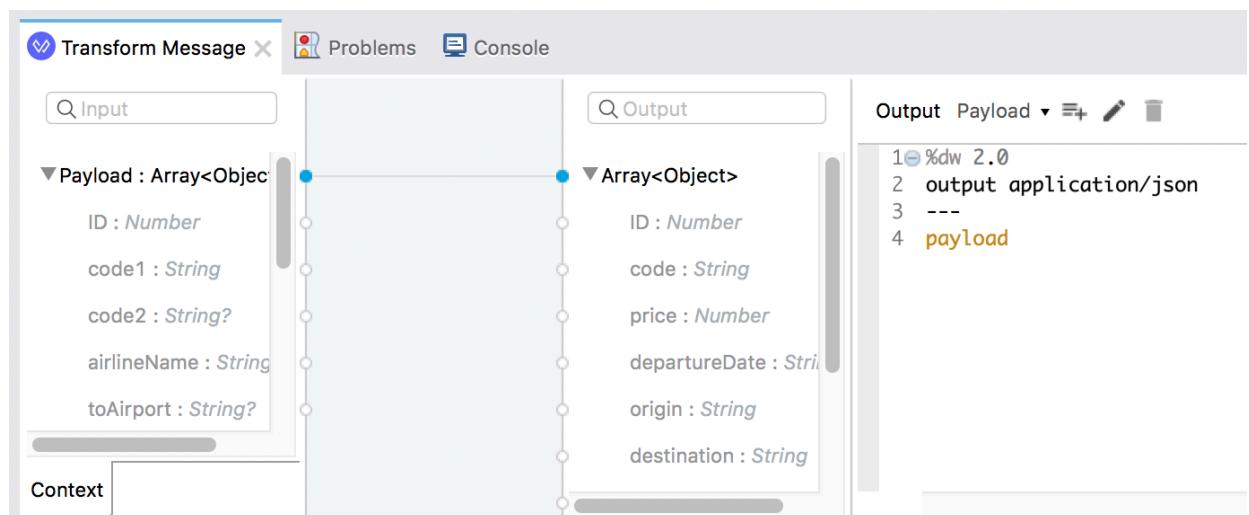
21. Change the Schema selection to Example.



22. Click the browse button and navigate to the student files.
23. Select american-flights-example.json in the resources/examples folder and click Open; you should see the example data for the metadata type.



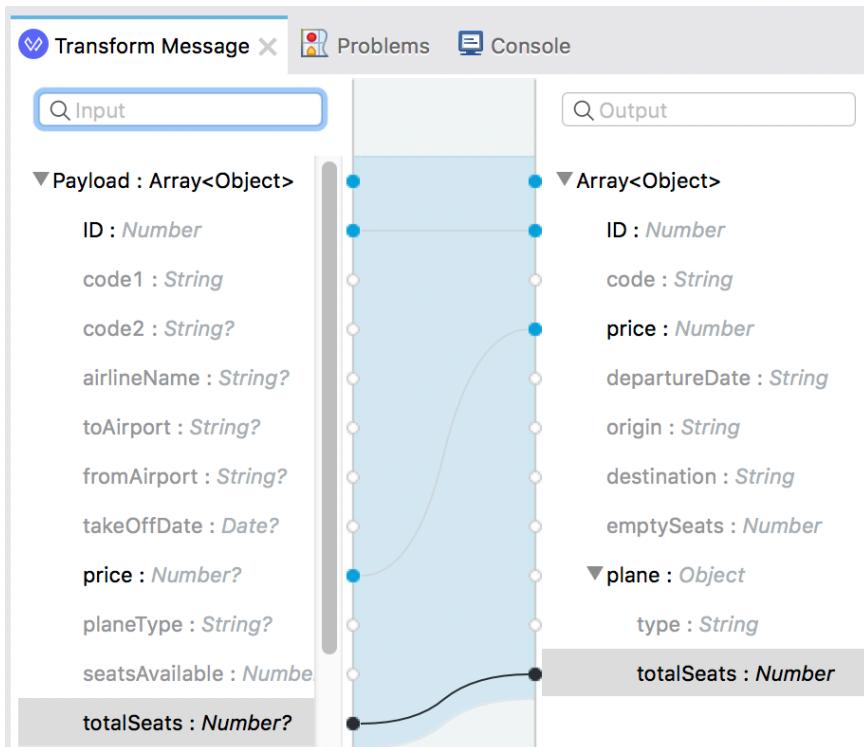
24. Click Select.
25. In training4-american-wsFlow, click the Transform Message component; you should now see output metadata in the output section of the Transform Message properties view.



Create the transformation

26. Map fields with the same names by dragging them from the input section and dropping them on the corresponding field in the output section.

- ID to ID
- price to price
- totalSeats to plane > totalSeats

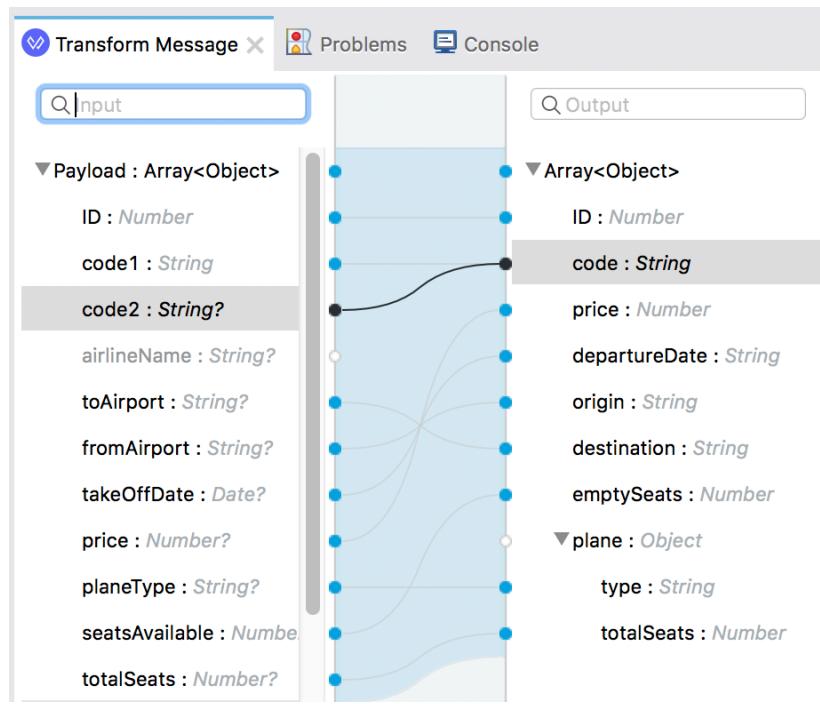


27. Map fields with different names by dragging them from the input section and dropping them on the corresponding field in the output section.

- toAirport to destination
- takeOffDate to departureDate
- fromAirport to origin
- seatsAvailable to emptySeats
- planeType to plane > type

28. Concatenate two fields by dragging them from the input section and dropping them on the same field in the output section.

- code1 to code
- code2 to code



Add sample data (optional)

29. Click the Preview button in the output section.

30. In the preview section, click the Create required sample data to execute preview link.

```
1 %dw 2.0
2 output application/json
3 ---
4 payload map ( payload01 , index0:
5   ID: payload01.ID,
6   code: (payload01.code1 defau
7   price: payload01.price defau
8   departureDate: payload01.tak
9   origin: payload01.fromAirport
10  destination: payload01.toAirpo
11  emptySeats: payload01.seatsAvai
12  plane: {
13    "type": payload01.planeT
14    totalSeats: payload01.to
15  }
16 }
```

The screenshot shows the 'Payload' tab of the 'Transform Message' component. It contains a script editor with MuleScript code. A specific line of code, 'code: (payload01.code1 defau', is highlighted and selected. A tooltip 'Create required sample data to execute preview' is displayed below the cursor, indicating that clicking it will generate sample data for the preview.

31. Look at the input section, you should see a new tab called payload with sample data generated from the input metadata.
32. Look at the output section, you should see a sample response for the transformation.

The screenshot shows the Mule ESB Transform Message editor. On the left, the input JSON is defined as:

```
[{"ID": 2, "code1": "?????", "code2": "?????", "airlineName": "?????", "toAirport": "?????", "fromAirport": "?????"}]
```

On the right, the output JSON is defined as:

```
[{"ID": Number, "code": String, "price": Number, "departureDate": String, "origin": String, "destination": String, "emptySeats": Number, "plane": Object, "type": String, "totalSeats": Number}]
```

The mapping is visualized with arrows connecting fields from the input to the output. The 'ID' field maps to 'ID'. The 'code1' and 'code2' fields map to 'code'. The 'airlineName' field maps to 'origin'. The 'toAirport' and 'fromAirport' fields map to 'destination'. The 'emptySeats' field maps to 'emptySeats'. The 'plane' object is mapped from the input array to the output array. The 'type' and 'totalSeats' fields map to their respective counterparts in the output object.

33. In the input section, replace all the ???? with sample values.
34. Look at the output section, you should see the sample values in the transformed data.

The screenshot shows the Mule ESB Transform Message editor. On the left, the input JSON is defined as:

```
[{"ID": 2, "code1": "4334", "code2": "fdss", "airlineName": "american", "toAirport": "ORD", "fromAirport": "SFO"}]
```

On the right, the output JSON is defined as:

```
[{"ID": Number, "code": String, "price": Number, "departureDate": String, "origin": String, "destination": String, "emptySeats": Number, "plane": Object, "type": String, "totalSeats": Number}]
```

The mapping is visualized with arrows connecting fields from the input to the output. The 'ID' field maps to 'ID'. The 'code1' and 'code2' fields map to 'code'. The 'airlineName' field maps to 'origin'. The 'toAirport' and 'fromAirport' fields map to 'destination'. The 'emptySeats' field maps to 'emptySeats'. The 'plane' object is mapped from the input array to the output array. The 'type' and 'totalSeats' fields map to their respective counterparts in the output object.

Test the application

35. Save the file to redeploy the project.

36. In Advanced REST Client, make another request to <http://localhost:8081/flights>; you should see all the flight data as JSON again but now with the example JSON format.

Method Request URL
GET ▼ <http://localhost:8081/flights>

Parameters ▾

200 OK 1622.70 ms

□ □ ▶ ■■■

```
[Array[11]
-0: {
  "ID": 1,
  "code": "rree0001",
  "price": 541,
  "departureDate": "2016-01-19T19:00:00",
  "origin": "MUA",
  "destination": "LAX",
  "emptySeats": 0,
  -"plane": {
    "type": "Boeing 787",
    "totalSeats": 200
  }
},
-1: {
  "ID": 2,
  "code": "eefd0123",
  "price": 650,
  "departureDate": "2016-01-19T19:00:00",
  "origin": "MUA",
  "destination": "LAX",
  "emptySeats": 0,
  -"plane": {
    "type": "Boeing 787",
    "totalSeats": 200
  }
},
-2: {
  "ID": 3,
  "code": "eefd0123",
  "price": 650,
  "departureDate": "2016-01-19T19:00:00",
  "origin": "MUA",
  "destination": "LAX",
  "emptySeats": 0,
  -"plane": {
    "type": "Boeing 787",
    "totalSeats": 200
  }
},
-3: {
  "ID": 4,
  "code": "eefd0123",
  "price": 650,
  "departureDate": "2016-01-19T19:00:00",
  "origin": "MUA",
  "destination": "LAX",
  "emptySeats": 0,
  -"plane": {
    "type": "Boeing 787",
    "totalSeats": 200
  }
},
-4: {
  "ID": 5,
  "code": "eefd0123",
  "price": 650,
  "departureDate": "2016-01-19T19:00:00",
  "origin": "MUA",
  "destination": "LAX",
  "emptySeats": 0,
  -"plane": {
    "type": "Boeing 787",
    "totalSeats": 200
  }
},
-5: {
  "ID": 6,
  "code": "eefd0123",
  "price": 650,
  "departureDate": "2016-01-19T19:00:00",
  "origin": "MUA",
  "destination": "LAX",
  "emptySeats": 0,
  -"plane": {
    "type": "Boeing 787",
    "totalSeats": 200
  }
},
-6: {
  "ID": 7,
  "code": "eefd0123",
  "price": 650,
  "departureDate": "2016-01-19T19:00:00",
  "origin": "MUA",
  "destination": "LAX",
  "emptySeats": 0,
  -"plane": {
    "type": "Boeing 787",
    "totalSeats": 200
  }
},
-7: {
  "ID": 8,
  "code": "eefd0123",
  "price": 650,
  "departureDate": "2016-01-19T19:00:00",
  "origin": "MUA",
  "destination": "LAX",
  "emptySeats": 0,
  -"plane": {
    "type": "Boeing 787",
    "totalSeats": 200
  }
},
-8: {
  "ID": 9,
  "code": "eefd0123",
  "price": 650,
  "departureDate": "2016-01-19T19:00:00",
  "origin": "MUA",
  "destination": "LAX",
  "emptySeats": 0,
  -"plane": {
    "type": "Boeing 787",
    "totalSeats": 200
  }
},
-9: {
  "ID": 10,
  "code": "eefd0123",
  "price": 650,
  "departureDate": "2016-01-19T19:00:00",
  "origin": "MUA",
  "destination": "LAX",
  "emptySeats": 0,
  -"plane": {
    "type": "Boeing 787",
    "totalSeats": 200
  }
},
-10: {
  "ID": 11,
  "code": "eefd0123",
  "price": 650,
  "departureDate": "2016-01-19T19:00:00",
  "origin": "MUA",
  "destination": "LAX",
  "emptySeats": 0,
  -"plane": {
    "type": "Boeing 787",
    "totalSeats": 200
  }
}]]
```

Try to retrieve information about a specific flight

37. Add a URI parameter to the URL to make a request to <http://localhost:8081/flights/3>; you should get a 404 Not Found response with a no listener message.

Method Request URL
GET ▼ <http://localhost:8081/flights/3>

Parameters ▾

404 Not Found 18.20 ms

DETAILS ▾

□ □ ▶ ○

```
No listener for endpoint: /flights/3
```

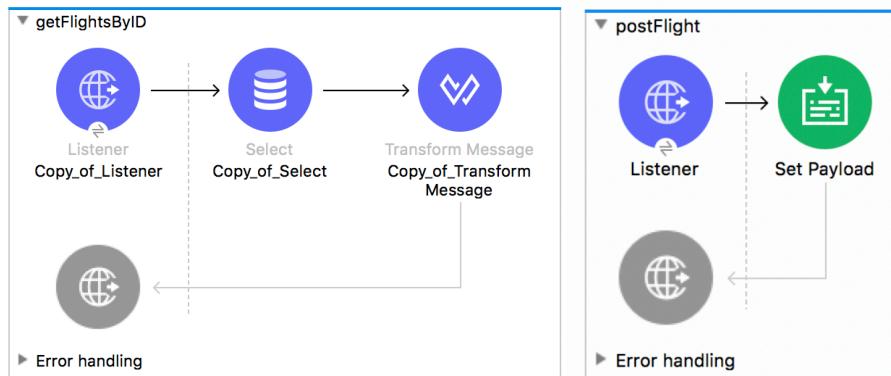
38. Return to Anypoint Studio.

39. Look at the console; you should get a no listener found for request (GET)/flights/3.

Walkthrough 4-4: Create a RESTful interface for a Mule application

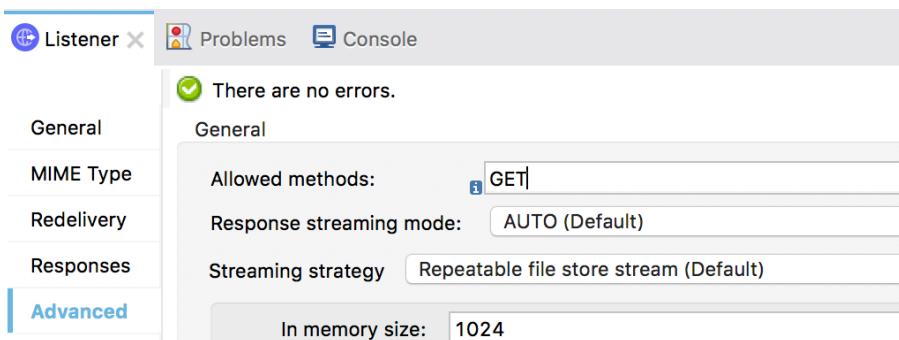
In this walkthrough, you continue to create a RESTful interface for the application. You will:

- Route based on path.
- Use a URI parameter in the path of a new HTTP Listener.
- Route based on HTTP method.



Restrict method calls to GET

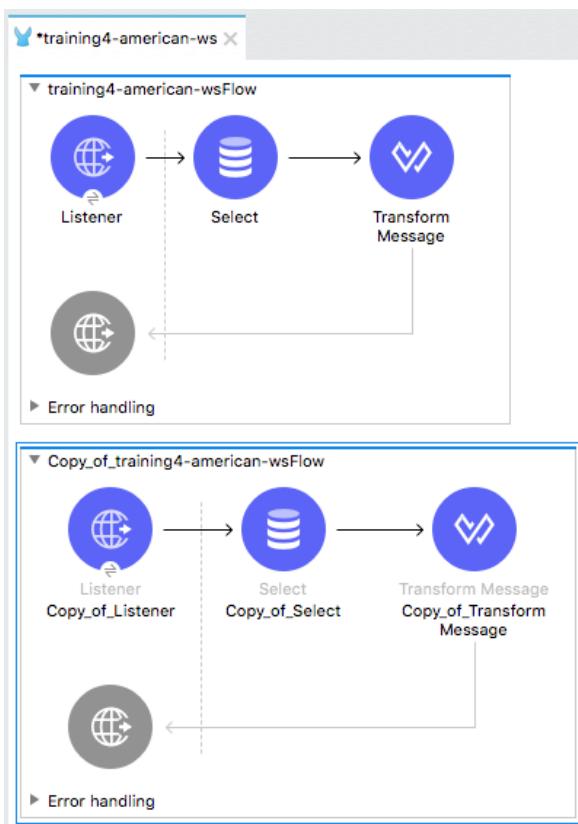
1. Return to Anypoint Studio.
2. Double-click the HTTP Listener in the flow.
3. In the left-side navigation of the Listener properties view, select Advanced.
4. Set the allowed methods to GET.



Make a copy of the existing flow

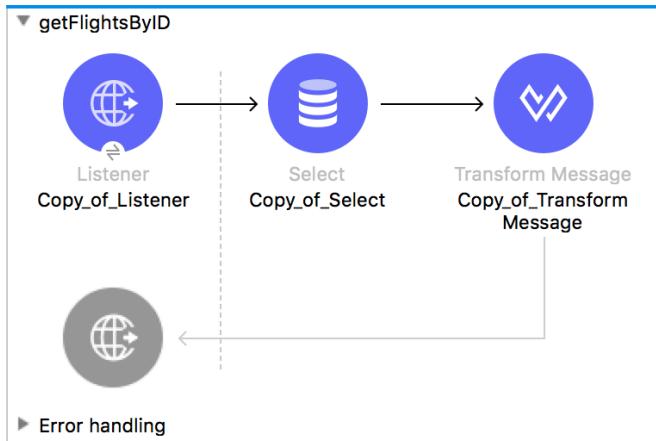
5. Click the flow in the canvas to select it.
6. From the main menu bar, select Edit > Copy.

7. Click in the canvas beneath the flow and select Edit > Paste.



Rename the flows

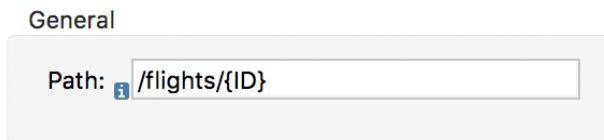
8. Double-click the first flow.
9. In the properties view, change its name to getFlights.
10. Change the name of the second flow to getFlightsByID.



Note: If you want, change the name of the event source and event processors.

Specify a URI parameter for the new HTTP Listener endpoint

11. Double-click the HTTP Listener in getFlightsByID.
12. Modify the path to have a URI parameter called ID.



Modify the Database endpoint

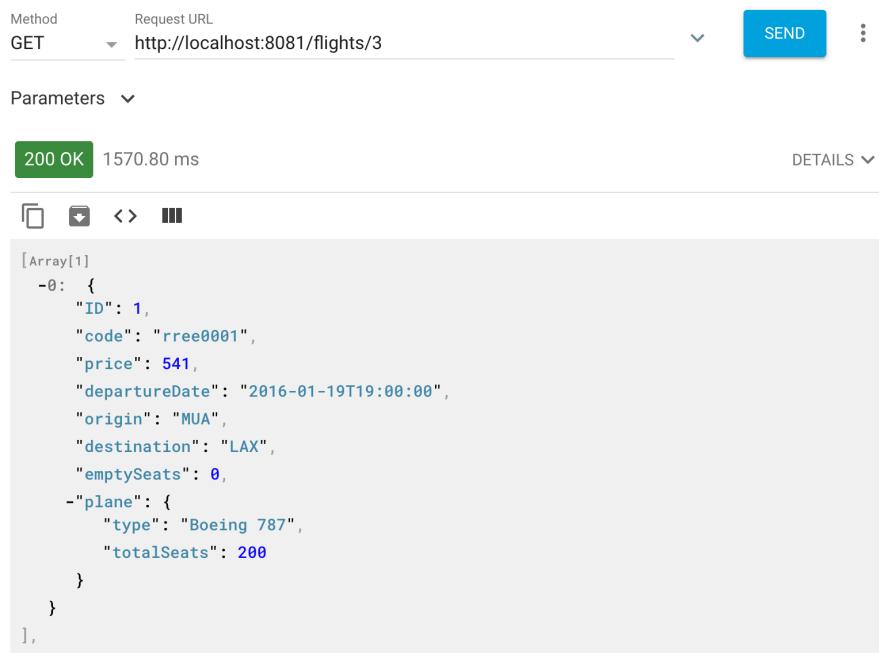
13. Double-click the Select operation in getFlightsByID.
14. Modify the query WHERE clause, to select flights with the ID equal to 1.

```
SELECT *
FROM american
WHERE ID = 1
```

Test the application

15. Save the file to redeploy the project.

16. In Advanced REST Client, make another request to <http://localhost:8081/flights/3>; you should see details for the flight with an ID of 1.



The screenshot shows the Advanced REST Client interface. At the top, the method is set to "GET" and the request URL is "http://localhost:8081/flights/3". Below the URL, there's a "SEND" button and a "DETAILS" dropdown menu. Under the "Parameters" section, there's a "200 OK" status indicator and a timestamp of "1570.80 ms". The main content area displays a JSON response:

```
[Array[1]
-0: {
  "ID": 1,
  "code": "rree0001",
  "price": 541,
  "departureDate": "2016-01-19T19:00:00",
  "origin": "MUA",
  "destination": "LAX",
  "emptySeats": 0,
  -"plane": {
    "type": "Boeing 787",
    "totalSeats": 200
  }
},
],
```

Note: You did not add logic to the application to search for a flight with a particular ID. You will deploy an application with this additional functionality implemented next.

Modify the database query to use the URI parameter

17. Return to the course snippets.txt file and copy the SQL input parameter expression.
18. Return to the getFlightsByID flow in Anypoint Studio.
19. In the Select properties view, locate the Query Input Parameters section, click the Switch to expression mode button, then paste the expression you copied.

```
{'ID' : attributes.uriParams.ID}
```

Note: You learn to write expressions in a later module in the Development Fundamentals course.

20. Change the WHERE clause in the SQL Query Text to use this input parameter.

```
SELECT *
FROM american
WHERE ID = :ID
```

The screenshot shows the 'Copy_of_Select' component configuration in Anypoint Studio. The 'General' tab is selected. The 'Display Name' is set to 'Copy_of_Select'. Under 'Basic Settings', the 'Connector configuration' is set to 'Database_Config'. In the 'Query' section, the 'SQL Query Text' is:

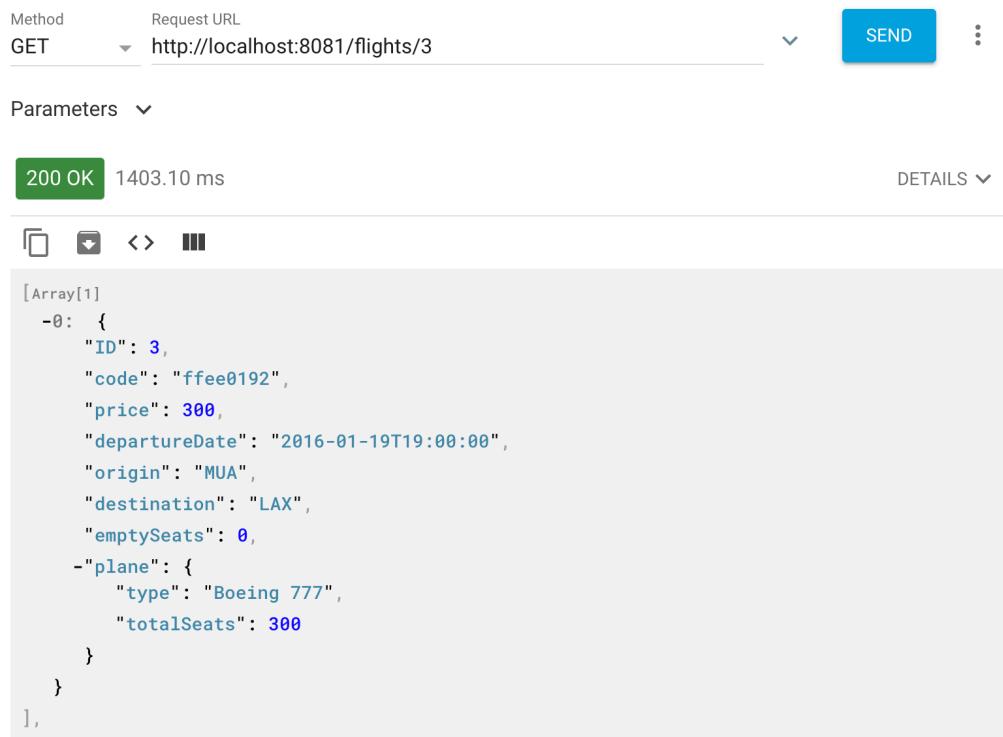
```
SELECT *
FROM american
WHERE ID = :ID
```

Under 'Input Parameters', there is one entry: '1 {"ID" : attributes.uriParams.ID}'.

Test the application

21. Save the file to redeploy the project.

22. In Advanced REST Client, make another request to <http://localhost:8081/flights/3>; you should now see the info for the flight with an ID of 3.



Method: GET Request URL: http://localhost:8081/flights/3

Parameters: **▼**

200 OK 1403.10 ms DETAILS **▼**

[Array[1]]

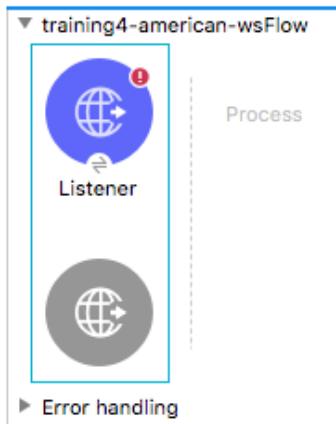
```
[{"ID": 3, "code": "ffee0192", "price": 300, "departureDate": "2016-01-19T19:00:00", "origin": "MUA", "destination": "LAX", "emptySeats": 0, "plane": {"type": "Boeing 777", "totalSeats": 300}}]
```

23. Return to Anypoint Studio.

Make a new flow to handle post requests

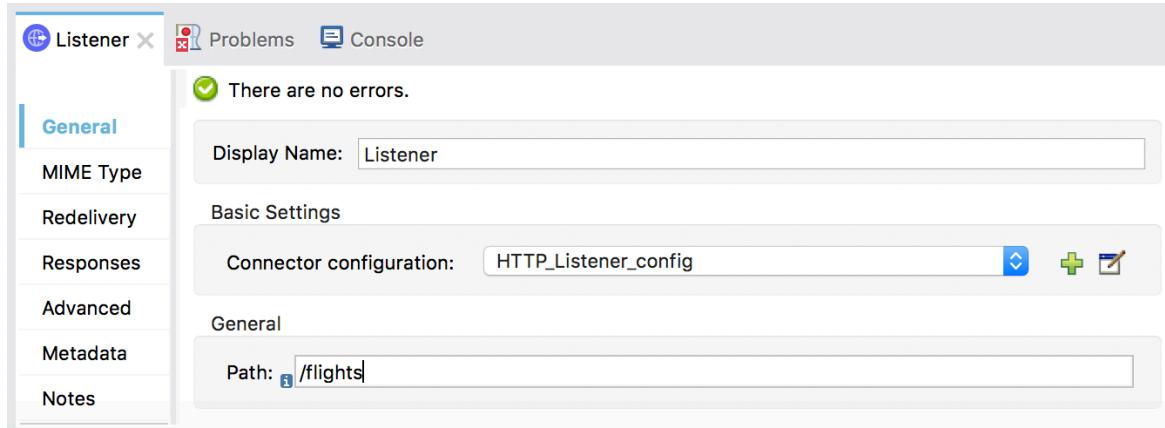
24. In the Mule Palette, select HTTP.

25. Drag Listener from the Mule Palette and drop it in the canvas below the two existing flows.

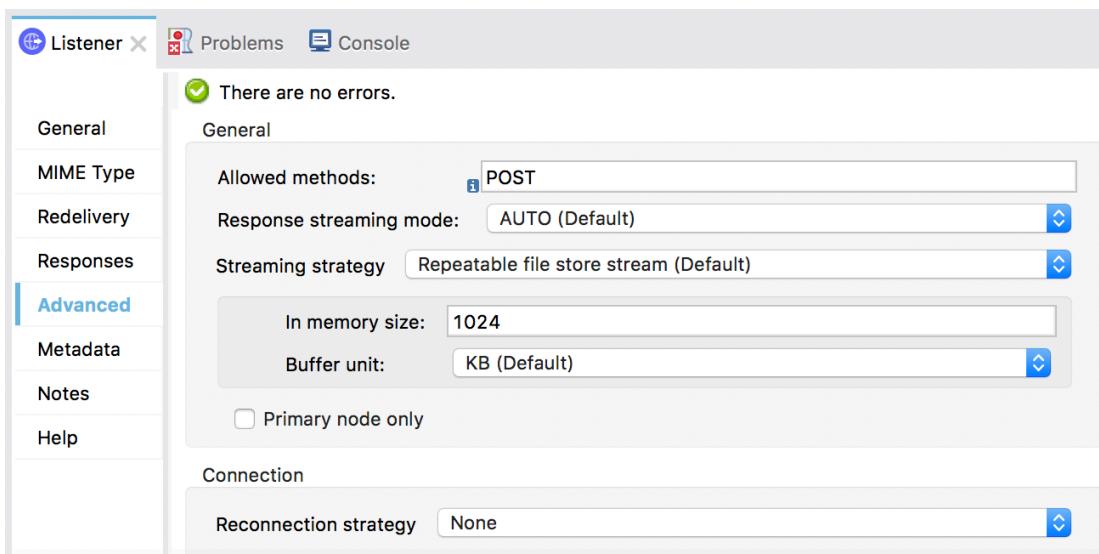


26. Change the name of the flow to postFlight.

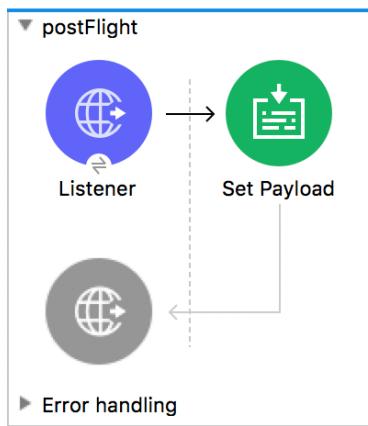
27. In the Listener properties view, ensure the connector configuration is set to the existing HTTP_Listener_config.
28. Set the path to /flights.



29. In the left-side navigation of the Listener properties view, select Advanced.
30. Set the allowed methods to POST.



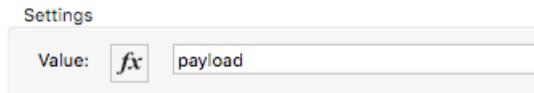
31. Drag the Set Payload transformer from the Mule Palette and drop it in the process section of the flow.



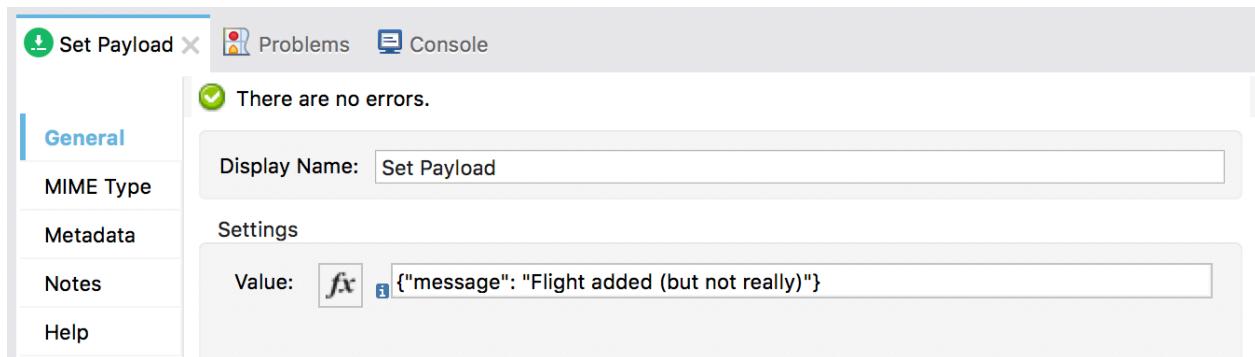
32. Return to the course snippets.txt file and copy the American Flights API - /flights POST response example.

```
{"message": "Flight added (but not really)"}
```

33. Return to Anypoint Studio and in the Set Payload properties view, click the Switch to literal mode button for the value field.



34. Set the value field to the value you copied.



Note: This flow is just a stub. For it to really work and add data to the database, you would need to add logic to insert the request data to the database.

Test the application

35. Save the file to redeploy the project.

36. In Advanced REST Client, change the request type from GET to POST.

37. Click Send; you should get a 405 Method Not Allowed response.

The screenshot shows the Advanced REST Client interface. The 'Method' dropdown is set to 'POST'. The 'Request URL' field contains 'http://localhost:8081/flights/3'. A large orange button labeled '405 Method Not Allowed' is displayed, indicating the status code and response time (5.90 ms). Below the status bar, there are four small icons: a clipboard, a download arrow, a refresh/circular arrow, and an eye. The main message area displays the error: 'Method not allowed for endpoint: /flights/3'.

38. Remove the URI parameter from the request URL: <http://localhost:8081/flights>.

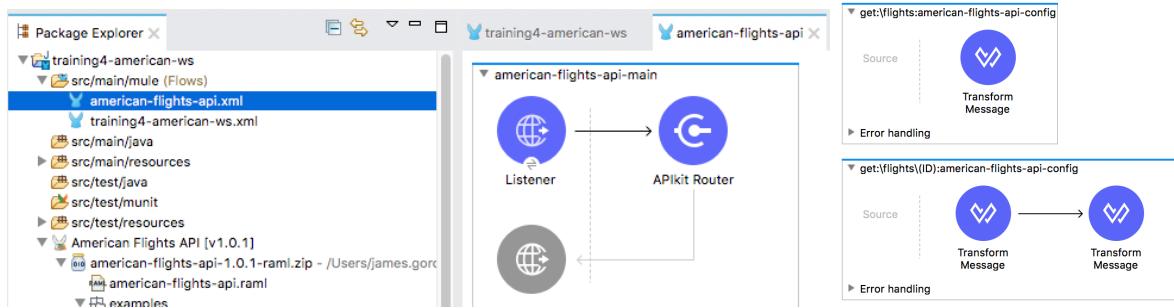
39. Send the request; you should now see the message the flight was added – even though you did not send any flight data to add.

The screenshot shows the Advanced REST Client interface. The 'Method' dropdown is set to 'POST'. The 'Request URL' field contains 'http://localhost:8081/flights'. A green button labeled '200 OK' is displayed, indicating the status code and response time (16.20 ms). Below the status bar, there are four small icons: a clipboard, a download arrow, a refresh/circular arrow, and an eye. The main message area displays the response body: '{"message": "Flight added (but not really)"}'.

Walkthrough 4-5: Use Anypoint Studio to create a RESTful API interface from a RAML file

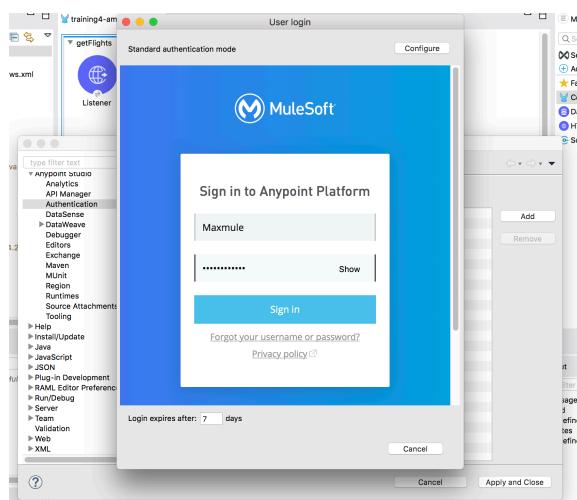
In this walkthrough, you generate a RESTful interface from the RAML file. You will:

- Add Anypoint Platform credentials to Anypoint Studio.
- Import an API from Exchange into an Anypoint Studio project.
- Use APIkit to generate a RESTful web service interface from an API.
- Test a web service using APIkit console and Advanced REST Client.



Add Anypoint Platform credentials to Anypoint Studio

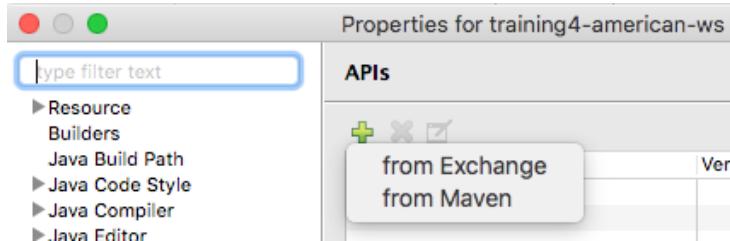
1. Return to Anypoint Studio.
2. In the Package Explorer, right-click the training4-american-ws project and select Anypoint Platform > Configure Credentials.
3. In the Authentication page of the Preferences dialog box, click the Add button.
4. In the User login dialog box, enter your username and password and click Sign In.



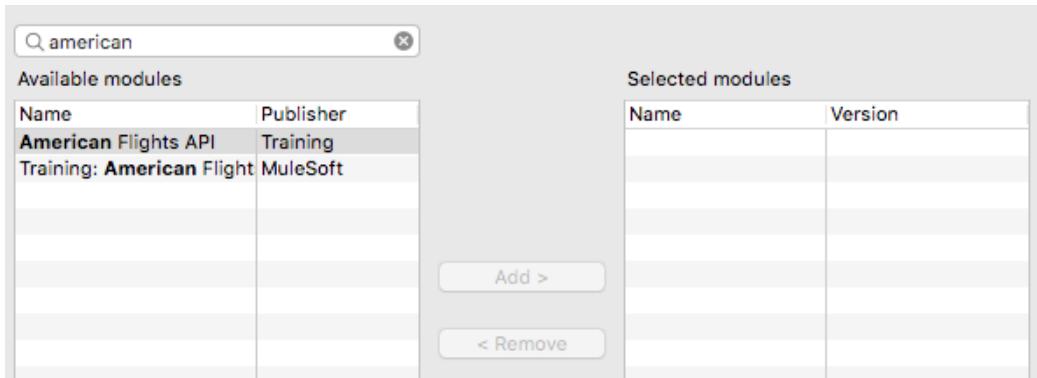
5. On the Authentication page, make sure your username is listed and selected.
6. Click Apply and Close.

Add an API from Exchange to the Anypoint Studio project

7. Right-click the training4-american-ws project and select Manage Dependencies > Manage APIs.
8. In the Properties for training4-american-ws dialog box, click the Add button and select from Exchange.



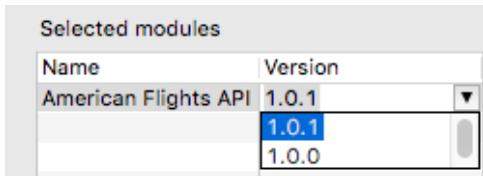
9. In the Add Dependencies to Project dialog box, enter american in the search field.



10. Select your American Flights API (**not** the Training: American Flights API) and click Add.

Note: If you did not successfully create the American Flights API in the first part of the course, you can use the pre-built Training: American Flights API connector.

11. In the selected modules, click American Flights API, then the 1.0.1 version then the version's down-arrow.



12. Note the list of available versions then select 1.0.1.

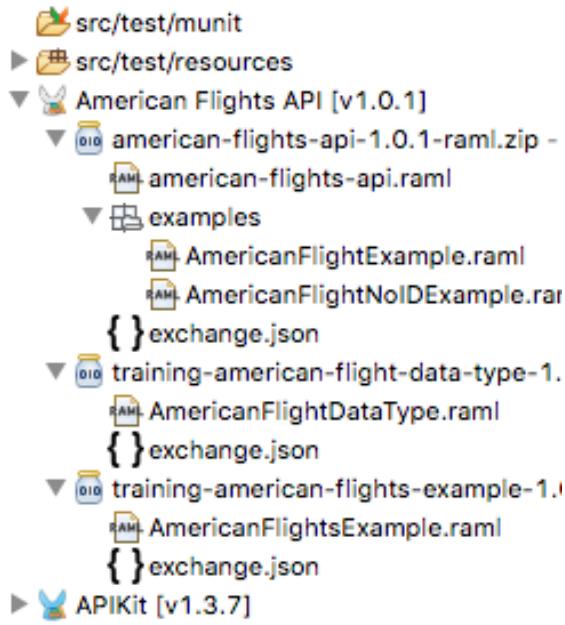
13. Click Finish.

14. In the Properties for training4-american-ws dialog box, click Apply and Close.

15. Click Yes when prompted to scaffold the American Flights API specification.

Locate the API files added to the project

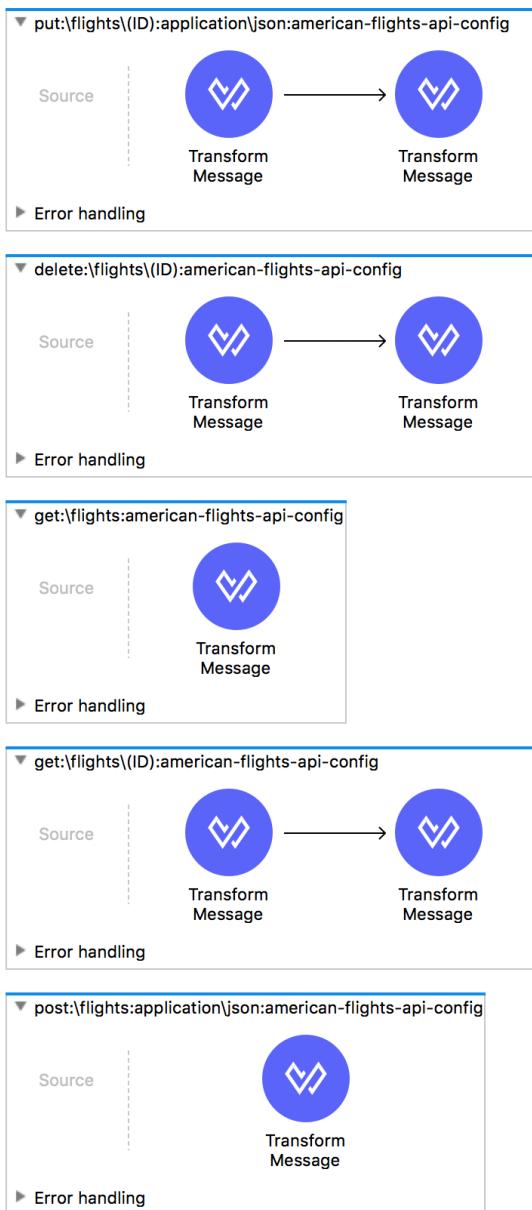
16. In the Package Explorer, American Flights API [v1.0.1].
17. Expand the API; you should see the RAML files imported from Exchange.



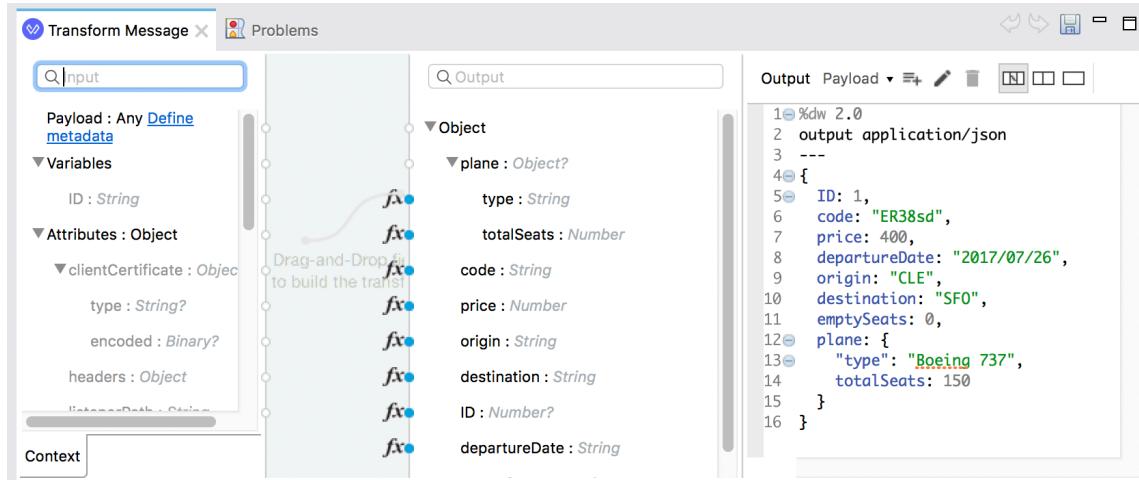
Examine the XML file created

18. Examine the generated american-flights-api.xml file and locate the following five flows:

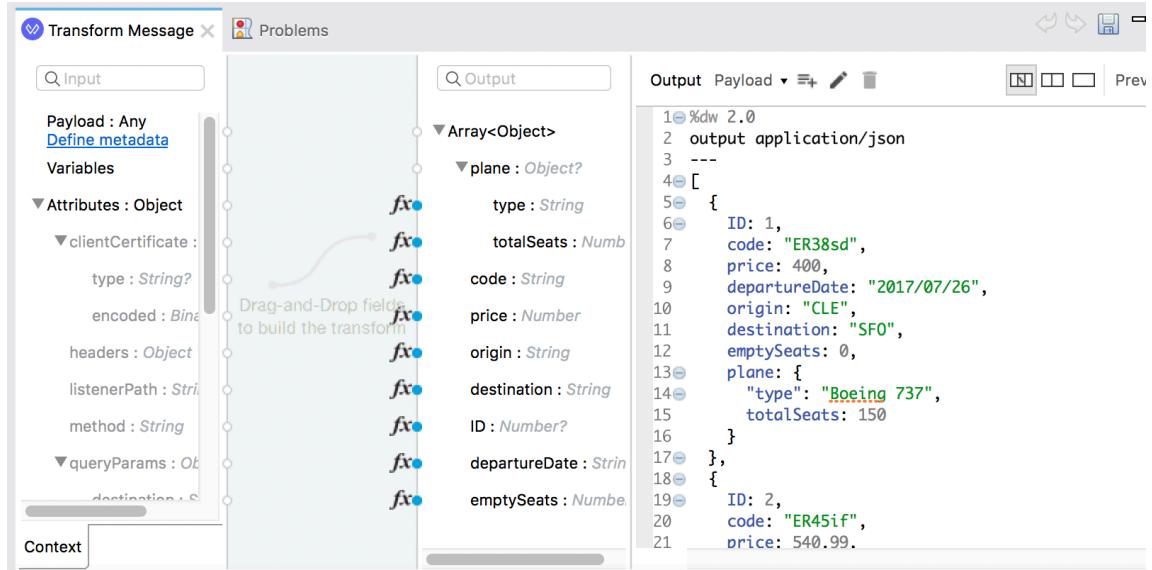
- put:/flights/{ID}
- delete:/flights/{ID}
- get:/flights
- get:/flights/{ID}
- post:/flights



19. In the get:/flights/{ID} flow, double-click the second Transform Message component and look at the output JSON in the properties view.



20. In the get:/flights flow, double-click the Transform Message component and look at the output JSON in the properties view.

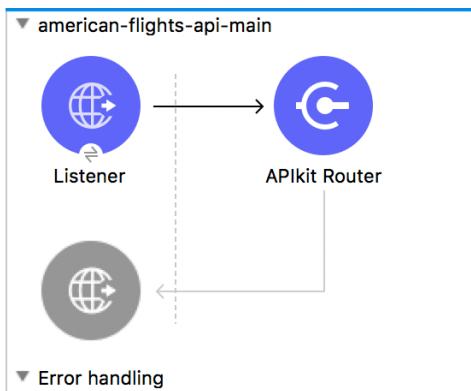


21. In the post:/flights flow, double-click the Transform Message component and look at the output JSON in the properties view.



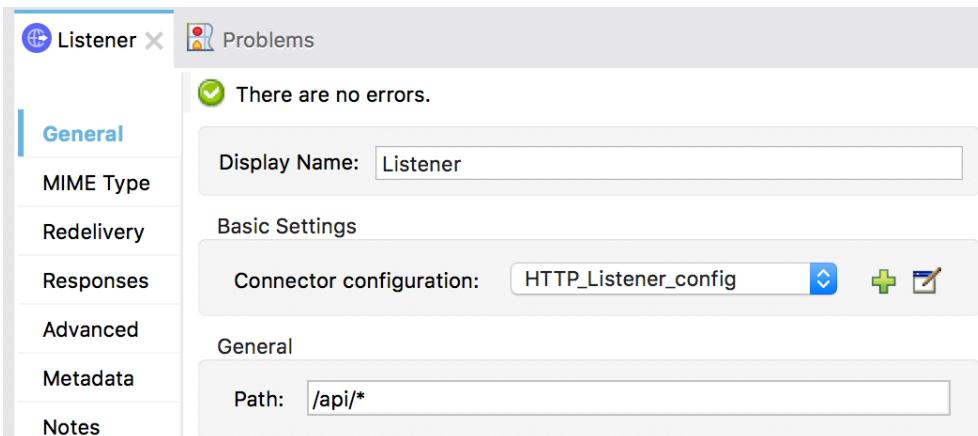
Examine the main flow and its HTTP Listener

22. Locate the american-flights-api-main flow.
23. Double-click its HTTP Listener.



24. In the Listener properties view, notice that the connector configuration is the existing `HTTP_Listener_config` and that path is set to `/api/*`.

*Note: The * is a wildcard allowing any characters to be entered after /api/.*

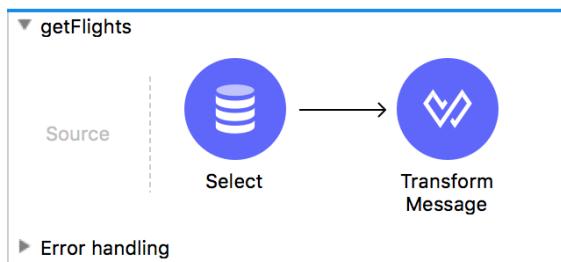


25. Click the Edit button for the connector configuration; you should see that the same port 8081 is used as the HTTP Listener you created previously.
26. Click Cancel.

Remove the other listeners

27. Return to `training4-american-ws.xml`.

28. In the Message Flow view, right-click the HTTP Listener in getFlights and select Delete.

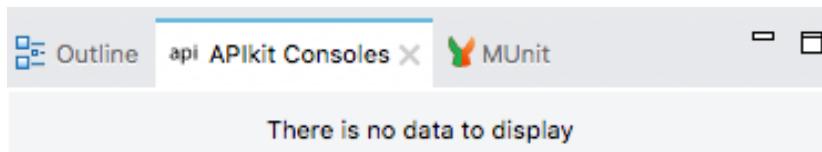


29. Delete the other two HTTP Listeners.

Test the web service using APIkit console

30. Save the files.

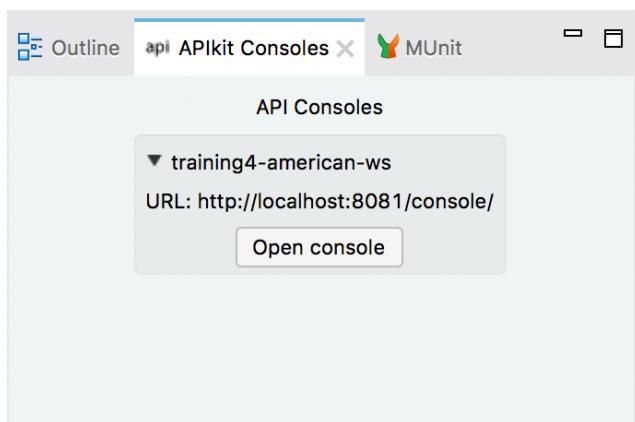
31. Locate the new APIkit Consoles view that is created and opened in Anypoint Studio and note that there is no data to display.



32. Stop the project.

33. Run the project and wait until Mule and the application restart.

34. Verify that the APIkit Consoles view is now populated.



35. Click the Open console button; a browser window should be open with an API console.

The screenshot shows the API console interface for the American Flights API. The top navigation bar has tabs for 'Summary' and 'API console'. Below the navigation, there's a summary of the API: 'American Flights API' and 'Version: v1'. On the left sidebar, there are sections for 'Endpoints', '/flights', '/{ID}', and 'Types'. Under '/flights', it shows the URL 'http://localhost:8081/api'. The main area is titled 'API endpoints' and lists two entries: '/flights' (with 'GET' and 'POST' methods) and '/flights/{ID}' (with 'GET', 'DELETE', and 'PUT' methods).

36. Select the GET method for /flights.

The screenshot shows the API console interface for the American Flights API. The top navigation bar has tabs for 'Summary' and 'API console'. Below the navigation, there's a summary of the API: 'American Flights API' and 'Version: v1'. On the left sidebar, there are sections for 'Endpoints', '/flights', '/{ID}', and 'Types'. Under '/flights', the 'GET' method is selected. The main area shows the request configuration: 'Request URL' is set to 'http://localhost:8081/api/flights', 'Parameters' tab is selected, and 'Query parameters' section is visible. It shows a 'destination' parameter of type 'String' with enum values: SFO, LAX, CLE. A 'SEND' button is present at the bottom.

37. Click Send; you should get a 200 response with the example flight data – not all the flights.

The screenshot shows the API console interface for the American Flights API. The top navigation bar has tabs for 'Summary' and 'API console'. Below the navigation, there's a summary of the API: 'American Flights API' and 'Version: v1'. On the left sidebar, there are sections for 'Endpoints', '/flights', '/{ID}', and 'Types'. Under '/flights', the 'GET' method is selected. The main area shows the request configuration: 'Request URL' is set to 'http://localhost:8081/api/flights', 'Parameters' tab is selected, and 'Query parameters' section is visible. It shows a 'destination' parameter of type 'String' with enum values: SFO, LAX, CLE. A 'SEND' button is present at the bottom. After clicking 'SEND', the response is shown: '200 OK' status with a duration of '115.59 ms'. The 'Body' section shows an array of flight data with one item: { "ID": 1, "code": "ER38sd" }.

Note: If you use the pre-built Training: American Flights API connector, you must enter any values for client_id and client_secret on the Headers tab.

38. Close the browser window.

Test the web service using Advanced REST Client

39. Return to Advanced REST Client.
40. Change the method to GET and click Send to make a request to <http://localhost:8081/flights>; you should get a 404 Not Found response.

The screenshot shows the Advanced REST Client interface. The 'Method' dropdown is set to 'GET'. The 'Request URL' field contains 'http://localhost:8081/flights'. A large orange button labeled 'SEND' is visible. Below the URL input, there's a 'Parameters' dropdown. The main response area shows a red box containing '404 Not Found' and '4.60 ms'. To the right is a 'DETAILS' button. Below this, there are four small icons: a square with a minus sign, a square with a checkmark, a double-headed arrow, and an eye. The text 'No listener for endpoint: /flights' is displayed.

41. Change the URL to <http://localhost:8081/api/flights> and send the request; you should get a 200 response with the example flight data.

The screenshot shows the Advanced REST Client interface. The 'Request URL' field now contains 'http://localhost:8081/api/flights'. The response status is '200 OK' with a time of '31.10 ms'. A green 'DETAILS' button is present. Below the status, there are four small icons: a square with a minus sign, a square with a checkmark, a double-headed arrow, and a vertical bar. The main content area displays a JSON array of two flight objects. The first flight has ID 1, code ER38sd, price 400, departure date 2017/07/26, origin CLE, destination SFO, and 0 empty seats. It is associated with a Boeing 737 plane with 150 total seats. The second flight has ID 2, code ER45if, and no other details shown.

```
[Array[2]
-0: {
  "ID": 1,
  "code": "ER38sd",
  "price": 400,
  "departureDate": "2017/07/26",
  "origin": "CLE",
  "destination": "SFO",
  "emptySeats": 0,
  "-plane": {
    "type": "Boeing 737",
    "totalSeats": 150
  }
},
-1: {
  "ID": 2,
  "code": "ER45if",
}
```

Note: If you use the pre-built Training: American Flights API connector, you must add client_id and client_secret headers with values from the course snippets.txt file.

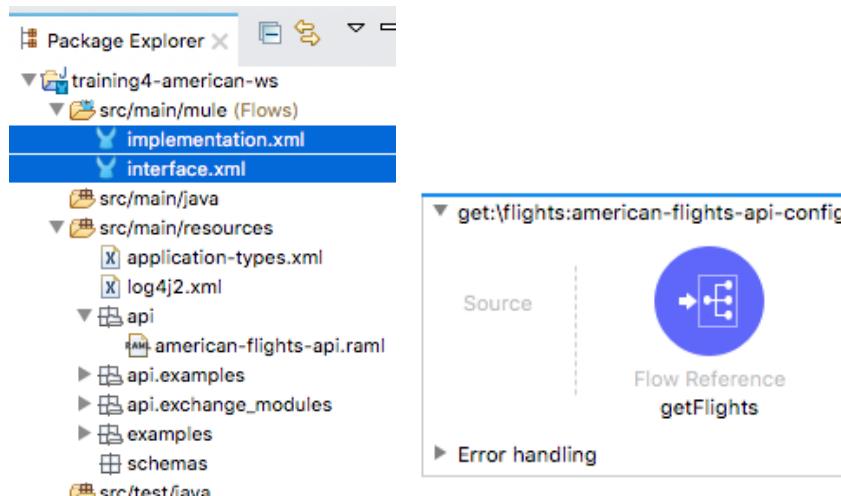
42. Make a request to <http://localhost:8081/api/flights/3>; you should see the example data returned for a flight with an ID of 1.

```
200 OK 45.04 ms DETAILS ▾  
  
[{"  
    "ID": 1,  
    "code": "ER38sd",  
    "price": 400,  
    "departureDate": "2017/07/26"  
}]
```

Walkthrough 4-6: Implement a RESTful web service

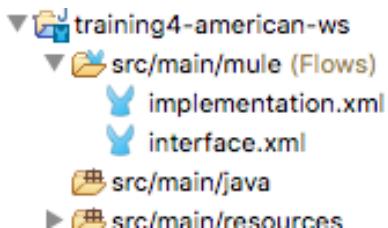
In this walkthrough, you wire the RESTful web service interface up to your back-end logic. You will:

- Pass an event from one flow to another.
- Call the backend flows.
- Create new logic for the nested resource call.
- Test the web service using Advanced REST Client.



Rename the configuration files

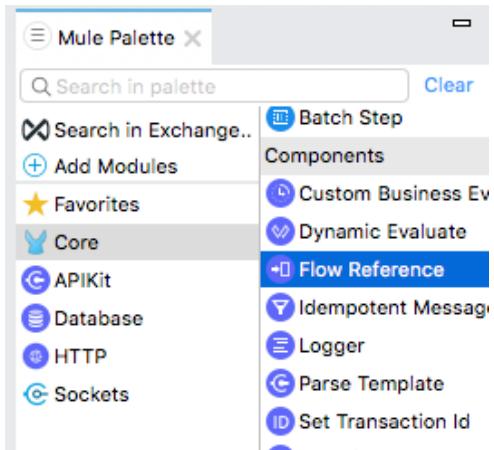
1. Return to Anypoint Studio.
2. Right-click `american-flights-api.xml` in the Package Explorer and select Refactor > Rename.
3. In the Rename Resource dialog box, set the new name to `interface.xml` and click OK.
4. Right-click `training4-american-ws.xml` and select Refactor > Rename.
5. In the Rename Resource dialog box, set the new name to `implementation.xml` and click OK.



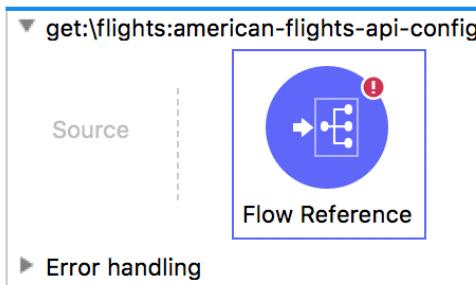
Use a Flow Reference in the /flights resource

6. Open `interface.xml`.
7. Delete the Transform Message component in the `get:/flights` flow.

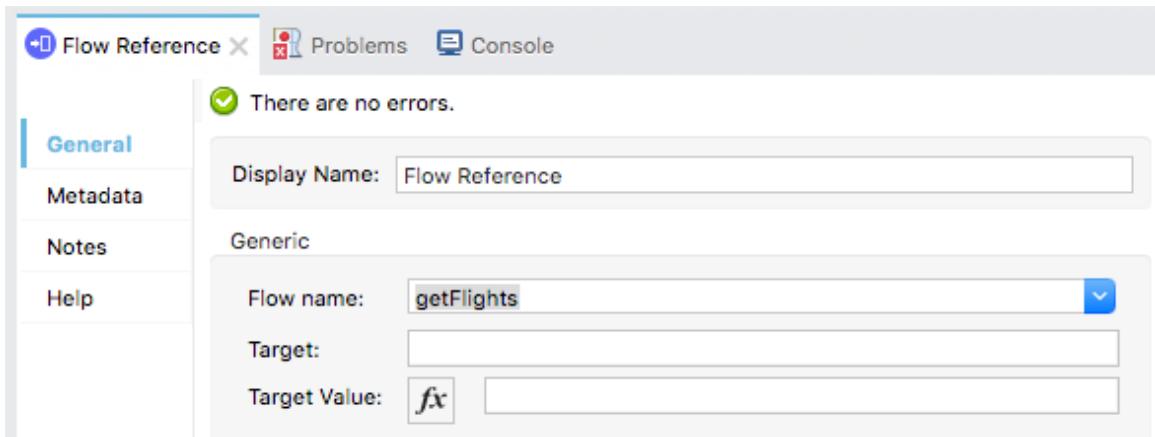
8. In the Mule Palette, select Core and locate the Components section in the right-side.



9. Drag a Flow Reference component from the Mule Palette and drop it into the process section of the flow.



10. In the Flow Reference properties view, select getFlights for the flow name.

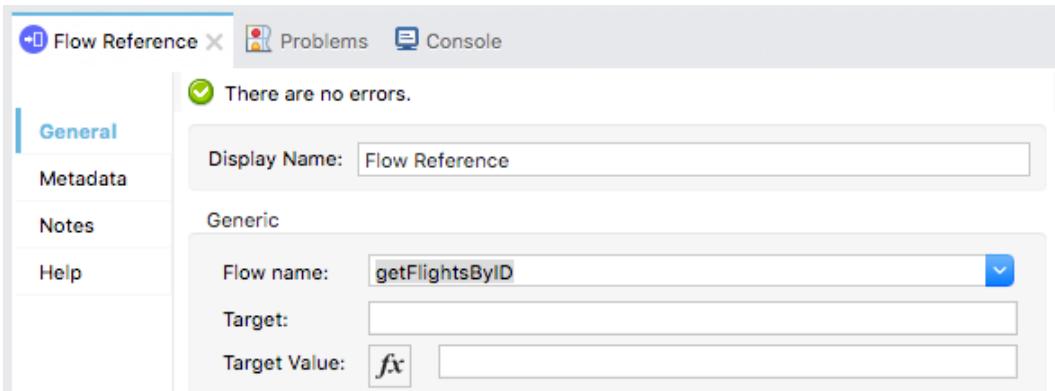


11. Change the display name to getFlights.

Use a Flow Reference in the /flights/{ID} resource

12. Delete both the Transform Message components in the get:/flights/{ID} flow.

13. Drag a Flow Reference component from the Mule Palette and drop it into the flow.
14. In the Flow Reference properties view, select getFlightsByID for the flow name.



15. Change the display name to getFlightsByID.

Test the web service using Advanced REST Client

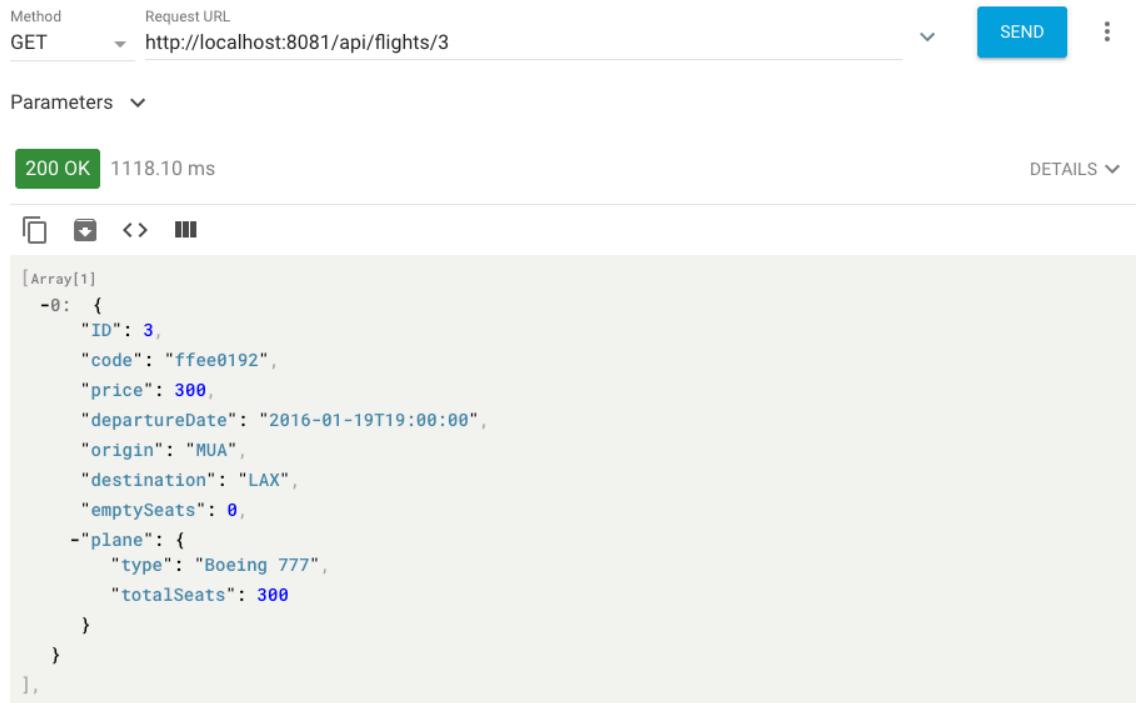
16. Save the file to redeploy the project.
17. In Advanced REST Client, make a request to <http://localhost:8081/api/flights>; you should now get the data for all the flights from the database instead of the sample data.

The screenshot shows the Advanced REST Client interface. The method is set to 'GET' and the request URL is 'http://localhost:8081/api/flights'. The response status is '200 OK' and the response time is '3256.70 ms'. The response body is an array of flight objects:

```
[Array[11]
-0: {
  "ID": 1,
  "code": "rree0001",
  "price": 541,
  "departureDate": "2016-01-19T19:00:00",
  "origin": "MUA",
  "destination": "LAX",
  "emptySeats": 0,
  "plane": {
    "type": "Boeing 787",
    "totalSeats": 200
  }
},
-1: {
  "ID": 2,
  "code": "eefd0123",
  "price": 300,
  "departureDate": "2016-01-24T19:00:00",
  "origin": "MUA",
  "destination": "CLE",
  "emptySeats": 7,
  "plane": {
    "type": "Boeing 747",
    "totalSeats": 345
  }
},
-2: {
  "ID": 3,
  "code": "ffee0192",
  "price": 300,
  "departureDate": "2016-01-19T19:00:00",
  "origin": "MUA",
  "destination": "LAX",
  "emptySeats": 0,
  "plane": {
    "type": "Boeing 787",
    "totalSeats": 200
  }
}...]
```

Note: If you use the pre-built Training: American Flights API connector, you must add client_id and client_secret headers with values from the course snippets.txt file.

18. Make a request to <http://localhost:8081/api/flights/3>; you should now get the data that flight from the database instead of the sample data.



Method: GET Request URL: http://localhost:8081/api/flights/3

Parameters: ▾

200 OK 1118.10 ms DETAILS ▾

◻ □ < > III

```
[Array[1]
-0: {
  "ID": 3,
  "code": "ffee0192",
  "price": 300,
  "departureDate": "2016-01-19T19:00:00",
  "origin": "MUA",
  "destination": "LAX",
  "emptySeats": 0,
  "plane": {
    "type": "Boeing 777",
    "totalSeats": 300
  }
},
]
```

19. Return to Anypoint Studio.

20. Stop the project.

Module 5: Deploying and Managing APIs



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

- Describe the options for deploying Mule applications.
- Deploy Mule applications to CloudHub.
- Use API Manager to create and deploy API proxies.
- Use API Manager to restrict access to API proxies.

Walkthrough 5-1: Deploy an application to CloudHub

In this walkthrough, you deploy and run your application on CloudHub. You will:

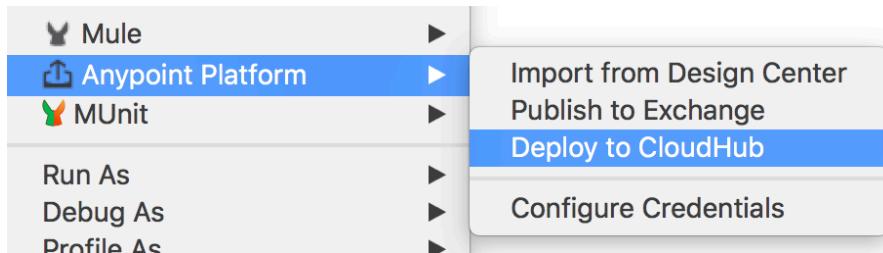
- Deploy an application from Anypoint Studio to CloudHub.
- Run the application on its new, hosted domain.
- Make calls to the web service.
- Update an API implementation deployed to CloudHub.

The screenshot shows the 'Runtime Manager' interface. At the top, there's a navigation bar with 'Training' and 'MM' buttons. Below it is a search bar with 'Deploy application' and a 'Search Applications' field. On the left, a sidebar has tabs for 'Sandbox', 'Applications' (which is selected), 'Servers', and 'Alerts'. The main area displays a table with columns 'Name', 'Server', 'Status', and 'File'. A single row is shown: 'training4-american-ws-maxmule' is running on 'CloudHub' with a green 'Started' status and the file 'training4-american-ws-1.0.0-SNAPSHOT-mule-application.jar'.

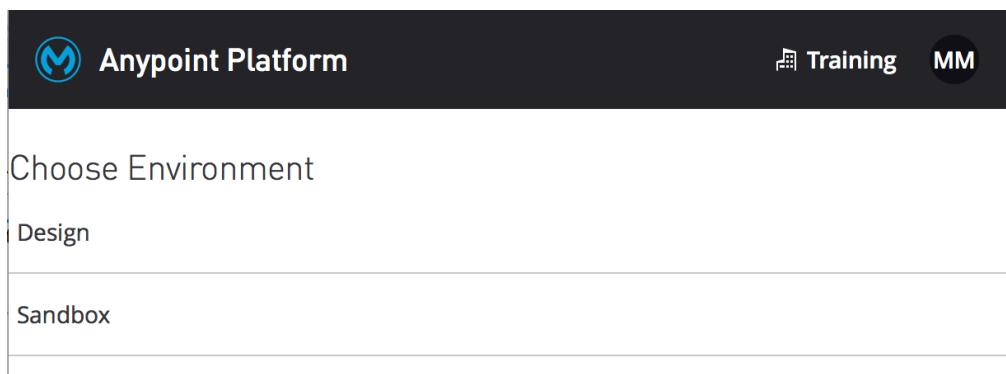
Note: If you do not have a working application at this point, import the wt4-6_training4-american-ws_solution.jar solution into Anypoint Studio and work with that project.

Deploy the application to CloudHub

1. Return to Anypoint Studio.
2. In the Package Explorer, right-click the project and select Anypoint Platform > Deploy to CloudHub.



3. In the Choose Environment dialog box, select Sandbox.



- At the top of the Anypoint Platform dialog box, set the application name to training4-american-ws-{your-lastname} so it is a unique value.

Note: This name will be part of the URL used to access the application on CloudHub. It must be unique across all applications on CloudHub. The availability of the domain is instantly checked and you will get a green check mark if it is available.

The screenshot shows the 'Deploying Application' dialog box. The application name is 'training4-american-ws-maxmule'. The deployment target is 'CloudHub' and the application file is 'training4-american-ws.jar'. The 'Runtime' tab is selected. Other tabs include 'Properties', 'Insight', 'Logging', and 'Static IPs'.

- Make sure the runtime version is set to the version your project is using.

Note: If you don't know what version it is using, look at the Package Explorer and find a library folder with the name of the server being used, like Mule Server 4.2.0 EE.

- Make sure the worker size to 0.1 vCores.

Runtime	Properties	Insight	Logging	Static IPs
Runtime version 4.2.0	Worker size 0.1 vCores	Workers 1		

- Click the Deploy Application button.
- Click the Open in Browser button.

The screenshot shows the 'Runtime Manager' dialog box. It displays the message 'Deploying training4-american-ws-maxmule to CloudHub'. Below it says 'You may close this window at any time.' There are two buttons: 'Open in Browser' and 'Close Window'.

- In the Anypoint Platform browser window that opens, locate the status of your deployment in the Runtime Manager.

The screenshot shows the Runtime Manager interface. On the left, there's a sidebar with 'Sandbox' selected. The main area has a 'Deploy application' button and a search bar. A table lists one application: 'training4-american-ws-maxmule' running on 'CloudHub' with a 'Status' of 'Started'. To the right of the table, the application file path is shown: 'training4-american-ws-1.0.0-SNAPSHOT-mule-application.jar'.

Watch the logs and wait for the application to start

- Click in the row of the application (not on its name); you should see information about the application appear on the right side of the window.

This screenshot shows the Runtime Manager interface with more detailed information for the deployed application. The application 'training4-american-ws-maxmule' is listed in the table, and its details are expanded on the right. The details include: Status: Started, Server: CloudHub, File: training4-american-ws-1.0.0-SNAPSHOT-mule-application.jar. Below this, deployment metadata is shown: Last Updated: 2019-05-08 8:45:08PM, App url: training4-american-ws-maxmule.us-e2.cloudhub.io. Configuration details are also provided: Runtime version: 4.2.0, Worker size: 0.1 vCores, Workers: 1, Region: US East (Ohio). At the bottom, there are buttons for 'Manage Application', 'Logs', and 'Insight', along with a link to 'View Associated Alerts'.

- Click the Logs button.
- Watch the logs as the application is deployed.

13. Wait until the application starts (or fails to start).

The screenshot shows the Mule Runtime Manager interface. On the left, there's a navigation sidebar with options like Applications, Dashboard, Insight, Logs (which is selected), Application Data, Queues, Schedules, and Settings. The main area displays the logs for the application 'training4-american-ws-maxmule'. The logs show the application starting up, including the configuration flow and the deployment log entry indicating success.

```
Starting flow: post:\flights:application\json:american-flights-api-config
20:45:08.771 05/08/2019 Worker-0 qtp485578806-35 INFO
Starting Bean: listener

20:45:08.796 05/08/2019 Worker-0 qtp485578806-35 INFO
Starting Bean: listener

20:45:08.820 05/08/2019 Worker-0 qtp485578806-35 INFO

*****
* Application: training4-american-ws-maxmule *
* OS encoding: UTF-8, Mule encoding: UTF-8 *
*
*****
20:45:08.965 05/08/2019 Deployment system SYSTEM
Worker(18.219.72.242): Your application has started successfully.

20:45:09.394 05/08/2019 Deployment system SYSTEM
Your application is started.
```

On the right side, there's a 'Deployments' panel showing a deployment log entry for '20:44 - Deployment'.

Note: If your application did not successfully deploy, read the logs to help figure out why the application did not deploy. If you had errors when deploying, troubleshoot them, fix them, and then redeploy.

Test the application

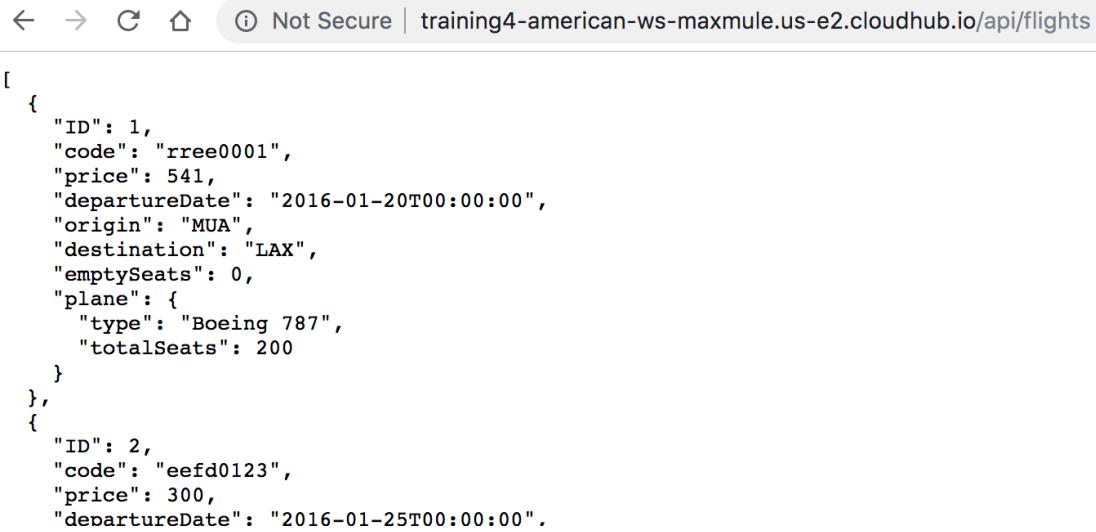
14. In the left-side navigation, select Dashboard.
15. Locate the link for the application on its new domain:
training4-american-ws-{lastname}.{region}.cloudbhub.io.

The screenshot shows the Mule Runtime Manager interface with the 'Dashboard' selected in the sidebar. It displays the application 'training4-american-ws-maxmule' and its domain as 'training4-american-ws-maxmule.us-e2.cloudbhub.io'. There are also sections for 'Mule messages'.

Note: {region} represents the worker region to which the Mule application is deployed. In North America, the default region is US East and is denoted by us-e2 in the application URL.

16. Click the link; a request will be made to that URL in a new browser tab and you should get a message that there is no listener for that endpoint.

17. Modify the path to <http://training4-american-ws-{lastame}.{region}.cloudbus.io/api/flights>; you should see the flights data.



The screenshot shows a browser window with the URL <http://training4-american-ws-maxmule.us-e2.cloudbus.io/api/flights>. The page displays a JSON array of flight records:

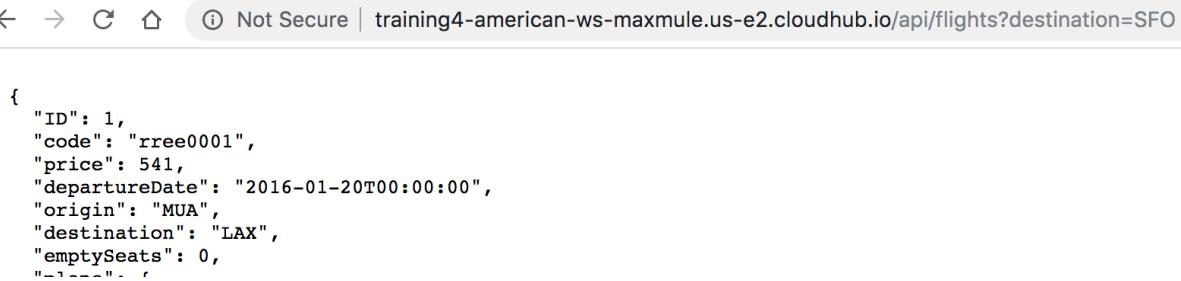
```
[{"ID": 1, "code": "rree0001", "price": 541, "departureDate": "2016-01-20T00:00:00", "origin": "MUA", "destination": "LAX", "emptySeats": 0, "plane": {"type": "Boeing 787", "totalSeats": 200}}, {"ID": 2, "code": "eefd0123", "price": 300, "departureDate": "2016-01-25T00:00:00"}]
```

Note: If you are using the local Derby database, your application will not return results when deployed to CloudHub. You will update the application with a version using the MySQL database in the next section, so it works.

18. Add a query parameter called destination to the URL and set it equal to SFO.

19. Send the request; you should still get all the flights.

Note: You did not add logic to the application to search for a particular destination. You will deploy an application with this additional functionality implemented next.



The screenshot shows a browser window with the URL <http://training4-american-ws-maxmule.us-e2.cloudbus.io/api/flights?destination=SFO>. The page displays a JSON array of flight records, filtered by destination:

```
[{"ID": 1, "code": "rree0001", "price": 541, "departureDate": "2016-01-20T00:00:00", "origin": "MUA", "destination": "LAX", "emptySeats": 0, "plane": {"type": "Boeing 787", "totalSeats": 200}}]
```

20. Leave this browser tab open.

Update the API implementation deployed to CloudHub

21. Return to the browser tab with Runtime Manager.
22. In the left-side navigation, select Settings.
23. Click the Choose file button and select Upload file.
24. Browse to the jars folder in the student files.

25. Select training4-american-ws-v2.jar and click Open.

Note: This updated version of the application adds functionality to return results for a particular destination. You will learn to do this later in the Development Fundamentals course.

26. Click the Apply Changes button.

The screenshot shows the 'Runtime Manager' interface. On the left, a sidebar lists 'Sandbox', 'Applications' (with 'Dashboard' selected), 'Insight', 'Logs', 'Application Data', 'Queues', 'Schedules', and 'Settings' (selected). The main area displays the application 'training4-american-ws-maxmule'. Under 'Application File', the file 'training4-american-ws-v2.jar' is selected. Below it, the 'App url' is shown as 'training4-american-ws-maxmule.us-e2.cloudhub.io'. The configuration tab 'Runtime' is active, showing 'Runtime version' as '4.2.0', 'Worker size' as '0.1 vCores', and 'Workers' set to '1'. A note says 'Recommendation: 2+ workers to form a CloudHub Fabric, for added reliability'. Under 'Region', 'US East (Ohio)' is selected. In the 'Settings' section, checkboxes are shown for 'Automatically restart application when not responding' (checked), 'Persistent queues' (unchecked), 'Encrypt persistent queues' (unchecked), and 'Use Object Store v2' (unchecked). At the bottom right is a blue 'Apply Changes' button.

27. Wait until the application is uploaded and then redeloys successfully.

Note: Because this can take some time for trial accounts, your instructor may move on with the next topic and then come back to test this later.

28. Close the browser tab with Runtime Manager.

Test the updated application

29. Return to the browser tab with a request to the API implementation on CloudHub with a destination of SFO and refresh it; you should now get only flights to SFO.

```
[{"ID": 5, "code": "rree1093", "price": 142, "departureDate": "2016-02-11T00:00:00", "origin": "MUA", "destination": "SFO", "emptySeats": 1, "plane": {"type": "Boeing 737", "totalSeats": 150}}, {"ID": 7, "code": "eefd1994", "price": 676, "departureDate": "2016-01-01T00:00:00", "origin": "MUA", "destination": "SFO", "emptySeats": 0, "plane": {"type": "Boeing 777", "totalSeats": 300}}, {"ID": 8, "code": "ffcc2000"}]
```

30. Close this browser tab.

Walkthrough 5-2: Create and deploy an API proxy

In this walkthrough, you create and deploy an API proxy for your API implementation on CloudHub. You will:

- Add an API to API Manager.
- Use API Manager to create and deploy an API proxy application.
- Set a proxy consumer endpoint so requests can be made to it from Exchange.
- Make calls to an API proxy from API portals for both internal and external developers.
- View API request data in API Manager.

The screenshot shows the Runtime Manager interface. On the left, a modal window titled "Endpoint configuration details" displays a flow diagram: "Client" connects to "API Gateway (CloudHub)" which then connects to "API Implementation" (<http://training4-american-ws-maxmule.us-e2.cloudhub.io/api>). On the right, the main "Runtime Manager" screen shows a "Deploy application" button and a search bar. Below these are tabs for "Applications", "Servers", "Alerts", and "VPCs". A table lists two applications: "training4-american-ws-maxmule" and "training4-american-ws-maxmule", both running on "CloudHub" and in the "Started" status.

Create and deploy a proxy application

1. Return to Anypoint Platform.
2. In the main menu, select API Manager; you should see no APIs listed.

The screenshot shows the API Manager interface. The top navigation bar includes "Training" and "MM" buttons. The main dashboard displays a message: "No APIs to display. Get started by adding your first API." To the left, a sidebar under "API Administration (Sandbox)" shows "Sandbox" and "API Administration" sections with links for "Automated Policies", "Client Applications", "Custom Policies", and "Analytics".

3. Click the Manage API button and select Manage API from Exchange.

The screenshot shows the API Manager interface with the "Manage API" button highlighted. A dropdown menu appears, showing "Manage API from Exchange" and "Manage an existing Exchange asset". Other options like "Create new API" and "New" are also visible.

4. For API name, start typing American in the text field and then select your American Flights API in the drop-down menu that appears.
5. Set the rest of the fields to the following values:
 - Asset type: RAML/OAS
 - API version: v1
 - Asset version: 1.0.1
 - Managing type: Endpoint with Proxy
 - Proxy deployment target: CloudHub
 - Implementation URI: `http://training4-american-ws-{lastname}.{region}.cloudbhub.io/api`

API Manager

API Administration (Sandbox) [Get from Exchange](#)

SANDBOX

Manage API from Exchange

← API Administration

API Configurations

API name:	American Flights API	<input type="button" value="Q"/>
Asset type:	RAML/OAS	<input type="button" value="▼"/>
API version:	v1	<input type="button" value="▼"/> View API in Exchange
Asset version:	1.0.1	<input type="button" value="▼"/>
Managing type:	<input type="radio"/> Basic Endpoint <input checked="" type="radio"/> Endpoint with Proxy	
Proxy deployment target:	<input checked="" type="radio"/> CloudHub <input type="radio"/> Hybrid <input type="radio"/> Runtime Fabric	
Mule Version:	<input type="checkbox"/> Check this box if you are managing this API in Mule 4 or above.	
Implementation URI:	<input type="text" value="http://training4-american-ws-maxmule.us-e2.cloudbhub.io/api"/> <input type="button" value=""/>	
Base URI not defined in the API specification.		
Path:	<input type="text" value="/"/>	
Advanced options >		

6. Select the checkbox: Check this box if you are managing this API in Mule 4 or above.

Mule Version: Check this box if you are managing this API in Mule 4 or above.

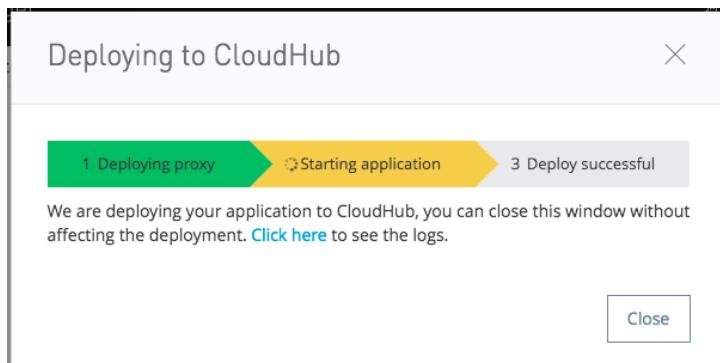
Implementation URI:

7. Click Save.
8. Scroll down if necessary and in the Deployment Configuration section, set the following values:
 - Runtime version: 4.2.0
 - Proxy application name: training4-american-api-{lastname}
9. Check Update application if exists.

Deployment Configuration ▾

Runtime version:	4.2.0	▼
Proxy application name: ⓘ	training4-american-api-maxmule	.clouduhub.io
<input checked="" type="checkbox"/> Update application if exists		Deploy

10. Click Deploy.
11. In the Deploying to CloudHub dialog box, click the Click here link to see the logs.



12. In the new browser tab that opens, watch the logs in Runtime Manager.

13. Wait until the proxy application starts.

Note: If it does not successfully deploy, read the logs to help figure out why the application did not deploy. If you had errors when deploying, troubleshoot them, fix them, and then redeploy.

The screenshot shows the Runtime Manager interface. On the left, there's a sidebar with 'Sandbox' selected, followed by 'Applications', 'Dashboard', 'Insight', and 'Logs'. The main area has a title 'training4-american-api-maxmule' and a 'Live Console' tab. Below it is a search bar and an 'Advanced' dropdown. The console output shows two log entries:

```
*****
21:51:06.747 05/08/2019 Deployment system SYSTEM
Worker(3.15.1.30): Your application has started successfully.

21:51:07.348 05/08/2019 Deployment system SYSTEM
Your application is started.
```

To the right, there's a 'Deployments' section with a 'Today' header and a single entry: '21:50 - Deployment' with a checkmark. A bell icon is also present.

14. In the left-side navigation, select Applications; you should see the proxy application.

15. Click the row for the proxy and review its information in the right section of the window.

The screenshot shows the Runtime Manager interface with 'Applications' selected in the sidebar. The main area displays a table of applications:

Name	Server	Status	File
training4-american-api-maxmu	CloudHub	Started	training4-american-api-maxmule-api.jar
training4-american-ws-maxmu	CloudHub	Started	training4-american-ws-v2.jar

On the right, a detailed view for 'training4-american-api-maxmu' is shown. It includes:

- Status: Started
- Server: CloudHub
- Last Updated: 2019-05-08 9:51:06PM
- App url: training4-american-api-maxmule.us-e2.cloudhub.io
- Runtime version: 4.2.0
- Worker size: 0.1 vCores
- Workers: 1
- Region: US East (Ohio)

Buttons at the bottom include 'Manage Application', 'Logs', and 'Insight'.

16. Close the browser tab.

View API details in API Manager

17. Return to the tab with API Manager and click the Close button in the Deploying to CloudHub dialog box.

18. Review the API proxy information at the top of the page.

The screenshot shows the API Manager interface. On the left, there's a sidebar with options like SANDBOX, Alerts, Contracts, Policies, SLA Tiers, and Settings. The Settings option is currently selected. The main content area is titled "American Flights API v1". It displays the following details:

- API Status: Active
- Asset Version: 1.0.1 (Latest)
- Type: RAML/OAS
- Implementation URL: <http://training4-american-ws-maxmule.us-e2.cloudhub.io/api>
- Autodiscovery: [Add consumer endpoint](#)
- ID: 15701606
- API ID: 15701606
- Label: [Add a label](#)
- Proxy:
 - Proxy Application: training4-american-api-maxmule
 - Proxy URL: <http://training4-american-api-maxmule.us-e2.cloudhub.io>

19. In the left-side navigation, click the API Administration link; you should now see your American Flights API listed.

20. Click in the row for the v1 version – but not on the v1 link.

21. Review the API version info that appears on the right side of the window; you should see there are no policies, SLA tiers, or client applications.

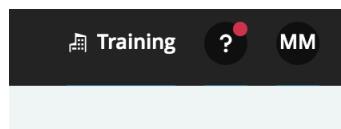
The screenshot shows the API Manager API Administration list. The left sidebar has options like SANDBOX, API Administration, Automated Policies, Client Applications, Custom Policies, and Analytics. The API Administration option is selected. The main list table shows the following data:

API Name	Version	Status	Client Applications	Creation Date
American Flights API	v1	Active	0	05-08-2019 21:48

To the right of the table, there's a summary section for "American Flights API v1" with links to "Manage CloudHub Proxy", "View API in Exchange", and "View Analytics Dashboard". Below that, tabs for "Applications", "Policies", and "SLA tiers" are shown, with a note that "There are no policies configured for this API version."

22. In the API list, click the v1 link for the API; you should be returned to the Settings page for the API.

23. Locate and review the links on the right side of the page.



Actions ▾

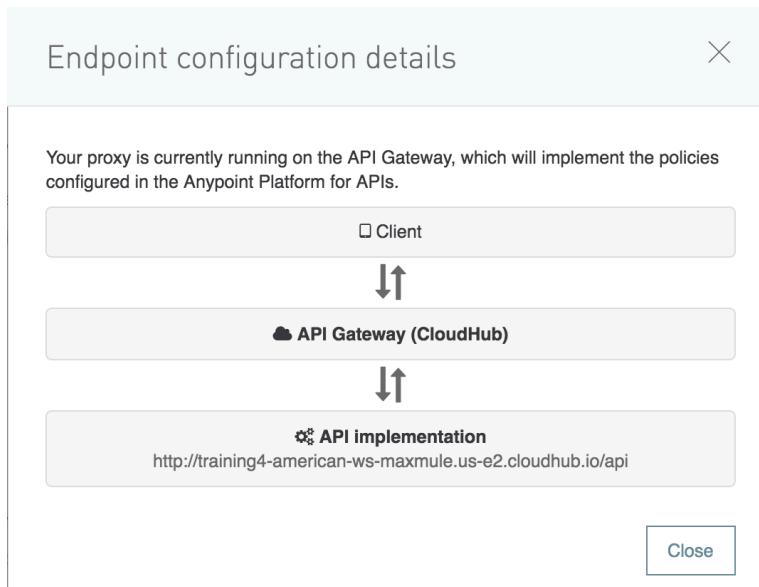
Manage CloudHub Proxy >

View API in Exchange >

View configuration details >

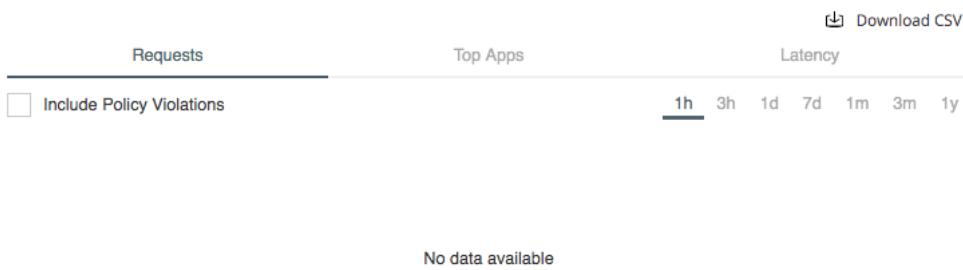
View Analytics Dashboard >

24. Click the View configuration details link.



25. In the Endpoint configuration details dialog box, click Close.

26. On the Settings page, look at the requests graph; you should not see any API requests yet.



View the new API proxy instance in Exchange

27. Return to the browser tab with Exchange.
28. Return to the home page for your American Flights API.
29. Locate the API instances now associated with asset version 1.0.1; you should see the Mocking Service instance and now the new proxy.

The screenshot shows a table titled "Asset versions for v1". The columns are "Version" and "Instances". There are two rows: one for version 1.0.1 containing "Mocking Service" and "Sandbox - v1:15701606", and one for version 1.0.0 containing a single row with three dots. At the bottom is a button labeled "+ Add new version".

Version	Instances
1.0.1	Mocking Service Sandbox - v1:15701606
1.0.0	⋮

+ Add new version

Note: You may need to refresh your browser page.

30. Click the GET method for the flights resource.
31. In the API console, click the drop-down arrow next to Mocking Service; you should NOT see the API proxy as a choice.

The screenshot shows a dropdown menu for "Mocking Service" which contains two entries: "Mocking Service" and "Mocking Service".

32. In the left-side navigation, select API instances; you should see that the new proxy instance does not have a URL.

The screenshot shows the "API instances" section of the Exchange interface. It lists two instances: "Mocking Service" (URL: https://anypoint.mulesoft.com/mockng/api/v1/sources/exchange/assets/c66beee7-52a3-4c79-a7be-5b56308f2850/american-flights-api/1.0.1/m) and "v1:15701606" (Environment: Sandbox). The "v1:15701606" instance has a "Private" visibility setting.

Instances	Environment	URL	Visibility
Mocking Service		https://anypoint.mulesoft.com/mockng/api/v1/sources/exchange/assets/c66beee7-52a3-4c79-a7be-5b56308f2850/american-flights-api/1.0.1/m	Public
v1:15701606	Sandbox		Private

+ Add new instance

Set a friendly label for the API instance in API Manager

33. Return to the browser tab with API Manager.
34. On the Settings page for your American Flights API, click the Add a label link.
35. Set the label to No policy and press Enter/Return.

API Instance ⓘ

ID: 15701606

Label: No policy 

Set a consumer endpoint for the proxy in API Manager

36. Locate the proxy URL.

Proxy

Proxy Application: training4-american-api-maxmule

Proxy URL: training4-american-api-maxmule.us-e2.cloudhub.io

37. Right-click it and copy the link address.

38. Click the Add consumer endpoint link.

American Flights API v1

API Status:  Active Asset Version: 1.0.1  Type: RAML/OAS

Implementation URL: <http://training4-american-ws-maxmule.us-e2.cloudhub.io/api>

 [Add consumer endpoint](#)

39. Paste the value of the proxy URL.

40. Press Enter/Return.

American Flights API v1

API Status:  Active Asset Version: 1.0.1  Type: RAML/OAS

Implementation URL: <http://training4-american-ws-maxmule.us-e2.cloudhub.io/api>

Consumer endpoint: <http://training4-american-api-maxmule.us-e2.cloudhub.io/> 

Make requests to the API proxy from Exchange

41. Return to the browser tab with Exchange.

42. Refresh the API instances page for your American Flights API; you should see the new label and the URL.

The screenshot shows the 'American Flights API' page in the Anypoint Platform. On the left, there's a sidebar with 'Assets list', 'American Flights API' (selected), 'Summary', 'Endpoints' (with '/flights' and '/{ID}'), 'Types', and 'API instances' (which is the active tab). The main area has a title 'American Flights API' with a edit icon, 'v1 Public'. Below it is a table titled 'API instances' with columns 'Instances', 'Environment', 'URL', and 'Visibility'. It lists two rows: 'Mocking Service' (Environment: Mocking Service, URL: https://anypoint.mulesoft.com/mocking/api/v1/sources/exchange/assets/c66beee7-52a3-4c79-a7be-5b56308f2850/american-flights-api/1.0.1/m, Visibility: Public) and 'No policy' (Environment: Sandbox, URL: http://training4-american-api-maxmule.us-e2.cloudhub.io/, Visibility: Private). A blue button '+ Add new instance' is at the bottom.

43. In the left-side navigation, select the name of the API to return to its main page.

44. Locate the API instances now associated asset version 1.0.1; you should see the new label for the API instance.

The screenshot shows the 'Asset versions for v1' page. It has a table with 'Version' and 'Instances' columns. The '1.0.1' row contains 'Mocking Service' (with a globe icon) and 'Sandbox - No policy' (with a lock icon). A blue button '+ Add new version' is at the bottom.

45. Click the GET method for the flights resource.

46. In the API console, click the drop-down arrow next to Mocking Service; you should now see your API proxy instance as a choice.

The screenshot shows a dropdown menu with three items: 'Mocking Service', 'Mocking Service' (selected and highlighted in grey), and 'Sandbox - No policy'.

47. Select the Sandbox - No policy instance.

48. Click the Send button; you should now see the real data from the database, which contains multiple flights.



A screenshot of a REST API response. At the top, it shows a green "200 OK" status box with "10679.48 ms" and a "Details" link. Below this is a toolbar with icons for copy, download, view code, and grid. The main content is a JSON array of 11 flight records. Each record includes fields like ID, code, price, departure date, origin, destination, empty seats, and plane type. The first two flights are detailed below:

```
[Array[11]
-0: {
  "ID": 1,
  "code": "rree0001",
  "price": 541,
  "departureDate": "2016-01-20T00:00:00",
  "origin": "MUA",
  "destination": "LAX",
  "emptySeats": 0,
  "plane": {
    "type": "Boeing 787",
    "totalSeats": 200
  }
},
-1: {
  "ID": 2,
  "code": "eefd0123",
  "price": 300,
```

49. Make several more calls to this endpoint.

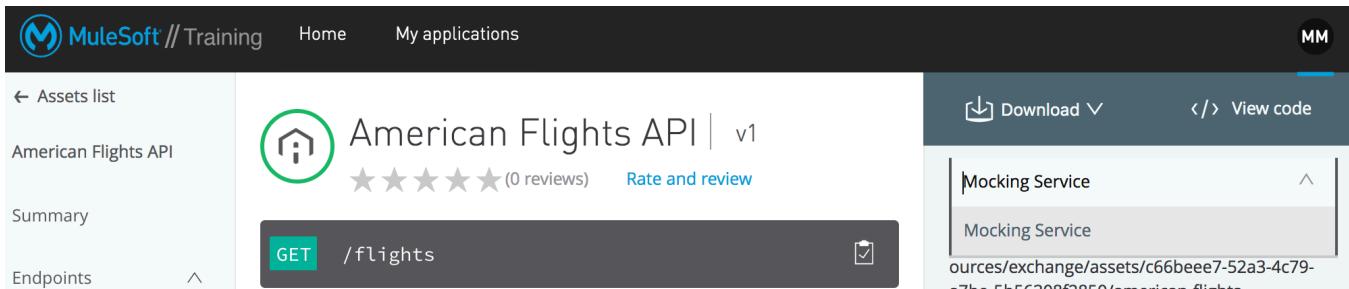
50. Make calls to different methods.

Make requests to the API proxy from the public portal

51. Return to the public portal in the private/incognito window.

52. Click the GET method for the flights resource.

53. In the API reference section, click the drop-down arrow next to Mocking Service; you should NOT see your API proxy instance as a choice.



A screenshot of the MuleSoft Anypoint Platform interface. At the top, there's a navigation bar with the MuleSoft logo, "Training", "Home", and "My applications". On the left, a sidebar shows "Assets list", "American Flights API", "Summary", and "Endpoints". The main area displays the "American Flights API | v1" page. It features a house icon, a 5-star rating (0 reviews), and "Rate and review" buttons. Below this is a "GET /flights" endpoint card with a checked checkbox. To the right, a "Mocking Service" dropdown menu is open, showing "Mocking Service" as the selected option. A URL in the background indicates the API is running on "Mocking Service".

Make an API instance visible in the public portal

54. Return to the browser with Exchange.
55. In the left-side navigation for American Flights API, select API instances.
56. Change the visibility of the No policy instance from private to public.

Instances	Environment	URL	Visibility
Mocking Service		https://anypoint.mulesoft.com/mockng/api/v1/sources/exchange/assets/c66beee7-52a3-4c79-a7be-5b56308f2850/american-flights-api/1.0.1/m	Public
No policy	Sandbox	http://training4-american-api-maxmule.us-e2.cloudhub.io/	Public

57. Return to the public portal in the private/incognito window.
58. Refresh the page.
59. In the API console, change the API instance from Mocking Service to Sandbox - No policy.

Download ▾ Request access </> View code

Sandbox - No policy

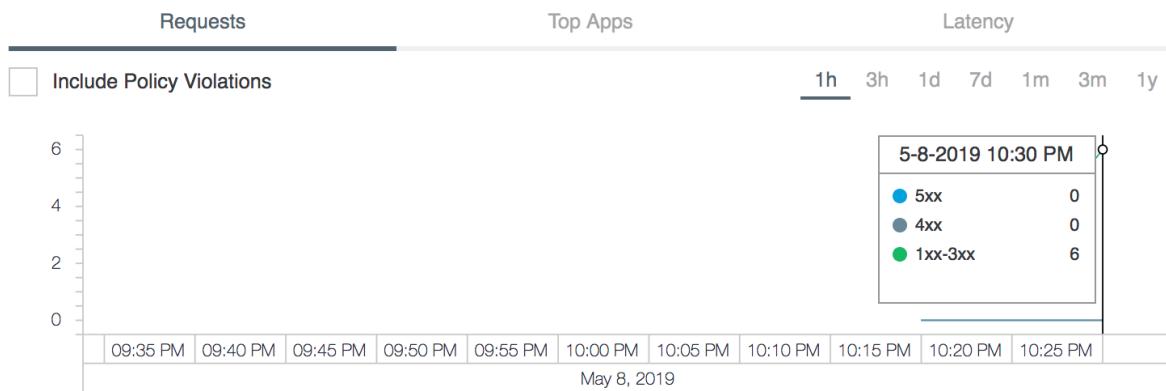
http://training4-american-api-maxmule.us-e2.cloudhub.io/flights

60. Click Send; you should a 200 response and flight data.

Look at the API request data

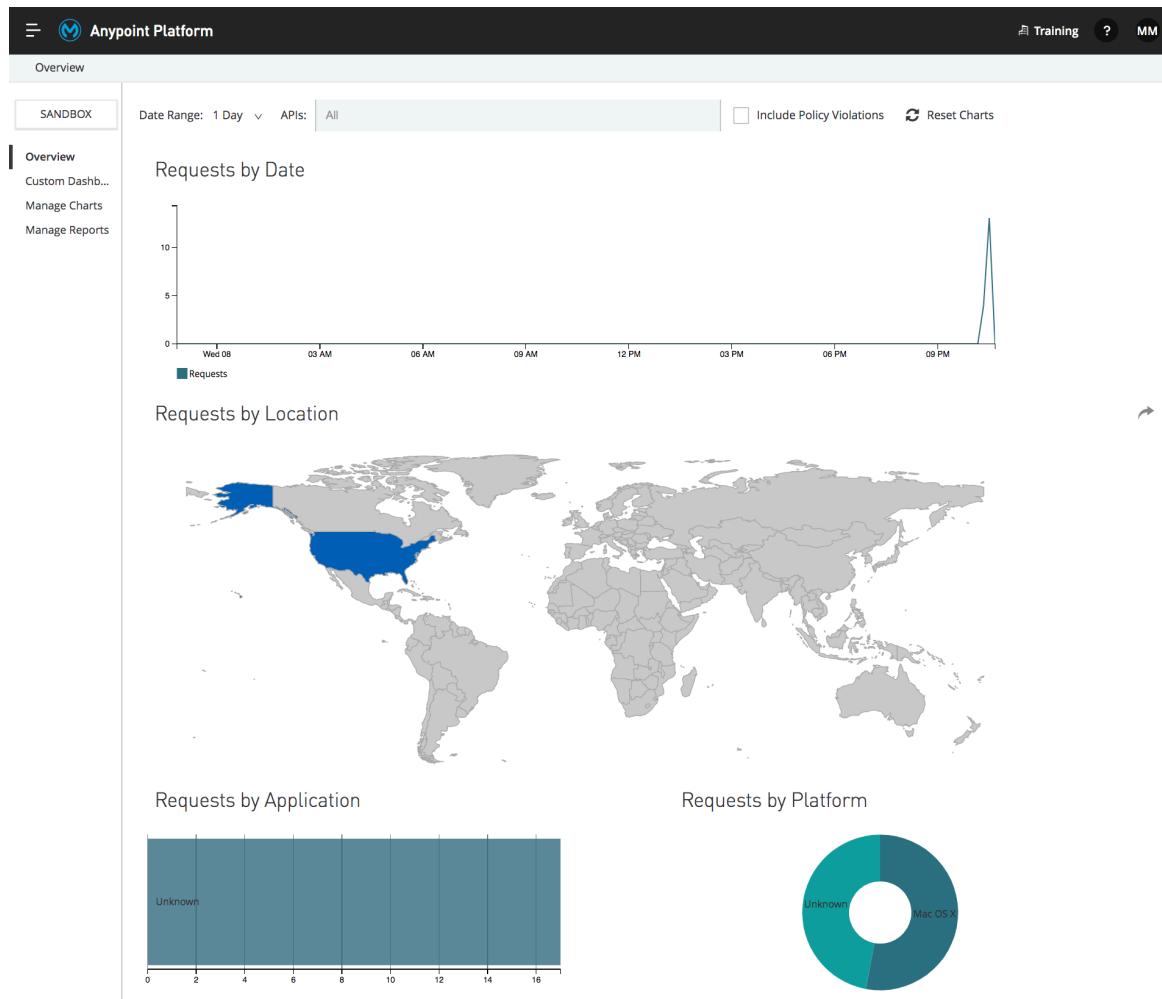
61. Return to the browser tab with API Manager.
62. Refresh the Settings page for you American Flights API.

63. Look at the Request chart again; you should now see data for some API calls.



64. Click the View Analytics Dashboard link located in the upper-right corner.

65. Review the data in the dashboard.



66. Close the browser tab.

Walkthrough 5-3: Restrict API access with policies and SLAs

In this walkthrough, you govern access to the API proxy. You will:

- Add and test a rate limiting policy.
- Add SLA tiers, one with manual approval required.
- Add and test a rate limiting SLA based policy.

American Flights API v1

API Status: Active Asset Version: 1.0.1 Latest Type: RAML/OAS

Implementation URL: <http://training4-american-ws-maxrule.us-e2.cloudhub.io/api>

Consumer endpoint: <http://training4-american-api-maxrule.us-e2.cloudhub.io/> Mule runtime version: 4.2.0

Automated Policies

There are no automated policies applied for the selected runtime version.

API level policies

Apply New Policy Edit policy order

Name	Category	Fulfils	Requires
Rate limiting - SLA based	Quality of service	SLA Rate Limiting, Client ID required	API Specification snippet

Order Method Resource URI

1 All API Methods All API Resources

View Detail Actions

Create a rate limiting policy

1. Return to the Settings page for your American Flights API in API Manager.
2. In the left-side navigation, select Policies.

American Flights API v1

API Status: Active Asset Version: 1.0.1 Latest Type: RAML/OAS

Implementation URL: <http://training4-american-ws-maxrule.us-e2.cloudhub.io/api>

Consumer endpoint: <http://training4-american-api-maxrule.us-e2.cloudhub.io/> Mule runtime version: 4.2.0

Automated Policies

There are no automated policies applied for the selected runtime version.

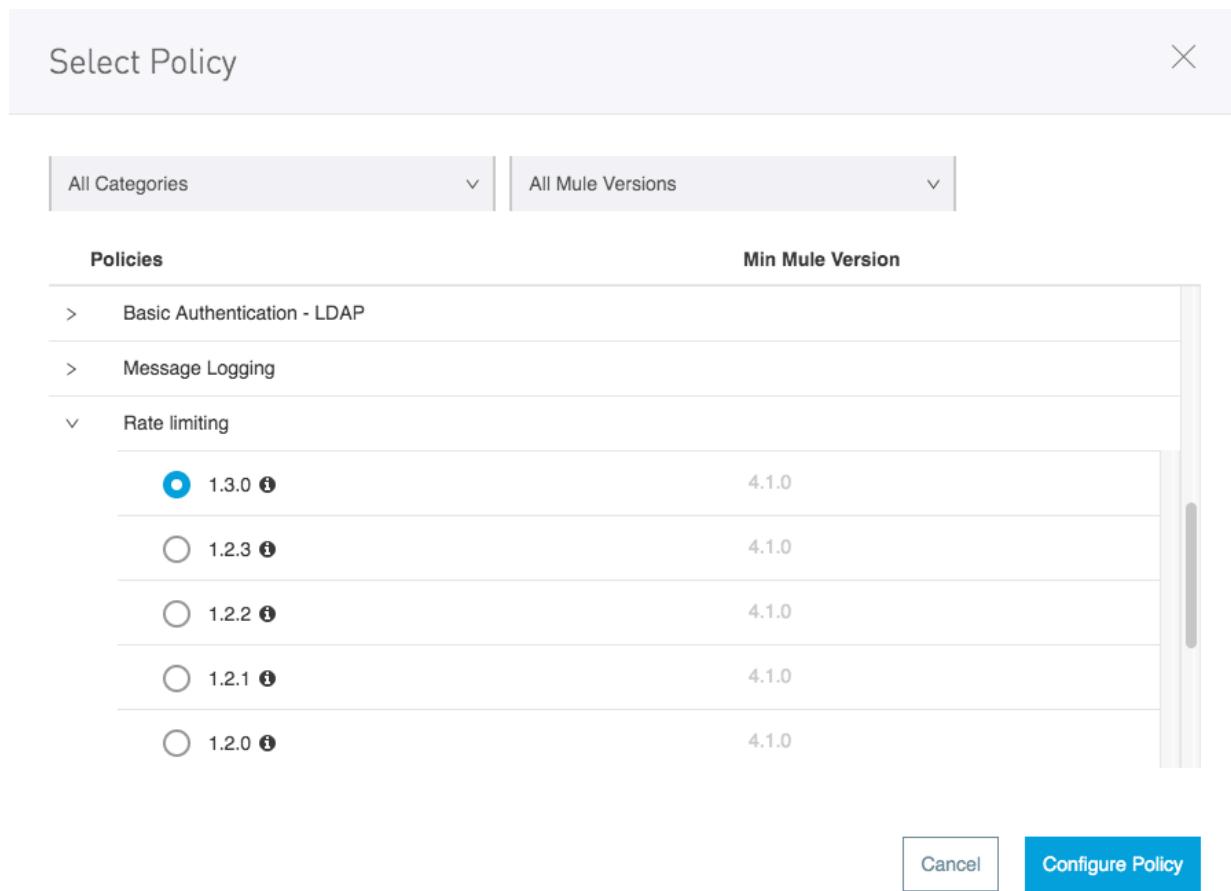
API level policies

Apply New Policy

There are no applied policies.

3. Click the Apply New Policy button.

4. In the Select Policy dialog box, expand Rate limiting and select the latest version for the Mule runtime version you are using.



5. Click Configure Policy.

6. On the Apply Rate limiting policy page, set the following values and click Apply:

- # of Reqs: 3
- Time Period: 1
- Time Unit: Minute
- Method & Resource conditions: Apply configurations to all API methods & resources

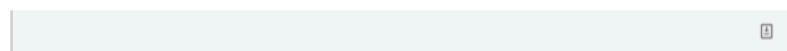
7. Select Expose Headers.

Apply Rate limiting policy

Specifies the maximum value for the number of messages processed per time period, and rejects any messages beyond the maximum. Applies rate limiting to all API calls, regardless of the source.

Identifier

For each identifier value, the set of Limits defined in the policy will be enforced independently. I.e.: # [attributes.queryParams['identifier']]



Limits

Pairs of maximum quota allowed and time window.

# of Reqs *	Time Period *	Time Unit *
3	1	Minute

[+ Add Limit](#)

Clusterizable

When using a clustered runtime with this flag enabled, configuration will be shared among all nodes.

Expose Headers

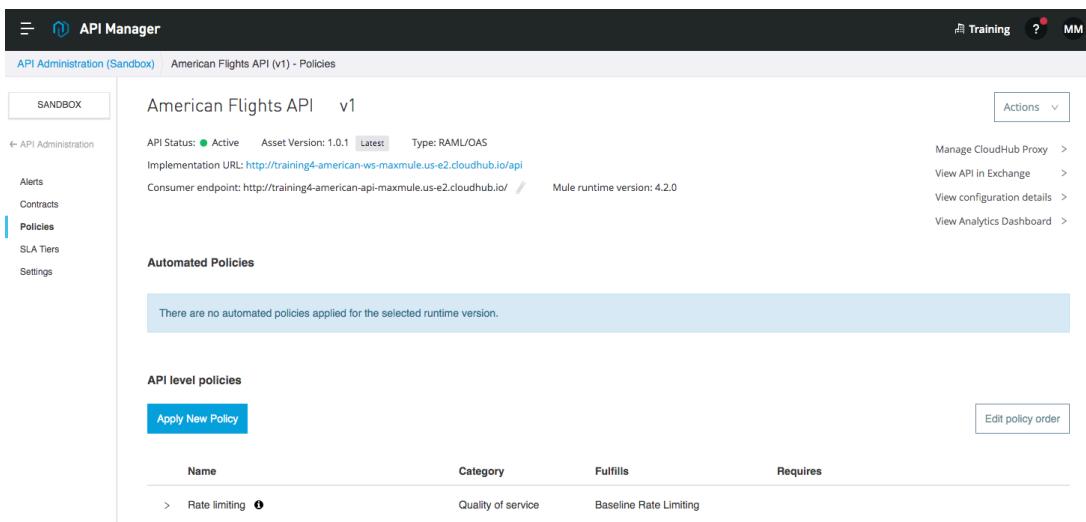
Defines if headers should be exposed in the response to the client. These headers are: x-ratelimit-remaining, x-ratelimit-limit and x-ratelimit-reset.

Method & Resource conditions

- Apply configurations to all API methods & resources
- Apply configurations to specific methods & resources

[Cancel](#) [Apply](#)

8. Click Apply; you should see the policy listed for your API.



Name	Category	Fulfils	Requires
> Rate limiting <small>?</small>	Quality of service	Baseline Rate Limiting	

9. In the left-side navigation, select Settings.
10. Change the API instance label to Rate limiting policy.

API Instance ⓘ

ID: 15701606

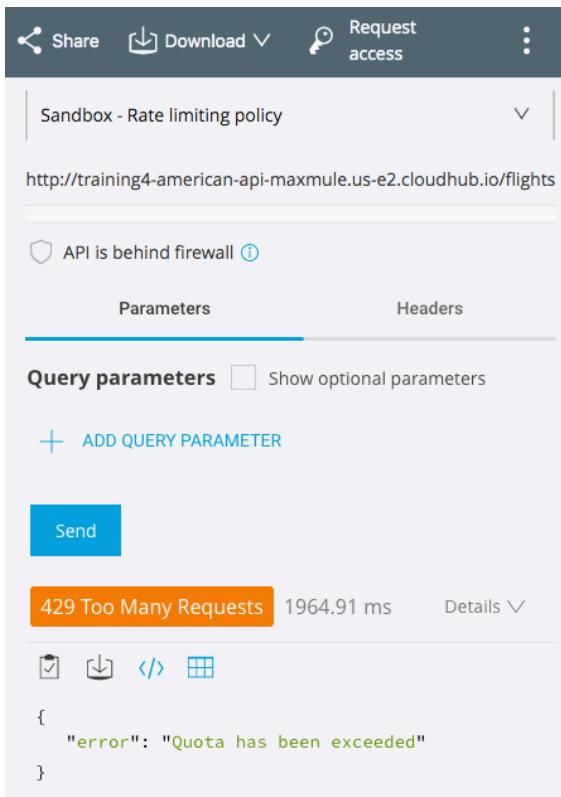
Label: Rate limiting policy 

Test the new rate limiting policy

11. Return to the browser tab with your American Flights API in Exchange.
12. Return to the page with the API console for the flights:/GET resource.
13. Select the Sandbox – Rate limiting policy API instance.

Note: You may need to refresh the page to see the new label for the API instance.

14. Press Send until you get a 429 Too Many Requests response.



The screenshot shows the API console interface. At the top, there are buttons for Share, Download, Request access, and a menu icon. Below that, the API instance is set to "Sandbox - Rate limiting policy". The URL is listed as "http://training4-american-api-maxmule.us-e2.cloudhub.io/flights". A note says "API is behind firewall ⓘ". There are tabs for "Parameters" and "Headers", with "Parameters" being active. Under "Query parameters", there is a checkbox for "Show optional parameters" and a button to "ADD QUERY PARAMETER". A large blue "Send" button is at the bottom. Below the button, the response status is shown in an orange box: "429 Too Many Requests" with a timestamp "1964.91 ms" and a "Details" link. The response body is displayed as JSON: "{ \"error\": \"Quota has been exceeded\" }".

Create SLA tiers

15. Return to the browser tab with your American Flights API in API Manager.
16. In the left-side navigation, select SLA Tiers.

17. Click the Add SLA tier button.

The screenshot shows the API Manager interface with the title "American Flights API v1". On the left, there's a sidebar with options like "Sandbox", "API Administration", "Alerts", "Contracts", "Policies", "SLA Tiers" (which is selected and highlighted in blue), and "Settings". The main content area displays the API status as "Active", asset version "1.0.1", type "RAML/OAS", implementation URL "http://training4-american-ws-maxmule.us-e2.cloudhub.io/api", consumer endpoint "http://training4-american-api-maxmule.us-e2.cloudhub.io/", and Mule runtime version "4.2.0". A search bar and an "Actions" dropdown menu are also visible.

18. In the Add SLA tier dialog box, set the following values:

- Name: Free
- Approval: Automatic
- # of Reqs: 1
- Time Period: 1
- Time Unit: Minute

The dialog box has fields for "Name" (set to "Free"), "Description" (empty), "Approval" (set to "Automatic"), and a "Limits" section. The "Limits" section contains fields for "# of Reqs" (1), "Time Period" (1), and "Time Unit" (Minute). There are also buttons for "Cancel" and "Add".

19. Click the Add button.

20. Create a second SLA tier with the following values:

- Name: Silver
- Approval: Manual
- # of Reqs: 1
- Time Period: 1
- Time Unit: Second

The screenshot shows a table titled 'SLA tiers' with columns: Name, Limits, Applications, Status, and Approval. There are two rows: 'Free' (Limits 1, Applications 0, Active, Auto) and 'Silver' (Limits 1, Applications 0, Active, Manual). Each row has 'Edit' and 'Delete' buttons.

Name	Limits	Applications	Status	Approval	
Free	1	0	Active	Auto	<button>Edit</button> <button>Delete</button>
Silver	1	0	Active	Manual	<button>Edit</button> <button>Delete</button>

Change the policy to rate limiting – SLA based

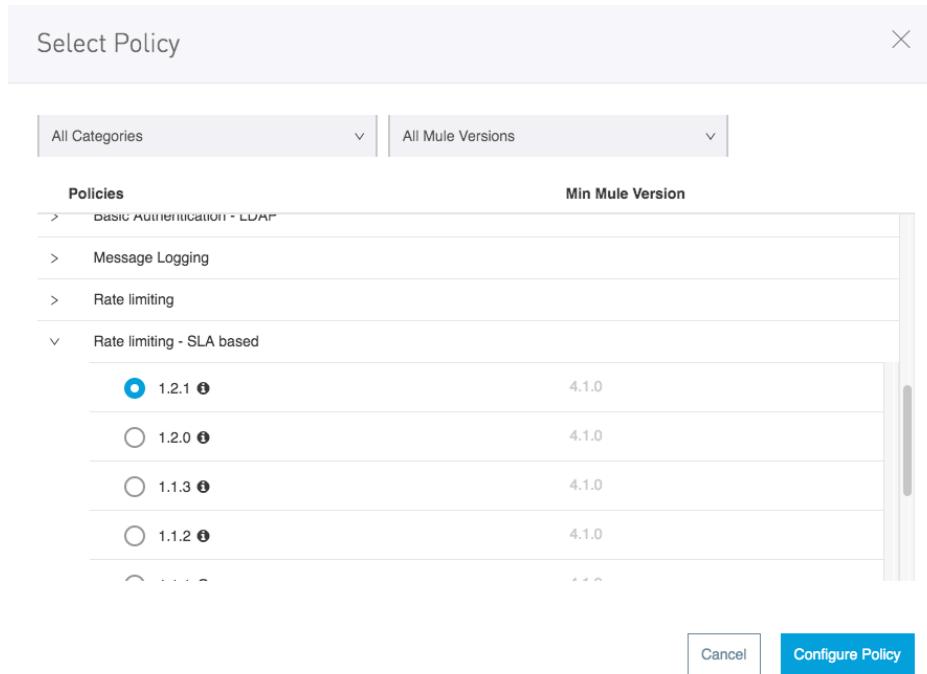
21. In the left-side navigation, select Policies.
22. Expand the Rate limiting policy.
23. Click the Actions button and select Remove.

The screenshot shows a table with columns: Name, Category, Fulfils, and Requires. A single row is expanded, showing 'Rate limiting' under 'Category' and 'Baseline Rate Limiting' under 'Fulfils'. Below this, a detailed view shows 'Order' (1), 'Method' (All API Methods), 'Resource URI' (All API Resources), and an 'Actions' dropdown menu with options: View Detail, Edit, Disable, and Remove.

Name	Category	Fulfils	Requires
Rate limiting ⓘ	Quality of service	Baseline Rate Limiting	
Order	Method	Resource URI	
1	All API Methods	All API Resources	<button>View Detail</button> <button>Actions</button>

24. In the Remove policy dialog box, click Remove.
25. Click the Apply New Policy button.
26. In the Select Policy dialog box, expand Rate limiting - SLA based and select the latest version for the Mule runtime version you are using.

27. Click Configure Policy.



28. On the Apply Rate limiting – SLA based policy page, look at the expressions and see that a client ID and secret need to be sent with API requests as headers.

29. Select Expose Headers.

The configuration page has a header with tabs: 'API Administration (Sandbox)', 'American Flights API (v1) - Policies', and 'Apply Rate limiting - SLA based policy'. A sidebar on the left shows 'Sandbox' and 'Policies'. The main content area is titled 'Apply Rate limiting - SLA based policy'. It contains the following fields:

- Client ID Expression ***: Mule Expression to extract Client ID from API requests. Value: #[attributes.headers['client_id']]
- Client Secret Expression**: Mule Expression to extract Client Secret from API requests. Value: #[attributes.headers['client_secret']]
- Clusterizable**: A checked checkbox. Description: When using a clustered runtime with this flag enabled, configuration will be shared among all nodes.
- Expose Headers**: A checked checkbox. Description: Defines if headers should be exposed in the response to the client. These headers are: x-ratelimit-remaining, x-ratelimit-limit and x-ratelimit-reset.

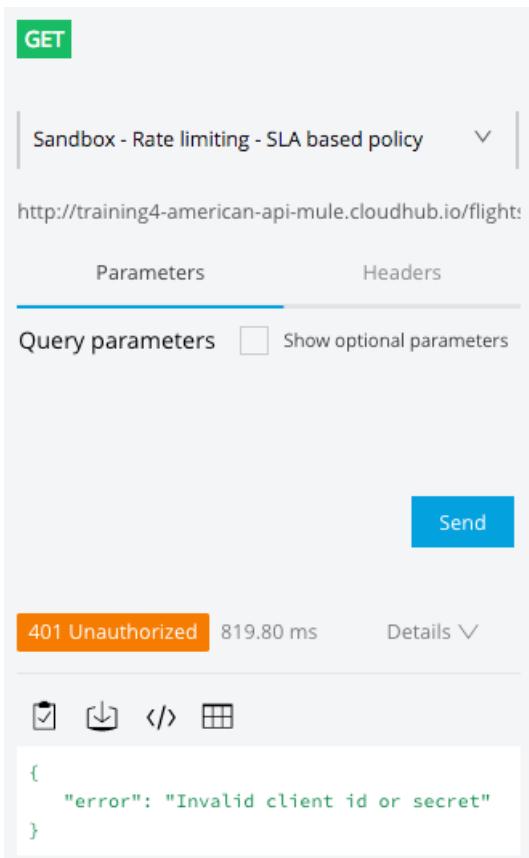
At the bottom are 'Method & Resource conditions' (radio buttons: 'Apply configurations to all API methods & resources' (selected) and 'Apply configurations to specific methods & resources') and 'Cancel' and 'Apply' buttons.

30. Click Apply.
31. In the left-side navigation, select Settings.
32. Change the API instance label to Rate limiting – SLA based policy.

API Instance ⓘ
ID: 15701606
Label: Rate limiting – SLA based policy 

Test the rate limiting – SLA based policy in Exchange

33. Return to the browser tab with your API in Exchange.
34. Refresh the page and select to make a call to the Sandbox – Rate limiting – SLA based policy.
35. Click Send; you should get a 401 Unauthorized response with the error Invalid client id or secret.



The screenshot shows the Anypoint Platform interface for testing APIs. A modal window is open for a GET request to the endpoint `http://training4-american-api-mule.cloudhub.io/flight`. The request is associated with the 'Sandbox - Rate limiting - SLA based policy' instance. The 'Parameters' tab is selected, and the 'Query parameters' section contains the value 'Show optional parameters'. A 'Send' button is visible at the bottom right of the modal. Below the modal, the status bar indicates a 401 Unauthorized response with a duration of 819.80 ms. The detailed error message is displayed in a code block:

```
401 Unauthorized 819.80 ms Details ▾  
  
✖ ↴ </> ⌂  
{  
  "error": "Invalid client id or secret"  
}
```

Walkthrough 5-4: Request and grant access to a managed API

In this walkthrough, clients request access to an API proxy and administrators grant access. You will:

- Request application access to SLA tiers from private and public API portals.
- Approve application requests to SLA tiers in API Manager.

The screenshot shows the API Manager interface with the 'Contracts' tab selected. The main area displays the 'American Flights API v1' details. Two access requests are listed:

Application	Current SLA tier	Requested SLA tier	Status	Action Buttons
> Training external app	N/A	Silver	Pending	[Approve] [Reject] [Delete]
> Training internal app	Free	N/A	Approved	[Revoke] [Delete]

Request access to the API as an internal consumer

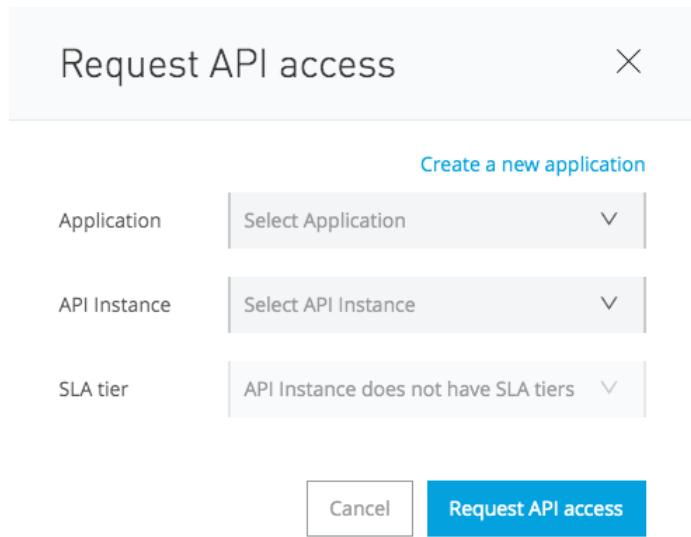
1. Return to the browser tab with Anypoint Exchange.
2. In the left-side navigation, select the name of the API to return to its home page.
3. Click the more options button in the upper-right corner and select Request access.

The screenshot shows the Anypoint Exchange interface with the 'American Flights API' selected in the left sidebar. The main area displays the API summary and supported operations. In the top right, the 'Request access' button is highlighted.

Note: Other internal users that you shared the API with that do not have Edit permissions will see a different menu.

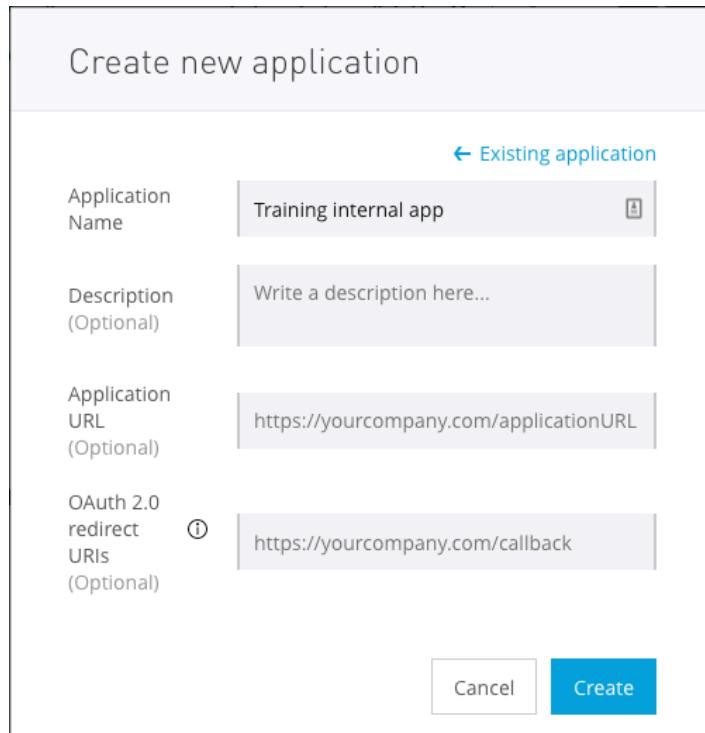
The screenshot shows the Anypoint Exchange interface with the 'American Flights API' selected in the left sidebar. The main area displays the API summary and supported operations. The 'Request access' button is visible in the top right corner of the header bar.

4. In the Request API access dialog box, click the Create a new application link.



The dialog box has a title bar "Request API access" with a close button. Below it is a section titled "Create a new application". It contains three dropdown menus: "Application" (set to "Select Application"), "API Instance" (set to "Select API Instance"), and "SLA tier" (set to "API Instance does not have SLA tiers"). At the bottom are two buttons: "Cancel" and a blue "Request API access" button.

5. In the Create new application dialog box, set the name to Training internal app and click Create.



The dialog box has a title bar "Create new application" with a back arrow "Existing application". It contains four input fields: "Application Name" (set to "Training internal app"), "Description (Optional)" (set to "Write a description here..."), "Application URL (Optional)" (set to "https://yourcompany.com/applicationURL"), and "OAuth 2.0 redirect URIs (Optional)" (set to "https://yourcompany.com/callback"). At the bottom are two buttons: "Cancel" and a blue "Create" button.

6. In the Request API access dialog box, set the API instance to Rate limiting - SLA based policy.

7. Set the SLA tier to Free.

Request API access X

[Create a new application](#)

Application	Training internal app	▼
API Instance	Rate limiting - SLA based policy	▼
SLA tier	Free	▼

# of Reqs	Time period	Time Unit
1	1	Minute

Cancel Request API access

8. Click Request API access.
9. In the Request API access dialog box, view the assigned values for the client ID and client secret.

Request API access X

✓ **API access has been successful!**

Client ID	8595a1a4f60340e1ad1a3f00ebf5837f
Client secret	437A492CCC894DF280c709d0e5f05A9f

Application details have been opened in a new tab.

Close

10. Click Close.

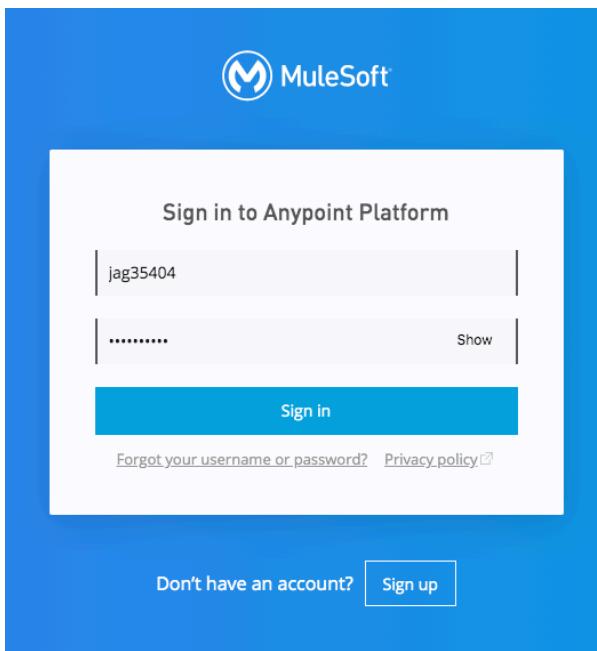
Request access to the API as an external consumer

11. Return to the public portal in the private/incognito window.
12. Refresh the page for the American Flights API; you should now see a Request access button.

The screenshot shows the MuleSoft Training portal. In the top navigation bar, there is a 'Login' link. On the left, a sidebar menu includes 'Assets list', 'American Flights API' (which is highlighted in grey), and 'Summary'. The main content area displays the 'American Flights API | v1' page. It features a green house icon, a 5-star rating with '(0 reviews)', and a brief description: 'The American Flights API is a system API for operations on the **american** table in the *training* database.' At the bottom right of this section are 'Download' and 'Request access' buttons. The 'Request access' button is blue with white text.

13. Click the Request access button; you should get a page to sign in or create an Anypoint Platform account.
14. Enter your existing credentials and click Sign in.

Note: Instead of creating an external user, you will just use your existing account.



15. Back in the public portal, click the Request access button again.
16. In the Request API access dialog box, click the Create a new application button

17. In the Create new application dialog box, set the name to Training external app and click Create.

Create new application

← Existing application

Application Name	Training external app
Description (Optional)	Write a description here...
Application URL (Optional)	https://yourcompany.com/applicationURL
OAuth 2.0 redirect URIs (Optional)	https://yourcompany.com/callback

Cancel Create

18. In the Request API access dialog box, set the API instance to Rate limiting - SLA based policy.

19. Set the SLA tier to Silver.

Request API access ×

Create a new application

Application	Training external app
API Instance	Rate limiting – SLA based policy
SLA tier	Silver

# of Reqs	Time period	Time Unit
1	1	Second

Cancel Request API access

20. Click Request API access.

21. In the Request API access dialog box, click Close.

22. In the portal main menu bar, right-click My applications and select to open it in a new tab; you should see the two applications you created.

The screenshot shows the MuleSoft // Training portal interface. At the top, there is a navigation bar with the MuleSoft logo, the text "MuleSoft // Training", and links for "Home" and "My applications". The "My applications" link is underlined, indicating it is the active page. Below the navigation bar, the page title "My applications" is displayed. A search bar with a magnifying glass icon and the placeholder "Search" is present. The main content area lists two applications in a table:

Name	Description
Training internal app	
Training external app	

23. Click the link for Training external app; you should see what APIs the application has access to, values for the client ID and secret to access them, and request data.

The screenshot shows the details page for the "Training external app". The top navigation bar is identical to the previous screenshot. The main content area shows the application name "Training external app" and three action buttons: "Edit" (highlighted in blue), "Reset client secret", and "Delete". On the left, there is a sidebar with a tree view:

- Show All (1)
- ✓ Sandbox
 - American Flights API - Rate limitin... v1

On the right, detailed application settings are shown in a box:

Application Description:	Client ID: 3c376d605d7f4a5b849e435729f6fe13
Application URL:	Client Secret: Show
Redirect URIs:	Grant Types: -

24. Leave this page open in a browser so you can return to it and copy these values.

Grant an application access

25. Return to the browser window and tab with the Settings page for American Flights API v1 in API Manager.

26. In the left-side navigation, select Contracts; you should see the two applications that requests access to the API.

Application	Current SLA tier	Requested SLA tier	Status
> Training external app	N/A	Silver	Pending
> Training internal app	Free	N/A	Approved

27. Click the Approve button for the application requesting Silver tier access.

28. Expand the Training external app row and review its information.

29. Copy the value of the Client ID.

Application	Current SLA tier	Requested SLA tier	Status
> Training external app	Silver	N/A	Approved
Owners	Max Mule [REDACTED]@mulesoft.com	Submitted	6 minutes ago
Client ID	3c376d605d7f4a5b849e435729f6fc13	Approved	a few seconds ago
URL	None	Rejected	-
Redirect URIs	None	Revoked	-
> Training internal app	Free	N/A	Approved

Add authorization headers to test the rate limiting – SLA based policy from an API portal

30. Return to the browser window and tab with the API console in the public portal.

31. Try again to make a call to the Sandbox – Rate limiting – SLA based policy; you should still get a 401 Unauthorized response.

32. Select the Headers tab.

33. Click Add header.

34. Set the header name to client_id.

35. Set the value of client_id to the value you copied.

The screenshot shows the MuleSoft API console interface. At the top, it says "Sandbox - Rate limiting - SLA based policy". Below that is the URL "http://training4-american-api-maxmule.us-e2.cloudhub.io/flights". A warning message "API is behind firewall" is displayed with an info icon. There are two tabs: "Parameters" and "Headers", with "Headers" being the active tab. Under "Headers", there is one entry: "client_id" with a value of "3c376d605d7f4a5b849e435729f6fe13". Below the header list is a blue "Send" button.

36. Return to the browser tab with My applications in the public portal.

37. Click the Show link next to Client Secret then copy its value.

38. Return to the browser window and tab with the API console in the public portal.

39. Add another header and set the name to client_secret.

40. Set the client_secret header to the value you copied.

The screenshot shows the 'Headers' tab of a configuration dialog. It contains two entries:

- client_id**:
Header name: client_id
Parameter value: 3c376d605d7f4a5b849e435729f6fe13
- client_secret**:
Header name: client_secret
Parameter value: DF04B05003f748D5b082e86C624D9DDe

At the bottom right is a blue '+ Add header' button.

41. In the course snippets.txt file, record these client_id and client_secret values in the section reserved for this module.

42. Click Send; you should now get a 200 response with flight results.

The screenshot shows the API response details for a successful 200 OK request. The response time is 968.80 ms. The response body is an array of flight results:

```
[{"ID": 1, "code": "rree0001", "price": 541, "departureDate": "2016-01-20T00:00:00", "origin": "MUA", "destination": "LAX", "emptySeats": 0, "plane": {"type": "Boeing 787", "totalSeats": 200}}, {"ID": 2, "code": "eefd0123", "price": 300, "departureDate": "2016-01-20T00:00:00"}]
```

Walkthrough 5-5: Add client ID enforcement to an API specification

In this walkthrough, you add client ID enforcement to the API specification. You will:

- Modify an API specification to require client id and client secret headers with requests.
- Update a managed API to use a new version of an API specification.
- Call a governed API with client credentials from API portals.

Note: If you do not complete this exercise for Fundamentals, the REST connector that is created for the API and that you use later in the course will not have client_id authentication.

```
1  #%RAML 1.0
2  version: v1
3  title: American Flights API
4
5  types:
6    | AmericanFlight: !include
7    | exchange_modules/68ef9520-24e9-4cf2-b2f5-620025690
8
9  traits:
10   | client-id-required:
11     | headers:
12       | client_id:
13         | type: string
14       | client_secret:
15         | type: string
16     | responses:
17       | 401:
18         | description: Unauthorized, The client_id o
19       | 429:
20         | description: The client used all of it's r
21       | 500:
22         | description: An error occurred, see the spe
```

The screenshot shows the 'Headers' tab of an API specification editor. It lists two required headers: 'client_id*' and 'client_secret*'. Each header entry has a small trash can icon to its right. Below the list is a blue '+ Add header' button. At the bottom of the screen, there are two buttons: 'Send' and 'Fill in required parameters'.

Copy the traits required to add authentication to the API specification

1. Return to the browser tab with the Settings page for American Flights API v1 in API Manager.
2. In the left-side navigation, select Policies.
3. Click the API Specification snippet link for the Rate limiting – SLA based policy.

The screenshot shows the 'API level policies' section of the API Manager. It displays a single policy named 'Rate limiting - SLA based'. Below the policy name is a small info icon. To the right of the policy are two buttons: 'Edit policy order' and 'Apply New Policy'. A table below the policy lists its details: Name, Category, Fulfils, and Requires. The 'Requires' column indicates that the policy fulfills 'SLA Rate Limiting, Client ID required' and is associated with the 'API Specification snippet'.

Name	Category	Fulfils	Requires
> Rate limiting - SLA based ⓘ	Quality of service	SLA Rate Limiting, Client ID required	API Specification snippet

- In the API Specification snippet for Rate limiting - SLA based dialog box, select RAML 1.0.
- Copy the value for the traits.

RAML 0.8

RAML 1.0

OAS 2.0

Client ID based policies by default expect to obtain the client ID and secret as headers. To enforce this in the API definition a trait can be defined in RAML as shown below.

```

traits:
  client-id-required:
    headers:
      client_id:
        type: string
      client_secret:
        type: string
    responses:
      401:
        description: Unauthorized, The client_id or client_secret are not valid or the client does not have access.
      429:
        description: The client used all of it's request quota for the current period.
      500:
        description: An error occurred, see the specific message (Only if it is a WSDL endpoint).
      503:
        description: Contracts Information Unreachable.

```

- Click Close.

Add authentication headers to the API specification

- Return to the browser tab with your API in Design Center.
- Go to a new line after the types declaration and paste the traits code you copied.

```

1  #RAML 1.0
2  version: v1
3  title: American Flights API
4
5  types:
6  | AmericanFlight: !include
7  | exchange_modules/68ef9520-24e9-4cf2-b2f5-620025690
8
9  traits:
10 | client-id-required:
11 |   headers:
12 |     client_id:
13 |       type: string
14 |     client_secret:
15 |       type: string
16 |   responses:
17 |     401:
18 |       description: Unauthorized, The client_id o
19 |     429:
20 |       description: The client used all of it's r
21 |     500:
22 |       description: An error occurred, see the spe
23 |     503:
24 |       description: Contracts Information Unreach
25 /flights:

```

9. Go to a new line after the /flights resource declaration and indent.

```
22      ||| 503:  
23      |||   description: Contracts Information Unreachable.  
24  
25  /flights:  
26  |||  
27  |||   get:  
28  |||     queryParameters:  
29  |||       destination:  
30  |||         required: false
```

10. Add a nested is node with an empty array.

```
is: []
```

```
22      ||| 503:  
23      |||   description: Contracts Information Unreachable.  
24  
25  /flights:  
26  |||   is: []  
27  |||   get:  
28  |||     queryParameters:  
29  |||       destination:
```

11. Make sure the cursor is inside the array brackets and add the client-id-required trait name as an array element.

```
25  ↘ /flights:  
26  |   is: [client-id-required]  
27  ↘   get:  
28  ↘     queryParameters:
```

12. Repeat this process so the trait is applied to all methods of the {ID} resource as well.

```
56  ↗ /{ID}:  
57  |   is: [client-id-required]  
58  |   get:  
59  |     responses:
```

Test the API in the API console in Design Center

13. In the API console, turn on the mocking service.

14. Select one of the resources and click Try it.

15. Select the Headers tab; you should now see fields to enter client_id and client_secret.

Parameters Headers ⓘ

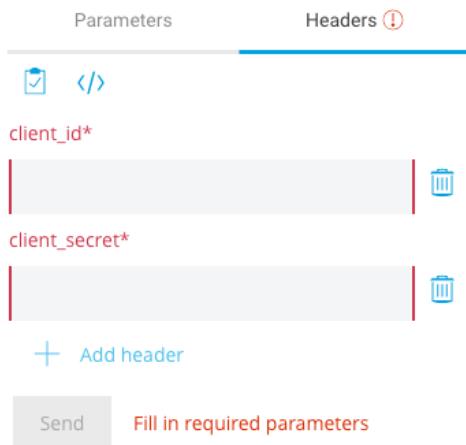
</>

client_id*

client_secret*

+ Add header

Send Fill in required parameters



16. Look at the Send button; you should see that it's disabled.

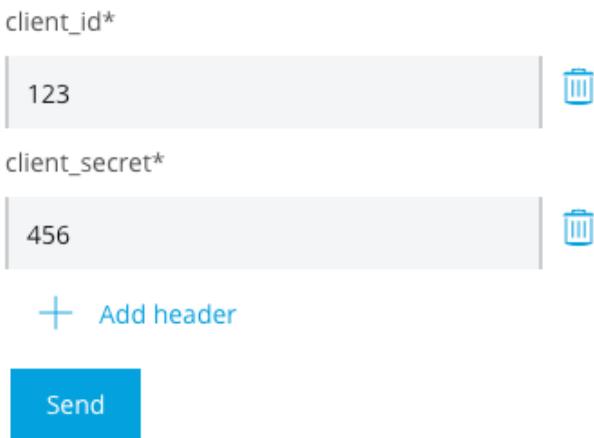
17. Enter any values for the client_id and client_secret and click Send; you should get a 200 response with the example results.

client_id*

client_secret*

+ Add header

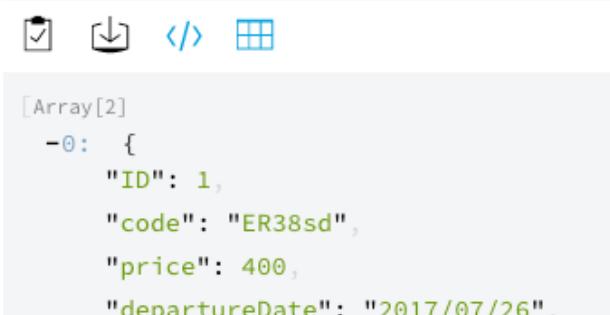
Send



200 OK 608.90 ms Details ▾

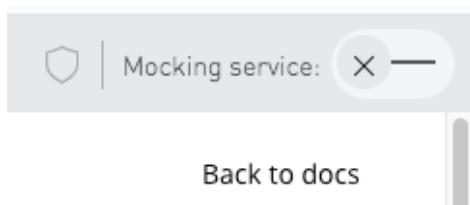
</>

```
[Array[2]
-0: {
  "ID": 1,
  "code": "ER38sd",
  "price": 400,
  "departureDate": "2017/07/26"
}]
```



Publish the new version of the API to Exchange

18. Turn off the mocking service.



19. Click the Publish button.

20. Click the Publish to Exchange button.

21. In the Publish API specification to Exchange dialog box, note the asset version and click Publish.

The dialog box has a title bar "Publish API specification to Exchange". Inside, there are two main sections: "Name (required)" containing "American Flights API" and "Asset version ⓘ (required)" containing "1.0.2". Below these, a note says "Current version: 1.0.1". The second section contains "Main file (required)" with "american-flights-api.raml" and "API version ⓘ (required)" with "v1". A green link "Valid Specification" is visible. At the bottom are buttons for "Show advanced >" and "Cancel" (disabled), and a prominent blue "Publish" button.

22. After the API is published, click Done in the Publish API specification to Exchange dialog box.

Update the managed API instance to use the new version of the API specification

23. Return to browser tab with American Flights API v1 in API Manager.

24. Refresh the page then locate the asset version displayed at the top of the page; you should see 1.0.1 and a new Update drop-down menu next to it.

American Flights API v1

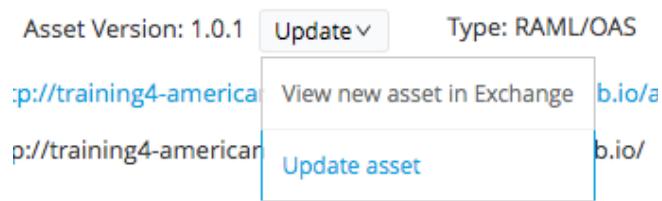
API Status: ● Active Asset Version: 1.0.1 Update Type: RAML/OAS

Implementation URL: <http://training4-american-ws-maxmule.us-e2.cloudhub.io/api>

Consumer endpoint: <http://training4-american-api-maxmule.us-e2.cloudhub.io/> Edit

Mule runtime version: 4.2.0

25. Select Update asset in the Update drop-down menu.



26. In the Update asset version dialog box, select 1.0.2 in the drop-down menu.



27. Click Change; you should see the 1.0.2 asset version displayed at the top of the page with the Latest label next to it.

American Flights API v1

API Status: Active Asset Version: 1.0.2 Latest Type: RAML/OAS
Implementation URL: <http://training4-american-ws-maxmule.us-e2.cloudhub.io/api>
Consumer endpoint: <http://training4-american-api-maxmule.us-e2.cloudhub.io/> edit Mule runtime version: 4.2.0

Redeploy a new proxy

28. In the left-side navigation, select Settings.

29. Scroll down to the Deployment Configuration settings

Deployment Configuration v

Runtime version:	4.2.0	▼
Proxy application name: (i)	training4-american-api-maxmule	.cloudhub.io
		Redeploy

30. Click Redeploy.
31. In the Deploying to CloudHub dialog box, click the Click here link to see the logs.
32. Watch the logs and wait until the proxy application is redeployed.

The screenshot shows the MuleSoft Runtime Manager interface. On the left, there's a sidebar with options like SANDBOX, Applications, Dashboard, Insight, Logs (which is selected), Application Data, Queues, and Schedules. The main area displays logs for an application named 'training4-american-api-maxmule'. The logs show the following entries:

```

*****
* Policy "gatekeeper-15701606-proxy" shut down normally on: 5/10/19 *
* 4:21 AM
* Up for: 0 days, 0 hours, 0 mins, 13.256 sec
*****
00:23:00.534 05/10/2019 Deployment system SYSTEM
Application was updated successfully with zero downtime. The new version of your application has been launched and the old version has been stopped.

00:23:00.560 05/10/2019 Deployment system SYSTEM
Your application is started.

```

To the right, there's a 'Deployments' panel showing a deployment log from 'Today' at '00:19 - Deployment' and another from 'Wednesday 05-08-2019' at '21:50 - Deployment'.

33. Close the browser tab.
34. Return to the browser tab with API Manager and click Close in the Deploying to CloudHub dialog box.

Test the rate limiting – SLA based policy in the API console in Exchange

35. Return to the browser tab with Exchange.
36. Return to the home page for the API (and refresh if necessary); you should see the new asset version listed.

The screenshot shows the API console in Exchange displaying 'Asset versions for v1'. A table lists the versions and their details:

Version	Instances
1.0.2	Mocking Service Sandbox - Rate limiting – SLA based policy
1.0.1	⋮
1.0.0	⋮

At the bottom, there's a button labeled '+ Add new version'.

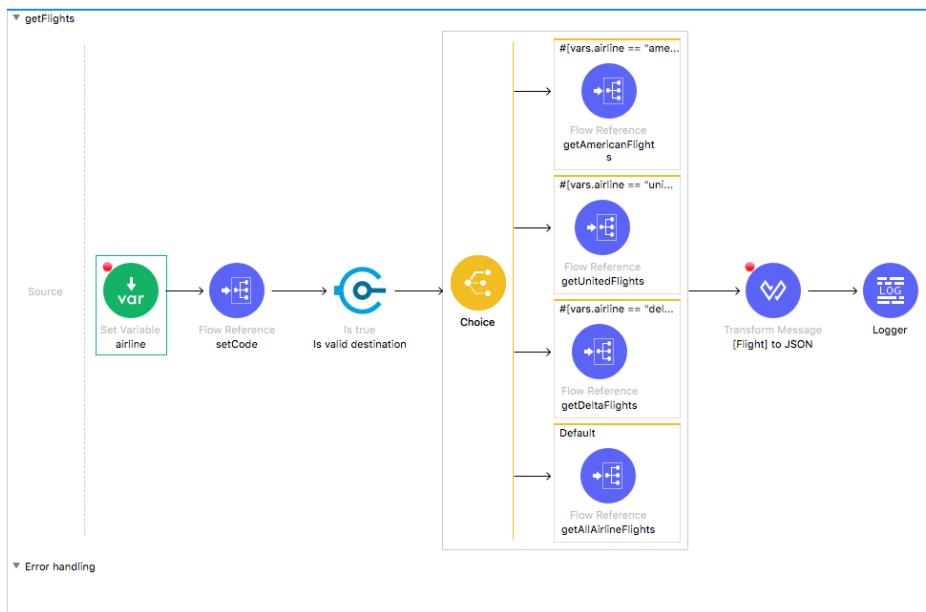
37. Click the GET method for the flights resource and select the Headers tab; you should see required text fields for client_id and client_secret and no longer need to add the headers manually for each request.

Note: You will test and use the authentication with the REST connector later in the Fundamentals course.

The screenshot shows a user interface for testing a REST API. At the top, there are two tabs: "Parameters" and "Headers". The "Headers" tab is currently selected, indicated by a blue underline. Below the tabs, there are two input fields. The first field is labeled "client_id*" and has a red asterisk next to it, indicating it is a required parameter. The second field is labeled "client_secret*" and also has a red asterisk. Both fields have a small trash can icon to their right. At the bottom left is a "Send" button, and at the bottom right is a red message: "Fill in required parameters".

38. Close all Anypoint Platform browser windows and tabs.

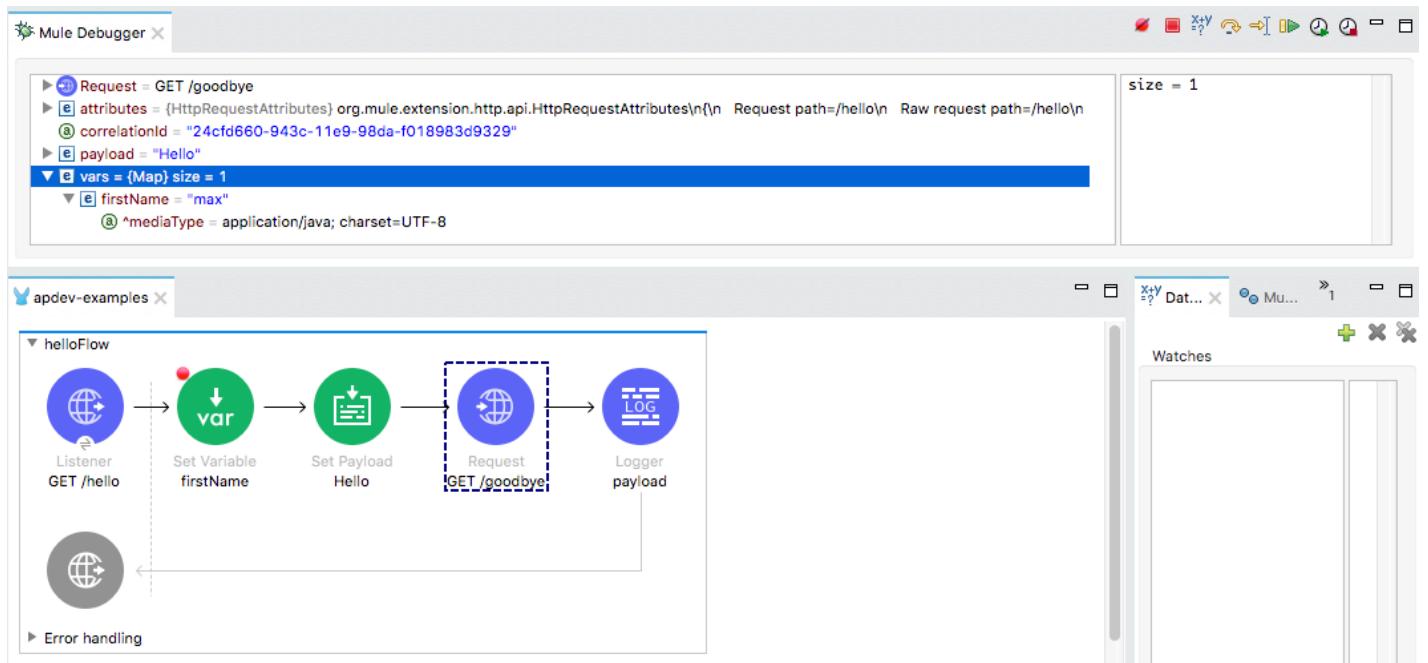
PART 2: Building Applications with Anypoint Studio



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

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

Module 6: Accessing and Modifying Mule Events



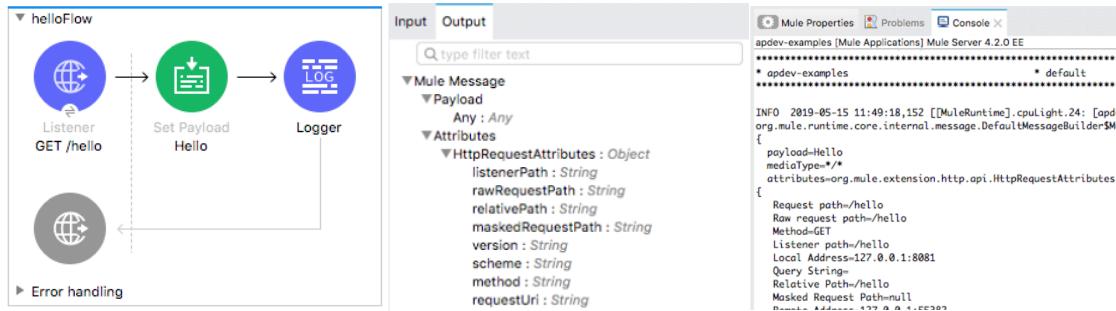
At the end this module, you should be able to:

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

Walkthrough 6-1: View event data

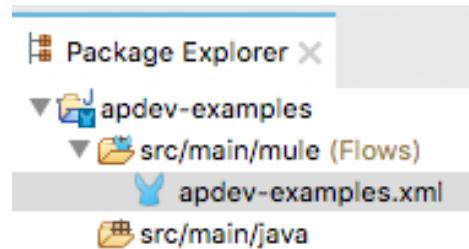
In this walkthrough, you create a new project to use in the next two modules to learn about Mule events and Mule applications. You will:

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



Create a new Mule project

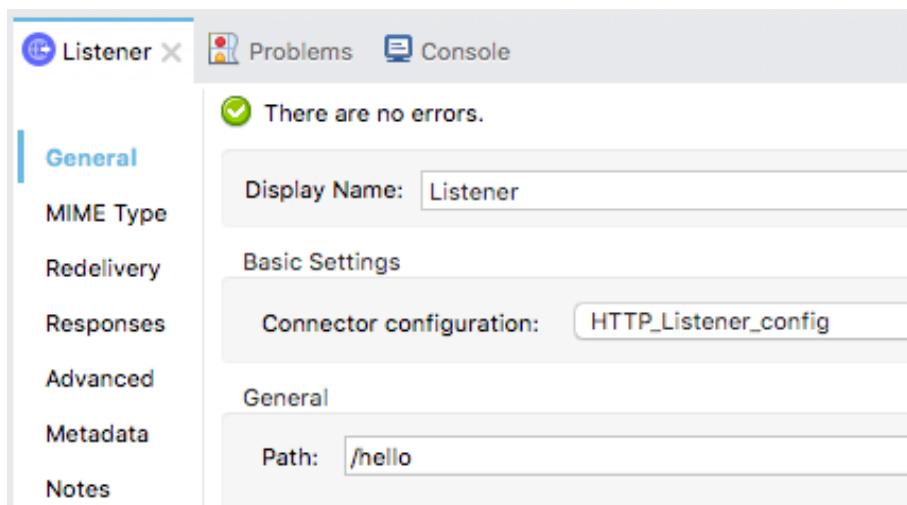
1. Return to Anypoint Studio.
2. Right-click training4-american-ws and select Close Project.
3. Select File > New > Mule Project.
4. Set the project name to apdev-examples and click Finish.



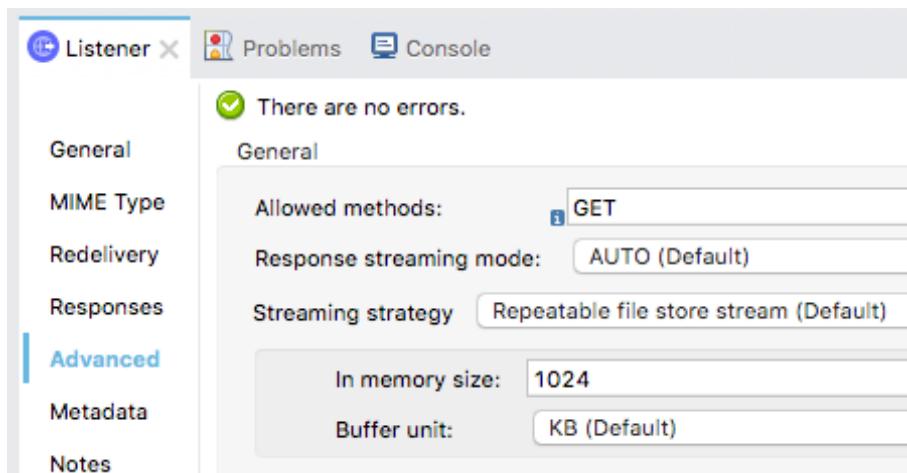
Create an HTTP Listener to receive requests

5. In the Mule Palette, select Favorites.
6. Drag an HTTP Listener from the Mule Palette to the canvas.
7. In the Listener properties view, click the Add button next to Connector configuration.
8. In the Global Element Properties dialog box, verify the host value is set to 0.0.0.0 and the port value to 8081.
9. Click OK.

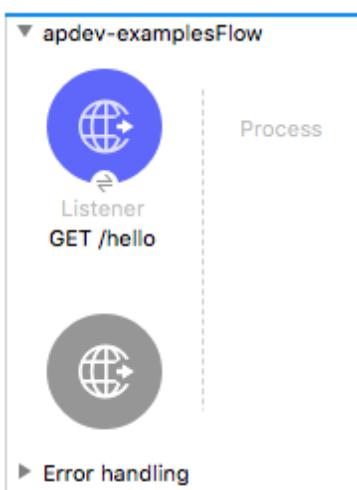
10. In the Listener properties view, set the path to /hello.



11. Click the Advanced tab and set the allowed methods to GET.



12. Click the General tab and set the display name to GET /hello.

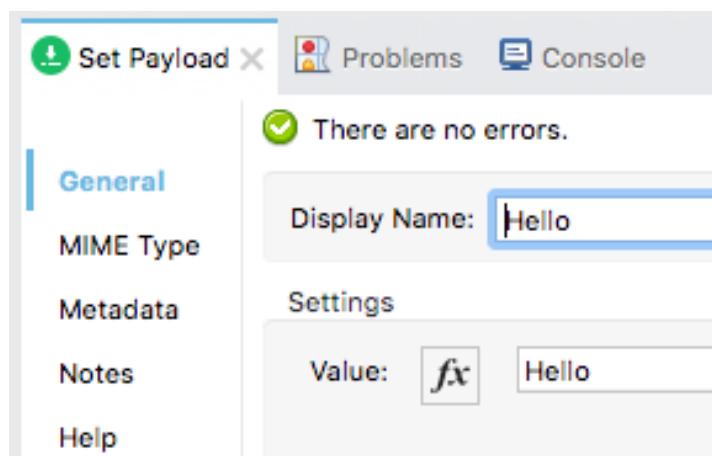


Change the flow name

13. Select the flow.
14. In the apdev-examplesFlow properties view, change the name to helloFlow.

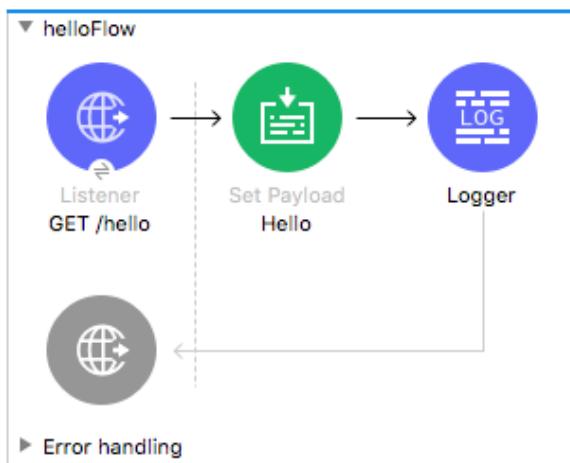
Set the event payload

15. Drag a Set Payload transformer from the Favorites section of the Mule Palette into the process section of the flow.
16. In the Set Payload properties view, set the display name to Hello.
17. Switch the value to literal mode and set its value to Hello.



Add a Logger

18. Drag a Logger component from the Mule Palette and drop it at the end of the flow.



View event structure and metadata in the DataSense Explorer

19. Select the GET /hello HTTP Listener and locate the DataSense Explorer in the right-side of its properties view.
20. In the Input tab, examine Payload and Attributes.

The screenshot shows the DataSense Explorer interface with the 'Input' tab selected. At the top, there are tabs for 'Input' and 'Output'. Below them is a search bar labeled 'type filter text'. The main content area displays the structure of a 'Mule Message':

- ▼ Mule Message
 - ▼ Payload
 - Undefined : Unknown
 - ▼ Attributes
 - Undefined : Unknown
- Variables

21. Select the Output tab and examine Payload, Attributes, and HttpRequestAttributes.

The screenshot shows the DataSense Explorer interface with the 'Output' tab selected. The structure of the 'Mule Message' is more detailed than in the Input tab:

- ▼ Mule Message
 - ▼ Payload
 - Any : Any
 - ▼ Attributes
 - ▼ HttpRequestAttributes : Object
 - listenerPath : String
 - rawRequestPath : String
 - relativePath : String
 - maskedRequestPath : String
 - version : String
 - scheme : String
 - method : String
 - requestUri : String
 - rawRequestUri : String
 - queryString : String
 - localAddress : String
 - remoteAddress : String
 - clientCertificate : Object?
 - queryParams : Object
 - uriParams : Object
 - requestPath : String
 - headers : Object
 - Variables

22. Select the Set Payload component in helloFlow.
23. In the DataSense Explorer, examine Payload and Attributes in the Input tab.

The screenshot shows the DataSense Explorer interface with the 'Input' tab selected. The structure of the 'Mule Message' is simplified compared to the previous screenshot:

- ▼ Mule Message
 - ▼ Payload
 - Any : Any
 - ▼ Attributes
 - HttpRequestAttributes : Object
- Variables

24. Select the Output tab and examine Payload and Attributes.

The screenshot shows the DataSense Explorer interface with the 'Output' tab selected. A search bar at the top says 'Q type filter text'. Below it is a tree structure under 'Mule Message': 'Payload' (String : String) and 'Attributes' (HttpRequestAttributes : Object). There is also a 'Variables' section.

25. Select the Logger component in helloFlow.

26. In the DataSense Explorer, examine Payload and Attributes in the Input tab.

27. Select the Output tab and examine Payload and Attributes.

Run the application and review response data

28. Save the file and run the project.

29. Return to Advanced REST Client and click the button to create a new tab.

Note: You are adding a new tab so that you can keep the request to your American API saved in another tab for later use.

30. In the new tab, make a GET request to <http://localhost:8081/hello>; you should see Hello displayed.

The screenshot shows the Advanced REST Client interface. At the top, 'Method' is set to 'GET' and 'Request URL' is 'http://localhost:8081/hello'. Below that is a 'Parameters' dropdown. On the right are 'SEND' and a more options button. Underneath, a green '200 OK' button indicates success, with '190.62 ms' response time. To the right is a 'DETAILS' dropdown. At the bottom, there are icons for copy, download, and refresh, followed by the word 'Hello'.

View event data in the Anypoint Studio console

31. Return to Anypoint Studio and look at the console.

32. Locate the data displayed by using the Logger.

33. Review the event attributes.

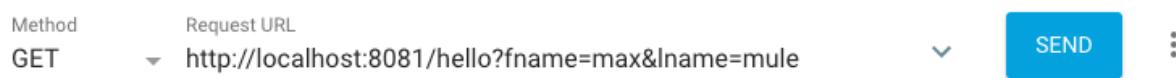


The screenshot shows the Anypoint Studio interface with the 'Console' tab selected. The output window displays the following log message:

```
apdev-examples [Mule Applications] Mule Server 4.2.0 EE
*****
* apdev-examples                               * default          * DEPLOYED      *
*****
INFO 2019-05-15 11:49:18,152 [[MuleRuntime].cpuLight.24: [apdev-examples].helloFlow.CPU_LITE @b0faf41] [ev
org.mule.runtime.core.internal.message.DefaultMessageBuilder$MessageImplementation
{
    payload=Hello
    mediaType=*/
    attributes=org.mule.extension.http.api.HttpRequestAttributes
    {
        Request path=/hello
        Raw request path=/hello
        Method=GET
        Listener path=/hello
        Local Address=127.0.0.1:8081
        Query String=
        Relative Path=/hello
        Masked Request Path=null
        Remote Address=127.0.0.1:55382
        Request Uri=/hello
        Raw request Uri=/hello
        Scheme=http
        Version=HTTP/1.1
        Headers=[
            host=localhost:8081
        ]
        Query Parameters={}
        URI Parameters={}
    }
    attributesMediaType=*/
}
```

Send query parameters with a request

34. Return to Advanced REST Client and add a query parameter with a key of fname and a value of max.
35. Add a second key/value pair of lname and mule.
36. Click Send.



The screenshot shows the Advanced REST Client configuration. The method is set to 'GET' and the request URL is 'http://localhost:8081/hello?fname=max&lname=mule'. There are 'SEND' and '⋮' buttons at the bottom right.

37. Return to Anypoint Studio and look at the console.

38. Locate the query parameters in the logged event data.

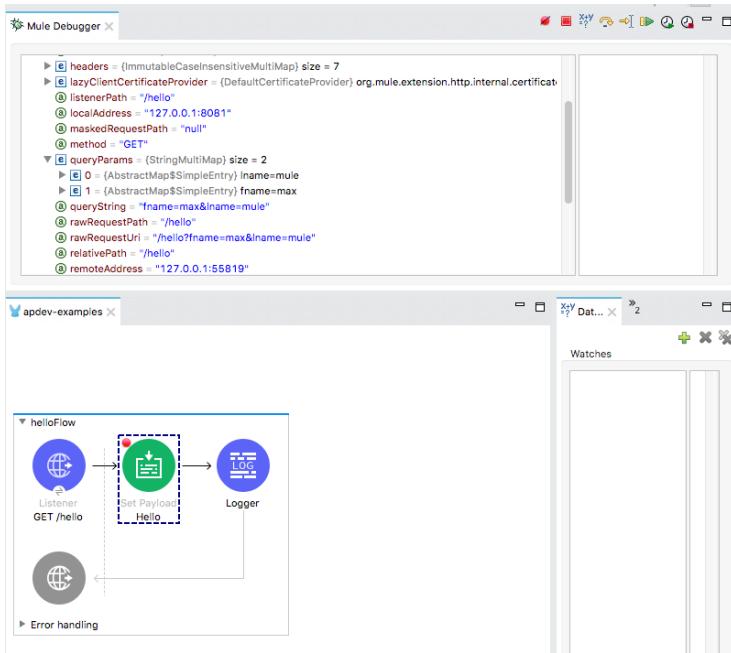
```
INFO 2019-05-15 12:01:45,043 [[MuleRuntime].cpuLight.24: [apdev-examples].helloFlow.CPU_LITE @b0faf41] [eve
org.mule.runtime.core.internal.message.DefaultMessageBuilder$MessageImplementation
{
    payload=Hello
    mediaType=*/
    attributes=org.mule.extension.http.api.HttpRequestAttributes
    {
        Request path=/hello
        Raw request path=/hello
        Method=GET
        Listener path=/hello
        Local Address=127.0.0.1:8081
        Query String=fname=max&lname=mule
        Relative Path=/hello
        Masked Request Path=null
        Remote Address=127.0.0.1:55435
        Request Uri=/hello?fname=max&lname=mule
        Raw request Uri=/hello?fname=max&lname=mule
        Scheme=http
        Version=HTTP/1.1
        Headers=[
            host=localhost:8081
        ]
        Query Parameters=[
            fname=max
            lname=mule
        ]
        URI Parameters={}
    }
    attributesMediaType=*/
}
```

39. Stop the project.

Walkthrough 6-2: Debug a Mule application

In this walkthrough, you debug and step through the code in a Mule application. You will:

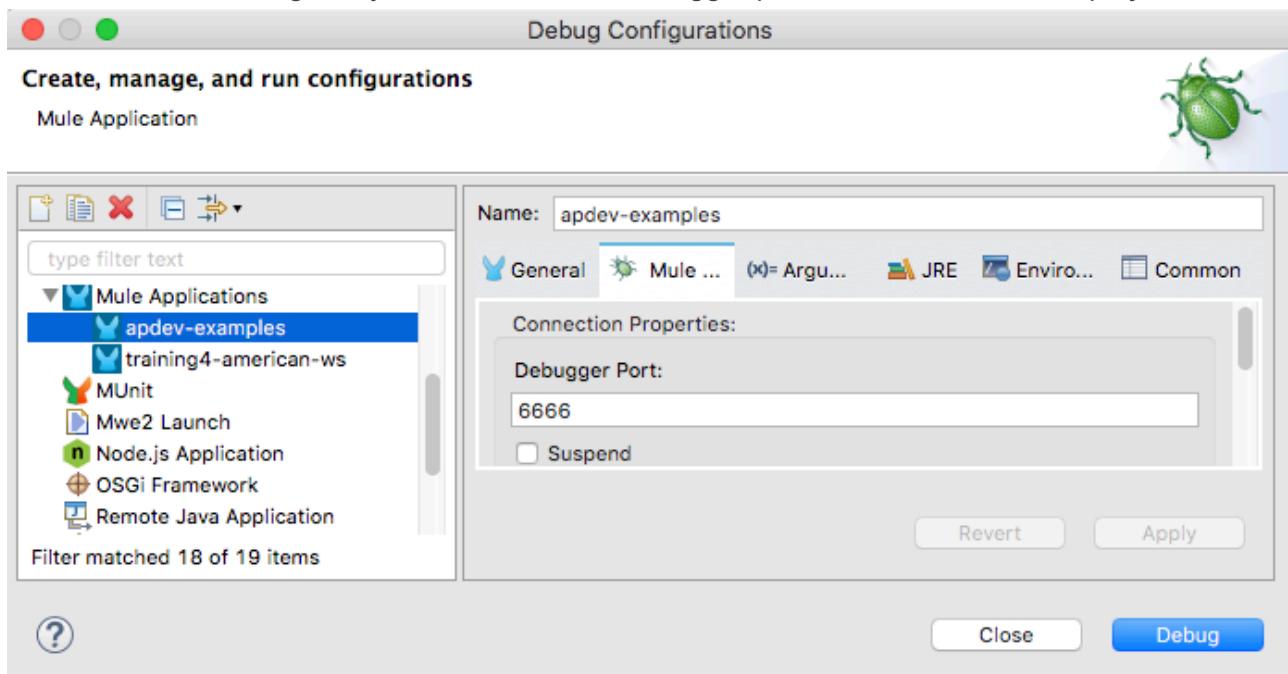
- Locate the port used by the Mule Debugger.
- Add a breakpoint, debug an application, and step through the code.
- Use the Mule Debugger to view event properties.
- Pass query parameters to a request and locate them in the Mule Debugger.
- Increase the request timeout for Advanced REST Client.



Locate the port used by the Mule Debugger

1. Return to Anypoint Studio.
2. In the main menu bar, select Run > Debug Configurations.
3. Select apdev-examples in the left menu bar under Mule Applications; you should see that the apdev-examples project is selected to launch.

4. Select the Mule Debug tab; you should see the debugger port is set to 6666 for the project.

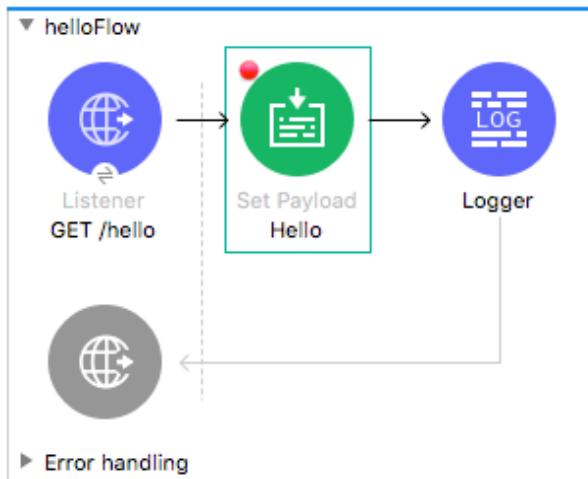


Note: If you know you have another program using port 6666 like McAfee on Windows, change this to a different value. Otherwise, you can test the Debugger first and come back and change the value here later if there is a conflict.

5. Click Close.

Add a breakpoint

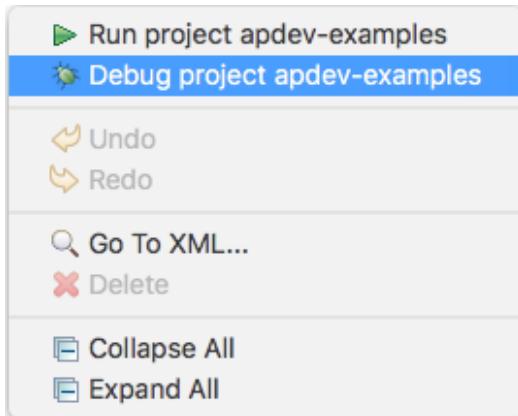
6. Right-click the Set Payload component and select Toggle breakpoint.



Debug the application

7. Right-click in the canvas and select Debug project apdev-examples.

Note: You can also select Run > Debug or click the Debug button in the main menu bar.



8. If you get a Confirm Perspective Switch dialog box, select Remember my decision and click Yes.
9. Look at the console and wait until the application starts.
10. In Advanced REST Client, make another request to <http://localhost:8081/hello?fname=max&lname=mule>.

View event data in the Mule Debugger

11. Return to Anypoint Studio and locate the Mule Debugger view.
12. Look at the value of the payload.
13. Expand Attributes and review the values.

14. Expand queryParams and review the values.

Mule Debugger X

```
▶ e headers = {ImmutableCaseInsensitiveMultiMap} size = 7
▶ e lazyClientCertificateProvider = {DefaultCertificateProvider} org.mule.extension.http.internal.certificat...
⑧ listenerPath = "/hello"
⑧ localAddress = "127.0.0.1:8081"
⑧ maskedRequestPath = "null"
⑧ method = "GET"
▼ e queryParams = {StringMultiMap} size = 2
  ▶ e 0 = {AbstractMap$SimpleEntry} lname=mule
  ▶ e 1 = {AbstractMap$SimpleEntry} fname=max
  ⑧ queryString = "fname=max&lname=mule"
  ⑧ rawRequestPath = "/hello"
  ⑧ rawRequestUri = "/hello?fname=max&lname=mule"
  ⑧ relativePath = "/hello"
  ⑧ remoteAddress = "127.0.0.1:55819"
```

apdev-examples X

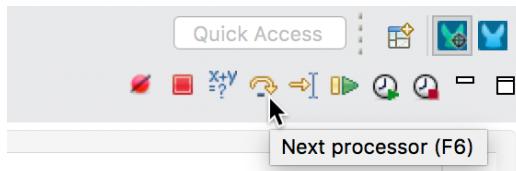
helloFlow

```
graph LR
    Listener[Listener  
GET /hello] --> SetPayload[Set Payload  
Hello]
    SetPayload --> Logger[Logger]
    Listener --> Listener
```

Error handling

Step through the application

15. Click the Next processor button.



16. Look at the new value of the payload; it should be Hello.

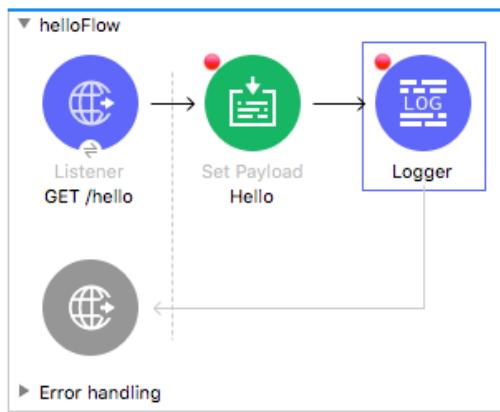
A screenshot of the Mule Debugger interface. The top window shows a stack trace with the payload variable highlighted in blue. To the right, a preview pane shows the value "Hello". Below the debugger is the Mule flow editor showing a simple helloFlow with a Listener, Set Payload, and Logger component. A 'Watches' panel on the right is empty.

17. Expand Attributes and review the values.

18. Click the Next processor button again to finish stepping through the application.

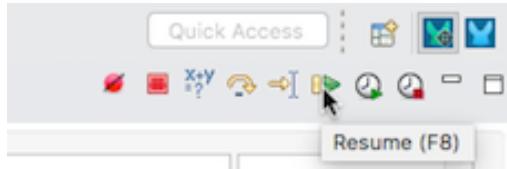
Use Resume to step to the next breakpoint and then the end of the application

19. Add a breakpoint to the Logger.



20. In Advanced REST Client, make another request to
<http://localhost:8081/hello?fname=max&lname=mule>.

21. In the Mule Debugger, click the Resume button; you should step to the Logger.



22. Click the Resume button again; you should step through the rest of the application.

Cause the HTTP request to timeout

23. In Advanced REST Client, make another request to
<http://localhost:8081/hello?fname=max&lname=mule>.

24. Do not step through the application and wait (45 seconds) for the request to timeout.

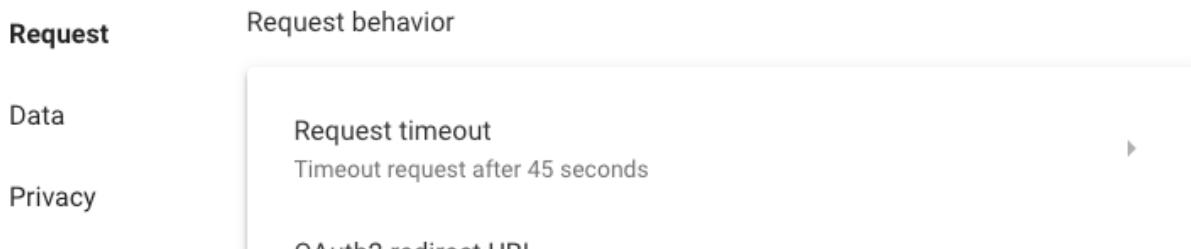
A screenshot of the Advanced REST Client interface. At the top, there is a red button labeled '0 Request error' and a 'DETAILS ▾' button. Below this, a large circular icon with a sad face and two crossed-out eyes is displayed. To its right, the text 'The requested URL can't be reached' is shown in red. Below that, a message in red text reads: 'The service might be temporarily down or it may have moved permanently to a new web address.' At the very bottom, the text 'Connection timeout.' is visible.

Increase the request timeout for Advanced REST Client

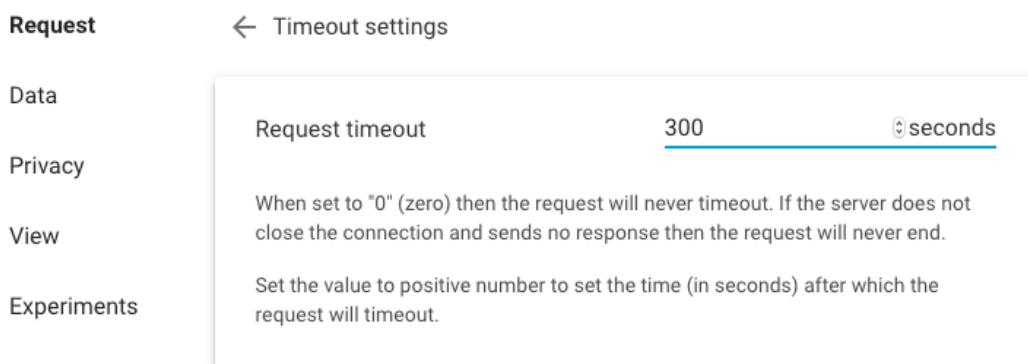
25. In the Advanced REST Client main menu, select File then Settings to navigate to the application settings.

Note: Depending on your platform, accessing the application settings may differ such as AdvancedRestClient then Preferences.

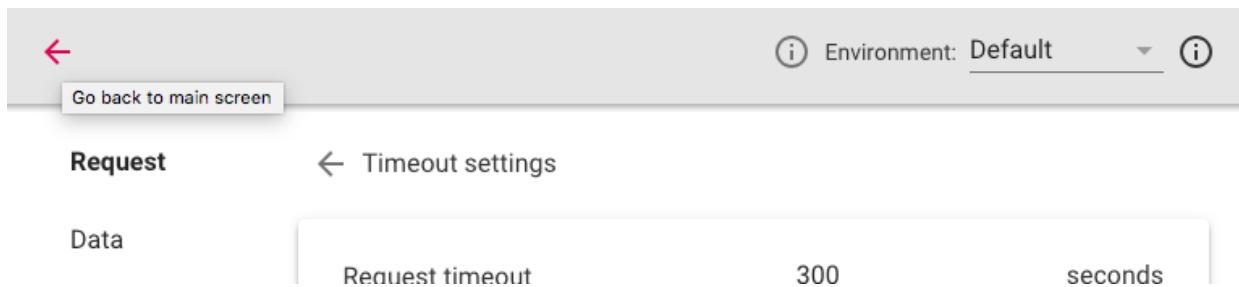
26. In the application settings, locate the Request timeout setting and click the arrow next to it.



27. Change the request timeout to 300 seconds.



28. Click the Go back to main screen button in the upper-left corner.

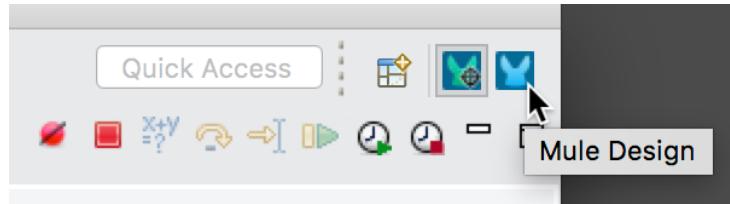


29. In the main screen, verify the tab with your last failed request is selected.

The screenshot shows a browser window with two tabs open. The active tab is titled "http://localhost:8081/hell..." and has a status message: "The requested URL can't be reached. The service might be temporarily down or it may have moved permanently to a new web address." A large circular icon with a sad face is displayed next to the error message. Below the tabs, there are fields for "Method" (set to "GET") and "Request URL" (set to "http://localhost:8081/hello?fname=max&lname=mule"). A "SEND" button is visible, along with a "DETAILS" link and a "Parameters" dropdown.

Switch perspectives

30. Return to Anypoint Studio.
31. Click the Resume button twice to finish stepping through the application.
32. Click the Mule Design button in the upper-right corner of Anypoint Studio to switch perspectives.



Note: 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.

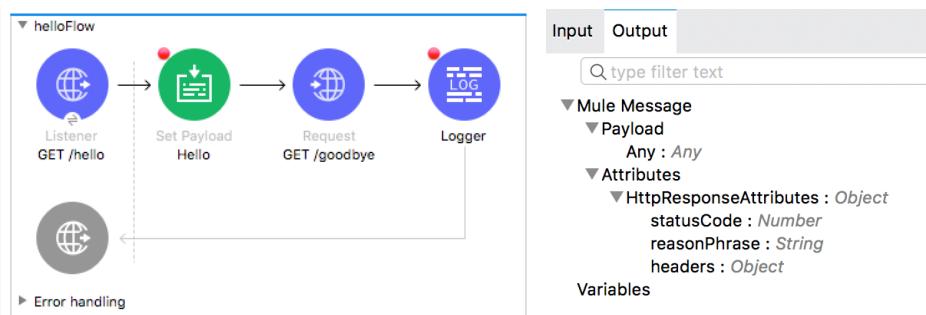
33. Leave the project running.

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

In this walkthrough, you call an external resource, which for simplicity is another HTTP Listener in the same application, so that you can watch event data as it moves in and out of a Mule application. You will:

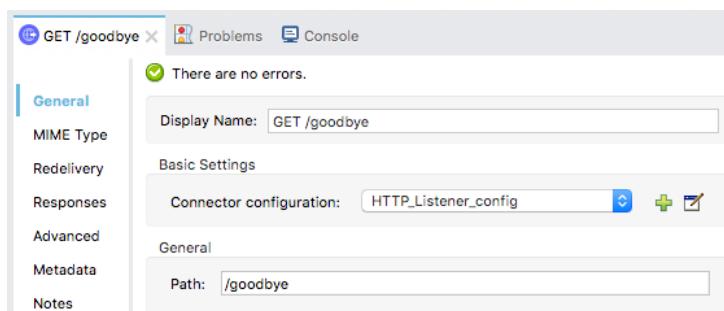
- Create a second flow with an HTTP Listener.
- Make an HTTP request from the first flow to the new HTTP Listener.
- View the event data as it moves through both flows.

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

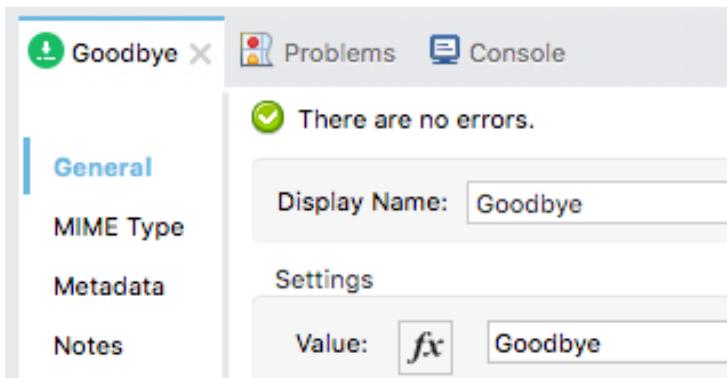


Create a second flow with an HTTP Listener

1. Return to apdev-examples.xml.
2. Drag an HTTP Listener from the Mule Palette and drop it in the canvas beneath the first flow.
3. Change the name of the flow to goodbyeFlow.
4. In the Listener view, ensure the connector configuration is set to the existing **HTTP_Listener_config**.
5. Set the path to **/goodbye** and the allowed methods to **GET**.
6. Set the display name to **GET /goodbye**.

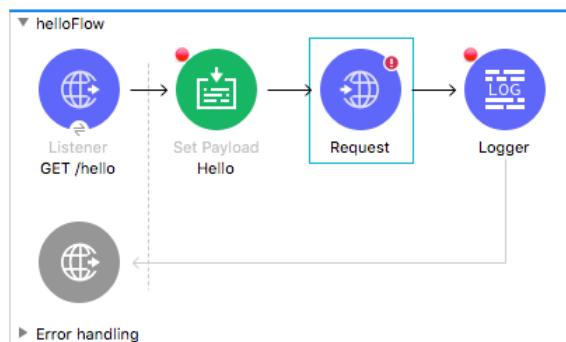


7. Add a Set Payload transformer and a Logger to the flow.
8. In the Set Payload properties view, set the display name and then use literal mode to set the value to Goodbye.



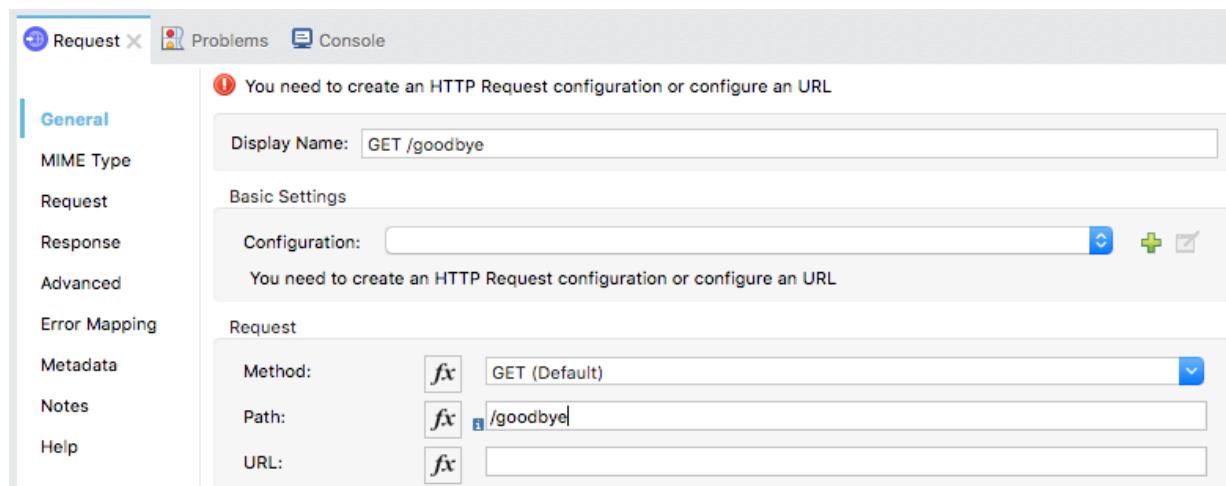
Make an HTTP request

9. From the Mule Palette, drag an HTTP Request to the canvas and drop it before the Logger in helloFlow.



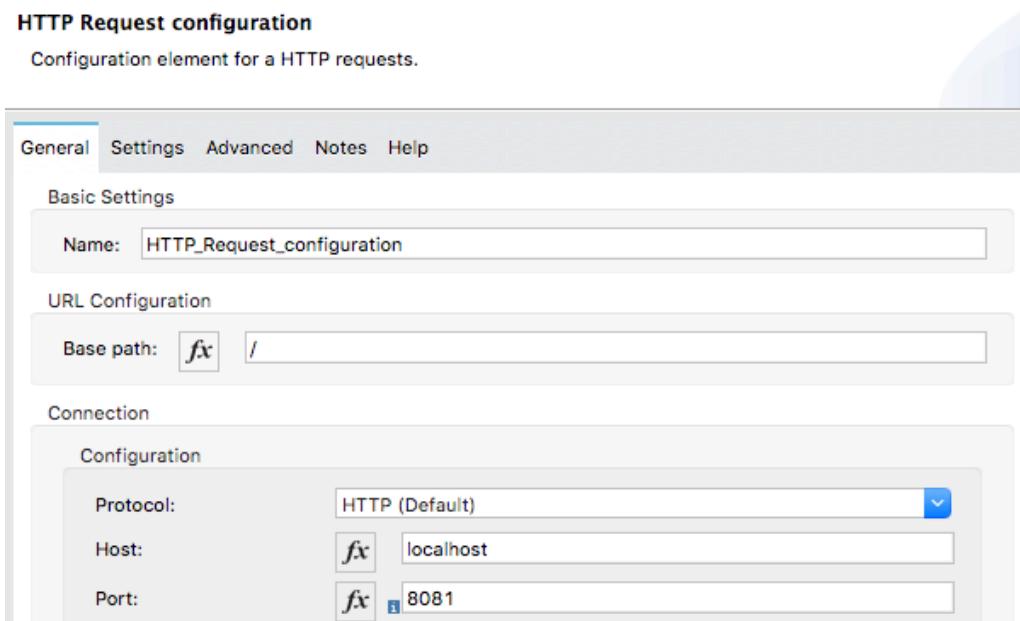
10. In the Request properties view, set the display name to GET /goodbye.

11. Set the path to /goodbye and leave the method set to GET.



12. Click the Add button next to configuration.

13. In the HTTP Request configuration dialog box, set the host to localhost and the port to 8081 and click OK.



View event structure and metadata in the DataSense Explorer

14. In the DataSense Explorer, examine Payload, Attributes, and HttpRequestAttributes on the Input tab.

The screenshot shows the DataSense Explorer interface with the 'Input' tab selected. At the top, there are two tabs: 'Input' (which is active) and 'Output'. Below the tabs is a search bar with the placeholder 'type filter text'. The main content area displays the structure of a 'Mule Message'. It includes sections for 'Payload' (with 'Actual' and 'Expected' types), 'Attributes' (including 'HttpRequestAttributes' with various properties like 'listenerPath', 'rawRequestPath', etc.), and other attributes like 'clientCertificate', 'queryParams', 'uriParams', 'requestPath', and 'headers'. A 'Variables' section is also present at the bottom.

15. Select the Output tab and examine Payload, Attributes, and HttpResponseAttributes.

The screenshot shows the DataSense Explorer interface with the 'Output' tab selected. At the top, there are two tabs: 'Input' and 'Output' (which is active). Below the tabs is a search bar with the placeholder 'type filter text'. The main content area displays the structure of a 'Mule Message'. It includes sections for 'Payload' (with 'Any' type), 'Attributes' (including 'HttpResponseAttributes' with properties like 'statusCode', 'reasonPhrase', and 'headers'), and a 'Variables' section.

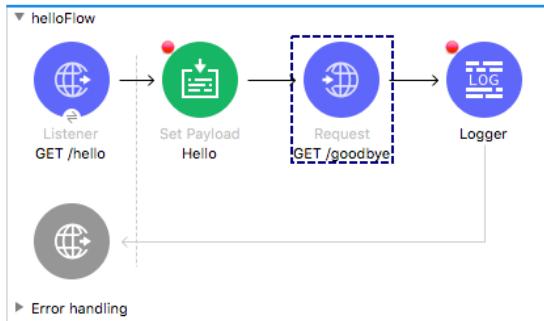
Change timeout for HTTP Request response

16. In the properties view for the GET /goodbye HTTP Request, click the Response tab.
17. Set the Response timeout to 300000.

Note: This is being set only for debugging purposes so that the HTTP Request does not timeout when you are stepping through the application and examining data in the Mule Debugger.

Debug the application

18. Save the file to redeploy the application.
19. In Advanced REST Client, send the same request to
<http://localhost:8081/hello?fname=max&lname=mule>.
20. In the Mule Debugger, step to the GET /goodbye request.

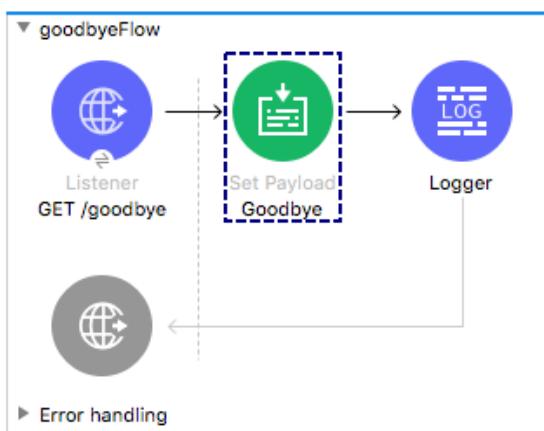


21. Look at the values of the payload and the attributes, including the queryParams.

Screenshot of the Mule Debugger showing the state of the Request object:

```
▶ e Request = GET /goodbye
  ▶ e attributes = {HttpRequestAttributes} org.mule.extension.http.api.HttpRequestAttributes\n{\n    @ DOUBLE_TAB = "  "
    @ TAB = " "
    @ ^mediaType = */*
    @ clientCertificate = {Certificate} null
  }
  ▶ e headers = {ImmutableCaseInsensitiveMultiMap} size = 1
  ▶ e lazyClientCertificateProvider = {DefaultCertificateProvider} org.mule.extension.http.internal
    @ listenerPath = "/hello"
    @ localAddress = "127.0.0.1:8081"
    @ maskedRequestPath = "null"
    @ method = "GET"
  ▶ e queryParams = {StringMultiMap} size = 2
    ▶ e 0 = {AbstractMap$SimpleEntry} lname=mule
    ▶ e 1 = {AbstractMap$SimpleEntry} fname=max
    @ queryString = "fname=max&lname=mule"
```

22. Step into goodbyeFlow.

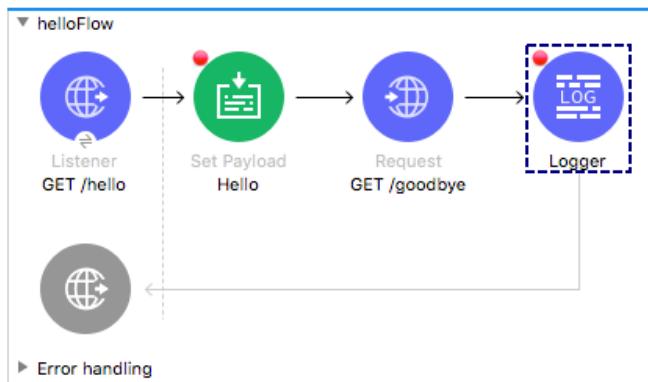


23. Look at the values of the payload and the attributes.

The screenshot shows the Mule Debugger interface with the title 'Mule Debugger'. The event stack is displayed, starting with a 'Set Payload' step. The payload is set to 'Goodbye'. Below the payload, there is a detailed view of the 'attributes' object, which is an instance of `org.mule.extension.http.api.HttpRequestAttributes`. The attributes include various properties such as `requestPath`, `rawRequestPath`, `method`, `headers`, and `payload`.

```
▶ [1] Set Payload = Goodbye
▼ [2] attributes = {HttpRequestAttributes} org.mule.extension.http.api.HttpRequestAttributes
  @ DOUBLE_TAB = "    "
  @ TAB = "  "
  @ ^mediaType = */*
  @ clientCertificate = {Certificate} null
  @ headers = {ImmutableCaseInsensitiveMultiMap} size = 6
  @ lazyClientCertificateProvider = {DefaultCertificateProvider} org.mule.extension.http.internal.certificate.DefaultCertificateProv
  @ listenerPath = "/goodbye"
  @ localAddress = "127.0.0.1:8081"
  @ maskedRequestPath = "null"
  @ method = "GET"
  @ queryParams = {ImmutableMultiMap} size = 0
  @ queryString = ""
  @ rawRequestPath = "/goodbye"
  @ rawRequestUri = "/goodbye"
  @ relativePath = "/goodbye"
  @ remoteAddress = "127.0.0.1:59664"
  @ requestPath = "/goodbye"
  @ requestUri = "/goodbye"
  @ schema = "http"
  @ serialVersionUID = 7227330842640270811
  @ uriParams = {EmptyMap} size = 0
  @ version = "HTTP/1.1"
  @ correlationId = "bde8e660-7b2e-11e9-b04d-f018983d9329"
  ▼ [3] payload =
    @ ^mediaType = */*; charset=UTF-8
```

24. Step through the flow until the event returns to the Logger in helloFlow.



25. Look at the values of the payload and the attributes.

The screenshot shows the Mule Debugger interface with the title 'Mule Debugger'. The event stack is displayed, starting with a 'Logger' step. The payload is set to 'Goodbye'. Below the payload, there is a detailed view of the 'attributes' object, which is an instance of `org.mule.extension.http.api.HttpResponseAttributes`. The attributes include properties such as `statusCode`, `reasonPhrase`, `serialVersionUID`, and `correlationId`.

```
▶ [1] Logger = Logger
▼ [2] attributes = {HttpResponseAttributes} org.mule.extension.http.api.HttpResponseAttributes
  @ DOUBLE_TAB = "    "
  @ TAB = "  "
  @ ^mediaType = */*
  @ headers = {ImmutableCaseInsensitiveMultiMap} size = 2
    @ reasonPhrase = ""
    @ serialVersionUID = -3131769059554988414
    @ statusCode = 200
  @ correlationId = "bde8e660-7b2e-11e9-b04d-f018983d9329"
  ▶ [3] payload = Goodbye
  @ vars = {Map} size = 0
```

26. Step through the rest of the application.

Review response data

27. Return to Advanced REST Client and view the return data and the http status code.
28. Click the Details button on the right side to expand this section.
29. Look at the response headers; you should see content-length, content-type, and date headers.

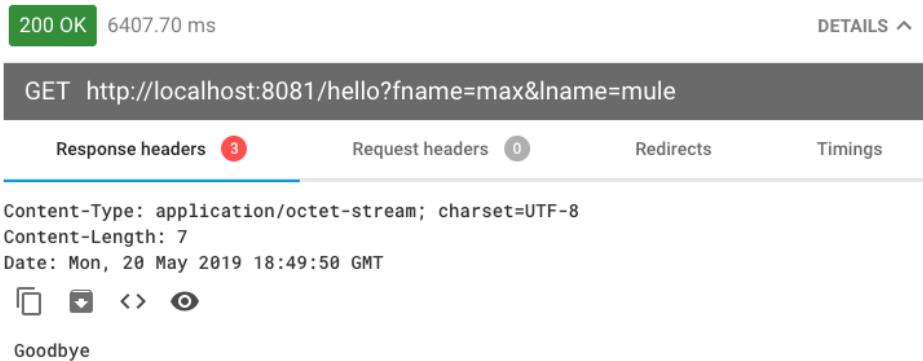
200 OK 6407.70 ms DETAILS ^

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

Response headers 3 Request headers 0 Redirects Timings

Content-Type: application/octet-stream; charset=UTF-8
Content-Length: 7
Date: Mon, 20 May 2019 18:49:50 GMT

Goodbye

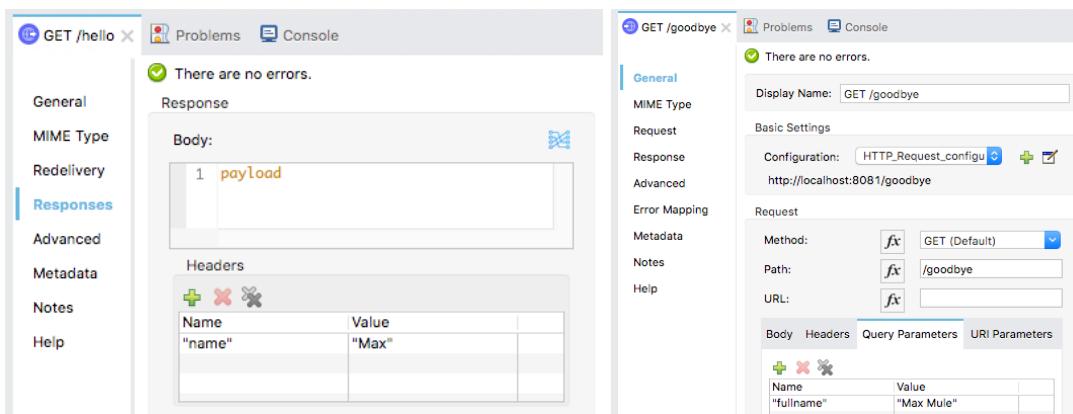


30. Return to Anypoint Studio and switch to the Mule Design perspective.

Walkthrough 6-4: Set request and response data

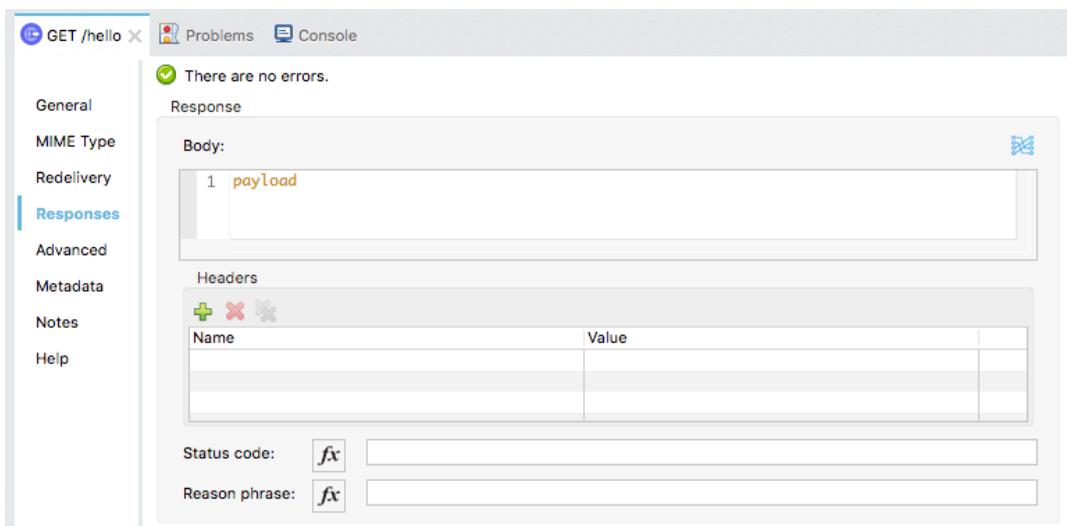
In this walkthrough, you set response and request data for HTTP Listener and HTTP Request operations. You will:

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



View default response body

1. Return to apdev-examples.xml.
2. Double-click the GET /hello HTTP Listener in helloFlow.
3. In the GET /hello properties view, click the Responses tab.
4. In the Response section, locate the expression that sets the response body by default to the value of the payload.



Set a response header

5. In the Headers section for the response, click the Add button.
6. Set the name to "name" and the value to "Max".
7. Locate the status code field and leave it blank so the default value 200 is returned.
8. Set the reason phrase to Success.

The screenshot shows the 'Headers' configuration dialog in Mule Studio. At the top, there's a table with two columns: 'Name' and 'Value'. A single row is present with 'name' in the Name column and '\"Max\"' in the Value column. Below the table, there are two input fields: 'Status code:' with a 'fx' icon and 'Reason phrase:' with a 'fx' icon and the word 'Success'.

Run the application and review response data

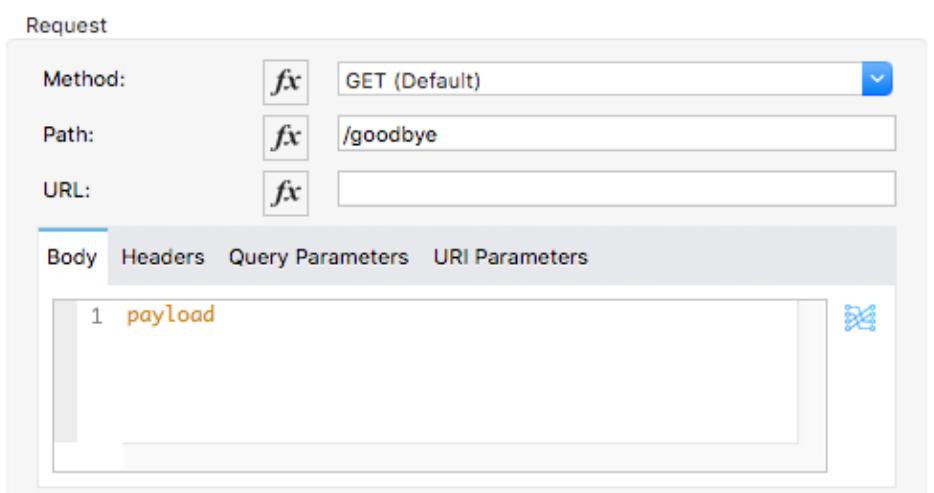
9. Save the file to deploy the project.
10. Return to Advanced REST Client and send the same request.
11. In the Mule Debugger, click Resume until you step through the application.
12. In Advanced REST Client, locate your new status code reason phrase.
13. Review the response headers; you should now see the new name header.

The screenshot shows the Advanced REST Client interface. At the top, it says '200 Success' and '8676.00 ms'. To the right, there's a 'DETAILS ^' link. Below that, a 'GET http://localhost:8081/hello?fname=max&lname=mule' request is shown. Under 'Response headers', there are four items: 'name: Max', 'Content-Type: application/octet-stream; charset=UTF-8', 'Content-Length: 7', and 'Date: Mon, 20 May 2019 19:21:18 GMT'. There are also tabs for 'Request headers' (0), 'Redirects', and 'Timings'. At the bottom, there are icons for download, copy, and refresh, followed by the text 'Goodbye'.

View default request body

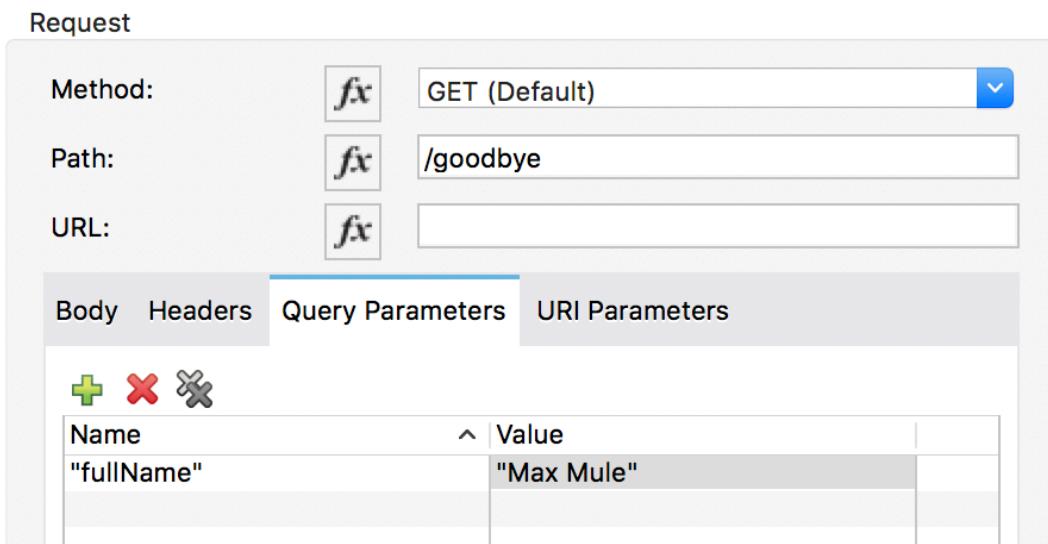
14. Return to Anypoint Studio and switch perspectives.

15. Double-click the GET /goodbye HTTP Request in helloFlow.
16. In the GET /goodbye properties view, locate the Request section on the General tab.
17. On the Body tab, locate the expression that sets the request body by default to the value of the payload.



Set a request query parameter

18. Select the Query Parameters tab and click the Add button.
19. Set the name to "fullName" and the value to "Max Mule".



Debug and verify the query parameter is sent with the request

20. Save the file to redeploy the project.

21. Return to Advanced REST Client and send the same request.
22. Return to the Mule Debugger and step into goodbyeFlow.
23. Expand Attributes and locate the fullName query parameter.



```
Mule Debugger X

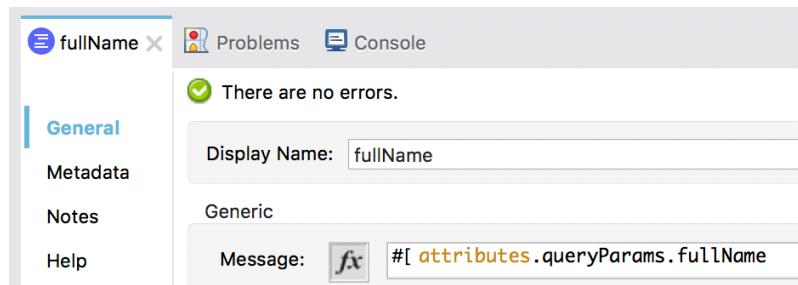
▶ Set Payload = Goodbye
▼ attributes = {HttpRequestAttributes} org.mule.extension.http.api.HttpRequestAttributes\n{\n  Request path=/
  @ DOUBLE_TAB = " "
  @ TAB = " "
  @ ^mediaType = /* 
  @ clientCertificate = {Certificate} null
  ► headers = {ImmutableCaseInsensitiveMultiMap} size = 6
  ► lazyClientCertificateProvider = {DefaultCertificateProvider} org.mule.extension.http.internal.certificate.Def
  @ listenerPath = "/goodbye"
  @ localAddress = "127.0.0.1:8081"
  @ maskedRequestPath = "null"
  @ method = "GET"
  ▼ queryParams = {StringMultiMap} size = 1
    ► 0 = {AbstractMap$SimpleEntry} fullName=Max Mule
    @ queryString = "fullName=Max Mule"
    @ rawRequestPath = "/goodbye"
```

24. Step through the rest of the application.
25. Switch to the Mule Design perspective.
26. Stop the project.

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

In this walkthrough, you get and set event data using DataWeave expressions. You will:

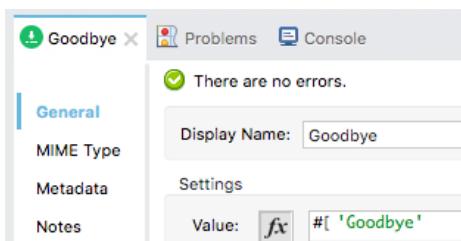
- Use expressions to set the payload and a logged value.
- Use expressions to set a response header and a request query parameter.
- In expressions, reference values for the event payload and attributes.
- Use the DataWeave upper() function, the concatenation, as, and default operators.



Use an expression to set the payload

1. Return to apdev-examples.xml.
2. Navigate to the properties view for the Goodbye Set Payload transformer in goodbyeFlow.
3. Use expression mode to change the value to an expression.

```
#['Goodbye']
```



4. Run the project.

5. In Advanced REST Client, send the same request; you should get the same result of Goodbye as before.

A screenshot of the Advanced REST Client interface. At the top, there is a green button labeled "200 Success" and the text "3165.37 ms". Below this are four small icons: a square with a double arrow, a downward arrow, a left-right arrow, and an eye. Underneath these icons, the word "Goodbye" is displayed in a monospaced font.

Use a DataWeave function

6. In Anypoint Studio, return to the Goodbye Set Payload properties view.
7. Inside the brackets and before the string, type the word up and press Ctrl+Spacebar.
8. Ensure auto-completion inserts the upper function.

A screenshot of the Anypoint Studio properties view for the "Goodbye" set payload. At the top, a green checkmark icon indicates "There are no errors". Below this, the "Display Name" is set to "Goodbye". Under the "Settings" section, the "Value" field contains the DataWeave expression "#[upper('Goodbye')]".

9. Move the Goodbye string into the parentheses.

```
# [upper( 'Goodbye' )]
```

10. Save the file to redeploy the application.

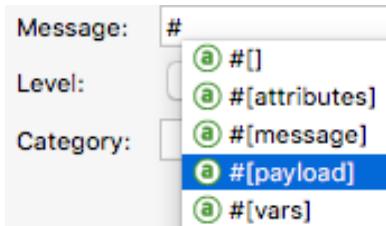
11. In Advanced REST Client, send the same request; the return string should now be in upper case.

A screenshot of the Advanced REST Client interface. At the top, there is a green button labeled "200 Success" and the text "2253.95 ms". Below this are four small icons: a square with a double arrow, a downward arrow, a left-right arrow, and an eye. Underneath these icons, the word "GOODBYE" is displayed in a monospaced font.

Use an expression that references the payload in a Logger

12. Return to Anypoint Studio.
13. Navigate to the properties view for the Logger in helloFlow.
14. Change the display name to payload.

15. Type # in the message field and double-click #[payload] in the auto-completion menu.



16. Save the file to redeploy the application.
17. In Advanced REST Client, send the same request.
18. Return to the console in Anypoint Studio; you should see GOODBYE displayed for the second Logger instead of the entire event object.

```
INFO 2019-05-20 17:24:27,590 [[MuleRuntime].cpuLight.24: [apdev-examples].helloFlow.CPU_LITE @af62455] [event: a583d0f1-7b45-11e9-b974-f018983d9329] org.mule.runtime.core.internal.processor.LoggerMessageProcessor: GOODBYE
```

Use a string literal and a DataWeave expression in a property value

19. Return to the properties view for the Logger in helloFlow.
20. Switch the message field to literal mode then change the value to display the string Message in front of the payload.

Message: #[payload]

21. Save the file to redeploy the application.
22. In Advanced REST Client, send the same request.
23. Return to the console in Anypoint Studio; you should see the new string displayed.

```
INFO 2019-05-20 17:35:38,943 [[MuleRuntime].cpuLight.23: [apdev-examples].helloFlow.CPU_LITE @4b708a6f] [event: 35f08380-7b47-11e9-b974-f018983d9329] org.mule.runtime.core.internal.processor.LoggerMessageProcessor: Message: GOODBYE
```

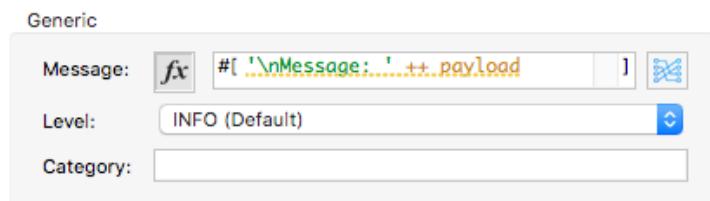
24. Return to the properties view for the Logger in helloFlow.

Use the DataWeave concatenation operator

25. Switch the message field to expression mode then change its value so the string is part of the evaluated expression and use the concatenation operator.

#['Message: ' ++ payload]

26. Add `\n` in front of the message to display it on a new line.



27. Save the file to redeploy the application.

28. In Advanced REST Client, send the same request.

29. Return to the console in Anypoint Studio; you should see the string displayed on a new line.

```
INFO 2019-05-20 17:38:57,360 [[MuleRuntime].cpuLight.23: [apdev-examples].helloFlow.CPU_LITE @1f410b9] [event: abf27981-7b47-11e9-b974-f018983d9329] org.mule.runtime.core.internal.processor.LoggerMessageProcessor: Message: GOODBYE
```

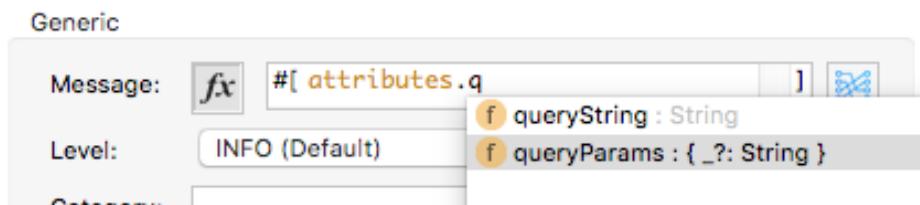
Use an expression that references an attribute in a Logger

30. Return to the properties view for the Logger in goodbyeFlow.

31. Change the display name to fullName.

32. Switch the message field to expression mode then type the word at and press Ctrl+Spacebar; you should see auto-completion insert the attributes keyword.

33. At the end of attributes, add a period, type q and double-click queryParams in the auto-completion menu.



##[attributes.queryParams]

34. Click Apply Changes to redeploy the application.

35. In Advanced REST Client, send the same request.

36. Return to the Anypoint Studio console; you should see an object displayed by the first Logger.

```
INFO 2019-05-20 21:59:21,841 [[MuleRuntime].cpuLight.24: [apdev-examples].goodbyeFlow.CPU_LITE @3709e5d] [event: 0ce290d1-7b6c-11e9-921d-f018983d9329] org.mule.runtime.core.internal.processor.LoggerMessageProcessor: {fullName=Max Mule}
```

37. Modify the message for the Logger in goodbyeFlow to display the value of the query parameter.

##[attributes.queryParams.fullName]

38. Click Apply Changes to redeploy the application.
39. In Advanced REST Client, send the same request.
40. Return to the console; you should now see the value of the parameter displayed.

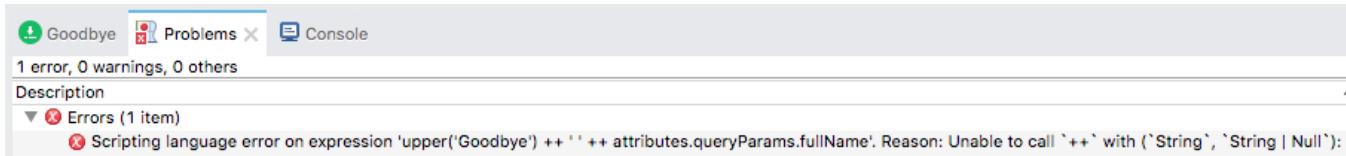
```
INFO 2019-05-20 22:01:50,910 [[MuleRuntime].cpuLight.23: [apdev-examples].goodbyeFlow.CPU_LITE @5ab79ee1] [event: 65b9f181-7b6c-11e9-921d-f018983d9329] org.mule.runtime.core.internal.processor.LoggerMessageProcessor: Max Mule
```

Use an expression that references an attribute when setting the payload

41. Navigate to the properties view for the Goodbye Set Payload in goodbyeFlow.
42. Use the concatenation operator to also display the value of the query parameter separated by a space.

```
# [upper('Goodbye') ++ ' ' ++ attributes.queryParams.fullName]
```

43. Select the Problems tab and review the error.



Use the as operator to coerce the attribute to a String

44. Use the as operator to also display the value of the query parameter separated by a space.

```
# [upper('Goodbye') ++ ' ' ++ attributes.queryParams.fullName as String]
```

Note: This step coerces a null value to a String, to avoid UI errors. The recommended way to handle null values is by using a default value which is explained later in this walkthrough.

45. Click Apply Changes to redeploy the application.
46. In Advanced REST Client, send the same request; you should now also see the name displayed.

The screenshot shows the Advanced REST Client results page. At the top, a green bar indicates '200 Success' with a response time of '329.76 ms'. Below this, there are several icons: a copy button, a refresh button, a diff button, and a zoom button. The main content area displays the response body: 'GOODBYE Max Mule'.

Use an expression to set a request header

47. Return to the Anypoint Studio and navigate to the properties view for the GET /goodbye HTTP Request in helloFlow.
48. In the Request section, select the Query Parameters tab.
49. Change the value of fullName to the value of the fname query parameter.

Name	Value
"fullName"	attributes.queryParams.fname

50. Click Apply Changes to redeploy the application.
51. In Advanced REST Client, send the same request; you should now see the name of the fname query parameter displayed.

200 Success 312.24 ms

DETAILS ▾

GOODBYE max

Make a request and do not send a query parameter

52. Remove the query parameters and make a request to <http://localhost:8081/hello>; you should get an error.

500 Server Error 164.60 ms DETAILS ▾

DETAILS ▾

HTTP GET on resource '<http://localhost:8081/goodbye>' failed: internal server error (500).

Set a default value in an expression

53. Return to the Anypoint Studio and navigate to the properties view for the Goodbye Set Payload in goodbyeFlow.
54. Remove as String.

55. Use the default operator to add a default value of Maxine.

```
# [upper('Goodbye') ++ ' ' ++ (attributes.queryParams.fullName default  
'Maxine')]
```

56. Click Apply Changes to redeploy the application.

57. In Advanced REST Client, send the same request; you should now see the default value Maxine displayed.

200 Success 307.33 ms

GOODBYE Maxine

Use a query parameter in the expression for a response header

58. Return to the Anypoint Studio and navigate to the properties view for the GET /hello HTTP Listener.

59. Select the Responses tab.

60. Change the name header value to the value of the fname query parameter.

```
attributes.queryParams.fname
```

Headers	
Name	Value
"name"	attributes.queryParams.fname

61. Click Apply Changes to redeploy the application.

62. In Advanced REST Client, add an fname and set it equal to max or some other value and send the request.

63. Look at the response headers; you should no longer see a name header.

200 Success 411.30 ms DETAILS ^

GET http://localhost:8081/hello?fname=max

Response headers 3 Request headers 0 Redirects Timings

Content-Type: application/java; charset=UTF-8
Content-Length: 11
Date: Tue, 21 May 2019 02:21:45 GMT

GOODBYE max

Debug and verify the query parameter is sent with the request

64. Return to Anypoint Studio.
65. Stop and then debug the project.
66. Return to Advanced REST Client and send the same request.
67. Return to the Mule Debugger and look at the attributes and query parameters; you should see the fname query parameter.

```
▼ e attributes = {HttpRequestAttributes} org.mule.extension.http.api.HttpRequestAttrib
  @ DOUBLE_TAB = "    "
  @ TAB = "  "
  @ ^mediaType = */
  @ clientCertificate = {Certificate} null
  ► e headers = {ImmutableCaseInsensitiveMultiMap} size = 1
  ► e lazyClientCertificateProvider = {DefaultCertificateProvider} org.mule.extension.
    @ listenerPath = "/hello"
    @ localAddress = "127.0.0.1:8081"
    @ maskedRequestPath = "null"
    @ method = "GET"
  ▼ e queryParams = {StringMultiMap} size = 1
    ► e 0 = {AbstractMap$SimpleEntry} fname=max
    @ queryString = "fname=max"
  ↻
```

68. Step into goodbyeFlow and look at the attributes and query parameters; you should see the fullName query parameter.

```
▼ e queryParams = {StringMultiMap} size = 1
  ► e 0 = {AbstractMap$SimpleEntry} fullName=max
  @ queryString = "fullName=max"
  @ rawRequestPath = "/goodbye"
```

69. Step back to helloFlow and look at the value of the attributes; you should not see any query parameters.

```
▼ [e] attributes = {HttpResponseAttributes} org.mule.extension.http.api.HttpResponse
  @ DOUBLE_TAB = "    "
  @ TAB = "  "
  @ ^mediaType = */*
► [e] headers = {ImmutableCaseInsensitiveMultiMap} size = 3
  @ reasonPhrase = ""
  @ serialVersionUID = -3131769059554988414
  @ statusCode = 200
  @ correlationId = "c926a490-7b6f-11e9-a330-f018983d9329"
► [e] payload = GOODBYE max
```

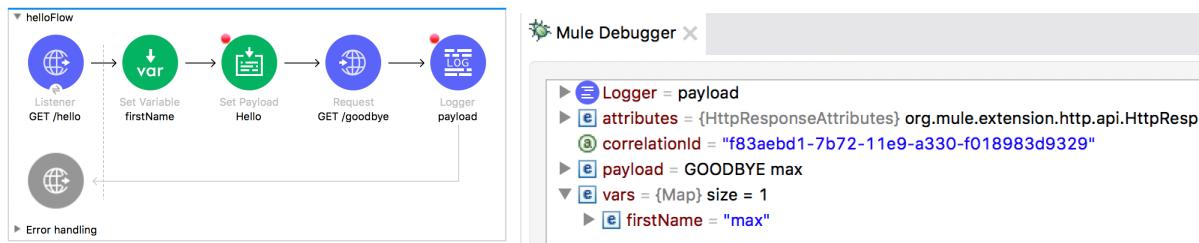
70. Step through the rest of the application.

71. Switch to the Mule Design perspective.

Walkthrough 6-6: Set and get variables

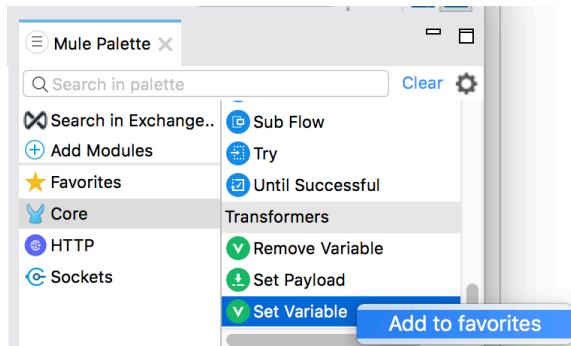
In this walkthrough, you create variables associated with an event. You will:

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



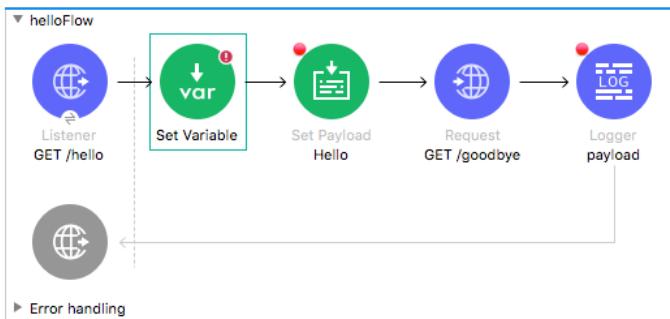
Add the Set Variable transformer to the Favorites section of the Mule Palette

1. Return to apdev-examples.xml.
2. In the Mule Palette, select Core.
3. Locate the Set Variable transformer and right-click it and select Add to favorites.

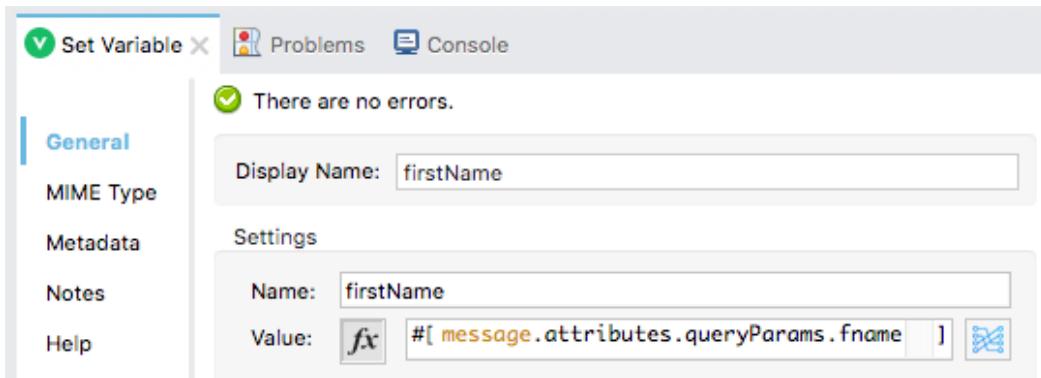


Create a variable

4. Drag the Set Variable transformer from the Favorites section of the Mule Palette and drop it after the GET /hello HTTP Listener in helloFlow.



5. In the Set Variable properties view, set the display name and name to firstName.
6. Set the value to your query parameter, fname.



Use an expression that references the variable to set a response header

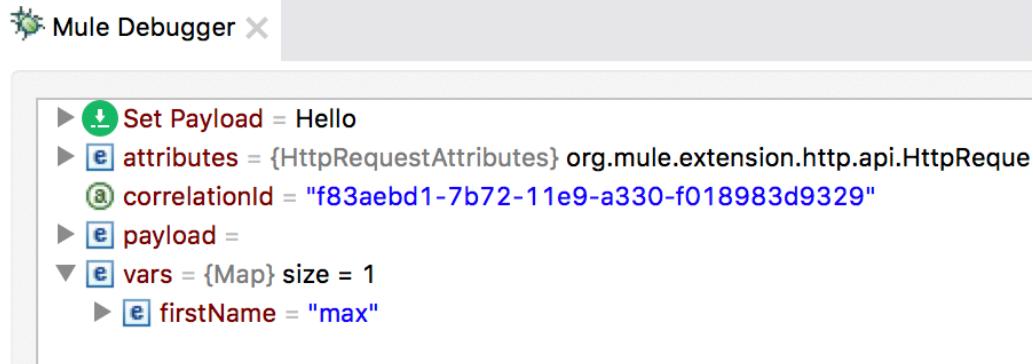
7. Return to the properties view for the GET /hello HTTP Listener in helloFlow.
8. Select the Responses tab.
9. Change the name header value from attributes.queryParams.fname to the value of the firstName variable.

vars.firstName



View variables in the Mule Debugger

10. Save the file to redeploy the project in debug mode.
11. In Advanced REST Client, send the same request.
12. In the Mule Debugger, locate and expand the new Variables section; you should see your firstName variable.



```
▶ Set Payload = Hello
▶ e attributes = {HttpRequestAttributes} org.mule.extension.http.api.HttpRequestAttributes
@ correlationId = "f83aebd1-7b72-11e9-a330-f018983d9329"
▶ e payload =
▼ e vars = {Map} size = 1
  ▶ e firstName = "max"
```

13. Step into goodbyeFlow; you should see no variables in the vars section.



```
▶ Set Payload = Goodbye
▶ e attributes = {HttpRequestAttributes} org.mule.extension.http.api.HttpRequestAttributes
@ correlationId = "f83aebd1-7b72-11e9-a330-f018983d9329"
▶ e payload =
@ vars = {Map} size = 0
```

14. Step back to helloFlow; you should see the Variables section again with your firstName variable.



```
▶ e Logger = payload
▶ e attributes = {HttpResponseAttributes} org.mule.extension.http.api.HttpResponseAttributes
@ correlationId = "f83aebd1-7b72-11e9-a330-f018983d9329"
▶ e payload = GOODBYE max
▼ e vars = {Map} size = 1
  ▶ e firstName = "max"
```

15. Step through the rest of the application.

16. Return to Advanced REST Client; you should see the header with the value of the query parameter.

200 Success 159858.30 ms DETAILS ^

GET http://localhost:8081/hello?fname=max

Response headers 4 Request headers 0 Redirects Timings

```
name: max
Content-Type: application/java; charset=UTF-8
Content-Length: 11
Date: Tue, 21 May 2019 02:51:33 GMT
```

✖️ ↴ <> ⚡

GOODBYE max

17. Change the value of the query parameter and send another request.
18. In the Mule Debugger, click the Resume button until you step through the application.
19. Return to Advanced REST client; you should see the new query parameter value returned in both the body and the header.

Method Request URL

GET ▾ http://localhost:8081/hello?fname=Maxwell

SEND ⋮

Parameters ▾

200 Success 7324.50 ms DETAILS ^

GET http://localhost:8081/hello?fname=Maxwell

Response headers 4 Request headers 0 Redirects Timings

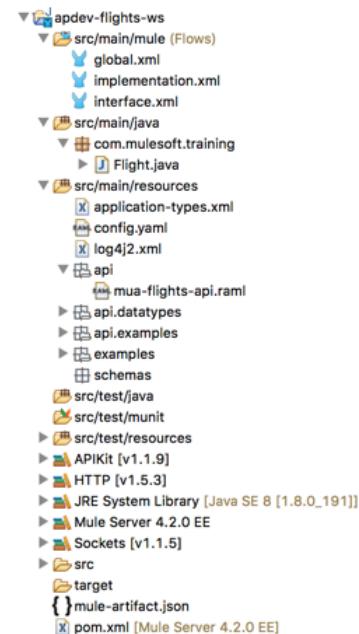
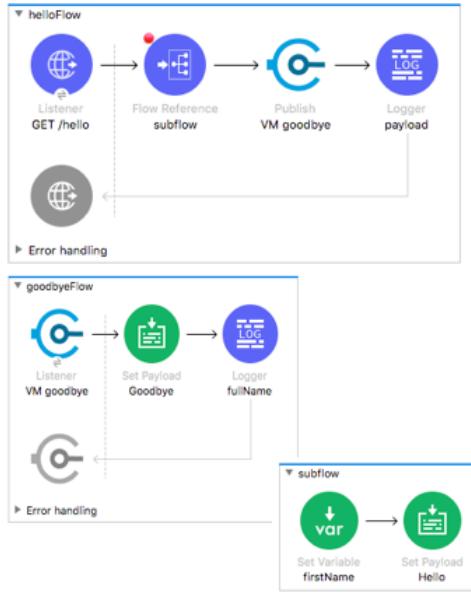
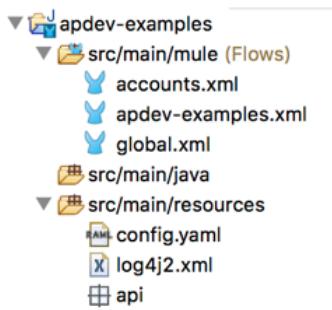
```
name: Maxwell
Content-Type: application/java; charset=UTF-8
Content-Length: 15
Date: Tue, 21 May 2019 02:53:03 GMT
```

✖️ ↴ <> ⚡

GOODBYE Maxwell

20. Return to Anypoint Studio and switch perspectives.

Module 7: Structuring Mule Applications



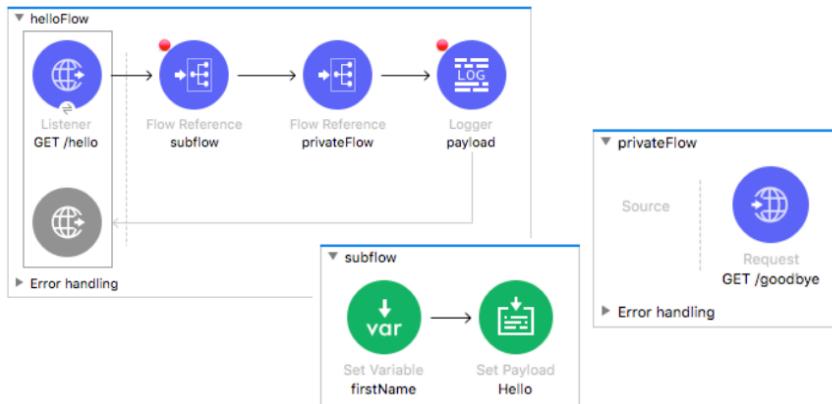
At the end of this module, you should be able to:

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

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

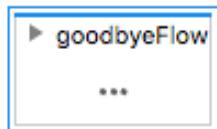
In this walkthrough, you continue to work with apdev-examples.xml. You will:

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



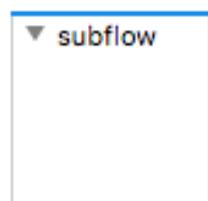
Collapse a flow

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

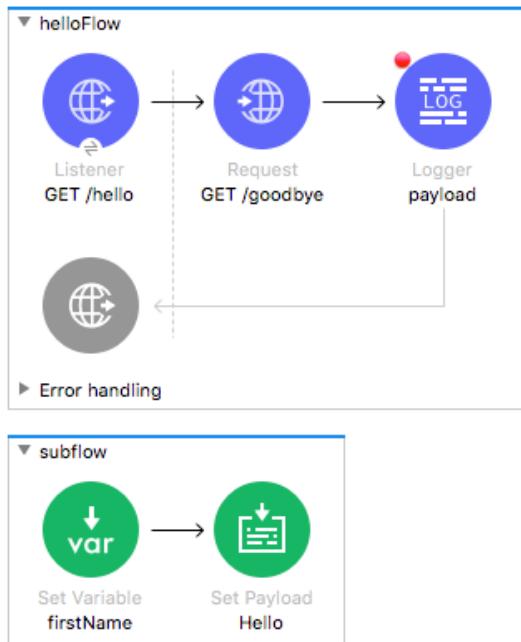


Create a subflow

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

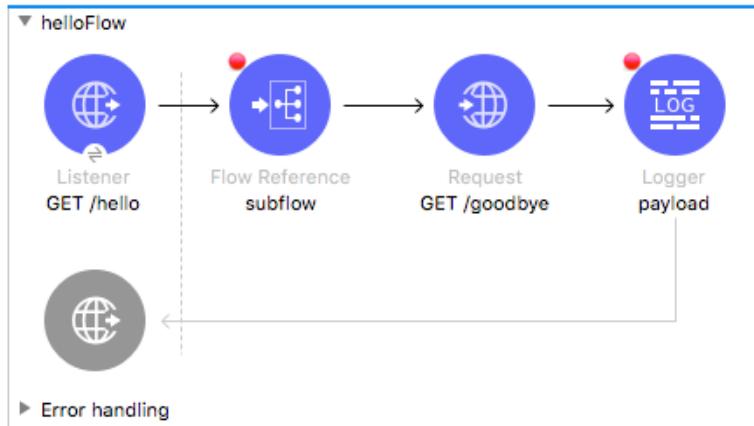


- Remove the breakpoint from the Set Payload transformer in helloFlow.
- Select the Set Variable and Set Payload transformers in helloFlow and drag them into the subflow.



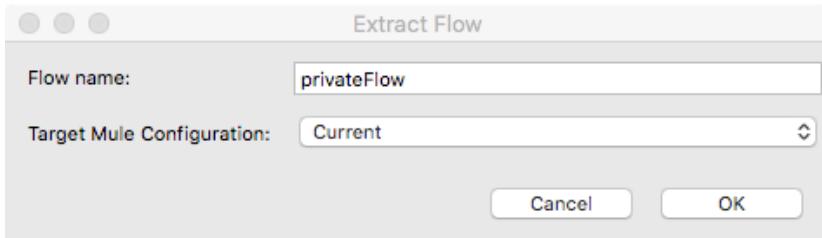
Reference a subflow

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

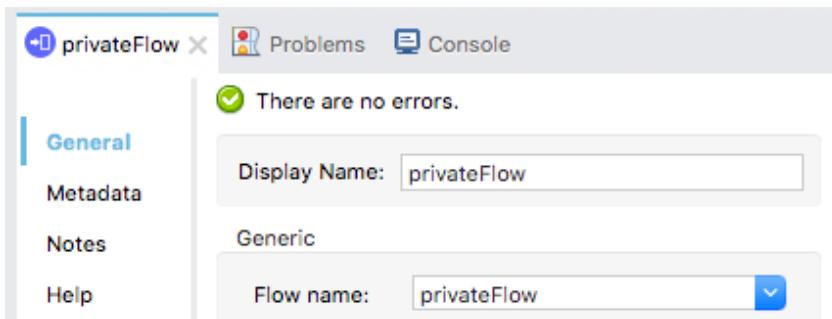


Extract processors into another flow

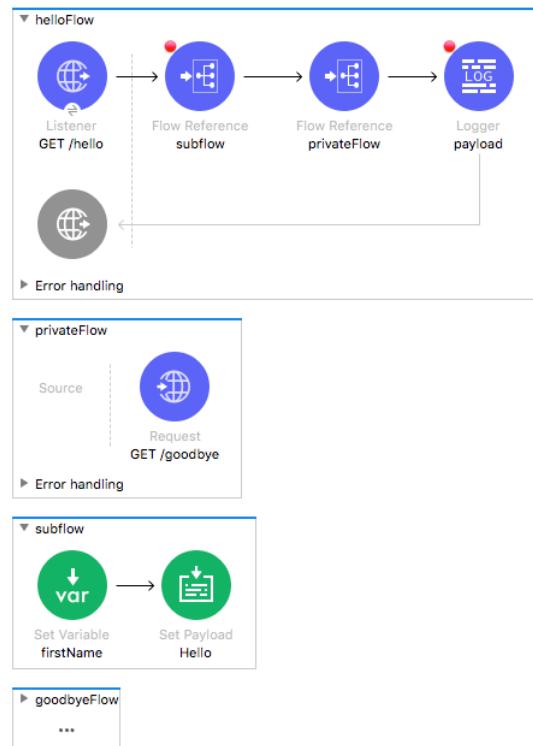
11. Right-click the GET /goodbye HTTP Request and select Extract to > Flow.
12. In the Extract Flow dialog box, set the flow name to privateFlow.



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

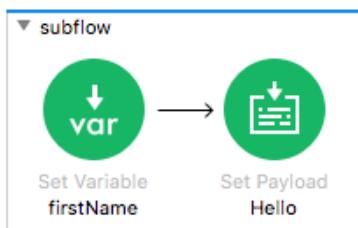


15. Drag privateFlow above subflow in the canvas.



Debug the application

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



19. Step through the rest of the application.
20. In Advanced REST Client, send the same request again.
21. In the Mule Debugger, step through the application again, this time watching the values of the attributes, payload, and variables in each of the flows.

The screenshot shows the Mule Debugger interface with three main sections:

- Mule Debugger**: Shows a list of attributes for a correlation ID: correlationId = "796b2da0-7cdb-11e9-a772-f018983d9329". Below it, the payload is listed as "Hello".
- apdev-examples**: Shows an "Error handling" section.
- privateFlow**: Shows a "Source" component followed by a "Request" component with the URL "GET /goodbye".
- subflow**: Shows the same subflow configuration as the first diagram: "Set Variable firstName" and "Set Payload Hello".

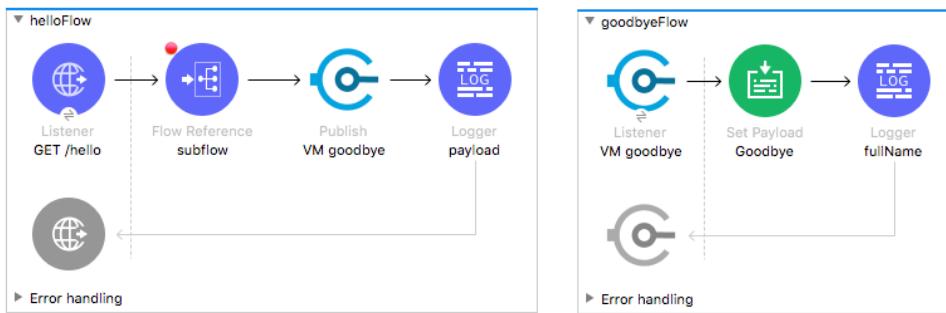
At the bottom, there are links for "Message Flow", "Global Elements", and "Configuration XML".

22. Step through the rest of the application.
23. Switch to the Mule Design perspective.

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

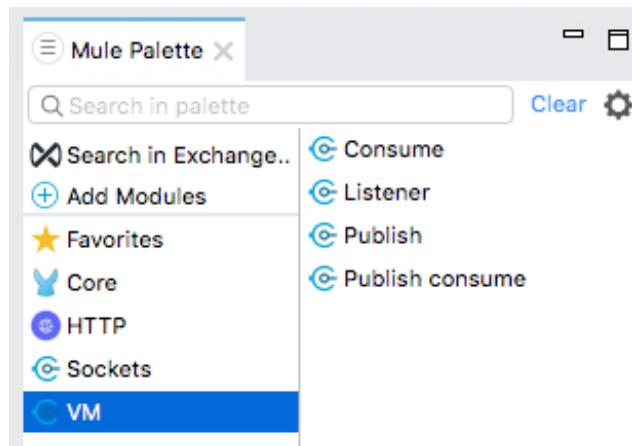
In this walkthrough, you pass events between flows using asynchronous queues. You will:

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

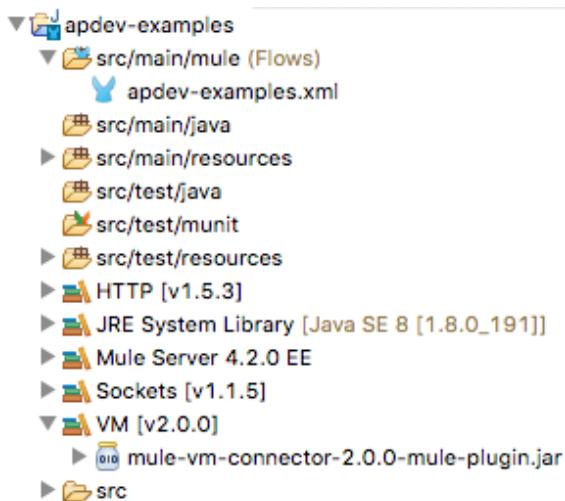


Add the VM module to the project

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

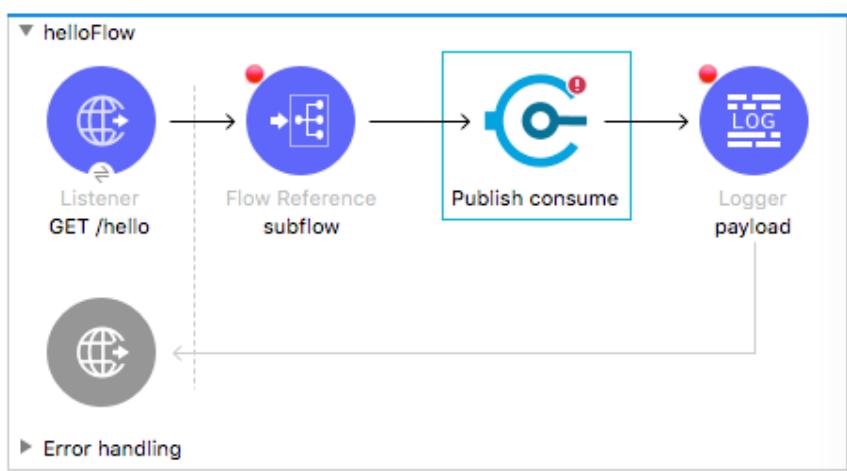


- In the Project Explorer, locate the JAR file for the VM connector.

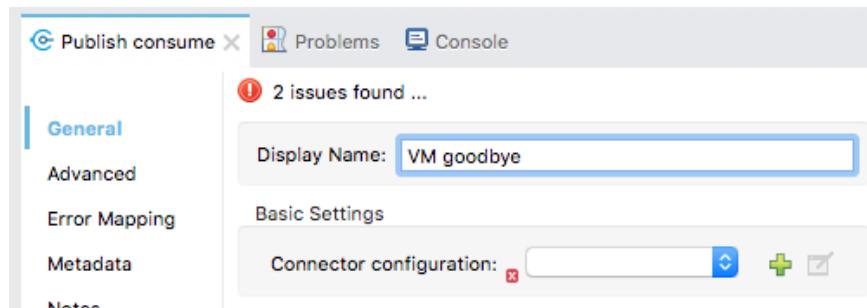


Add a VM Publish Consume operation

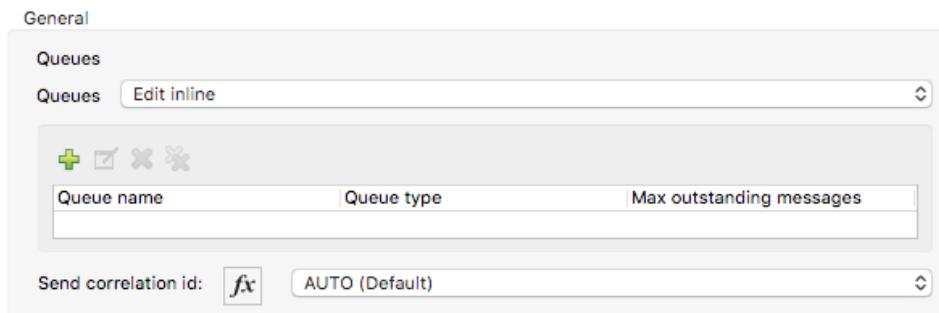
- In helloFlow, delete the privateFlow Flow Reference.
- Select VM in the Mule Palette.
- Select the Publish consume operation and drag and drop it before the Logger in helloFlow.



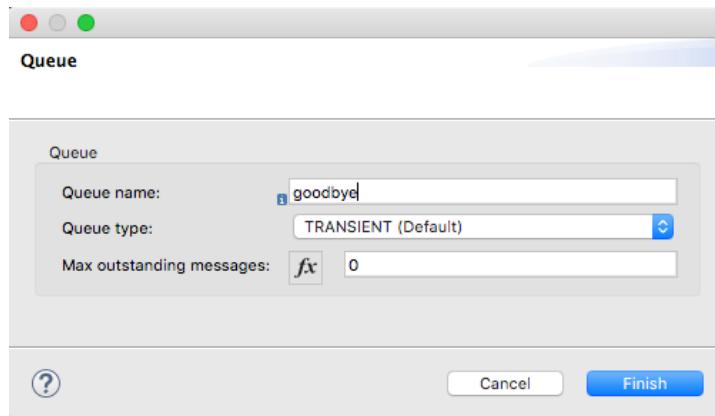
- In the Publish consume properties view, change the display name to VM goodbye.



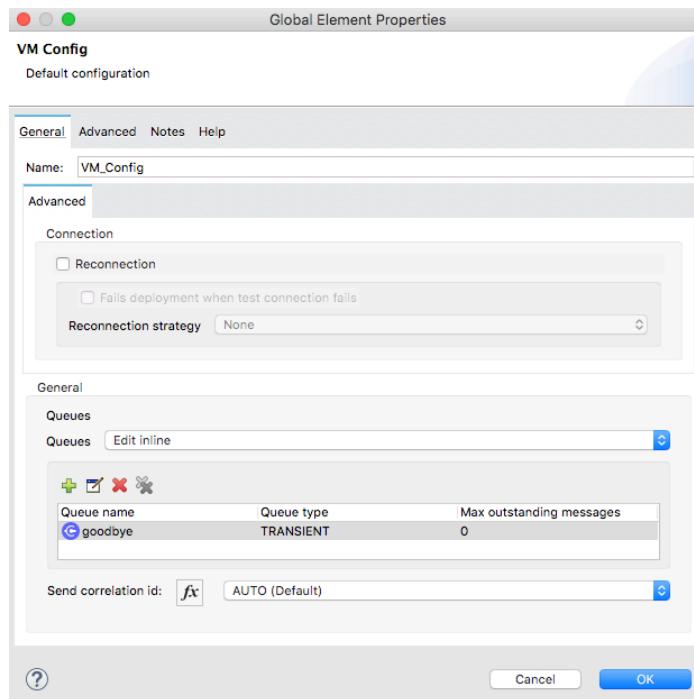
9. Click the Add button next to connector configuration.
10. In the queues drop-down menu, select Edit inline then click the Add Queue button.



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



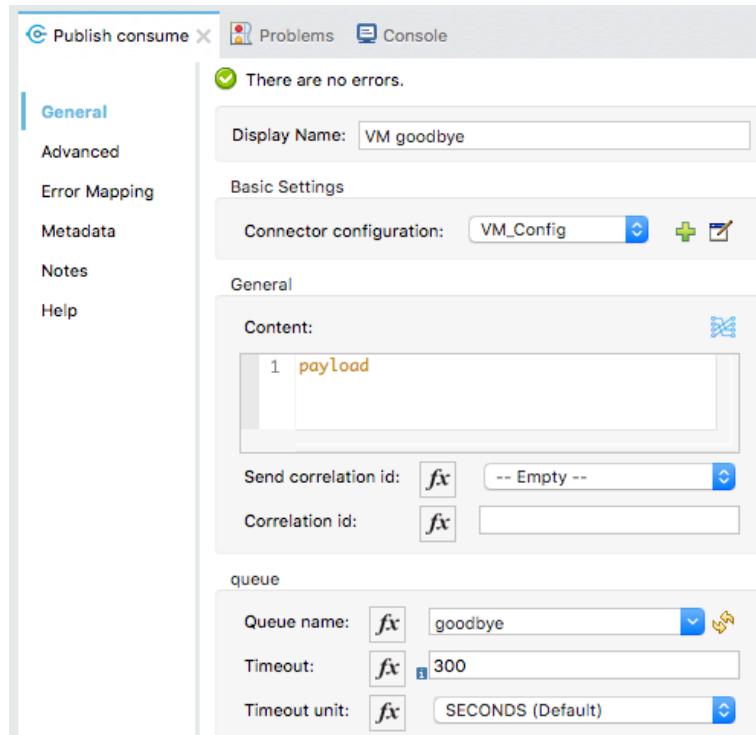
12. In the Global Element Properties dialog box, click OK.



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

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

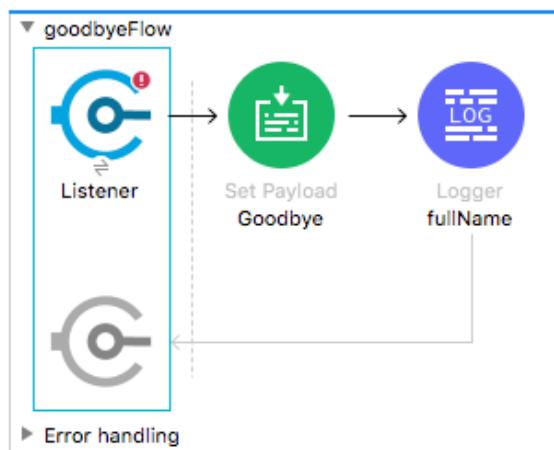
14. Set the timeout to 300 seconds for debugging purposes.



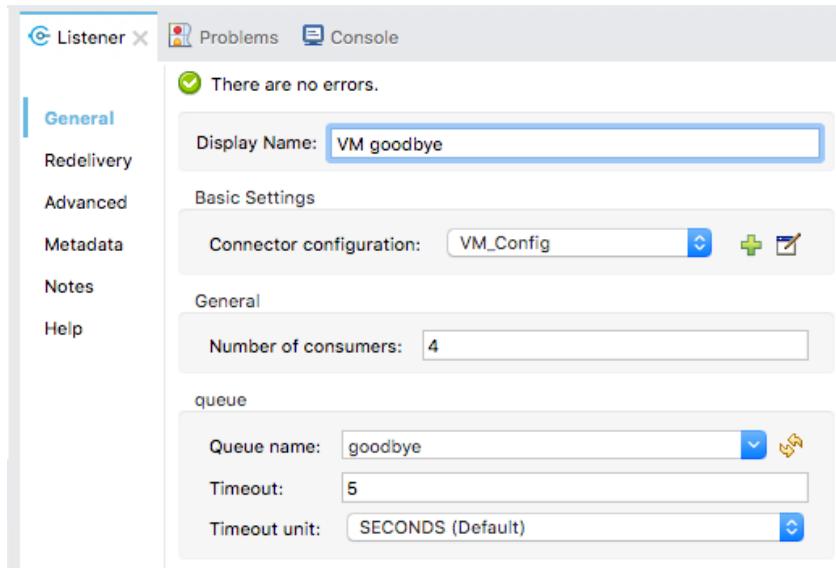
Add a VM Listener

15. Expand goodbyeFlow and delete its HTTP Listener.

16. Locate the Listener operation for the VM connector in the right side of the Mule Palette and drag and drop it in the source section of goodbyeFlow.

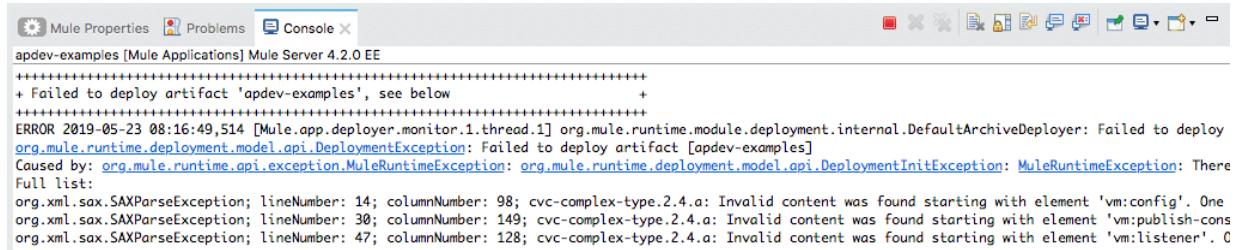


17. In the VM Listener properties view, change the display name to VM goodbye.
18. Ensure the connector configuration is set to the existing VM_Config and the queue name is set to goodbye.



Debug the application

19. Save the file to redeploy the project in debug mode; the deployment should fail with a number of errors including 3 invalid content errors.



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

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

Mule Debugger

```
▶ C Publish Consume = VM goodbye
▶ E attributes = {HttpRequestAttributes} org.mule.extension.http.api.HttpRequestAttrib
  @ correlationId = "c86364c0-7d55-11e9-9b95-f018983d9329"
  ▼ E payload = "Hello"
    @ ^mediaType = */*
  ▼ E vars = {Map} size = 1
    ▶ E firstName = "Maxwell"
```

apdev-examples

```
graph LR; L1((Listener GET /hello)) --> FR((Flow Reference subflow)); FR --> P((Publish consume VM goodbye)); P --> L2((Logger payload)); L2 --> L1; L3((Listener GET /goodbye))
```

Error handling

25. Step into goodbyeFlow.

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

Mule Debugger

```
▶ S Set Payload = Goodbye
▶ E attributes = {VMMessageAttributes} org.mule.extensions.vm.api.VMMessages
  @ correlationId = "c86364c0-7d55-11e9-9b95-f018983d9329"
  ▼ E payload = "Hello"
    @ ^mediaType = */*; charset=UTF-8
    @ vars = {Map} size = 0
```

apdev-examples

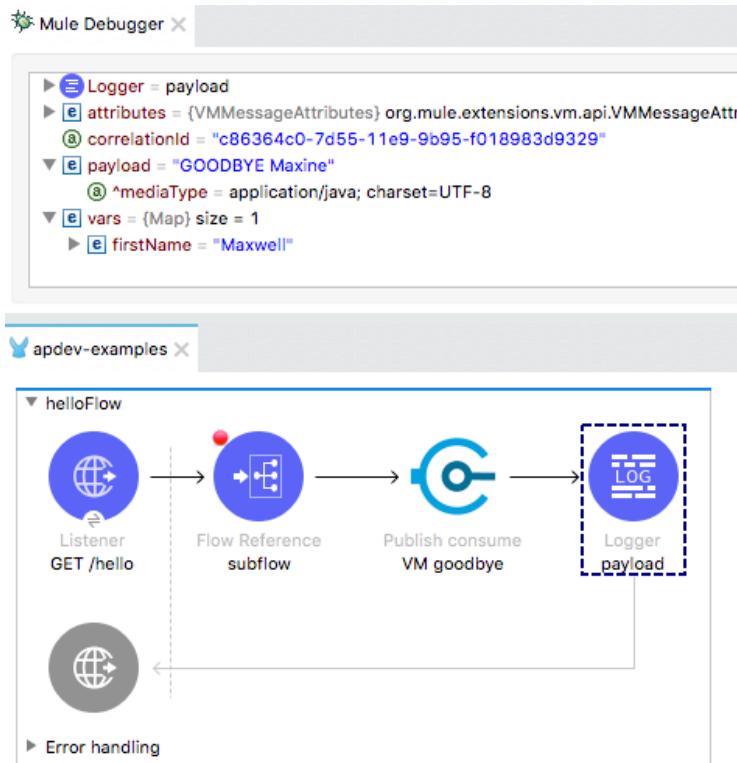
goodbyeFlow

```
graph LR; L1((Listener VM goodbye)) --> SP1((Set Payload Goodbye)); SP1 --> L2((Logger fullName)); L2 --> L1
```

Message Flow Global Elements Configuration XML

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

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



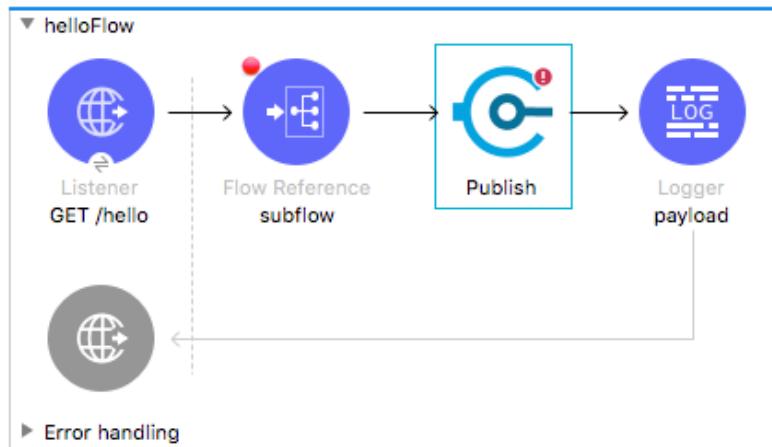
29. Step through the rest of the application.

30. Switch perspectives.

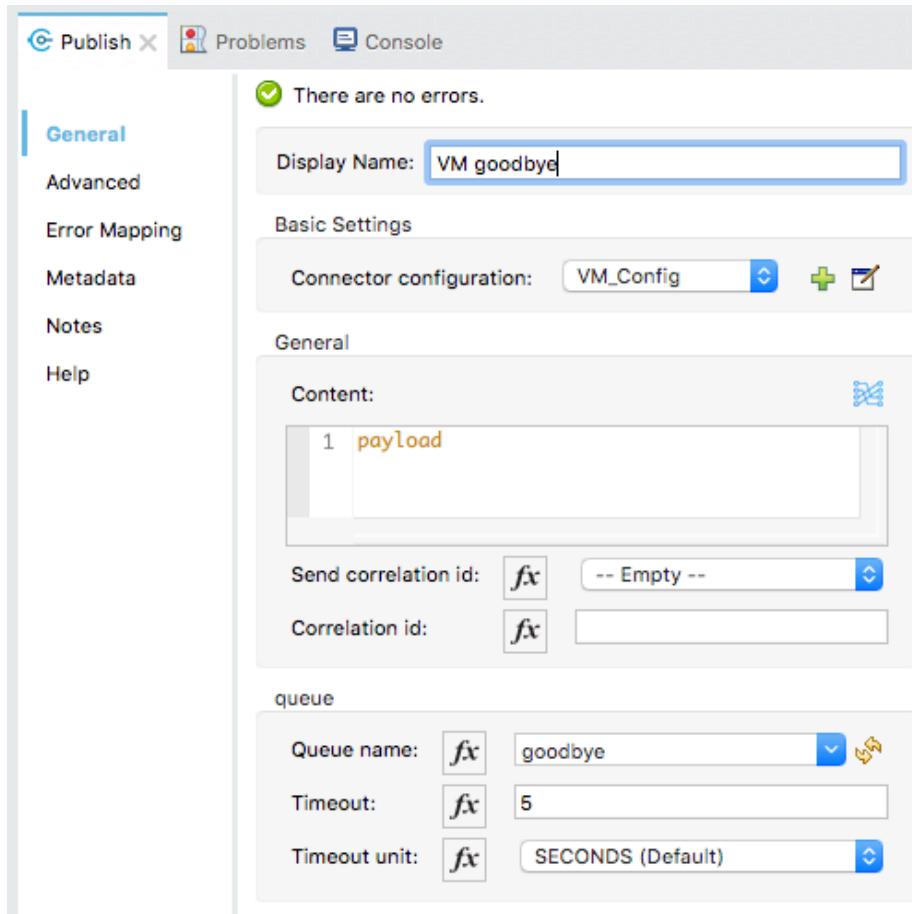
Change the VM Publish Consume operation to Publish

31. Delete the VM Publish consume operation in helloFlow.

32. Drag a VM Publish operation from the Mule Palette and drop it before the Logger.



33. In the Publish properties view, set the display name to VM goodbye.
34. Ensure the connector configuration is set to the existing VM_Config and the queue name is set to goodbye.



Debug the application

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

39. Look at the value of the payload.

The screenshot shows the Mule Debugger interface with two panes. The left pane displays a tree view of variables:

- Logger = payload
- attributes = {HttpRequestAttributes} org.mule.extension.http.api.HttpRequestAttributes
- correlationId = "3bd089a1-7d57-11e9-9b95-f018983d9329"
- payload = "Hello"**
- vars = {Map} size = 1
 - firstName = "Maxwell"

The right pane shows the value of the payload variable: "Hello".

The screenshot shows the Mule Studio flow editor with a flow named "helloFlow". The flow consists of the following components and connections:

- A Listener icon labeled "Listener GET /hello" is connected to a "Flow Reference subflow" icon.
- The "Flow Reference subflow" icon is connected to a "Publish VM goodbye" icon.
- The "Publish VM goodbye" icon is connected to a "Logger payload" icon, which is enclosed in a dashed box.
- An "Error handling" icon is connected to the "Listener" icon.

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

41. Step to the end of the application.

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

200 Success 160792.30 ms



Hello

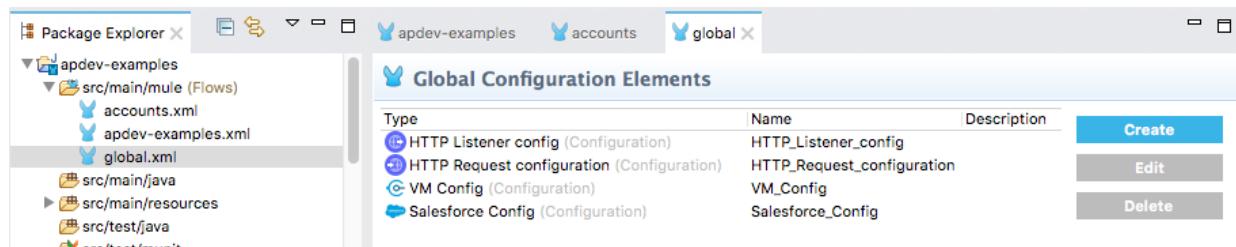
43. Return to Anypoint Studio and switch perspectives.

44. Stop the project.

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

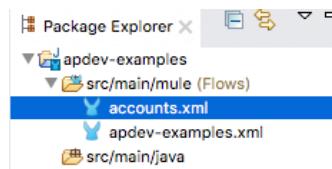
In this walkthrough, you refactor your apdev-examples project. You will:

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

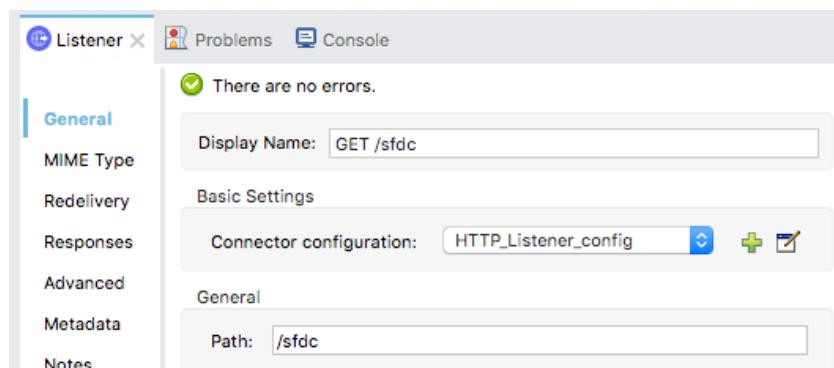


Create a new configuration file

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



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



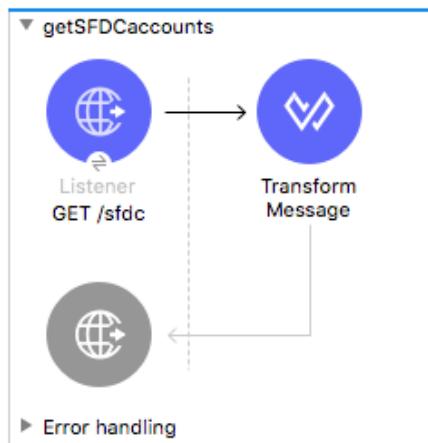
8. Add a Transform Message component to the flow.
9. In the Transform Message properties view, change the output type to application/json and set the expression to payload.

```

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

```

10. Change the name of the flow to getSFDCCaccounts.



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

Type	Name	Description	Create	Edit	Delete

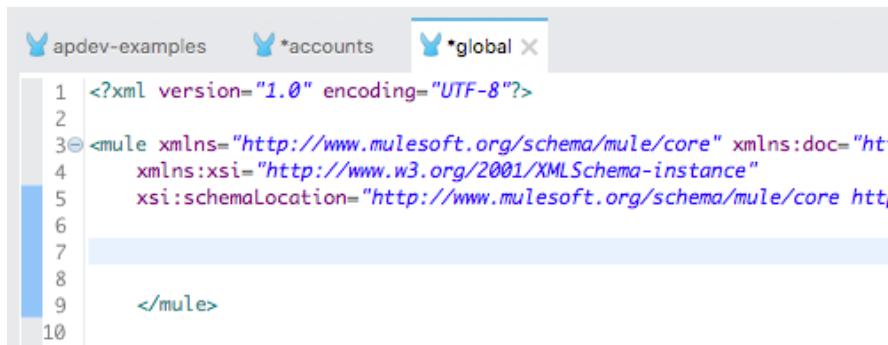
Create a global configuration file

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



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

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

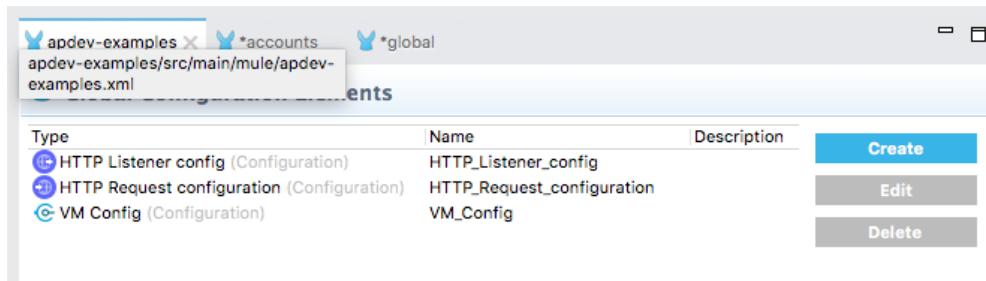


```
1 <?xml version="1.0" encoding="UTF-8"?>
2
3<mule xmlns="http://www.mulesoft.org/schema/mule/core" xmlns:doc="htt
4     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5     xsi:schemaLocation="http://www.mulesoft.org/schema/mule/core http://
6
7
8
9    </mule>
10
```

Move the existing global elements to the new global configuration file

15. Return to apdev-examples.xml.

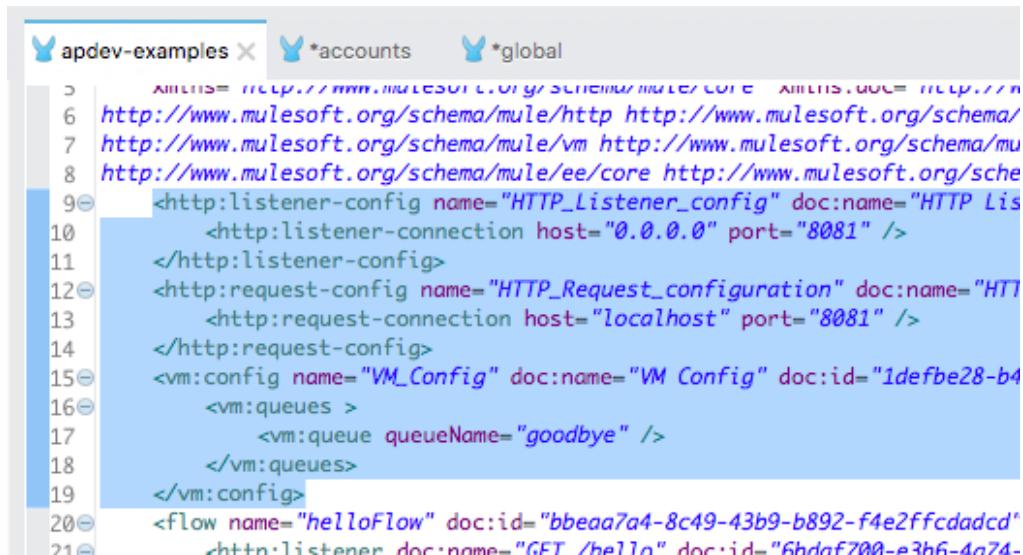
16. Switch to the Global Elements view and see there are three configurations.



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

17. Switch to the Configuration XML view.

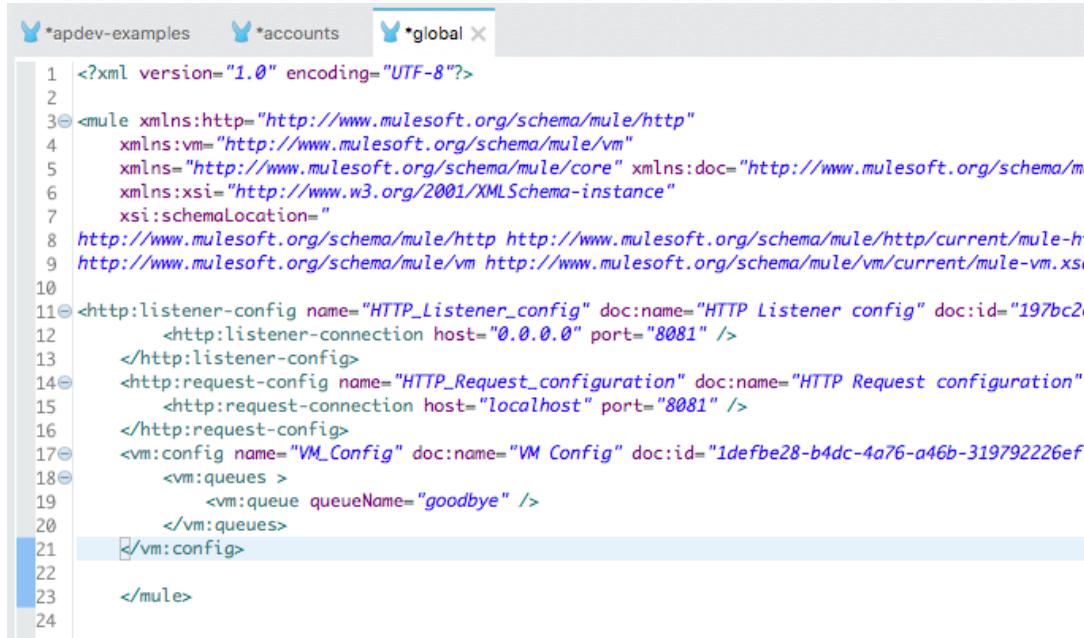
18. Select and cut the three configuration elements defined before the flows.



```
1 <?xml version="1.0" encoding="UTF-8"?>
2   xmlns="http://www.mulesoft.org/schema/mule/core" xmlns:doc="http://www.mulesoft.org/schema/
3   http://www.mulesoft.org/schema/mule/http http://www.mulesoft.org/schema/
4   http://www.mulesoft.org/schema/mule/vm http://www.mulesoft.org/schema/mu
5   http://www.mulesoft.org/schema/mule/ee/core http://www.mulesoft.org/sche
6
7
8
9<http:listener-config name="HTTP_Listener_config" doc:name="HTTP Lis
10    <http:listener-connection host="0.0.0.0" port="8081" />
11</http:listener-config>
12<http:request-config name="HTTP_Request_configuration" doc:name="HTT
13    <http:request-connection host="localhost" port="8081" />
14</http:request-config>
15<vm:config name="VM_Config" doc:name="VM Config" doc:id="1defbe28-b4
16    <vm:queues>
17      <vm:queue queueName="goodbye" />
18    </vm:queues>
19</vm:config>
20<flow name="helloFlow" doc:id="bbeaa7a4-8c49-43b9-b892-f4e2ffcdadcd"
21    <http:listener doc:name="HTTP /hello" doc:id="6hdnf700-e3hf-4n74-
```

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

19. Return to the Message Flow view.
20. Return to global.xml.
21. Paste the global elements you cut to the clipboard between the start and end mule tags.



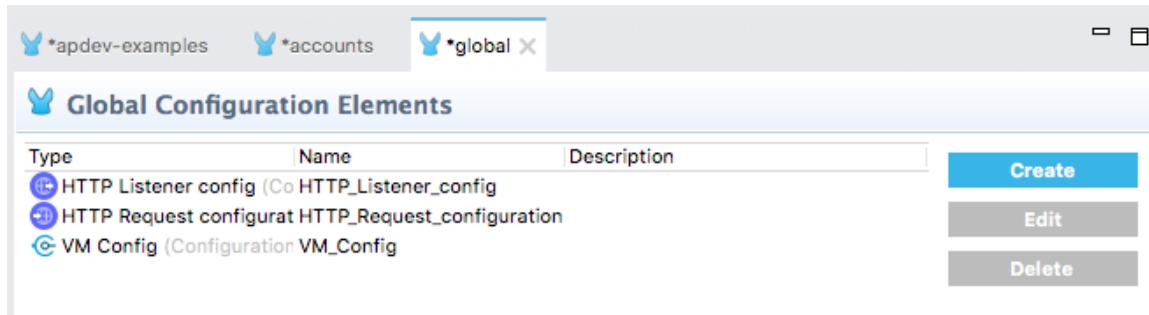
```

1 <?xml version="1.0" encoding="UTF-8"?>
2
3<mule xmlns:http="http://www.mulesoft.org/schema/mule/http"
4   xmlns:vm="http://www.mulesoft.org/schema/mule/vm"
5   xmlns="http://www.mulesoft.org/schema/mule/core" xmlns:doc="http://www.mulesoft.org/schema/mi
6   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
7   xsi:schemaLocation="
8     http://www.mulesoft.org/schema/mule/http http://www.mulesoft.org/schema/mule/http/current/mule-h
9     http://www.mulesoft.org/schema/mule/vm http://www.mulesoft.org/schema/mule/vm/current/mule-vm.xsc
10
11<http:listener-config name="HTTP_Listener_config" doc:name="HTTP Listener config" doc:id="197bc2e
12   <http:listener-connection host="0.0.0.0" port="8081" />
13</http:listener-config>
14<http:request-config name="HTTP_Request_configuration" doc:name="HTTP Request configuration"
15   <http:request-connection host="localhost" port="8081" />
16</http:request-config>
17<vm:config name="VM_Config" doc:name="VM Config" doc:id="1defbe28-b4dc-4a76-a46b-319792226ef.
18  <vm:queues>
19    <vm:queue queueName="goodbye" />
20  </vm:queues>
21</vm:config>
22
23
24</mule>

```

Note: If you are prompted to regenerate ID values, click Yes.

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

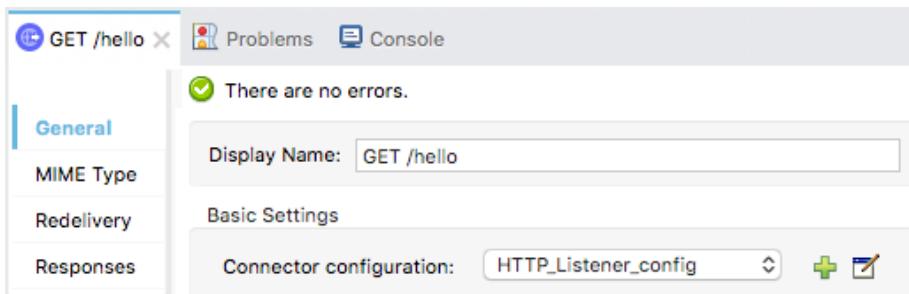


Type	Name	Description	
HTTP Listener config (Co HTTP_Listener_config)	HTTP_Listener_config		Create
HTTP Request configurat	HTTP_Request_configuration		Edit
VM Config (Configuration	VM_Config		Delete

Test the application

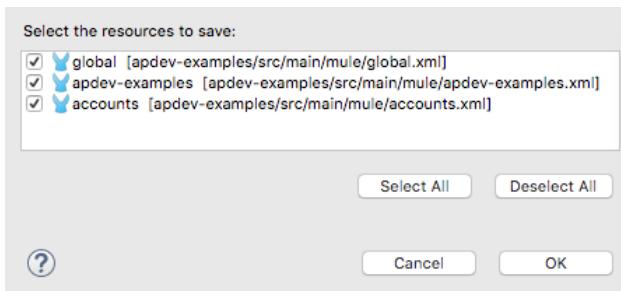
23. Return to apdev-examples.xml.

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

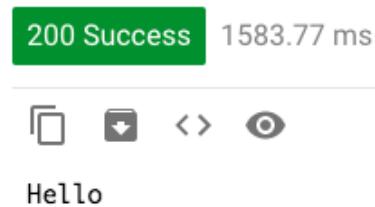


25. Run the project.

26. In the dialog box, ensure all the resources are selected and click OK.



27. In Advanced REST Client, send the same request; you should still get a response of Hello.

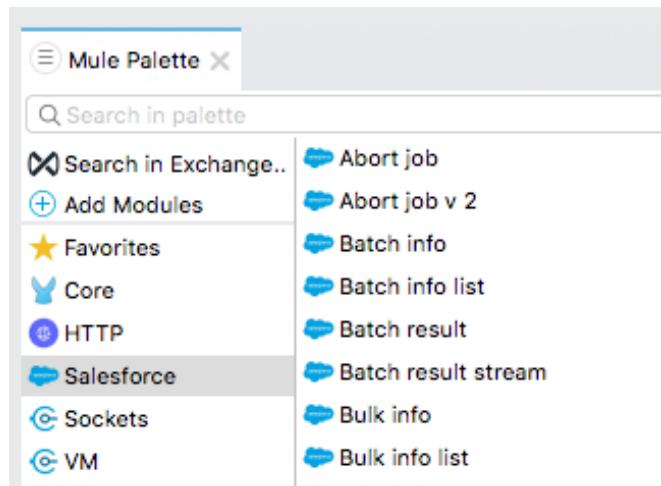


Create a new global element for the Salesforce component in global.xml

28. Return to apdev-examples.xml.

29. In the Mule Palette, select Add Modules.

30. Select the Salesforce connector in the right side of the Mule Palette and drag and drop it into the left side.

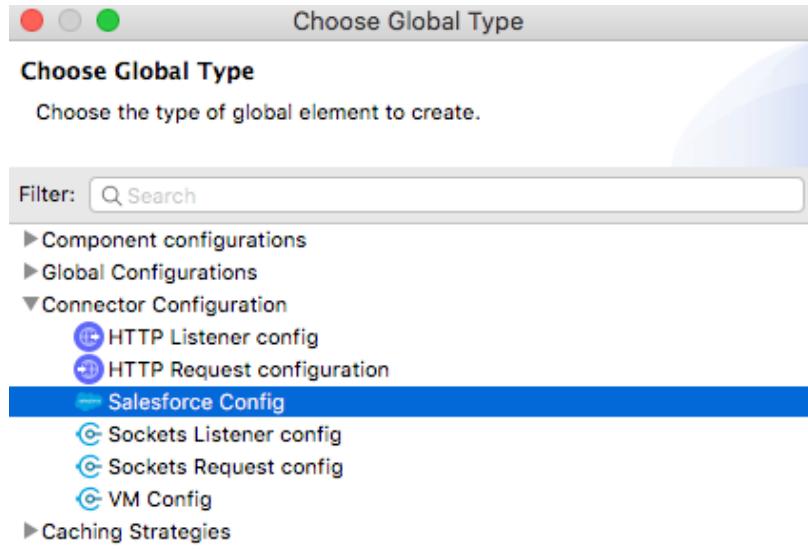


31. Locate the new set of Salesforce JAR files in the project.

32. Return to global.xml.

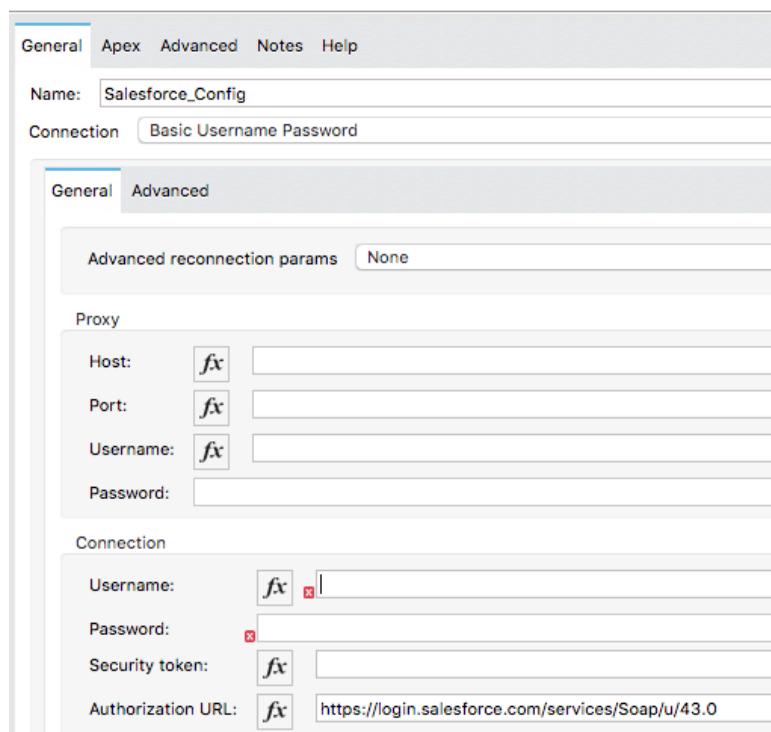
33. In the Global Elements view, click Create.

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

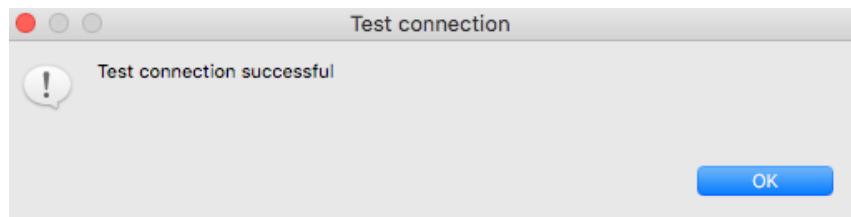


35. In the Global Element Properties dialog box locate the Connection section and enter your Salesforce username, password, and security token.

Salesforce Config



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

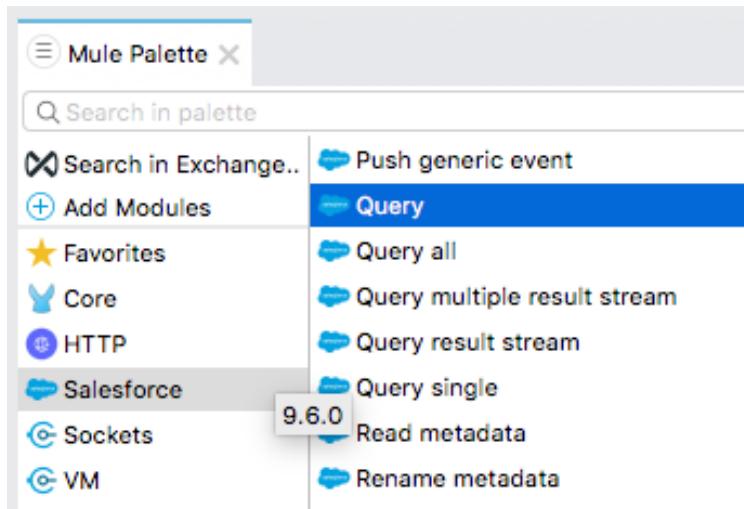


37. In the Test connection dialog box, click OK.
38. In the Global Element Properties dialog box, click OK; you should now see the new configuration in global.xml.

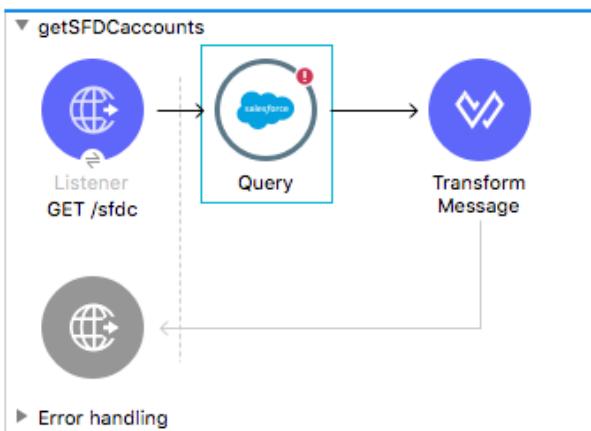
Type	Name	Description	Create	Edit	Delete
HTTP Listener config (Config)	HTTP_Listener_config				
HTTP Request configurat	HTTP_Request_configuration				
VM Config (Config)	VM_Config				
Salesforce Config (Config)	Salesforce_Config				

Add a new Salesforce operation that uses the global configuration

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

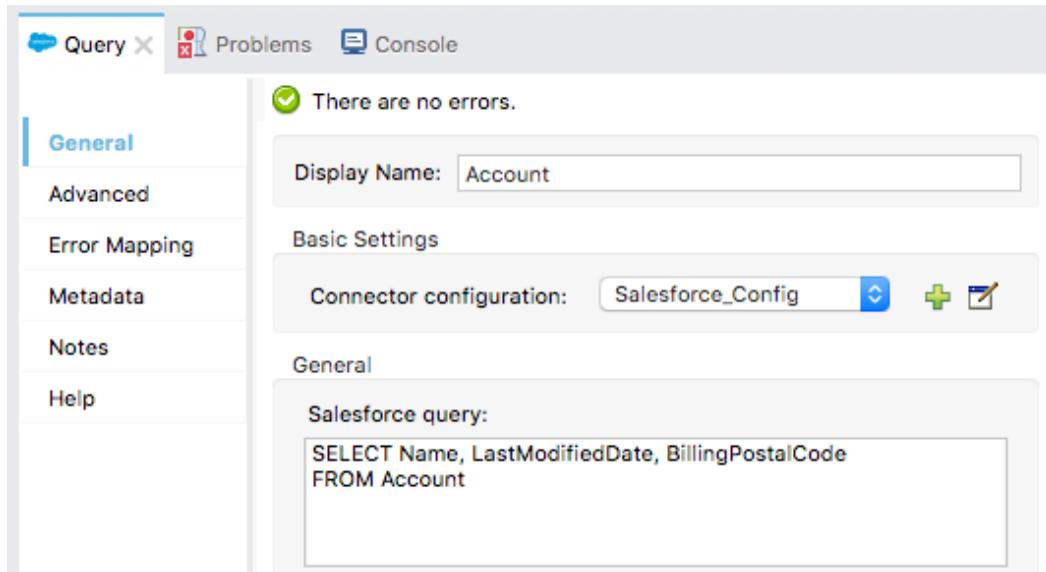


41. Locate the Query operation in the right side of the Mule Palette and drag and drop it before the Transform Message component in getSFDCCaccounts.



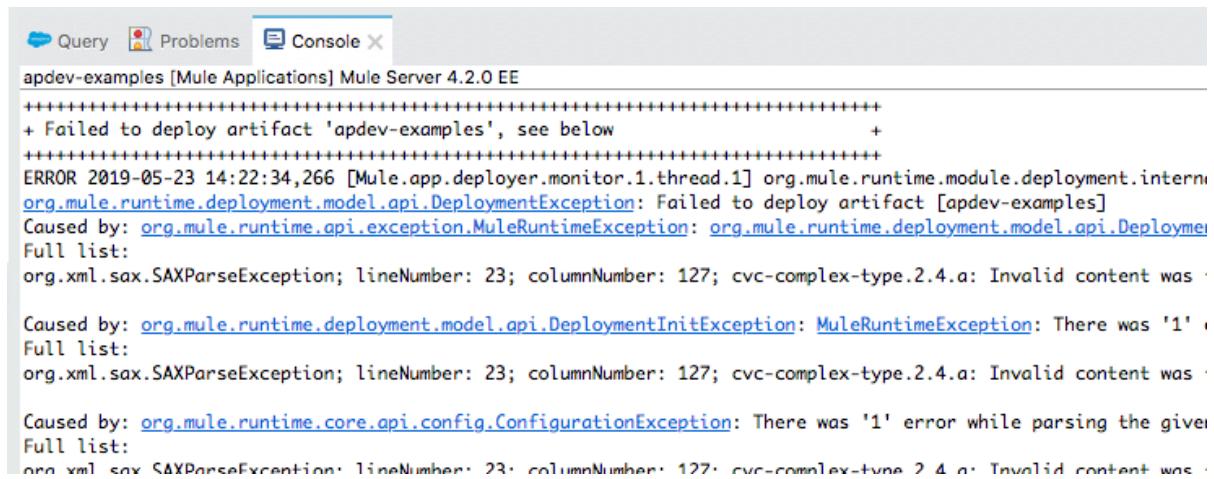
42. In the Query properties view, set the display name to Account.
43. Ensure the connector configuration is set to the existing Salesforce_Config.
44. Return to the course snippets.txt file and copy the Salesforce query.

45. Return to Anypoint Studio and paste the query in the Salesforce query section of the Query properties view.



Test the application

46. Save all the files; you should get a number of invalid content errors in the console.



47. Stop the project.

48. Run the project; the application should successfully deploy.

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

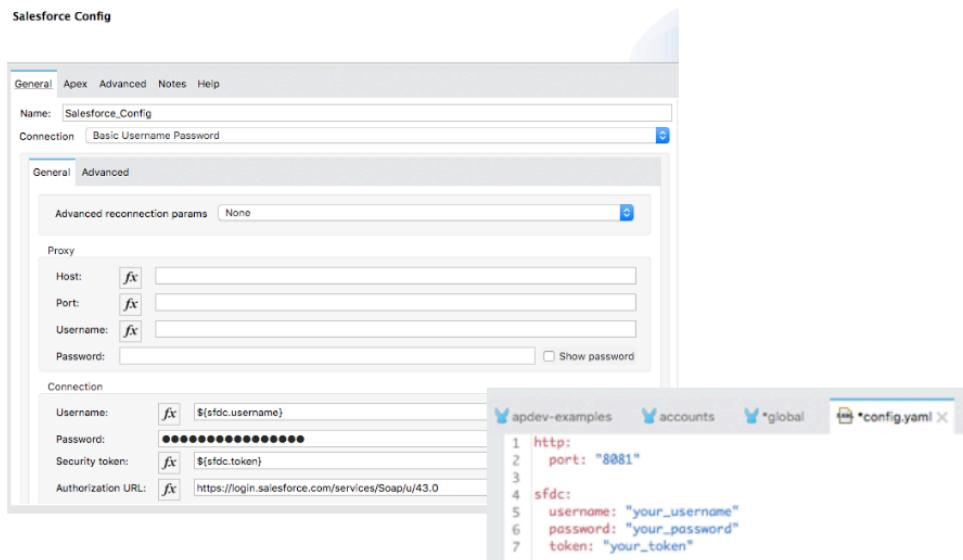
The screenshot shows the Advanced REST Client interface. At the top, there are fields for 'Method' (set to 'GET') and 'Request URL' (set to 'http://localhost:8081/sfdc'). To the right are buttons for 'SEND' and more options. Below this, a 'Parameters' dropdown is shown. The main area displays the response details: a green '200 OK' box indicating success, a timestamp of '2468.20 ms', and a 'DETAILS' button. The response body is a JSON array of three account objects:

```
[{"LastModifiedDate": "2019-01-11T01:10:38.000Z", "BillingPostalCode": null, "Id": null, "type": "Account", "Name": "GenePoint"}, {"LastModifiedDate": "2019-01-11T01:10:38.000Z", "BillingPostalCode": null, "Id": null, "type": "Account", "Name": "United Oil & Gas, UK"}, {"LastModifiedDate": "2019-01-11T01:10:38.000Z", "BillingPostalCode": null, "Id": null, "type": "Account", "Name": "United Oil & Gas. Sindaore"}
```

Walkthrough 7-4: Use property placeholders in connectors

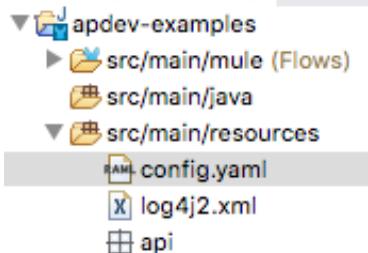
In this walkthrough, you introduce properties into your API implementation. You will:

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



Create a properties file

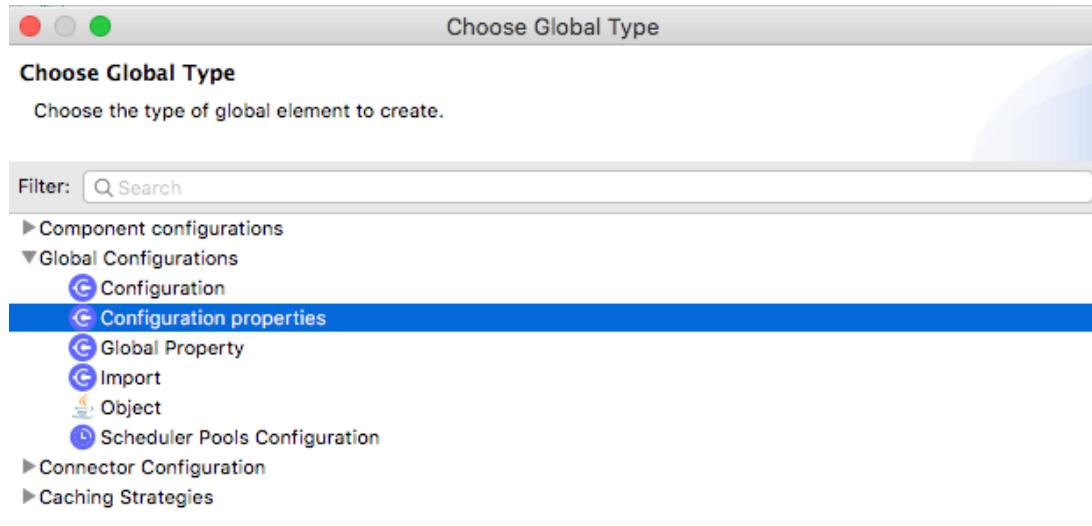
1. Return to the apdev-examples project.
2. Right-click the src/main/resources folder in the Package Explorer and select New > File.
3. Set the file name to config.yaml and click Finish.



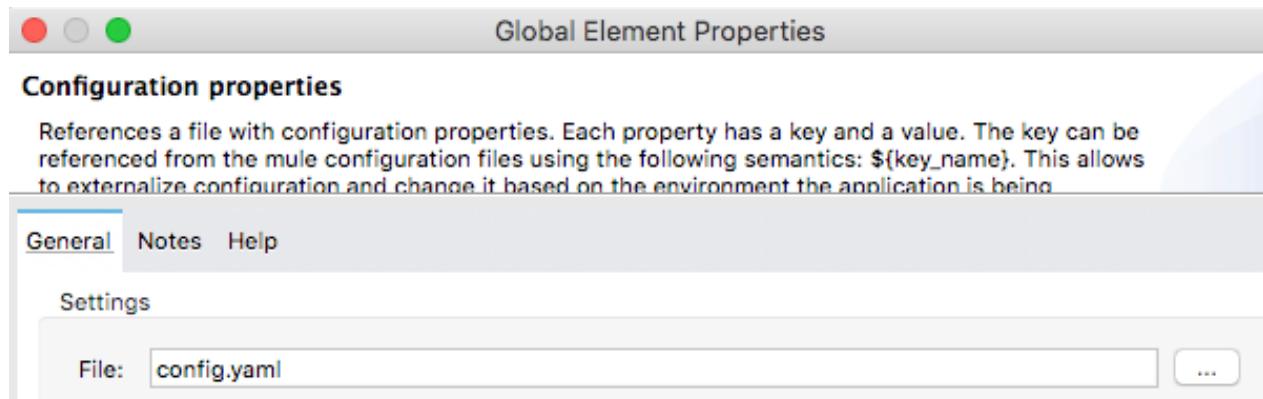
Create a Configuration properties global element

4. Return to global.xml.
5. In the Global Elements view, click Create.

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



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



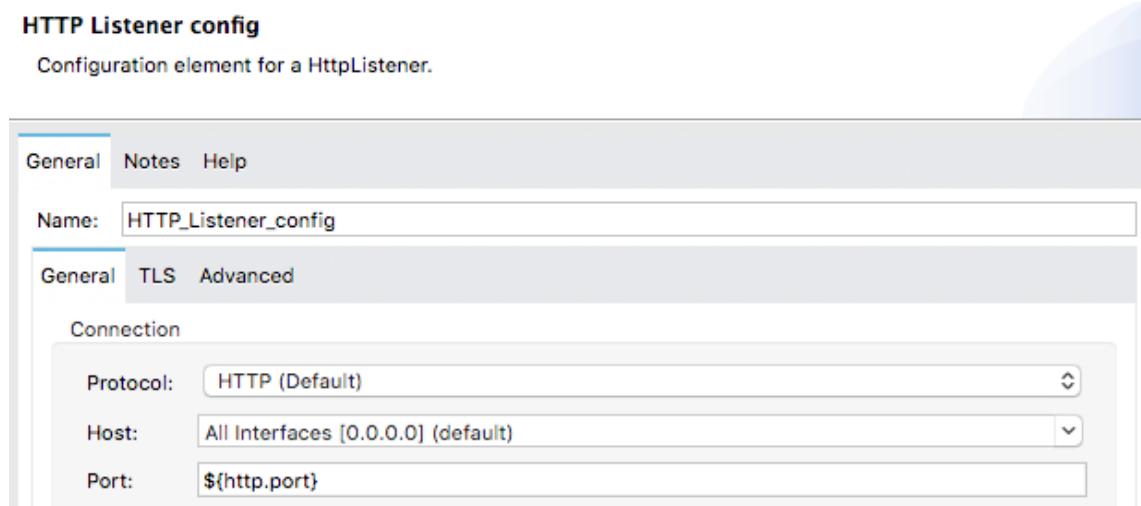
Parameterize the HTTP Listener port

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

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

- Save the file.
- Return to global.xml.
- Double-click the HTTP Listener config global element.

13. Change the port from 8081 to the application property, \${http.port}.



14. Click OK.

Parameterize the Salesforce credentials

15. Double-click the Salesforce Config global element.

16. Copy the username value and click OK.

17. Return to config.yaml .

18. Define a property called sfdc.username and set it to your Salesforce username.

19. Add properties for password and token and set them to your values.

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

20. Save the file.

21. Return to global.xml.

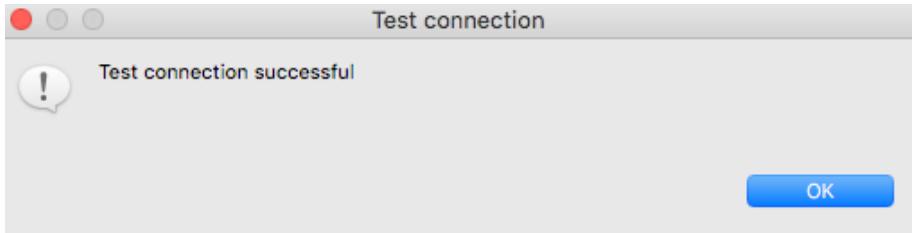
22. Double-click the Salesforce Config global element.

23. Change the values to use the application properties.

The screenshot shows the 'Connection' dialog box with the following fields:

- Username: \${sfdc.username}
- Password: \${sfdc.password} (with 'Show password' checked)
- Security token: \${sfdc.token}
- Authorization URL: https://login.salesforce.com/services/Soap/u/43.0

24. Click Test Connection and make sure it succeeds.



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

25. Click OK.

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

Test the application

27. Save all files to redeploy the application.

28. In Advanced REST Client, make the same request to <http://localhost:8081/sfdc> and confirm you still get data.

The screenshot shows the Advanced REST Client interface with the following details:

- Method: GET
- Request URL: http://localhost:8081/sfdc
- Parameters: None
- Status: 200 OK (2405.00 ms)
- Response:

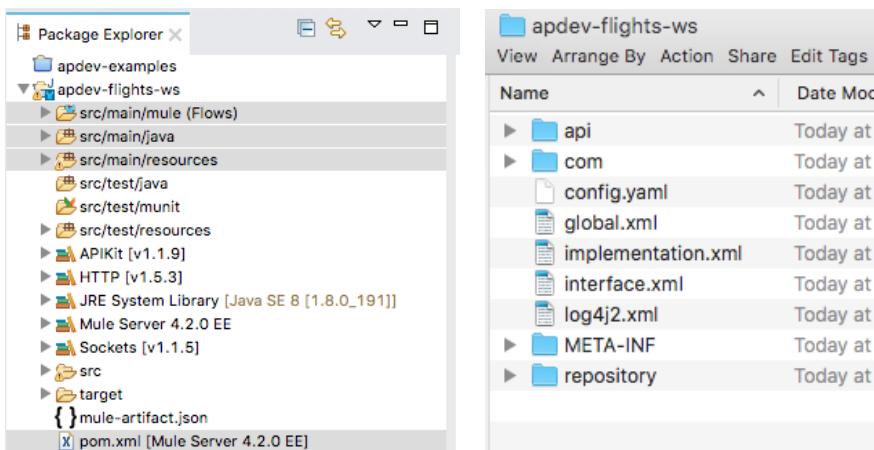
```
[{"LastModifiedDate": "2019-01-11T01:10:38.000Z", "BillingPostalCode": null, "Id": null, "type": "Account", "Name": "GenePoint"}, {"LastModifiedDate": "2019-01-11T01:10:38.000Z", "BillingPostalCode": null, "Id": null, "type": "Account", "Name": "GenePoint"}]
```

29. Return to Anypoint Studio and stop the project.

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

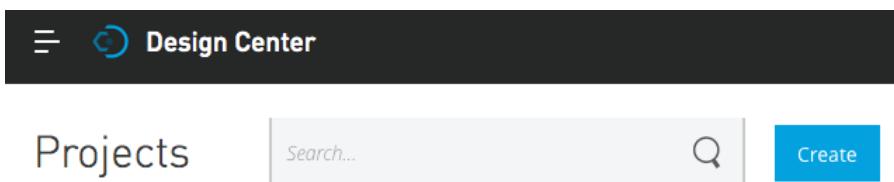
In this walkthrough, you create a new project for the Mule United Airlines (MUA) flights application that you will build during the course and then review and organize its files and folders. You will:

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



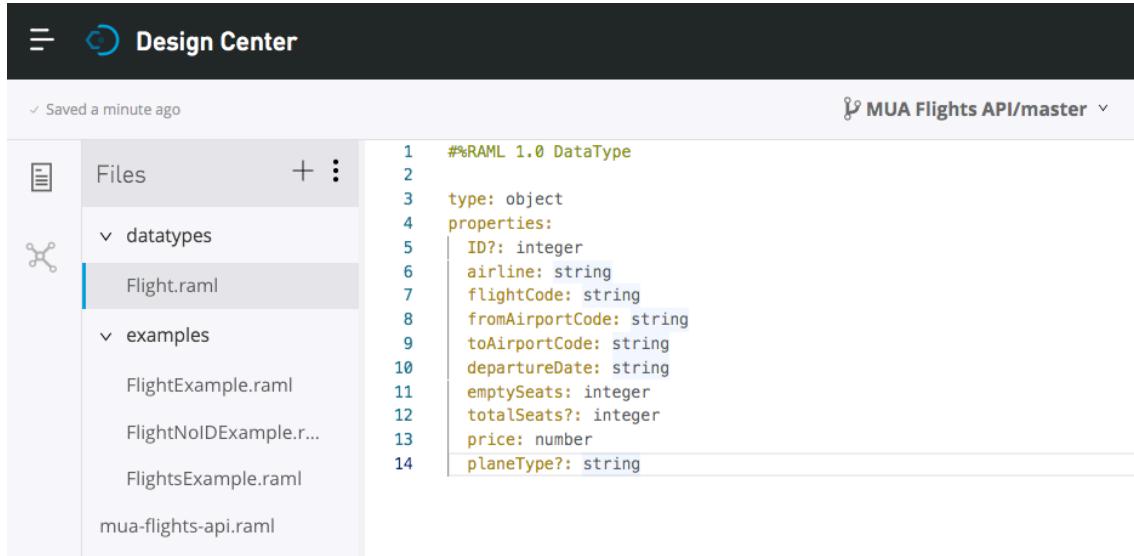
Create a new API in Design Center

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



3. In API Designer, click the options menu in the file browser and select Import.
4. In the Import dialog box, browse to your student files and select the MUA Flights API.zip located in the resources folder.
5. Click Import.
6. In the Replace dialog box, select mua-flights-api.raml and click Replace file.

7. Explore the API; be sure to look at what resources are defined, the name of the query parameter, and the structure of the Flight data type.



The screenshot shows the Mule Design Center interface. The left sidebar lists files: Flight.raml (selected), FlightExample.raml, FlightNoIDExample.raml, FlightsExample.raml, and mua-flights-api.raml. The right pane displays the RAML code for Flight.raml:

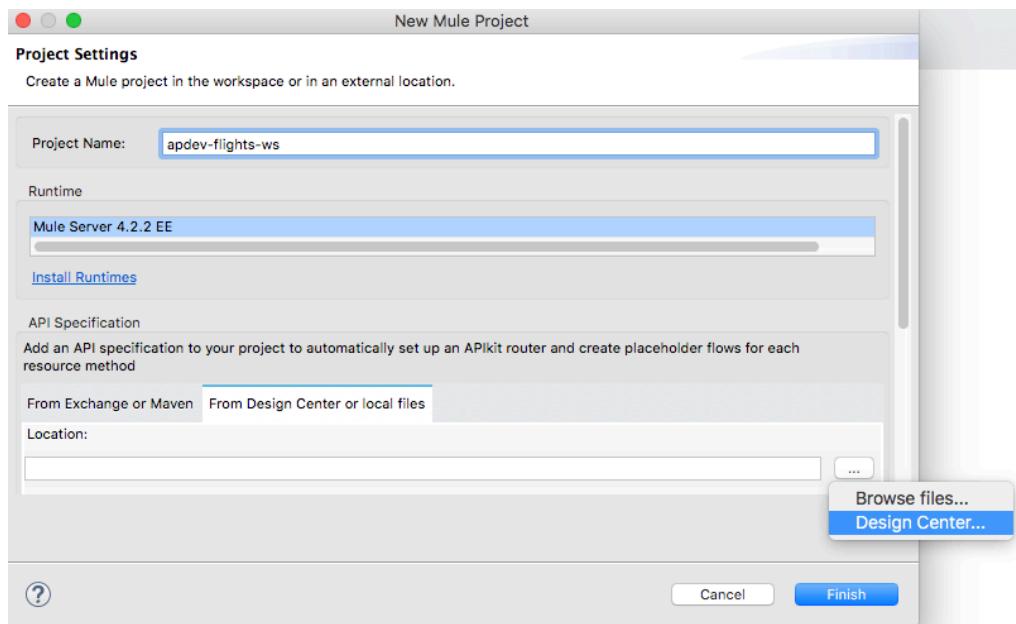
```

1  #%RAML 1.0 DataType
2
3  type: object
4  properties:
5    ID?: integer
6    airline: string
7    flightCode: string
8    fromAirportCode: string
9    toAirportCode: string
10   departureDate: string
11   emptySeats: integer
12   totalSeats?: integer
13   price: number
14   planeType?: string

```

Create a new project to implement this API in Anypoint Studio

8. Return to Anypoint Studio.
9. Right-click in the Package Explorer and select New > Mule Project.
10. In the New Mule Project dialog box, set the project name to apdev-flights-ws.
11. Select the From Design Center or local files tab.
12. Click the browse button next to Location and select Design Center.



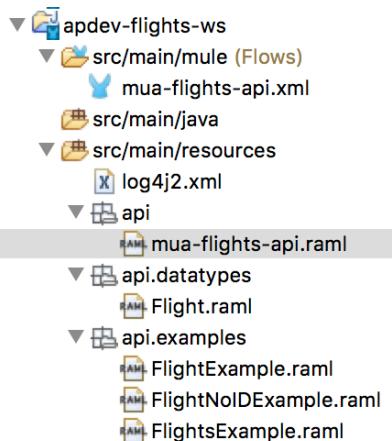
13. Sign into Anypoint Platform if prompted, then, in the Browse Design Center for APIs dialog box, select MUA Flights API and click OK.

Project name	Branch
American Flights API	master
MUA Flights API	master

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

Locate the new RAML files in Anypoint Studio

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

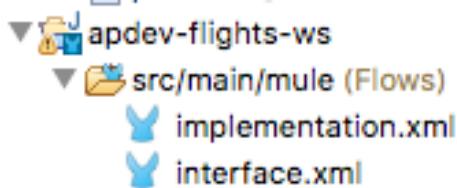


17. Open mua-flights-api.raml and review the file.
18. Leave the file open.

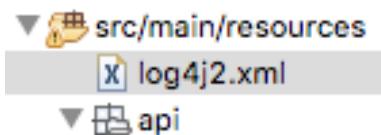
Review project configuration files

19. Look at the mua-flights-api.xml file that was created; it should have a get:\flights flow and a post:\flights flow.
20. Rename mua-flights-api.xml to interface.xml.

21. Create a new Mule configuration file called implementation.xml.

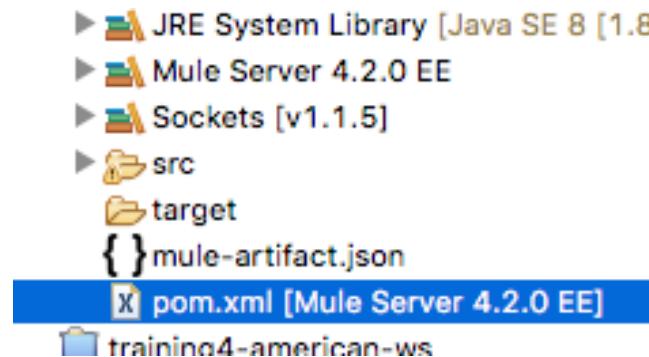


22. Locate log4j2.xml in src/main/resources and open it.



23. Review its contents and then close the file.

24. Locate pom.xml in the project and open it.



25. Review its contents.

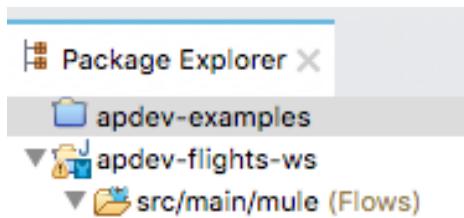
26. Return to the apdev-examples project and open its pom.xml file.

27. Compare the two pom.xml files.

```
33<dependencies>
34    <dependency>
35        <groupId>org.mule.connectors</groupId>
36        <artifactId>mule-http-connector</artifactId>
37        <version>1.5.3</version>
38        <classifier>mule-plugin</classifier>
39    </dependency>
40    <dependency>
41        <groupId>org.mule.connectors</groupId>
42        <artifactId>mule-sockets-connector</artifactId>
43        <version>1.1.5</version>
44        <classifier>mule-plugin</classifier>
45    </dependency>
46
47<dependency>
48    <groupId>org.mule.connectors</groupId>
49    <artifactId>mule-vm-connector</artifactId>
50    <version>2.0.0</version>
51    <classifier>mule-plugin</classifier>
52</dependency>
53<dependency>
54    <groupId>com.mulesoft.connectors</groupId>
55    <artifactId>mule-salesforce-connector</artifactId>
56    <version>9.6.0</version>
57    <classifier>mule-plugin</classifier>
58</dependency>
59</dependencies>
```

28. Close both pom.xml files.

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

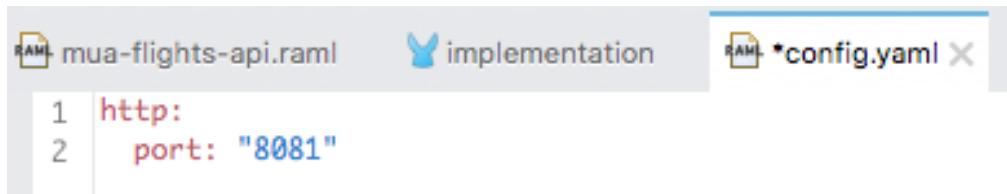


Create a properties file for application properties

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

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

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



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

33. Save and close the file.

Create a global configuration file

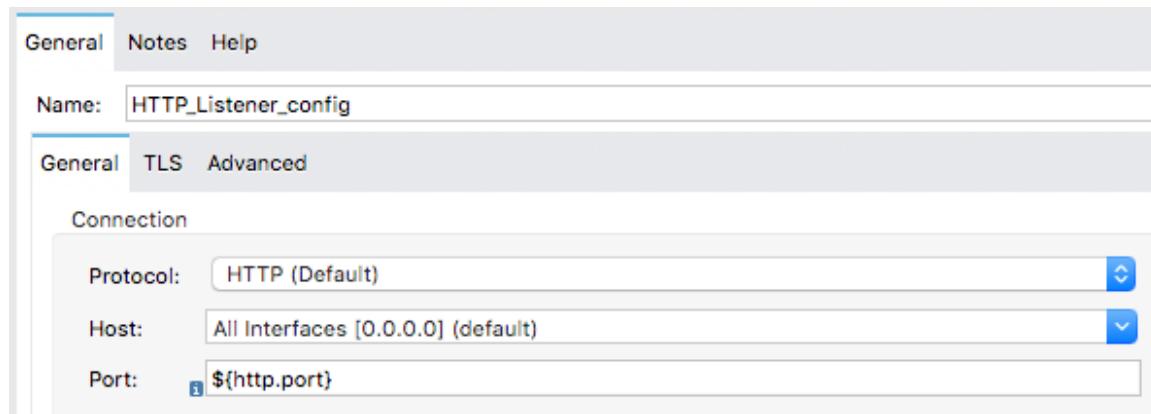
34. In src/main/mule, create a new Mule configuration file called global.xml.

35. In global.xml, switch to the Global Elements view.

36. Create a new Configuration properties element with the file set to config.yaml.

Global Configuration Elements			
Type	Name	Description	
Configuration properties (Configuration)	Configuration properties		Create
			Edit
			Delete

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



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

Type	Name	Description	Create	Edit	Delete
Configuration properties (Configuration)	Configuration properties				
HTTP Listener config (Configuration)	HTTP_Listener_config				

39. Save and close the file.

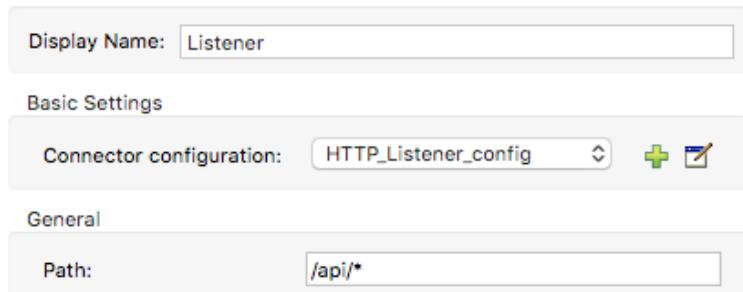
40. Go to the Global Elements view in interface.xml.

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

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

42. Return to the Message Flow view.

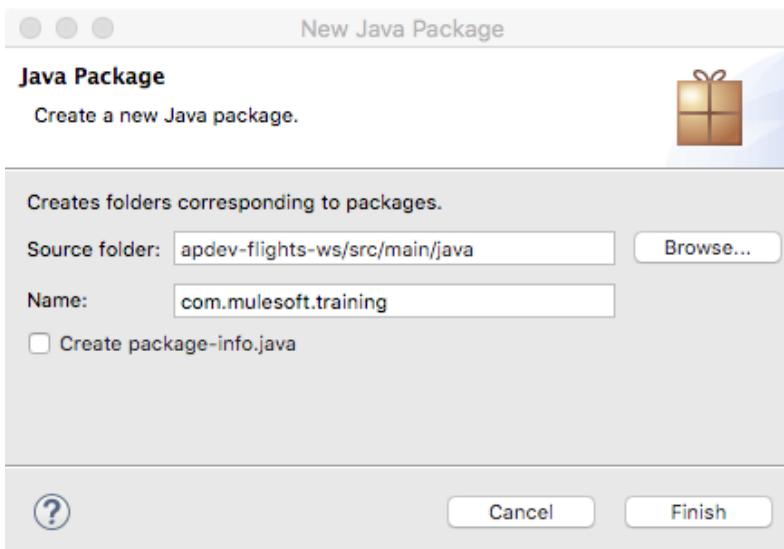
43. Select the HTTP Listener in mua-flights-api-main, and in the Listener properties view, ensure the Connector configuration is set to HTTP_Listener_config.



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

Add Java files to src/main/java

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



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

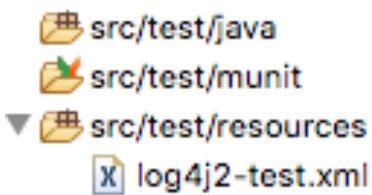
51. Review the code.



Review src/test folders

52. In the project, locate the three src/test folders.

53. Expand the src/test/resources folder.



Add test resources

54. In your computer's file browser, expand the resources/examples folder in the student files.

55. Select the three flights and one united-flights files (JSON and XML) and drag them into the src/test/resources folder in the apdev-flights-ws project.

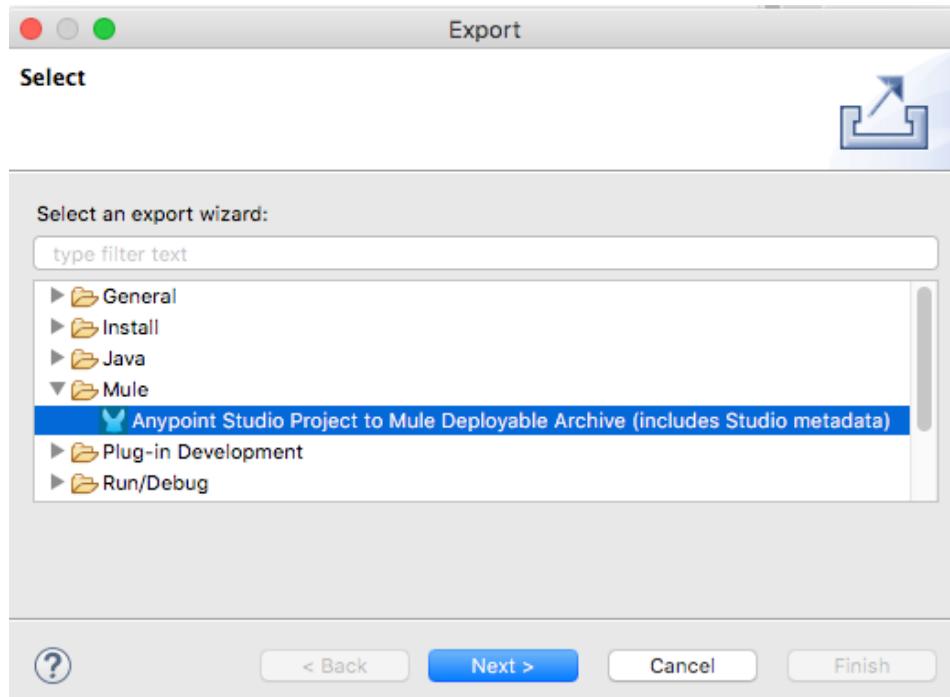


56. In the File Operation dialog box, ensure Copy files is selected and click OK.

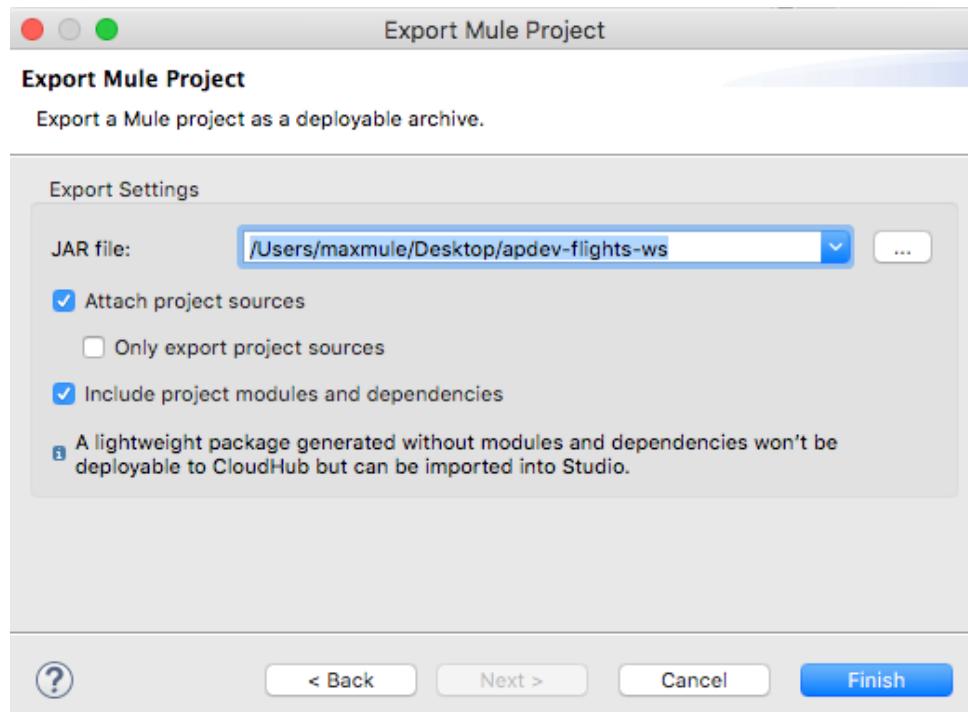
Examine the contents of a deployable archive

57. In Anypoint Studio, save all files, and then right-click the project and select Export.

58. In the Export dialog box, select Mule > Anypoint Studio Project to Mule Deployable Archive and click Next.

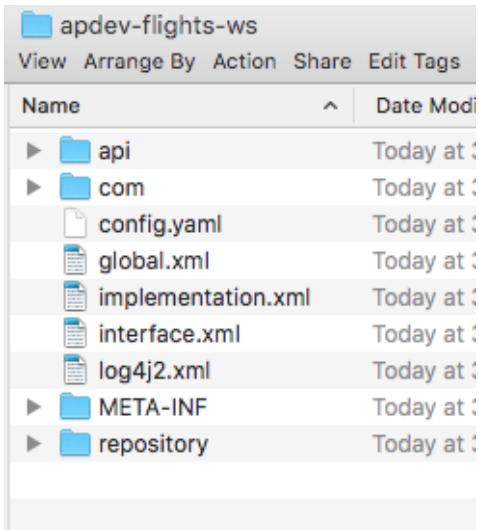


59. In the Export Mule Project dialog box, set the JAR file to a location that you can find and click Finish.



60. In the Export Mule Project dialog box, click OK.

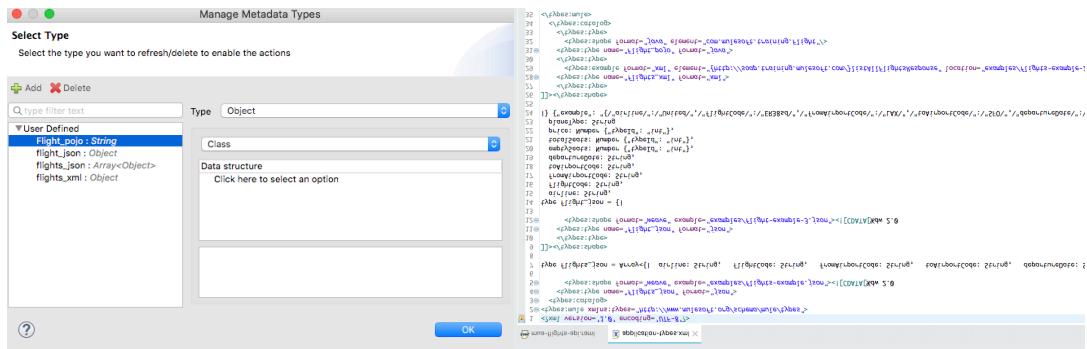
61. In your computer's file browser, locate the JAR file and expand it.
62. Open the resulting apdev-flights-ws folder and examine the contents.



Walkthrough 7-6: Manage metadata for a project

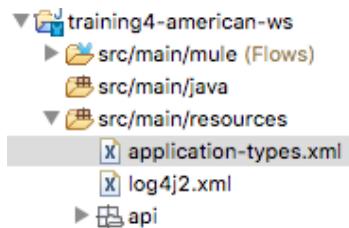
In this walkthrough, you define metadata for the apdev-flights-ws project. You will:

- Review existing metadata for the training4-american-ws project.
- Define new metadata to be used in transformations in the new apdev-flights-ws project.
- Manage metadata.



Locate existing metadata in the training4-american-ws project

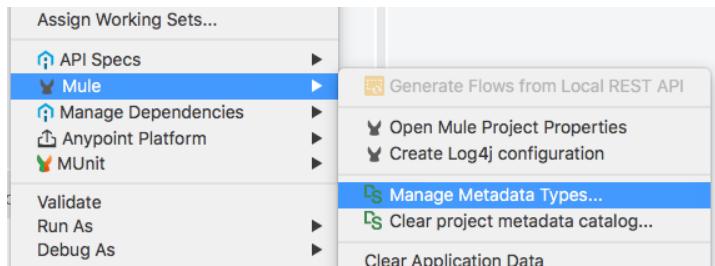
1. Open the training4-american-ws project.
2. Locate application-types.xml in src/main/resources.



3. Open the file and review the code.

```
<?xml version='1.0' encoding='UTF-8'?>
<types:mule xmlns:types="http://www.mulesoft.org/schema/mule/types">
    <types:catalog>
        <types:type name="american_flights_json" format="json">
            <types:shape format="weave" example="examples/american-flights-example.json"><![CDATA[
                type american_flights_json = Array<{ ID: Number {"typeId": "int"}, code: String, p
            ]]></types:shape>
            <types:type>
            </types:catalog>
            <types:enrichment select="#1f5c4b76-0525-4604-b99c-a107ae677a9d">
                <types:processor-declaration>
                    <types:output-event>
                        <types:message>
                            <types:payload type="american_flights_json"/>
                        </types:message>
                    </types:output-event>
                </types:processor-declaration>
            </types:enrichment>
        </types:type>
    </types:catalog>
</types:mule>
```

- Right-click the training4-american-ws project in the Project Explorer and review the options under Mule; you should see two for metadata.



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

Select Type

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

The screenshot shows the 'Manage Metadata Types' dialog box. On the left, there's a list of types under 'User Defined': 'american_flights_json : Array<Object>'. On the right, there's a panel with tabs for 'Type' (selected) and 'JSON'. Under 'Type', there's an 'Example' dropdown set to 'examples/american-flights' and a preview area showing a JSON schema with fields: ID : Number, code : String, price : Number, departureDate : String. Below the schema, there are 'Add' and 'Delete' buttons.

- Click OK.
- Close the project.

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

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

```

responses:
  200:
    body:
      application/json:
        type: Flight[]
        examples:
          output: !include examples/FlightsExample.raml

  post:
    body:
      application/json:
        type: Flight
        examples:
          input: !include examples/FlightNoIDExample.raml

```

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

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

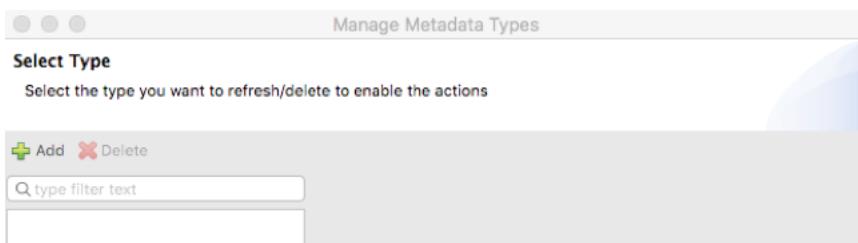
12. Locate the file you added to the src/main/java folder.

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

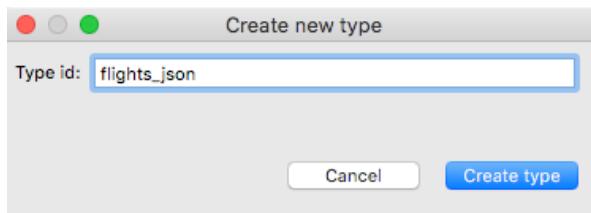
Create flights_json metadata type

13. Right-click the project and select Mule > Manage Metadata Types.

14. In the Manage Metadata Types dialog box, click the Add button.



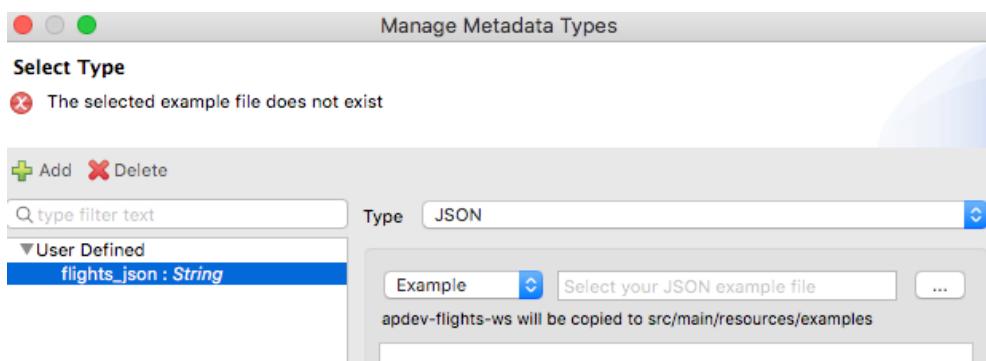
15. In the Create new type dialog box, set the type id to flights_json.



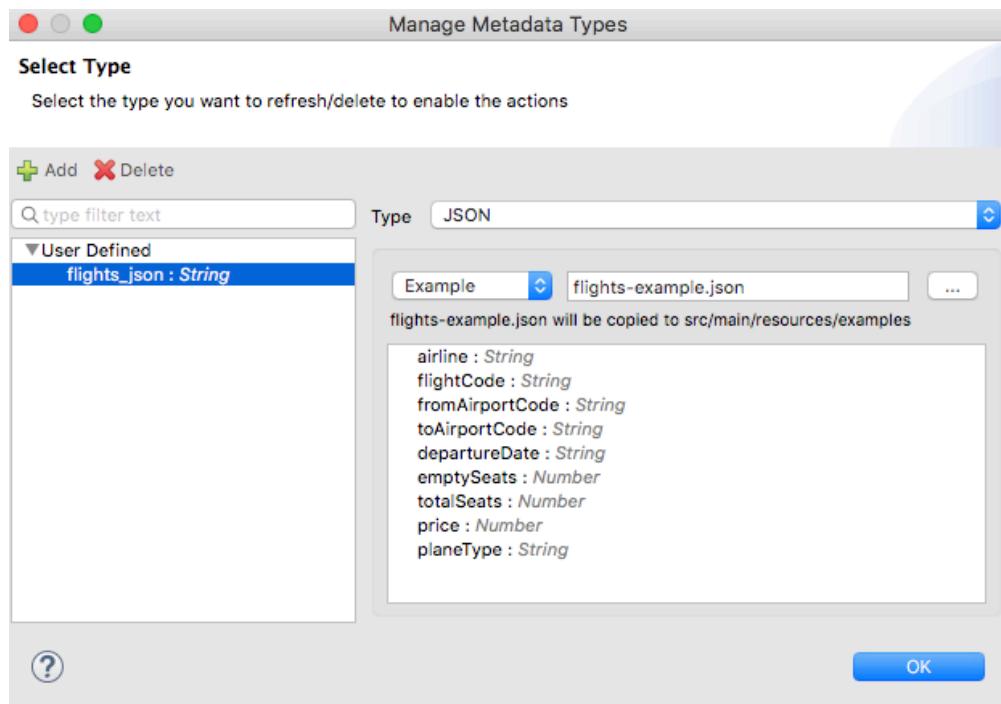
16. Click Create type.

17. In the Manage Metadata Types dialog box, set the first type to JSON.

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

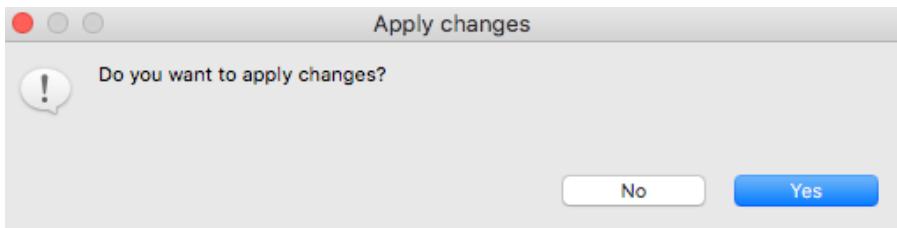


19. Click the browse button and select the flights-example.json file in src/test/resources.



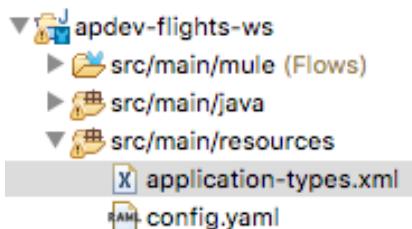
20. In the Manage Metadata Types dialog box, click OK.

21. In the Apply changes dialog box, click Yes.

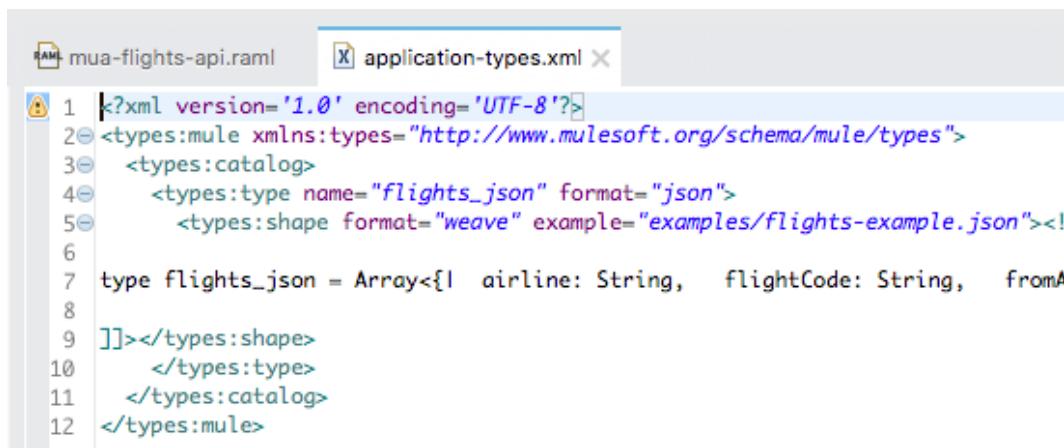


Locate the new metadata definition file

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



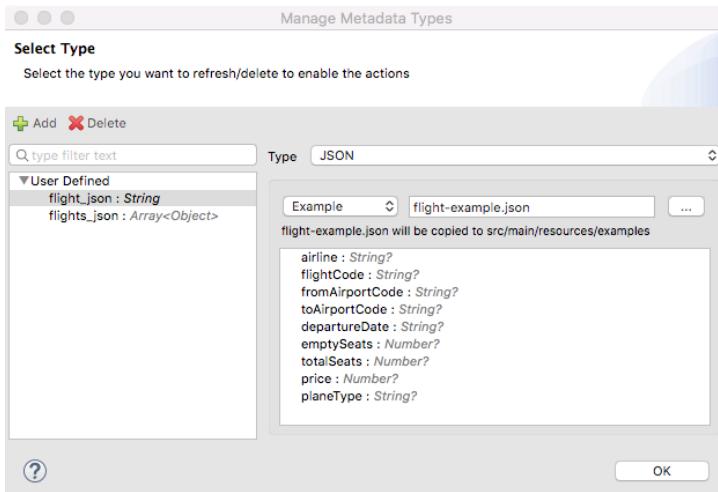
23. Open the file and review the code.



```
<?xml version='1.0' encoding='UTF-8'?>
<types:mule xmlns:types="http://www.mulesoft.org/schema/mule/types">
  <types:catalog>
    <types:type name="flights_json" format="json">
      <types:shape format="weave" example="examples/flights-example.json"><!
      6
      7 type flights_json = Array<{> airline: String, flightCode: String, fromA
      8
      9 }]></types:shape>
     10 </types:type>
     11 </types:catalog>
   12 </types:mule>
```

Create flight_json metadata type

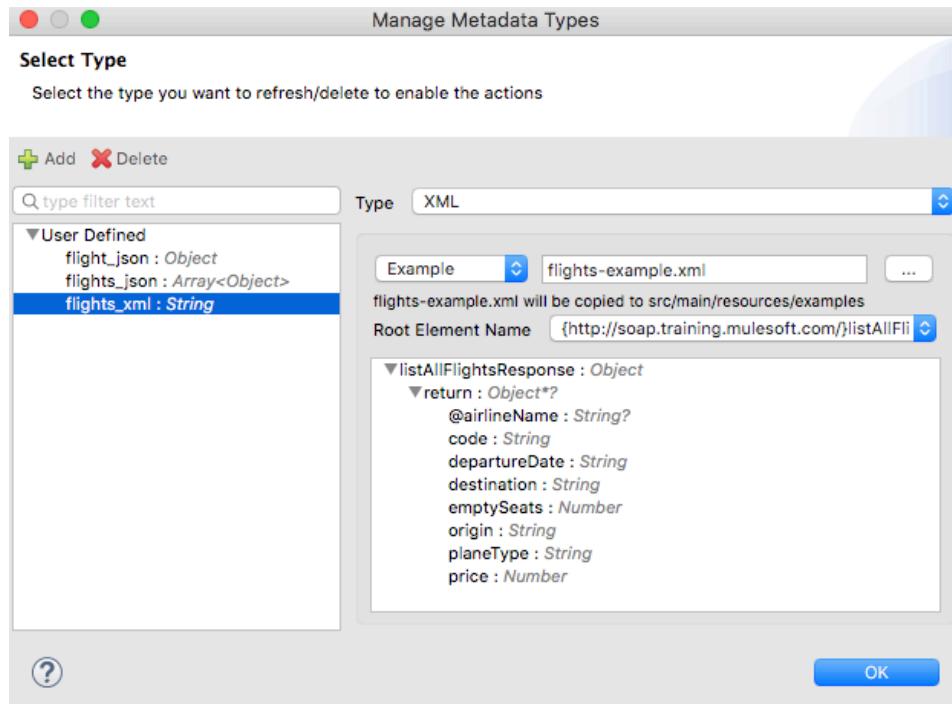
24. Right-click the project and select Mule > Manage Metadata Types.
25. In the Manage Metadata Types dialog box, click Add.
26. In the Create new type dialog box, set the type id to flight_json and click Create type.
27. In the Manage Metadata Types dialog box, set the first type to JSON.
28. Select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



Create flights_xml metadata type

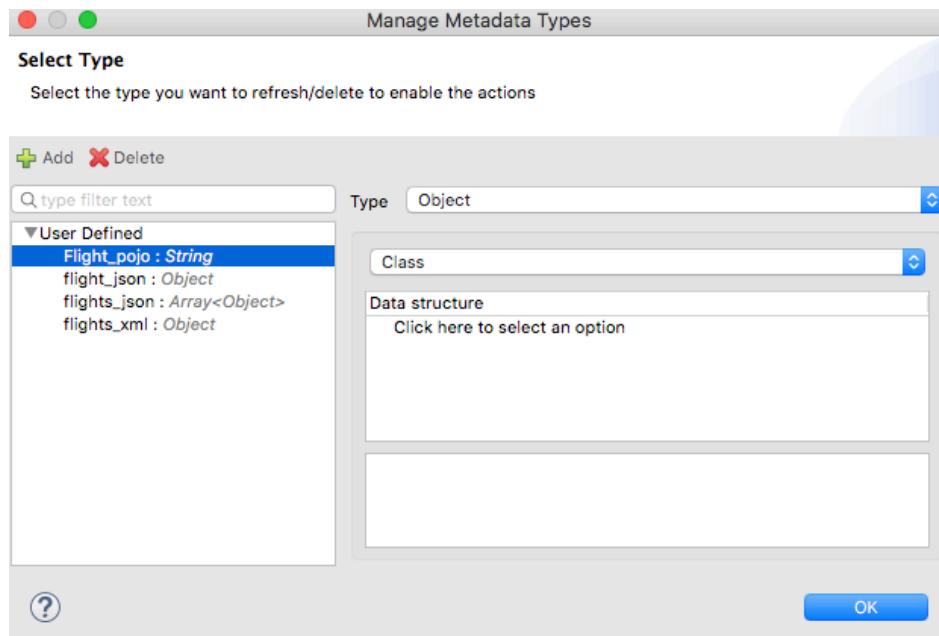
29. In the Manage Metadata Types dialog box, click Add.
30. In the Create new type dialog box, set the type id to flights_xml and click Create type.
31. In the Apply changes dialog box, click Yes.
32. In the Manage Metadata Types dialog box, set the first type to XML.

33. Select Example and browse to and select the flights-example.xml file in src/test/resources.

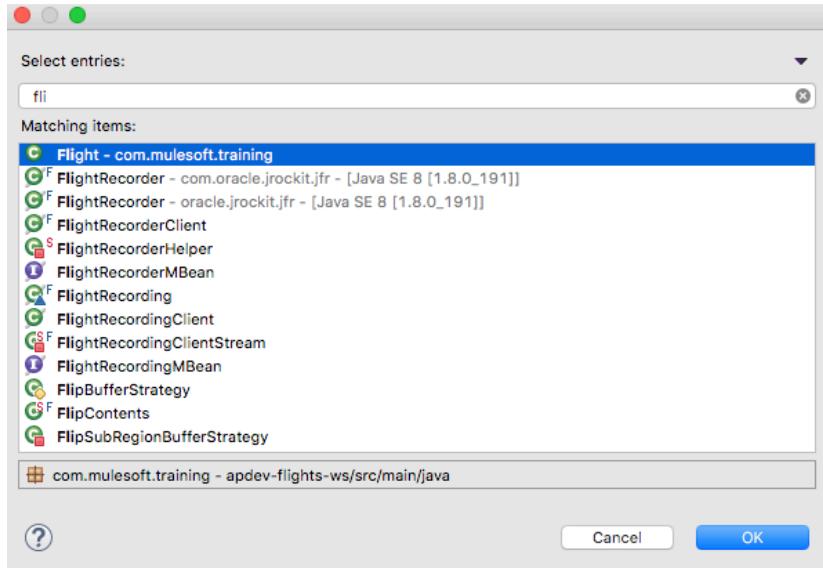


Create Flight_pojo metadata type

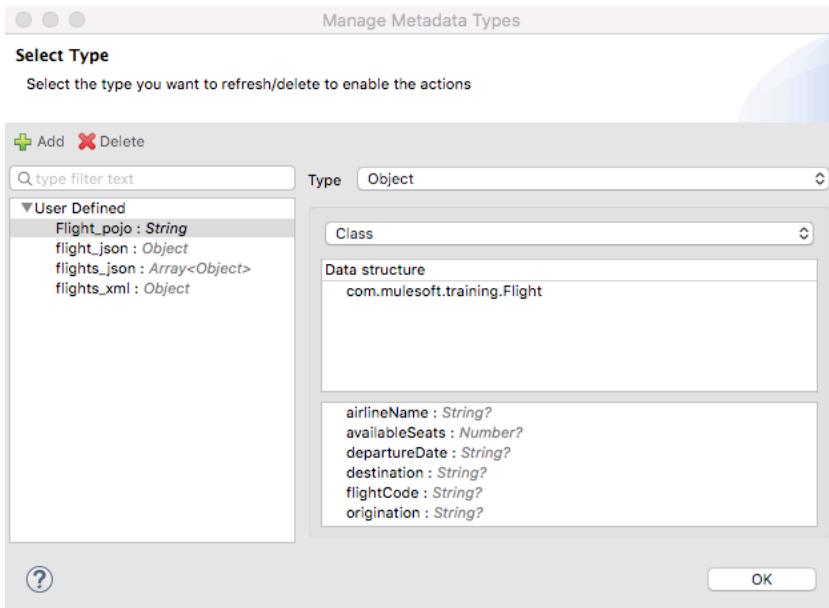
34. In the Manage Metadata Types dialog box, click the Add button.
35. In the Create new type dialog box, set the type id to Flight_pojo and click Create type.
36. In the Apply changes dialog box, click Yes.
37. In the Manage Metadata Types dialog box, set the first type to Object.



38. In the Data structure section, click the Click here to select an option link.
39. In the drop-down menu that appears, select Java object.
40. In the dialog box that opens, type fli in the Select entries field and then in the list of classes that appears, select Flight – com.mulesoft.training.



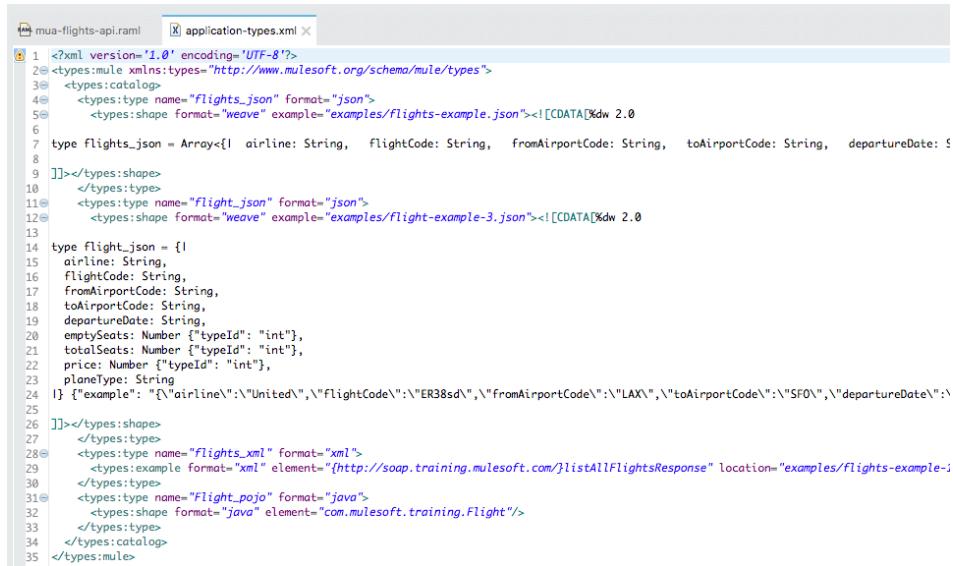
41. Click OK.



42. In the Manage Metadata Types dialog box, click OK.
43. In the Apply changes dialog box, click Yes.

Review the metadata definition file

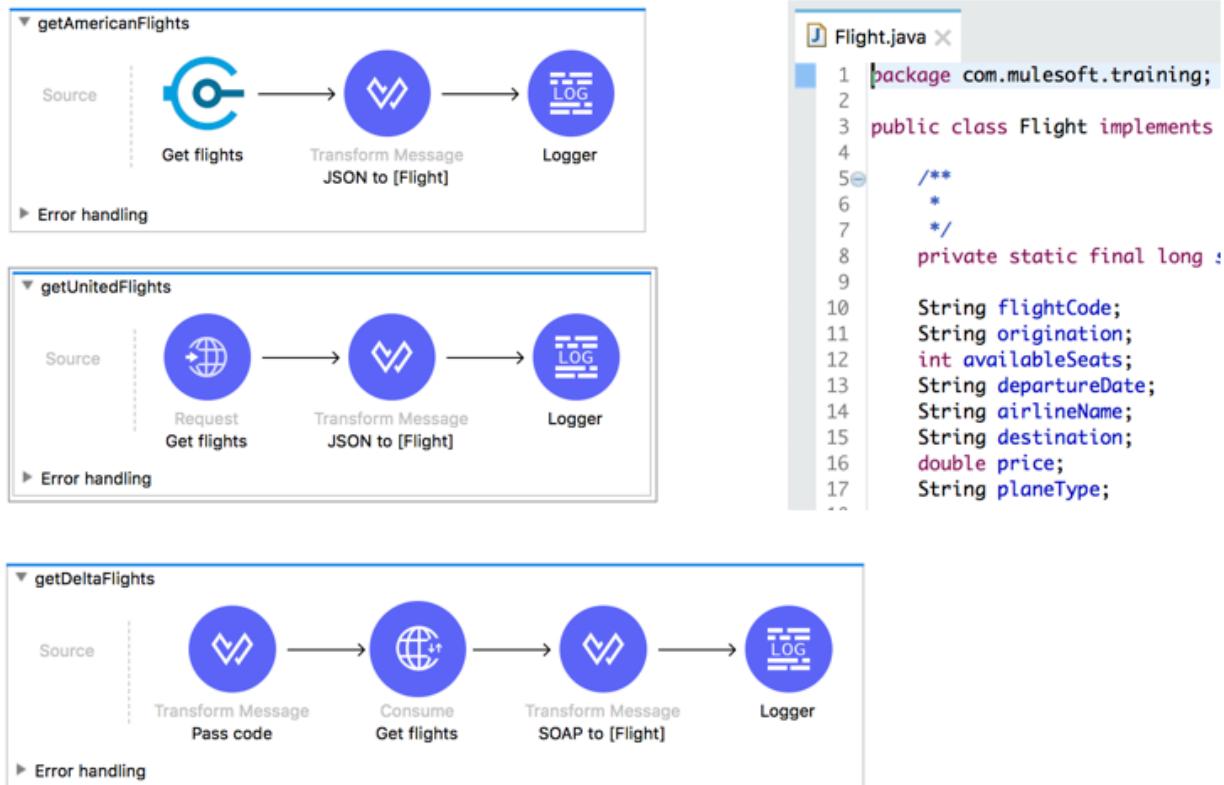
44. Return to application-types.xml in src/main/resources.



```
1 <?xml version='1.0' encoding='UTF-8'?>
2<types:mule xmlns:types="http://www.mulesoft.org/schema/mule/types">
3<types:catalog>
4<types:type name="flights_json" format="json">
5<types:shape format="weave" example="examples/flights-example.json"><![CDATA[%dw 2.0
6 type flights_json = Array<{! airline: String, flightCode: String, fromAirportCode: String, toAirportCode: String, departureDate: $7
7 }]></types:shape>
8</types:type>
9<types:type name="flight_json" format="json">
10<types:shape format="weave" example="examples/flight-example-3.json"><![CDATA[%dw 2.0
11 type flight_json = {
12   airline: String,
13   flightCode: String,
14   fromAirportCode: String,
15   toAirportCode: String,
16   departureDate: String,
17   emptySeats: Number {"typeId": "int"},
18   totalSeats: Number {"typeId": "int"},
19   price: Number {"typeId": "int"},
20   planeType: String
21 } {"example": "{\"airline\":\"United\", \"FlightCode\":\"ER38sd\", \"fromAirportCode\":\"LAX\", \"toAirportCode\":\"SFO\", \"departureDate\":\\\"2023-01-15T12:00:00Z\\\"}"}></types:shape>
22</types:type>
23<types:type name="flights_xml" format="xml">
24<types:example format="xml" element="http://soap.training.mulesoft.com/jlistAllFlightsResponse" location="examples/flights-example-3.xml">
25</types:example>
26</types:type>
27<types:type name="Flight_pojo" format="java">
28<types:shape format="java" element="com.mulesoft.training.Flight"/>
29</types:type>
30</types:catalog>
31</types:mule>
32</types:catalog>
33</types:mule>
34</types:catalog>
35</types:mule>
```

45. Close the file.

Module 8: Consuming Web Services



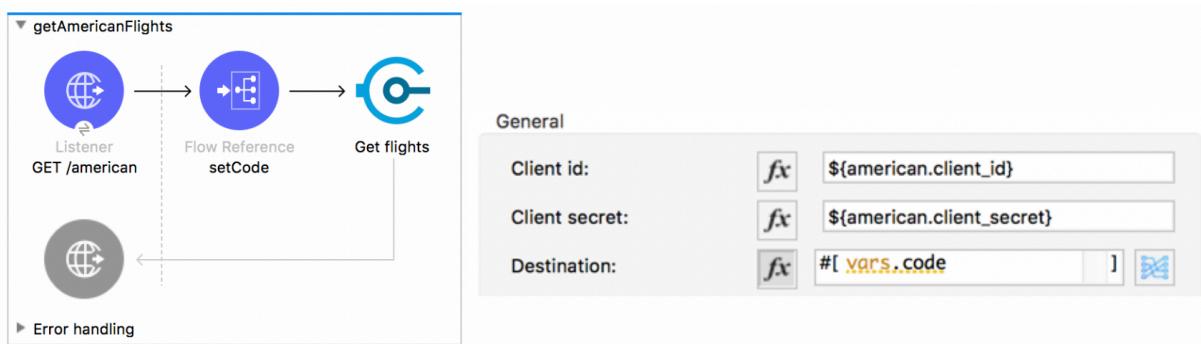
At the end of this module, you should be able to:

- Consume web services that have an API (and connector) in Exchange.
- Consume RESTful web services.
- Consume SOAP web services.
- Pass parameters to SOAP web services using the Transform Message component.
- Transform data from multiple services to a canonical format.

Walkthrough 8-1: Consume a RESTful web service that has an API (and connector) in Exchange

In this walkthrough, you consume the American flights RESTful web service that you built that has an API and a connector in Exchange. You will:

- Create a new flow to call the American RESTful web service.
- Add a REST connector from Exchange to an Anypoint Studio project.
- Configure and use a REST connector to make a call to a web service.
- Dynamically set a query parameter for a web service call.



Review the auto-generated REST connectors in Exchange

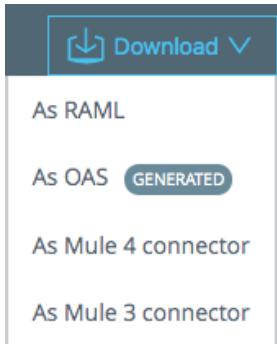
1. Return to Exchange in Anypoint Platform.
2. Locate your American flights API.

The screenshot shows the Exchange interface in Anypoint Platform. The left sidebar shows "All assets" and "Training (master)". The main area displays "Training assets (master)" with a search bar and a "New asset" button. A card for the "American Flights API" is shown, categorized under "REST API" with a 5-star rating and created by "Max Mule".

- Select the API and review its information.

The screenshot shows the Mule Exchange interface. On the left, there's a sidebar with navigation links like 'Assets list', 'PAGES', 'Home', 'SPECIFICATION', 'Summary', 'Endpoints', 'Types', and 'OTHER DETAILS'. The main content area displays the 'American Flights API' with a green icon, version v1, and public status. It has a rating of 0 reviews and a 'Rate and review' button. Below this, there's a section for 'Add description' and a summary: 'The American Flights API is a system API for operations on the american table in the training database.' A 'Supported operations' section lists endpoints: '/flights' (Get all flights, Get a flight with a specific ID), '/{ID}' (Add a flight, Delete a flight, Update a Flight), and 'Types'. To the right, there are sections for 'Share', 'Download' (with a dropdown menu), 'Edit', and 'MM'. Below these are details: Type (REST API), Organization (Training), Created by (Max Mule), Published on (Nov 11, 2019), and Visibility (Public). A table titled 'Asset versions for v1' shows three versions: 1.0.2 (Mocking Service), 1.0.1, and 1.0.0.

- Click the download drop-down menu button; you should see that REST Connect has created connectors for the API in Exchange.



Make a request to the web service

- Return to Advanced REST Client.
- Select the first tab and set client_id and client_secret as headers using the values you saved in the course snippets.txt file.

7. Send a GET request to your American Flights API

<http://training4-american-api-{lastname}.{region}.cloudbus.io/flights>; you should still see JSON data for the American flights as a response.

The screenshot shows a REST client interface with the following details:

- Method:** GET
- Request URL:** http://training4-american-api-maxmule.us-e2.clc
- SEND** button
- Parameters** section:
 - Headers tab (selected)
 - Authorization tab
 - Actions tab

Header name	Header value
client_id	3c376d605d7f4a5b849e435729f6fe1c
client_secret	DF04B05003f748D5b082e86C624D9E

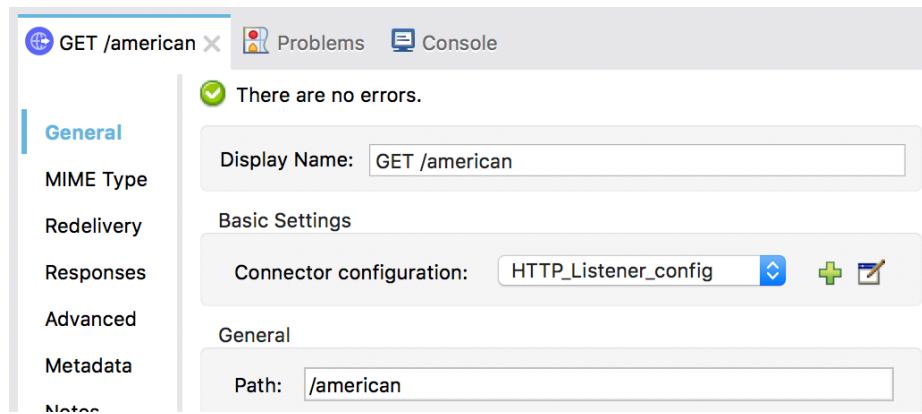
ADD HEADER button
- Response:** 200 OK | 1059.00 ms
- Details:** [JSON Response] (clickable)

```
[{"ID": 1, "code": "rree0001", "price": 541, "departureDate": "2016-01-20T00:00:00", "origin": "MUA", "destination": "LAX", "emptySeats": 0, "plane": {"type": "Boeing 787", "totalSeats": 200}}, {"ID": 2,
```

Add a new flow with an HTTP Listener

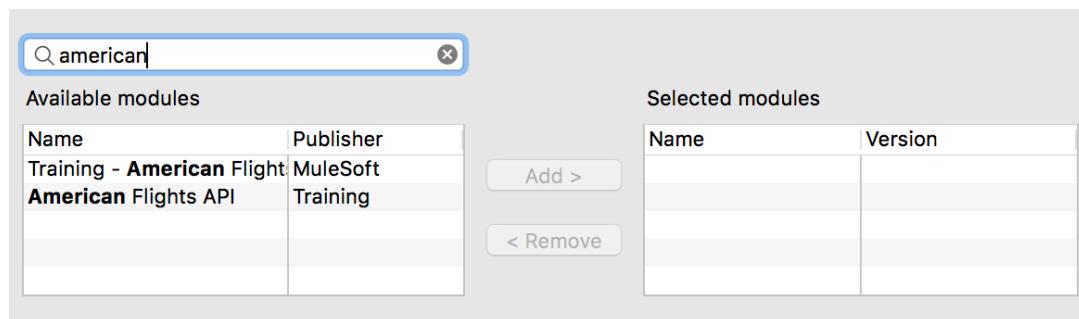
8. Return to the apdev-flights-ws project in Anypoint Studio.
9. Open implementation.xml
10. Drag out an HTTP Listener and drop it in the canvas.
11. Rename the flow to getAmericanFlights.
12. In the Listener properties view, set the display name to GET /american.
13. Ensure the connector configuration is set to the existing HTTP_Listener_config.
14. Set the path to /american.

- Set the allowed methods to GET.



Add the American Flights API module to Anypoint Studio

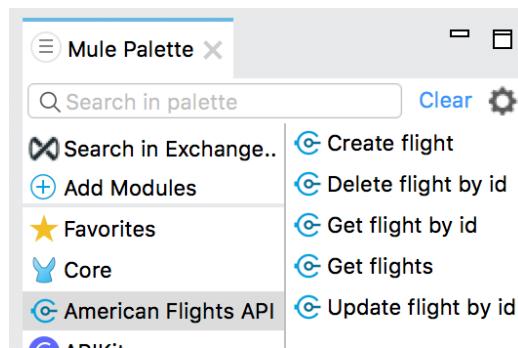
- In the Mule Palette, select Search in Exchange.
- In the Add Modules to Project dialog box, enter american in the search field.



- Select your American Flights API (not the Training: American Flights API) and click Add.

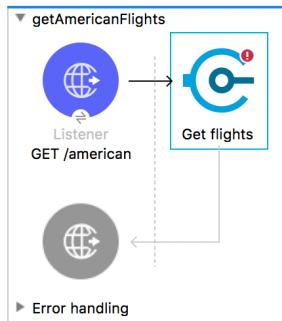
Note: If you did not successfully create the American Flights API in the first part of the course, you can use the pre-built Training: American Flights API connector.

- Click Finish.
- Wait for the module to be downloaded.
- Select the new American Flights API module in the Mule Palette.

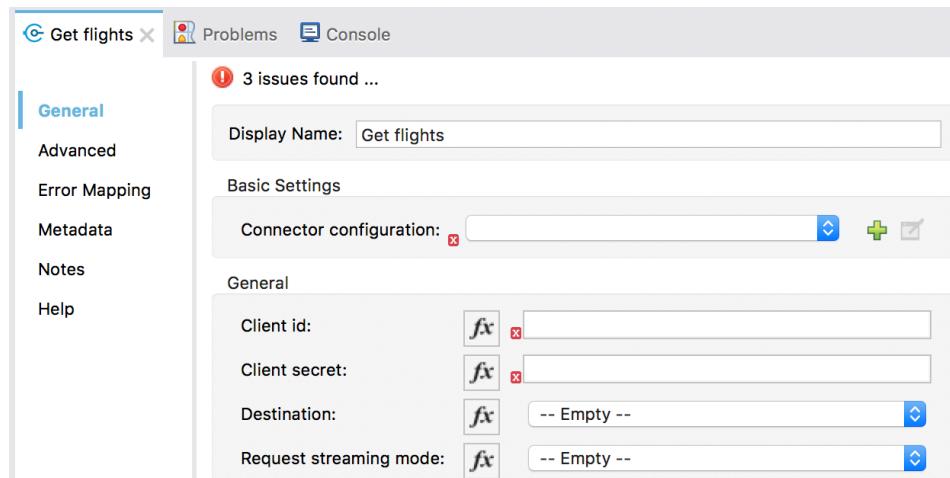


Add a Get all flights operation

22. Select the Get flights operation in the right side of the Mule Palette and drag and drop it in the process section of the flow.



23. Select the Get flights operation in the canvas and in its properties view, see that you need to create a new configuration and enter a client_id and client_secret.



Configure the American Flights API connector

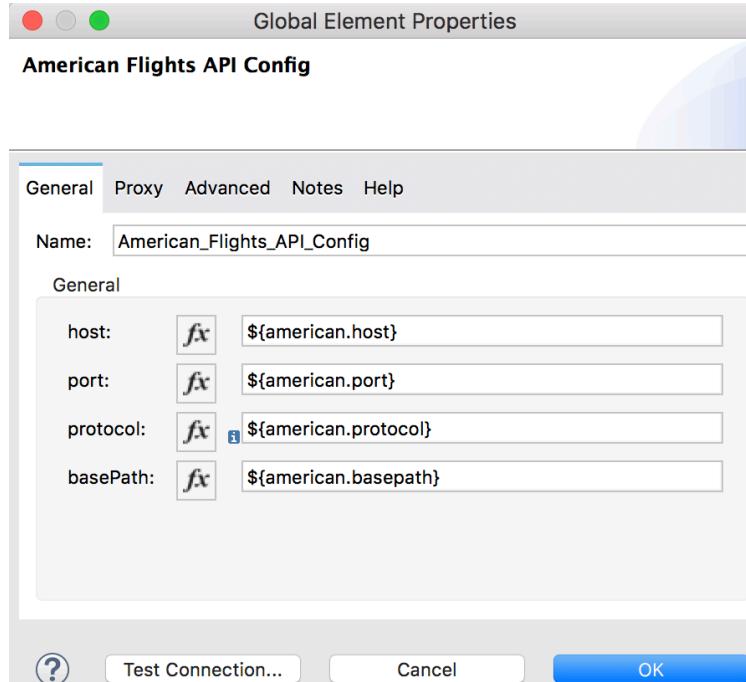
24. Return to global.xml.
25. In the Global Elements view, click Create.

26. In the Choose Global Type dialog box, select Connector Configuration > American Flights API Config and click OK.



27. In the Global Element Properties dialog box, set each field to a corresponding property placeholder (that you will define and set next):

- host: \${american.host}
- port: \${american.port}
- protocol: \${american.protocol}
- basePath: \${american.basepath}



28. Click OK.

29. Return to the course snippets.txt file and copy the text for the American RESTful web service properties.

30. Return to config.yaml in src/main/resources and paste the code at the end of the file.

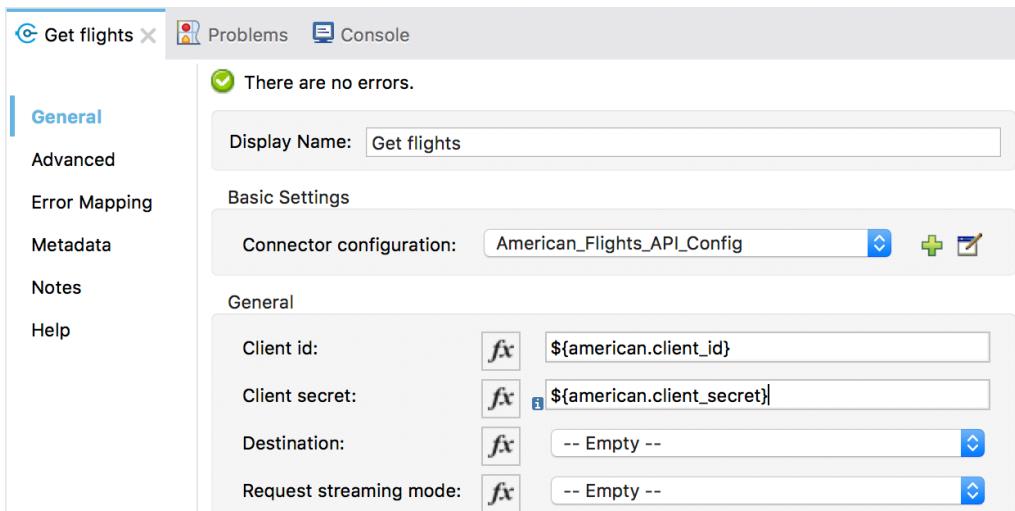
```
1 http:  
2   port: "8081"  
3  
4 american:  
5   host: "training4-american-api-{lastname}.{region}.cloudbhub.io"  
6   port: "80"  
7   basepath: "/"  
8   protocol: "HTTP"  
9   client_id:  
10  client_secret:
```

31. In the american.host property, replace {lastname} and {region} with your application identifiers.
32. Return to Advanced REST Client and copy the value for your client_id.
33. Return to config.yaml and paste the value within double quotes.
34. Repeat for your client_secret.

Note: If you have used the pre-built Training: American Flights API, use the host, client_id, and client_secret values from the course snippets.txt file.

Configure the Get flights operation

35. Return to implementation.xml.
36. In the Get flights properties view, ensure the Connector configuration is set to American_Flights_API_Config.
37. Set the client id to the \${american.client_id} property.
38. Set the client secret to the \${american.client_secret} property.
39. Leave the destination field blank.



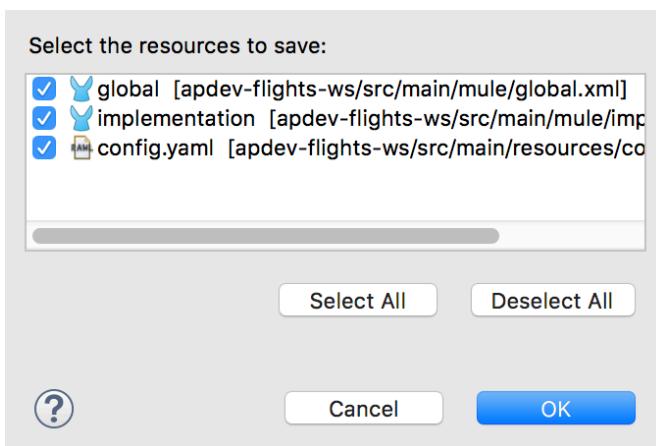
Review metadata associated with the operation

40. Select the output tab in the DataSense Explorer and expand Payload.

The screenshot shows the DataSense Explorer interface with the 'Output' tab selected. A search bar at the top contains the placeholder 'type filter text'. Below it, the 'Mule Message' section is expanded, showing the 'Payload' section. The 'Payload' section is expanded, revealing an array of objects with properties: plane (Object?), code (String), price (Number), origin (String), destination (String), ID (Number?), departureDate (String), and emptySeats (Number). There are also sections for 'Attributes' (Void) and 'Variables'.

Test the application

41. Run the project.
42. In the dialog box, ensure all the resources are selected and click OK.



43. In Advanced REST Client, return to the tab with the localhost request.

44. Change the URL to make a request to <http://localhost:8081/american>; you should get all the flights.

The screenshot shows a REST client interface with the following details:

- Method: GET
- Request URL: <http://localhost:8081/american>
- SEND button
- Parameters dropdown
- Status: 200 OK (highlighted in green) | 3283.10 ms
- DETAILS dropdown
- Content pane:

```
[Array[11]
-0: {
  "ID": 1,
  "code": "rree0001",
  "price": 541,
  "departureDate": "2016-01-20T00:00:00",
  "origin": "MUA",
  "destination": "LAX",
  "emptySeats": 0,
  "plane": {
    "type": "Boeing 787",
    "totalSeats": 200
  }
},
-1: {
  "ID": 2,
  "code": "eefd0123",
}
```

Review the specification for the API you are building

45. Return to mua-flights-api.raml in src/main/resources/api.

46. Review the optional query parameters.

```
/flights:
get:
  displayName: Get flights
  queryParameters:
    code:
      displayName: Destination airport code
      required: false
      enum:
        - SFO
        - LAX
        - PDX
        - CLE
        - PDF
    airline:
      displayName: Airline
      required: false
      enum:
        - united
        - delta
        - american
```

47. Locate the return type for the get method of the /flights resource.

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

48. Open FlightsExample.raml in src/main/resources/api.examples and review its structure.

```
#%RAML 1.0 NamedExample  
value:  
  -  
    airline: United  
    flightCode: ER38sd  
    fromAirportCode: LAX  
    toAirportCode: SFO  
    departureDate: May 21, 2016  
    emptySeats: 0  
    totalSeats: 200  
    price: 199  
    planeType: Boeing 737  
  -  
    airline: Delta  
    flightCode: ER0945  
    fromAirportCode: PDX  
    toAirportCode: CLE  
    departureDate: June 1, 2016  
    emptySeats: 24  
    totalSeats: 350  
    price: 450  
    planeType: Boeing 747
```

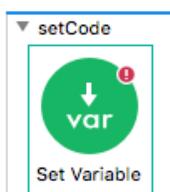
Create a variable to set the destination airport code

49. Return to implementation.xml.

50. Drag a Sub Flow scope from the Mule Palette and drop it at the top of the canvas.

51. Change the name of the flow to setCode.

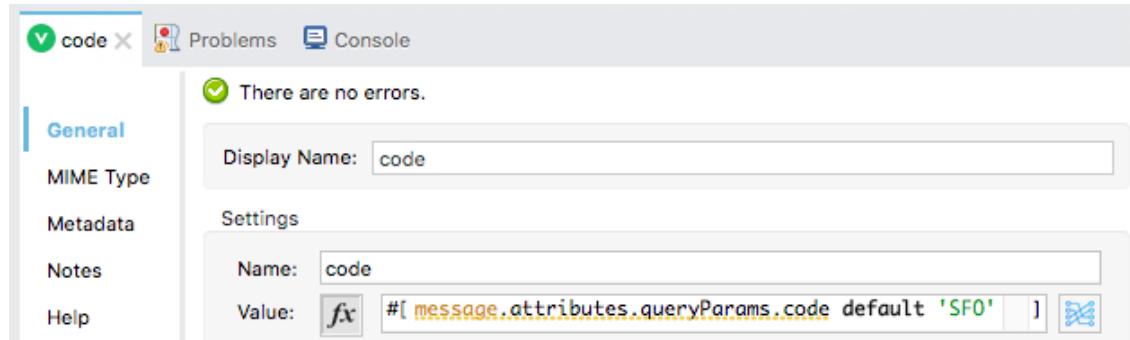
52. Drag a Set Variable transformer from the Mule Palette and drop it in the setCode subflow.



53. In the Set Variable properties view, set the name and display name to code.

54. Switch the value field to expression mode then set its value to a query parameter called code and give it a default value of SFO if no query parameter is passed to the flow.

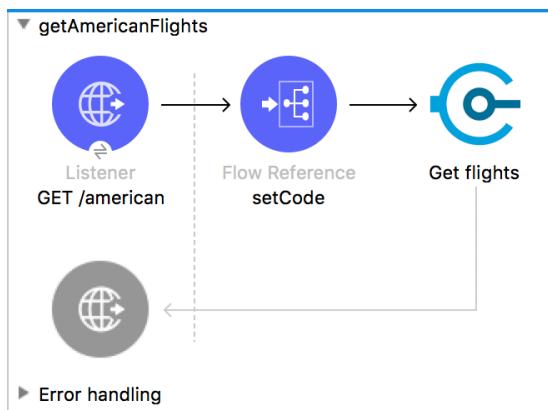
```
message.attributes.queryParams.code default 'SFO'
```



Dynamically set a query parameter for the web service call

55. Drag a Flow Reference component from the Mule Palette and drop it after the GET /american Listener in getAmericanFlights.

56. In the Flow Reference properties view, set the flow name and display name to setCode.



57. In the Get flights properties view, switch the destination field to expression mode then set its value to the variable containing the airport code.

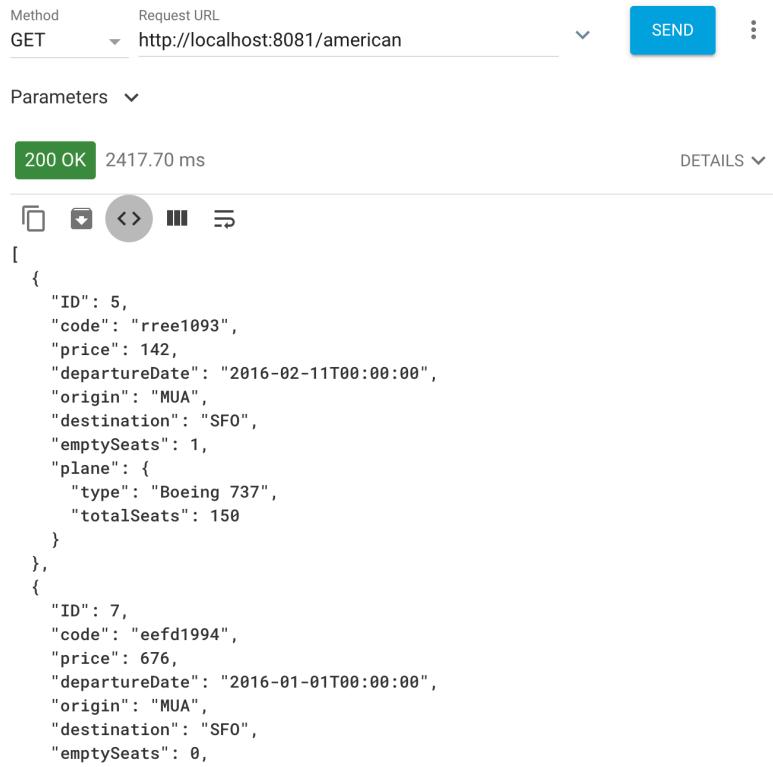
```
vars.code
```

The screenshot shows the 'Get flights' properties view. The 'General' tab is selected. Under 'Settings', the 'Destination' field contains the expression '#[vars.code]'.

Test the application

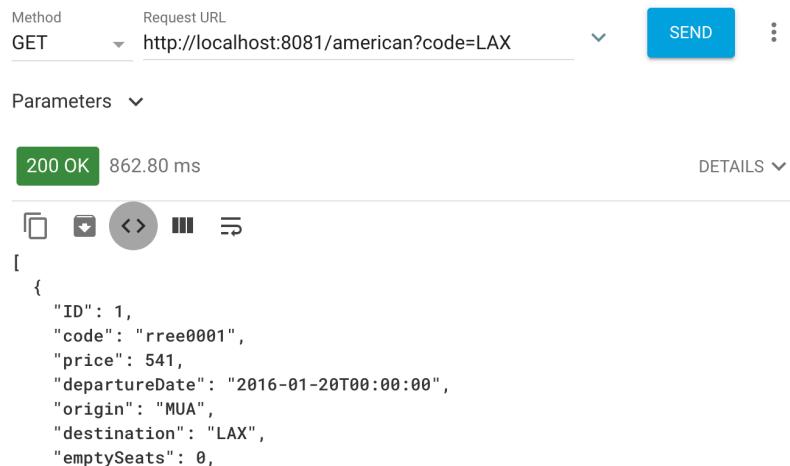
58. Save the file to redeploy the application.

59. In Advanced REST Client, send the same request; this time you should only get flights to SFO.



```
[{"ID": 5, "code": "rree1093", "price": 142, "departureDate": "2016-02-11T00:00:00", "origin": "MUA", "destination": "SFO", "emptySeats": 1, "plane": {"type": "Boeing 737", "totalSeats": 150}}, {"ID": 7, "code": "eefd1994", "price": 676, "departureDate": "2016-01-01T00:00:00", "origin": "MUA", "destination": "SFO", "emptySeats": 0, "plane": {"type": "Boeing 737", "totalSeats": 150}}]
```

60. Add query parameter called code equal to LAX and make the request; you should now only see flights to LAX.



```
[{"ID": 1, "code": "rree0001", "price": 541, "departureDate": "2016-01-20T00:00:00", "origin": "MUA", "destination": "LAX", "emptySeats": 0, "plane": {"type": "Boeing 737", "totalSeats": 150}}]
```

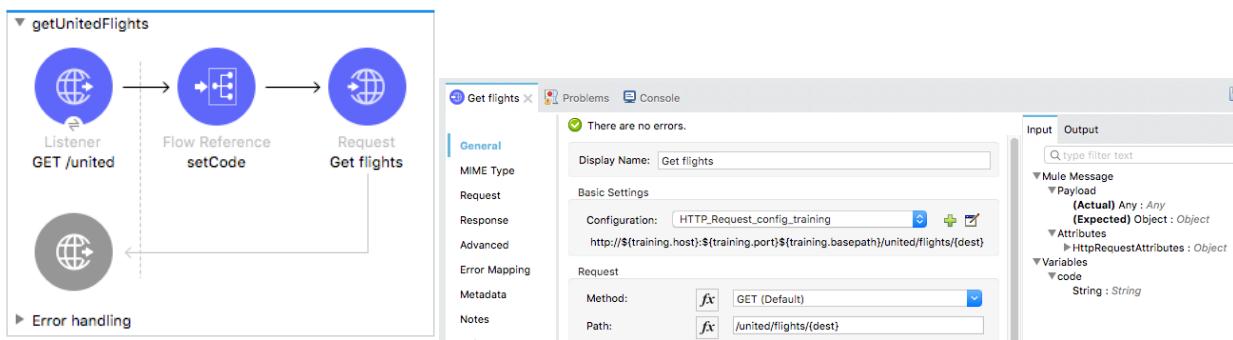
61. Examine the data structure of the JSON response.

Note: You will transform the data to the format specified by the mua-flights-api in a later walkthrough.

Walkthrough 8-2: Consume a RESTful web service

In this walkthrough, you consume the United RESTful web service that does not have an API in Exchange. You will:

- Create a new flow to call the United RESTful web service.
- Use the HTTP Request operation to call a RESTful web service.
- Dynamically set a URI parameter for a web service call.
- Add metadata for an HTTP Request operation's response.



Make a request to the United web service

1. Return to the course snippets.txt file.
2. Locate and copy the United RESTful web service URL.
3. In Advanced REST Client, make a new tab and make a GET request to this URL.
4. Review the structure of the JSON response.

Method Request URL

GET http://mu.learn.mulesoft.com/united/flights

SEND

Parameters

200 OK 336.10 ms DETAILS

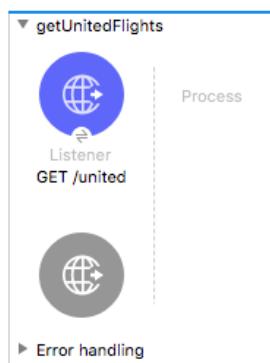
```
{  
  "flights": [Array[12],  
    0: {  
      "code": "ER38sd",  
      "price": 400,  
      "origin": "MUA",  
      "destination": "SFO",  
      "departureDate": "2015/03/20",  
      "planeType": "Boeing 737",  
      "airlineName": "United",  
      "emptySeats": 0  
    }]
```

5. Look at the destination values; you should see SFO, LAX, CLE, PDX, and PDF.

6. In the URL field, add the destination CLE as a URI parameter:
<http://mu.learn.mulesoft.com/united/flights/CLE>.
7. Send the request; you should now only see flights to CLE.

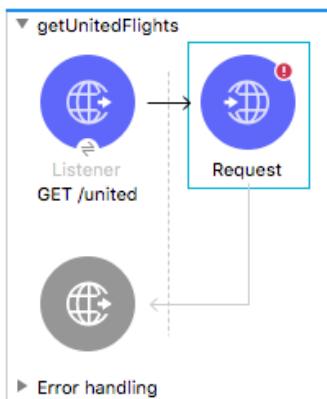
Add a new flow with an HTTP Listener operation

8. Return to implementation.xml in Anypoint Studio.
9. Drag out an HTTP Listener from the Mule Palette and drop it at the bottom of the canvas.
10. Change the name of the flow to getUnitedFlights.
11. In the Listener properties view, set the display name to GET /united.
12. Ensure the connector configuration is set to the existing `HTTP_Listener_config`.
13. Set the path to `/united`.
14. Set the allowed methods to GET.



Add an HTTP Request operation

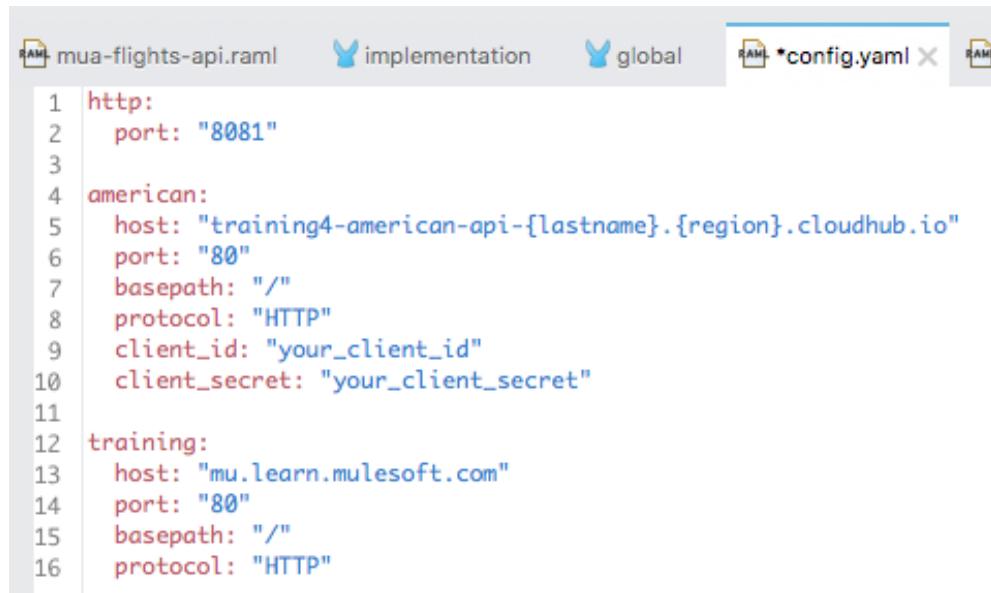
15. Drag out an HTTP Request from the Mule Palette and drop it into the process section of `getUnitedFlights`.



16. In the Request properties view, set the display name to Get flights.

Create an HTTP Request configuration

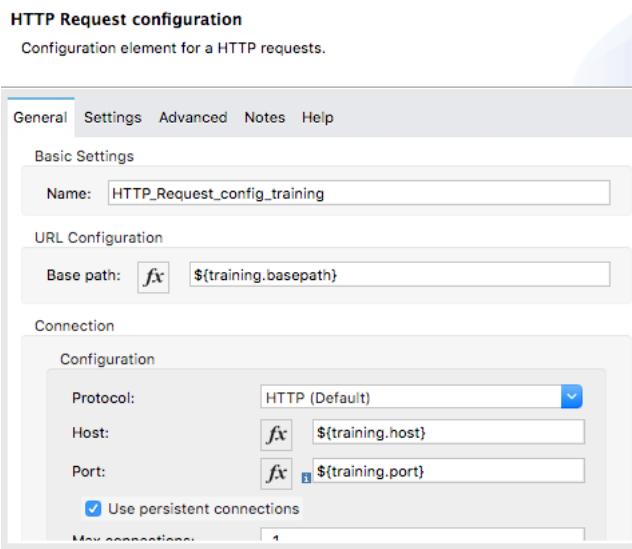
17. Return to the course snippets.txt file and copy the text for the Training web service properties.
18. Return to config.yaml and paste the code at the end of the file.



```
1 http:
2   port: "8081"
3
4 american:
5   host: "training4-american-api-{lastname}.{region}.cloudbhub.io"
6   port: "80"
7   basepath: "/"
8   protocol: "HTTP"
9   client_id: "your_client_id"
10  client_secret: "your_client_secret"
11
12 training:
13   host: "mu.learn.mulesoft.com"
14   port: "80"
15   basepath: "/"
16   protocol: "HTTP"
```

19. Return to the Global Elements view in global.xml.
20. Create a new HTTP Request configuration with the following values:

- Name: HTTP_Request_config_training
- Base Path: \${training.basepath}
- Host: \${training.host}
- Port: \${training.port}



21. Click OK.

Configure the HTTP Request operation

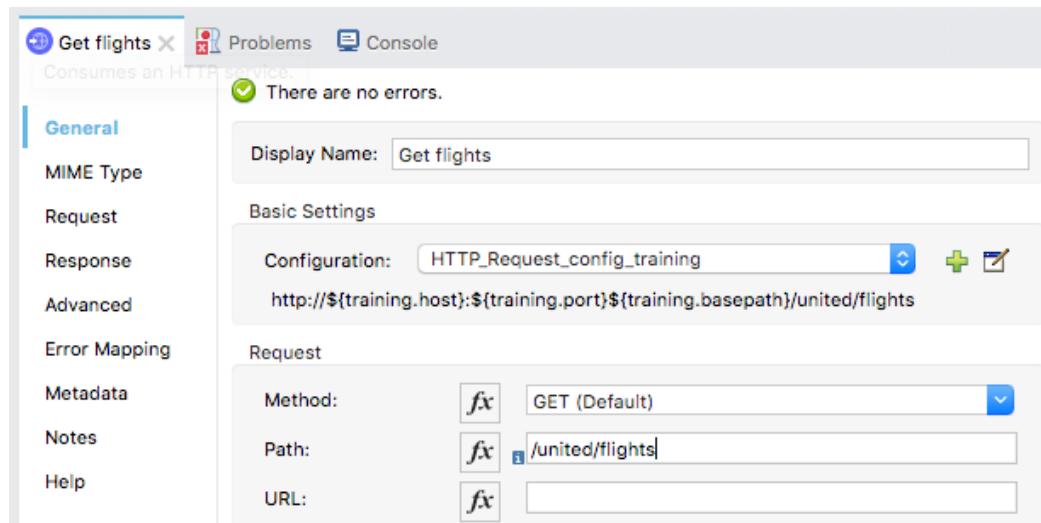
22. Return to implementation.xml.

23. Navigate to the Get flights Request properties view in getUnitedFlights.

24. Set the configuration to the existing HTTP_Request_config_training.

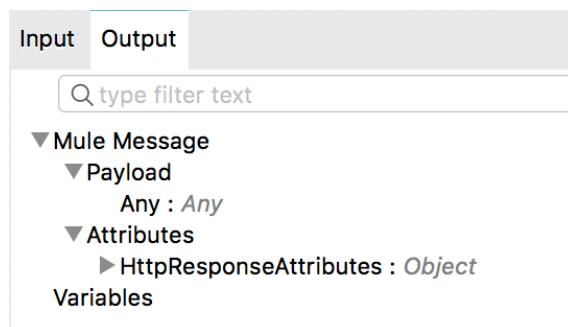
25. Leave the method set to GET.

26. Set the path to /united/flights.



Review metadata associated with the United Get flights operation response

27. Select the Output tab in the DataSense Explorer; you should see no metadata for the payload.



- Input
- Output

type filter text

- ▼ Mule Message
 - ▼ Payload
 - Any : Any
 - ▼ Attributes
 - HttpResponseAttributes : Object
 - Variables

Test the application

28. Save the files to redeploy the project.

29. In Advanced REST Client, return to the tab with the localhost request.

30. Change the URL to make a request to <http://localhost:8081/united>; you should get JSON flight data for all destinations returned.

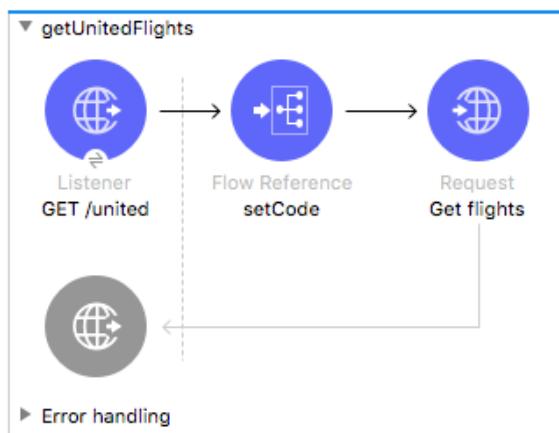
The screenshot shows a REST client interface. At the top, there are fields for 'Method' (set to 'GET') and 'Request URL' (set to 'http://localhost:8081/united'). Below these are buttons for 'SEND' and a three-dot menu. Underneath, a 'Parameters' dropdown is shown. The main area displays the response: a green '200 OK' box indicating success, a '523.70 ms' latency, and a 'DETAILS' button. The response body is a JSON object:

```
{
  "flights": [Array[12]
    -0: {
      "code": "ER38sd",
      "price": 400,
      "origin": "MUA",
      "destination": "SFO",
      "departureDate": "2015/03/20",
      "planeType": "Boeing 737",
      "airlineName": "United",
      "emptySeats": 0
    },
    -1: {
      "code": "ER45if",
      "price": 345.99,
      "origin": "MUA"
    }
  ]
}
```

31. Add a query parameter named code and set it to one of the destination airport code values: <http://localhost:8081/united?code=CLE>; you should still get flights to all destinations.

Add a URI parameter to the web service call

32. Return to implementation.xml in Anypoint Studio.
 33. Drag a Flow Reference component from the Mule Palette and drop it after the GET /united Listener in getUnitedFlights.
 34. In the Flow Reference properties view, set the flow name and display name to setCode.



35. Navigate to the Get flights Request properties view in getUnitedFlights.

36. In the Request section, change the path to /united/flights/{dest}.

The screenshot shows the 'Get flights' request configuration in the Anypoint Studio interface. The 'General' tab is selected. The 'Display Name' is set to 'Get flights'. Under 'Basic Settings', the 'Configuration' is set to 'HTTP_Request_config_training' and the 'Path' is set to 'http://\${training.host}:\${training.port}\${training.basepath}/united/flights/{dest}'. The 'Request' section shows the 'Method' as 'GET (Default)' and the 'Path' as '/united/flights/{dest}'.

37. Select the URI Parameters tab.

38. Click the Add button.

39. Set the name to dest.

40. Set the value to the value of the code variable.

`vars.code`

The screenshot shows the 'Get flights' request configuration in the Anypoint Studio interface. The 'General' tab is selected. The 'Display Name' is set to 'Get flights'. Under 'Basic Settings', the 'Configuration' is set to 'HTTP_Request_config_training' and the 'Path' is set to 'http://\${training.host}:\${training.port}\${training.basepath}/united/flights/{dest}'. The 'Request' section shows the 'Method' as 'GET (Default)' and the 'Path' as '/united/flights/{dest}'. Below the request fields, there is a 'Query' tab with a table showing a single entry: 'dest' with a value of 'vars.code'. There is also a note at the bottom of the table: 'Switch to expression mode.'

Name	Value
"dest"	vars.code

Test the application

41. Save the file to redeploy the application.
42. In Advanced REST Client, make the same request; you should now only get flights to CLE.

The screenshot shows the Advanced REST Client interface. At the top, the method is set to GET and the URL is http://localhost:8081/united?code=CLE. Below the URL, there's a 'Parameters' dropdown. The response status is 200 OK (green box) with a time of 2617.30 ms. To the right is a 'DETAILS' button. The payload is displayed as JSON:

```
{  
  "flights": [Array[2]]  
  -0: {  
    "code": "ER9fje",  
    "price": 845,  
    "origin": "MUA",  
    "destination": "CLE",  
    "departureDate": "2015/07/11",  
  }  
}
```

43. Remove the code parameter and make the request; you should now only get results for SFO.

Note: You will transform the data to the format specified by the mua-flights-api in a later walkthrough.

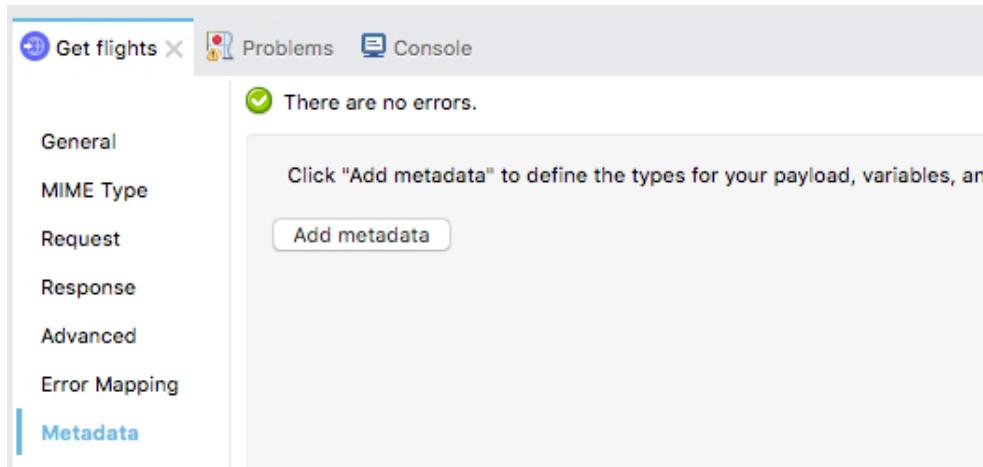
Add metadata for the United Get flights operation response

44. Return to Anypoint Studio.
45. Navigate to the properties view for the Get flights operation in getUnitedFlights.
46. Select the Output tab in the DataSense Explorer and expand Payload; you should still see no metadata for the payload.

The screenshot shows the DataSense Explorer in Anypoint Studio. The 'Output' tab is selected. Under 'Mule Message', the 'Payload' section is expanded, showing 'Any : Any'. Other sections like 'Attributes' and 'Variables' are also visible.

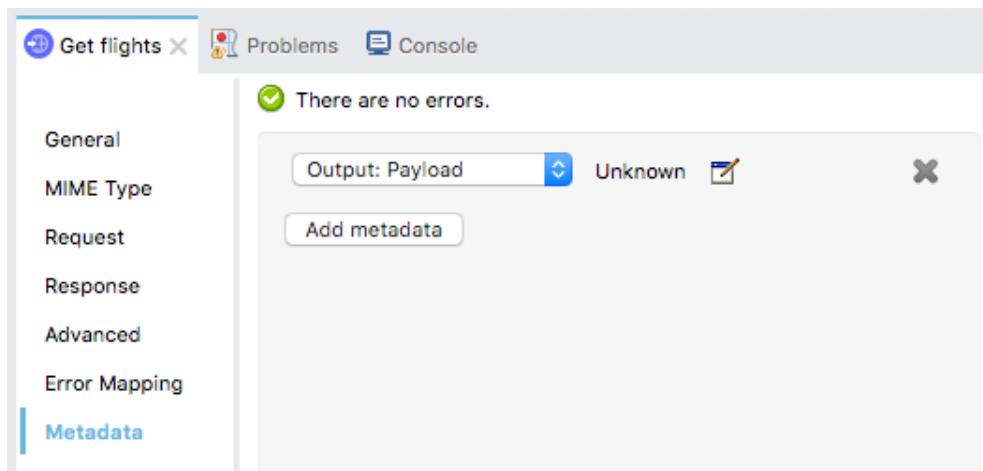
47. In the Get flights properties view, select the Metadata tab.

48. Click the Add metadata button.



49. Change the drop-down menu to Output: Payload.

50. Click the Edit button.



51. In the Select metadata type dialog box, click the Add button.

52. In the Create new type dialog box, set the type id to united_flights_json.

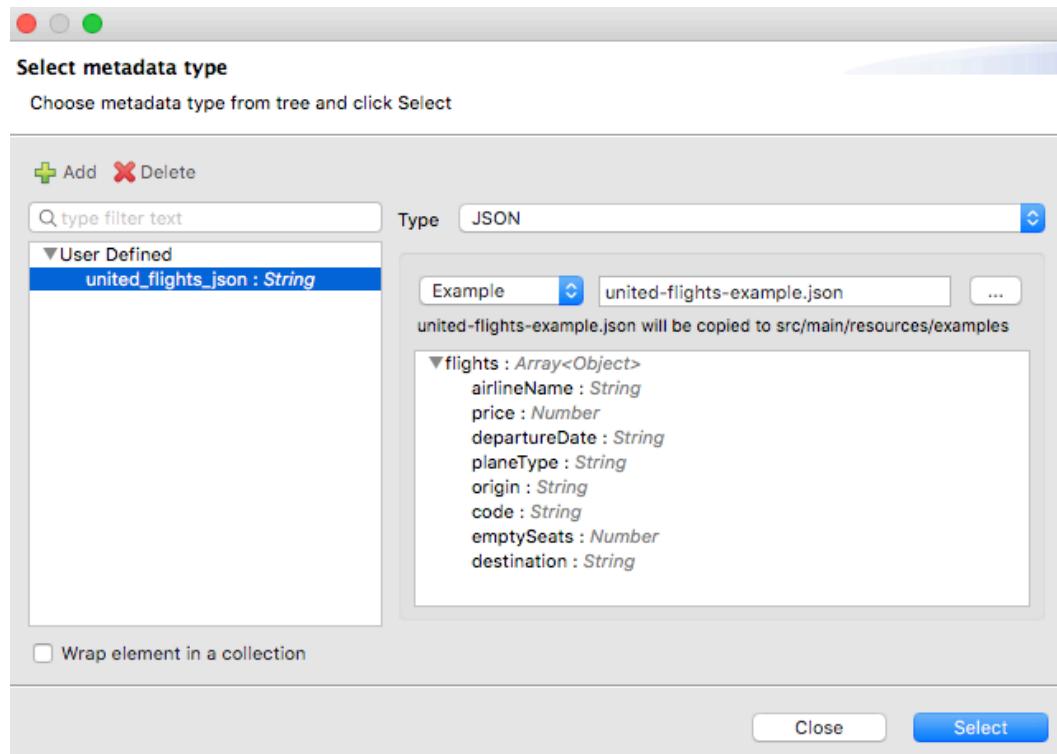
53. Click Create type.

54. In the Select metadata type dialog box, set the first type to JSON.

55. Change the Schema selection to Example.

56. Click the browse button and navigate to the project's src/test/resources folder.

57. Select `united-flights-example.json` and click Open; you should see the example data for the metadata type.



58. Click Select.

59. In the DataSense Explorer, ensure the Output tab is selected and expand Payload; you should now see the structure for the payload.

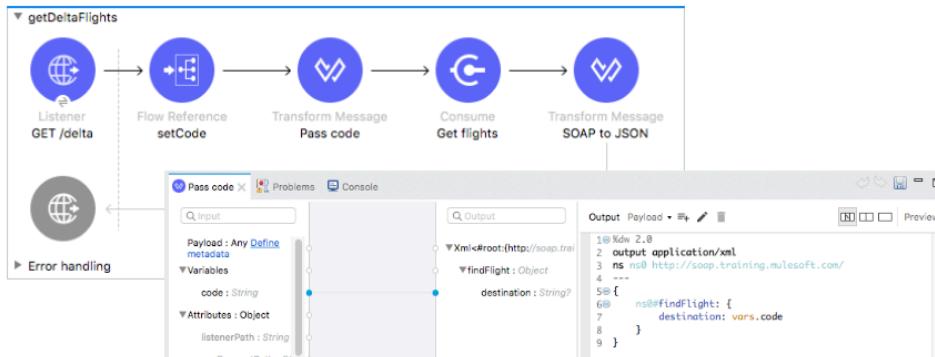
The screenshot shows the DataSense Explorer interface. The 'Output' tab is selected. At the top, there are tabs for 'Input' and 'Output', a search bar labeled 'type filter text', and a tree view. The tree view shows the following structure under 'Mule Message' and 'Payload':

- Object : Object
 - flights : Array<Object>
 - airlineName : String
 - price : Number
 - departureDate : String
 - planeType : String
 - origin : String
 - code : String
 - emptySeats : Number
 - destination : String
- Attributes
 - HttpResponseAttributes : Object
- Variables
 - code
 - String : String

Walkthrough 8-3: Consume a SOAP web service

In this walkthrough, you consume a Delta SOAP web service. You will:

- Create a new flow to call the Delta SOAP web service.
- Use a Web Service Consumer connector to consume a SOAP web service.
- Use the Transform Message component to pass arguments to a SOAP web service.



Browse the WSDL

1. Return to the course snippets.txt file and copy the Delta SOAP web service WSDL.
2. In Advanced REST Client, return to the third tab, the one with the learn.mulesoft request.
3. Paste the URL and send the request; you should see the web service WSDL returned.
4. Browse the WSDL; you should find references to operations listAllFlights and findFlight.

Method Request URL
GET <http://mu.learn.mulesoft.com/delta?wsdl> SEND :

Parameters [DETAILS](#)

200 OK 301.20 ms

This view is retired and will be replaced around June 2019. [Learn more.](#)

```
<wsdl:definitions xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:tns="http://soap.training.mulesoft.com/"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:ns1="http://schemas.xmlsoap.org/soap/http"
  name="TicketServiceService"
  targetNamespace="http://soap.training.mulesoft.com/">...
```

^ <wsdl:types>

```
<xsd:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:tns="http://soap.training.mulesoft.com/"
  elementFormDefault="unqualified"
  targetNamespace="http://soap.training.mulesoft.com/"
  version="1.0">
```

 <xsd:element name="findFlight" type="tns:findFlight" />

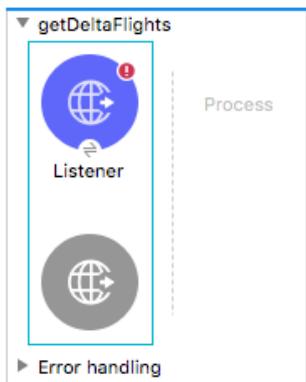
```
  <xsd:element name="findFlightResponse" type="tns:findFlightResponse" />
```

 <xsd:element name="listAllFlights" type="tns:listAllFlights" />

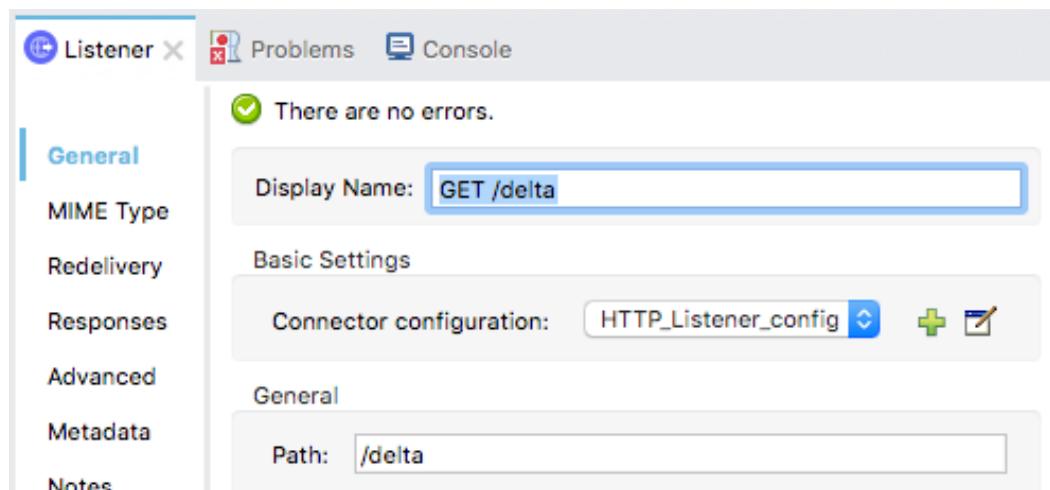
```
  <xsd:element name="listAllFlightsResponse" type="tns:listAllFlightsResponse" />
```

Create a new flow with an HTTP Listener connector endpoint

5. Return to Anypoint Studio.
6. Drag out another HTTP Listener from the Mule Palette and drop it in the canvas after the existing flows.
7. Rename the flow to getDeltaFlights.



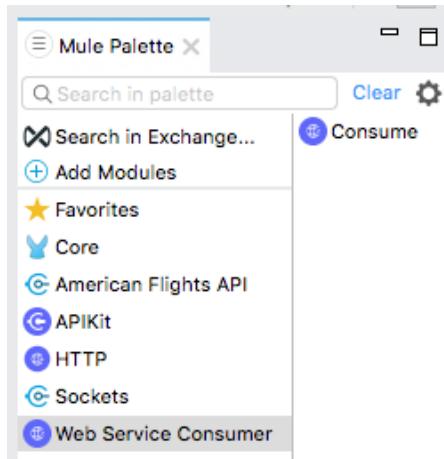
8. In the Listener properties view, set the display name to GET /delta.
9. Ensure the connector configuration is set to the existing HTTP_Listener_config.
10. Set the path to /delta and the allowed methods to GET.



Add the Web Service Consumer module to the project

11. In the Mule Palette, select Add Modules.
12. Select the Web Service Consumer connector in the right side of the Mule Palette and drag and drop it into the left side.

13. If you get a Select module version dialog box, select the latest version and click Add.



Configure the Web Service Consumer connector

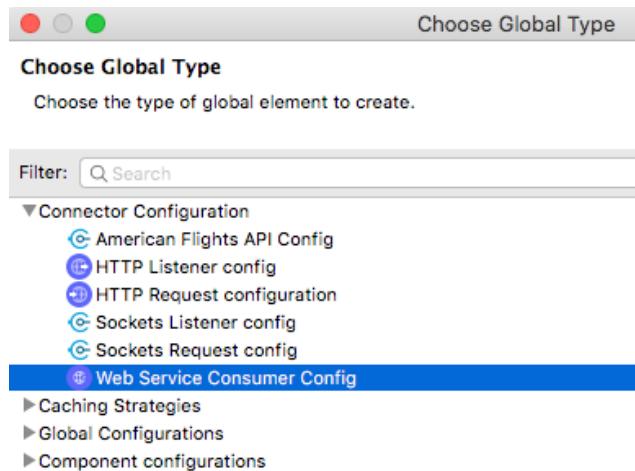
14. Return to the course snippets.txt file and copy the text for the Delta web service properties.
15. Return to config.yaml in src/main/resources and paste the code at the end of the file.

```
mua-flights-api.raml *implementation global *config.yaml FlightsExample.raml
1 http:
2   port: "8081"
3
4 american:
5   host: "training4-american-api-{lastname}.{region}.cloudbhub.io"
6   port: "80"
7   basepath: "/"
8   protocol: "HTTP"
9   client_id: "your_client_id"
10  client_secret: "your_client_secret"
11
12 training:
13   host: "mu.learn.mulesoft.com"
14   port: "80"
15   basepath: "/"
16   protocol: "HTTP"
17
18 delta:
19   wsdl: "http://mu.learn.mulesoft.com/delta?wsdl"
20   service: "TicketServiceService"
21   port: "TicketServicePort"
```

Note: The Delta web service properties you see may be different than what is shown here; the values in the snippets file differ for instructor-led and self-study training classes.

16. Save the file.
17. Return to global.xml.
18. Click Create.

19. In the Choose Global Type dialog box, select Connector Configuration > Web Service Consumer Config.

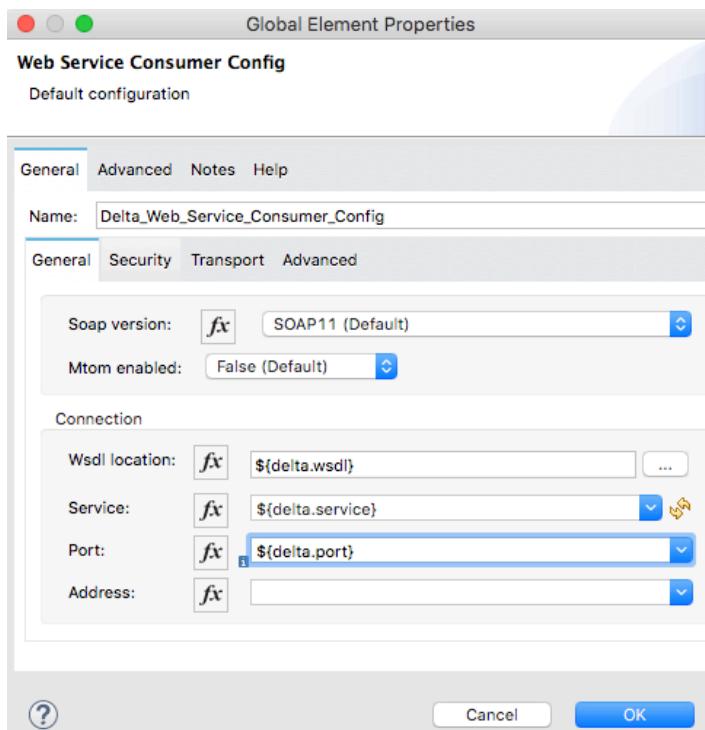


20. Click OK.

21. In the Global Element Properties dialog box, change the name to Delta_Web_Service_Consumer_Config.

22. Set the

- Wsdl location: \${delta.wsdl}
- Service: \${delta.service}
- Port: \${delta.port}

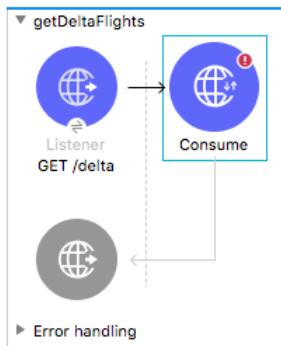


23. Click OK.

Add and configure a Consume operation

24. Return to implementation.xml.

25. Locate the Consume operation for the Web Service Consumer connector in the Mule Palette and drag and drop it in the process section of getDeltaFilghts.



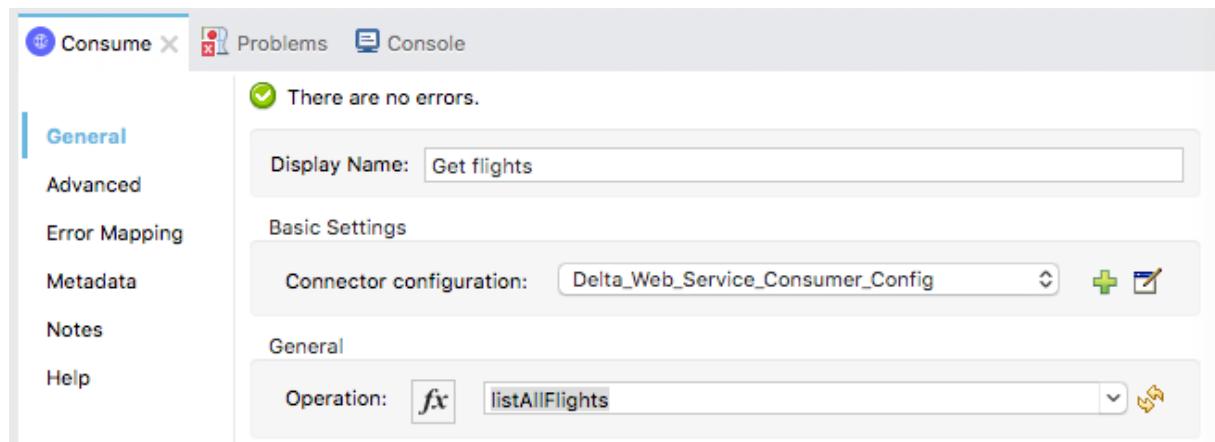
26. In the Consume properties view, change its display name to Get flights.

27. Ensure the connector configuration is set to the existing

Delta_Web_Service_Consumer_Config.

28. Click the operation drop-down menu; you should see the web service operations listed.

29. Select the listAllFlights operation.



Review metadata associated with the Delta Get flights operation response

30. Select the Output tab in the DataSense Explorer and expand Payload; you should see the payload structure.

The screenshot shows the DataSense Explorer interface with the 'Output' tab selected. A search bar at the top contains the placeholder 'type filter text'. Below it, a tree view shows the message structure:

- Mule Message
 - Payload
 - Object : Object
 - body : Object?
 - listAllFlightsResponse : Object
 - return : Object*
- Attributes
- Variables

Under 'listAllFlightsResponse', the following properties are listed:

- airlineName : String?
- code : String?
- departureDate : String?
- destination : String?
- emptySeats : Number
- origin : String?
- planeType : String?
- price : Number

31. Save the file.

Test the application

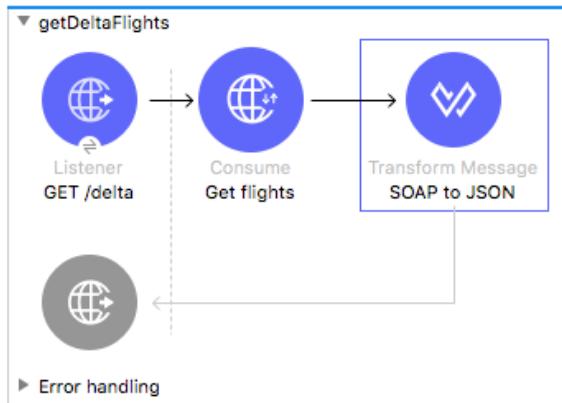
32. Save the files then stop and run the project.
33. In Advanced REST Client, return to the middle tab – the one with the localhost request.
34. Make a request to <http://localhost:8081/delta>; you should get a 200 status code and a response listing the contents of listAllFlightsResponse.

The screenshot shows the Advanced REST Client interface. At the top, there are fields for Method (GET), Request URL (<http://localhost:8081/delta>), and a blue 'SEND' button. Below this, a 'Parameters' dropdown is shown. The response section shows a green '200 OK' status with a time of '1266.10 ms'. There are 'DETAILS' and a 'More' button. The response body is displayed as XML:

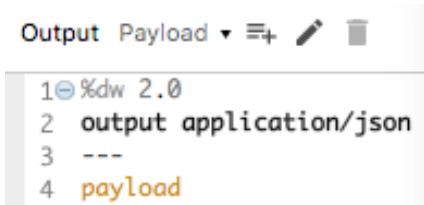
```
{  
  body:<?xml version="1.0" encoding="UTF-8"?>  
  <ns2:listAllFlightsResponse xmlns:ns2="http://soap.training.mulesoft.com/"><return><airlineName>Delta</airlineName><code>A1B2C3</code><departureDate>2015/03/20</departureDate><destination>SF0</destination><emptySeats>40</emptySeats><origin>MUA</origin><planeType>Boing 737</planeType><price>400.0</price></return><return><airlineName>Delta</airlineName><code>A1B2C4</code><departureDate>2015/02/11</departureDate><destination>LAX</destination>
```

Transform the response to JSON

35. Return to Anypoint Studio.
36. Add a Transform Message component to the end of the flow.
37. Set the display name to SOAP to JSON.



38. In the Transform Message properties view, change the output type to application/json and set the expression to payload.



Test the application

39. Save the file to redeploy the project.

40. In Advanced REST Client, make another request to <http://localhost:8081/delta>; you should get JSON returned.

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

Parameters ▾

200 OK 1699.70 ms

□ □ <> ▪▪▪

```
{
  "headers": {},
  "attachments": {},
  "body": {
    "listAllFlightsResponse": {
      "return": {
        "airlineName": "Delta",
        "code": "A1B3D4",
        "departureDate": "2015/02/12",
        "destination": "PDX",
        "emptySeats": "10",
        "origin": "MUA",
        "planeType": "Boing 777",
        "price": "385.0"
      }
    }
  }
}
```

41. Click the Toggle raw response view (or Toggle source view) button; you should see all the flights.

```
200 OK 1699.70 ms

{
  "headers": {
  },
  "attachments": {
  },
  "body": {
    "listAllFlightsResponse": {
      "return": [
        {
          "airlineName": "Delta",
          "code": "A1B2C3",
          "departureDate": "2015/03/20",
          "destination": "SFO",
          "emptySeats": "40",
          "origin": "MUA",
          "planeType": "Boing 737",
          "price": "400.0"
        },
        {
          "airlineName": "Delta",
          "code": "A1B2C4",
          "departureDate": "2015/02/11",
          ...
        }
      ]
    }
  }
}
```

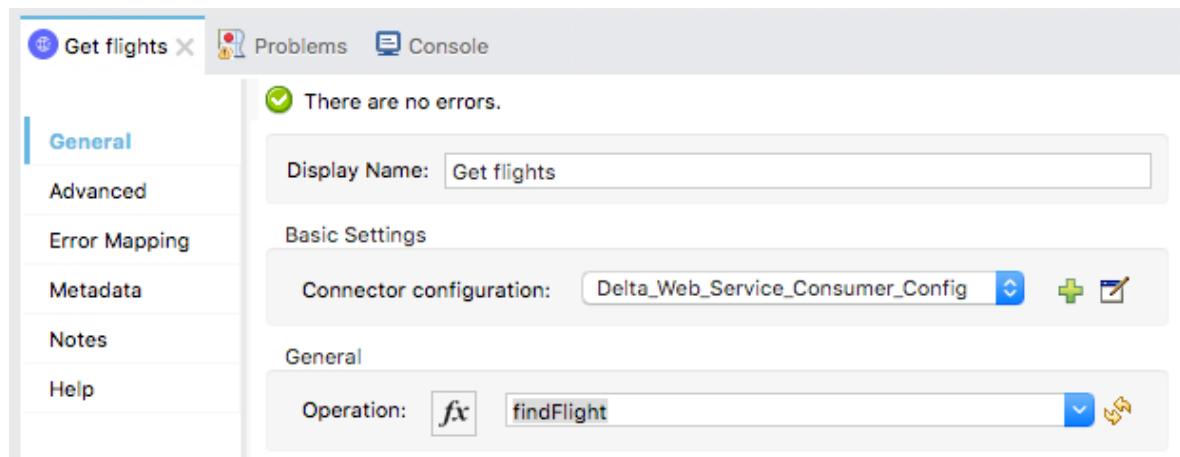
42. Add a query parameter called code and set it equal to LAX.

43. Send the request; you should still get all flights.

Call a different web service operation

44. Return to getDeltaFlights in Anypoint Studio.

45. In the properties view for Get flights Consume operation, change the operation to findFlight.



46. Save all files to redeploy the application.

Review metadata associated with the United Get flights operation response

47. Return to the Get flights properties view.

48. Select the Output tab in the DataSense Explorer and expand Payload; you should now see the payload metadata for findFlightResponse.

The screenshot shows the DataSense Explorer interface with the Output tab selected. A search bar at the top contains the placeholder "Q type filter text". Below it, the payload metadata for the findFlightResponse operation is displayed. The structure is as follows:

- Mule Message
 - Payload
 - Object : Object
 - body : Object?
 - findFlightResponse : Object
 - return : Object*?
 - airlineName : String?
 - code : String?
 - departureDate : String?
 - destination : String?
 - emptySeats : Number
 - origin : String?
 - planeType : String?
 - price : Number

49. Select the Input tab and expand Payload; you should see this operation now expects a destination.

The screenshot shows the DataSense Explorer interface with the Input tab selected. A search bar at the top contains the placeholder "Q type filter text". Below it, the payload metadata for the findFlight operation is displayed. The structure is as follows:

- Mule Message
 - Payload
 - (Actual) Any : Any
 - (Expected) Xml<#root:(http://soap.tr)
 - findFlight : Object
 - destination : String?

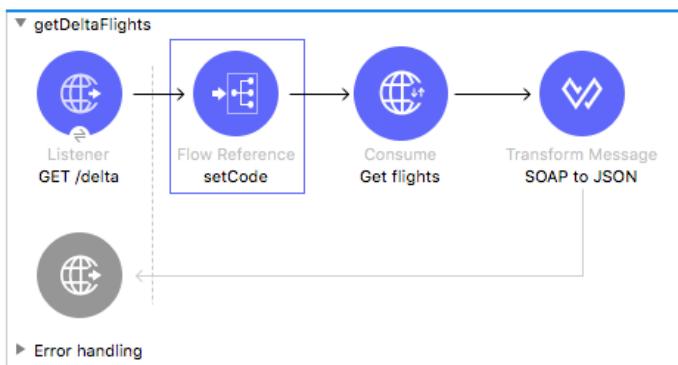
Test the application

50. In Advanced REST Client, send the same request with the query parameter; you should get a 500 Server Error with a message that the operation requires input parameters.

The screenshot shows a 500 Server Error response from the Advanced REST Client. The status code is 500 Server Error, and the time taken is 1116.10 ms. The error message is "Cannot build default body request for operation [findFlight], it requires input parameters". There are also standard browser control icons like back, forward, and search.

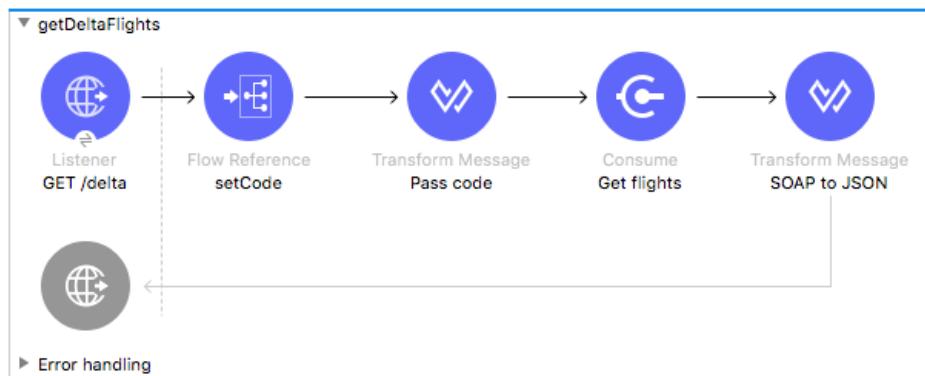
Use the set airport code subflow

51. Return to getDeltaFlights in Anypoint Studio.
52. Add a Flow Reference component after the GET /delta Listener.
53. In the Flow Reference properties view, set the flow name and display name to setCode.



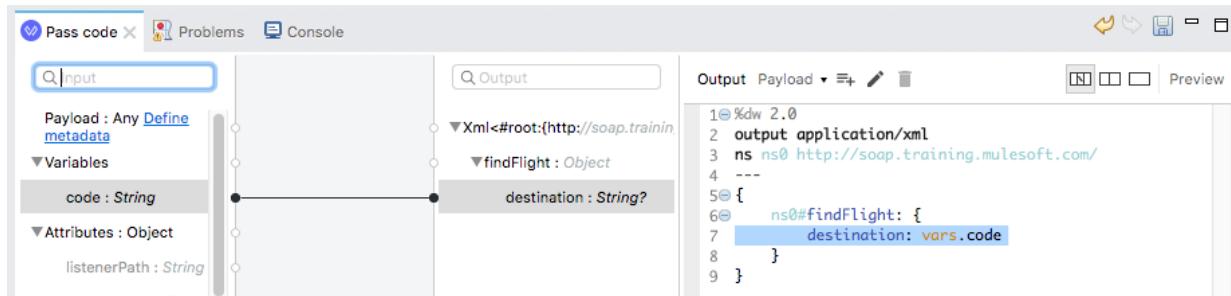
Use the Transform Message component to pass a parameter to the web service

54. Add a Transform Message component after the Flow Reference component.
55. Change its display name to Pass code.



56. In the Pass code properties view, look at the input and output sections.

57. Drag the code variable in the input section to the destination element in the output section.



Test the application

58. Save the file to redeploy the application.

59. In Advanced REST Client, make another request; you should now only see flights to LAX.

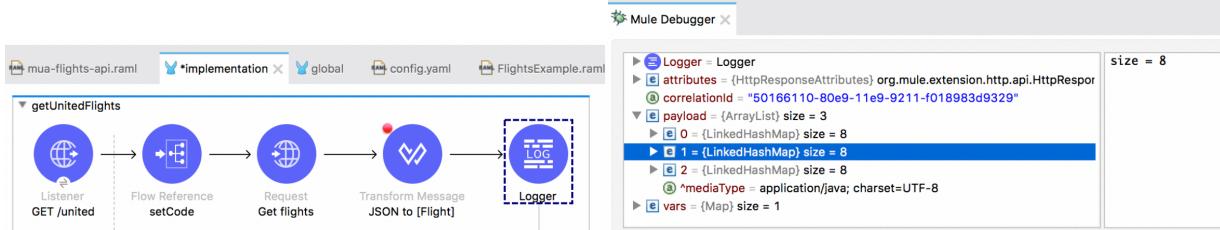
200 OK 1497.90 ms DETAILS ▾

{
 "headers": {
 },
 "attachments": {
 },
 "body": {
 "findFlightResponse": {
 "return": {
 "airlineName": "Delta",
 "code": "A1B2C4",
 "departureDate": "2015/02/11",
 "destination": "LAX",
 "emptySeats": "10",
 "origin": "MUA",
 "planeType": "Boing 737",
 "price": "199.99"
 },
 "return": {
 "airlineName": "Delta",
 "code": "A134DS",
 "departureDate": "2015/04/11",
 "destination": "LAX",
 "emptySeats": "40",
 "origin": "MUA",
 "planeType": "Boing 737",
 "price": "199.99"
 }
 }
 }
}

Walkthrough 8-4: Transform data from multiple services to a canonical format

In this walkthrough, you will transform the JSON returned from the American and United web services and the SOAP returned from the Delta web service to the same format. You will:

- Define a metadata type for the Flight Java class.
- Transform the results from RESTful and SOAP web service calls to a collection of Flight objects.



Review Flight.java

1. Return to apdev-flights-ws in Anypoint Studio.
2. Open Flight.java in src/main/java and review the file.

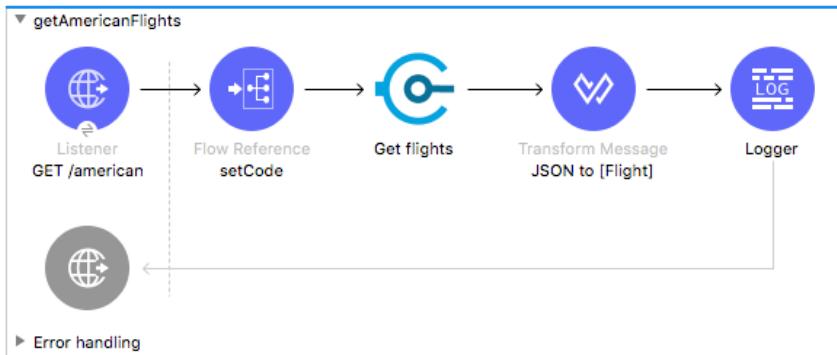


3. Close the file.

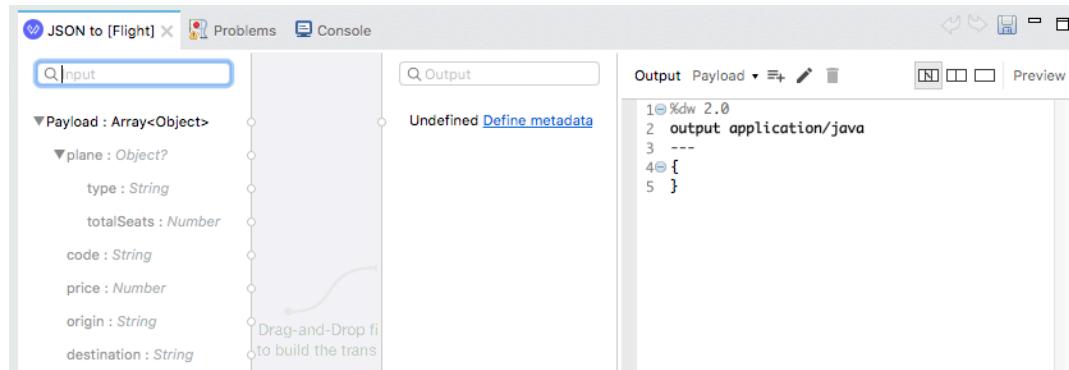
Change the American flow to return Java Flight objects instead of JSON

4. Return to getAmericanFlights in implementation.xml.
5. Add a Transform Message component to the end of the flow.
6. Add a Logger at the end of the flow.

7. Change the name of the Transform Message component to JSON to [Flight].



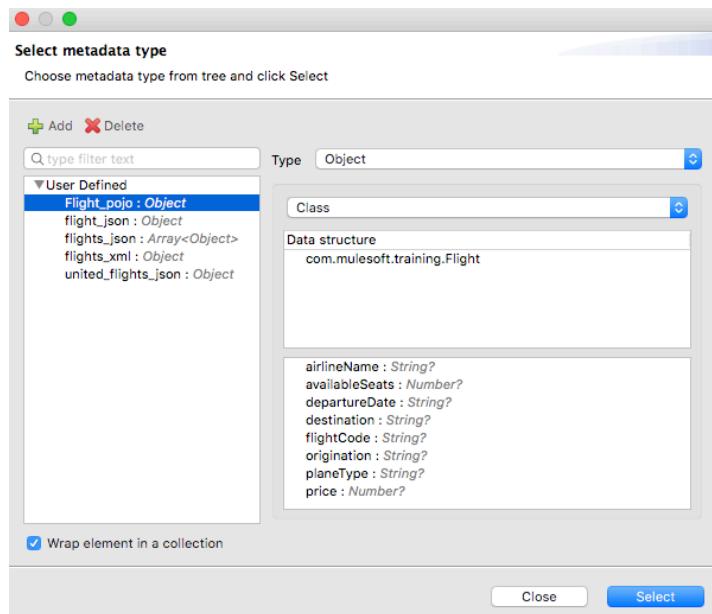
8. In the Transform Message properties view, look at the input section in the Transform Message properties view; you should see metadata already defined.



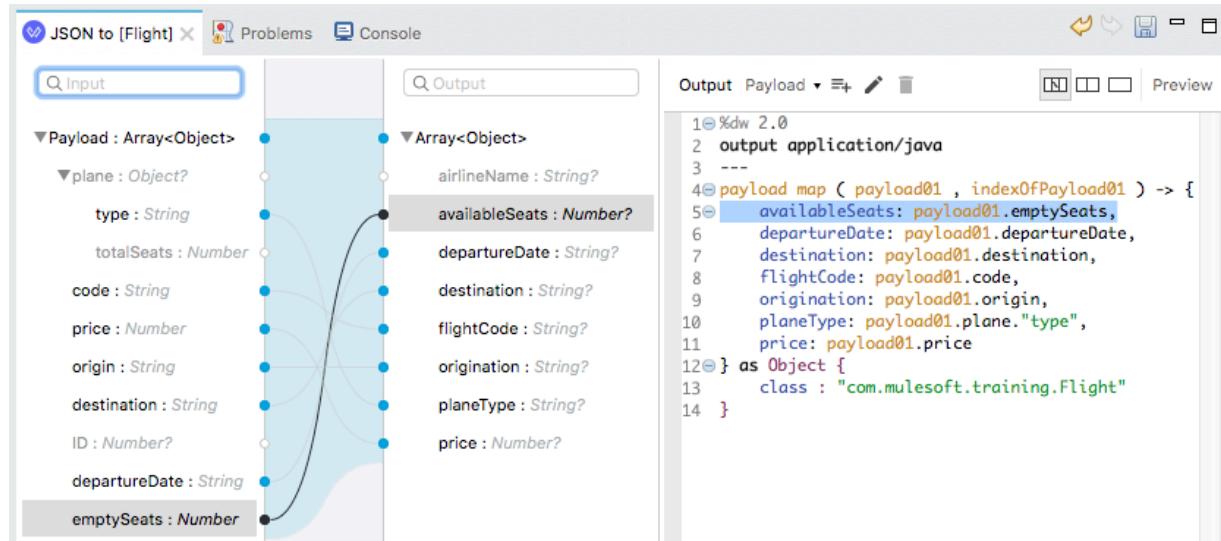
9. In the output section of the Transform Message properties view, click the Define metadata link.

10. In the Select metadata type dialog box, select the user-defined Flight_pojo.

11. Select Wrap element in a collection in the lower-left corner.



12. Click Select; you should now see output metadata in the output section of the Transform Message properties view.
13. Map fields (except ID and totalSeats) by dragging them from the input section and dropping them on the corresponding field in the output section.



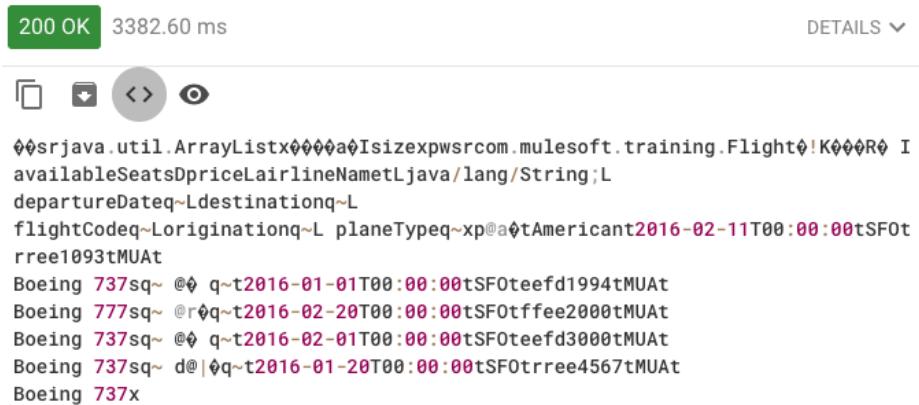
14. Double-click the airlineName field in the output section.
15. In the generated DataWeave expression, change the airlineName value from null to "American".

```
%dw 2.0
output application/java
---
payload map ( payload01 , indexOfPayload01 ) ->
    airlineName: "American",
    availableSeats: payload01.emptySeats.
```

Test the application

16. Save to redeploy the project.

17. In Advanced REST Client, change the URL and make a request to <http://localhost:8081/american>; you should see a representation of a collection of Java objects.



200 OK 3382.60 ms DETAILS ▾

srjava.util.ArrayList@1a91size=1 com.mulesoft.training.Flight@1K00R0 I
availableSeatsDpriceLairlineNameLjava/lang/String;L
departureDateq~Ldestinationq~L
flightCodeq~Loriginationq~L planeTypeq~xp@a@tAmericant2016-02-11T00:00:00tSF0t
rree1093tMUAT
Boeing 737sq~ @q~t2016-01-01T00:00:00tSF0teefd1994tMUAT
Boeing 777sq~ @r@q~t2016-02-20T00:00:00tSF0tfee2000tMUAT
Boeing 737sq~ @q~t2016-02-01T00:00:00tSF0teefd3000tMUAT
Boeing 737sq~ d@|@q~t2016-01-20T00:00:00tSF0trree4567tMUAT
Boeing 737x

Debug the application

18. Return to Anypoint Studio.
19. Stop the project.
20. Add a breakpoint to the Get flights operation.
21. Debug the project.
22. In Advanced REST Client, make another request to <http://localhost:8081/american>.
23. In the Mule Debugger, step to the Transform Message component and examine the payload.



24. Step to the Logger and look at the payload it should be a collection of Flight objects.

The screenshot shows the Mule Debugger interface. At the top, there's a tree view with nodes like 'Logger = Logger', 'attributes = {HttpServletResponseAttributes} org.mule.extension.http.api.HttpResponse', and 'payload = {ArrayList} size = 5'. Under 'payload', five flight objects are listed: '0 = {Flight} com.mulesoft.training.Flight@6ac734f3', '1 = {Flight} com.mulesoft.training.Flight@fe1c435', '2 = {Flight} com.mulesoft.training.Flight@2003dfc8', '3 = {Flight} com.mulesoft.training.Flight@12dd3302', and '4 = {Flight} com.mulesoft.training.Flight@41eefd06'. Below this, there's a note about media type: '^mediaType = application/java; charset=UTF-8'. At the bottom, there's a 'vars' node with a size of 1. Below the debugger, the application navigation bar shows 'mua-flights-api.raml', 'implementation', 'global', 'config.yaml', and 'FlightsExample.raml'. The 'implementation' tab is selected. On the right, a flow diagram for the 'getAmericanFlights' endpoint is visible, consisting of a Listener, Flow Reference, Get flights, Transform Message, and a final step labeled 'Logger'.

25. Step through the rest of the application and switch perspectives.

Change the United airline flows to return Java Flight objects instead of JSON

26. Return to getUnitedFlights in implementation.xml.

27. Add a Transform Message component at the end of the flow.

28. Change the name of the Transform Message component to JSON to [Flight].

29. Add a Logger to the end of the flow.



30. In the Transform Message properties view, look at the input section; you should see metadata already defined.

31. In the output section of the Transform Message properties view, click the Define metadata link.

32. In the Select metadata type dialog box, select the user-defined Flight_pojo.
 33. Select Wrap element in a collection in the lower-left corner.
 34. Click Select; you should now see output metadata in the output section of the Transform
- Message properties view.

35. Map fields by dragging them from the input section and dropping them on the corresponding field in the output section.

Note: If you do not see the object type set to com.mulesoft.training.Flight at the end of the DataWeave code, return to the course snippets.txt file and copy the DataWeave code to transform an object to a custom data type, and paste it at the end of the expression.

Debug the application

36. Add a breakpoint to the Transform Message component.
37. Save the files to redeploy the project.
38. In Advanced REST Client, change the URL to make a request to <http://localhost:8081/united>.

39. In the Mule Debugger, examine the payload.

The screenshot shows the Mule Debugger interface with the title bar "Mule Debugger". The left pane displays a tree view of variables:

- Transform = JSON to [Flight]
- attributes = {HttpResponseAttributes} org.mule.extension.htt
- correlationId = "68a1a181-80e5-11e9-9211-f018983d932"
- payload** = {"flights": [{"code": "ER38sd", "price": 400, "origin": "MUA", "destination": "SFO", "departureDate": "2015/03/20", "planeType": "Boeing 737", "airlineName": "United", "emptySeats": 0}, {"code": "ER39rk", "price": 945, "origin": "MUA", "destination": "SFO", "departureDate": "2015/09/11", "planeType": "Boeing 757", "airlineName": "United", "emptySeats": 54}, {"code": "ER39rj", "price": 954, "origin": "MUA", "destination": "SFO", "departureDate": "2015/02/12", "planeType": "Boeing 777", "airlineName": "United", "emptySeats": 23}]}
vars = {Map} size = 1

The right pane shows the raw JSON payload:

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

40. Step to the Logger and look at the payload it should be a collection of Flight objects.

The screenshot shows the Mule Debugger interface with the title bar "Mule Debugger". The left pane displays a tree view of variables:

- Logger = Logger
- attributes = {HttpResponseAttributes} org.mule.extension.htt
- correlationId = "d8345c40-aa70-11e9-ba22-f018983d932"
- payload** = {ArrayList} size = 3
 - 0 = {Flight} com.mulesoft.training.Flight@7b4aa0a0
 - 1 = {Flight} com.mulesoft.training.Flight@76e8ef74
 - 2 = {Flight} com.mulesoft.training.Flight@621100b5
- mediaType = application/java; charset=UTF-8
- vars = {Map} size = 1

The right pane shows the raw JSON payload:

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

The bottom navigation bar shows the current perspective is "implementation". Other perspectives include "global", "config.yaml", and "FlightsExample.raml".

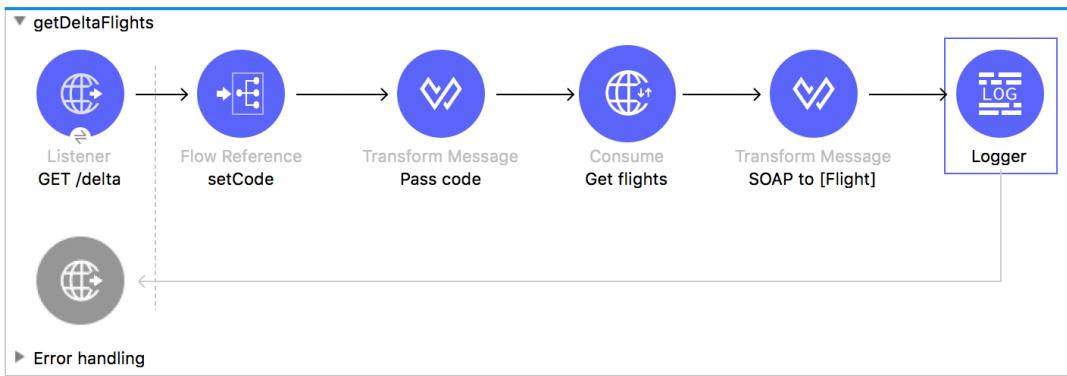
41. Step through the rest of the application and switch perspectives.

Change the Delta flow to return Java Flight objects

42. Return to geDeltaFlights in implementation.xml.

43. Change the name of the SOAP to JSON Transform Message component to SOAP to [Flight].

44. Add a Logger to the end of the flow.



45. Look at the input section in the SOAP to [Flight] Transform Message properties view; you should see metadata already defined.

46. In the output section of the Transform Message properties view, click the Define metadata link.

47. In the Select metadata type dialog box, select the user-defined Flight_pojo.

48. Select Wrap element in a collection in the lower-left corner.

49. Click Select; you should now see output metadata in the output section of the Transform Message properties view.

50. Map the fields by dragging them from the input section and dropping them on the corresponding field in the output section.

```
1<!> <?xml version="1.0"?>
2<!-->
3<!-->
4<!-->
5<!-->
6<!-->
7<!-->
8<!-->
9<!-->
10<!-->
11<!-->
12<!-->
13<!-->
14<!-->
15<!-->
16<!-->
```

Debug the application

51. Add a breakpoint to the SOAP to [Flight] Transform Message component.

52. Save the file to redeploy the project.

53. In Advanced REST Client, change the URL to make a request to <http://localhost:8081/delta>.

54. In the Mule Debugger, examine the payload.

The screenshot shows the Mule Debugger interface with the title bar "Mule Debugger". The left pane displays a tree view of the payload structure:

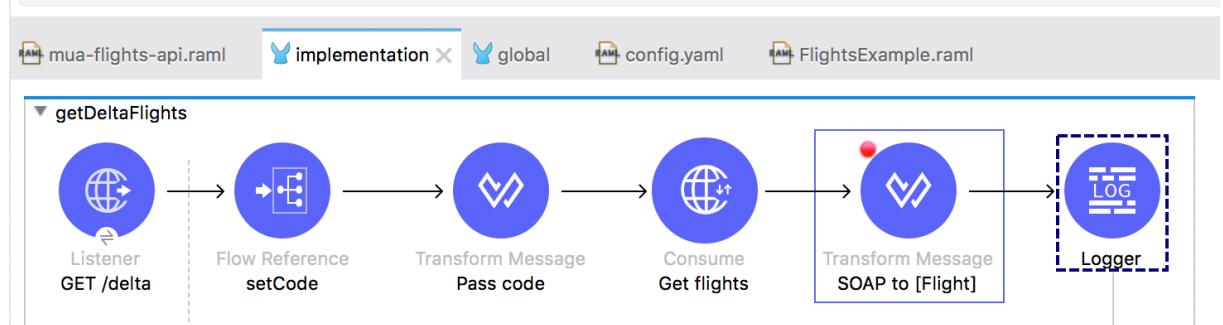
- Transform = SOAP to [Flight]
- e attributes = {SoapAttributes} {
additionalTransportData = [
reasonPh
correlationId = "32f41e41-80eb-11e9-9211-f018983d9329"
e payload = {SoapOutputEnvelope} {
nbody:<?xml version="1.0" encoding="UTF-8?
ns2:findFlightResponse xmlns:ns2="http://airlineName<code>A1B2C3</code><depart destination><emptySeats>40</emptySeats price></return><return><airlineName>D
departureDate><destination>SFO</destination>777</planeType><price>593.0</price></r code><departureDate>2015/02/12</departureDate><emptySeats><origin>MUA</origin><planeType>777</planeType><price>593.0</price></r headers=<headers>ns2:findFlightResponse>,
headers. □

55. Step to the Logger and look at the payload it should be a collection of Flight objects.

The screenshot shows the Mule Debugger interface with the title bar "Mule Debugger". The left pane displays a tree view of the payload structure:

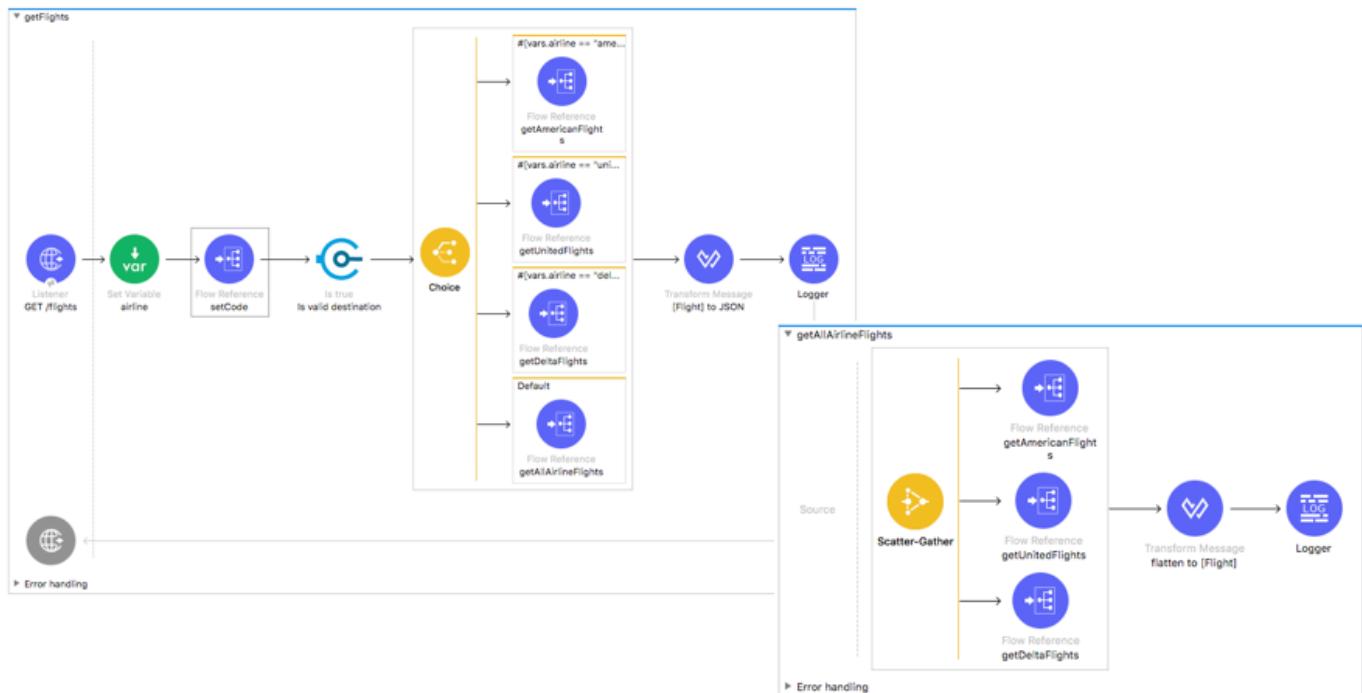
- Logger = Logger
- e attributes = {SoapAttributes} {
additionalTransportData = [
reasonPh
correlationId = "32f41e41-80eb-11e9-9211-f018983d9329"
e payload = {ArrayList} size = 3
 - 0 = {Flight} com.mulesoft.training.Flight@3f50a3c3
 - 1 = {Flight} com.mulesoft.training.Flight@aa37472
 - 2 = {Flight} com.mulesoft.training.Flight@6fd0c7af
e vars = {Map} size = 1

The right pane shows the value of the payload: `com.mulesoft.training.Flight@3f50a3c3`.



56. Step through the rest of the application and switch perspectives.

Module 9: Controlling Event Flow



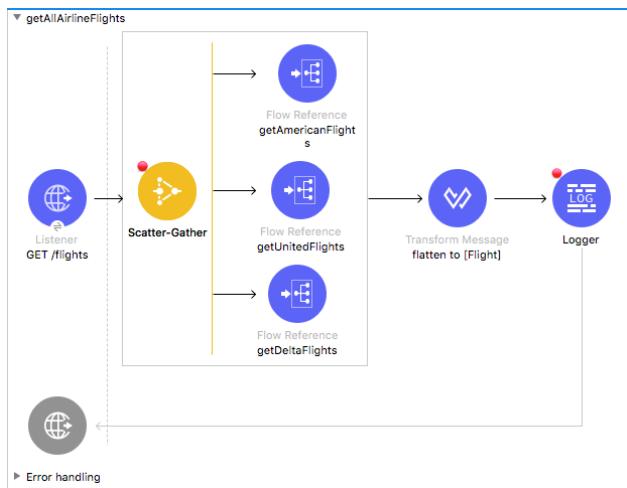
At the end of this module, you should be able to:

- Multicast events.
- Route events based on conditions.
- Validate events.

Walkthrough 9-1: Multicast an event

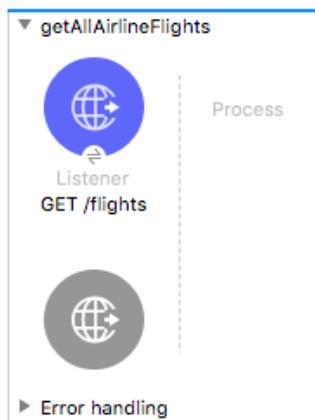
In this walkthrough, you create a flow that calls each of the three airline services and combines the results. You will:

- Use a Scatter-Gather router to concurrently call all three flight services.
- Use DataWeave to flatten multiple collections into one collection.



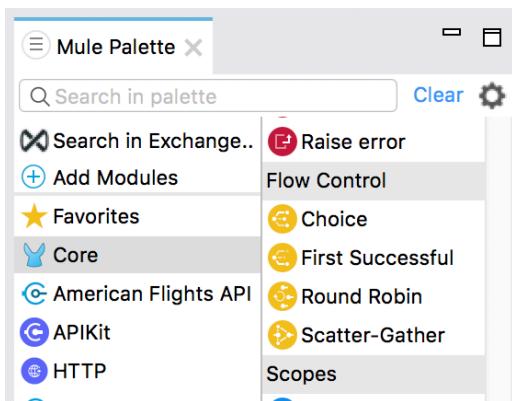
Create a new flow

1. Return to implementation.xml.
2. From the Mule Palette, drag an HTTP Listener and drop it at the top of the canvas.
3. Change the flow name to getAllAirlineFlights.
4. In the Listener properties view, set the display name to GET /flights.
5. Ensure the connector configuration is set to the existing HTTP_Listener_config.
6. Set the path to /flights and the allowed methods to GET.



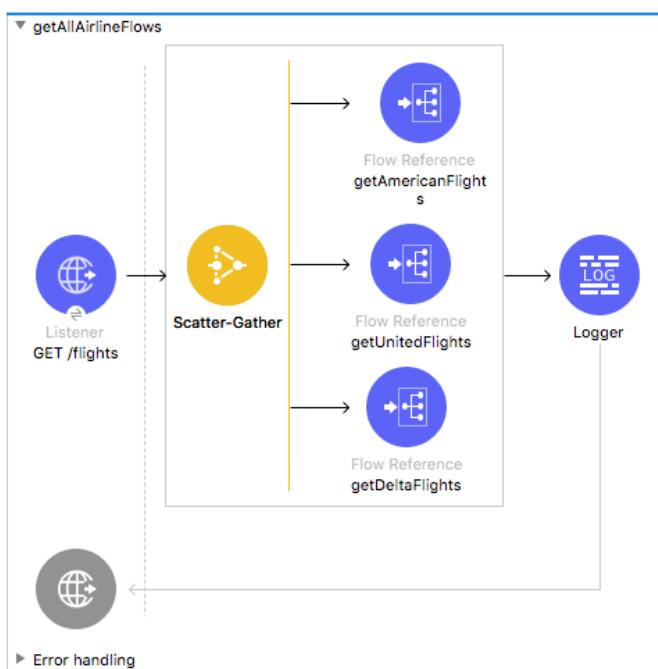
Browse the flow control elements in the Mule Palette

7. In the Core section of the Mule Palette, locate the Flow Control elements.



Add a Scatter-Gather to call all three airline services

8. Drag a Scatter-Gather flow control element from the Mule Palette and drop it in the process section of getAllAirlineFlights.
9. Add three parallel Flow Reference components to the Scatter-Gather router.
10. In the first Flow Reference properties view, set the flow name and display name to getAmericanFlights.
11. Set the flow name and display name of the second Flow Reference to getUnitedFlights.
12. Set the flow name and display name of the third Flow Reference to getDeltaFlights.
13. Add a Logger after the Scatter-Gather.



Review the metadata for the Scatter-Gather output

14. In the Logger properties view, explore the input payload structure in the DataSense Explorer.

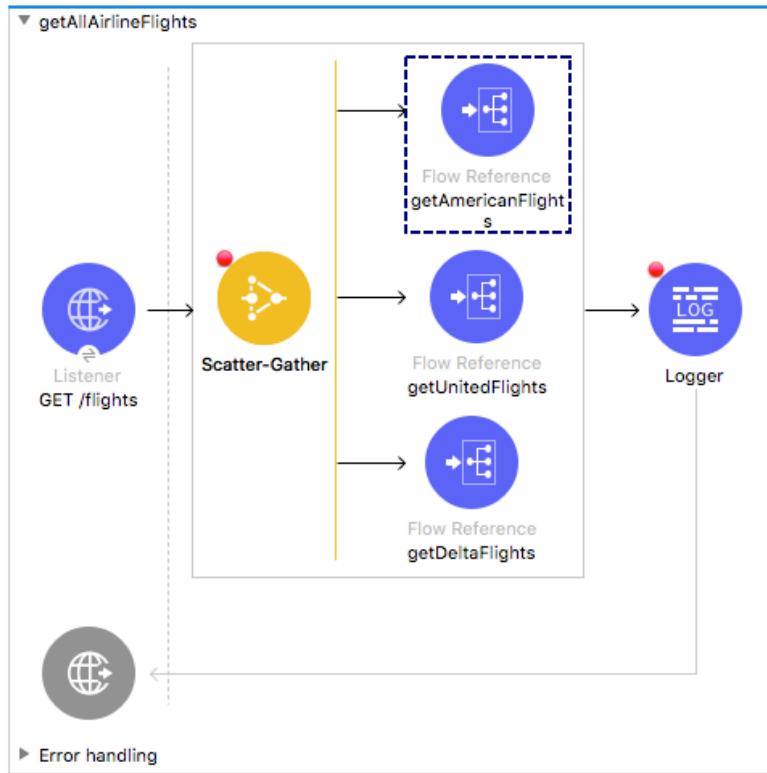
The screenshot shows the DataSense Explorer interface with the 'Input' tab selected. A search bar at the top contains the placeholder 'Q type filter text'. Below it, the 'Mule Message' section is expanded, showing the 'Payload' structure. The payload is defined as an array of objects, indexed from 0 to 2. Each object has a 'payload' field (an array of objects) and an 'attributes' field (an object). The first object's payload array contains fields: airlineName, availableSeats, departureDate, destination, flightCode, origination, planeType, and price. The second object's payload array contains the same set of fields. The third object's payload array is currently empty. The 'Attributes' and 'Variables' sections are also visible but contain no data.

```
Input Output
Q type filter text
▼ Mule Message
  ▼ Payload
    ▶ Object : Object
      ▶ 0 : Object
      ▶ 1 : Object
        ▶ payload : Array<Object>
        ▶ attributes : Object
      ▶ 2 : Object
        ▶ payload : Array<Object>
          airlineName : String?
          availableSeats : Number?
          departureDate : String?
          destination : String?
          flightCode : String?
          origination : String?
          planeType : String?
          price : Number?
        ▶ attributes : Object
    ▼ Attributes
      Void : Void
  ▼ Variables
    ▼ code
      String : String
```

Debug the application

15. Add a breakpoint to the Scatter-Gather.
16. Add a breakpoint to the Logger.
17. Save the file to redeploy the project in debug mode.
18. In Advanced REST Client, change the URL to make a request to <http://localhost:8081/flights>.

19. In the Mule Debugger, step through the application; you should step through each of the airline flows.



20. Stop at the Logger after the Scatter-Gather and explore the payload.

A screenshot of the Mule Debugger interface. The top part shows a list of variables in the current scope:

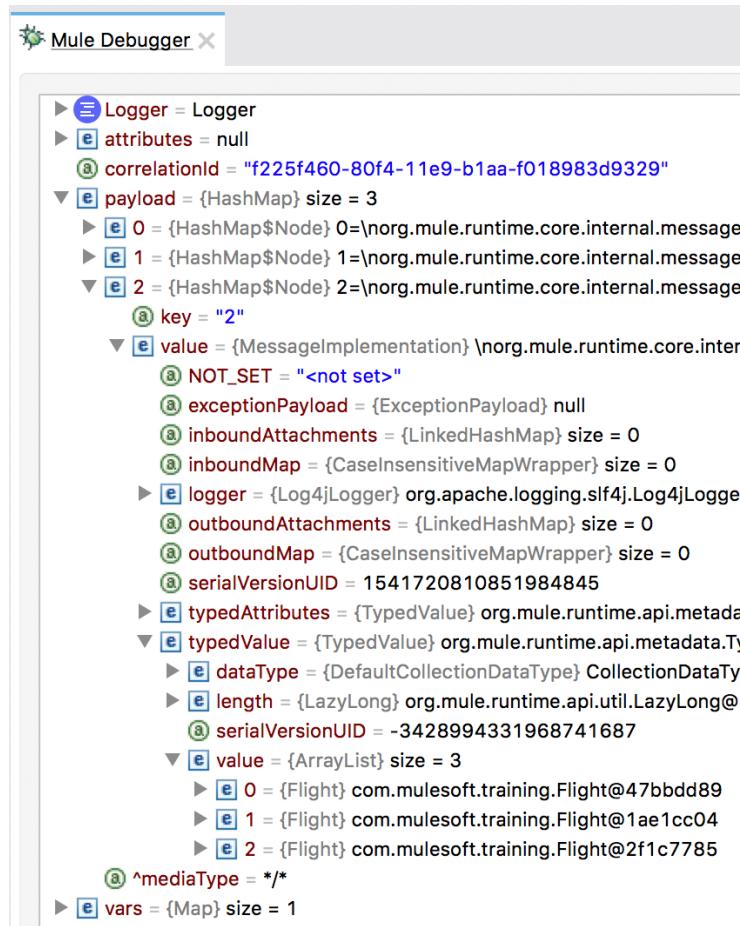
- Logger = Logger
- attributes = null
- correlationId = "190e0bf1-80f3-11e9-9211-f018983d9329"
- payload = {HashMap} size = 3** (highlighted in blue)
- vars = {Map} size = 1

The value for "payload" is shown as "size = 3". Below the debugger, the Mule flow is displayed:

```
mua-flights-api.raml implementation global config.yaml FlightsExample
```

The flow consists of a "Listener GET /flights" followed by a "Scatter-Gather" component. Three parallel arrows from the "Scatter-Gather" lead to three "Flow Reference" components: "getUnitedFlights" and two others (partially visible). Each "Flow Reference" has an arrow pointing to a "Logger" component. A "LOG" icon is present next to the "Logger" components.

21. Drill-down into one of the objects in the payload.



The screenshot shows the Mule Debugger interface with a tree view of object properties. The root node is a logger object. The payload is expanded to show three entries (0, 1, 2) which are instances of `norg.mule.runtime.core.internal.message`. Each entry has a key ("2") and a value (MessageImplementation). The value implementation has several attributes: NOT_SET, exceptionPayload, inboundAttachments, inboundMap, logger, outboundAttachments, outboundMap, and serialVersionUID. The value is also an ArrayList containing three Flight objects. The vars map contains one entry.

```
► [Logger] = Logger
► [attributes] = null
@ correlationId = "f225f460-80f4-11e9-b1aa-f018983d9329"
▼ [payload] = {HashMap} size = 3
► [0] = {HashMap$Node} 0=\norg.mule.runtime.core.internal.message
► [1] = {HashMap$Node} 1=\norg.mule.runtime.core.internal.message
▼ [2] = {HashMap$Node} 2=\norg.mule.runtime.core.internal.message
@ key = "2"
▼ [value] = {MessageImplementation} \norg.mule.runtime.core.inter
@ NOT_SET = "<not set>"
@ exceptionPayload = {ExceptionPayload} null
@ inboundAttachments = {LinkedHashMap} size = 0
@ inboundMap = {CaseInsensitiveMapWrapper} size = 0
► [logger] = {Log4jLogger} org.apache.logging.slf4j.Log4jLogge
@ outboundAttachments = {LinkedHashMap} size = 0
@ outboundMap = {CaseInsensitiveMapWrapper} size = 0
@ serialVersionUID = 1541720810851984845
► [typedAttributes] = {TypedValue} org.mule.runtime.api.metada
▼ [typedValue] = {TypedValue} org.mule.runtime.api.metadata.Ty
► [dataType] = {DefaultCollectionDataType} CollectionDataTy
► [length] = {LazyLong} org.mule.runtime.api.util.LazyLong@
@ serialVersionUID = -3428994331968741687
▼ [value] = {ArrayList} size = 3
► [0] = {Flight} com.mulesoft.training.Flight@47bbdd89
► [1] = {Flight} com.mulesoft.training.Flight@1ae1cc04
► [2] = {Flight} com.mulesoft.training.Flight@2f1c7785
@ ^mediaType = */
► [vars] = {Map} size = 1
```

22. Step through the rest of the application and switch perspectives.

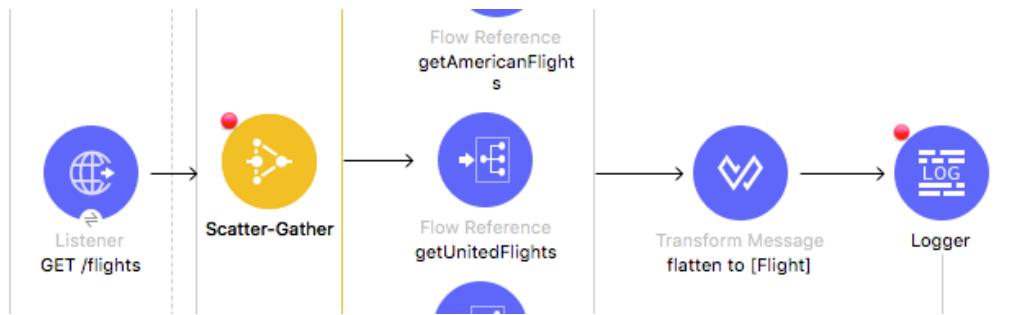
23. Return to Advanced REST Client and review the response; you should get a 500 Server Error with a message that the object could not be serialized.

Flatten the combined results

24. Return to Anypoint Studio.

25. In getAllAirlineFlights, add a Transform Message component before the Logger.

26. Change the display name to flatten to [Flight].



27. In the input section of the Transform Message properties view, review the payload data structure.

A screenshot of the 'flatten to [Flight]' Transform Message properties view. The title bar says 'flatten to [Flight] X Problems'. Below is a search bar with 'Input'. The main content shows the payload structure:

```
▼ Payload : Object
  ▶ 0 : Object
  ▶ 1 : Object
  ▶ 2 : Object
    ▼ payload : Array<Object>
      airlineName : String?
      availableSeats : Number?
      departureDate : String?
      destination : String?
      flightCode : String?
      origination : String?
```

28. In the expression section, use the DataWeave flatten function to flatten the collection of objects into a single collection.

```
1 @%dw 2.0
2   output application/java
3   ---
4   flatten(payload..payload)
```

Debug the application

29. Save the file to redeploy the project.

30. In Advanced REST Client, make the same request to <http://localhost:8081/flights>.

31. In the Mule Debugger, press Resume until you are stopped at the Logger at the end of the Scatter-Gather; you should see the payload is now one ArrayList of Flights.

The screenshot shows the Mule Debugger interface. At the top, it says "Mule Debugger". Below that, there's a tree view of variables:

- Logger = Logger
- attributes = null
- correlationId = "e70e9d20-aae8-11e9-8cd9-f018983d9329"
- payload = {ArrayList} size = 11** (This item is expanded, indicated by a blue bar)
- 0 = {Flight} com.mulesoft.training.Flight@7bc6c3df
- 1 = {Flight} com.mulesoft.training.Flight@201dfa86
- 2 = {Flight} com.mulesoft.training.Flight@3a5a1b98
- 3 = {Flight} com.mulesoft.training.Flight@7492eec
- 4 = {Flight} com.mulesoft.training.Flight@4adccde1
- 5 = {Flight} com.mulesoft.training.Flight@7d50c689
- 6 = {Flight} com.mulesoft.training.Flight@7f187ba6
- 7 = {Flight} com.mulesoft.training.Flight@321a86a3
- 8 = {Flight} com.mulesoft.training.Flight@3ede5340
- 9 = {Flight} com.mulesoft.training.Flight@75690d77
- 10 = {Flight} com.mulesoft.training.Flight@62a75312
- ^mediaType = application/java; charset=UTF-8

At the bottom, there's a flow diagram titled "implementation":

```
graph LR; Listener((Listener  
GET /flights)) --> ScatterGather((Scatter-Gather)); ScatterGather --> FlowRef((Flow Reference  
getUnitedFlights)); FlowRef --> Transform((Transform Message  
flatten to [Flight])); Transform --> Logger((Logger));
```

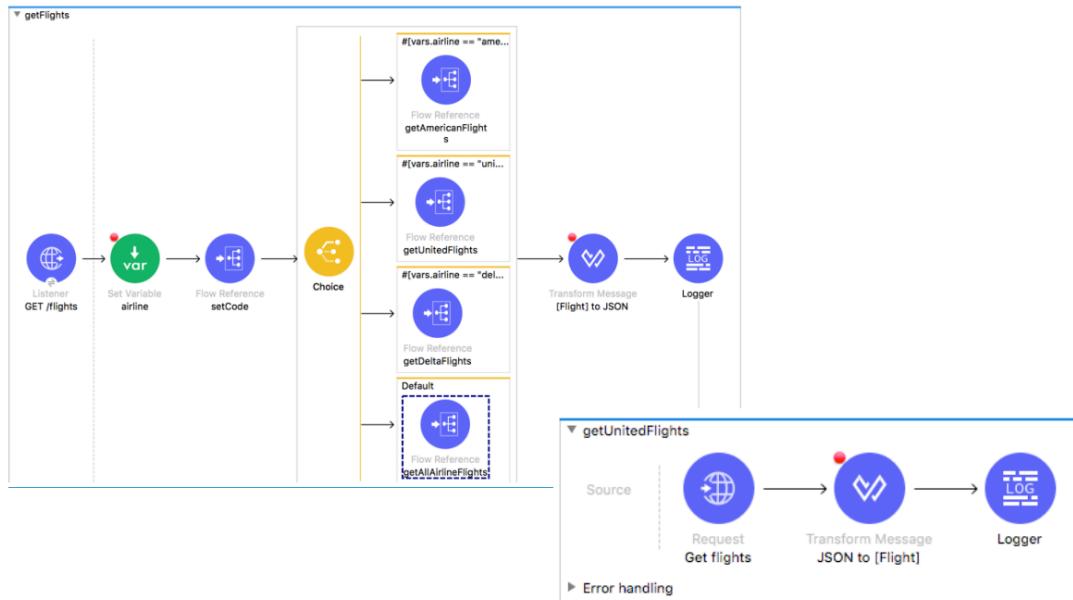
The "implementation" tab is selected. Other tabs include "mua-flights-api.raml", "global", "config.yaml", and "FlightsExample.raml".

32. Step through the rest of the application and switch perspectives.

Walkthrough 9-2: Route events based on conditions

In this walkthrough, you create a flow to route events to either the American, United, Delta, or get all airline flows based on the value of an airline query parameter. You will:

- Use a Choice router.
- Use DataWeave expressions to set the router paths.
- Route all flight requests through the router.



Look at possible airline values specified in the API

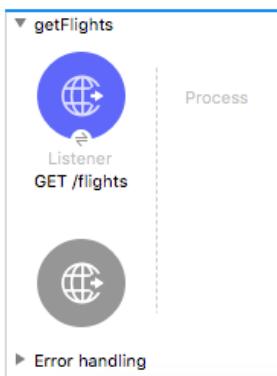
1. Return to the apdev-flights-ws project in Anypoint Studio.
2. Return to mua-flights-api.raml in src/main/resources/api.
3. Locate the airline query parameter and its possible values.

```
airline:  
  displayName: Airline  
  required: false  
  enum:  
    - united  
    - delta  
    - american
```

Create a new flow

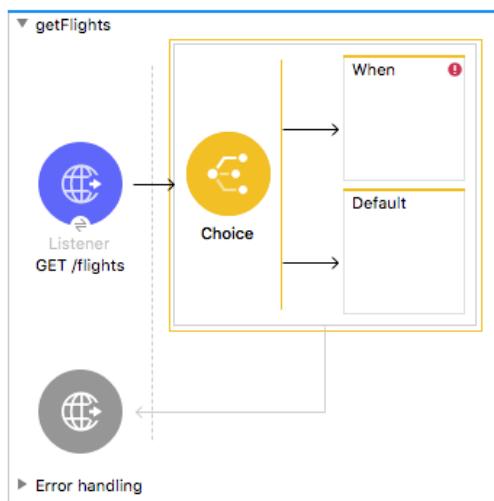
4. Return to implementation.xml.

5. Drag a Flow scope from the Mule Palette and drop it at the top of the canvas above all the other flows.
6. Change the name of the flow to getFlights.
7. Move the GET /flights HTTP Listener from getAllAirlineFlights to the source section of getFlights.



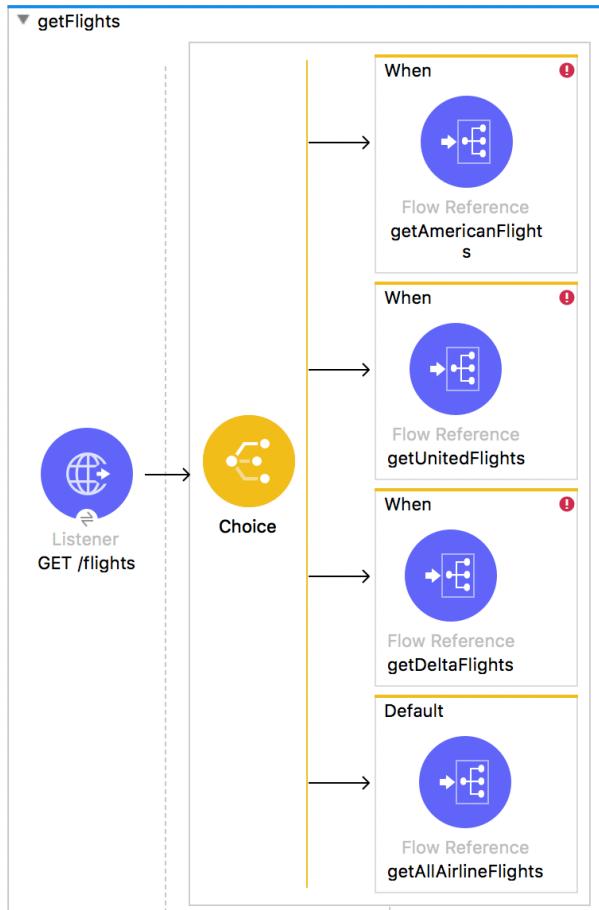
Add a Choice router

8. Drag a Choice flow control element from the Mule Palette and drop it in process section of getFlights.



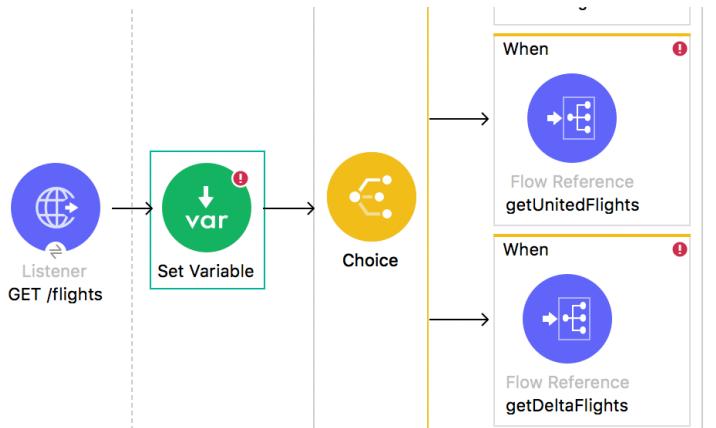
9. Add three parallel Flow Reference components to the Choice router.
10. Add a Flow Reference component to the default branch of the router.
11. In the first Flow Reference properties view, set the flow name and display name to getAmericanFlights.
12. Set the flow names and display names for the other two Flow References to getUnitedFlights and getDeltaFlights.

13. For the Flow Reference in the default branch, set the flow name and display name to getAllAirlineFlights.



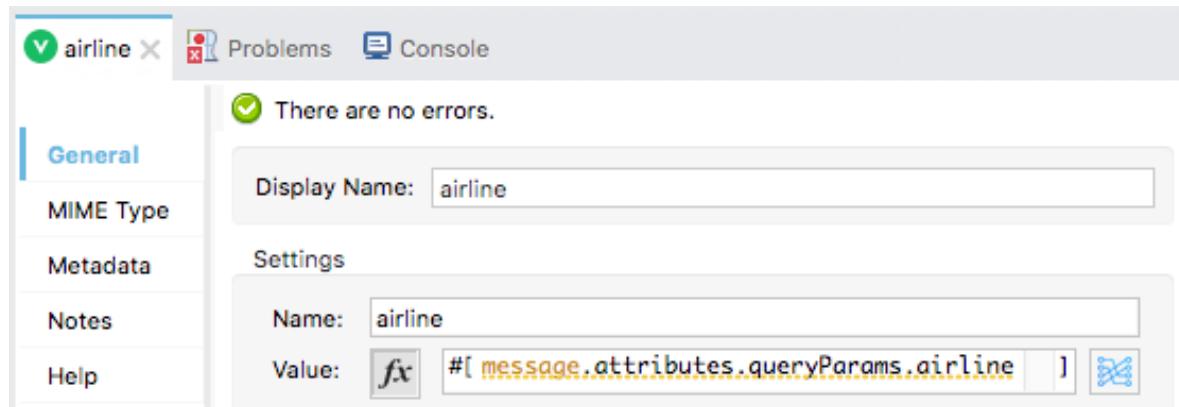
Store the airline query parameter in a variable

14. Add a Set Variable transformer before the Choice router.



15. In the Set Variable properties view, set the name and display name to airline.
16. Switch the value field to expression mode then set its value to a query parameter called airline.

`message.attributes.queryParams.airline`



Configure the Choice router

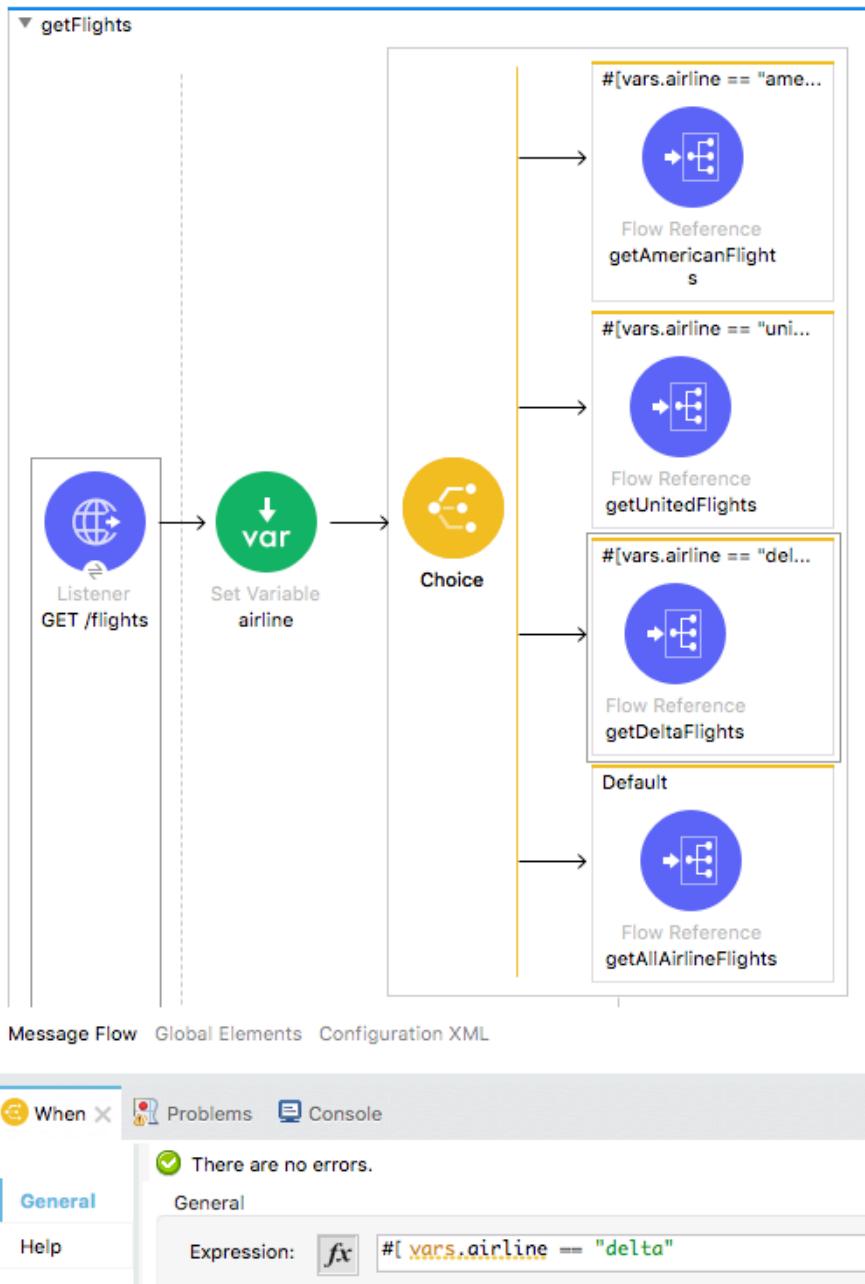
17. In the Choice router getAmericanFlights flow reference branch, click the When scope.
18. In the When properties view, switch the value field to expression mode, and then add an expression to check if the airline variable is equal to american.

`vars.airline == "american"`



19. Set a similar expression for the United route, routing to it when `vars.airline` is equal to united.

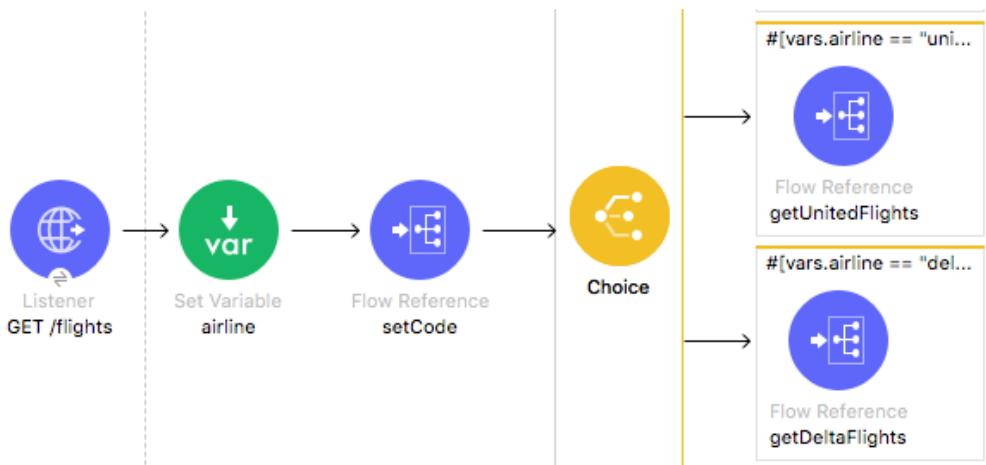
20. Set a similar expression for the Delta route, routing to it when vars.airline is equal to delta.



Route all requests through the router

21. Add a Flow Reference to the flow before the Choice router.

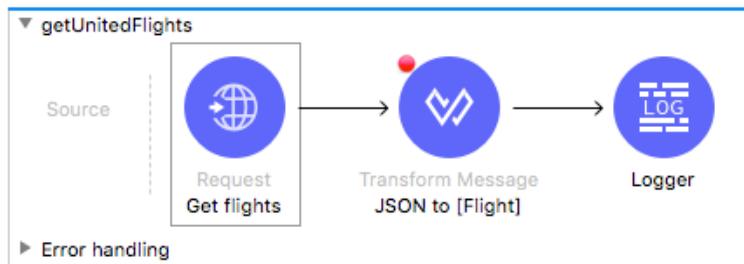
22. Set the flow name and display name to setCode.



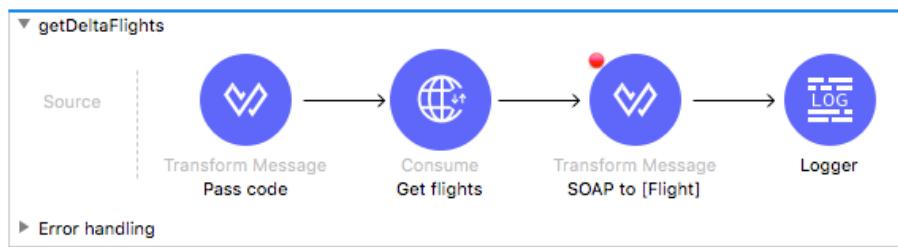
23. In getAmericanFlights, delete the HTTP Listener and the setCode Flow Reference.



24. In getUnitedFlights, delete the HTTP Listener and the setCode Flow Reference.

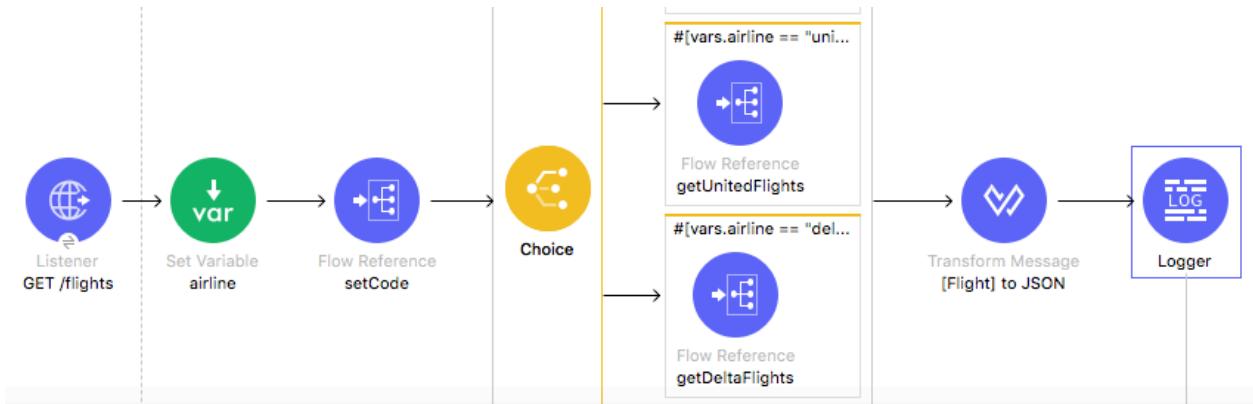


25. In getDeltaFlights, delete the HTTP Listener and the setCode Flow Reference.



Return JSON from the flow

26. In the getFlights flow, add a Transform Message component after the Choice router.
27. Change its display name to [Flight] to JSON.
28. Add a Logger at the end of the flow.



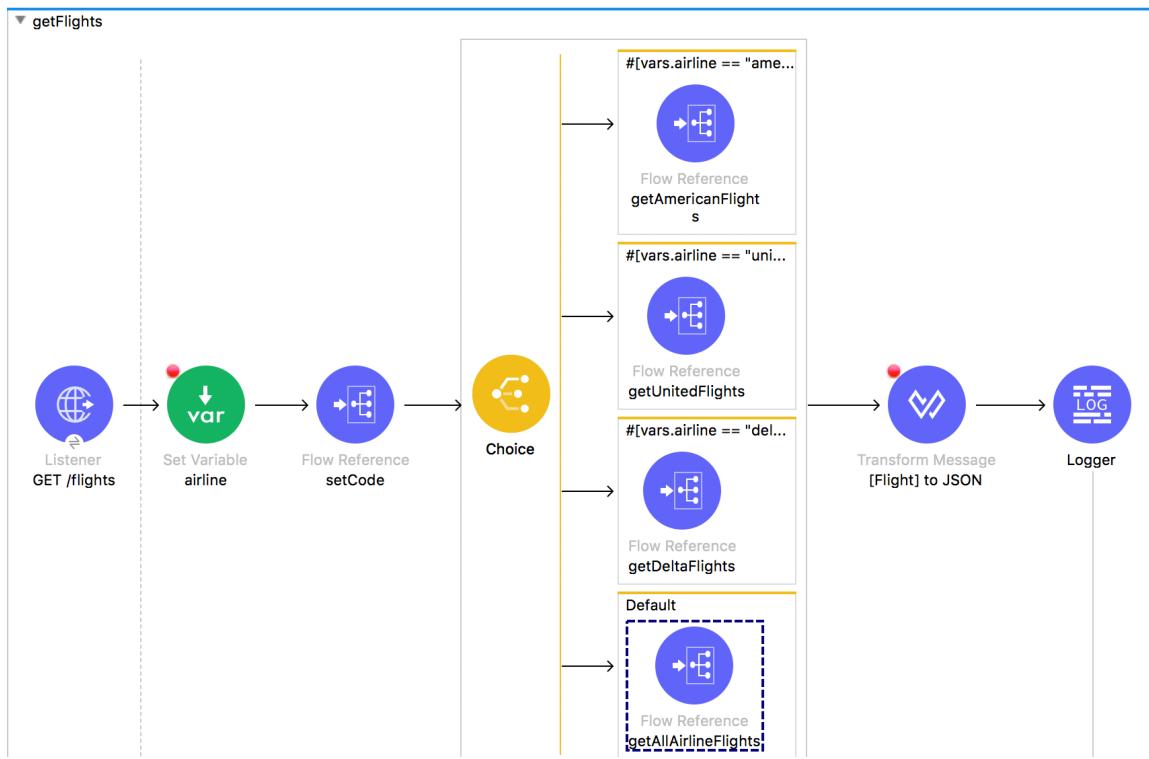
29. In the Transform Message properties view, change the output type to application/json and set the expression to payload.

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

Debug the application

30. Add a breakpoint to the airline Set Variable transformer at the beginning of getFlights.
31. Add a breakpoint to the [Flight] to JSON Transform Message component after the Choice router.
32. Save the file to redeploy the project in debug mode.
33. In Advanced REST Client, make a request to <http://localhost:8081/flights>.

34. In the Mule Debugger, step through the application; you should see the Choice router pass the event to the default branch.

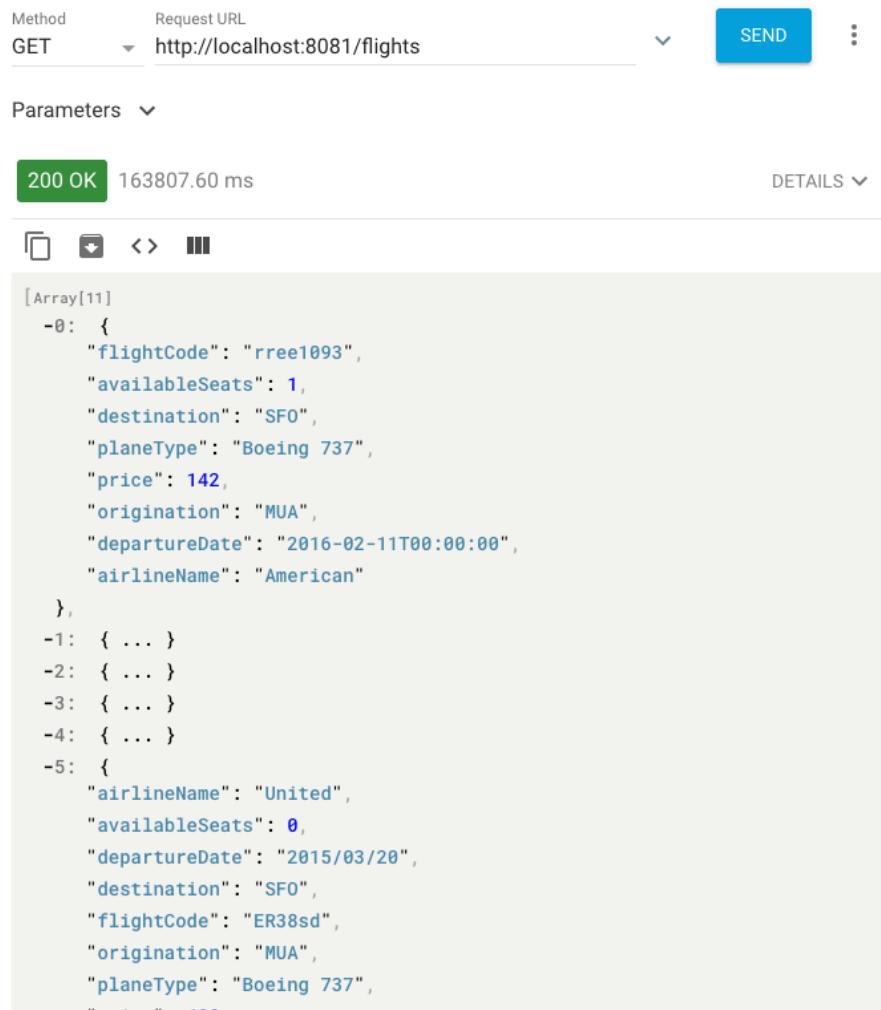


35. Resume to the Transform Message component after the Choice router; the payload should be an ArrayList of Flight objects

36. Step to the Logger; the payload should be JSON.

37. Step to the end of the application.

38. Return to Advanced REST Client; you should see American, United, and Delta flights to SFO (the default airport code).



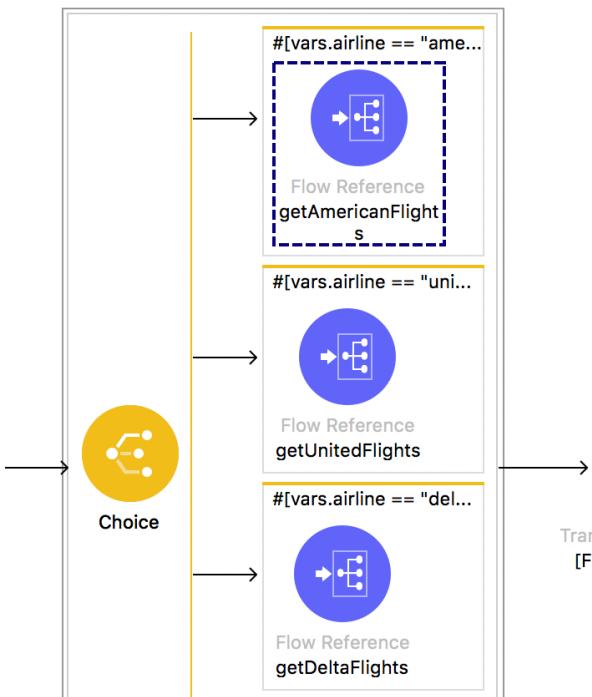
The screenshot shows the Advanced REST Client interface. At the top, it displays "Method: GET" and "Request URL: http://localhost:8081/flights". Below this is a "Parameters" dropdown. On the right side, there are "SEND" and "DETAILS" buttons. The main area shows a "200 OK" status with a timestamp of "163807.60 ms". A "DETAILS" button is also present here. Below the status, there are several icons: a copy icon, a refresh icon, a comparison icon, and a search icon. The response body is displayed in a large text area:

```
[Array[11]
-0: {
  "flightCode": "rree1093",
  "availableSeats": 1,
  "destination": "SFO",
  "planeType": "Boeing 737",
  "price": 142,
  "origination": "MUA",
  "departureDate": "2016-02-11T00:00:00",
  "airlineName": "American"
},
-1: { ... }
-2: { ... }
-3: { ... }
-4: { ... }
-5: {
  "airlineName": "United",
  "availableSeats": 0,
  "departureDate": "2015/03/20",
  "destination": "SFO",
  "flightCode": "ER38sd",
  "origination": "MUA",
  "planeType": "Boeing 737",
  ...
}...]
```

39. Add an airline query parameter set to american and send the request:

<http://localhost:8081/flights?airline=american>.

40. In the Mule Debugger, step through the rest of the application; you should see the event passed to getAmericanFlights and then back to getFlights.



41. Return to Advanced REST Client; you should see only American flights to SFO returned.

200 OK 35700.30 ms DETAILS ▾

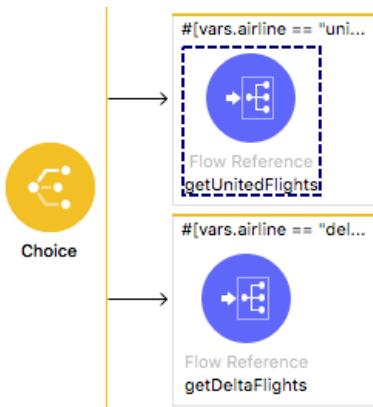
□ ▶ ⟲ ⟳ ⌂ ⌃

```
[  
 {  
   "flightCode": "rree1093",  
   "availableSeats": 1,  
   "destination": "SFO",  
   "planeType": "Boeing 737",  
   "price": 142.0,  
   "origination": "MUA",  
   "departureDate": "2016-02-11T00:00:00",  
   "airlineName": "American"  
 },  
 {  
   "flightCode": "eefd1994",  
   "availableSeats": 0,  
   "destination": "SFO",  
   "planeType": "Boeing 777",  
   "price": 676.0,  
   "origination": "MUA",  
   "departureDate": "2016-01-01T00:00:00",  
   "airlineName": "American"  
 },  
 {  
 }
```

42. Change the airline to united and add a second query parameter code to LAX:

<http://localhost:8081/flights?airline=united&code=LAX>.

43. Send the request.
44. In the Mule Debugger, step through the application; the event should be routed to the United branch.



45. Return to Advanced REST Client; you should see only United flights to LAX are returned.

200 OK 6116.70 ms DETAILS ▾

[

{

 "airlineName": "United",
 "availableSeats": 52,
 "departureDate": "2015/02/11",
 "destination": "LAX",
 "flightCode": "ER45if",
 "origination": "MUA",
 "planeType": "Boeing 737",
 "price": 345.99

},

{

 "airlineName": "United",
 "availableSeats": 12,
 "departureDate": "2015/04/11",
 "destination": "LAX",
 "flightCode": "ER45jd",
 "origination": "MUA",
 "planeType": "Boeing 777",
 "price": 346

},

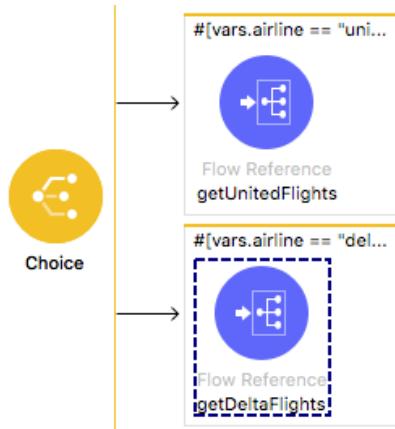
{

46. Change the airline to delta and the code to PDX:

<http://localhost:8081/flights?airline=delta&code=PDX>.

47. Send the request.

48. In the Mule Debugger, step through the application; the event should be routed to the Delta branch.



49. Return to Advanced REST Client; you should see only Delta flights to PDX are returned.

```
200 OK 34742.10 ms DETAILS ▾  
[  
 {  
   "flightCode": "A1FGF4",  
   "availableSeats": 80,  
   "destination": "PDX",  
   "planeType": "Boing 777",  
   "price": 958.0,  
   "origination": "MUA",  
   "departureDate": "2015/02/13",  
   "airlineName": "Delta"  
 },  
 {  
   "flightCode": "AFFFC4",  
   "availableSeats": 30,  
   "destination": "PDX",  
   "planeType": "Boing 777",  
   "price": 283.0,  
   "origination": "MUA",  
   "departureDate": "2015/02/20",  
   "airlineName": "Delta"  
 },  
 {
```

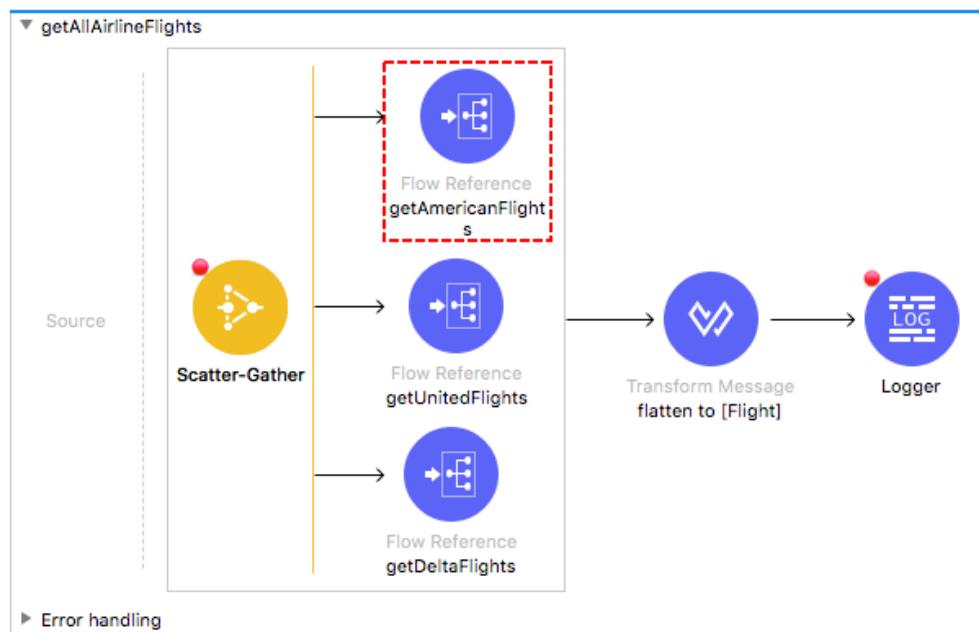
*Note: This JSON structure does not match that specified in the API: mua-flights-api.raml in src/main/resources/api. The data **should**, of course, be transformed so the response from the API matches this specification – but we are going to hold off doing that right now so that the scenario stays simpler for the error handling module (there is one less error to handle at the beginning).*

Test the application with a destination that has no flights

50. Remove the airline parameter and set the code to FOO:

<http://localhost:8081/flights?code=FOO>

51. In the Mule Debugger, step through the application; you should see multiple errors.



52. Return to Advanced REST Client; you should get a 500 Server Response and an exception.

Method Request URL
GET <http://localhost:8081/flights?code=FOO> **SEND** **⋮**

Parameters **⋮**

500 Server Error 11997.90 ms DETAILS **⋮**

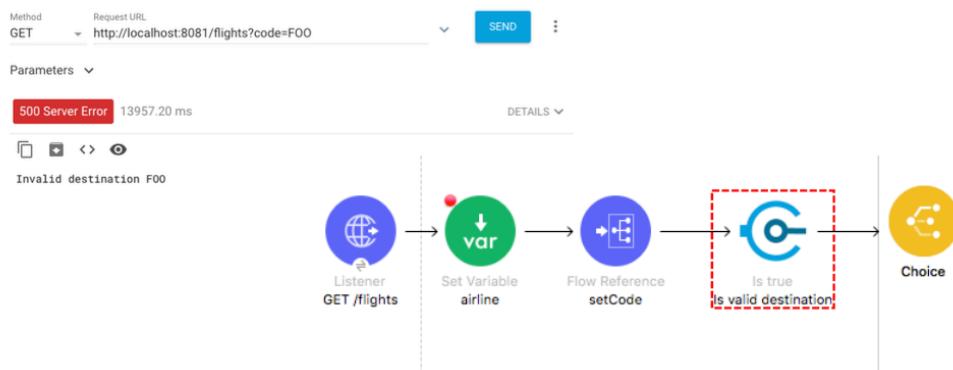
✖ **⬇** **<>** **✖**

Exception(s) were found for route(s):
0: org.mule.extension.http.api.request.validator.ResponseValidatorTypedException:
n: HTTP GET on resource '<http://training4-american-api-maxmule.us-e2.cloudhub.io:80/flights>' failed: bad request (400).
1: org.mule.runtime.core.api.expression.ExpressionRuntimeException: "You called
the function 'Value Selector' with these arguments:
1: Binary ("" as Binary {encoding: "UTF-8", mediaType: "application/octet-stream":

Walkthrough 9-3: Validate events

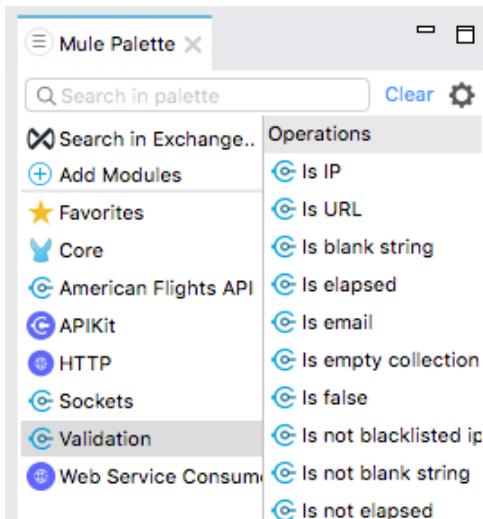
In this walkthrough, you use a validator to check if a query parameter called code with a value of SFO, LAX, CLE, PDX, or PDF is sent with a request and to throw an error if it is not. You will:

- Add the Validation module to a project.
- Use an Is true validator to check if a query parameter called code with a value of SFO, LAX, CLE, PDX, or PDF is sent with a request.
- Return a custom error message if the condition is not met.



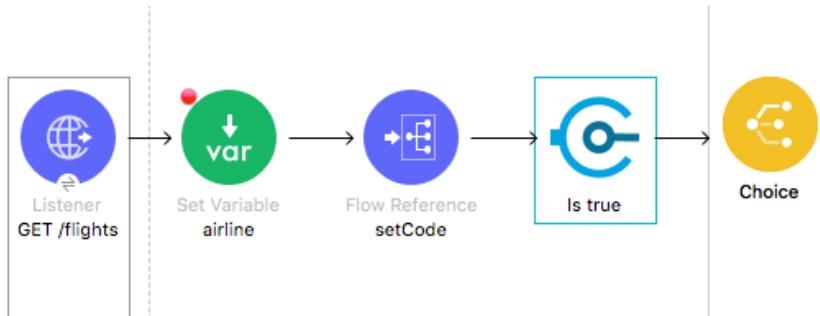
Add the Validation module to the project

1. Return to implementation.xml.
2. In the Mule Palette, select Add Modules.
3. Select the Validation module in the right side of the Mule Palette and drag and drop it into the left side.
4. If you get a Select module version dialog box, select the latest version and click Add.



Use the Is true validator to check for a valid destination code

5. Locate the Is true validator in the right side of the Mule Palette.
6. Drag and drop the Is true validator after the setCode Flow Reference in the getFlights flow.



7. In the Is true properties view, set the display name to Is valid destination.
8. Change the expression from False (Default) to Expression.
9. Add a DataWeave expression to check if the code variable is one of the five destination codes.

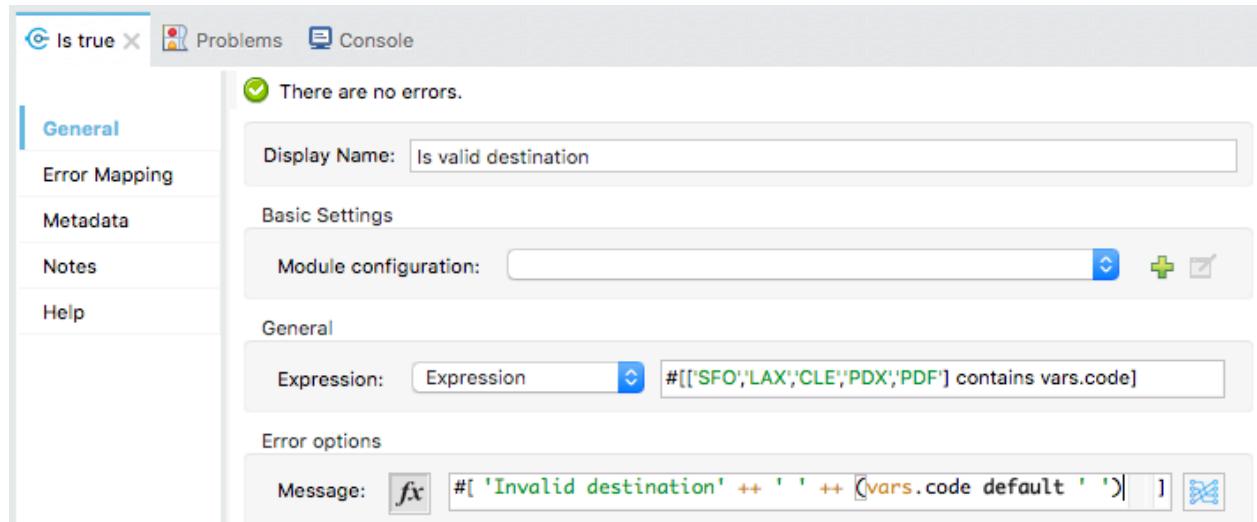
```
#[['SFO','LAX','CLE','PDX','PDF'] contains vars.code]
```

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

10. Using expression mode, set the error message to Invalid destination followed by the provided code.

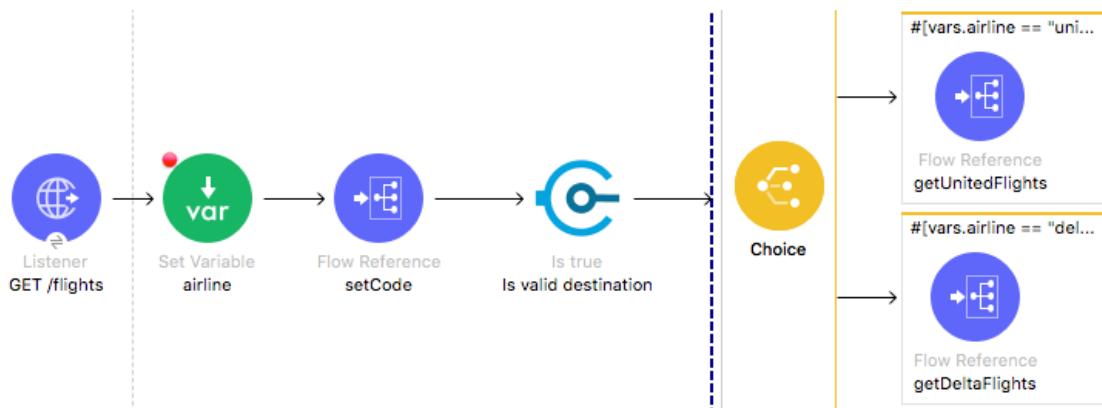
```
'Invalid destination' ++ ' ' ++ (vars.code default ' ')
```

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



Debug the application

11. Save the file then stop and debug the project.
12. In Advanced REST Client, change the code to make a request to <http://localhost:8081/flights?code=CLE>.
13. In the Mule Debugger, step past the validator; you should see the code is valid and application execution steps to the Choice router.



14. Resume through the rest of the application.
15. In Advanced REST Client, you should see flights for CLE.

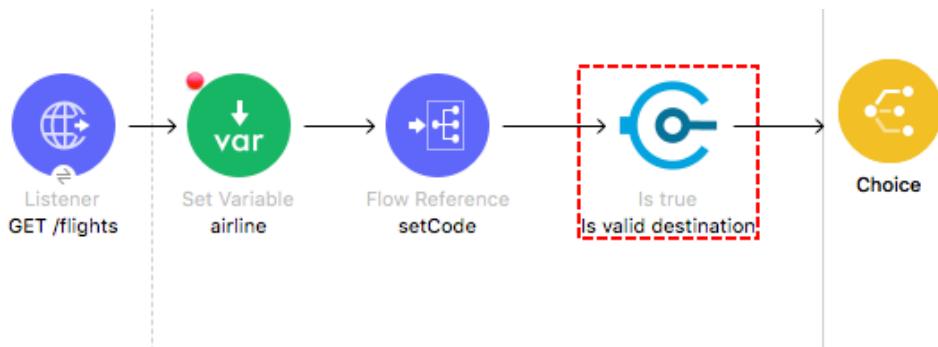
200 OK 128449.00 ms

□ □ □ □ □

```
[Array[8]
 -0: {
    "flightCode": "eefd0123",
    "availableSeats": 7,
    "destination": "CLE",
    "planeType": "Boeing 747",
    "price": 300,
    "origination": "MUA",
    "departureDate": "2016-01-25T00:00:00",
    "airlineName": "American"
 },
 -1: {
    "flightCode": "rree1000",
    "availableSeats": 10,
    "destination": "CLE",
    "planeType": "Airbus A320",
    "price": 250,
    "origination": "MUA",
    "departureDate": "2016-01-25T00:00:00",
    "airlineName": "Delta"
 },
 -2: {
    "flightCode": "fdd12345",
    "availableSeats": 5,
    "destination": "CLE",
    "planeType": "Boeing 777",
    "price": 400,
    "origination": "MUA",
    "departureDate": "2016-01-25T00:00:00",
    "airlineName": "American"
 },
 -3: {
    "flightCode": "ggg12345",
    "availableSeats": 12,
    "destination": "CLE",
    "planeType": "Airbus A330",
    "price": 350,
    "origination": "MUA",
    "departureDate": "2016-01-25T00:00:00",
    "airlineName": "Delta"
 },
 -4: {
    "flightCode": "ggg12345",
    "availableSeats": 12,
    "destination": "CLE",
    "planeType": "Airbus A330",
    "price": 350,
    "origination": "MUA",
    "departureDate": "2016-01-25T00:00:00",
    "airlineName": "Delta"
 },
 -5: {
    "flightCode": "ggg12345",
    "availableSeats": 12,
    "destination": "CLE",
    "planeType": "Airbus A330",
    "price": 350,
    "origination": "MUA",
    "departureDate": "2016-01-25T00:00:00",
    "airlineName": "Delta"
 },
 -6: {
    "flightCode": "ggg12345",
    "availableSeats": 12,
    "destination": "CLE",
    "planeType": "Airbus A330",
    "price": 350,
    "origination": "MUA",
    "departureDate": "2016-01-25T00:00:00",
    "airlineName": "Delta"
 },
 -7: {
    "flightCode": "ggg12345",
    "availableSeats": 12,
    "destination": "CLE",
    "planeType": "Airbus A330",
    "price": 350,
    "origination": "MUA",
    "departureDate": "2016-01-25T00:00:00",
    "airlineName": "Delta"
 }]
```

16. Change the code and make a request to <http://localhost:8081/flights?code=FOO>.

17. In the Mule Debugger, step to the validator; you should see an error.



18. Resume through the rest of the application; you should see your Invalid destination message in the console.

```
ERROR 2019-05-28 18:29:42,311 [[MuleRuntime].cpuLight.23: [apdev-flights-ws].getFlights.CPU]
*****
Message          : Invalid destination FOO.
Error type       : VALIDATION:INVALID_BOOLEAN
Element          : getFlights/processors/2 @ apdev-flights-ws:implementation.xml:15 (Is
Element XML     : <validation:is=true doc:name="Is valid destination" doc:id="11386e4f
                  (set debug level logging or '-Dmule.verbose.exceptions=true' for everything)
*****
```

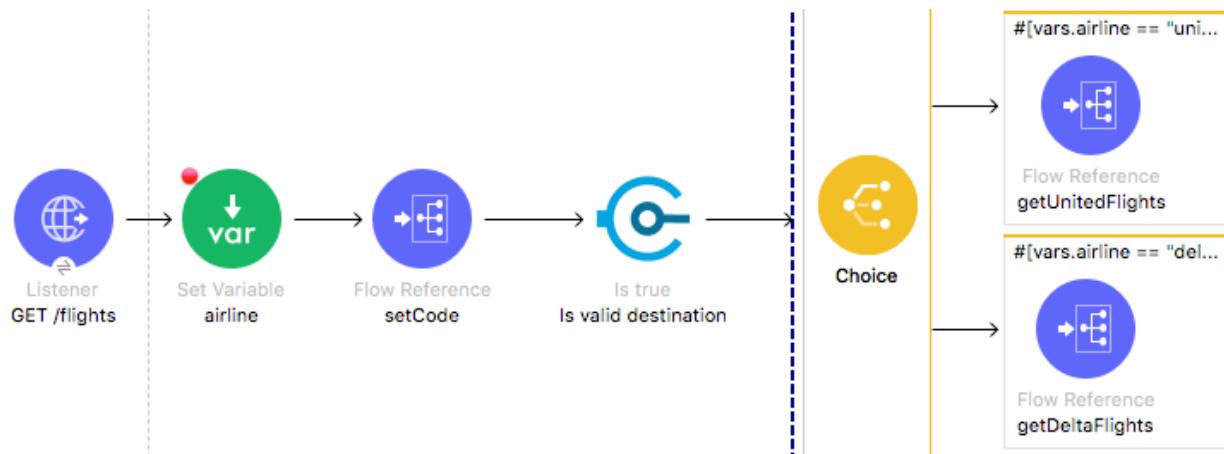
19. Return to Advanced REST Client; you should get a 500 Server Error with your Invalid destination message.

The screenshot shows the Advanced REST Client interface. The 'Method' is set to 'GET' and the 'Request URL' is 'http://localhost:8081/flights?code=FOO'. Below the URL, there's a 'Parameters' dropdown. The response status is '500 Server Error' with a duration of '13957.20 ms'. The response body contains the message 'Invalid destination FOO'. There are also 'DETAILS' and 'MORE' buttons.

Note: You will catch this error and send a JSON response with a different status code in the next module.

20. Remove the code and make a request to <http://localhost:8081/flights>.

21. In the Mule Debugger, step past the validator; you should not get any errors.



22. Resume through the rest of the application.

23. Return to Advanced REST Client; you should get a 200 response with all airline flights to SFO.

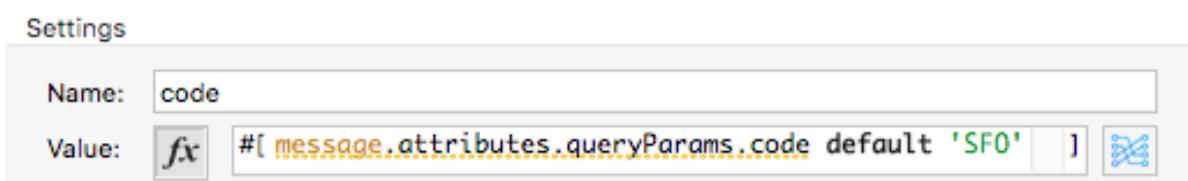
```
200 OK 31745.90 ms
[{"flightCode": "rree1093", "availableSeats": 1, "destination": "SFO", "planeType": "Boeing 737", "price": 142.0, "origination": "MUA", "departureDate": "2016-02-11T00:00:00", "airlineName": "American"}, {"flightCode": "eefd1994", "availableSeats": 0.}
```

24. Return to Anypoint Studio and switch to the Mule Design perspective.

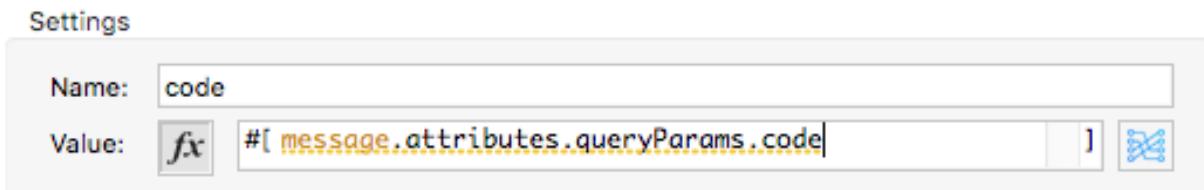
Remove the default destination

25. Locate the setCode subflow.

26. In the Set Variable properties view, review the default value.



27. Remove the default value from the value.

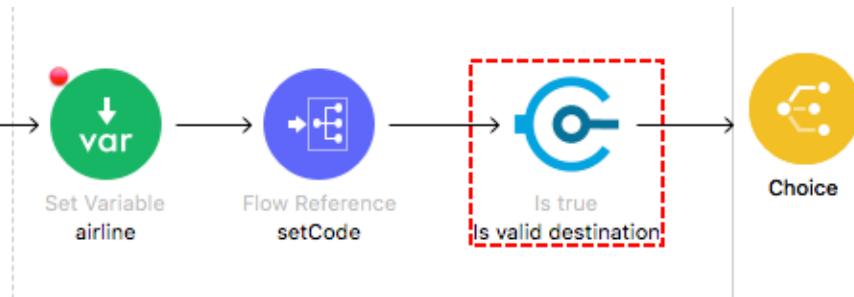


Debug the application

28. Save the file to redeploy the project.

29. In Advanced REST Client, make another request to <http://localhost:8081/flights>.

30. In the Mule Debugger, step to the validator; you should now get an error.



31. Resume through the rest of the application.

32. Return to Advanced REST Client; you should get a 500 Server Error with your Invalid destination message.

Method Request URL
GET <http://localhost:8081/flights> SEND

Parameters

500 Server Error 26591.16 ms DETAILS

Invalid destination

Note: You will catch this error and send a JSON response with a different status code in the next module.

33. Return to Anypoint Studio and stop the project.

Module 10: Handling Errors



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

- Handle messaging errors at the application, flow, and processor level.
- Handle different types of errors, including custom errors.
- Use different error scopes to either handle an error and continue execution of the parent flow or propagate an error to the parent flow.
- Set the success and error response settings for an HTTP Listener.
- Set reconnection strategies for system errors.

Walkthrough 10-1: Explore default error handling

In this walkthrough, you get familiar with the three errors you will learn to handle in this module. You will:

- Explore information about different types of errors in the Mule Debugger and the console.
- Review the default error handling behavior.
- Review and modify the error response settings for an HTTP Listener.

The screenshot shows the Mule Debugger interface with two panes. The left pane displays the flow configuration for a 'getDeltaFlights' flow, showing various variables and their values. The right pane shows an 'Error Response' configuration for a '400 Bad Request' status code. The 'Body' section contains the JSON template: `output application/json --- error.errorType`. Below it, the 'Headers' section is empty. At the bottom, the 'Status code:' field is set to '400' and the 'Reason phrase:' field is empty.

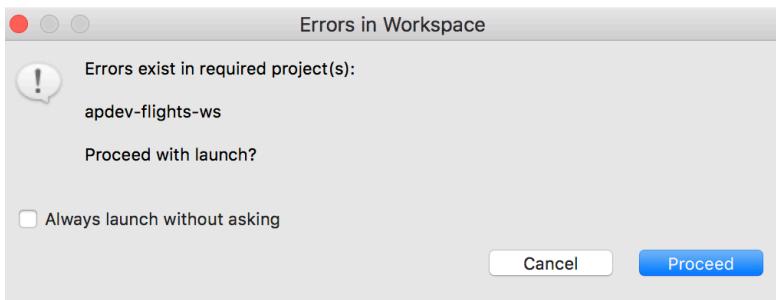
Create a connectivity error

1. Return to apdev-flights-ws in Anypoint Studio.
2. Open config.yaml.
3. Change the delta.wsdl property from /delta?wsdl to /deltas?wsdl.

```
delta:  
  wsdl: "http://mu.learn.mulesoft.com/deltas?wsdl"  
  service: "TicketServiceService"  
  port: "TicketServicePort"
```

4. Save the file.
5. Return to implementation.xml and debug the project.

- If you get the Errors in Workspace dialog box, click Proceed.



Review a validation error in the Mule Debugger and console

- In Advanced REST Client, add a code and make a request to <http://localhost:8081/flights?code=FOO>.
- In the Mule Debugger, step to where the error is thrown and expand the error object; you should see an error description, errorType, and other properties.

```

Mule Debugger X

▶ [C] Is True = Is valid destination
▶ [E] attributes = {HttpRequestAttributes} org.mule.extension.http.api.HttpRequestAttributes
  @ correlationId = "45fd9160-8249-11e9-b668-f018983d9329"
▼ [E] error = {ErrorImplementation} \org.mule.runtime.core.internal.message.ErrorBuilder$ErrorImpl
  @ description = "Invalid destination FOO"
  @ detailedDescription = "Invalid destination FOO"
  ► [E] errorType = (ErrorTypeImplementation) VALIDATION:INVALID_BOOLEAN
    @ errors = {UnmodifiableRandomAccessList} size = 0
    ► [E] exception = {ValidationException} org.mule.extension.validation.api.ValidationException
    ► [E] muleMessage = {MessageImplementation} \org.mule.runtime.core.internal.message.MessageImpl
    ► [E] payload =
    ► [E] vars = {Map} size = 1
  
```

VALIDATION:INVALID_BOOLEAN

- Step through the rest of the application and review the error information logged in the console.

```

Console X Mule Properties
apdev-flights-ws [Mule Applications] Mule Server 4.2.0 EE
ERROR 2019-05-29 15:43:43,381 [[MuleRuntime].cpuLight.24: [apdev-flights-ws].getFlights.CPU_L]
*****
Message      : Invalid destination FOO.
Error type   : VALIDATION:INVALID_BOOLEAN
Element       : getFlights/processors/2 @ apdev-flights-ws:implementation.xml:15 (Is valid destination)
Element XML  : <validation:is=true doc:name="Is valid destination" doc:id="11386e4f-1138-4381-8249-11e9-b668-f018983d9329" />
(set debug level logging or '-Dmule.verbose.exceptions=true' for everything)
*****
```

10. In Advanced REST Client, locate the response status code, status reason, and body; you should see a 500 status code, a status reason of Server Error, and a response body equal to the error description.

The screenshot shows the Advanced REST Client interface. At the top, it displays the method as 'GET' and the request URL as 'http://localhost:8081/flights?code=FOO'. Below this, there's a 'Parameters' dropdown. The main content area shows a red box indicating a '500 Server Error' with a duration of '8202.90 ms'. A 'DETAILS' button is available. Below the error message, there are several icons: a square, a downward arrow, a double-headed arrow, and an eye icon. The text 'Invalid destination FOO' is displayed at the bottom.

Review a connectivity error in the Mule Debugger and console

11. Add an airline and change the code to make a request to <http://localhost:8081/flights?airline=delta&code=PDX>.
12. In the Mule Debugger, step to where the error is thrown and review the error object.

The screenshot shows the Mule Debugger interface with a tree view of an error object. The root node is 'Flow Ref = getDeltaFlights'. It has children: 'attributes' (HttpRequestAttributes), 'correlationId' (a UUID), and 'error' (ErrorImplementation). The 'error' node is expanded, showing its properties: 'description' (Error trying to acquire a new connection:Error fetching the resource [http://mu.learn.mulesoft.com/deltas?wsdl]), 'detailedDescription' (Error trying to acquire a new connection:Error fetching the resource [http://mu.learn.mulesoft.com/deltas?wsdl]), 'errorType' (WSC:CONNECTIVITY), 'errors' (UnmodifiableRandomAccessList with size 0), and 'exception' (ConnectionException with the same detailed description).

13. Review the error information logged in the console.
14. Step through the rest of the application.
15. Return to Advanced REST Client; you should see a 500 Server Error with the error description.

The screenshot shows the Advanced REST Client interface again. The request URL is now 'http://localhost:8081/flights?airline=delta&code=PDX'. The response shows a red box for a '500 Server Error' with a duration of '30045.30 ms'. The 'DETAILS' button is present. Below the error message are the same icons as before. The text 'Error trying to acquire a new connection:Error fetching the resource [http://mu.learn.mulesoft.com/deltas?wsdl]: http://mu.learn.mulesoft.com/deltas?wsdl' is displayed at the bottom.

Review a bad request error in the Mule Debugger and console

16. Change the airline to make a request to

<http://localhost:8081/flights?airline=american&code=PDX>.

17. In the Mule Debugger, step to where the error is thrown and review the error object.

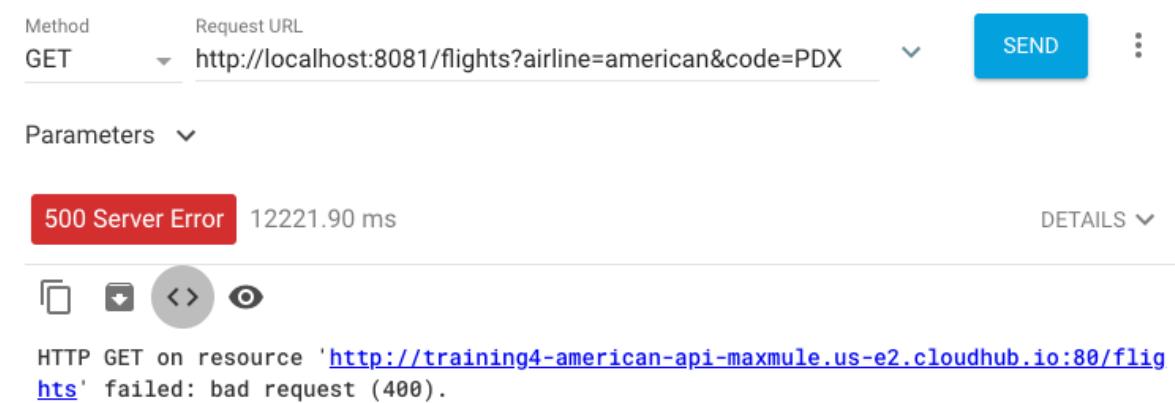


The screenshot shows the Mule Debugger interface with the title bar "Mule Debugger". Below it is a tree view of an error object. The root node is "error" (ErrorImplementation). Other nodes include "Flow Ref = getAmericanFlights", "attributes = {HttpRequestAttributes} org.mule.extension.http.api.HttpRequestAttributes", "correlationId = "70025570-824b-11e9-b668-f018983d9329", "errorType = {ErrorTypeImplementation} AMERICAN-FLIGHTS-API:BAD_REQUEST", "errors = {UnmodifiableRandomAccessList} size = 0", and "exception = {ResponseValidatorTypedException} org.mule.extension.http.api.request.validator.ResponseValidatorTypedException: HTTP GET on resource 'http://training4-american-api-maxmule.us-e2.cloudhub.io:80/flights' failed: bad request (400)".

18. Step through the rest of the application.

19. Review the error information logged in the console.

20. Return to Advanced REST Client; you should see a 500 Server Error with the error description.



The screenshot shows the Advanced REST Client interface. At the top, there are fields for "Method" (GET), "Request URL" (<http://localhost:8081/flights?airline=american&code=PDX>), and a "SEND" button. Below this is a "Parameters" dropdown. The main area shows a red box containing "500 Server Error" and "12221.90 ms". To the right is a "DETAILS" button. Below the error message are several icons: a clipboard, a download arrow, a copy icon, and a magnifying glass. The error description text reads: "HTTP GET on resource '<http://training4-american-api-maxmule.us-e2.cloudhub.io:80/flights>' failed: bad request (400)".

21. Return to Anypoint Studio and switch to the Mule Design perspective.

22. Stop the project.

Review the default error response settings for an HTTP Listener

23. Navigate to the properties view for the GET /flights Listener in getFlights.

24. Select the Responses tab.

25. Locate the Error Response section and review the default settings for the body, status code, and reason phrase.

Error Response

Body:	<input type="button" value="fx"/>	1 <code>output text/plain --- error.description</code>	<input type="button" value="X"/>				
Headers:	<input type="button" value="fx"/>	Headers	<input type="button" value="X"/>				
<table border="1"><tr><td>Name</td><td>Value</td></tr><tr><td></td><td></td></tr></table>				Name	Value		
Name	Value						
Status code:	<input type="button" value="fx"/>						
Reason phrase:	<input type="button" value="fx"/>						

Remove the default error response settings for the HTTP Listener

26. Select and cut the body expression (so it has no value, but you can paste it back later).

Error Response

Body:	<input type="button" value="fx"/>	1	<input type="button" value="X"/>				
Headers:	<input type="button" value="fx"/>	Headers	<input type="button" value="X"/>				
<table border="1"><tr><td>Name</td><td>Value</td></tr><tr><td></td><td></td></tr></table>				Name	Value		
Name	Value						
Status code:	<input type="button" value="fx"/>	1					
Reason phrase:	<input type="button" value="fx"/>						

Test the application

27. Save the file, run the project, and proceed through any errors in the workspace.

28. In Advanced REST Client, make another request to

http://localhost:8081/flights?airline=american&code=PDX; you should get the same response.

The screenshot shows the Advanced REST Client interface. At the top, there are fields for 'Method' (GET) and 'Request URL' (http://localhost:8081/flights?airline=american&code=PDX). To the right are buttons for 'SEND' and more options. Below these are sections for 'Parameters' and 'Headers'. Under 'Headers', there is a 'Content-Type' field set to 'application/json'. The main area displays an error message: '500 Server Error' in a red box, followed by '2562.70 ms'. Below this is a toolbar with icons for copy, paste, refresh, and others. The detailed error message is: 'HTTP GET on resource '<http://training4-american-api-maxmule.us-e2.cloudhub.io:80/flights>' failed: bad request (400)'.

Modify the default error response settings for the HTTP Listener

29. Return to Anypoint Studio.

30. Return to the Responses tab in the properties view for the GET /flights Listener.

31. In the error response body, paste back the original value.

```
output text/plain --- error.description
```

32. Change the output type of the error response to application/json and display the error.errorType.

```
output application/json --- error.errorType
```

Note: Ensure the error response body is in expression mode.

33. Set the error response status code to 400.

The screenshot shows the 'Error Response' configuration for a GET /flights Listener. It includes sections for 'Body', 'Headers', 'Status code', and 'Reason phrase'. The 'Body' section contains the expression 'output application/json --- error.errorType'. The 'Headers' section is empty. The 'Status code' is set to 400, and the 'Reason phrase' is empty.

Test the application

34. Save the file to redeploy the application.
35. In Advanced REST Client, make another request to <http://localhost:8081/flights?airline=american&code=PDX>; you should get the new status code and reason and the response body should be the errorType object.
36. Look at the error namespace and identifier.

The screenshot shows the Advanced REST Client interface. The request URL is <http://localhost:8081/flights?airline=american&code=PDX>. The response is a 400 Bad Request with a duration of 596.10 ms. The response body is a JSON object representing an error type:

```
{
  "identifier": "BAD_REQUEST",
  "parentErrorType": {
    "identifier": "ANY",
    "parentErrorType": null,
    "namespace": "MULE"
  },
  "namespace": "AMERICAN-FLIGHTS-API"
}
```

37. Change the airline to make a request to <http://localhost:8081/flights?airline=delta&code=PDX>.
38. Look at the error namespace and identifier.

The screenshot shows the Advanced REST Client interface. The request URL is <http://localhost:8081/flights?airline=delta&code=PDX>. The response is a 400 Bad Request with a duration of 389.10 ms. The response body is a JSON object representing an error type:

```
{
  "identifier": "CONNECTIVITY",
  "parentErrorType": {
    "identifier": "CONNECTIVITY",
    "parentErrorType": {
      "identifier": "ANY",
      "parentErrorType": null,
      "namespace": "MULE"
    },
    "namespace": "MULE"
  },
  "namespace": "WSC"
}
```

39. Change the code to make a request to <http://localhost:8081/flights?airline=delta&code=FOO>.

40. Look at the namespace and identifier.

The screenshot shows the Anypoint Studio interface with a request configuration at the top. The method is set to GET and the URL is http://localhost:8081/flights?airline=delta&code=FOO. Below the configuration is a 'Parameters' dropdown. The main content area displays a 400 Bad Request response with a timestamp of 13.10 ms. The response body is a JSON object representing a validation error:

```
{  
  "identifier": "INVALID_BOOLEAN",  
  "parentErrorType": {  
    "identifier": "VALIDATION",  
    "parentErrorType": {  
      "identifier": "VALIDATION",  
      "parentErrorType": {  
        "identifier": "ANY",  
        "parentErrorType": null,  
        "namespace": "MULE"  
      },  
      "namespace": "MULE"  
    },  
    "namespace": "VALIDATION"  
  },  
  "namespace": "VALIDATION"  
}
```

Return the default error response settings for the HTTP Listener

41. Return to Anypoint Studio.

42. Return to the Responses tab in the properties view for the GET /flights Listener.

43. Change the error response output type back to text/plain and display the error.description.

```
output text/plain --- error.description
```

44. Remove the status code.

The screenshot shows the 'Error Response' configuration in Anypoint Studio. It includes fields for Body (containing '1 output text/plain --- error.description'), Headers (empty), Status code (empty), and Reason phrase (empty).

Test the application

45. Save the file to redeploy the application.

46. In Advanced REST Client, change the airline and code to make a request to <http://localhost:8081/flights?airline=american&code=PDX>; you should get the 500 Server Error again with the plain text error description.

The screenshot shows the Advanced REST Client interface. It has a 'Method' dropdown set to 'GET' and a 'Request URL' input field containing 'http://localhost:8081/flights?airline=american&code=PDX'. Below the URL is a 'Parameters' dropdown. The response section shows a red '500 Server Error' button, '674.90 ms' latency, and a 'DETAILS' link. The details page shows the error message: 'HTTP GET on resource '<http://training4-american-api-maxmule.us-e2.cloudhub.io:80/flights>' failed: bad request (400)'.

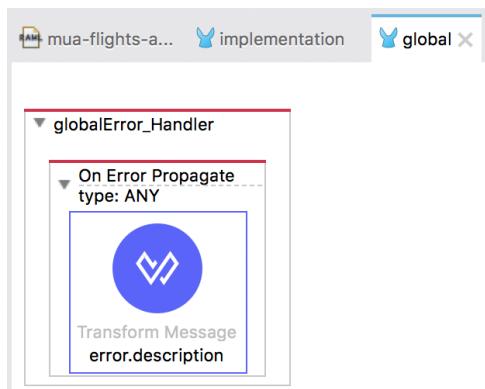
47. Return to Anypoint Studio.

48. Stop the project.

Walkthrough 10-2: Handle errors at the application level

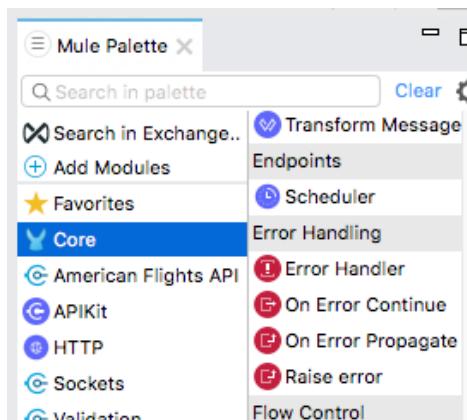
In this walkthrough, you create a default error handler for the application. You will:

- Create a global error handler in an application.
- Configure an application to use a global default error handler.
- Explore the differences between the On Error Continue and On Error Propagate scopes.
- Modify the default error response settings for an HTTP Listener.



Browse the error handling elements in the Mule Palette

1. Return to global.xml and switch to the Message Flow view.
2. In the Core section of the Mule Palette, locate the Error Handling elements.



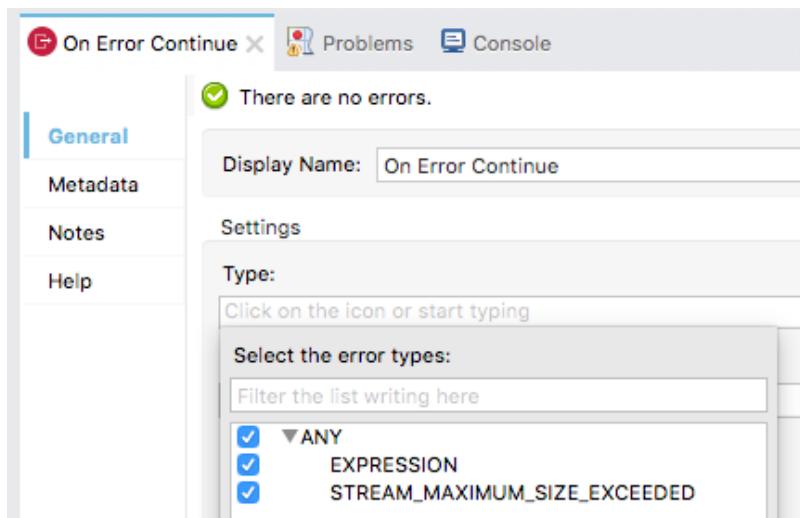
Create a global error handler with an On Error Continue scope

3. Drag out an Error Handler element and drop it in the canvas of global.xml.

4. Drag out an On Error Continue element and drop it in globalError_Handler.

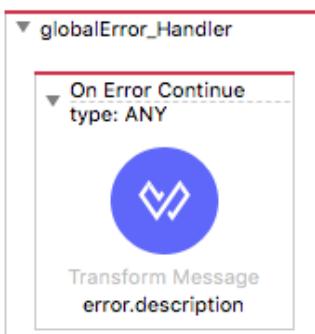


5. In the On Error Continue properties view, click the Search button for type.
6. In the drop-down menu that appears, select ANY.



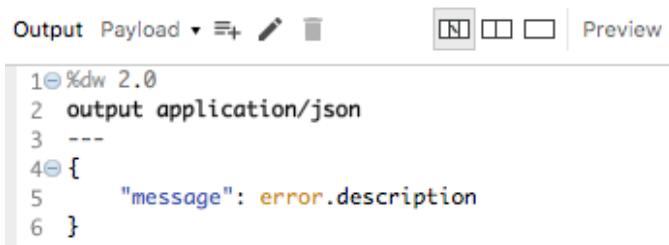
Set the payload in the error handler to a JSON message

7. Add a Transform Message component to the On Error Continue.
8. Set the Transform Message display name to error.description.



9. In the Transform Message properties view, change the output type to application/json.

10. Add a message property to the output JSON and give it a value of the error.description.



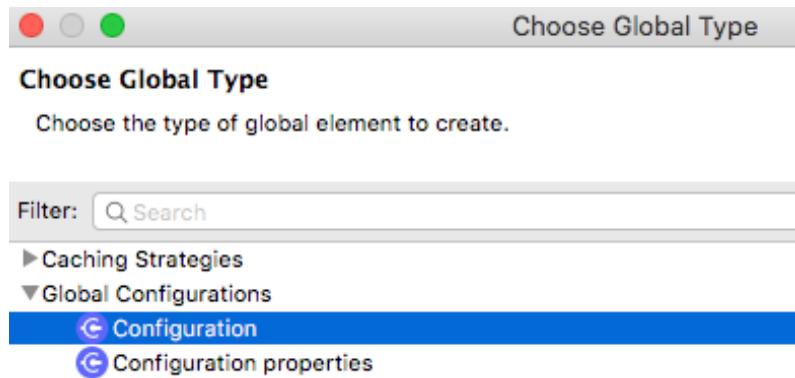
```
Output Payload ▾ + ⚪ Preview  
1 %dw 2.0  
2 output application/json  
3 ---  
4 {  
5   "message": error.description  
6 }
```

Set a default error handler for the application

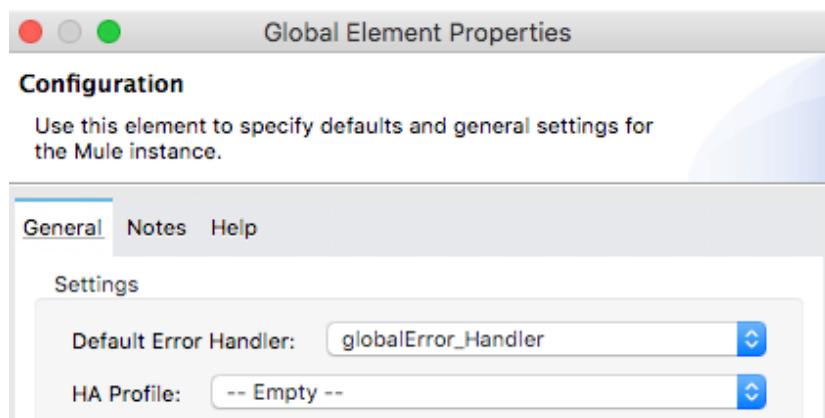
11. Switch to the Global Elements view of global.xml.

12. Click Create.

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



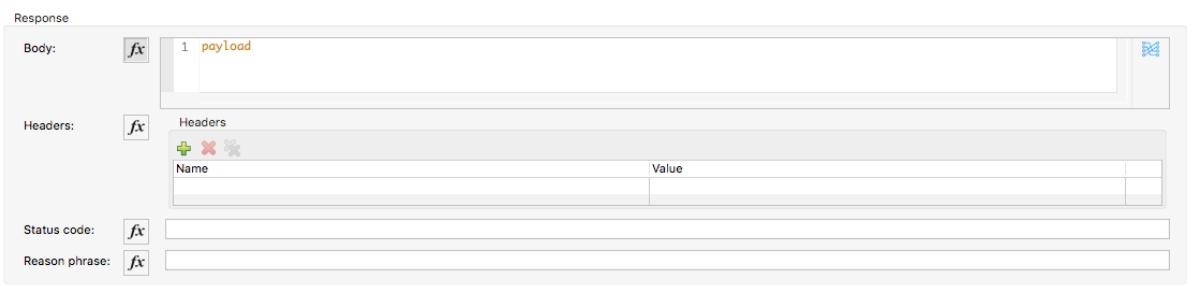
14. In the Global Element Properties dialog box, set the default error handler to globalError_Handler and click OK.



15. Switch to the Message Flow view.

Review the default response settings for an HTTP Listener

16. Return to implementation.xml.
17. Navigate to the properties view for the GET /flights Listener in getFlights.
18. Select the Responses tab.
19. Locate the Response section (not the Error Response section) and review the default settings for the body, status code, and reason phrase.



20. Locate the Error Response section and review the default settings for the body, status code, and reason phrase.

Test the On Error Continue behavior

21. Save all files, debug the project, and proceed through any errors in the workspace.
22. In Advanced REST Client, make another request to
<http://localhost:8081/flights?airline=american&code=PDX>.
23. In the Mule Debugger, step through the application until the event is passed to globalError_Handler.

24. Expand the error object and review the exception.

The screenshot shows the Mule Debugger interface with the title bar "Mule Debugger". Below it is a tree view of an error object:

- Transform = error.description
- attributes = {HttpRequestAttributes} org.mule.extension.http.api.HttpRequestAttributes
Request path=/flights
- correlationId = "98203240-8267-11e9-962b-f018983d9329"
- error = {ErrorImplementation} \org.mule.runtime.core.internal.message.ErrorBuilder\$ErrorImplementation
description = "HTTP GET on resource 'http://training4-american-api-maxmule.us-e2.cloudhub.io:80/flights'"
detailedDescription = "HTTP GET on resource 'http://training4-american-api-maxmule.us-e2.cloudhub.io:80/flights'"
errorType = {ErrorTypeImplementation} AMERICAN-FLIGHTS-API:BAD_REQUEST
errors = {UnmodifiableRandomAccessList} size = 0
- exception = {ResponseValidatorTypedException} org.mule.extension.http.api.request.validator.ResponseValidator
CAUSE_CAPTION = "Caused by:"
EMPTY_THROWABLE_ARRAY = {Throwable[]} size = 0
NULL_CAUSE_MESSAGE = "Cannot suppress a null exception."

Below the debugger, the Mule Studio interface is visible with tabs for "mua-flights-api.raml", "implementation", "global" (which is selected), "config.yaml", and "FlightsExample.raml".

On the "global" tab, there is a configuration for "globalError_Handler":

- On Error Continue type: ANY
- transform Message error.description

25. Step again; the event should be passed back to getFlights, which continues to execute after the error occurs.

26. Review the payload and note the absence of an error object.

The screenshot shows the Mule Debugger interface with the title bar "Mule Debugger". On the left, the payload is shown as a JSON object:

```
{ "message": "HTTP GET on resource 'http://training4-american-api-maxmule.us-e2.cloudhub.io:80/flights' failed with status code 400." }
```

Below the debugger, the Mule Studio interface is visible with tabs for "mua-flights-api.raml", "implementation" (which is selected), "global", "config.yaml", and "FlightsExample.raml".

On the "implementation" tab, the flow diagram is displayed:

```
graph LR; Start(( )) --> Choice{Choice}; Choice -- "Is true valid destination" --> GetUnitedFlights[Flow Reference getUnitedFlights]; Choice -- "[vars.airline == \"delta\"]" --> Transform[Transform Message [Flight] to JSON]; GetUnitedFlights --> Logger[Logger]; Transform --> Logger;
```

The flow starts with a "Choice" component. One path from the "Choice" component leads to a "Get United Flights" component, which then connects to a "Logger". The other path from the "Choice" component leads to a "Transform" component, which also connects to the "Logger".

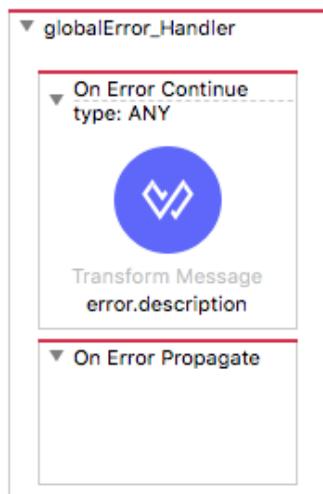
27. Resume through the rest of the application.
28. Return to Advanced REST Client; you should see get a 200 OK response with the response body equal to the payload value set in the error handler.

The screenshot shows the Advanced REST Client interface. At the top, it displays the method as "GET" and the request URL as "http://localhost:8081/flights?airline=american&code=PDX". Below the URL, there's a "SEND" button and a more options menu. Under "Parameters", there's a dropdown menu. The main response area shows a green "200 OK" status bar with a timestamp of "17492.40 ms" and a "DETAILS" link. Below this, there are several icons: a copy icon, a refresh icon, a diff icon, and a search icon. The JSON response body is displayed in a code block:

```
{
  "message": "HTTP GET on resource 'http://training4-american-api-maxmule.us-e2.cloudbus.io:80/flights' failed: bad request (400)."
}
```

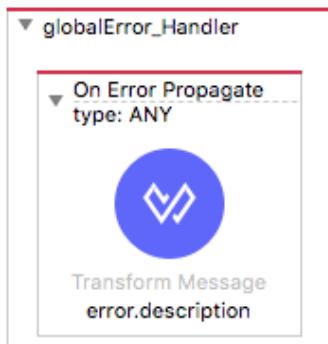
Change the default error handler to use an On Error Propagate scope

29. Return to Anypoint Studio and switch to the Mule Design perspective.
30. Return to global.xml.
31. Drag an On Error Propagate element from the Mule Palette and drop it to the right or left of the On Error Continue to add it to globalError_Handler.



32. In the On Error Propagate properties view, set the type to ANY.
33. Move the Transform Message component from the On Error Continue to the On Error Propagate.

34. Delete the On Error Continue scope.



Test the On Error Propagate behavior

35. Save the files to redeploy the project.

36. In Advanced REST Client, make another request to

<http://localhost:8081/flights?airline=american&code=PDX>.

37. Step through the application until the event is passed to globalError_Handler.

38. Look at the payload and the exception in the Mule Debugger; they should be the same as before.

39. Step again; the error is propagated up to the getFlights parent flow.

40. Look at the payload and the error object in the Mule Debugger.

```
description = "HTTP GET on resource 'http://training4-american-api-maxmule.us-e...'  
detailedDescription = "HTTP GET on resource 'http://training4-american-api-maxm...'  
errorType = {ErrorTypeImplementation} AMERICAN-FLIGHTS-API:BAD_REQUEST  
errors = {UnmodifiableRandomAccessList} size = 0  
exception = {ResponseValidatorTypedException} org.mule.extension.http.api.request.  
muleMessage = {MessageImplementation} \norg.mule.runtime.core.internal.message...  
payload = {\n  "message": "HTTP GET on resource 'http://training4-american-api-maxm..."  
vars = {Map} size = 2
```

```
{  
  "message": "HTTP GET on resource 'http://training4-american-api-maxmule.us-e...'"}
```

mua-flights-api.raml implementation global config.yaml FlightsExample.raml

Flow Reference
getAmericanFlight

41. Step again; the error is handled by the application's default error handler again.

42. Look at the payload and the exception in the Mule Debugger.
43. Resume through the rest of the application.
44. Return to Advanced REST Client; you should see get a 500 Server Error response with the response body equal to the plain text error description – not the payload.

Method: GET Request URL: http://localhost:8081/flights?airline=american&code=PDX

Parameters: ▾

500 Server Error 22791.80 ms DETAILS ▾

HTTP GET on resource '<http://training4-american-api-maxmule.us-e2.cloudhub.io:80/flights>' failed: bad request (400).

45. Return to Anypoint Studio and switch to the Mule Design perspective.
46. Stop the project.

Modify the default error response settings for the HTTP Listener

47. Return to implementation.xml.
48. Navigate to the Responses tab in the properties view for the GET /flights Listener.
49. Change the error response body to return the payload instead of error.description.

payload

Name	Value

Test the application

50. Save the file, run the project, and proceed through any errors in the workspace.

51. In Advanced REST Client, make another request to <http://localhost:8081/flights?airline=american&code=PDX>; you should now get the message that you set in the error handler.

The screenshot shows the Advanced REST Client interface. The 'Method' is set to 'GET' and the 'Request URL' is <http://localhost:8081/flights?airline=american&code=PDX>. Below the request area, there's a 'Parameters' dropdown. The main response area shows a red '500 Server Error' box with '142.80 ms' underneath. To the right is a 'DETAILS' button. The error message is displayed in a code block:

```
{  
  "message": "HTTP GET on resource 'http://training4-american-api-maxmule.us-e2.cloudhub.io:80/flights' failed: bad request (400)."  
}
```

Note: You will handle this error differently in a later walkthrough, so it does not return a server error. It is a valid request; there are just no American flights to PDX.

52. Change the airline to make a request to <http://localhost:8081/flights?airline=delta&code=PDX>; the error is handled by the same default handler.

The screenshot shows the Advanced REST Client interface. The 'Method' is set to 'GET' and the 'Request URL' is <http://localhost:8081/flights?airline=delta&code=PDX>. Below the request area, there's a 'Parameters' dropdown. The main response area shows a red '500 Server Error' box with '356.40 ms' underneath. To the right is a 'DETAILS' button. The error message is displayed in a code block:

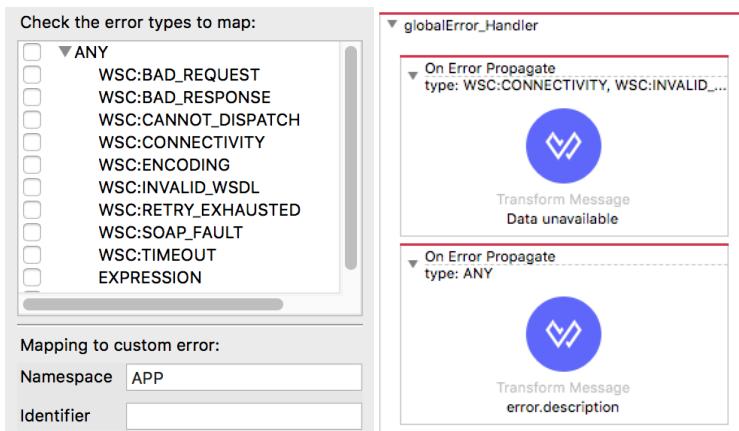
```
{  
  "message": "Error trying to acquire a new connection:Error fetching the resource [http://mu.learn.mulesoft.com/deltas?wsdl]: http://mu.learn.mulesoft.com/deltas?wsdl"  
}
```

53. Return to Anypoint Studio.

Walkthrough 10-3: Handle specific types of errors

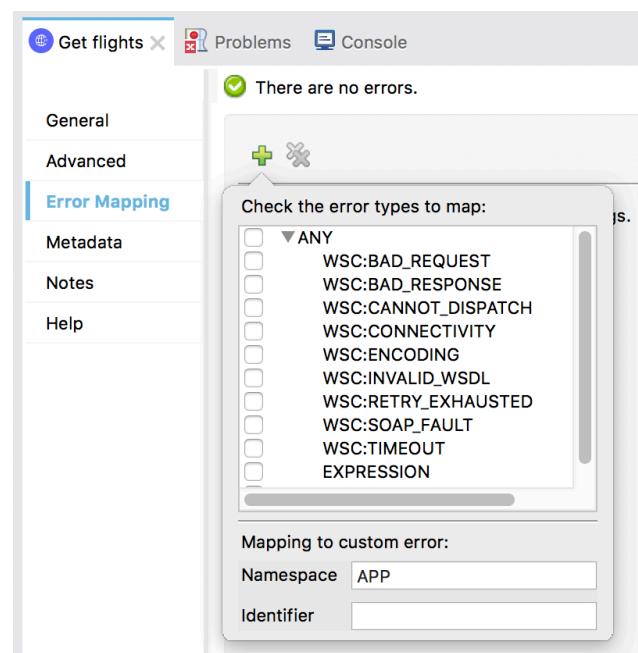
In this walkthrough, you continue to work with the application's default error handler. You will:

- Review the possible types of errors thrown by different processors.
- Create error handler scopes to handle different error types.



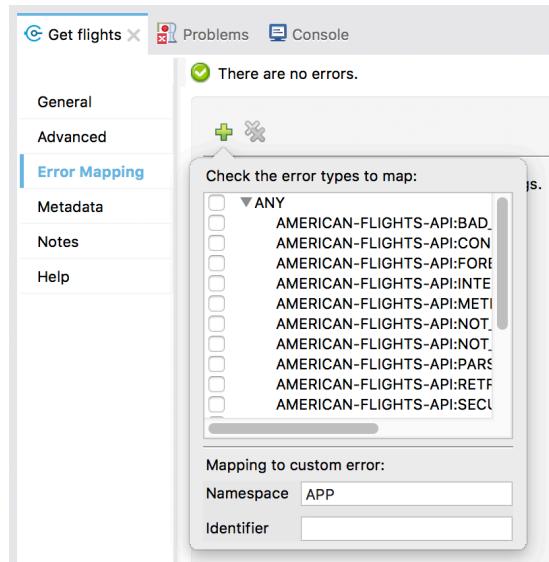
Review the possible types of errors thrown by a Web Service Consumer operation

1. Return to implementation.xml.
2. Navigate to the properties view for the Get flights Consume operation in getDeltaFlights.
3. Select the Error Mapping tab.
4. Click the Add new mapping button and review (but don't select!) the WSC error types.



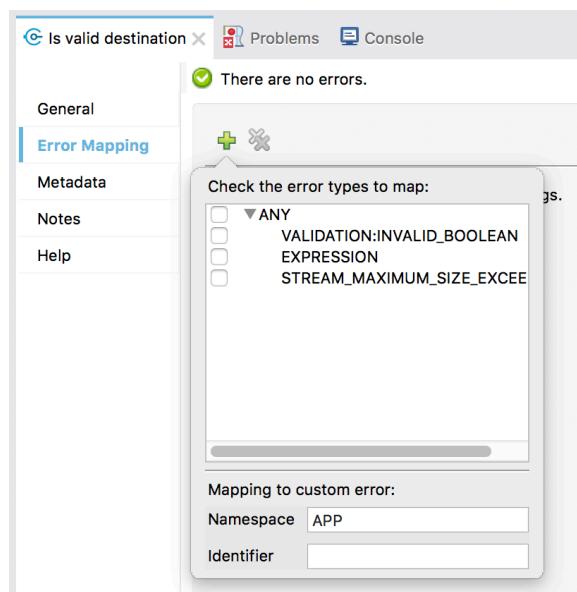
Review the possible types of errors thrown by a REST Connector operator

5. Navigate to the properties view for the Get flights operation in getAmericanFlights.
6. Select the Error Mapping tab.
7. Click the Add new mapping button, and review (but don't select!) the AMERICAN-FLIGHTS-API error types.



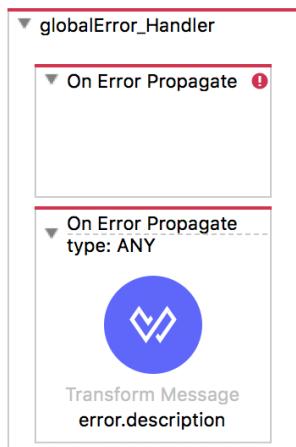
Review the possible types of errors thrown by a Validator operation

8. Navigate to the properties view for the Is true validator in getFlights.
9. Select the Error Mapping tab.
10. Click the Add new mapping button and locate (but don't select!) the VALIDATION error type.

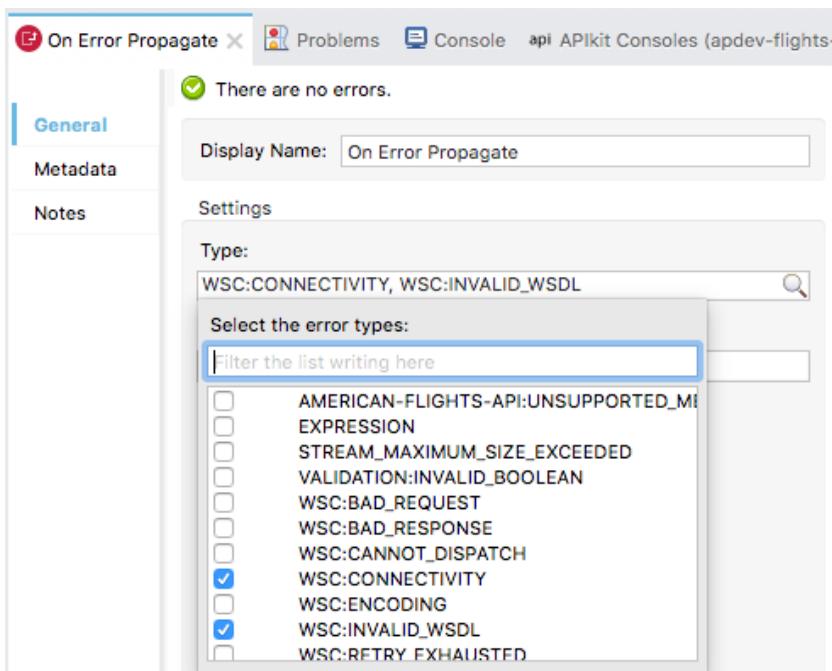


Add a second error handler to catch Web Service Consumer connectivity errors

11. Return to global.xml.
12. Add a second On Error Propagate to globalError_Handler.
13. If necessary, drag and drop it so it is the first scope inside the error handler.

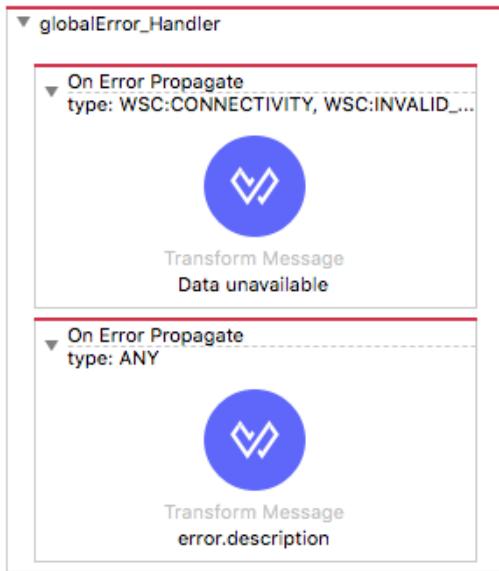


14. In the properties view for the new On Error Propagate, click the Search button.
15. Select WSC:CONNECTIVITY and WSC:INVALID_WSDL.



Note: If these values do not show up in the drop-down menu, type this information instead. The two values should be comma-separated.

16. Add a Transform Message component to the new On Error Propagate and set its display name to Data unavailable.



17. In the Transform Message properties view, change the output type to application/json.
18. Add a message property to the output JSON and set give it a value of "Data unavailable. Try later. ".
19. For development purposes, concatenate the error.description to the message; be sure to use the auto-completion menu.
20. Coerce the variable to a String.

```
1 %dw 2.0
2 output application/json
3 ---
4 {
5     "message": "Data unavailable. Try later. " ++ error.description as String
6 }
```

Test the application

21. Save the file to redeploy the project.

22. In Advanced REST Client, make another request to <http://localhost:8081/flights?airline=delta&code=PDX>; you should now get a 500 Server Error response with your new Data unavailable message.

The screenshot shows the Advanced REST Client interface. The method is set to GET and the URL is <http://localhost:8081/flights?airline=delta&code=PDX>. The response status is 500 Server Error with a duration of 2870.80 ms. The response body is a JSON object with a single key "message": "Data unavailable. Try later. Error trying to acquire a new connection:Error fetching the resource [http://mu.learn.mulesoft.com/deltas?wsdl]: http://mu.learn.mulesoft.com/deltas?wsdl".

23. Change the code to make a request to <http://localhost:8081/flights?airline=delta&code=FOO>; you should still get a 500 Server Error response with the Invalid destination JSON message.

Note: This should also not be a server error, but either a 4XX bad request or a 200 OK with an appropriate message. You will handle this error differently in the next walkthrough.

The screenshot shows the Advanced REST Client interface. The method is set to GET and the URL is <http://localhost:8081/flights?airline=delta&code=FOO>. The response status is 500 Server Error with a duration of 27.60 ms. The response body is a JSON object with a single key "message": "Invalid destination FOO".

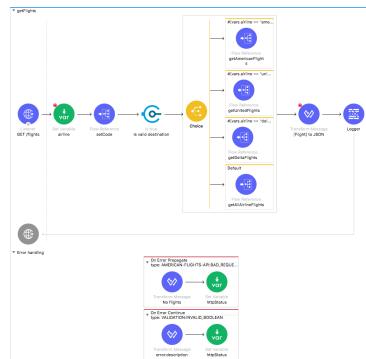
24. Return to Anypoint Studio.

25. Stop the project.

Walkthrough 10-4: Handle errors at the flow level

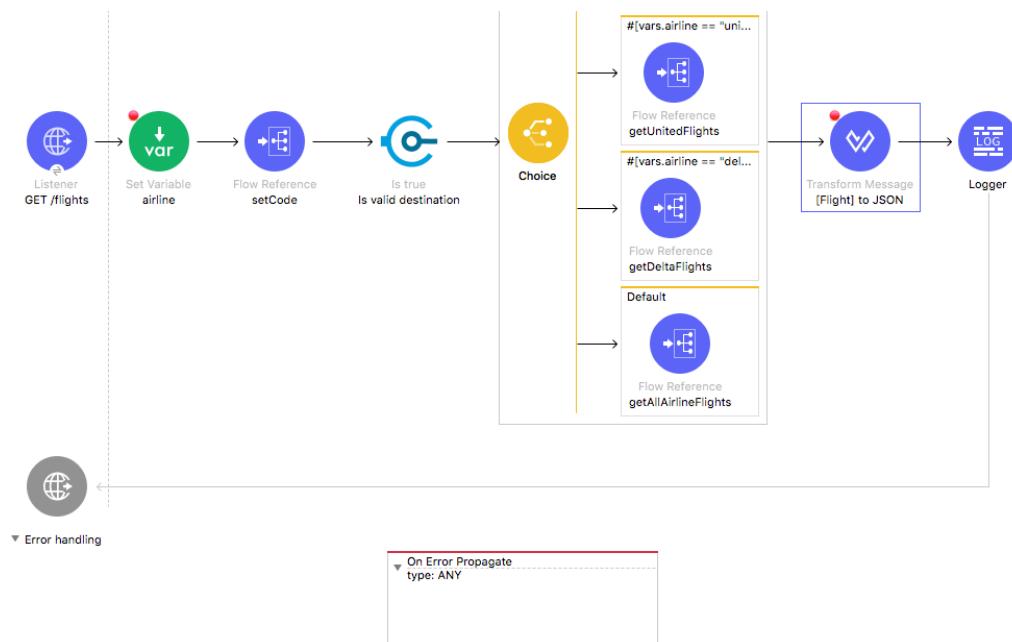
In this walkthrough, you add multiple error handlers to a flow. You will:

- Add error handlers to a flow.
- Test the behavior of errors thrown in a flow and by a child flow.
- Compare On Error Propagate and On Error Continue scopes in a flow.
- Set an HTTP status code in an error handler and modify an HTTP Listener to return it.



Add an On Error Propagate error handler to a flow

1. Return to implementation.xml.
2. Expand the Error handling section of the getFlights flow.
3. Add an On Error Propagate scope.
4. Set the error type to ANY so it will initially catch both the validation and American flights errors.



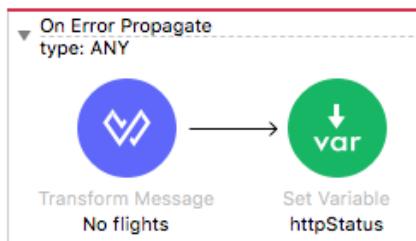
Set the payload in the error handler to the error description

5. Add a Transform Message component to the scope and set the display name to No flights.
6. In the Transform Message properties view, set the payload to a JSON message property equal to the string No flights to and concatenate the code variable.
7. Coerce the variable to a String.

```
1 ⊕ %dw 2.0
2   output application/json
3   ---
4 ⊕ [
5     "message": "No flights to " ++ vars.code as String
6 }
```

Set the HTTP status code in the error handler to 200

8. Add a Set Variable transformer to the On Error Propagate.
9. In the Set Variable properties view, set the display name and name to httpStatus.
10. Set the value to 200.



Set the HTTP Listener error response status code to use a variable

11. Navigate to the Responses tab in the properties view for the GET /flights Listener.
12. Use expression mode to change the error response status code (not the response status code!) to an expression that sets it equal to the value of an httpStatus variable with a default value of 500.

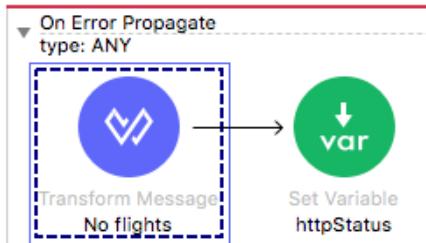
```
vars.httpStatus default 500
```

Error Response

Body:	<input type="button" value="fx"/>	1 payload	<input type="button" value="x"/>
Headers:	<input type="button" value="fx"/>	Headers	
		<input type="button" value="+"/> <input type="button" value="x"/> <input type="button" value="x"/>	
		Name	Value
Status code:	<input type="button" value="fx"/>	# <input type="text" value="vars.httpStatus default 500"/>	<input type="button" value="l"/> <input type="button" value="x"/>
Reason phrase:	<input type="button" value="fx"/>		

Test the flow's On Error Propagate with an error in the flow

13. Save the file, debug the project, and proceed through any errors in the workspace.
14. In Advanced REST Client, change the airline to make a request to
<http://localhost:8081/flights?airline=american&code=FOO>.
15. In the Mule Debugger, step until the event is passed to the flow's error handler.



16. Step through the rest of the application.
17. Return to Advanced REST Client; you should get the 200 status code and the JSON message.

200 OK 81289.88 ms

```
{
  "message": "No flights to FOO"
}
```

Test the flow's On Error Propagate with an error in a child flow

18. In Advanced REST Client, change the code to make a request to
<http://localhost:8081/flights?airline=american&code=PDX>.

19. Step until the event is passed to the globalError_Handler.

The screenshot shows the Mule Debugger interface with the 'global' tab selected. In the top pane, a detailed error stack trace is displayed, including the error description, attributes, correlationId, and various error types. Below this, the 'globalError_Handler' configuration is shown. It contains two 'On Error Propagate' rules: one for 'WSC:CONNECTIVITY, WSC:INVALID...' which uses a 'Transform Message' component with the payload 'Data unavailable'; and another for 'ANY' which also uses a 'Transform Message' component, but with the payload 'error.description'. Both components have a dashed border around them.

20. Step until the error is passed to the parent flow's error handler.

The screenshot shows the Mule Debugger interface with the 'implementation' tab selected. In the top pane, a detailed exception stack trace is shown, including the cause, suppressed exceptions, and the original detail message. Below this, the 'Error handling' configuration is displayed. It features an 'On Error Propagate' rule for 'ANY' that uses a 'Transform Message' component with the payload 'No flights', followed by a 'Set Variable' component named 'var' that sets the variable 'httpStatus'.

21. Step through the rest of the application.
22. Return to Advanced REST Client; you should get the 200 status code and the JSON error message.

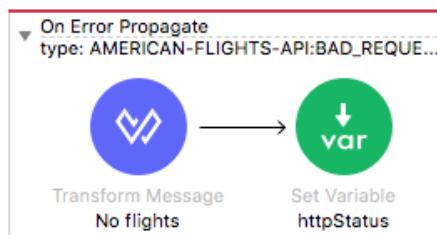
```
200 OK 109017.67 ms

{
    "message": "No flights to PDX"
}
```

23. Return to Anypoint Studio and switch to the Mule design perspective.

Specify the type of error to be handled by the flow's error handler

24. Navigate to the properties view for the On Error Propagate in getFlights.
25. Change the type from ANY to AMERICAN-FLIGHTS-API:BAD_REQUEST.



Test the application

26. Save the file to redeploy the project.
27. In Advanced REST Client, make another request to
<http://localhost:8081/flights?airline=american&code=PDX>.
28. In the Mule Debugger, step through the application; the error should still be handled by globalError_Handler and then the getFlights flow error handler.

```
200 OK 109017.67 ms

{
    "message": "No flights to PDX"
}
```

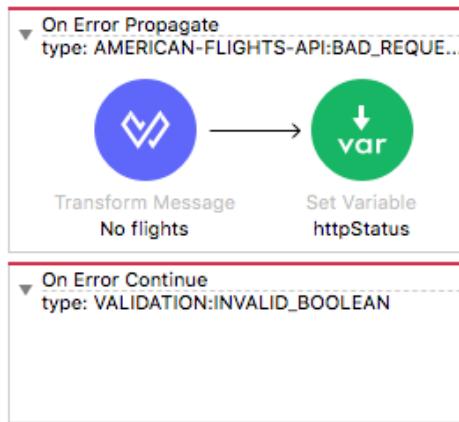
29. In Advanced REST Client, change the code to make a request to <http://localhost:8081/flights?airline=american&code=FOO>.
30. In the Mule Debugger, step through the application; you should get a 500 Server Error and no message because the error is not handled by the getFlights flow error handler or by globalError_handler.

The screenshot shows the Mule Debugger interface. At the top, there are fields for 'Method' (GET), 'Request URL' (http://localhost:8081/flights?airline=american&code=FOO), and a 'SEND' button. Below these are sections for 'Parameters' and 'Headers'. Under 'Parameters', there is a red box labeled '500 Server Error' and '9631.16 ms'. To the right of this box is a 'DETAILS' button. The main body of the response is currently empty.

31. Return to Anypoint Studio and switch to the Mule Design perspective.
32. Stop the project.
33. Navigate to the Responses tab in the properties view for the GET /flights Listener.
34. Review the error response body (not the response body!) and ensure you know why you got the last response for <http://localhost:8081/flights?airline=american&code=FOO>.

Use an On Error Continue error handler in a flow

35. Add an On Error Continue scope to getFlights.
36. In the On Error Continue properties view, set the type to VALIDATION:INVALID_BOOLEAN.



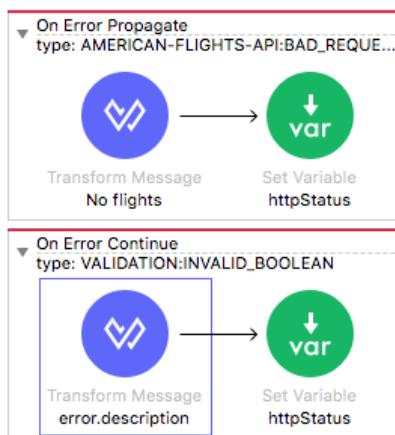
37. Add a Transform Message component to the scope and set the display name to error.description.

38. In the Transform Message properties view, set the payload to a JSON message property equal to the error.description.

```
1 %dw 2.0
2   output application/json
3   ---
4   [
5     "message": error.description
6 }
```

39. Add a Set Variable transformer to the On Error Continue scope.

40. Set the display name and name to httpStatus and set the value to 400.



Test the application

41. Save the file, run the project, and proceed through any errors in the workspace.

42. In Advanced REST Client, make another request to

<http://localhost:8081/flights?airline=american&code=FOO>; you should get a 200 response and the invalid destination message again – not the 400 response.



Set the HTTP Listener response status code

43. Return to Anypoint Studio.

44. Navigate to the Responses tab in the properties view for the GET /flights Listener.

45. Use expression mode to set the response status code (not the error response status code!) to an expression that sets it equal to the value of an httpStatus variable with a default value of 200.

```
vars.HttpStatus default 200
```

Test the application

46. Save the file to redeploy the project.

47. In Advanced REST Client, make another request to

<http://localhost:8081/flights?airline=american&code=FOO>; you should now get the 400 response.

The screenshot shows a screenshot of the Advanced REST Client interface. At the top, there's an orange button labeled "400 Bad Request" and the text "2325.20 ms". Below this is a toolbar with icons for copy, paste, and refresh. The main area contains a JSON response: { "message": "Invalid destination FOO" }.

Test the application to see what happens with the Scatter-Gather

48. In Advanced REST Client, change the code and make a request to

<http://localhost:8081/flights?airline=american&code=PDX>; you should get a 200 response and no flights to PDX.

49. Change the airline to make a request to <http://localhost:8081/flights?airline=united&code=PDX>; you should get United flights to PDX.

The screenshot shows a screenshot of the Advanced REST Client interface. At the top, there's a green button labeled "200 OK" and the text "271.70 ms". Below this is a toolbar with icons for copy, paste, and refresh. The main area contains a JSON response: [Array[3], -0: { "airlineName": "United", "availableSeats": 0, "departureDate": "2015/02/13", "destination": "PDX", "flightCode": "ER49fd", "origination": "MUA", "planeType": "Boeing 777", "price": 853 }, -1: { "airlineName": "United", "availableSeats": 95, "departureDate": "2015/02/20" }].

50. Remove the airline to make a request to <http://localhost:8081/flights?code=PDX>; you should get flights to PDX because United has flights, but you get none.



A screenshot of a browser window showing a 500 Server Error. The error message is as follows:

```
{ "message": "Exception(s) were found for route(s): 0: org.mule.extension.http.api.request.validator.ResponseValidatorTypedException: HTTP GET on resource 'http://training4-american-api-maxmule.us-e2.cloudhub.io:80/flights' failed: bad request (400). 2: org.mule.runtime.api.connection.ConnectionException: Error trying to acquire a new connection:Error fetching the resource [http://mu.learn.mulesoft.com/deltas?wsdl]: http://mu.learn.mulesoft.com/deltas?wsdl" }
```

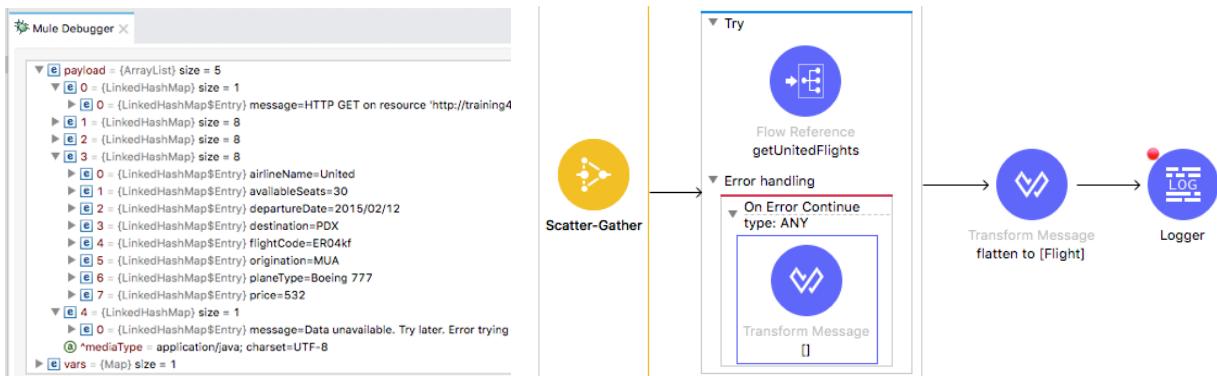
51. Return to Anypoint Studio.

52. Stop the project.

Walkthrough 10-5: Handle errors at the processor level

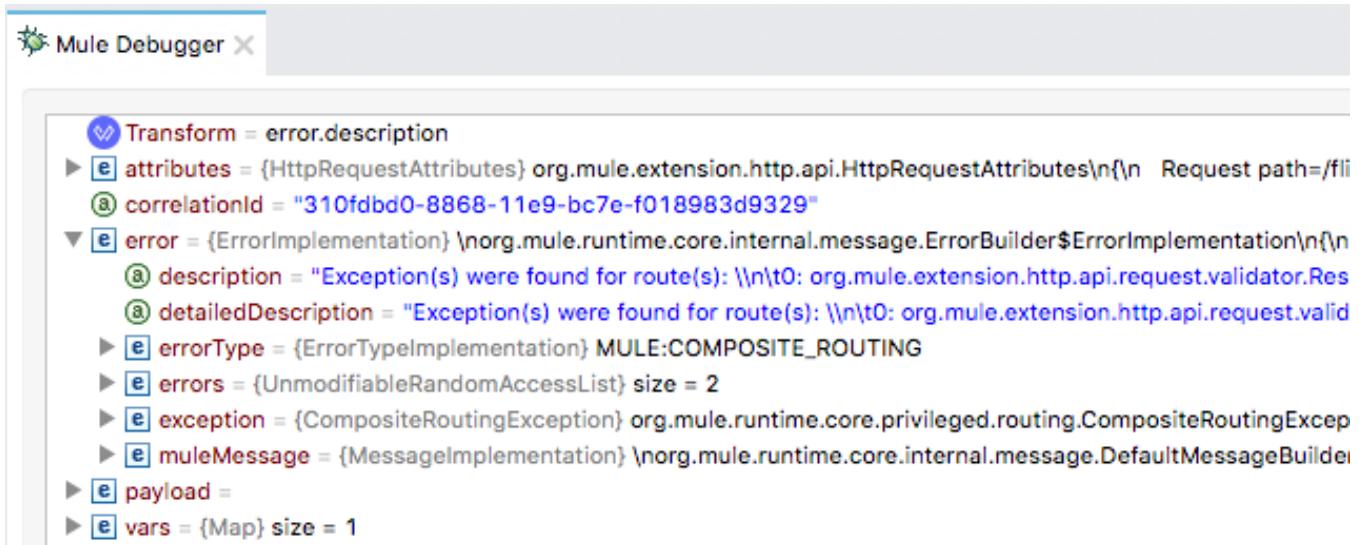
In this walkthrough, you work with the Scatter-Gather in getAllAirlineFlights so that it correctly returns results when one but not all airlines have flights. You will:

- Wrap the Flow Reference in each branch of a Scatter-Gather in a Try scope.
- Use an On Error Continue scope in each Try scope error handler to provide a valid response so flow execution can continue.



Debug the application when a Scatter-Gather branch has an error

1. Return to implementation.xml in Anypoint Studio.
2. Debug the project and proceed through any errors in the workspace.
3. In Advanced REST Client, make another request to <http://localhost:8081/flights?code=PDX>.
4. Return to the Mule Debugger and step through the application until you see a routing exception.



5. Step through the rest of the application.

6. Return to Advanced REST client, you should get the 500 Server Error that there were exceptions in routes 0 and 2.

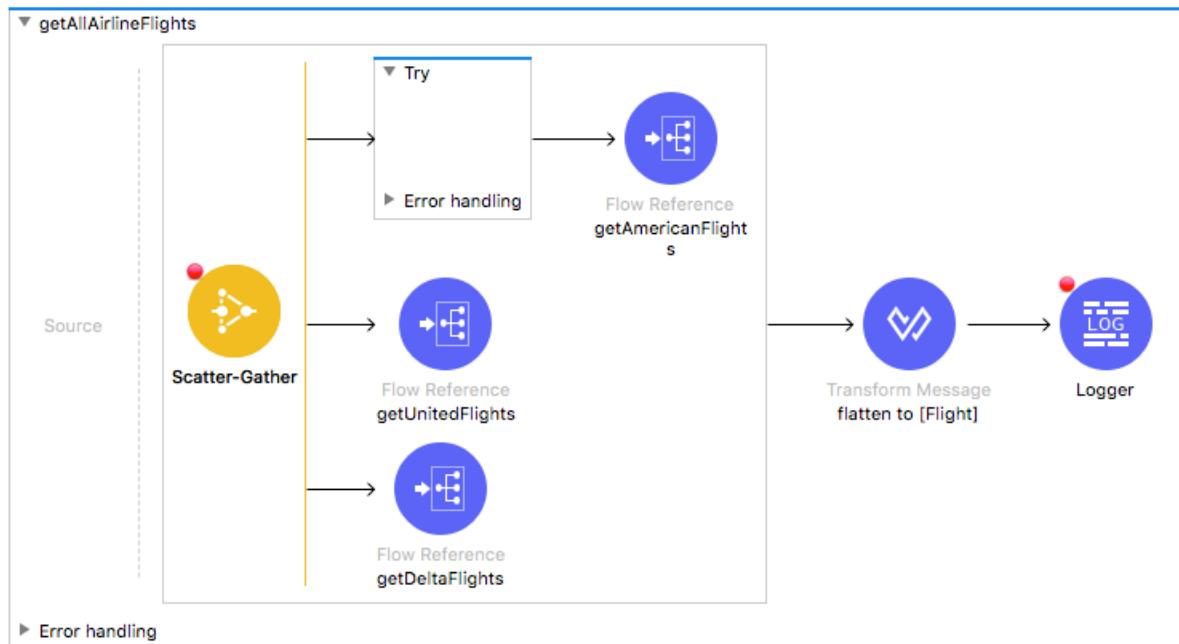
500 Server Error 65571.70 ms DETAILS ▾

{
 "message": "Exception(s) were found for route(s): 0:
 org.mule.extension.http.api.request.validator.ResponseValidatorTypedException: HTTP GET
 on resource 'http://training4-american-api.cloudhub.io:80/flights' failed: bad request
 (400). 2: org.mule.runtime.api.connection.ConnectionException: Error trying to acquire
 a new connection:Error fetching the resource [http://mu.learn.mulesoft.com/deltas?
 wsdl]: http://mu.learn.mulesoft.com/deltas?wsdl"
}

7. Return to Anypoint Studio and switch to the Mule Design perspective.

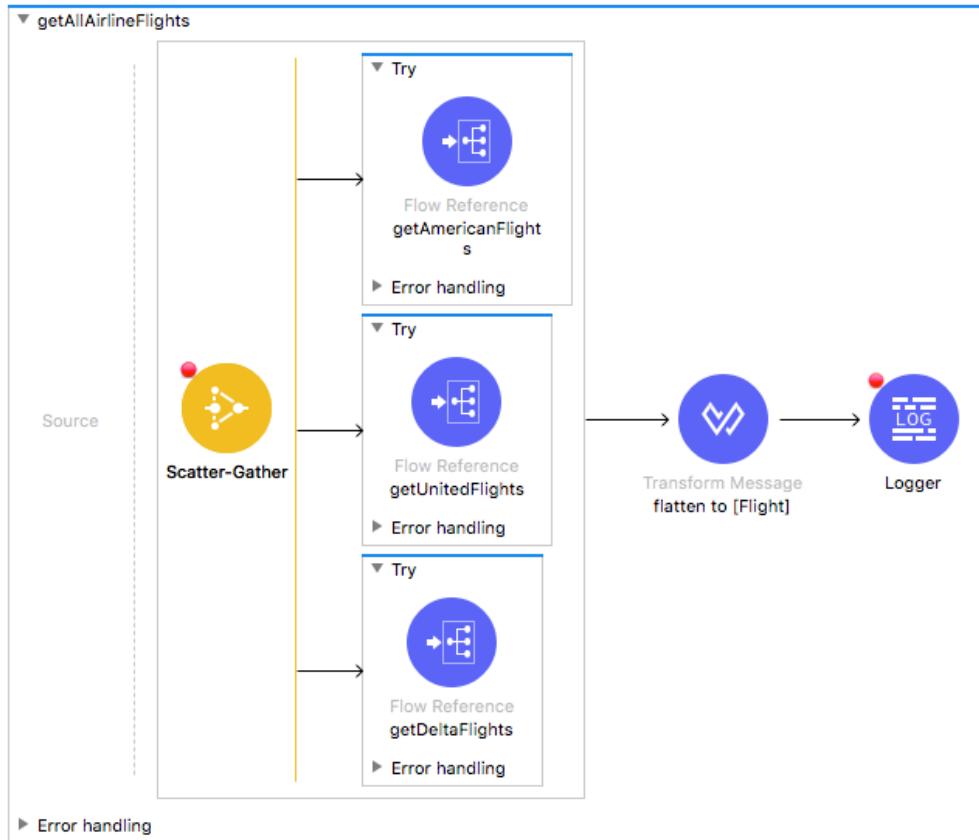
Place each Flow Reference in the Scatter-Gather in a Try scope

8. Locate getAllAirlineFlights.
9. Drag a Try scope from the Mule Palette and drop it in the Scatter-Gather before the getAmericanFlights Flow Reference.



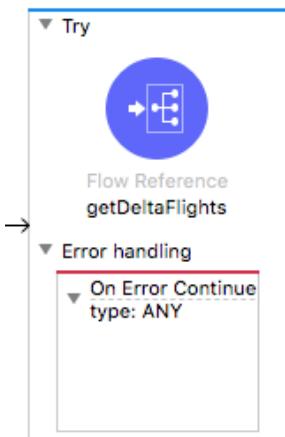
10. Drag the `getAmericanFlights` Flow Reference into the Try scope.
11. Add Try scopes to the other two routes of the Scatter-Gather.

12. Move the getUnitedFlights Flow Reference into one of the Try scopes and the getDeltaFlights Flow Reference into the other.



Add an error handler to each branch that passes through the error message

13. Expand the error handling section of the first Try scope.
14. Add an On Error Continue scope and set the types of errors it handles to ANY.
15. Repeat these steps to add the same error handler to the two other Try scopes.



Debug the application

16. Save the file to redeploy the project.
17. In Advanced REST Client, make another request to <http://localhost:8081/flights?code=PDX>.
18. In the Mule Debugger, step through the application until you are at the Logger after the Scatter-Gather in getAllAirlineFlights.
19. Expand Payload; you should see three flights (from United) and two error messages.

Mule Debugger

```
[{"id": "payload", "type": "ArrayList", "size": 5, "children": [{"id": "0", "type": "LinkedHashMap", "size": 1, "children": [{"id": "0", "type": "LinkedHashMap$Entry", "message": "HTTP GET on resource 'http://training4-american-api.cloudhub.io:80/flights' failed: bad request (400)."}]}, {"id": "1", "type": "LinkedHashMap", "size": 8, "children": [{"id": "1", "type": "LinkedHashMap$Entry", "airlineName": "United"}, {"id": "2", "type": "LinkedHashMap$Entry", "availableSeats": 30}, {"id": "3", "type": "LinkedHashMap$Entry", "departureDate": "2015/02/12"}, {"id": "4", "type": "LinkedHashMap$Entry", "destination": "PDX"}, {"id": "5", "type": "LinkedHashMap$Entry", "flightCode": "ER04kf"}, {"id": "6", "type": "LinkedHashMap$Entry", "origination": "MUA"}, {"id": "7", "type": "LinkedHashMap$Entry", "planeType": "Boeing 777"}, {"id": "8", "type": "LinkedHashMap$Entry", "price": 532}], "vars": {"vars": 1}}, {"id": "implementation", "label": "mua-flights-api.raml", "type": "implementation"}, {"id": "global", "label": "global"}, {"id": "config.yaml", "label": "config.yaml"}, {"id": "FlightsExample.raml", "label": "FlightsExample.raml"}, {"id": "Source", "label": "Source"}, {"id": "Scatter-Gather", "label": "Scatter-Gather"}, {"id": "Flow Reference", "label": "Flow Reference getUnitedFlights"}, {"id": "Error handling", "label": "Error handling"}, {"id": "On Error Continue", "label": "On Error Continue type: ANY"}, {"id": "Transform Message", "label": "Transform Message flatten to [Flight]"}, {"id": "Logger", "label": "Logger"}
```

20. Step through the rest of the application.
21. Return to Advanced REST Client; you should see United flights and error messages.

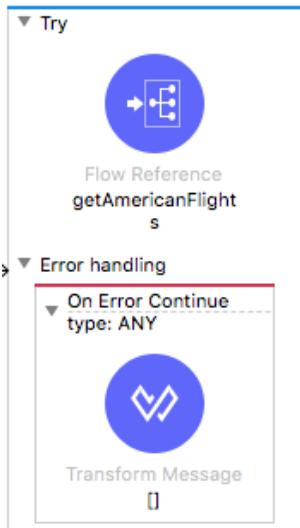
200 OK 120611.60 ms DETAILS ▾

Array[5]

```
[{"-0": {"message": "HTTP GET on resource 'http://training4-american-api.cloudhub.io:80/flights' failed: bad request (400)."}, {"-1": {...}, "-2": {...}, "-3": {...}, "-4": {"message": "Data unavailable. Try later. Error trying to acquire a new connection:Error fetching the resource [http://mu.learn.mulesoft.com/deltas?wsdl]: http://mu.learn.mulesoft.com/deltas?wsdl"}]}
```

Modify each error handler to set the payload to an empty array

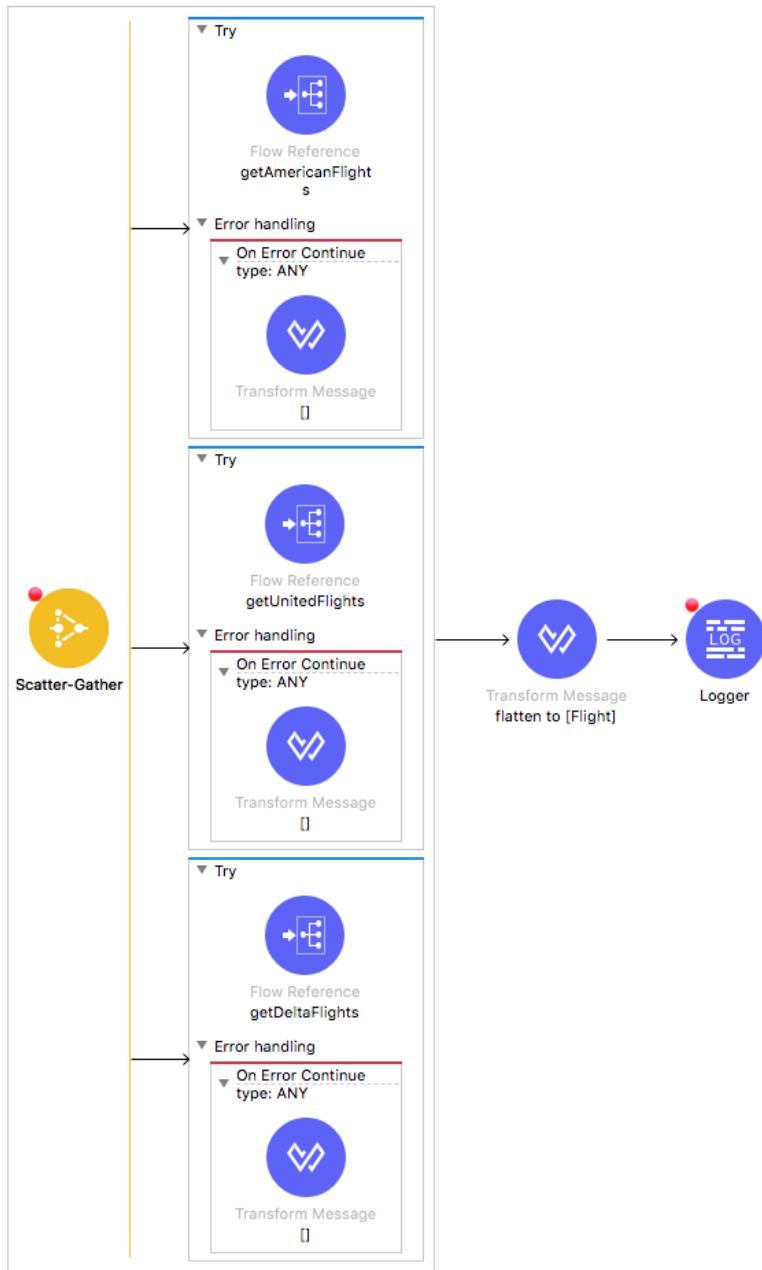
22. Return to Anypoint Studio and switch to the Mule Design perspective.
23. In getAllAirlineFlights, add a Transform Message component to one of the On Error Continue scopes.
24. Set the display name to [].



25. In the Transform Message properties view, set the payload to an empty array.

```
1 %dw 2.0
2 output application/java
3 ---
4 [ ]
```

26. Repeat the steps to add the same Transform Message component to the two other error handlers.



Debug the application

27. Save the file to redeploy the project.
28. In Advanced REST Client, make another request to <http://localhost:8081/flights?code=PDX>.
29. In the Mule Debugger, step through the application until you are at the Logger after the Scatter-Gather in getAllAirlineFlights.

30. Expand Payload; you should now see the three flights and no error messages.

The screenshot shows the Mule Studio interface. At the top, there's a 'Mule Debugger' window displaying the payload structure:

```
payload = {ArrayList} size = 3
  0 = {LinkedHashMap} size = 8
    0 = {LinkedHashMap$Entry} airlineName=United
    1 = {LinkedHashMap$Entry} availableSeats=0
    2 = {LinkedHashMap$Entry} departureDate=2015/02/13
    3 = {LinkedHashMap$Entry} destination=PDX
    4 = {LinkedHashMap$Entry} flightCode=ER49fd
    5 = {LinkedHashMap$Entry} origination=MUA
    6 = {LinkedHashMap$Entry} planeType=Boeing 777
    7 = {LinkedHashMap$Entry} price=853
  1 = {LinkedHashMap} size = 8
  2 = {LinkedHashMap} size = 8
  ^mediaType = application/java; charset=UTF-8
vars = {Map} size = 1
```

Below the debugger is the Mule flow diagram:

```
graph LR
    Source[Source] --> ScatterGather((Scatter-Gather))
    ScatterGather --> getUnitedFlights[getUnitedFlights]
    getUnitedFlights --> ErrorHandling[Error handling]
    ErrorHandling -- "On Error Continue type: ANY" --> Transform[Transform Message  
flatten to [Flight]]
    Transform --> Logger[Logger]
```

The flow starts with a 'Source' component, followed by a 'Scatter-Gather' component. The 'Scatter-Gather' component connects to a 'getUnitedFlights' service. This is followed by an 'Error handling' section with the configuration 'On Error Continue type: ANY'. Finally, the flow goes through a 'Transform Message' component with the rule 'flatten to [Flight]' and ends at a 'Logger' component.

31. Step through the rest of the application.

32. Return to Advanced REST Client; you should now only see the three flights.

The screenshot shows the Advanced REST Client results page. The status is '200 OK' and the time taken is '108441.70 ms'. The response body is an array of three flight objects:

```
[Array[3]
-0: { ... }
-1: { ... }
-2: {
  "airlineName": "United",
  "availableSeats": 30,
  "departureDate": "2015/02/12",
  "destination": "PDX",
  "flightCode": "ER04kf",
  "origination": "MUA",
  "planeType": "Boeing 777",
  "price": 532
}
```

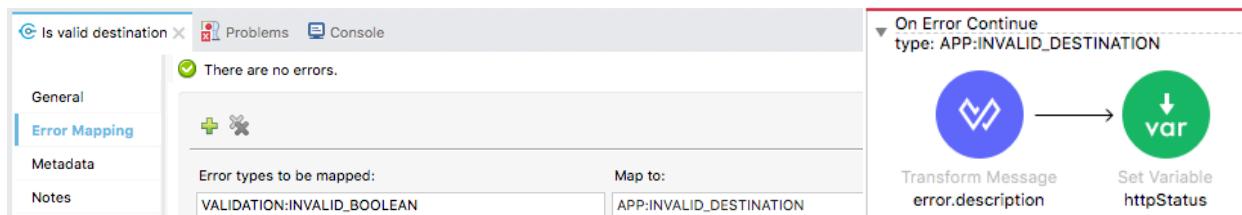
33. Return to Anypoint Studio and switch to the Mule Design perspective.

34. Stop the project.

Walkthrough 10-6: Map an error to a custom error type

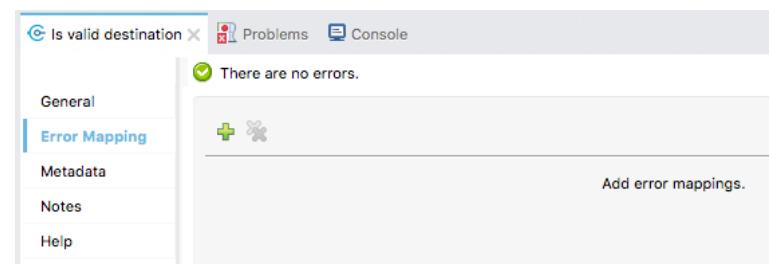
In this walkthrough, you map the Validation error to a custom error type for the application. You will:

- Map a module error to a custom error type for an application.
- Create an event handler for the custom error type.

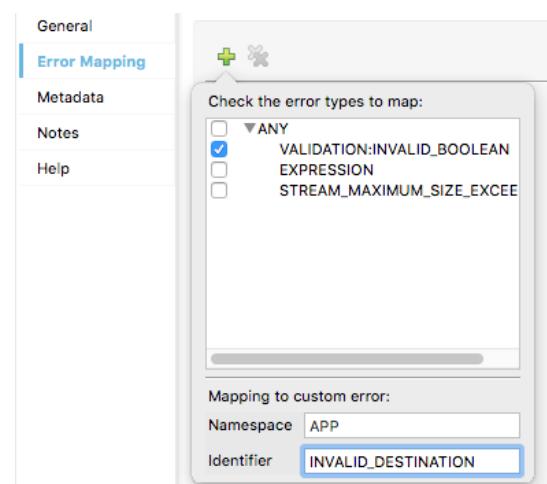


Map a validation module error to a custom error type

1. Return to implementation.xml.
2. Navigate to the properties view for the validator in getFlights.
3. Select the Error Mapping tab.
4. Click the Add new mapping button.



5. Select the VALIDATION:INVALID_BOOLEAN error type.
6. Leave the namespace of the custom error to map to set to APP.
7. Set the identifier to INVALID_DESTINATION.

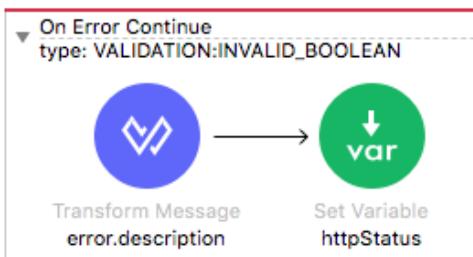


8. Press Enter; you should see your new error mapping.

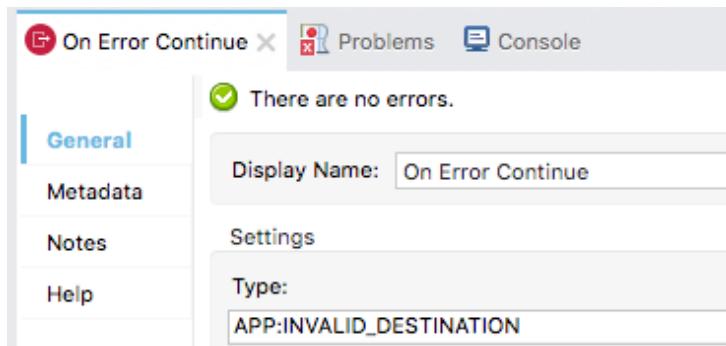


Change the existing validation error handler to catch the new custom type

9. Navigate to the properties view for the getFlights validation On Error Continue error handler.



10. Change the type from VALIDATION:INVALID_BOOLEAN to the new APP:INVALID_DESTINATION that should now appear in the type drop-down menu.



Test the application

11. Save the file, run the project, and proceed through any errors in the workspace.

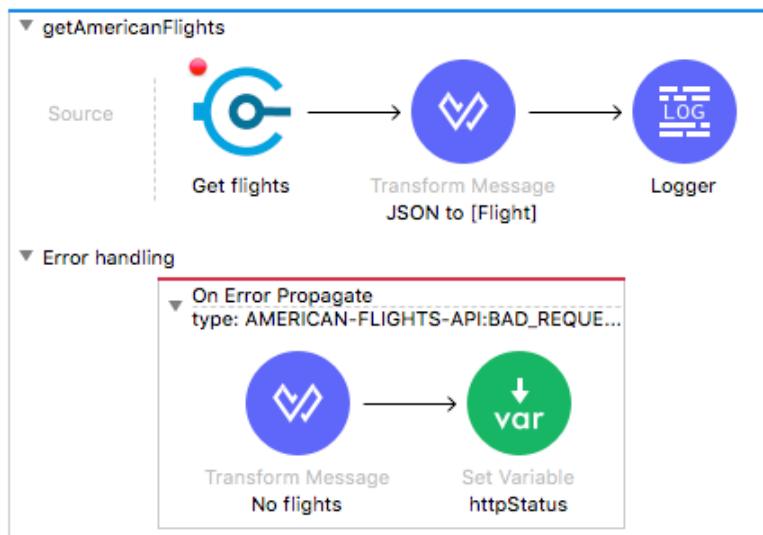
12. In Advanced REST Client, change the code and make a request to <http://localhost:8081/flights?code=FOO>; you should still get a 400 response with the invalid destination error.

400 Bad Request 2325.20 ms

{
 "message": "Invalid destination FOO"
}

Move the American error handler into the American flow

13. Collapse the getAllAirlineFlights and setCode flows.
14. Move the AMERICAN-FLIGHTS-API error scope from getFlights into getAmericanFlights.

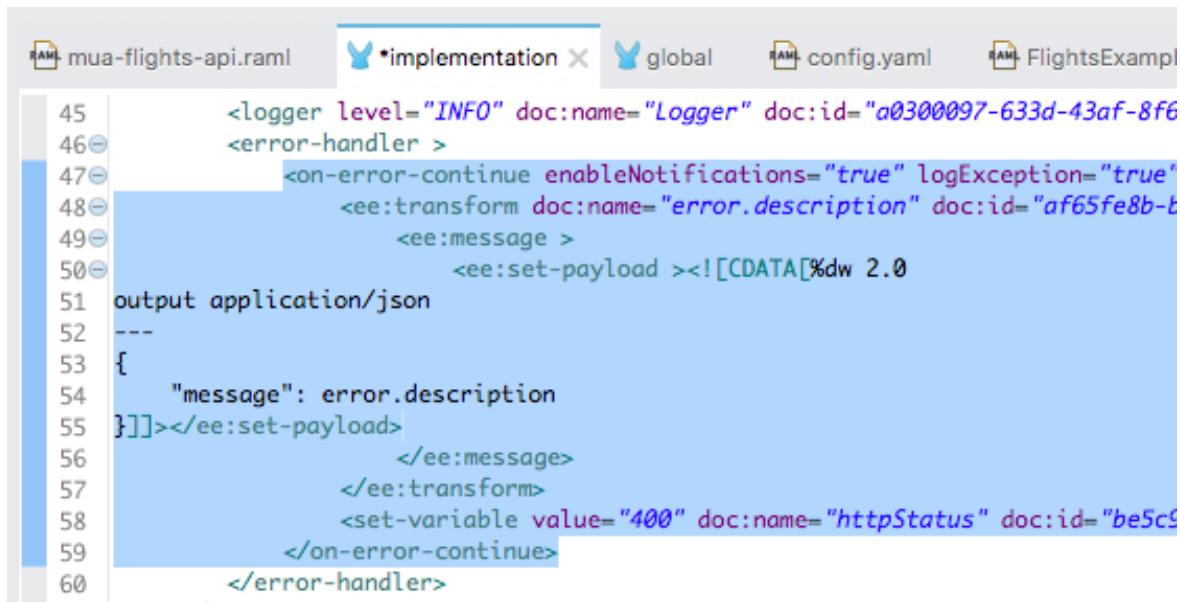


Move the error handler to the global error handler

Note: The following steps have instructions to make changes in the XML. If you prefer, you can copy and paste the error handler to move it between files and then delete the original.

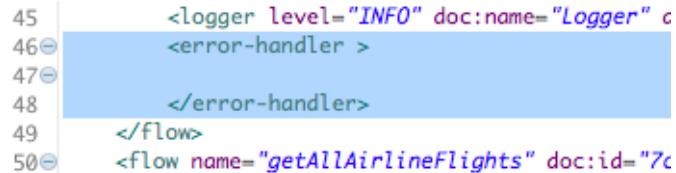
15. Right-click the validation On Error Continue error handler in getFlights and select Go to XML.

16. Select and cut the APP:INVALID_DESTINATION on-error-continue.



```
45     <logger level="INFO" doc:name="Logger" doc:id="a0300097-633d-43af-8f6
46     <error-handler>
47         <on-error-continue enableNotifications="true" logException="true"
48             <ee:transform doc:name="error.description" doc:id="af65fe8b-b
49                 <ee:message>
50                     <ee:set-payload><![CDATA[%dw 2.0
51 output application/json
52 ---
53 {
54     "message": error.description
55 }]]></ee:set-payload>
56             </ee:message>
57         </ee:transform>
58         <set-variable value="400" doc:name="httpStatus" doc:id="be5c9
59             </on-error-continue>
60         </error-handler>
```

17. Delete the remaining, empty error-handler tag set.



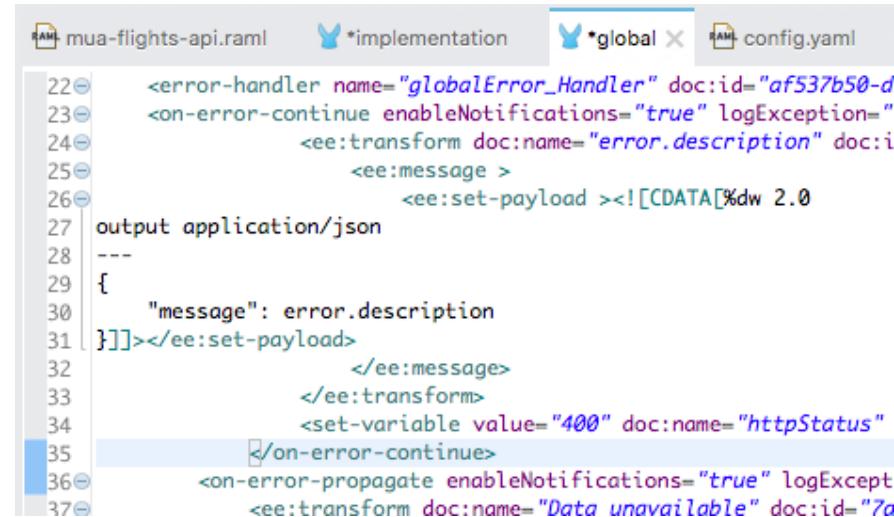
```
45     <logger level="INFO" doc:name="Logger" c
46     <error-handler>
47         </error-handler>
48     </flow>
49     <flow name="getAllAirlineFlights" doc:id="7c
```

18. Return to global.xml.

19. Right-click the globalError_Handler and select Go to XML.

20. Place the cursor on a new line inside and at the start of the globalError_Handler error-handler.

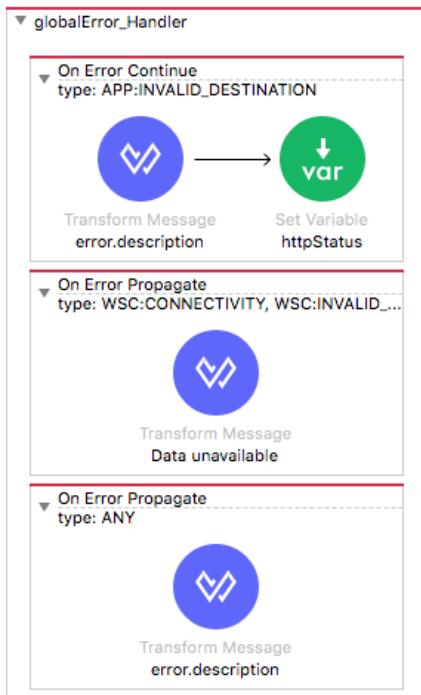
21. Paste the on-error-continue you cut to the clipboard.



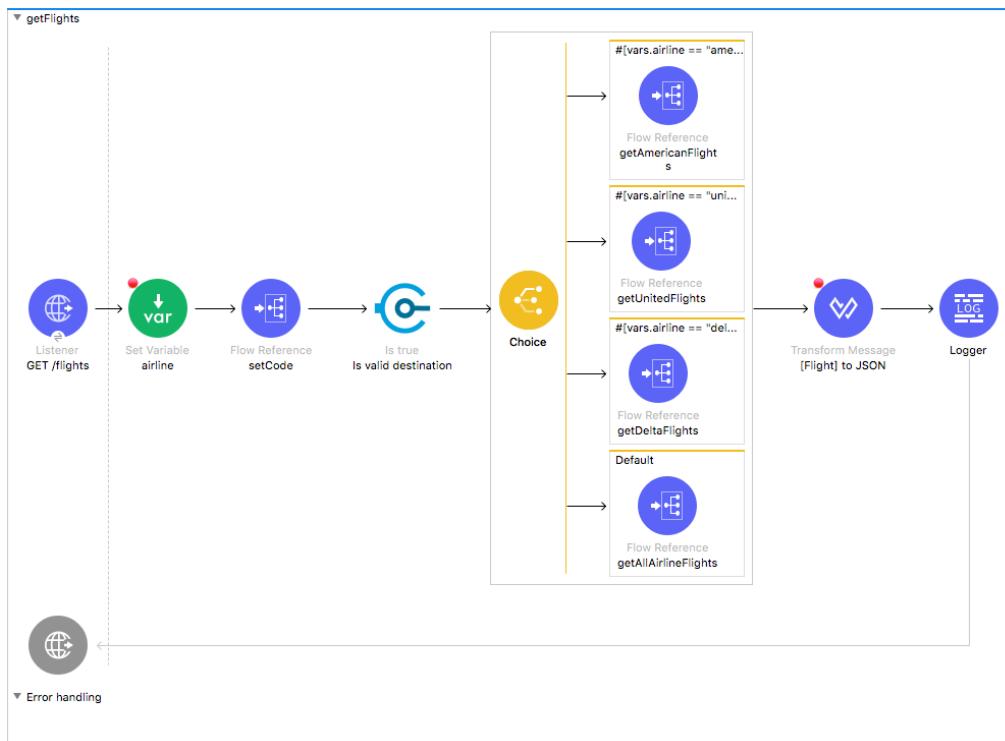
```
22     <error-handler name="globalError_Handler" doc:id="af537b50-d
23         <on-error-continue enableNotifications="true" logException="
24             <ee:transform doc:name="error.description" doc:i
25                 <ee:message>
26                     <ee:set-payload><![CDATA[%dw 2.0
27 output application/json
28 ---
29 {
30     "message": error.description
31 }]]></ee:set-payload>
32             </ee:message>
33         </ee:transform>
34         <set-variable value="400" doc:name="httpStatus"
35             </on-error-continue>
36         <on-error-propagate enableNotifications="true" logExcept
37             <ee:transform doc:name="Data unavailable" doc:id="7a
```

Note: If you are prompted to regenerate ID values, click Yes.

22. Switch back to the Message Flow view; you should see the APP:INVALID_DESTINATION handler.



23. Return to implementation.xml and switch to the Message Flow view; you should no longer see an error handler in getFlights.



Test the application

24. Save the files to redeploy the project.
25. In Advanced REST Client, make another request to <http://localhost:8081/flights?code=FOO>; you should still get a 400 response with the invalid destination error.

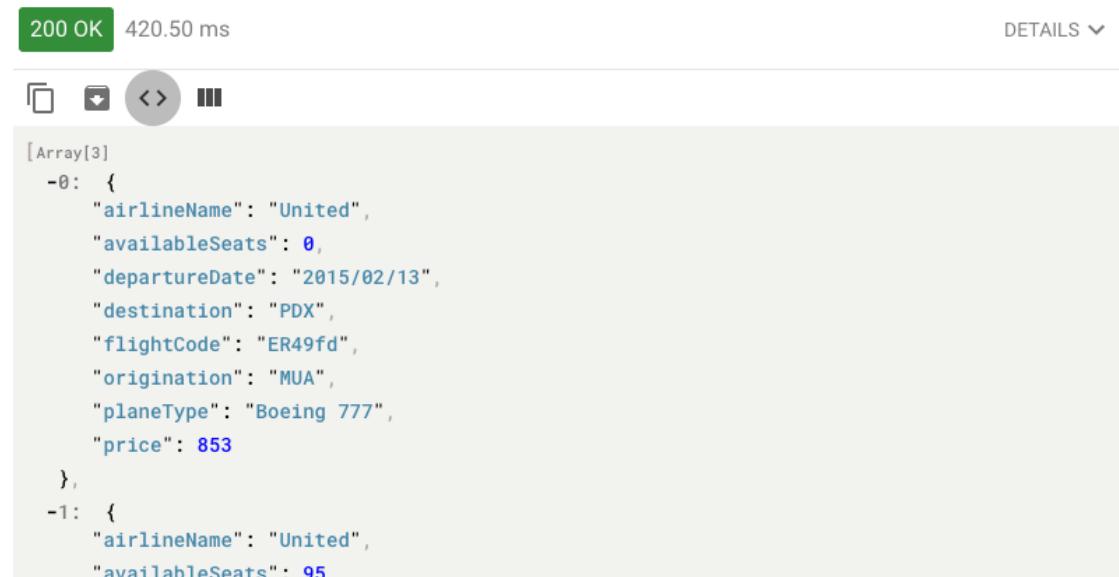
400 Bad Request 2325.20 ms



```
{  
  "message": "Invalid destination FOO"  
}
```

26. Change the code and make a request to <http://localhost:8081/flights?code=PDX>; you should still get only United flights to PDX.

200 OK 420.50 ms DETAILS ▾

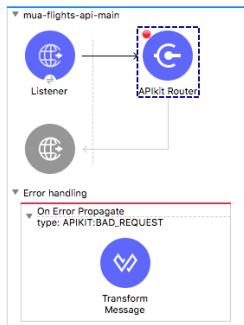


```
[Array[3]  
-0: {  
  "airlineName": "United",  
  "availableSeats": 0,  
  "departureDate": "2015/02/13",  
  "destination": "PDX",  
  "flightCode": "ER49fd",  
  "origination": "MUA",  
  "planeType": "Boeing 777",  
  "price": 853  
},  
-1: {  
  "airlineName": "United",  
  "availableSeats": 95
```

Walkthrough 10-7: Review and integrate with APIkit error handlers

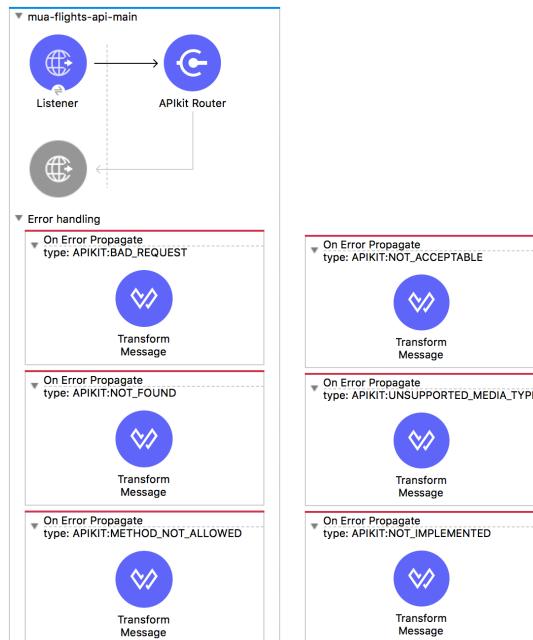
In this walkthrough, you connect the application's implementation to the interface and ensure all the error handling works. You will:

- Review the error handlers generated by APIkit.
- Review settings for the APIkit Router and HTTP Listener in the APIkit generated interface.
- Connect the implementation to the interface and test the error handling behavior.
- Modify implementation error scopes so they work with the APIkit generated interface.



Review APIkit generated error handlers

1. Return to apdev-flights-ws in Anypoint Studio.
2. Open interface.xml.
3. Review the error handling section in mua-flights-api-main.



- Review the types of errors handled by each error handler scope.
- Navigate to the Transform Message properties view for the first error handling scope.
- Review the expression that sets the payload.

Output Payload  

1 `%dw 2.0`
 2 `output application/json`
 3 `---`
 4 `{message: "Bad request"}`

- Use the output drop-down menu to change the output to Variable – httpStatus.

Output Payload  

1 `%d`
 2 `ou`
 3 `---`
 4 `{message: "Bad request"}`

A dropdown menu is open over the code editor, showing two options:
 ✓ Payload
Variable - httpStatus

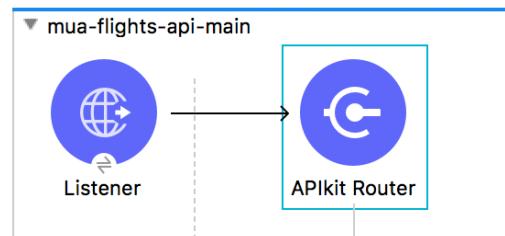
- Review the expression that sets an httpStatus variable; you should see that for a bad request the httpStatus variable is set to 400.

Output Variable - httpStatus  

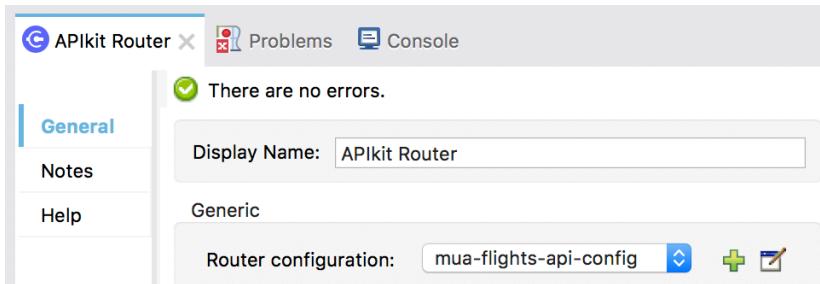
1 `400`

Review settings for the APIkit Router

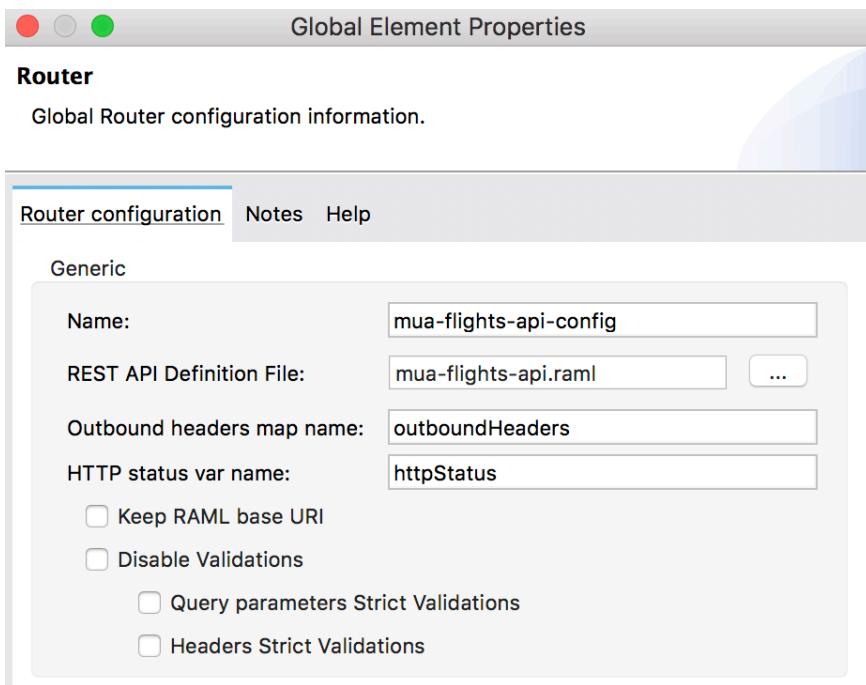
- Navigate to the properties view for the APIkit Router in mua-flights-api-main.



10. Click the Edit button next to router configuration.



11. In the Global Element Properties dialog box, locate the HTTP status var name setting; you should see `httpStatus`.



12. Click OK.

Review settings for the HTTP Listener

13. Navigate to the Responses tab in the properties view for the HTTP Listener in `mua-flights-api-main`.

14. Review the response body and status code.

Response

Body:	<code>fx</code>	1 <code>payload</code>	<code>fx</code>
Headers:	<code>fx</code>	1 <code>vars.outboundHeaders.default {}</code>	<code>fx</code>
Status code:	<code>fx</code>	<code>#[vars.httpStatus default 200]</code>	<code>fx</code>
Reason phrase:	<code>fx</code>		

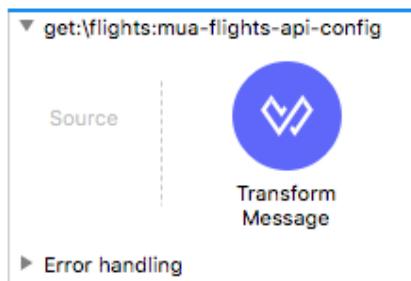
15. Review the error response body and status code.

Error Response

Body:	<code>fx</code>	1 <code>payload</code>	<code>fx</code>
Headers:	<code>fx</code>	1 <code>vars.outboundHeaders.default {}</code>	<code>fx</code>
Status code:	<code>fx</code>	<code>#[vars.httpStatus default 500]</code>	<code>fx</code>
Reason phrase:	<code>fx</code>		

Connect the interface to the implementation

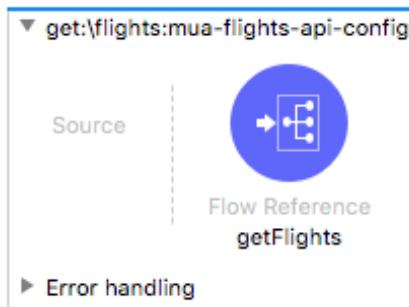
16. In interface.xml, locate the get:\flights flow.



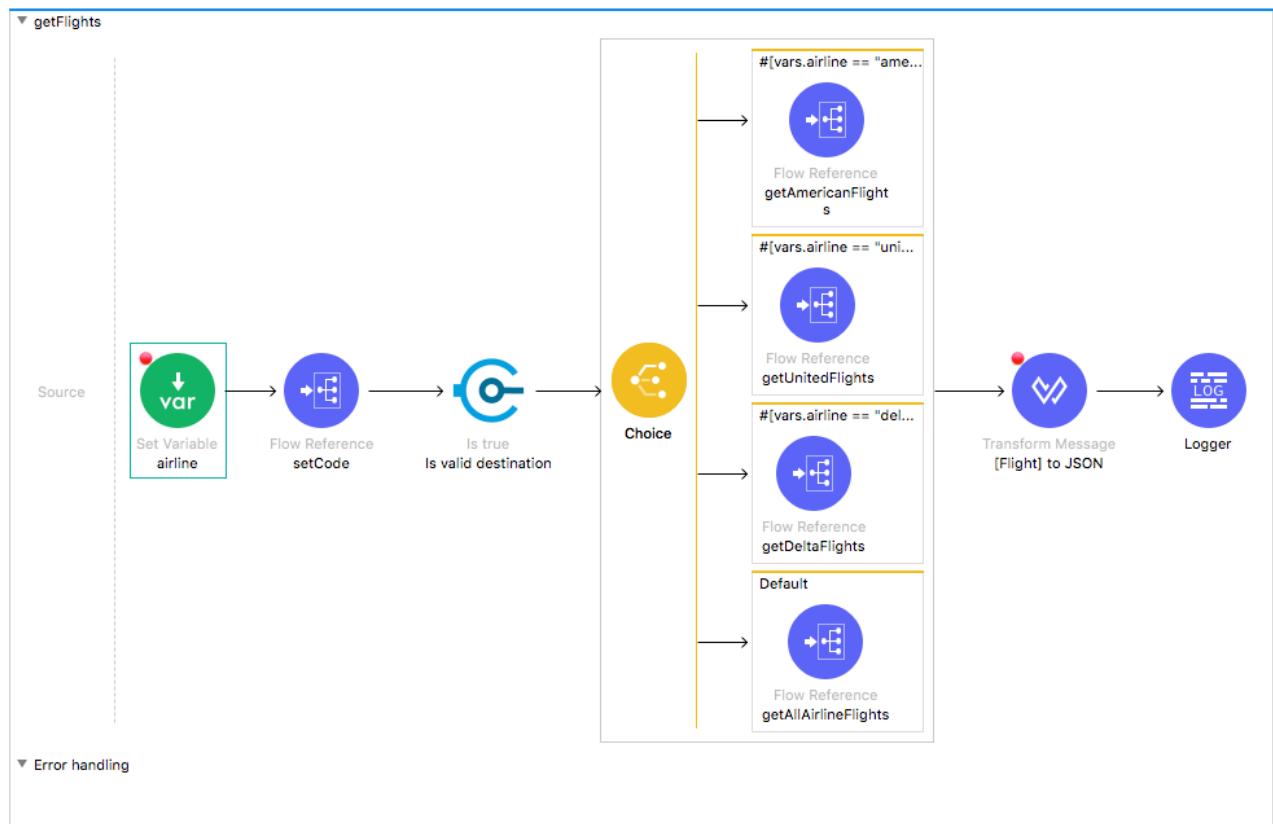
17. Review the default transformation in this flow.

18. Delete the Transform Message component.

19. Add a Flow Reference component to the flow.
20. In the Flow Reference properties view, set the flow name and display name to getFlights.



21. Return to implementation.xml.
22. Delete the GET /flights Listener in getFlights.



23. Save all the files to redeploy the application.

Test the application

24. In Advanced REST Client, make another request to <http://localhost:8081/flights?code=PDX>; you should get a 404 response with a no listener message.

The screenshot shows the Advanced REST Client interface. At the top, there is an orange button labeled "404 Not Found" and "102.11 ms". Below the status bar are several icons: a refresh symbol, a download symbol, a copy/paste symbol, a comparison symbol, and an eye symbol. The main content area displays the message "No listener for endpoint: /flights".

25. Add /api to the URL and make a request to <http://localhost:8081/api/flights?code=PDX>; you should get the Untied flights to PDX as before.

The screenshot shows the Advanced REST Client interface. At the top, there is a green button labeled "200 OK" and "420.50 ms". To the right, there is a "DETAILS" dropdown menu. Below the status bar are icons: a refresh symbol, a download symbol, a copy/paste symbol, a comparison symbol, and a three-line symbol. The main content area displays an array of flight data. The first item in the array is:

```
[Array[3]
-0: {
  "airlineName": "United",
  "availableSeats": 0,
  "departureDate": "2015/02/13",
  "destination": "PDX",
  "flightCode": "ER49fd",
  "origination": "MUA",
  "planeType": "Boeing 777",
  "price": 853
},
-1: {
  "airlineName": "United",
  "availableSeats": 95
}]
```

26. Change the code to make a request to <http://localhost:8081/api/flights?code=FOO>; you should now get a 400 Bad Request response instead of your custom message.

The screenshot shows the Advanced REST Client interface. At the top, there is an orange button labeled "400 Bad Request" and "39.57 ms". Below the status bar are icons: a refresh symbol, a download symbol, a copy/paste symbol, a comparison symbol, and a three-line symbol. The main content area displays a JSON object with a single key-value pair:

```
{
  "message": "Bad request"
}
```

27. Remove the code to make a request to <http://localhost:8081/api/flights>; you should now get your custom error message.

```
400 Bad Request 22660.33 ms

{
  "message": "Invalid destination"
}
```

28. Add the airline and code to make a request to

<http://localhost:8081/api/flights?airline=american&code=PDX>; you should now get a 200 response with the bad request description in the error message.

```
200 OK 136.16 ms DETAILS ▾

COPY SAVE SOURCE VIEW DATA TABLE

{
  "message": "HTTP GET on resource 'http://training4-american-api.cloudhub.io:80/flights' failed: bad request (400)."
}
```

Review the API

29. Return to Anypoint Studio and stop the project.

30. Return to mua-flights-api.raml and review the code; you should see the code query parameter is not required but it has allowed values enumerated.

```
queryParameters:
  code:
    displayName: Destination airport code
    required: false
    enum:
      - SFO
      - LAX
      - PDX
      - CLE
      - PDF
```

Debug the application

31. Return to interface.xml.

32. Add a breakpoint to the APIkit Router in mua-flights-api-main.
33. Debug the project and proceed through any errors in the workspace.
34. In Advanced REST Client, make another request to
<http://localhost:8081/api/flights?airline=american&code=PDX>.
35. In the Mule Debugger, watch the payload and variables and step through the application.
36. Step back to the APIkit router.
37. Review the exception, the payload, and the variables.

```

Router =
attributes = {HttpRequestAttributes} org.mule.extension.http.api.HttpRequestAttributes
correlationId = "f4466300-06f1-11ea-b8ed-f018983d9329"
error = (ErrorImplementation) \org.mule.runtime.core.internal.message.ErrorBuilder$ErrorImplementation
description =
  description = "HTTP GET on resource 'http://training4-american-api.cloudhub.io:80/flights' failed: bad request (400)."
  detailedDescription = "HTTP GET on resource 'http://training4-american-api.cloudhub.io:80/flights' failed: bad request (400)."
errorType = (ErrorTypeImplementation) AMERICAN-FLIGHTS-API:BAD_REQUEST
errors = {UnmodifiableRandomAccessList} size = 0
exception = (ResponseValidatorTypedException) org.mule.extension.http.api.request.validator.ResponseValidatorTypedException
muleMessage = (MessageImplementation) \org.mule.runtime.core.internal.message.DefaultMessageBuilder$MessageImpl
payload = {In "message": "HTTP GET on resource 'http://training4-american-api.cloudhub.io:80/flights' failed: bad request (400)."}
vars = {Map} size = 4
airline = "american"
code = "PDX"
httpStatus = "200"
outboundHeaders = {HashMap} size = 0
  
```

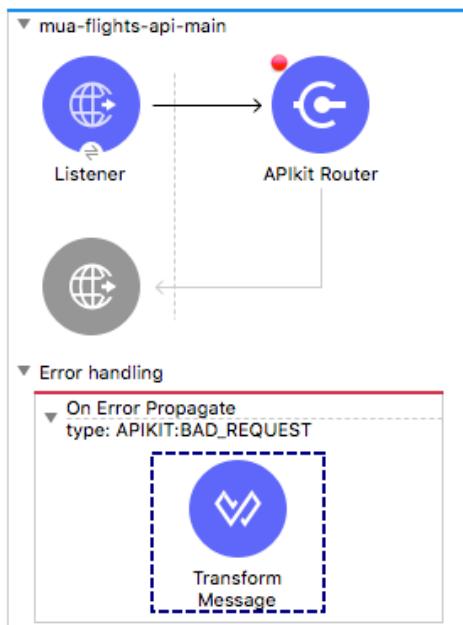
mua-flights-api.raml implementation global config.yaml FlightsExample.raml interface

```

graph LR
    Listener((Listener)) --> APIkit[APIkit Router]
  
```

38. Resume through the rest of the application.
39. In Advanced REST Client, change the code to make a request to
<http://localhost:8081/api/flights?airline=american&code=FOO>.
40. In the Mule Debugger, step through the application; the APIkit router should immediately throw an error.

41. Step again; you should see the error is handled by the APIKIT:BAD_REQUEST handler.



42. Resume through the rest of the application.

43. In Advanced REST Client, remove the airline and code to make a request to

<http://localhost:8081/api/flights>.

44. In the Mule Debugger, step through the application; the validator should throw an error and execution should **not** return to the APIkit router.

45. Return to Advanced REST Client; you should successfully get a 400 response with the custom message.

400 Bad Request 22660.33 ms



```
{  
    "message": "Invalid destination "  
}
```

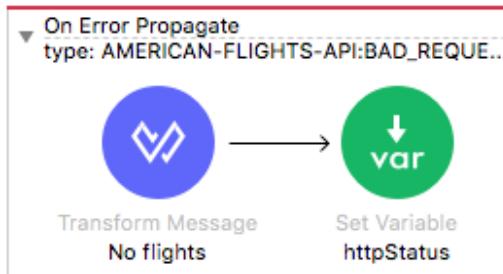
Note: If you had specified the code query parameter to be required in the RAML file from which the interface was generated, the APIkit router would catch this immediately and respond with a 400 Bad Request response. The event would never get to the validator in your implementation.

Change the American flights error scope to On Error Continue

46. Return to Anypoint Studio and switch to the Mule Design perspective.

47. Return to implementation.xml.

48. Locate the error handler in the getAmericanFlights flow.



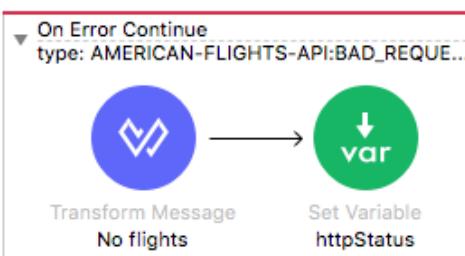
49. Right-click it and select Go To XML.

50. Change the on-error-propagate start and end tags in the getAmericanFlights flow to on-error-continue.

```
<logger level="INFO" doc:name="Logger" doc:id="1c122">
<error-handler>
    <on-error-continue enableNotifications="true" lo...
        <ee:transform doc:name="No flights" doc:id="...
            <ee:message>
                <ee:set-payload><! [CDATA[%dw 2.0
output application/json
---
{
    "message": "No flights to " ++ vars.code as String
}]]></ee:set-payload>
            </ee:message>
        </ee:transform>
        <set-variable value="200" doc:name="httpStat...
    </on-error-continue>
</error-handler>
</flow>
```

51. Change the doc:name in the start tag to On Error Continue.

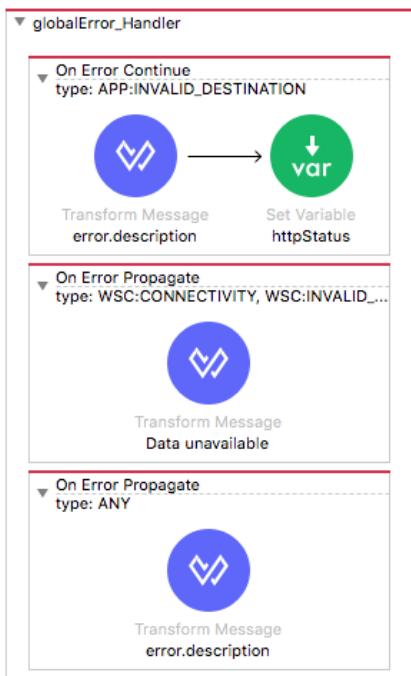
52. Switch back to the Message Flow view; you should now see an On Error Continue.



Change the global default error scopes to On Error Continue

53. Return to global.xml.

54. Review the types of error handler scopes.

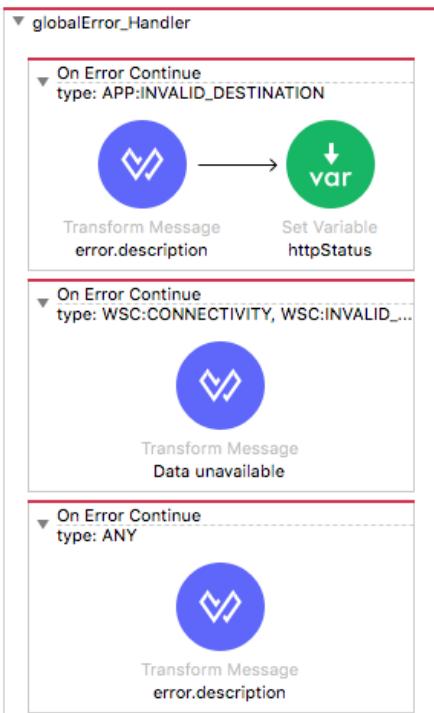


55. Right-click the WSC error handler and select Go To XML.

56. Change the four on-error-propagate start and end tags to on-error-continue.

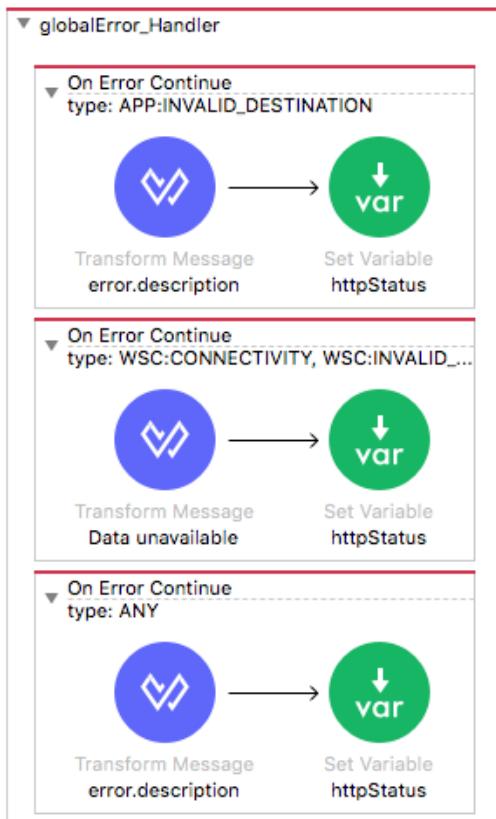
57. Change the doc:names in both start tags to On Error Continue.

58. Switch back to the Message Flow view; you should now see both are On Error Continue scopes.



Set the HTTP status code for the On Error Continue scopes so you do not get 200

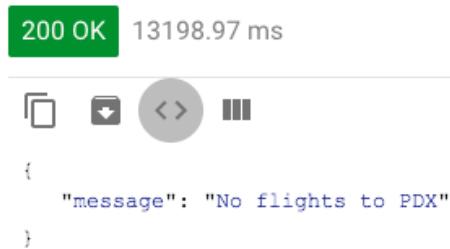
59. Add a Set Variable transformer to the WSC error handler.
60. In the Set Variable properties view, set the display name and name to httpStatus.
61. Set the value to 500.
62. Copy the Set Variable transformer and add it to the ANY error scope.
63. Set the display name to httpStatus.



Test the application

64. Save the files to redeploy the project.
65. In Advanced REST Client, add the airline and code to make a request to <http://localhost:8081/api/flights?airline=american&code=PDX>.

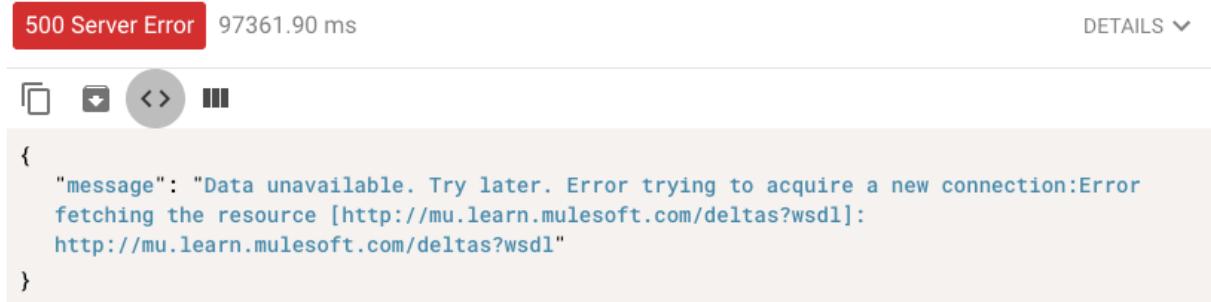
66. In the Mule Debugger, step through the application; the application now does not return to the APIkit router.
67. Return to Advanced REST Client; you should now get a 200 response with the No flights to PDX message.



200 OK 13198.97 ms

{
 "message": "No flights to PDX"
}

68. In Advanced REST Client, change the airline to make a request to <http://localhost:8081/api/flights?airline=delta&code=PDX>.
69. In the Mule Debugger, step through the application.
70. Return to Advanced REST Client; you should now get the 500 response with the data unavailable message.



500 Server Error 97361.90 ms DETAILS ▾

{
 "message": "Data unavailable. Try later. Error trying to acquire a new connection:Error
 fetching the resource [http://mu.learn.mulesoft.com/deltas?wsdl]:
 http://mu.learn.mulesoft.com/deltas?wsdl"
}

71. Return to Anypoint Studio and switch perspectives.

Walkthrough 10-8: Set a reconnection strategy for a connector

In this walkthrough, you will:

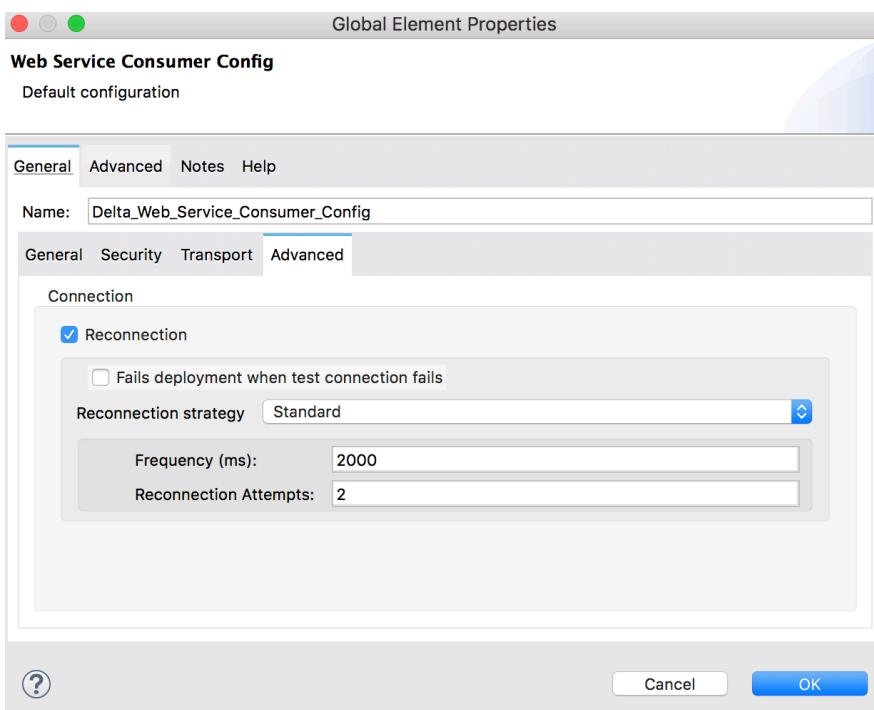
- Set a reconnection strategy for the Web Service Consumer connector.

Set a reconnection strategy for a connector

1. Return to global.xml in Anypoint Studio.
2. Switch to the Global Elements view.
3. Double-click the Web Service Consumer Config.

Global Configuration Elements		
Type	Name	Description
Configuration properties (Configuration)	Configuration properties	Create
HTTP Listener config (Configuration)	HTTP_Listener_config	Edit
American Flights API Config (Configuration)	American_Flights_API_Config	Delete
HTTP Request configuration (Configuration)	HTTP_Request_config_training	
Web Service Consumer Config (Configuration)	Delta_Web_Service_Consumer_Config	
Configuration (Configuration)	Configuration	

4. In the Global Element Properties dialog box, select the Advanced tab in the second tab bar.
5. Select the Reconnection checkbox.
6. Change the reconnection strategy to standard.
7. Review the default frequency and attempts.



8. Click OK.
9. Save the file.

Module 11: Writing DataWeave Transformations

The screenshot shows the Mule Studio interface with a DataWeave script on the left and its resulting JSON output on the right.

DataWeave Script (Left):

```
1@%dw 2.0
2 output application/dw
3 import dasherize from dw::core::Strings
4 type Currency = String {format: "##.00"}
5 type Flight = Object {class: "com.mulesoft.training.Flight"}
6
7 //var numSeats = 400
8 //var numSeats = (x=400) -> x
9/* var numSeats = (planeType: String) ->
10   if (planeType contains('737'))
11     150
12   else
13     300
14 */
15 fun getNumSeats(planeType: String) =
16   if (planeType contains('737'))
17     150
18   else
19     300
20 ---
21 using (flights =
22 payload..return map (object,index) -> {
23   destination: object.destination,
24   price: object.price as Number as Currency,
25   totalSeats: getNumSeats(object.planeType as String),
26   // totalSeats: lookup("getTotalSeats", {planeType: object.planeType}),
27   planeType: dasherize(replace(object.planeType, /(Boeing)/ with "Boeing")),
28   departureDate: object.departureDate as Date {format: "yyyy/MM/dd"}
29   as String {format: "MM dd, yyyy"},
30   availableSeats: object.emptySeats as Number
31 } as Object
32 )
33
34 flights distinctBy $
35   filter ($.availableSeats !=0)
36   orderBy $.departureDate
37   orderBy $.price
```

Output (Right):

```
[{"destination": "LAX", "price": "199.99" as Currency {format: "##.00"}, "totalSeats": 150, "planeType": "boeing-737", "departureDate": "Oct 21, 2015" as String {format: "MMM dd, yyyy"}, "availableSeats": 10}, {"destination": "PDX", "price": "283.00" as Currency {format: "##.00"}, "totalSeats": 300, "planeType": "boeing-777", "departureDate": "Oct 20, 2015" as String {format: "MMM dd, yyyy"}, "availableSeats": 23}, {"destination": "PDX", "price": "283.00" as Currency {format: "##.00"}, "totalSeats": 300, "planeType": "boeing-777", "departureDate": "Oct 21, 2015" as String {format: "MMM dd, yyyy"}, "availableSeats": 30}, {"destination": "SFO", "price": "400.00" as Currency {format: "##.00"}, "totalSeats": 150, "planeType": "boeing-737", "departureDate": "Oct 20, 2015" as String {format: "MMM dd, yyyy"}, "availableSeats": 40}]
```

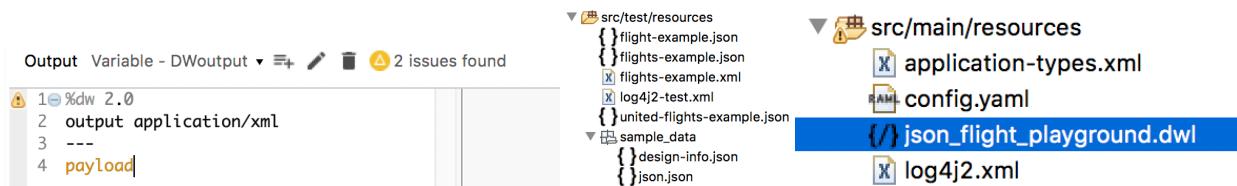
At the end of this module, you should be able to:

- Write DataWeave expressions for basic XML, JSON, and Java transformations.
- Write DataWeave transformations for complex data structures with repeated elements.
- Define and use global and local variables and functions.
- Use DataWeave functions.
- Coerce and format strings, numbers, and dates.
- Define and use custom data types.
- Call Mule flows from DataWeave expressions.
- Store DataWeave scripts in external files.

Walkthrough 11-1: Create transformations with the Transform Message component

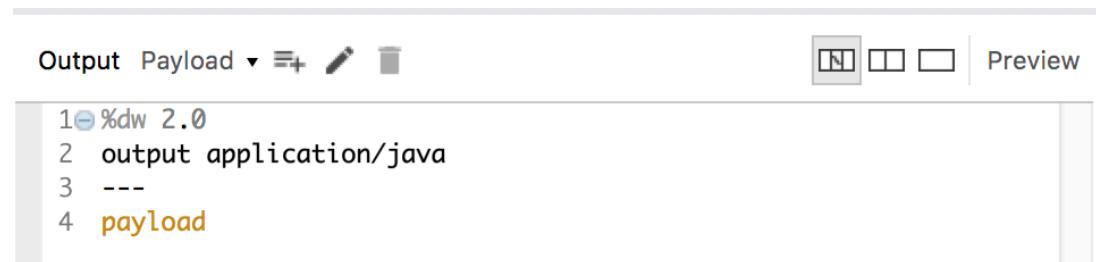
In this walkthrough, you create a new flow that receives flight POST requests that you will use as the input for writing transformations in the next several walkthroughs. You will:

- Create a new flow that receives POST requests of JSON flight objects.
- Add sample data and use live preview.
- Create a second transformation that stores the output in a variable.
- Save a DataWeave script in an external file.
- Review DataWeave script errors.



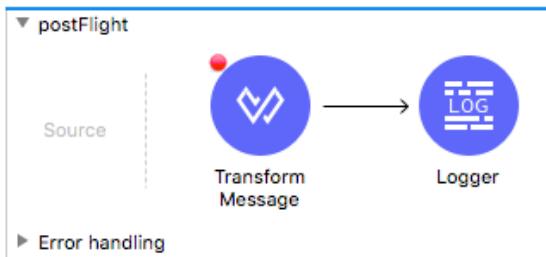
Create a new flow that receives POST requests and returns the payload as Java

1. In the apdev-flights-ws Mule project, return to config.yaml.
2. Change the delta.wsdl property from /deltas?wsdl to /delta?wsdl.
3. Return to implementation.xml.
4. Right-click in the canvas and select Collapse All.
5. Drag a Transform Message component from the Mule Palette to the bottom of the canvas; a new flow should be created.
6. Change the name of the flow to postFlight.
7. In the Transform Message properties view, set the expression to payload.



8. Add a breakpoint to the Transform Message component.

9. Add a Logger to the flow.

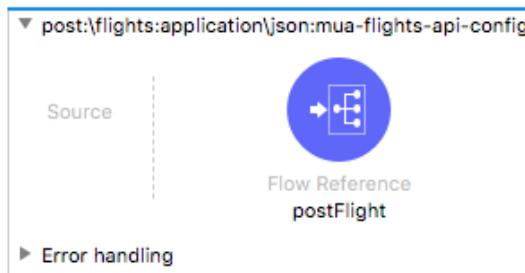


10. Return to interface.xml.

11. Locate the post:\flights:application\json:mua-flights-api-config flow.

12. Delete the Transform Message component and replace it with a Flow Reference.

13. In the Flow Reference properties view, set the flow name and display name to postFlight.



14. Save the files to redeploy the project in debug mode.

15. Close interface.xml.

Post a flight to the flow

16. Open flight-example.json in src/test/resources.

17. Copy the code and close the file.

18. In Advanced REST Client, change the method to POST and remove any query parameters.

19. Add a request header called Content-Type and set it equal to application/json.

20. Set the request body to the value you copied from flight-example.json.

21. Make the request to <http://localhost:8081/api/flights>.

The screenshot shows a REST client interface with the following details:

- Method: POST
- Request URL: <http://localhost:8081/api/flights>
- Headers: Body content type is set to application/json.
- Body: The content type is application/json, and the payload is a JSON object representing a flight:

```
{
  "airline": "United",
  "flightCode": "ER38sd",
  "fromAirportCode": "LAX",
  "toAirportCode": "SFO",
  "departureDate": "May 21, 2016",
  "emptySeats": 0,
  "totalSeats": 200,
  "price": 199,
  "planeType": "Boeing 737"
}
```
- Authorization: None
- Actions: SEND button

22. In the Mule Debugger, expand Attributes and locate the content-type header.

23. Review the payload; you should see it has a mime-type of application/json.

The screenshot shows the Mule Debugger with the following details:

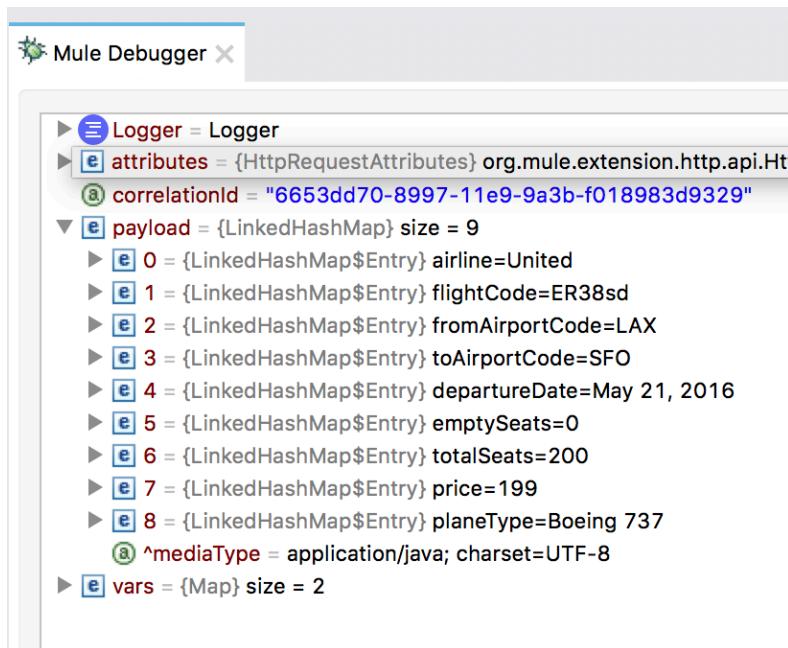
- Transform = Transform Message
- attributes = {HttpRequestAttributes} org.mule.extension.http.api.HttpRequestAttribute
- correlationId = "6653dd70-8997-11e9-9a3b-f018983d9329"
- payload = {** (highlighted)

 - mediaType = application/json; charset=UTF-8
 - vars = {Map} size = 2
 - httpStatus = "201"
 - outboundHeaders = {HashMap} size = 0

- }** (highlighted)
- { (highlighted)

```
{
  "airline": "United",
  "flightCode": "ER38sd",
  "fromAirportCode": "LAX",
  "toAirportCode": "SFO",
  "departureDate": "May 21, 2016",
  "emptySeats": 0,
  "totalSeats": 200,
  "price": 199,
  "planeType": "Boeing 737"
}
```

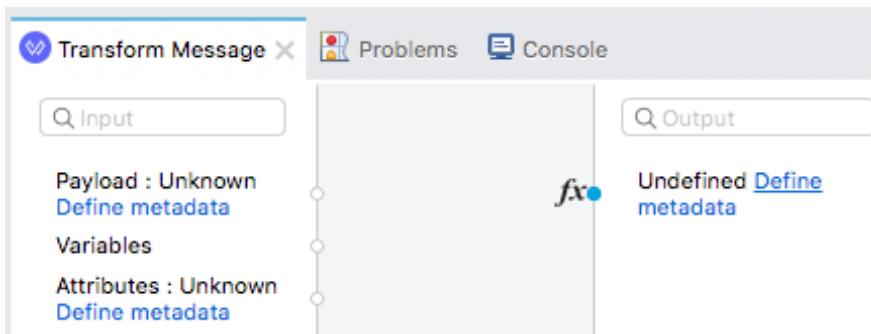
24. Step to the Logger; you should see the payload now has a mime-type of application/java and is a LinkedHashMap.



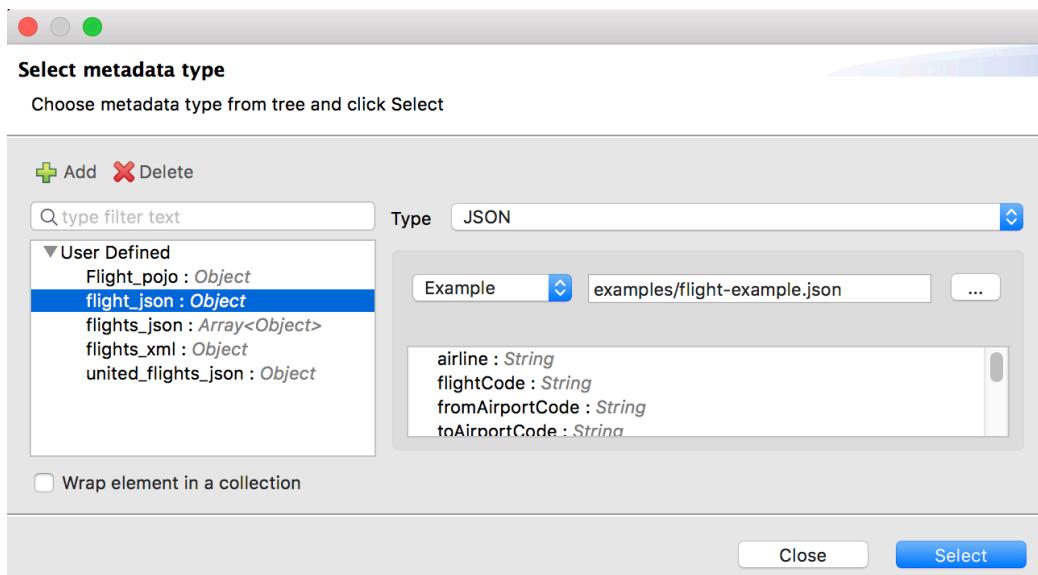
25. Step to the end of the application and switch perspectives.

Add input metadata for the transformation

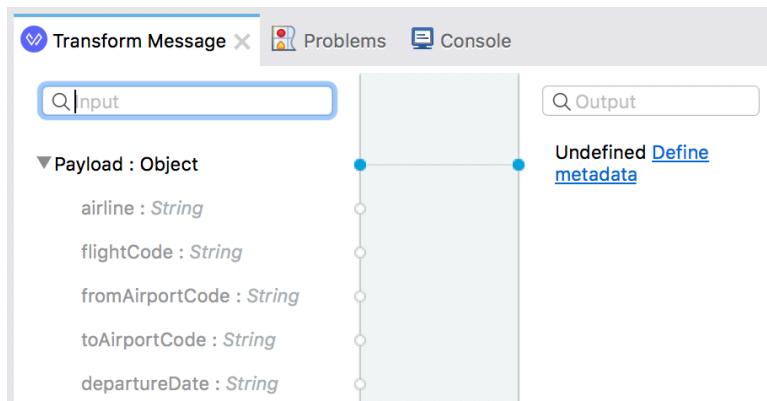
26. In the input section of the Transform Message properties view, click Define metadata for the payload.



27. In the Select metadata type dialog box, select flight_json and click Select.



28. In the Transform Message Properties view, you should now see metadata for the input.



29. Open application-types.xml in src/main/resources.

30. Review the enrichment elements for flight_json.

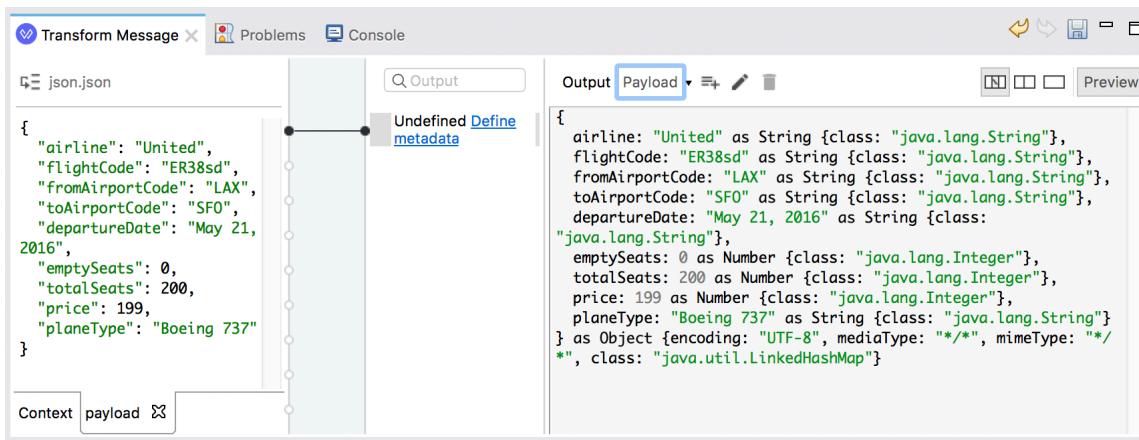
```
77      </types:output-event>
78      </types:processor-declaration>
79  </types:enrichment>
80  <types:enrichment select="#5cfedadd-58a4-486b-97ff-333994ab8cea">
81    <types:processor-declaration>
82      <types:input-event>
83        <types:message>
84          <types:payload type="flight_json"/>
85        </types:message>
86      </types:input-event>
87    </types:processor-declaration>
88  </types:enrichment>
89 </types:mule>
```

31. Close the file.

Preview sample data and sample output

32. In the input section of the Transform Message properties view, right-click the payload and select Edit Sample Data; you should see a new tab called payload and it should contain the sample input of type JSON.
33. Click the Preview button; you should see a new preview section and it should contain the sample output of type Java.

If sample data doesn't automatically appear in the preview section, click the Create required sample data to execute preview link.

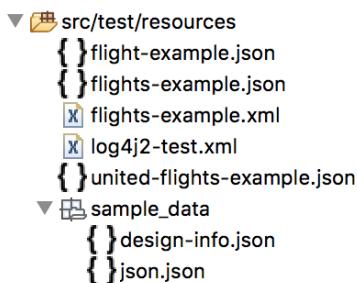


34. Click the Source Only button to the left of the Preview button.



Locate the new sample data file

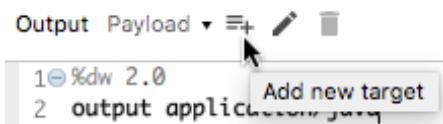
35. In the project explorer, locate the new sample_data folder in src/test/resources.



36. Open json.json.
37. Review the code and close the file.

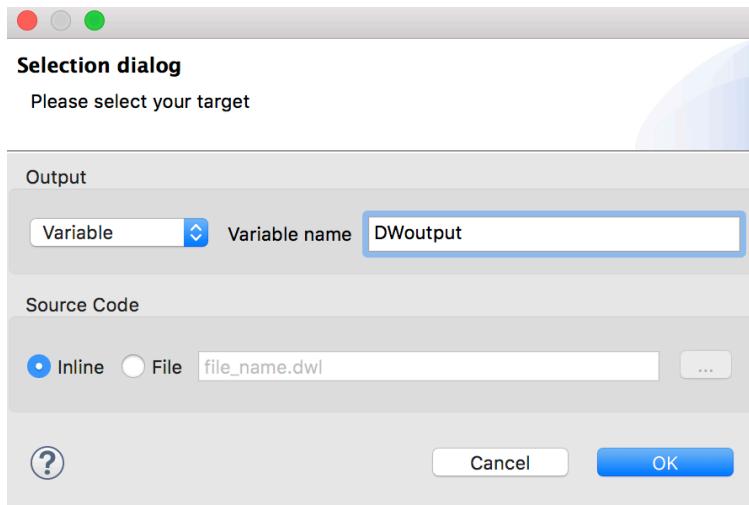
Create a second transformation with the same component

38. In the Transform Message properties view, click the Add new target button.



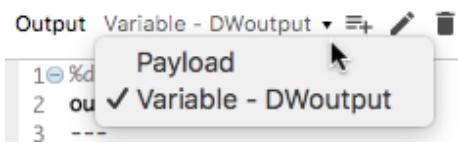
39. In the Selection dialog box, change the output to variable.

40. Set the variable name to DWoutput.



41. Click OK.

42. Click the drop-down menu for the output; you should see that you can switch between the two transformations.



43. For the new transformation, set the output type to json and set the expression to payload.

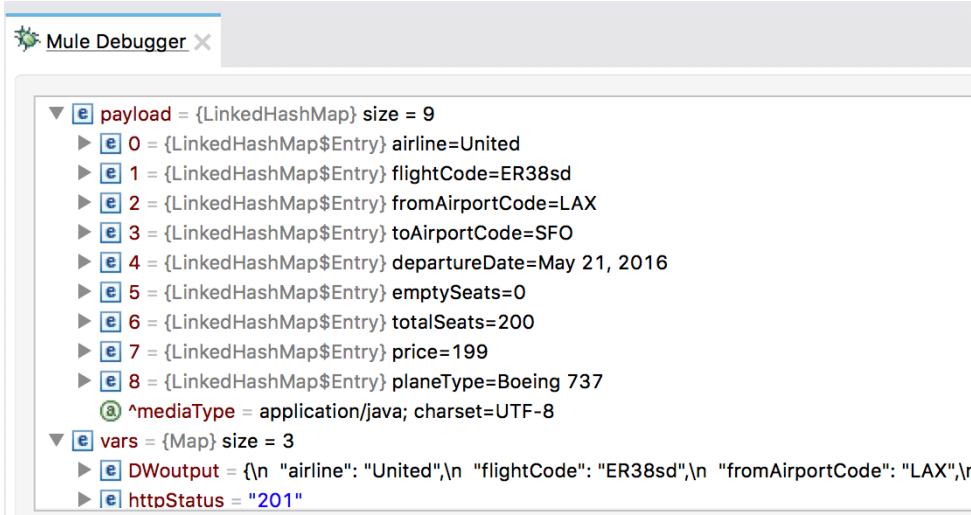


Test the application

44. Save the file to redeploy the project in debug mode.

45. In Advanced REST Client, post the same request to <http://localhost:8081/api/flights>.

46. In the Mule Debugger, step to the Logger.
47. Review Payload; it should be Java as before.
48. Expand Variables; you should see the new DWoutput variable.



The screenshot shows the Mule Debugger interface with the title bar "Mule Debugger". Below the title bar, there is a tree view of variables:

- payload** = {LinkedHashMap} size = 9
 - 0 = {LinkedHashMap\$Entry} airline=United
 - 1 = {LinkedHashMap\$Entry} flightCode=ER38sd
 - 2 = {LinkedHashMap\$Entry} fromAirportCode=LAX
 - 3 = {LinkedHashMap\$Entry} toAirportCode=SFO
 - 4 = {LinkedHashMap\$Entry} departureDate=May 21, 2016
 - 5 = {LinkedHashMap\$Entry} emptySeats=0
 - 6 = {LinkedHashMap\$Entry} totalSeats=200
 - 7 = {LinkedHashMap\$Entry} price=199
 - 8 = {LinkedHashMap\$Entry} planeType=Boeing 737
 - mediaType = application/java; charset=UTF-8
- vars** = {Map} size = 3
 - DWoutput = {\n "airline": "United",\n "flightCode": "ER38sd",\n "fromAirportCode": "LAX",\n "toAirportCode": "SFO",\n "departureDate": "May 21, 2016",\n "emptySeats": 0,\n "totalSeats": 200,\n "price": 199,\n "planeType": "Boeing 737"}
HTTP Status = "201"

49. Step through the rest of the application and switch perspectives.

Review the Transform Message XML code

50. Right-click the Transform Message component and select Go to XML.
51. Locate and review the code for both DataWeave transformations.

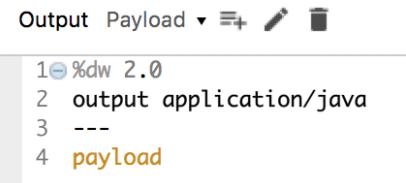
```
<flow doc:id="a2d732ff-aaec-448e-9bdd-a25319cc55a2" name="postFlight">
    <ee:transform doc:name="Transform Message" doc:id="ccda7d44-19b9">
        <ee:message>
            <ee:set-payload><![CDATA[%dw 2.0
output application/java
---
payload]]></ee:set-payload>
        </ee:message>
        <ee:variables>
            <ee:set-variable variableName="DWoutput" ><![CDATA[%dw 2.0
output application/json
---
payload]]></ee:set-variable>
            </ee:variables>
        </ee:transform>
        <logger level="INFO" doc:name="Logger" doc:id="6b956315-38e9-421
</flow>
```

52. Switch back to the Message Flow view.

Save a DataWeave script to an external file

53. In the Transform Message Properties view, make sure the payload output expression is selected.

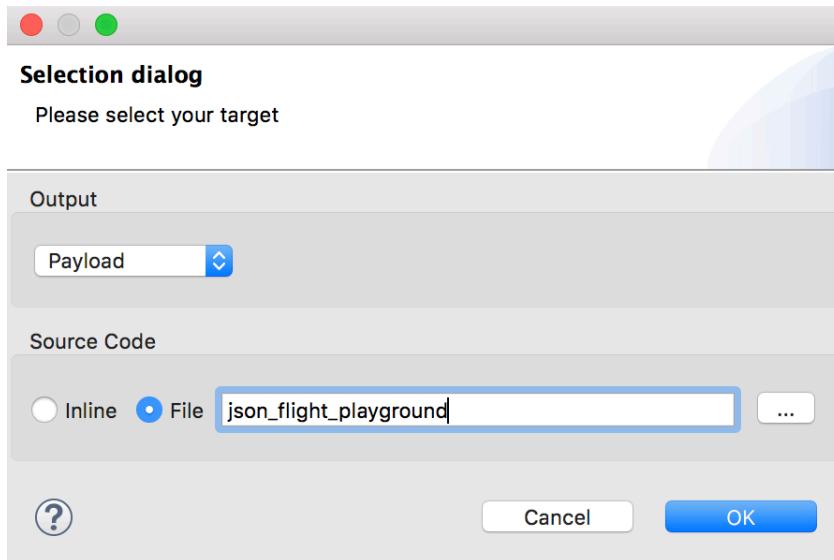
54. Click the Edit current target button (the pencil) above the code.



```
Output Payload ▾  
1 %dw 2.0
2   output application/java
3   ---
4   payload
```

55. In the Selection dialog box, change the source code selection from inline to file.

56. Set the file name to json_flight_playground.



57. Click OK.

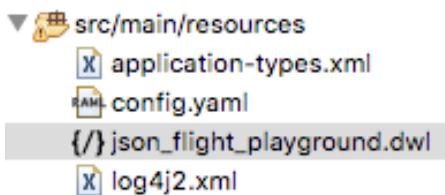
Locate and review the code that uses an external DataWeave script

58. Switch to Configuration XML view.

59. Locate the new code for the transformation in postFlight.

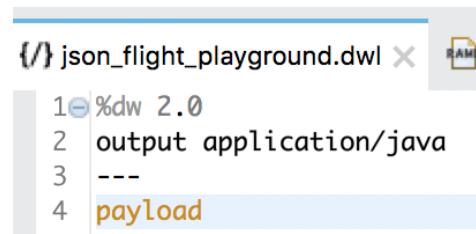
```
<flow doc:id="a2d732ff-aaec-448e-9bdd-a25319cc55a2" name="postFlight">
    <ee:transform doc:name="Transform Message" doc:id="ccda7d44-19b9-4245-a6
        <ee:message>
            <ee:set-payload resource="json_flight_playground.dwl" />
        </ee:message>
        <ee:variables>
            <ee:set-variable variableName="DWoutput" ><!CDATA[%dw 2.0
output application/json
---
payload]]></ee:set-variable>
        </ee:variables>
    </ee:transform>
    <logger level="INFO" doc:name="Logger" doc:id="6b956315-38e9-4216-96c0-a
    </flow>
```

60. In the Package Explorer, expand src/main/resources.



61. Open json_flight_playground.dwl.

62. Review and then close the file.



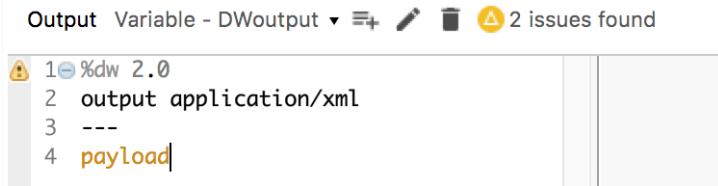
63. Return to Message Flow view in implementation.xml.

Change the output type to XML and review errors

64. In the Transform Message properties view, select the Variable – DWoutput output.

65. Change the output type from application/json to application/xml.

66. Locate the warning icons indicating that there is a problem.



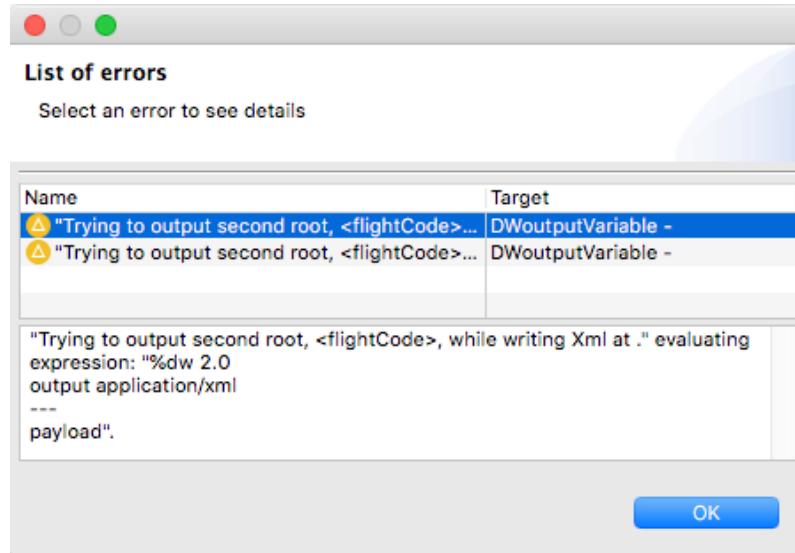
The screenshot shows the 'Output Variable - DWoutput' dialog. At the top, it says '2 issues found'. Below that, there are four numbered lines of code:

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

The first two lines have yellow warning icons (triangle with exclamation) to their left.

Note: If you do not see the warnings, click the Preview button. If you do not see a data preview, select the payload tab in the input section. If you don't see the payload tab, right-click payload in the input section and select Edit Sample Data.

67. Mouse over the icon located to the left of the code; you should see a message that there was an exception trying to output the second root <flightCode>.
68. Click the icon above the code; a List of errors dialog box should open.
69. Select and review the first issue.



70. In the List of errors dialog box, click OK.

Note: You will learn how to successfully transform to XML in the next walkthrough.

Test the application

71. Save the file to redeploy the project in debug mode.
72. In Advanced REST Client, post the same request to <http://localhost:8081/api/flights>.
73. In the Mule Debugger, step once; you should get an error.

74. Expand the error and review the error information; you should see there is a DataWeave error when trying to output the second root.

The screenshot shows the Mule Debugger interface with an expanded error stack trace. The error details indicate a `javax.xml.stream.XMLStreamException` related to trying to output a second root while writing XML. Below the debugger, the Mule flow editor displays a flow starting from a Source component, followed by a Transform Message component (which is highlighted with a red dashed box), and finally a Logger component.

```
▶ [e] attributes = {HttpRequestAttributes} org.mule.extension.http.api.HttpRequestAttributes
  @ correlationId = "2afd4f51-8a0e-11e9-955c-f018983d9329"
  ▼ [e] error = {ErrorImplementation} \org.mule.runtime.core.internal.message.ErrorBuilder$ErrorImplementation
    @ description = "[javaj.xml.stream.XMLStreamException - Trying to output second root, <flightCode>, while writing Xml at payload.]" evaluating expression
    @ detailedDescription = "[javaj.xml.stream.XMLStreamException - Trying to output second root, <flightCode>, while writing Xml at payload.]" evaluating expression
    ▶ [e] errorType = {ErrorTypeImplementation} MULE:EXPRESSION
    @ errors = {UnmodifiableRandomAccessList} size = 0
    ▼ [e] exception = {ExpressionRuntimeException} org.mule.runtime.core.api.expression.ExpressionRuntimeException: "javaj.xml.stream.XMLStreamException"
      @ CAUSE_CAPTION = "Caused by: "
      @ EMPTY_THROWABLE_ARRAY = {Throwable[]} size = 0
      @ NULL_CAUSE_MESSAGE = "Cannot suppress a null exception."
      @ SELF_SUPPRESSION_MESSAGE = "Self-suppression not permitted"
      @ SUPPRESSED_CAPTION = "Suppressed: "
      @ SUPPRESSED_SENTINEL = {UnmodifiableRandomAccessList} size = 0
```

mua-flights-api.raml *implementation global config.yaml FlightsExample.raml interface

Source → Transform Message → Logger

75. Step to the end of the application and switch perspectives.

Walkthrough 11-2: Transform basic JSON, Java, and XML data structures

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

- Write scripts to transform the JSON payload to various JSON and Java structures.
- Write scripts to transform the JSON payload to various XML structures.

The screenshot shows a DataWeave (DW) transformation script on the left and its XML preview on the right. The script is as follows:

```
1 %dw 2.0
2 output application/xml
3 ---
4 data: {
5   hub: "MUA",
6   flight @{airline: payload.airline}: {
7     code: payload.toAirportCode,
8   }
9 }
```

The XML preview shows the resulting structure:

```
<?xml version='1.0' encoding='UTF-8'?>
<data>
  <hub>MUA</hub>
  <flight airline="United">
    <code>SFO</code>
  </flight>
</data>
```

Write expressions to transform JSON to various Java structures

1. Return to the Transform Message properties view for the transformation in postFlight.
2. In the output drop-down menu, make sure the payload output expression is selected.
3. Change the output type from java to json.
4. Type a period after payload and double-click price in the auto-completion menu.

The screenshot shows a DataWeave script with a tooltip for the 'payload' variable. The tooltip lists several properties:

- f airline : String
- f flightCode : String
- f fromAirportCode : String
- f toAirportCode : String
- f departureDate : String
- f emptySeats : Number {"typeId": "int"}
- f totalSeats : Number {"typeId": "int"}
- f price : Number {"typeId": "int"}** (highlighted in blue)
- f planeType : String

5. Look at the preview.

The screenshot shows the DataWeave script and its preview. The preview displays the value '199' corresponding to the 'price' field.

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

6. Change the output type from json to java.

Output Payload ▾  

1 %dw 2.0	199 as Number {encoding: "UTF-8", mediaType: "*/*", mimeType: "*/*", class: "java.lang.Integer"}
2 output application/java	
3 ---	
4 payload.price	

Preview    Preview

7. Change the DataWeave expression to data: payload.price.

Output Payload ▾  

1 %dw 2.0	{ data: 199 as Number {class: "java.lang.Integer" } as Object {encoding: "UTF-8", mediaType: "*/*", mimeType: "*/*", class: "java.util.LinkedHashMap"}
2 output application/java	
3 ---	
4 data: payload.price	

Preview    Preview

8. Change the output type from java to json.

Output Payload ▾  

1 %dw 2.0	{ "data": 199 }
2 output application/json	
3 ---	
4 data: payload.price	

Preview    Preview

9. Change the DataWeave expression to data: payload.

Output Payload ▾  

1 %dw 2.0	{ "data": { "airline": "United", "flightCode": "ER38sd", "fromAirportCode": "LAX", "toAirportCode": "SFO", "departureDate": "May 21, 2016", "emptySeats": 0, "totalSeats": 200, "price": 199, "planeType": "Boeing 737" } }
2 output application/json	
3 ---	
4 data: payload	

Preview    Preview

10. Change the output type from json to java.

The screenshot shows the DataWeave editor interface with the following code and preview:

```
1 %dw 2.0
2 output application/java
3 ---
4 data: payload
```

Preview pane content:

```
{  
    data: {  
        airline: "United" as String {class: "java.lang.String"},  
        flightCode: "ER38sd" as String {class: "java.lang.String"},  
        fromAirportCode: "LAX" as String {class: "java.lang.String"},  
        toAirportCode: "SFO" as String {class: "java.lang.String"},  
        departureDate: "May 21, 2016" as String {class:  
            "java.lang.String"},  
        emptySeats: 0 as Number {class: "java.lang.Integer"},  
        totalSeats: 200 as Number {class: "java.lang.Integer"},  
        price: 199 as Number {class: "java.lang.Integer"},  
        planeType: "Boeing 737" as String {class: "java.lang.String"}  
    } as Object {class: "java.util.LinkedHashMap"}  
} as Object {encoding: "UTF-8", mediaType: "*/*", mimeType: "*/*",  
class: "java.util.LinkedHashMap"}
```

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

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

The screenshot shows the DataWeave editor interface with the following code and preview:

```
1 %dw 2.0
2 output application/java
3 ---
4 data: {  
5     hub: "MUA"  
6 }
```

Preview pane content:

```
{  
    data: {  
        hub: "MUA" as String {class: "java.lang.String"}  
    } as Object {class: "java.util.LinkedHashMap"}  
} as Object {encoding: "UTF-8", mediaType: "*/*",  
mimeType: "*/*", class: "java.util.LinkedHashMap"}
```

13. Change the output type from java to json.

The screenshot shows the DataWeave editor interface with the following code and preview:

```
1 %dw 2.0
2 output application/json
3 ---
4 data: {  
5     hub: "MUA"  
6 }
```

Preview pane content:

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

14. Add a field called code and use auto-completion to set it to the toAirportCode property of the payload.

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

```
%dw 2.0
output application/json
---
data: {
    hub: "MUA",
    code: payload.toAirportCode,
    airline: payload.airline
}
```

Write an expression to output data as XML

16. In the output drop-down menu, select Variable – DW output.
17. Change the DataWeave expression to data: payload.
18. Look at the preview; you should now see XML.

```
%dw 2.0
output application/xml
---
data: payload
```

19. In the output drop-down menu, select Payload.
20. Copy the DataWeave expression.
21. Switch back to Variable – DWoutput and replace the DataWeave expression with the one you just copied.

```
%dw 2.0
output application/xml
---
data: {
    hub: "MUA",
    code: payload.toAirportCode,
    airline: payload.airline
}
```

22. Modify the expression so the code and airline properties are child elements of a new element called flight.

Output Variable - DWoutput ▾ Preview

```
1 %dw 2.0
2   output application/xml
3   ---
4   data: {
5     hub: "MUA",
6     flight: {
7       code: payload.toAirportCode,
8       airline: payload.airline
9     }
10 }
```

```
<?xml version='1.0' encoding='UTF-8'?>
<data>
  <hub>MUA</hub>
  <flight>
    <code>SF0</code>
    <airline>United</airline>
  </flight>
</data>
```

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

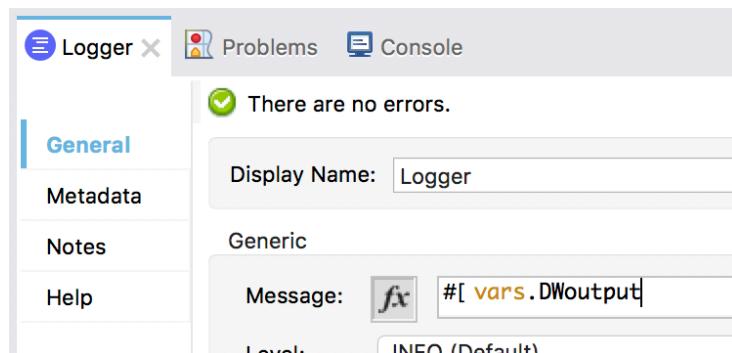
Output Variable - DWoutput ▾ Preview

```
1 %dw 2.0
2   output application/xml
3   ---
4   data: {
5     hub: "MUA",
6     flight @airline: payload.airline: {
7       code: payload.toAirportCode,
8     }
9 }
```

```
<?xml version='1.0' encoding='UTF-8'?>
<data>
  <hub>MUA</hub>
  <flight airline="United">
    <code>SF0</code>
  </flight>
</data>
```

Debug the application

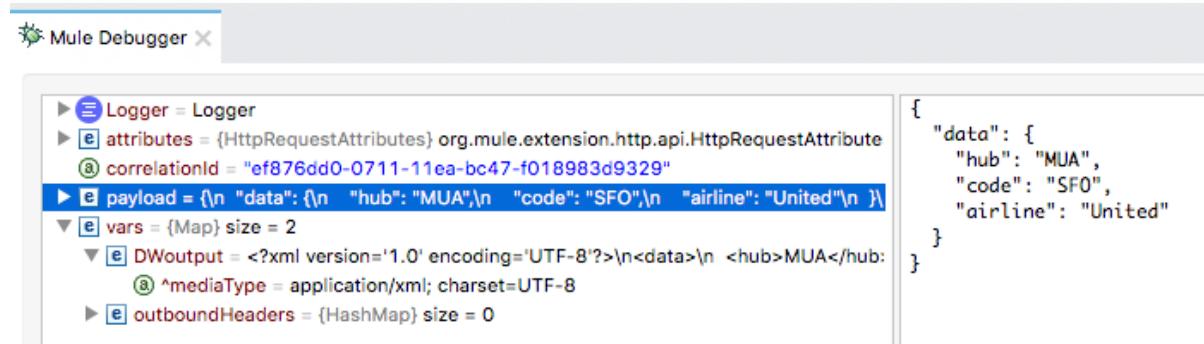
24. In the Logger properties view, set the message to the value of the DWoutput variable.



25. Save the file to redeploy the project in debug mode.

26. In Advanced REST Client, post the same request to <http://localhost:8081/api/flights>.

27. In the Mule Debugger, step to the Logger; you should see the transformed JSON payload and that the DWoutput variable has a mime-type of application/xml.



The screenshot shows the Mule Debugger interface with the title bar "Mule Debugger". The left pane displays a tree view of variables:

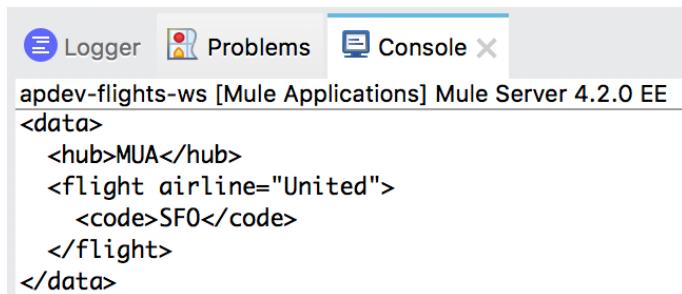
- Logger = Logger
- attributes = {HttpRequestAttributes} org.mule.extension.http.api.HttpRequestAttribute
- correlationId = "ef876dd0-0711-11ea-bc47-f018983d9329"
- payload = {
 "data": {
 "hub": "MUA",
 "code": "SFO",
 "airline": "United"
 }
}**
- vars = {Map} size = 2
 - DWoutput = <?xml version='1.0' encoding='UTF-8'?>|n<data>|n <hub>MUA</hub>
 - ^mediaType = application/xml; charset=UTF-8
- outboundHeaders = {HashMap} size = 0

The right pane shows the expanded XML content of the payload variable:

```
{  
  "data": {  
    "hub": "MUA",  
    "code": "SFO",  
    "airline": "United"  
  }  
}
```

28. Step through the application and switch perspectives.

29. Look at the Logger output in the console; you should see the XML.



The screenshot shows the Mule Application Console interface with the title bar "apdev-flights-ws [Mule Applications] Mule Server 4.2.0 EE". The tabs at the top are "Logger", "Problems", and "Console". The "Console" tab is selected. The output window displays the XML payload:

```
<data>  
  <hub>MUA</hub>  
  <flight airline="United">  
    <code>SF0</code>  
  </flight>  
</data>
```

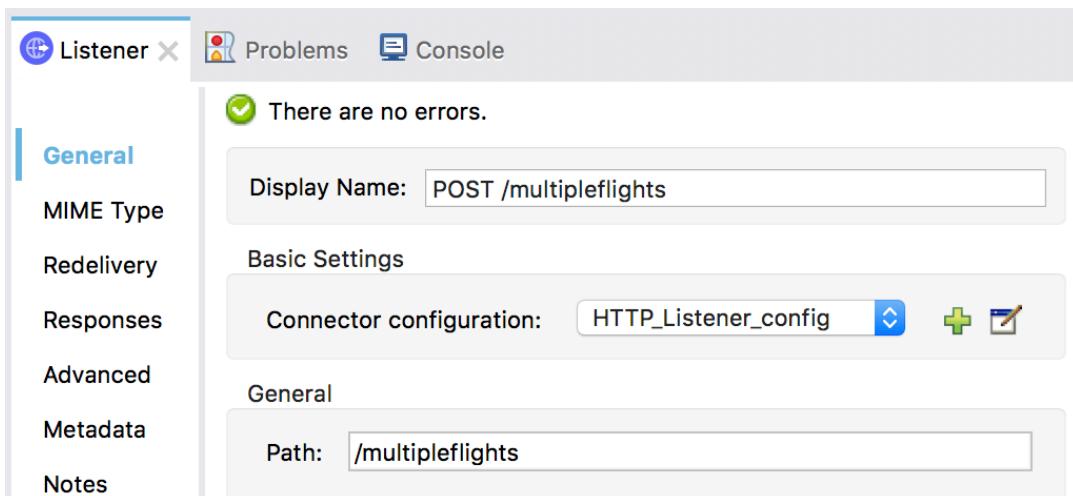
Walkthrough 11-3: Transform complex data structures with arrays

In this walkthrough, you create a new flow that allows multiple flights to be posted to it, so you have more complex data with which to work. You will:

- Create a new flow that receives POST requests of a JSON array of flight objects.
- Transform a JSON array of objects to DataWeave, JSON, and Java.

Create a new flow that receives POST requests of a JSON array of flight objects

1. Return to implementation.xml.
2. Drag out an HTTP Listener from the Mule Palette to the bottom of the canvas.
3. Change the flow name to postMultipleFlights.
4. In the HTTP Listener properties view, set the display name to POST /multipleflights.
5. Ensure the connector configuration is set to the existing HTTP_Listener_config.
6. Set the path to /multipleflights and set the allowed methods to POST.



Note: You are bypassing the APIkit router because a corresponding resource/method pair is not defined in the API.

- Add a Transform Message component to the flow.
- In the Transform Message properties view, set the expression to payload.

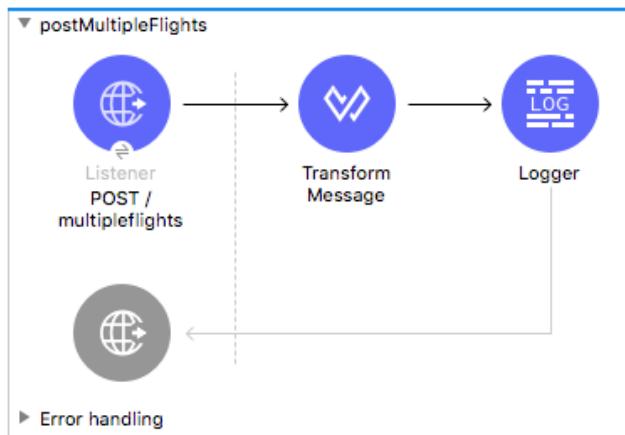
Output Payload ▾ Preview

```

1 @%dw 2.0
2   output application/java
3   ---
4   payload

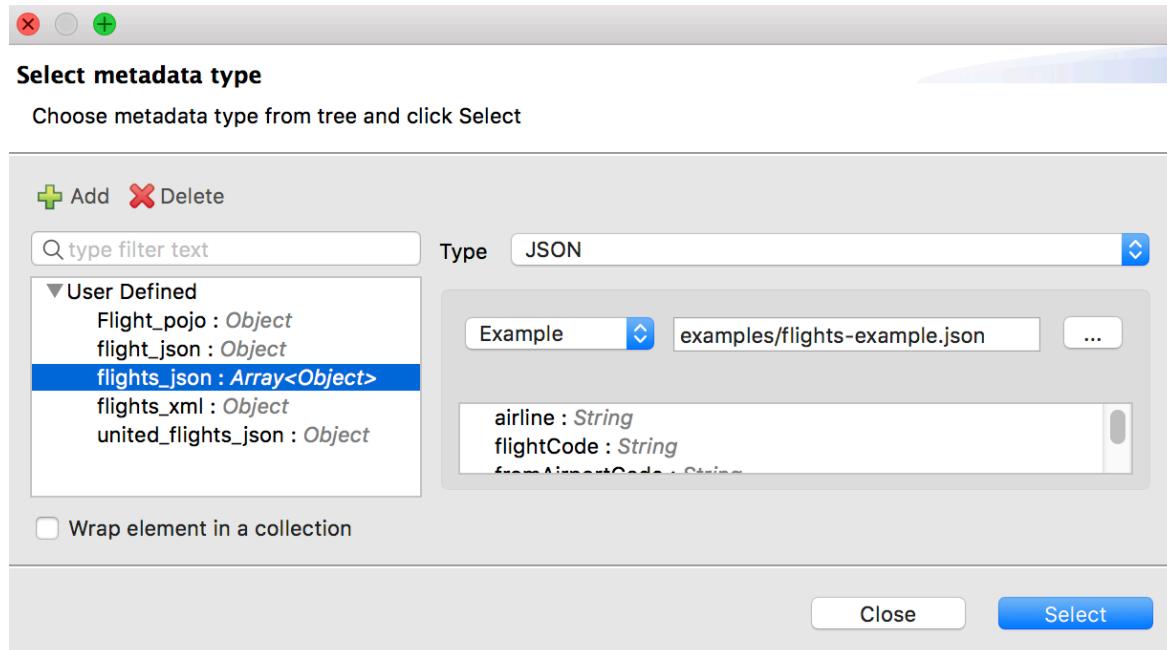
```

- Add a Logger to the flow.

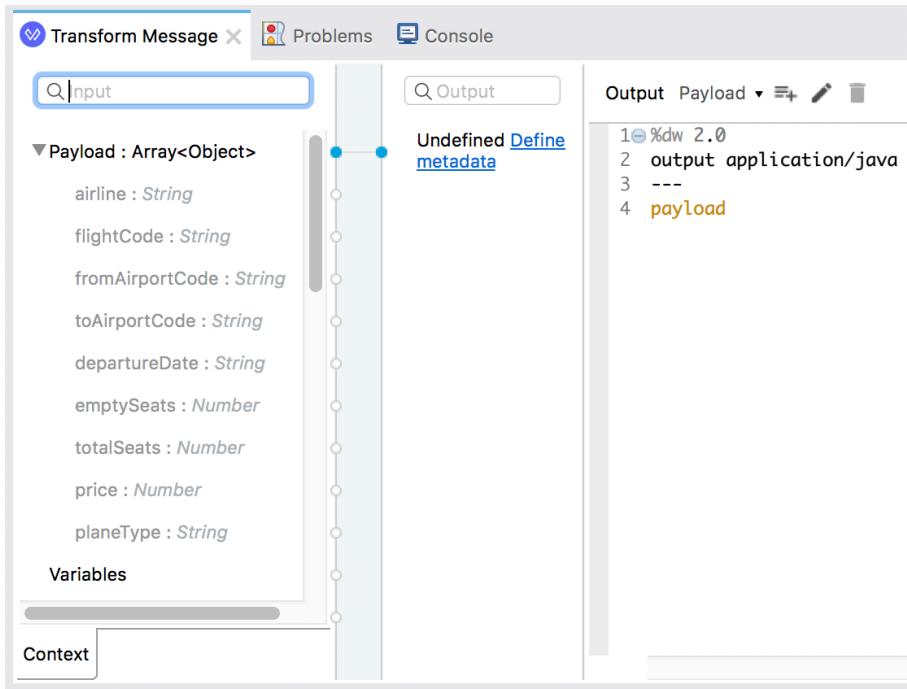


Add input metadata for the transformation

- In the Transform Message properties view, click Define metadata in the input section.
- In the Select metadata type dialog box, select flights_json and click Select.



12. In the Transform Message Properties view, you should now see metadata for the input.

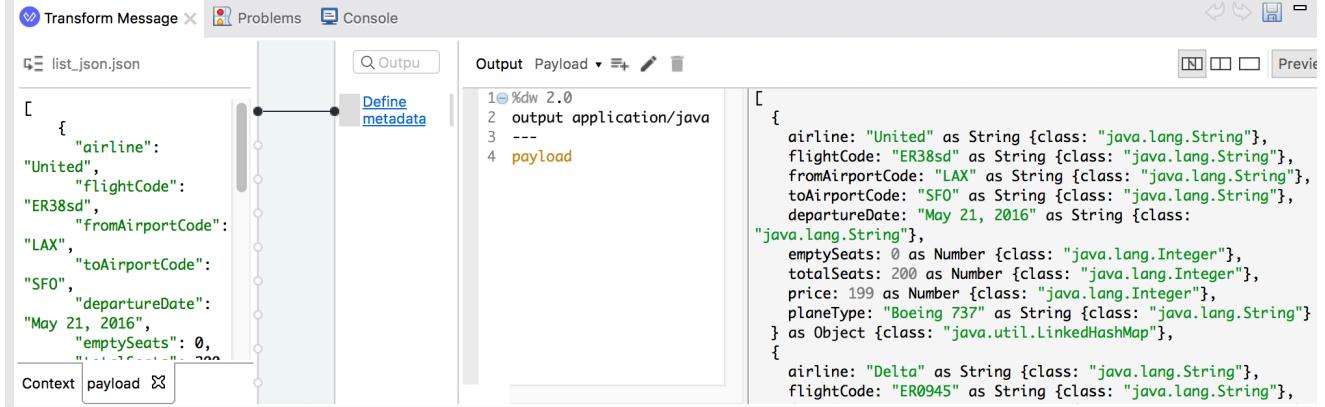


Preview sample data and sample output

13. In the input section, right-click the payload and select Edit Sample Data to get sample input data.
 14. Click the Preview button to get sample output data.

If sample data doesn't automatically appear in the preview section, click the Create required sample data to execute preview link.

15. Look at the preview section; the sample output should be an ArrayList of LinkedHashMaps.



16. Change the output type to application/dw.

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

```
[{"airline": "United", "flightCode": "ER38sd", "fromAirportCode": "LAX", "toAirportCode": "SFO", "departureDate": "May 21, 2016", "emptySeats": 0, "totalSeats": 200, "price": 199, "planeType": "Boeing 737"}, {"airline": "Delta", "flightCode": "ER0945", "fromAirportCode": "PDX", "toAirportCode": "ATL", "departureDate": "May 21, 2016", "emptySeats": 0, "totalSeats": 200, "price": 229, "planeType": "Boeing 737"}]
```

Return values for the first object in the collection

17. Change the DataWeave expression to payload[0].

```
1 %dw 2.0
2 output application/dw
3 ---
4 payload[0]
```

```
{ "airline": "United", "flightCode": "ER38sd", "fromAirportCode": "LAX", "toAirportCode": "SFO", "departureDate": "May 21, 2016", "emptySeats": 0, "totalSeats": 200, "price": 199, "planeType": "Boeing 737"}
```

18. Change the DataWeave expression to payload[0].price.

```
1 %dw 2.0
2 output application/dw
3 ---
4 payload[0].price
```

```
199
```

19. Change the DataWeave expression to payload[0].*price.

```
1 %dw 2.0
2 output application/dw
3 ---
4 payload[0].*price
```

```
[199]
```

Return collections of values

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

Output Payload ▾  

```
1 %dw 2.0
2 output application/dw
3 ---
4 payload.*price
```

[
199,
450
]

21. Change the DataWeave expression to payload.price; you should get the same result.

Output Payload ▾  

```
1 %dw 2.0
2 output application/dw
3 ---
4 payload.price
```

[
199,
450
]

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

[payload.price, payload.emptySeats]

Output Payload ▾  

```
1 %dw 2.0
2 output application/dw
3 ---
4 [payload.price, payload.emptySeats]
```

[
[
199,
450
],
[
0,
24
]
]

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

23. Change the DataWeave expression to payload map \$.

The screenshot shows the DataWeave editor interface with the following code and output:

```
Output Payload ▾ + P
```

```
1 %dw 2.0
2 output application/dw
3 ---
4 payload map $
```

```
[{"airline": "United", "flightCode": "ER38sd", "fromAirportCode": "LAX", "toAirportCode": "SFO", "departureDate": "May 21, 2016", "emptySeats": 0, "totalSeats": 200, "price": 199, "planeType": "Boeing 737"}, {"airline": "Delta", "flightCode": "ER0945"}]
```

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

The screenshot shows the DataWeave editor interface with the following code and output:

```
Output Payload ▾ + P
```

```
1 %dw 2.0
2 output application/dw
3 ---
4 payload map {
5   flight: $$
6 }
```

```
[{"flight": 0}, {"flight": 1}]
```

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

The screenshot shows the DataWeave editor interface with the following code and output:

```
Output Payload ▾ + P
```

```
1 %dw 2.0
2 output application/dw
3 ---
4 payload map {
5   flight: $$,
6   destination: $.toAirportCode
7 }
```

```
[{"flight": 0, "destination": "SFO"}, {"flight": 1, "destination": "CLE"}]
```

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

Output Payload ▾

```
1 %dw 2.0
2 output application/dw
3 ---
4 payload map {
5   flight: $$,
6   ($$): $
7 }
```

[

```
{
  flight: 0,
  "0": {
    airline: "United",
    flightCode: "ER38sd",
    fromAirportCode: "LAX",
    toAirportCode: "SFO",
    departureDate: "May 21, 2016",
    emptySeats: 0,
    totalSeats: 200,
    price: 199,
    planeType: "Boeing 737"
  },
  {
    flight: 1,
    "1": {
      airline: "Delta",
      flightCode: "ER0945"
    }
  }
}
```

27. Remove the first flight property assignment.

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

Output Payload ▾

```
1 %dw 2.0
2 output application/dw
3 ---
4 payload map {
5   'flight$$': $
6 }
```

[

```
{
  flight0: {
    airline: "United",
    flightCode: "ER38sd",
    fromAirportCode: "LAX",
    toAirportCode: "SFO",
    departureDate: "May 21, 2016",
    emptySeats: 0,
    totalSeats: 200,
    price: 199,
    planeType: "Boeing 737"
  },
  {
    flight1: {
      airline: "Delta",
      flightCode: "ER0945",
      fromAirportCode: "PDX".
    }
  }
}
```

Use explicit named parameters with the map operator

29. Change the map operator so that it uses two parameters called object and index and set the field name to just the index value.

Output Payload ▾  

1 %dw 2.0
2 output application/dw
3 ---
4 payload map (object, index) -> {
5 (index): object
6 }

[
 {
 "0": {
 airline: "United",
 flightCode: "ER38sd",
 fromAirportCode: "LAX",
 toAirportCode: "SFO",
 departureDate: "May 21, 2016",
 emptySeats: 0,
 totalSeats: 200,
 price: 199,
 planeType: "Boeing 737"
 }
 },
 {
 "1": {
 airline: "Delta",
 flightCode: "ER0945",
 fromAirportCode: "PDX".
 }
 }

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

Output Payload ▾  

1 %dw 2.0
2 output application/dw
3 ---
4 payload map (object, index) -> {
5 'flight\$(index)': object
6 }

[
 {
 flight0: {
 airline: "United",
 flightCode: "ER38sd",
 fromAirportCode: "LAX",
 toAirportCode: "SFO",
 departureDate: "May 21, 2016",
 emptySeats: 0,
 totalSeats: 200,
 price: 199,
 planeType: "Boeing 737"
 }
 },
 {
 flight1: {
 airline: "Delta",
 flightCode: "ER0945",
 fromAirportCode: "PDX".
 }
 }

Note: If you want to test the application, change the output type to application/json and then in Advanced REST Client, make a request to <http://localhost:8081/multipleflights> and replace the request body with JSON from the flights-example.json file.

Change the expression to output different data types

31. Change the DataWeave expression output type from application/dw to application/json.

The screenshot shows the MuleSoft Anypoint Studio interface. On the left, the DataWeave editor window has the title "Output Payload" and contains the following code:

```
1 %dw 2.0
2 output application/json
3 ---
4 payload map (object, index) -> {
5     'flight$(index)': object
6 }
```

On the right, the "Preview" tab displays the resulting JSON output:

```
[{"flight0": {"airline": "United", "flightCode": "ER38sd", "fromAirportCode": "LAX", "toAirportCode": "SFO", "departureDate": "May 21, 2016", "emptySeats": 0, "totalSeats": 200, "price": 199, "planeType": "Boeing 737"}, "flight1": {"airline": "Delta", "flightCode": "FR0045"}]
```

32. Change the output type to application/java.

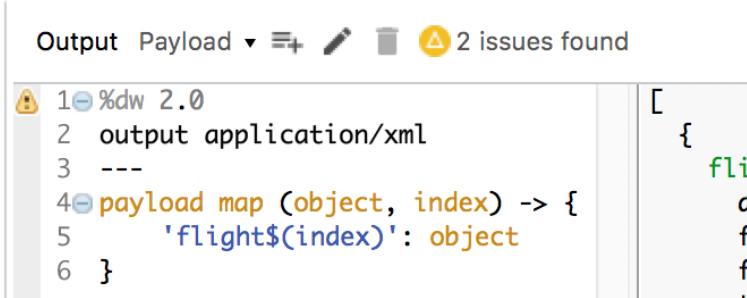
The screenshot shows the MuleSoft Anypoint Studio interface. On the left, the DataWeave editor window has the title "Output Payload" and contains the following code:

```
1 %dw 2.0
2 output application/java
3 ---
4 payload map (object, index) -> {
5     'flight$(index)': object
6 }
```

On the right, the "Preview" tab displays the resulting Java output:

```
[{"flight0": {"airline": "United" as String {class: "java.lang.String"}, flightCode: "ER38sd" as String {class: "java.lang.String"}, fromAirportCode: "LAX" as String {class: "java.lang.String"}, toAirportCode: "SFO" as String {class: "java.lang.String"}, departureDate: "May 21, 2016" as String {class: "java.lang.String"}, emptySeats: 0 as Number {class: "java.lang.Integer"}, totalSeats: 200 as Number {class: "java.lang.Integer"}, price: 199 as Number {class: "java.lang.Integer"}, planeType: "Boeing 737" as String {class: "java.lang.String"} } as Object {class: "java.util.LinkedHashMap"}, } as Object {class: "java.util.LinkedHashMap"}, {"flight1": {"airline": "Delta" as String {class: "java.lang.String"}, flightCode: "FR0045" as String {class: "java.lang.String"} } as Object {class: "java.util.LinkedHashMap"}, } as Object {class: "java.util.LinkedHashMap"}]
```

33. Change the output type to application/xml; you should get issues displayed.

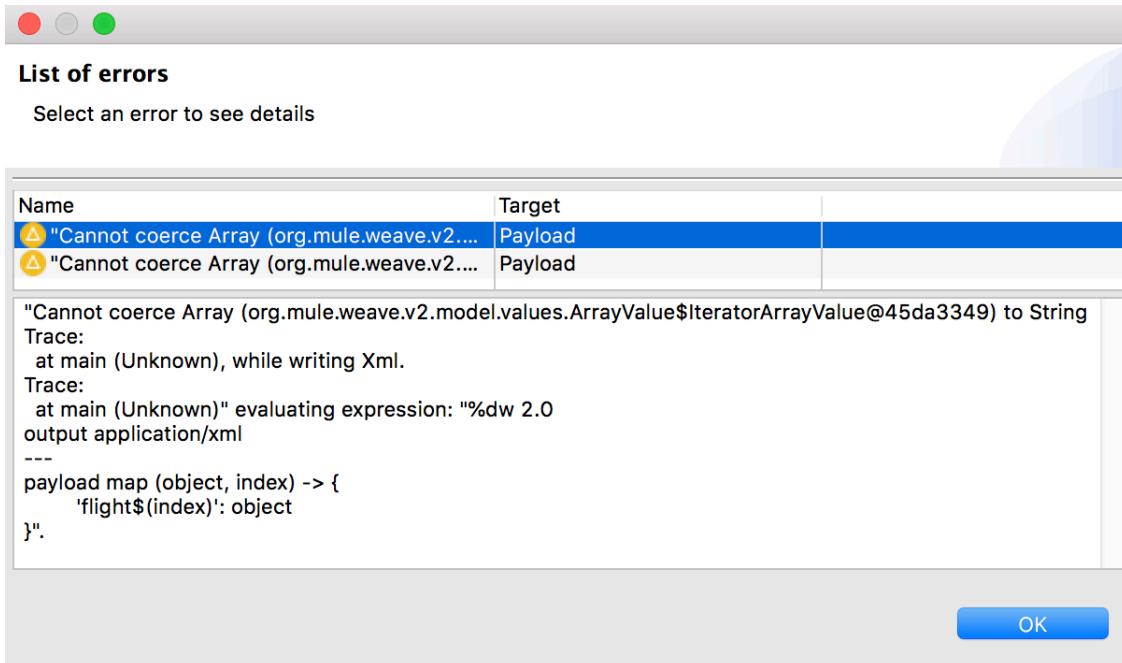


Output Payload ▾ ⚠ 2 issues found

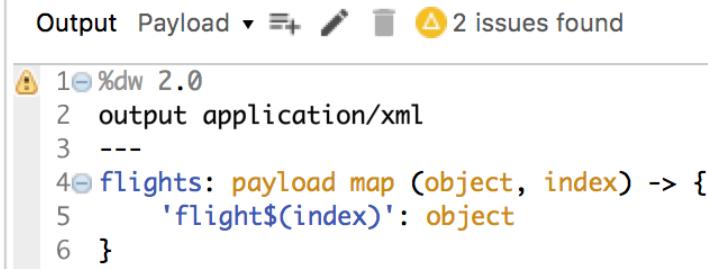
```
1 %dw 2.0
2 output application/xml
3 ---
4 payload map (object, index) -> {
5   'flight$(index)': object
6 }
```

The screenshot shows the Mule Studio interface with the DataWeave editor open. A warning icon (⚠) is visible in the top bar. Below it, a message says "2 issues found". The DataWeave code is displayed in the editor area. The fourth line contains a warning symbol (⚠) before "payload map". The code defines a map where each key is a flight indexed by its position.

34. Click the warning icon, review the issues, then click OK.



35. Change the DataWeave expression to create a root node called flights; you should still get issues.

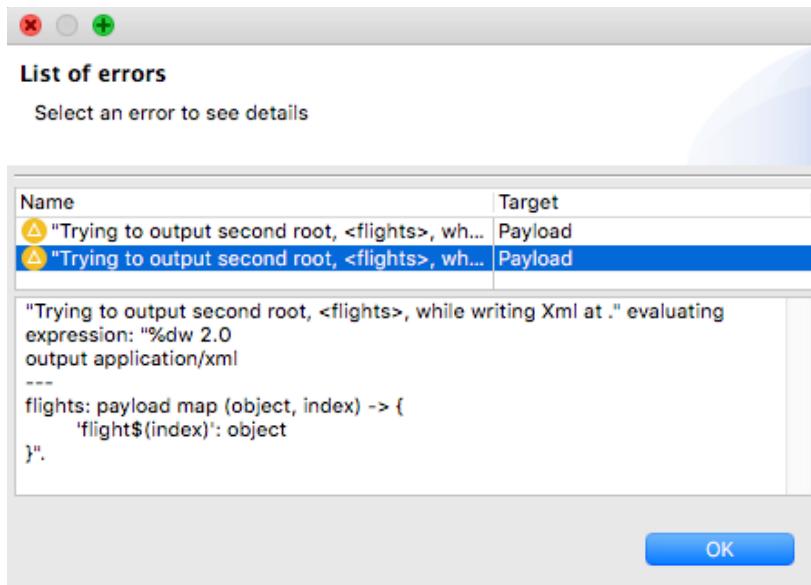


Output Payload ▾ ⚠ 2 issues found

```
1 %dw 2.0
2 output application/xml
3 ---
4 flights: payload map (object, index) -> {
5   'flight$(index)': object
6 }
```

The screenshot shows the Mule Studio interface with the DataWeave editor open. A warning icon (⚠) is visible in the top bar. Below it, a message says "2 issues found". The DataWeave code is displayed in the editor area. The fourth line contains a warning symbol (⚠) before "flights: payload map". The code defines a root node "flights" containing a map where each key is a flight indexed by its position.

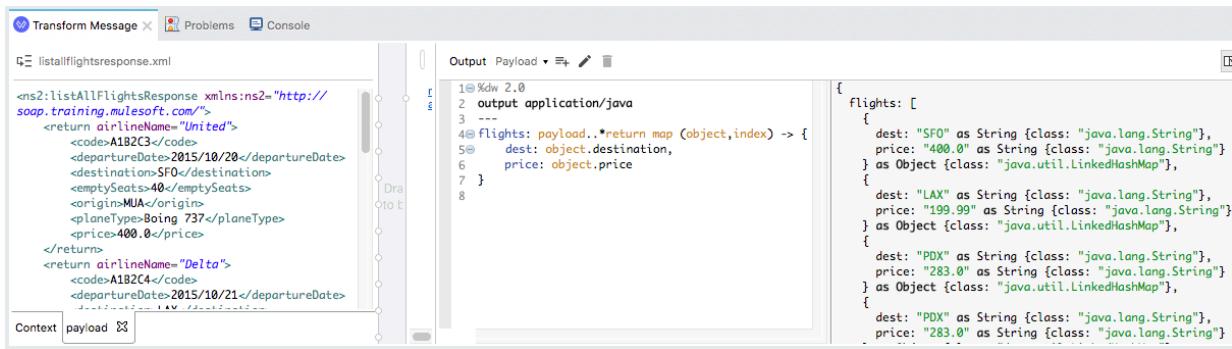
36. Click the warning icon and review the issues.



Walkthrough 11-4: Transform to and from XML with repeated elements

In this walkthrough, you continue to work with the JSON data for multiple flights posted to the flow. You will:

- Transform a JSON array of objects to XML.
- Replace sample data associated with a transformation.
- Transform XML with repeated elements to different data types.

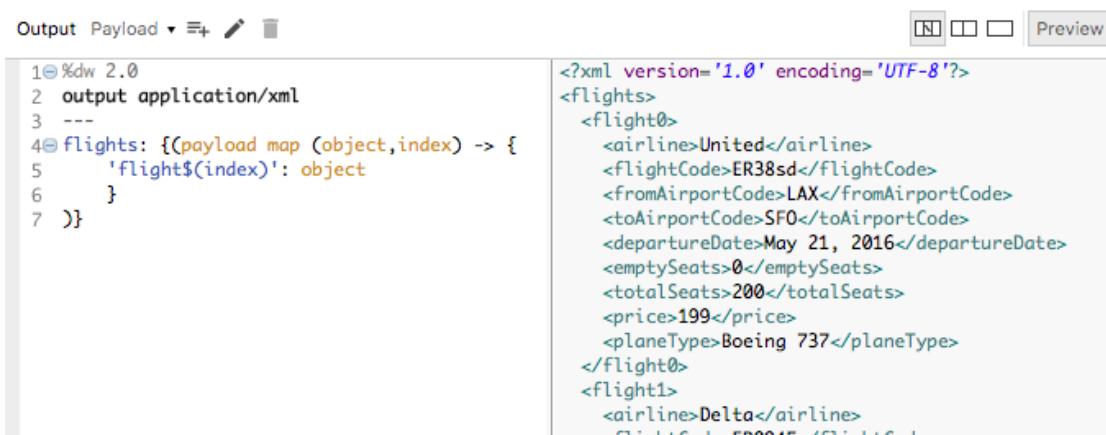


Transform the JSON array of objects to XML

1. Return to the Transform Message properties view for the transformation in postMultipleFlights.
2. Change the expression to map each item in the input array to an XML element.

```
flights: {  
  (payload map (object,index) -> {  
    'flight$(index)': object  
  })  
}
```

3. Look at the preview; the JSON should be transformed to XML successfully.



4. Modify the expression so the flights object has a single property called flight.

The screenshot shows a Mule Studio interface with a transformation step. The left pane displays the MEL code:

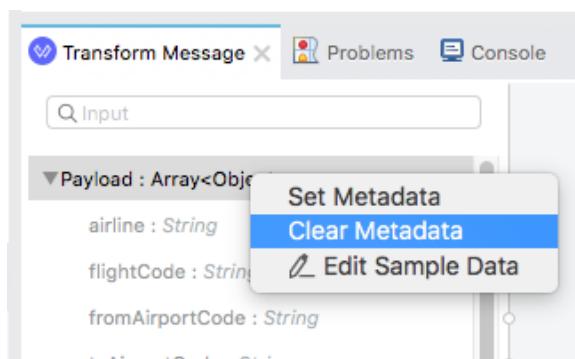
```
1@ %dw 2.0
2  output application/xml
3  ---
4 @ Flights: {(payload map (object,index) -> {
5     flight: object
6   }
7 )}
```

The right pane shows the resulting XML preview:

```
<?xml version='1.0' encoding='UTF-8'?>
<flights>
  <flight>
    <airline>United</airline>
    <flightCode>ER38sd</flightCode>
    <fromAirportCode>LAX</fromAirportCode>
    <toAirportCode>SFO</toAirportCode>
    <departureDate>May 21, 2016</departureDate>
    <emptySeats>0</emptySeats>
    <totalSeats>200</totalSeats>
    <price>199</price>
    <planeType>Boeing 737</planeType>
  </flight>
  <flight>
    <airline>Delta</airline>
```

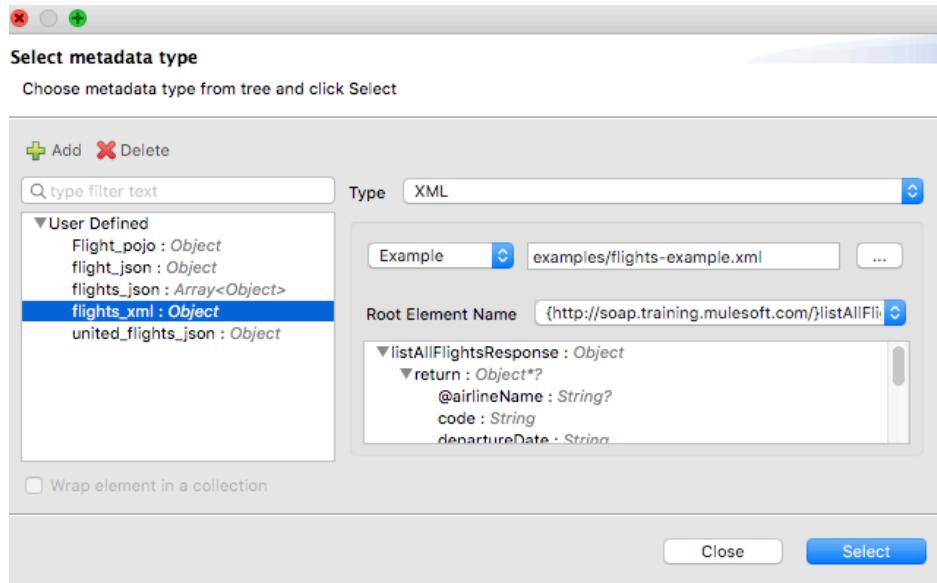
Change the input metadata to XML

5. In the input section, right-click the payload and select Clear Metadata.

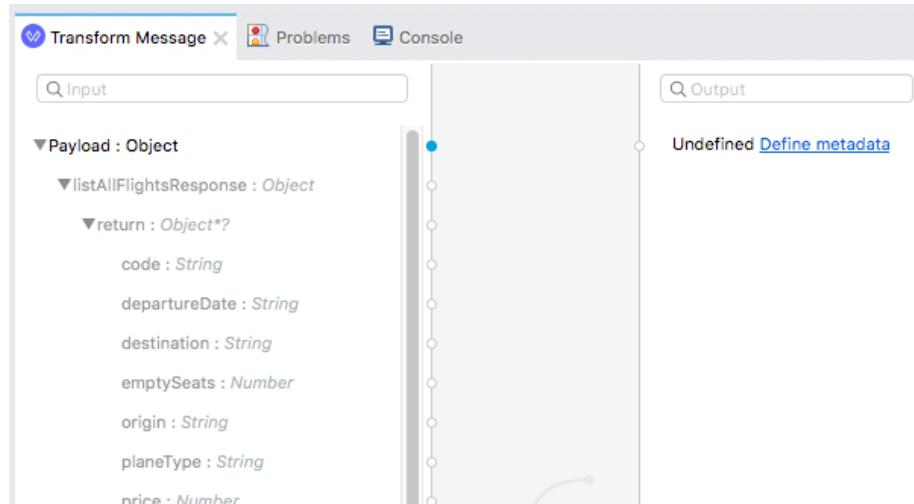


6. Click the Define metadata link.

7. In the Select metadata type dialog box, select flights_xml and click Select.



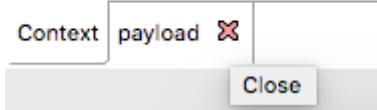
8. In the Transform Message Properties view, you should now see new metadata for the input.



Preview sample data and sample output

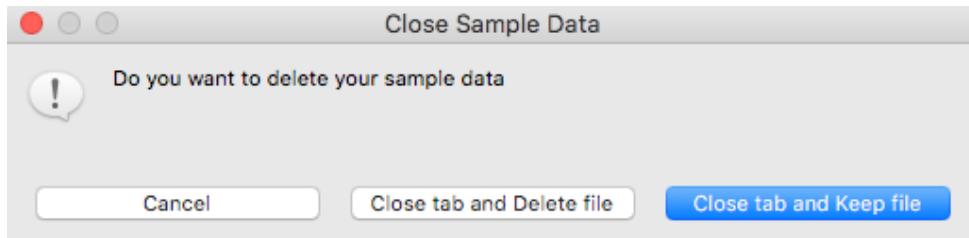
9. In the input section, click the x on the payload tab to close it.

```
▼clientCertificate : Object?  
  encoded : Binary?  
  publicKey : Object?  
  type : String?  
  
queryParams : Object
```



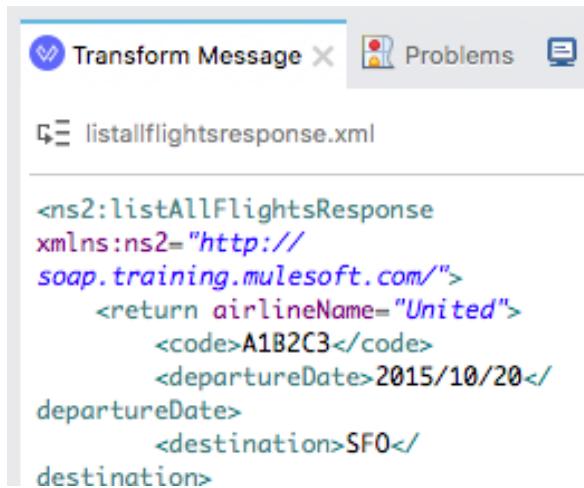
A screenshot of the Mule Studio interface. It shows a 'Context' tab and a 'payload' tab. The 'payload' tab has a red 'X' icon in its top right corner, indicating it is closed. Below the tabs is a 'Close' button.

10. In the Close Sample Data dialog box, click Close tab and Keep file.



11. Right-click the payload and select Edit Sample Data to get sample input data.

12. Look at the XML sample payload.



A screenshot of the 'Transform Message' editor. The title bar says 'Transform Message'. The main area shows an XML document named 'listallflightsresponse.xml'. The XML code is:

```
<ns2:listAllFlightsResponse  
xmlns:ns2="http://  
soap.training.mulesoft.com/">  
  <return airlineName="United">  
    <code>A1B2C3</code>  
    <departureDate>2015/10/20</  
departureDate>  
    <destination>SFO</  
destination>
```

13. Look at the sample_data folder in src/test/resources.

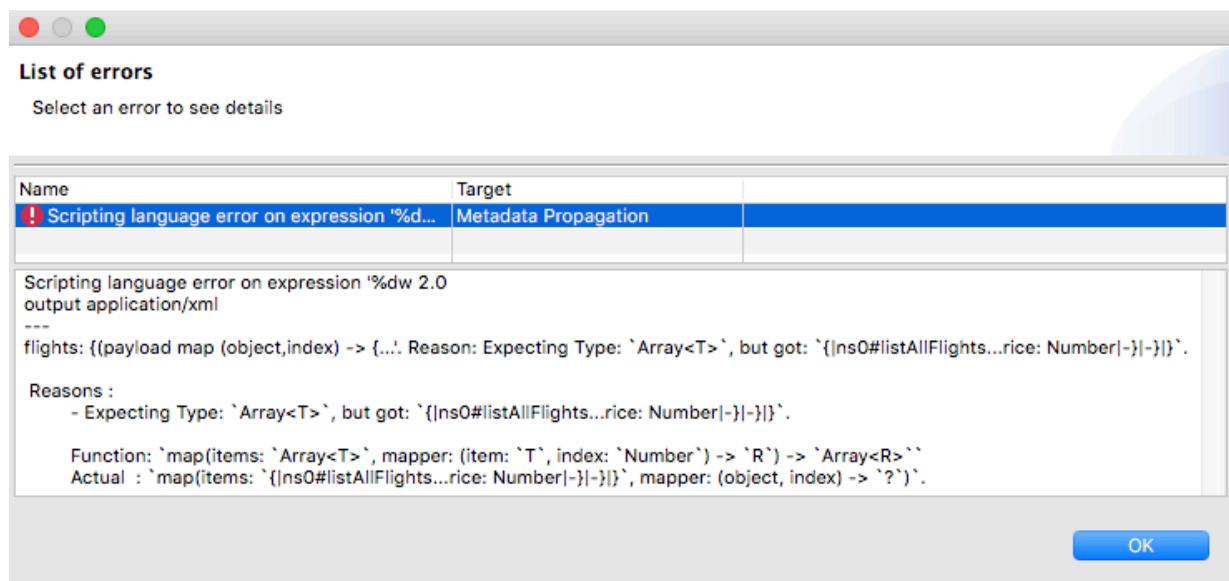


14. Review the transformation expression; you should see an issue.

Output Payload ▾ ! 1 issue found

```
1 @%dw 2.0
2   output application/xml
3   ---
4   flights: {((payload map (object,index) -> {
5     flight: object
6   })
7 ))}
```

15. Look at the error.



Transform XML with repeated elements to a different XML structure

16. Look at the payload metadata in the input section.

The screenshot shows the 'Transform Message' editor in Mule Studio. The 'Input' tab is selected. Under 'Payload : Object', there is a single entry: 'listAllFlightsResponse : Object'. This object has a property 'return' of type 'Object*', which contains three properties: 'code' (String), 'departureDate' (String), and 'destination' (String).

17. Change the DataWeave expression so that the map is iterating over payload..return.

The screenshot shows the 'Transform Message' editor. The 'Output' tab is selected. The DataWeave code is:

```
1 %dw 2.0
2 output application/xml
3 ---
4 flights: {payload..return map (object,index) -> {
5   flight: object
6 }
7 }}
```

The resulting XML output is:

```
<?xml version='1.0' encoding='UTF-8'?>
<flights>
  <flight>
    <code>A1B2C3</code>
    <departureDate>2015/10/20</departureDate>
    <destination>SFO</destination>
    <emptySeats>40</emptySeats>
    <origin>MUA</origin>
    <planeType>Boing 737</planeType>
    <price>400.0</price>
  </flight>
</flights>
```

18. Change the DataWeave expression so that the map is iterating over the repeated return elements of the input.

The screenshot shows the 'Transform Message' editor. The 'Output' tab is selected. The DataWeave code is identical to the previous one:

```
1 %dw 2.0
2 output application/xml
3 ---
4 flights: {payload..*return map (object,index) -> {
5   flight: object
6 }
7 }}
```

The resulting XML output is:

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

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

The screenshot shows the DataWeave editor with the following code:

```

1@ %dw 2.0
2  output application/xml
3  ---
4@ flights: {$(payload..*)return map (object,index) -> {
5@   flight: [
6@     dest: object.destination,
7@     price: object.price
8@   ]
9@ }
10 )}

```

The preview pane shows the resulting XML output:

```

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

```

Note: If you want to test the application with Advanced REST Client, be sure to change the content-type header to application/xml and replace the request body with XML from the flights-example.xml file.

Change the expression to output different data types

20. Change the output type from application/xml to application/dw.

The screenshot shows the DataWeave editor with the following code:

```

1@ %dw 2.0
2  output application/dw
3  ---
4@ flights: {$(payload..*)return map (object,index) -> {
5@   flight: [
6@     dest: object.destination,
7@     price: object.price
8@   ]
9@ }
10 )}

```

The preview pane shows the resulting JSON output:

```

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

```

21. Change the output type to application/json.

The screenshot shows the DataWeave editor with the following code:

```

1@ %dw 2.0
2  output application/json
3  ---
4@ flights: {$(payload..*)return map (object,index) -> {
5@   flight: [
6@     dest: object.destination,
7@     price: object.price
8@   ]
9@ }
10 )}

```

The preview pane shows the resulting JSON output:

```

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

```

22. Change the output type to application/java.

The screenshot shows the DataWeave editor with the following code:

```
1@%dw 2.0
2 output application/java
3 ---
4@ flights: {payload..*return map (object,index) -> {
5@   flight: {
6@     dest: object.destination,
7@     price: object.price
8@   }
9@ }
10 )}
```

The resulting payload is displayed on the right:

```
{ flights: [
  flight: {
    dest: "PDX" as String {class: "java.lang.String"}, price: "283.0" as String {class: "java.lang.String"} } as Object {class: "java.util.LinkedHashMap"} } as Object {class: "java.util.LinkedHashMap"} } as Object {encoding: "UTF-8", mediaType: "*/*", mimeType: "*/*", class: "java.util.LinkedHashMap"}
```

23. In the DataWeave expression, remove the {} around the map expression.

The screenshot shows the DataWeave editor with the following code:

```
1@%dw 2.0
2 output application/java
3 ---
4@ flights: payload..*return map (object,index) -> {
5@   flight: {
6@     dest: object.destination,
7@     price: object.price
8@   }
9@ }
10 )
```

The resulting payload is displayed on the right:

```
{ flights: [
  flight: {
    dest: "SFO" as String {class: "java.lang.String"}, price: "400.0" as String {class: "java.lang.String"} } as Object {class: "java.util.LinkedHashMap"} } as Object {class: "java.util.LinkedHashMap"}, [
  flight: {
    dest: "LAX" as String {class: "java.lang.String"}, price: "199.99" as String {class: "java.lang.String"} } as Object {class: "java.util.LinkedHashMap"} } as Object {class: "java.util.LinkedHashMap"}, [
  flight: {
    dest: "PDX" as String {class: "java.lang.String"}, price: "283.0" as String {class: "java.lang.String"} } as Object {class: "java.util.LinkedHashMap"} ] as Array {class: "java.util.ArrayList"} } as Object {encoding: "UTF-8", mediaType: "*/*", mimeType: "*/*", class: "java.util.LinkedHashMap"}
```

24. In the DataWeave expression, remove the flight property; you should get an ArrayList with five LinkedHashMaps.

The screenshot shows the DataWeave editor with the following code:

```
1@%dw 2.0
2 output application/java
3 ---
4@ flights: payload..*return map (object,index) -> {
5@   dest: object.destination,
6@   price: object.price
7@ }
8 )
```

The resulting payload is displayed on the right:

```
{ flights: [
  {
    dest: "SFO" as String {class: "java.lang.String"}, price: "400.0" as String {class: "java.lang.String"} } as Object {class: "java.util.LinkedHashMap"}, [
    {
      dest: "LAX" as String {class: "java.lang.String"}, price: "199.99" as String {class: "java.lang.String"} } as Object {class: "java.util.LinkedHashMap"}, [
        {
          dest: "PDX" as String {class: "java.lang.String"}, price: "283.0" as String {class: "java.lang.String"} } as Object {class: "java.util.LinkedHashMap"}, [
            {
              dest: "PDX" as String {class: "java.lang.String"}, price: "283.0" as String {class: "java.lang.String"} } as Object {class: "java.util.LinkedHashMap"} ] as Array {class: "java.util.ArrayList"} } as Object {encoding: "UTF-8", mediaType: "*/*", mimeType: "*/*", class: "java.util.LinkedHashMap"}
```

Walkthrough 11-5: Define and use variables and functions

In this walkthrough, you continue to work with the DataWeave transformation in postMultipleFlights. You will:

- Define and use a global constant.
- Define and use a global variable that is equal to a lambda expression.
- Define and use a lambda expression assigned to a variable as a function.
- Define and use a local variable.

The screenshot shows the Mule Studio interface with the DataWeave editor open. The left pane displays the transformation script:

```
1@%dw 2.0
2  output application/dw
3
4@fun getNumSeats(planeType: String) =
5@  if (planeType contains('737'))
6@    150
7@  else
8@    300
9 ---
10@using (flights =
11@  payload..*return map (object,index) -> {
12@    dest: object.destination,
13@    price: object.price,
14@    totalSeats: getNumSeats(object.planeType as String),
15@    plane: object.planeType
16@  }
17 )
18
19 flights
```

The right pane shows the resulting JSON output:

```
[{"dest": "SFO", "price": "400.0", "totalSeats": 150, "plane": "Boing 737"}, {"dest": "LAX", "price": "199.99", "totalSeats": 150, "plane": "Boing 737"}, {"dest": "PDX", "price": "283.0", "totalSeats": 300, "plane": "Boing 777"}, {"...}]
```

Define and use a global constant

1. Return to the Transform Message properties view for the transformation in postMultipleFlights.
2. Change the output type to application/dw.
3. In the header, define a global variable called numSeats that is equal to 400.

```
var numSeats = 400
```

4. In the body, add a totalSeats property that is equal to the numSeats constant.

```
totalSeats: numSeats
```

```
4  var numSeats = 400
5  ---
6@flights: payload..*return map (object,index) -> {
7@  dest: object.destination,
8@  price: object.price,
9@  totalSeats: numSeats
10 }
```

5. Look at the preview.

The screenshot shows a DataWeave editor window. On the left, the code is:

```
1@ %dw 2.0
2  output application/dw
3
4  var numSeats = 400
5  ---
6@ flights: payload..*return map (object,index) -> {
7@     dest: object.destination,
8     price: object.price,
9     totalSeats: numSeats
10 }
11
```

On the right, the preview shows the resulting JSON:

```
{
  flights: [
    {
      dest: "SFO",
      price: "400.0",
      totalSeats: 400
    },
    {
      dest: "LAX",
      price: "199.99",
      totalSeats: 400
    }
  ]
}
```

6. Comment out the variable declaration in the header.

```
//var numSeats = 400
```

Define and use a global variable that is equal to a lambda expression that maps to a constant

7. In the header, define a global variable called numSeats that is equal to a lambda expression with an input parameter x equal to 400.
8. Inside the expression, set numSeats to x.

```
var numSeats = (x=400) -> x
```

9. Add parentheses after numSeats in the totalSeats property assignment.

```
totalSeats: numSeats()
```

10. Look at the preview.

The screenshot shows a DataWeave editor window. On the left, the code is:

```
1@ %dw 2.0
2  output application/dw
3
4 //var numSeats = 400
5  var numSeats = (x = 400) -> x
6  ---
7@ flights: payload..*return map (object,index) -> {
8@     dest: object.destination,
9     price: object.price,
10    totalSeats: numSeats()
11 }
```

On the right, the preview shows the resulting JSON:

```
{
  flights: [
    {
      dest: "SFO",
      price: "400.0",
      totalSeats: 400
    },
    {
      dest: "LAX",
      price: "199.99",
      totalSeats: 400
    }
  ]
}
```

Add a plane property to the output

11. Add a plane property to the DataWeave expression equal to the value of the input planeType values.

12. Look at the values of plane type in the preview.



The screenshot shows the DataWeave editor interface. The left pane displays the DW script:

```
1 %dw 2.0
2  output application/dw
3
4 //var numSeats = 400
5 var numSeats = (x = 400) -> x
6 ---
7 flights: payload..*return map (object,index) -> {
8     dest: object.destination,
9     price: object.price,
10    totalSeats: numSeats(),
11    plane: object.planeType
12 }
```

The right pane shows the preview of the JSON output:

```
[{"dest": "LAX", "price": "199.99", "totalSeats": 400, "plane": "Boing 737"}, {"dest": "PDX", "price": "283.0", "totalSeats": 400, "plane": "Boing 777"}]
```

Define and use a global variable that is that is equal to a lambda expression with input parameters

13. Comment out the numSeats variable declaration in the header.

14. In the header, define a global variable called numSeats that is equal to a lambda expression with an input parameter called planeType of type String.

```
var numSeats = (planeType: String) ->
```

15. Inside the expression, add an if/else block that checks to see if planeType contains the string 737 and sets numSeats to 150 or 300.

```
6---->
7 var numSeats = (planeType: String) ->
8     if (planeType contains('737'))
9         150
10        300
```

16. Change the totalSeats property assignment to pass the object's planeType property to numSeats.

```
totalSeats: numSeats(object.planeType)
```

17. Force the argument to a String.

```
totalSeats: numSeats(object.planeType as String)
```

18. Look at the preview; you should see values of 150 and 300.

The screenshot shows the Mule Studio interface with a dw (DataWeave) script editor on the left and a preview pane on the right. The dw script contains logic to calculate the number of seats based on the plane type. The preview pane shows three flight records with their respective details.

```
1@ %dw 2.0
2  output application/dw
3
4 //var numSeats = 400
5 //var numSeats = (x = 400) -> x
6@ var numSeats = (planeType: String) ->
7@   if (planeType contains('737'))
8     150
9   else
10    300
11 ---
12@ flights: payload..*return map (object,index) -> {
13@   dest: object.destination,
14@   price: object.price,
15@   totalSeats: numSeats(object.planeType as String),
16@   plane: object.planeType
17 }
```

```
price: "400.0",
totalSeats: 150,
plane: "Boing 737"
},
{
dest: "LAX",
price: "199.99",
totalSeats: 150,
plane: "Boing 737"
},
{
dest: "PDX",
price: "283.0",
totalSeats: 300,
plane: "Boing 777"
},
```

19. Surround the variable declaration in the header with /* and */ to comment it out.

Define and use a lambda expression assigned to a variable as a function

20. In the header, use the fun keyword to define a function called getNumSeats with a parameter called planeType of type String.

```
fun getNumSeats(planeType: String)
```

21. Copy the if/else expression in the numSeats declaration.

22. Set the getNumSeats function equal to the if/else expression.

```
fun getNumSeats(planeType: String) =
  if (planeType contains('737'))
    150
  else
    300
```

23. In the body, change the totalSeats property to be equal to the result of the getNumSeats function.

```
totalSeats: getNumSeats(object.planeType as String)
```

24. Look at the preview.

The screenshot shows the MuleSoft Anypoint Studio interface. On the left, there is a code editor window titled "Output Payload" containing a DataWeave script. The script includes logic to calculate the number of seats based on plane type and a loop to generate flight data. On the right, there is a preview window showing the resulting JSON output, which lists three flights with their destination, price, total seats, and plane type.

```
6  /* var numSeats = (planeType: String) ->
7      if (planeType contains('737'))
8          150
9      else
10         300
11 */
12 fun getNumSeats(planeType: String) =
13     if (planeType contains('737'))
14         150
15     else
16         300
17 ---
18 flights: payload..*return map (object,index) -> {
19     dest: object.destination,
20     price: object.price,
21     totalSeats: getNumSeats(object.planeType as String),
22     plane: object.planeType
23 }
```

```
{
  flights: [
    {
      dest: "SFO",
      price: "400.0",
      totalSeats: 150,
      plane: "Boing 737"
    },
    {
      dest: "LAX",
      price: "199.99",
      totalSeats: 150,
      plane: "Boing 737"
    },
    {
      dest: "PDX",
      price: "283.0",
      totalSeats: 300,
      plane: "Boing 777"
    }
  ]
}
```

Create and use a local variable

25. In the body, surround the existing DataWeave expression with parentheses and add the using keyword in front of it.

```
18 using (flights: payload..*return map (object,index) -> {
19     dest: object.destination,
20     price: object.price,
21     totalSeats: getNumSeats(object.planeType as String),
22     plane: object.planeType
23 }
24 )
```

26. Change the colon after flights to an equal sign.

27. Modify the indentation to your liking.

28. After the local variable declaration, set the transformation expression to be the local flights variable.

```
18 using (flights =
19     payload..*return map (object,index) -> {
20         dest: object.destination,
21         price: object.price,
22         totalSeats: getNumSeats(object.planeType as String),
23         plane: object.planeType
24     }
25 )
26
27 flights
```

29. Look at the preview.

Output Payload ▾ ⌂

```
10      300
11  */
12 fun getNumSeats(planeType: String) =
13     if (planeType contains('737'))
14         150
15     else
16         300
17 ---
18 using (flights =
19 payload.*return map (object,index) -> {
20     dest: object.destination,
21     price: object.price,
22     totalSeats: getNumSeats(object.planeType as String),
23     plane: object.planeType
24 }
25 )
26 |
27 flights
```

[

```
{
    dest: "SFO",
    price: "400.0",
    totalSeats: 150,
    plane: "Boing 737"
},
{
    dest: "LAX",
    price: "199.99",
    totalSeats: 150,
    plane: "Boing 737"
},
{
    dest: "PDX",
    price: "283.0",
    totalSeats: 300,
    plane: "Boing 777"
}
```

Walkthrough 11-6: Coerce and format strings, numbers, and dates

In this walkthrough, you continue to work with the XML flights data posted to the postMultipleFlights flow. You will:

- Coerce data types.
 - Format strings, numbers, and dates.

```
Output Payload ▾ 🖊️ 🗑️ Preview
18@ using (flights =
19@   payload..*return map (object,index) -> {
20@     dest: object.destination,
21@     price: object.price as Number as String {format: "###.00"},
22@     totalSeats: getNumSeats(object.planeType as String),
23@     plane: upper(object.planeType as String),
24@     date: object.departureDate as Date {format: "yyyy/MM/dd"}
25@       as String {format: "MMM dd, yyyy"}
26@   }
27@ }
28@ 
29@ flights
```

dest: "SFO",
price: "400.00" as String {format: "##,.00"},
totalSeats: 150,
plane: "BOING 737",
date: "Oct 20, 2015" as String {format: "MMM dd, yyyy"}
},
{
 dest: "LAX",
 price: "199.99" as String {format: "##,.00"},
 totalSeats: 150,
 plane: "BOING 737",
 date: "Oct 21, 2015" as String {format: "MMM dd, yyyy"}
},

Format a string

1. Return to the Transform Message properties view for the transformation in postMultipleFlights.
 2. Use the upper function to return the plane value in uppercase.

```
18@ using (flights =  
19@     payload..*return map (object,index) -> {  
20@         dest: object.destination,  
21@         price: object.price,  
22@         totalSeats: getNumSeats(object.planeType as String),  
23@         plane: upper(object.planeType)  
24@     }  
25@ }
```

3. Coerce the argument to a String.

```
plane: upper(object.planeType as String)
```

- #### 4. Look at the preview.

```
Output Payload ▾ ⌂ ⌂ ⌂ Previous  
15     else  
16         300  
17 ---  
18 using (flights =  
19 payload.*return map (object,index) -> {  
20     dest: object.destination,  
21     price: object.price,  
22     totalSeats: getNumSeats(object.planeType as String),  
23     plane: upper(object.planeType as String)  
24 }  
25 )  
26  
27 flights
```

[
{
 dest: "SFO",
 price: "400.0",
 totalSeats: 150,
 plane: "BOING 737"
},
{
 dest: "LAX",
 price: "199.99",
 totalSeats: 150,
 plane: "BOING 737"
},
{

Coerce a string to a number

5. In the preview, look at the data type of the prices; you should see that they are strings.
6. Change the DataWeave expression to use the as keyword to return the prices as numbers.

```
price: object.price as Number,
```

7. Look at the preview.

The screenshot shows the Mule Studio interface with the DataWeave editor and a preview pane. The DataWeave code is as follows:

```
15     else
16         300
17 ---
18 @using (flights =
19 @    payload.*return map (object,index) -> {
20 @        dest: object.destination,
21 @        price: object.price as Number,
22 @        totalSeats: getNumSeats(object.planeType as String),
23 @        plane: upper(object.planeType as String)
24    }
25 )
26
27 flights
```

The preview pane shows the resulting JSON output:

```
[{"dest": "SFO", "price": 400.0, "totalSeats": 150, "plane": "BOING 737"}, {"dest": "LAX", "price": 199.99, "totalSeats": 150, "plane": "BOING 737"}],
```

8. Change the output type from application/dw to application/java.
9. Look at the preview; you should see the prices are now either Integer or Double objects.

The screenshot shows the Mule Studio interface with the DataWeave editor and a preview pane. The DataWeave code is as follows:

```
12@ fun getNumSeats (planeType: String) =
13@   if (planeType contains('737'))
14@     150
15@   else
16@     300
17 ---
18@using (flights =
19@    payload.*return map (object,index) -> {
20@        dest: object.destination,
21@        price: object.price as Number,
22@        totalSeats: getNumSeats(object.planeType as String),
23@        plane: upper(object.planeType as String)
24    }
25 )
26
27 flights
```

The preview pane shows the resulting Java objects:

```
[{"dest": "SFO" as String {class: "java.lang.String"}, "price": 400 as Number {class: "java.lang.Integer"}, "totalSeats": 150 as Number {class: "java.lang.Integer"}, "plane": "BOING 737" as String {class: "java.lang.String"}] as Object {class: "java.util.LinkedHashMap"}, {"dest": "LAX" as String {class: "java.lang.String"}, "price": 199.99 as Number {class: "java.lang.Double"}, "totalSeats": 150 as Number {class: "java.lang.Integer"}, "plane": "BOING 737" as String {class: "java.lang.String"}] as Object {class: "java.util.LinkedHashMap"}, {"dest": "PDX" as String {class: "java.lang.String"}, "price": 283 as Number {class: "java.lang.Integer"},
```

Coerce a string to a specific type of number object

10. Change the DataWeave expression to use the class metadata key to coerce the prices to java.lang.Double objects.

```
price: object.price as Number {class:"java.lang.Double"},
```

11. Look at the preview; you should see the prices are now all Double objects.

The screenshot shows the Mule Studio interface with two main panes. The left pane is titled "Output Payload" and contains Java code. The right pane is titled "Preview" and displays JSON data.

```
12 fungetNumSeats(planeType: String) =  
13     if (planeType contains('737'))  
14         150  
15     else  
16         300  
17 ---  
18 using (flights =  
19     payload..*return map (object,index) -> {  
20         dest: object.destination,  
21         price: object.price as Number [class: "java.lang.Double"],  
22         totalSeats: getNumSeats(object.planeType as String),  
23         plane: upper(object.planeType as String)  
24     }  
25 )  
26  
27 flights
```

```
[  
{  
    dest: "SFO" as String [class: "java.lang.String"],  
    price: 400.0 as Number [class: "java.lang.Double"],  
    totalSeats: 150 as Number [class: "java.lang.Integer"],  
    plane: "BOING 737" as String [class: "java.lang.String"]  
} as Object [class: "java.util.LinkedHashMap"],  
{  
    dest: "LAX" as String [class: "java.lang.String"],  
    price: 199.99 as Number [class: "java.lang.Double"],  
    totalSeats: 150 as Number [class: "java.lang.Integer"],  
    plane: "BOING 737" as String [class: "java.lang.String"]  
} as Object [class: "java.util.LinkedHashMap"],  
{  
    dest: "PDX" as String [class: "java.lang.String"],  
    price: 283.0 as Number [class: "java.lang.Double"],  
}
```

12. Change the output type from application/java back to application/dw.

The screenshot shows the Mule Studio interface with two main panes. The left pane is titled "Output Payload" and contains Java code. The right pane is titled "Preview" and displays JSON data.

```
12 fungetNumSeats(planeType: String) =  
13     if (planeType contains('737'))  
14         150  
15     else  
16         300  
17 ---  
18 using (flights =  
19     payload..*return map (object,index) -> {  
20         dest: object.destination,  
21         price: object.price as Number [class: "java.lang.Double"],  
22         totalSeats: getNumSeats(object.planeType as String),  
23         plane: upper(object.planeType as String)  
24     }  
25 )  
26  
27 flights
```

```
[  
{  
    dest: "SFO",  
    price: 400.0 as Number [class: "java.lang.Double"],  
    totalSeats: 150,  
    plane: "BOING 737"  
},  
{  
    dest: "LAX",  
    price: 199.99 as Number [class: "java.lang.Double"],  
    totalSeats: 150,  
    plane: "BOING 737"  
},  
{  
    dest: "PDX",  
    price: 283.0 as Number [class: "java.lang.Double"],  
}
```

Format a number

13. Remove the coercion of the price to a java.lang.Double.

14. Coerce the price to a String and use the format schema property to format the prices to zero decimal places.

```
price: object.price as Number as String {format: "###"},
```

15. Look at the preview; the prices should now be formatted.

Output Payload ▾

```
17 ---  
18@using (flights =  
19@    payload..*return map (object,index) -> {  
20@        dest: object.destination,  
21@        price: object.price as Number as String {format: "###"},  
22@        totalSeats: getNumSeats(object.planeType as String),  
23@        plane: upper(object.planeType as String)  
24@    }  
25 )  
26  
27 flights
```

[
{
dest: "SFO",
price: "400" as String {format: "###"},
totalSeats: 150,
plane: "BOING 737"
},
{
dest: "LAX",
price: "200" as String {format: "###"},
totalSeats: 150,

16. Change the DataWeave expression to format the prices to two decimal places.

```
price: object.price as Number as String {format: "###.##"},
```

17. Look at the preview.

Output Payload ▾

```
17 ---  
18@using (flights =  
19@    payload..*return map (object,index) -> {  
20@        dest: object.destination,  
21@        price: object.price as Number as String {format: "##.##"},  
22@        totalSeats: getNumSeats(object.planeType as String),  
23@        plane: upper(object.planeType as String)  
24@    }  
25 )  
26  
27 flights
```

[
{
dest: "SFO",
price: "400" as String {format: "##.##"},
totalSeats: 150,
plane: "BOING 737"
},
{
dest: "LAX",
price: "199.99" as String {format: "##.##"},
totalSeats: 150,

18. Change the DataWeave expression to format the prices to two minimal decimal places.

```
price: object.price as Number as String {format: "##.00"},
```

19. Look at the preview; all the prices should now be formatted to two decimal places.

Output Payload ▾

```
17 ---  
18@using (flights =  
19@    payload..*return map (object,index) -> {  
20@        dest: object.destination,  
21@        price: object.price as Number as String {format: "##.00"},  
22@        totalSeats: getNumSeats(object.planeType as String),  
23@        plane: upper(object.planeType as String)  
24@    }  
25 )  
26  
27 flights
```

[
{
dest: "SFO",
price: "400.00" as String {format: "##.00"},
totalSeats: 150,
plane: "BOING 737"
},
{
dest: "LAX",
price: "199.99" as String {format: "##.00"},
totalSeats: 150,

Note: If you are not a Java programmer, you may want to look at the Java documentation for the DecimalFormat class to review documentation on DecimalFormat patterns.

<https://docs.oracle.com/javase/8/docs/api/java/text/DecimalFormat.html>

Coerce a string to a date

20. Add a date field to the return object.

```
date: object.departureDate
```

21. Look at the preview; you should see the date property is a String.

The screenshot shows the DataWeave editor with the following code:

```
17 ---  
18 @using(flights =  
19@ payload.*return map(object,index) -> {  
20@   dest: object.destination,  
21@   price: object.price as Number {format: "###.00"},  
22@   totalSeats: getNumSeats(object.planeType as String),  
23@   plane: upper(object.planeType as String),  
24@   date: object.departureDate  
25 }  
26 )  
27  
28 flights
```

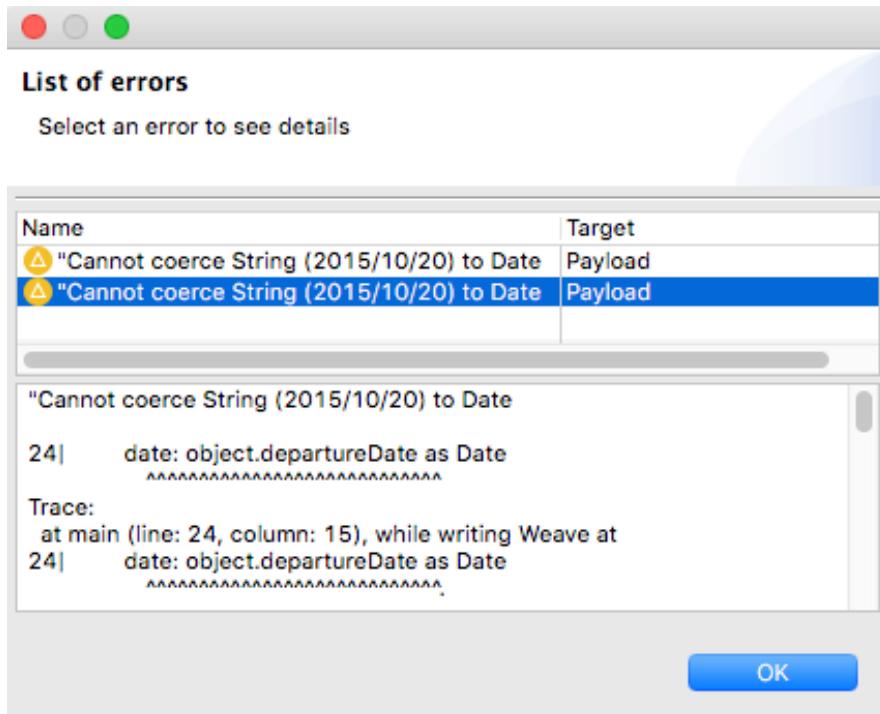
The preview pane shows the resulting JSON output:

```
price: "400.00" as String {format: "###.00"},  
totalSeats: 150,  
plane: "BOING 737",  
date: "2015/10/20"  
},  
{  
dest: "LAX",  
price: "199.99" as String {format: "###.00"},  
totalSeats: 150,  
plane: "BOING 737",  
date: "2015/10/21"  
},
```

22. Change the DataWeave expression to use the as operator to convert departureDate to a date object; you should get issues displayed.

```
date: object.departureDate as Date
```

23. Look at the errors.



24. Look at the format of the dates in the preview.

25. Change the DataWeave expression to use the format schema property to specify the pattern of the input date strings.

```
date: object.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.

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

26. Look at the preview; you should see date is now a Date object.

The screenshot shows the MuleSoft Anypoint Studio interface. On the left, the DataWeave editor contains the following code:

```
18@using (flights =
19@    payload.*return map (object,index) -> {
20@      dest: object.destination,
21@      price: object.price as Number {format: "##.##"}, // Using format schema
22@      totalSeats: getNumSeats(object.planeType as String),
23@      plane: upper(object.planeType as String),
24@      date: object.departureDate as Date {format: "yyyy/MM/dd"} // Using format schema
25@    }
26@  )
27@ flights
```

On the right, the Preview tab shows the resulting JSON output:

```
price: "400.00" as String {format: "##.##"}, // Using format schema
totalSeats: 150,
plane: "BOING 737",
date: 2015-10-20 as Date {format: "yyyy/MM/dd"} // Using format schema
},
{
dest: "LAX",
price: "199.99" as String {format: "##.##"}, // Using format schema
totalSeats: 150,
plane: "BOING 737",
date: 2015-10-21 as Date {format: "yyyy/MM/dd"} // Using format schema
},
```

Format a date

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

```
date: object.departureDate as Date {format: "yyyy/MM/dd"} as String
```

28. Look at the preview section; the date is again a String – but with a different format.

The screenshot shows the MuleSoft Anypoint Studio interface. On the left, the DataWeave editor contains the following code:

```
17 ---
18@using (flights =
19@    payload.*return map (object,index) -> {
20@      dest: object.destination,
21@      price: object.price as Number {format: "##.##"}, // Using format schema
22@      totalSeats: getNumSeats(object.planeType as String),
23@      plane: upper(object.planeType as String),
24@      date: object.departureDate as Date {format: "yyyy/MM/dd"} as String // Using as operator to convert Date to String
25@    }
26@  )
27@ flights
```

On the right, the Preview tab shows the resulting JSON output:

```
price: "400.00" as String {format: "##.##"}, // Using format schema
totalSeats: 150,
plane: "BOING 737",
date: "2015-10-20" // Using as operator to convert Date to String
},
{
dest: "LAX",
price: "199.99" as String {format: "##.##"}, // Using format schema
totalSeats: 150,
plane: "BOING 737",
date: "2015-10-21" // Using as operator to convert Date to String
},
```

29. Use the format schema property with any pattern letters and characters to format the date strings.

```
date: object.departureDate as Date {format: "yyyy/MM/dd"} as String
{format: "MMM dd, yyyy"}
```

30. Look at the preview; the dates should now be formatted according to the pattern.

The screenshot shows a MuleSoft Anypoint Studio interface. On the left, there is a code editor window with the following Java code:

```
18@ using (flights =  
19@     payload..*return map (object,index) -> {  
20@         dest: object.destination,  
21@         price: object.price as Number {format: "###.##"},  
22@         totalSeats: getNumSeats(object.planeType as String),  
23@         plane: upper(object.planeType as String),  
24@         date: object.departureDate as Date {format: "yyyy/MM/dd"}  
25@             as String {format: "MMM dd, yyyy"}  
26@     }  
27@ }  
28@  
29@ flights
```

On the right, there is a preview pane showing the resulting JSON output:

```
dest: "SFU",  
price: "400.00" as String {format: "##.##"},  
totalSeats: 150,  
plane: "BOING 737",  
date: "Oct 20, 2015" as String {format: "MMM dd, yyyy"}  
,  
{  
dest: "LAX",  
price: "199.99" as String {format: "##.##"},  
totalSeats: 150,  
plane: "BOING 737",  
date: "Oct 21, 2015" as String {format: "MMM dd, yyyy"}  
,
```

Walkthrough 11-7: Define and use custom data types

In this walkthrough, you continue to work with the flight JSON posted to the flow. You will:

- Define and use custom data types.
- Transform objects to POJOs.

The screenshot shows the Mule Studio interface with the DataWeave editor open. The code on the left is:

```
18    300
19 ---
20@ using (flights =
21@   payload..*return map (object,index) -> {
22@     destination: object.destination,
23@     price: object.price as Number as Currency,
24@   }
25@     totalSeats: getNumSeats(object.planeType as String),
26@     planeType: upper(object.planeType as String),
27@     departureDate: object.departureDate as Date [format:
28@       as String {format: "MMM dd, yyyy"}]
29  } as Flight
30
31 flights
```

The resulting JSON output on the right is:

```
[{"flightCode": null, "availableSeats": 0 as Number {class: "java.lang.Integer"}, "destination": "SFO" as String {class: "java.lang.String"}, "planeType": "BOING 737" as String {class: "java.lang.String"}, "price": 400.0 as Number {class: "java.lang.Double"}, "origination": null, "departureDate": "Oct 20, 2015" as String {class: "java.lang.String"}, "airlineName": null} as Object {class: "com.mulesoft.training.Flight"}, {"flightCode": null, "availableSeats": 0 as Number {class: "java.lang.Integer"}, "destination": "LAX" as String {class: "java.lang.String"}]
```

Define a custom data type

1. Return to the Transform Message properties view for the transformation in postMultipleFlights.
2. Select and cut the string formatting expression for the prices.

```
18@ using (flights =
19@   payload..*return map (object,index) -> {
20@     dest: object.destination,
21@     price: object.price as Number as String [format: "###.00"],
22@     totalSeats: getNumSeats(object.planeType as String),
23@     plane: upper(object.planeType as String)
```

3. In the header section of the expression, define a custom data type called Currency.
4. Set it equal to the value you cut.

```
type Currency = String {format: "###.00"}
```

The screenshot shows the Mule Studio interface with the DataWeave editor open. The code is:

```
1@ %dw 2.0
2@ output application/dw
3@ type Currency = String [format: "###.00"]
4@ //var numSeats = 400
5@ //var numSeats = (v-100) - v
```

Use a custom data type

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

```
price: object.price as Number as Currency,
```

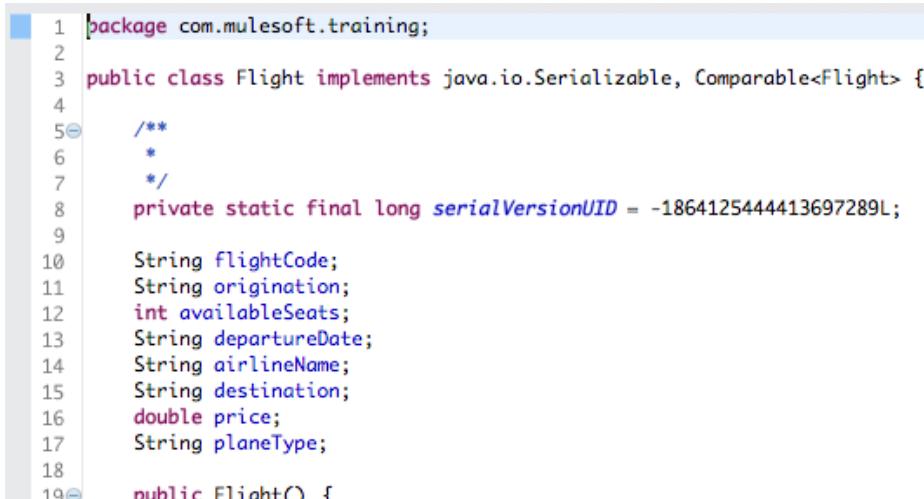
6. Look at the preview; the prices should still be formatted as strings to two decimal places.



```
18 using (flights =  
19     payload..*return map (object,index) -> {  
20         dest: object.destination,  
21         price: object.price as Number as Currency,  
22         totalSeats: getNumSeats(object.planeType as String),  
23         plane: upper(object.planeType as String),  
24         date: object.departureDate as Date [format: "yyyy/MM/dd"]  
25     }  
26 }  
27  
[  
{  
    dest: "SFO",  
    price: "400.00" as String [format: "###.##"],  
    totalSeats: 150,  
    plane: "BOING 737",  
    date: "Oct 20 2015" as String [format: "MMM dd  
yy"]  
}  
]
```

Transform objects to POJOs

7. Open the Flight.java class in the project's src/main/java folder, look at the names of the properties then close the file.



```
1 package com.mulesoft.training;  
2  
3 public class Flight implements java.io.Serializable, Comparable<Flight> {  
4  
5     /**  
6      *  
7      */  
8     private static final long serialVersionUID = -1864125444413697289L;  
9  
10    String flightCode;  
11    String origination;  
12    int availableSeats;  
13    String departureDate;  
14    String airlineName;  
15    String destination;  
16    double price;  
17    String planeType;  
18  
19    public Flight() {  
20    }
```

8. Return to the Transform Message properties view for the transformation in postMultipleFlights.
9. In the header section of the expression, define a custom data type called flight that is of type com.mulesoft.training.Flight.

```
type Flight = Object {class: "com.mulesoft.training.Flight"}
```



```
1 %dw 2.0  
2 output application/dw  
3 type Currency = String {format: "###.##"}  
4 type Flight = Object {class: "com.mulesoft.training.Flight"}
```

10. In the DataWeave expression, set the map objects to be of type Flight.

```
20 using (flights =
21   payload..*return map (object,index) -> {
22     dest: object.destination,
23     price: object.price as Number as Currency,
24     totalSeats: getNumSeats(object.planeType as String),
25     plane: upper(object.planeType as String),
26     date: object.departureDate as Date {format: "yyyy/MM/dd"}
27       as String {format: "MMM dd, yyyy"}
28   } as Flight
29 )
```

11. Look at the preview.

Output Payload ▾

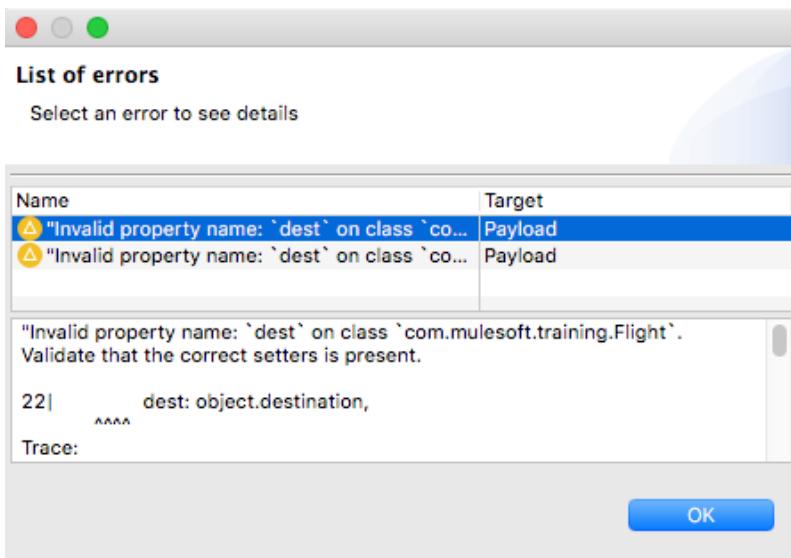
17
18 300
19 ---
20 using (flights =
21 payload..*return map (object,index) -> {
22 dest: object.destination,
23 price: object.price as Number as Currency,
24 totalSeats: getNumSeats(object.planeType as String),
25 plane: upper(object.planeType as String),
26 date: object.departureDate as Date {format: "yyyy/MM/dd"}
27 as String {format: "MMM dd, yyyy"}
28 } as Flight
29)
30
31 flights

[{"dest": "SFO", "price": "400.00" as String {format: "###.00"}, "totalSeats": 150, "plane": "BOING 737", "date": "Oct 20, 2015" as String {format: "MMM dd, yyyy"}} as Object {class: "com.mulesoft.training.Flight"}, {"dest": "LAX", "price": "199.99" as String {format: "###.00"}, "totalSeats": 150, "plane": "BOING 737", "date": "Oct 21, 2015" as String {format: "MMM dd, yyyy"}} as Object {class: "com.mulesoft.training.Flight"},]

12. Change the output type from application/dw to application/java.

```
1 %dw 2.0
2 output application/java
3 type Currency = String {format: "###.00"}
4 type Flight = Object {class: "com.mulesoft.training.Flight"}  
-
```

13. Look at the issues you get.



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

```
destination: object.destination,
```

15. Change the other keys to match the names of the Flight class properties:

- plane to planeType
- date to departureDate

16. Comment out the totalSeats property.

17. Look at the preview.

The screenshot shows the Mule Studio interface with the 'Preview' tab selected. On the left, the Java code for a script component is displayed:

```
18      300
19  ---
20@using (Flights =
21@    payload..*return map (object,index) -> {
22@      destination: object.destination,
23@      price: object.price as Number as Currency,
24@      /| totalSeats: getNumSeats(object.planeType as String,
25@      planeType: upper(object.planeType as String),
26@      departureDate: object.departureDate as Date {format
27@        as String {format: "MMM dd, yyyy"}}
28@      } as Flight
29@    )
30
31 flights
```

On the right, the resulting JSON output is shown:

```
{
  flightCode: null,
  availableSeats: 0 as Number {class: "java.lang.Integer"},
  destination: "SFO" as String {class: "java.lang.String"},
  planeType: "BOING 737" as String {class: "java.lang.String"},
  price: 400.0 as Number {class: "java.lang.Double"},
  origination: null,
  departureDate: "Oct 20, 2015" as String {class: "java.lang.String"},
  airlineName: null
} as Object {class: "com.mulesoft.training.Flight"},

{
  flightCode: null,
  availableSeats: 0 as Number {class: "java.lang.Integer"},
  destination: "LAX" as String {class: "java.lang.String"}  
}
```

Note: If you want to test the application with Advanced REST Client, be sure to change the content-type header to application/xml and replace the request body with XML from the flights-example.xml file.

The screenshot shows the Mule Studio interface with the 'implementation' tab selected. At the top, the 'Mule Debugger' window displays the state of variables:

```
Logger = Logger
attributes = {HttpRequestAttributes} org.mule.extension.http.api.I
correlationId = "7a7756d0-8b1b-11e9-a9cb-f018983d9329"
payload = {ArrayList} size = 5
  0 = {Flight} com.mulesoft.training.Flight@7c14388b
  1 = {Flight} com.mulesoft.training.Flight@183f85b9
  2 = {Flight} com.mulesoft.training.Flight@41d6c73
  3 = {Flight} com.mulesoft.training.Flight@49f370a9
  4 = {Flight} com.mulesoft.training.Flight@147ec97c
    airlineName = "null"
    availableSeats = 0
    departureDate = "Oct 20, 2015"
    destination = "PDX"
    flightCode = "null"
    origination = "null"
    planeType = "BOING 777"
    price = 283.0
    serialVersionUID = -1864125444413697289
    ^mediaType = application/java; charset=UTF-8
vars = {Map} size = 0
```

Below the debugger, the 'implementation' tab shows the flow diagram:

```
postMultipleFlights
  Listener POST /multipleflights --> Transform Message --> Logger
```

Walkthrough 11-8: Use DataWeave functions

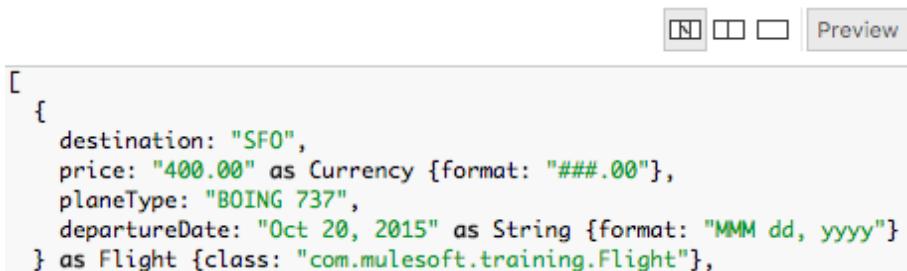
In this walkthrough, you continue to work with the flights JSON posted to the flow. You will:

- Use functions in the Core module that are imported automatically.
- Replace data values using pattern matching.
- Order data, remove duplicate data, and filter data.
- Use a function in another module that you must explicitly import into a script to use.
- Dasherize data.

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

Use the replace function

1. Return to the Transform Message properties view for the transformation in postMultipleFlights.
2. Change the output type from application/java to application/dw.
3. Look at the preview and see that Boeing is misspelled.



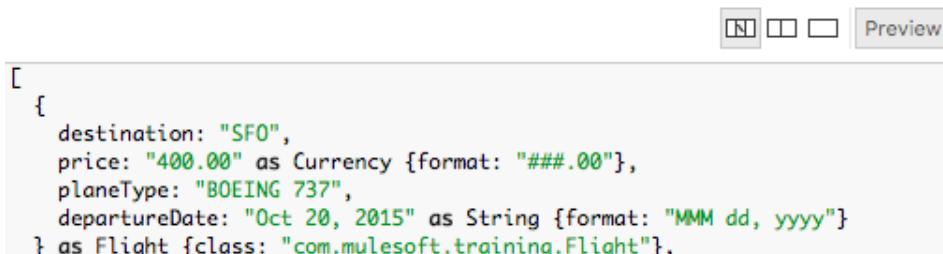
The screenshot shows the DataWeave preview pane with the following code and output:

```
[  
  {  
    destination: "SFO",  
    price: "400.00" as Currency {format: "###.00"},  
    planeType: "BOING 737",  
    departureDate: "Oct 20, 2015" as String {format: "MMM dd, yyyy"}  
  } as Flight {class: "com.mulesoft.training.Flight"},
```

4. For planeType, use the replace function to replace the string Boing with Boeing.

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

5. Look at the preview; Boeing should now be spelled correctly.



The screenshot shows the DataWeave preview pane with the following code and output, demonstrating the replacement of 'Boing' with 'Boeing':

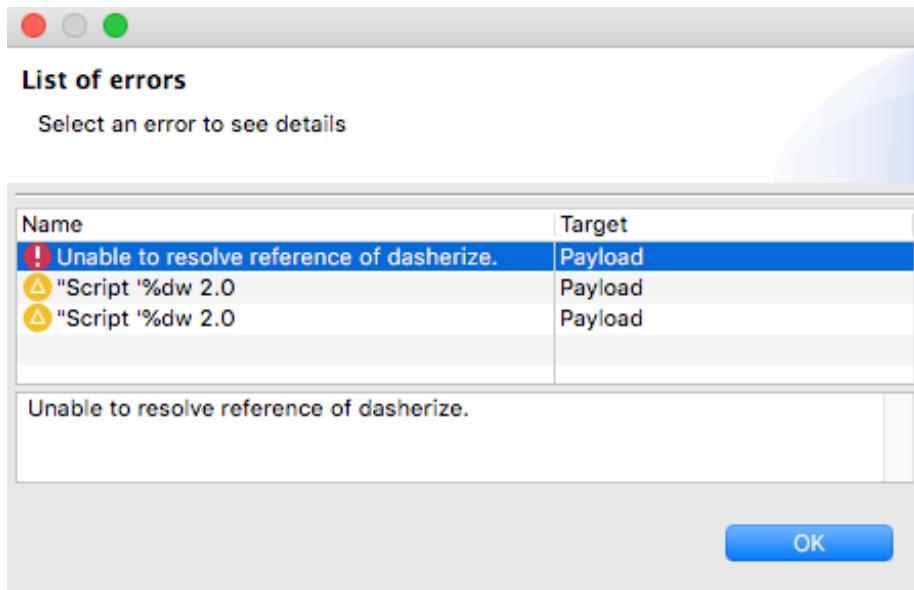
```
[  
  {  
    destination: "SFO",  
    price: "400.00" as Currency {format: "###.00"},  
    planeType: "BOEING 737",  
    departureDate: "Oct 20, 2015" as String {format: "MMM dd, yyyy"}  
  } as Flight {class: "com.mulesoft.training.Flight"},
```

Use the dasherize function in the Strings module

6. For planeType, replace the upper function with the dasherize function.

```
planeType: dasherize(replace(object.planeType,/(Boing)/) with  
"Boeing"),
```

7. Look at the error you get.



8. In the script header, add an import statement to import dasherize from the Strings module.

```
import dasherize from dw::core::Strings  
  
1 ⚡ %dw 2.0  
2   output application/dw  
3   import dasherize from dw::core::Strings  
4   type Currency = String {format: "###.00"}  
5   type Flight = Object {class: "com.mulesoft.training.Flight"}
```

9. Look at the preview.

```
[  
  {  
    destination: "SFO",  
    price: "400.00" as Currency {format: "###.00"},  
    planeType: "boeing-737",  
    departureDate: "Oct 20, 2015" as String {format: "MMM dd, yyyy"}  
  } as Flight {class: "com.mulesoft.training.Flight"},
```

Use the orderBy function

10. In the preview section, look at the flight prices; the flights are not ordered by price.
11. Use the orderBy function to order the flights object by price.

```
flights orderBy $.price
```

12. Look at the preview; the flights should now be ordered by price.

The screenshot shows the DataWeave editor with the following code:

```
13    300
14 /**
15@ fun getNumSeats(planeType: String) =
16@   if (planeType contains('737'))
17     150
18   else
19     300
20 ---
21@ using (flights =
22@   payload..*return map (object,index) -> {
23@     destination: object.destination,
24@     price: object.price as Number as Currency,
25@     // totalSeats: getNumSeats(object.planeType as String),
26@     planeType: dasherize(replace(object.planeType,/(Boin
27@     departureDate: object.departureDate as Date {format:
28@       as String {format: "MMM dd, yyyy"}}
29@     } as Flight
30 )
31
32 flights orderBy $.price
```

The preview pane shows three flight objects:

```
[{"destination": "LAX", "price": "199.99" as Currency {format: "###.00"}, "planeType": "boeing-737", "departureDate": "Oct 21, 2015" as String {format: "MMM dd, yyyy"}}, {"destination": "PDX", "price": "283.00" as Currency {format: "###.00"}, "planeType": "boeing-777", "departureDate": "Oct 21, 2015" as String {format: "MMM dd, yyyy"}}, {"destination": "PDX", "price": "283.00" as Currency {format: "###.00"}, "planeType": "boeing-777", "departureDate": "Oct 20, 2015" as String {format: "MMM dd, yyyy"}}]
```

13. Look at the three \$283 flights; there is a duplicate and they are not ordered by date.
14. Change the DataWeave expression to sort flights by price and then by date.

```
flights      orderBy $.departureDate
              orderBy $.price
```

15. Look at the preview; the flights should now be ordered by price and flights of the same price should be sorted by date.

The screenshot shows the DataWeave editor with the following code:

```
14 /**
15@ fun getNumSeats(planeType: String) =
16@   if (planeType contains('737'))
17     150
18   else
19     300
20 ---
21@ using (flights =
22@   payload..*return map (object,index) -> {
23@     destination: object.destination,
24@     price: object.price as Number as Currency,
25@     // totalSeats: getNumSeats(object.planeType as String),
26@     planeType: dasherize(replace(object.planeType,/(Boin
27@     departureDate: object.departureDate as Date {format:
28@       as String {format: "MMM dd, yyyy"}}
29@     } as Flight
30 )
31
32 flights orderBy $.departureDate
            |orderBy $.price
```

The preview pane shows four flight objects:

```
[{"destination": "PDX", "price": "283.00" as Currency {format: "###.00"}, "planeType": "boeing-777", "departureDate": "Oct 20, 2015" as String {format: "MMM dd, yyyy"}}, {"destination": "PDX", "price": "283.00" as Currency {format: "###.00"}, "planeType": "boeing-777", "departureDate": "Oct 21, 2015" as String {format: "MMM dd, yyyy"}}, {"destination": "SFO", "price": "400.00" as Currency {format: "###.00"}}]
```

Remove duplicate data

16. Use the distinctBy function to first remove any duplicate objects.

```
flights    distinctBy $  
          orderBy $.departureDate  
          orderBy $.price
```

17. Look at the preview; you should now see only two \$283 flights to PDX instead of three.

Output Payload ▾ ↻ 🔍 🖊️ 🗑️

Preview

```
15@ fun getNumSeats(planeType: String) =  
16@   if (planeType contains('737'))  
17@     150  
18@   else  
19@     300  
20 ---  
21@ using (flights =  
22@   payload..*return map (object,index) -> {  
23@     destination: object.destination,  
24@     price: object.price as Number as Currency,  
25@     // totalSeats: getNumSeats(object.planeType as String),  
26@     planeType: dasherize(replace(object.planeType,/(Boin  
27@     departureDate: object.departureDate as Date {format:  
28@       as String {format: "MMM dd, yyyy"}  
29@     } as Flight  
30  )  
31  
32@   flights distinctBy $  
33@     orderBy $.departureDate  
34@     orderBy $.price
```

price: "199.99" as Currency {format: "##,.##"},
planeType: "boeing-737",
departureDate: "Oct 21, 2015" as String {format: "MMM dd, yyyy"}
} as Flight {class: "com.mulesoft.training.Flight"},
{
destination: "PDX",
price: "283.00" as Currency {format: "##,.##"},
planeType: "boeing-777",
departureDate: "Oct 20, 2015" as String {format: "MMM dd, yyyy"}
} as Flight {class: "com.mulesoft.training.Flight"},
{
destination: "PDX",
price: "283.00" as Currency {format: "##,.##"},
planeType: "boeing-777",
departureDate: "Oct 21, 2015" as String {format: "MMM dd, yyyy"}
} as Flight {class: "com.mulesoft.training.Flight"},
{
destination: "SFO",
price: "400.00" as Currency {format: "##,.##"},
planeType: "boeing-737",

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

```
availableSeats: object.emptySeats as Number
```

19. Look at the preview; you should get three \$283 flights to PDX again.

Output Payload ▾ ↻ 🔍 🖊️ 🗑️

Preview

```
15@ fun getNumSeats(planeType: String) =  
16@   if (planeType contains('737'))  
17@     150  
18@   else  
19@     300  
20 ---  
21@ using (flights =  
22@   payload..*return map (object,index) -> {  
23@     destination: object.destination,  
24@     price: object.price as Number as Currency,  
25@     // totalSeats: getNumSeats(object.planeType as String),  
26@     planeType: dasherize(replace(object.planeType,/(Boin  
27@     departureDate: object.departureDate as Date {format:  
28@       as String {format: "MMM dd, yyyy"}  
29@     availableSeats: object.emptySeats as Number  
30  } as Flight  
31  )  
32  
33@   flights distinctBy $  
34@     orderBy $.departureDate
```

} as Flight {class: "com.mulesoft.training.Flight"},
{
destination: "PDX",
price: "283.00" as Currency {format: "##,.##"},
planeType: "boeing-777",
departureDate: "Oct 20, 2015" as String {format: "MMM dd, yyyy"},
availableSeats: 0
} as Flight {class: "com.mulesoft.training.Flight"},
{
destination: "PDX",
price: "283.00" as Currency {format: "##,.##"},
planeType: "boeing-777",
departureDate: "Oct 20, 2015" as String {format: "MMM dd, yyyy"},
availableSeats: 23
} as Flight {class: "com.mulesoft.training.Flight"},
{
destination: "PDX",
price: "283.00" as Currency {format: "##,.##"},
planeType: "boeing-777",
departureDate: "Oct 21, 2015" as String {format: "MMM dd, yyyy"},
availableSeats: 23
} as Flight {class: "com.mulesoft.training.Flight"},
{
destination: "PDX",
price: "283.00" as Currency {format: "##,.##"},
planeType: "boeing-777",
departureDate: "Oct 21, 2015" as String {format: "MMM dd, yyyy"},
availableSeats: 23
} as Flight {class: "com.mulesoft.training.Flight"},
{
destination: "SFO",
price: "400.00" as Currency {format: "##,.##"},
planeType: "boeing-737",

Use the filter function

20. In the preview, look at the values of the availableSeats properties; you should see one is equal to zero.

21. Use the filter function to remove any objects that have availableSeats equal to 0.

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

22. Look at the preview; you should no longer get the flight that had no available seats.

Output Payload ▾ ↻ Preview

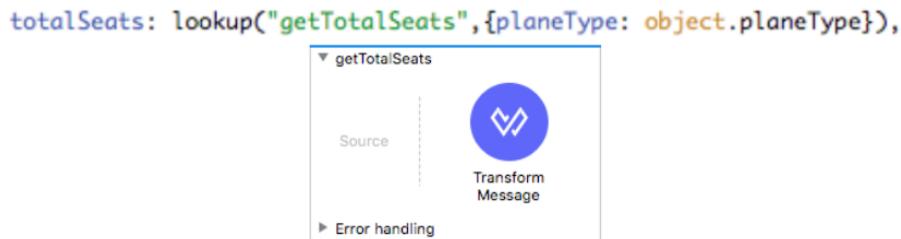
```
17      150  
18  else  
19      300  
20 ---  
21@using (Flights =  
22@  payload.*return map (object,index) -> {  
23@    destination: object.destination,  
24@    price: object.price as Number as Currency,  
25@    // totalSeats: getNumSeats(object.planeType as String),  
26@    planeType: dasherize(replace(object.planeType,/(Boin  
27@    departureDate: object.departureDate as Date {format:  
28@      as String {format: "MMM dd, yyyy"},  
29@      availableSeats: object.emptySeats as Number  
30@    } as Flight  
31 )  
32  
33@flights distinctBy $  
34  filter ($.availableSeats !=0)  
35  orderBy $.departureDate  
36  orderBy $.price
```

```
price: "199.99" as Currency {format: "###.00"},  
planeType: "boeing-737",  
departureDate: "Oct 21, 2015" as String {format: "MMM dd, yyyy"},  
availableSeats: 10  
} as Flight {class: "com.mulesoft.training.Flight"},  
{  
  destination: "PDX",  
  price: "283.00" as Currency {format: "###.00"},  
  planeType: "boeing-777",  
  departureDate: "Oct 20, 2015" as String {format: "MMM dd, yyyy"},  
  availableSeats: 23  
} as Flight {class: "com.mulesoft.training.Flight"},  
{  
  destination: "PDX",  
  price: "283.00" as Currency {format: "###.00"},  
  planeType: "boeing-777",  
  departureDate: "Oct 21, 2015" as String {format: "MMM dd, yyyy"},  
  availableSeats: 30  
} as Flight {class: "com.mulesoft.training.Flight"},  
{
```

Walkthrough 11-9: Look up data by calling a flow

In this walkthrough, you continue to work with the transformation in `getMultipleFlights`. You will:

- Call a flow from a DataWeave expression.

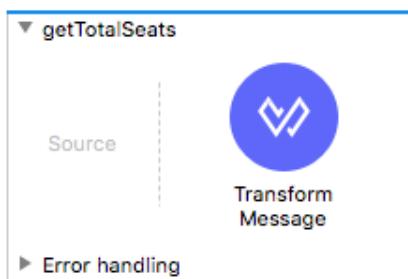


Create a lookup flow

1. Return to the Transform Message properties view for the transformation in `postMultipleFlights`.
2. Copy the `getNumSeats` function.

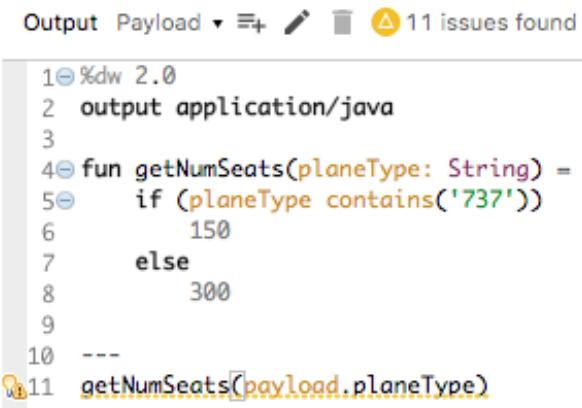
```
15① fun getNumSeats (planeType: String) =  
16②     if (planeType contains('737'))  
17         150  
18     else  
19         300
```

3. Drag a Transform Message component from the Mule Palette and drop it at the bottom of the canvas to create a new flow.
4. Change the name of the flow to `getTotalSeats`.



5. In the Transform Message properties view of the component in `getTotalSeats`, paste the function you copied into the script header.

6. In the script body, call the function, passing to it the payload's planeType property.
- ```
getNumSeats(payload.planeType)
```

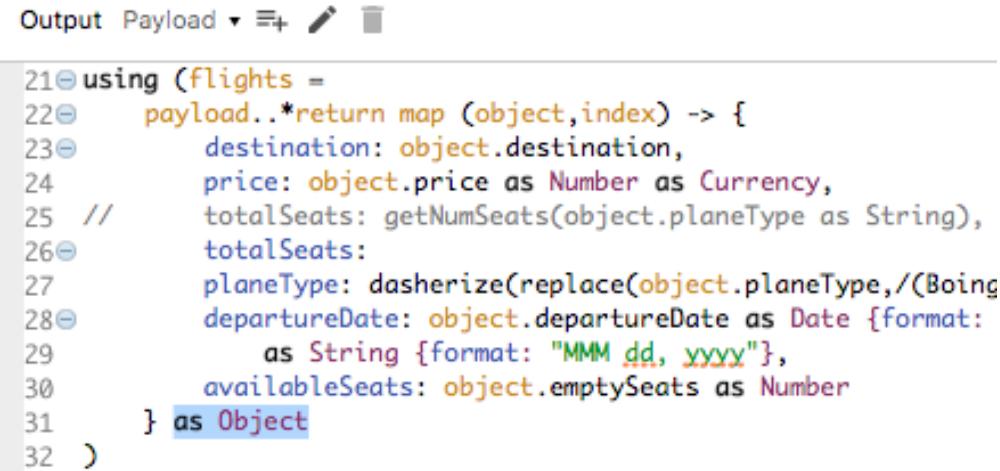


```
Output Payload ▾ + 🖊 🗑 11 issues found

1 ⊕ %dw 2.0
2 output application/java
3
4 ⊕ fun getNumSeats(planeType: String) =
5 if (planeType contains('737'))
6 150
7 else
8 300
9
10 ---
11 getNumSeats(payload.planeType)
```

## Call a flow from a DataWeave expression

7. Return to the Transform Message properties view for the transformation in postMultipleFlights.
8. Add a property called totalSeats inside the expression (keep the other one commented out).
9. Change the return type of the map function from Flight to Object.



```
Output Payload ▾ + 🖊 🗑

21 ⊕ using (flights =
22 ⊕ payload..*return map (object,index) -> {
23 ⊕ destination: object.destination,
24 price: object.price as Number as Currency,
25 // totalSeats: getNumSeats(object.planeType as String),
26 ⊕ totalSeats:
27 planeType: dasherize(replace(object.planeType,/(Boing
28 ⊕ departureDate: object.departureDate as Date {format:
29 as String {format: "MMM dd, yyyy"}},
30 availableSeats: object.emptySeats as Number
31 } as Object
32)
```

10. Set totalSeats equal to the return value from the DataWeave lookup() function.

```
totalSeats: lookup()
```

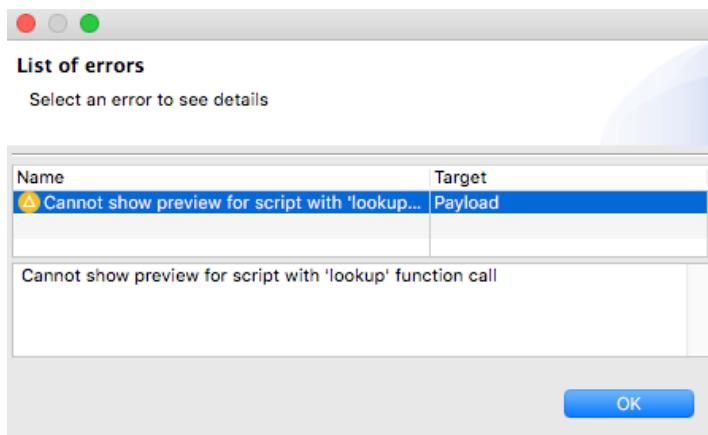
11. Pass to lookup() an argument that is equal to the name of the flow to call: "getTotalSeats".
12. Pass an object to lookup() as the second argument.

13. Give the object a field called planeType (to match what the flow is expecting) and set it equal to the value of the planeType field.

```
totalSeats: lookup("getTotalSeats", {planeType: object.planeType})
```

14. Look at the preview.

15. Look at the warning you get.



## Test the application

16. Save the file and, if necessary, run or debug the project.
17. In Advanced REST Client, make sure the method is set to POST and the request URL is set to <http://localhost:8081/multipleflights>.
18. Set a Content-Type header to application/xml.
19. Set the request body to the value contained in the flights-example.xml file in the src/test/resources folder.

The screenshot shows the Advanced REST Client interface. At the top, it displays 'Method: POST' and 'Request URL: http://localhost:8081/multipleflights'. Below this, there are tabs for 'Headers', 'Body', 'Authorization', and 'Actions'. The 'Headers' tab shows a dropdown menu with 'Body content type' selected and 'application/xml' chosen. The 'Body' tab is active, showing the XML content of the 'flights-example.xml' file:

```
<ns2:listAllFlightsResponse xmlns:ns2="http://soap.training.mulesoft.com/">
 <return airlineName="United">
 <code>A1B2C3</code><departureDate>2015/10/20</departureDate>
 <destination>SFO</destination><emptySeats>40</emptySeats>
 <origin>MUA</origin>
 <planeType>Boing 737</planeType>
 <price>400.0</price>
 </return>
 <return airlineName="Delta">
 <code>A1B2C4</code>
 <departureDate>2015/10/21</departureDate><destination>LAX</destination><emptySeats>40</emptySeats>
 <origin>MUA</origin>
 <planeType>Boing 737</planeType>
 <price>199.99</price>
 </return>
</ns2:listAllFlightsResponse>
```

20. Send the request; you should get the DataWeave representation of the return flight data and each flight should have a totalSeats property equal to 150 or 300.

200 OK 5214.81 ms

DETAILS ▾

[COPY](#) [SAVE](#) [SOURCE VIEW](#)

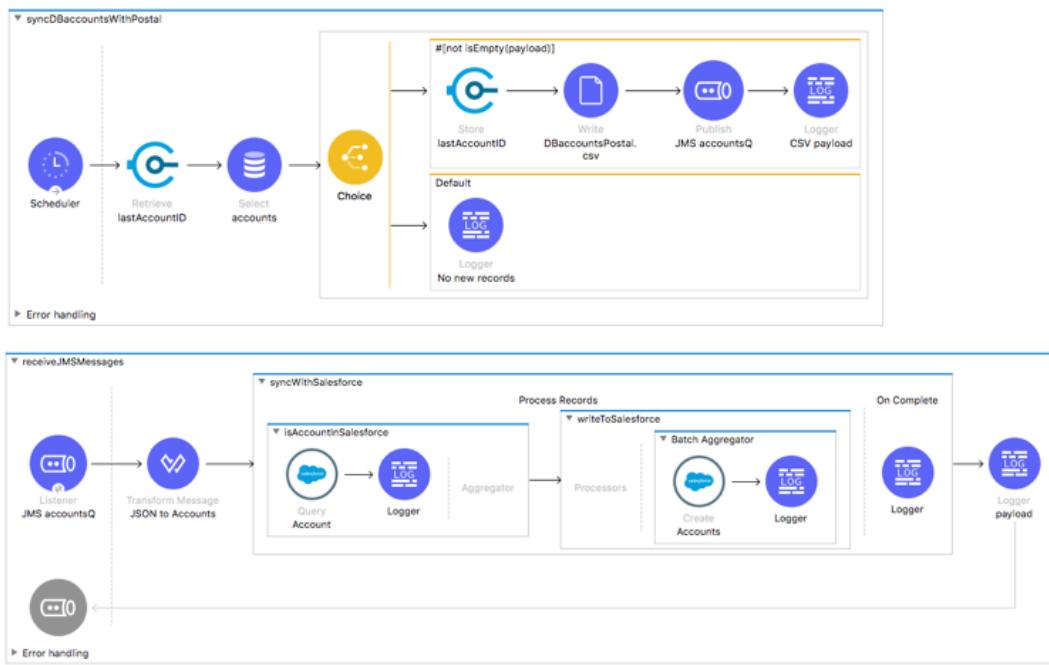
```
[
 {
 destination: "LAX",
 price: "199.99" as String {format: "###.00"},
 totalSeats: 150 as Number {encoding: "UTF-8", mediaType: "application/java;
charset=UTF-8", mimeType: "application/java", class: "java.lang.Integer"},
 planeType: "boeing-737",
 departureDate: "Oct 21, 2015" as String {format: "MMM dd, yyyy"},
 availableSeats: 10
 } as Object {class: "com.mulesoft.training.Flight"},
 {
 destination: "PDX",
 price: "283.00" as String {format: "###.00"},
 totalSeats: 300 as Number {encoding: "UTF-8", mediaType: "application/java;
charset=UTF-8", mimeType: "application/java", class: "java.lang.Integer"},
 planeType: "boeing-777",
 departureDate: "Oct 20, 2015" as String {format: "MMM dd, yyyy"},
 availableSeats: 23
 } as Object {class: "com.mulesoft.training.Flight"},
 {
 destination: "PDX",
 price: "283.00" as String {format: "###.00"},
 totalSeats: 300 as Number {encoding: "UTF-8", mediaType: "application/java;
charset=UTF-8", mimeType: "application/java", class: "java.lang.Integer"},
 planeType: "boeing-777"
```

21. Return to Anypoint Studio.

22. Stop the project.

23. Close the project.

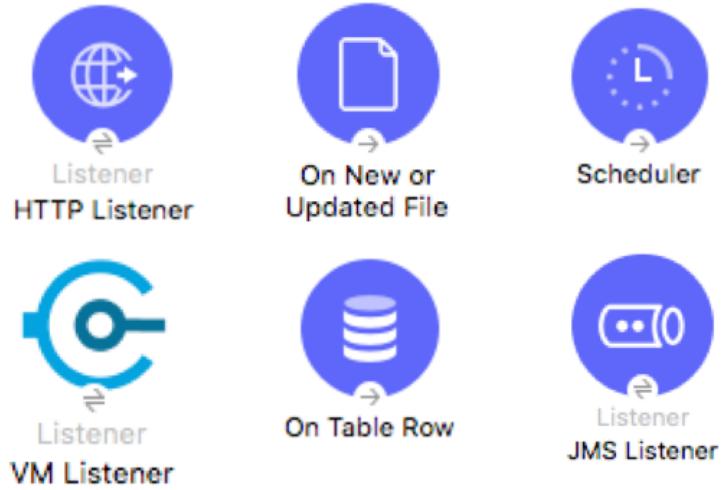
# PART 3: Building Applications to Synchronize Data



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

- Trigger flows when files or database records are added or updated.
- Schedule flows.
- Persist and share data across flow executions.
- Publish and consume JMS messages.
- Process items in a collection sequentially.
- Process records asynchronously in batches.

# Module 12: Triggering Flows



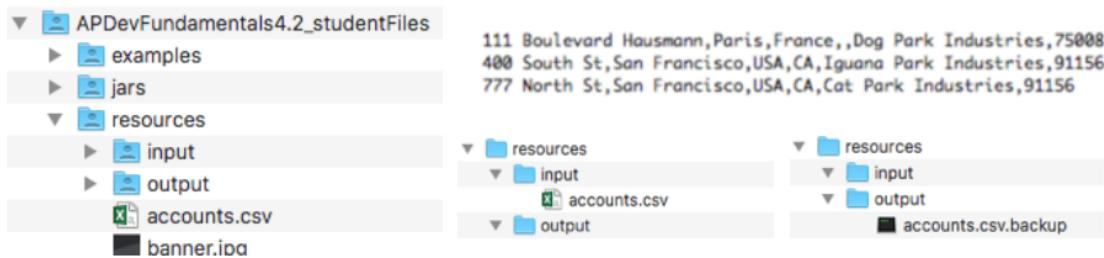
At the end of this module, you should be able to:

- Read and write files.
- Trigger flows when files are added, created, or updated.
- Trigger flows when new records are added to a database table.
- Schedule flows to run at a certain time or frequency.
- Persist and share data in flows using the Object Store.
- Publish and consume JMS messages.

## Walkthrough 12-1: Trigger a flow when a new file is added to a directory

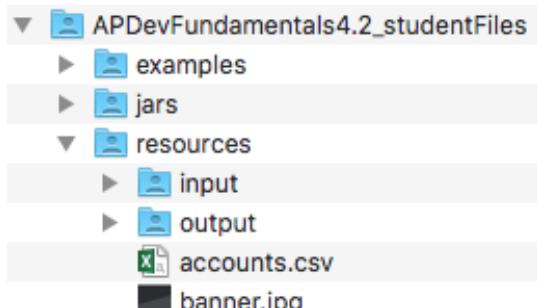
In this walkthrough, you load data from a local CSV file when a new file is added to a directory. You will:

- Add and configure a File listener to watch an input directory.
- Restrict the type of file read.
- Rename and move the processed files.



### Locate files and folders

1. In your computer's file browser, return to the student files for the course.
2. Open the resources folder and locate the accounts.csv file and the input and output folders.

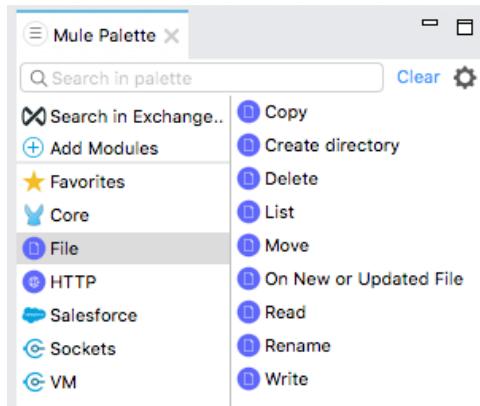


3. Leave this folder open.

### Add the File module to the project

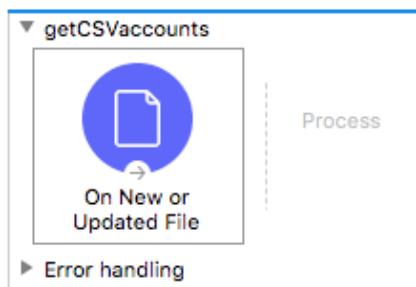
4. Return to Anypoint Studio and open the apdev-examples project.
5. Open accounts.xml.
6. In the Mule Palette, select Add Modules.
7. Select the File connector in the right side of the Mule Palette and drag and drop it into the left side.

8. If you get a Select module version dialog box, select the latest version and click Add.



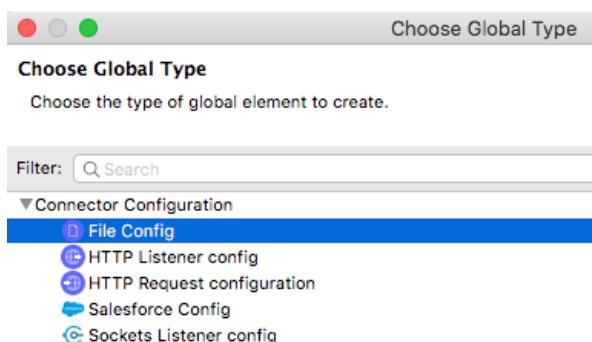
## Create a flow that monitors a location for new files

9. Locate the On New or Updated File operation for the File connector in the right side of the Mule Palette and drag and drop it at the top of the canvas to create a new flow.
10. Rename the flow to getCSVaccounts.

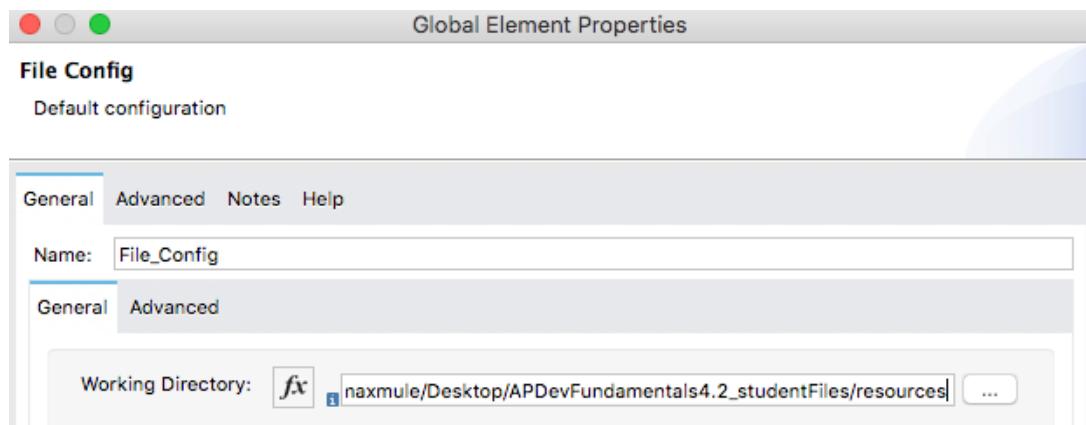


## Configure the File connector

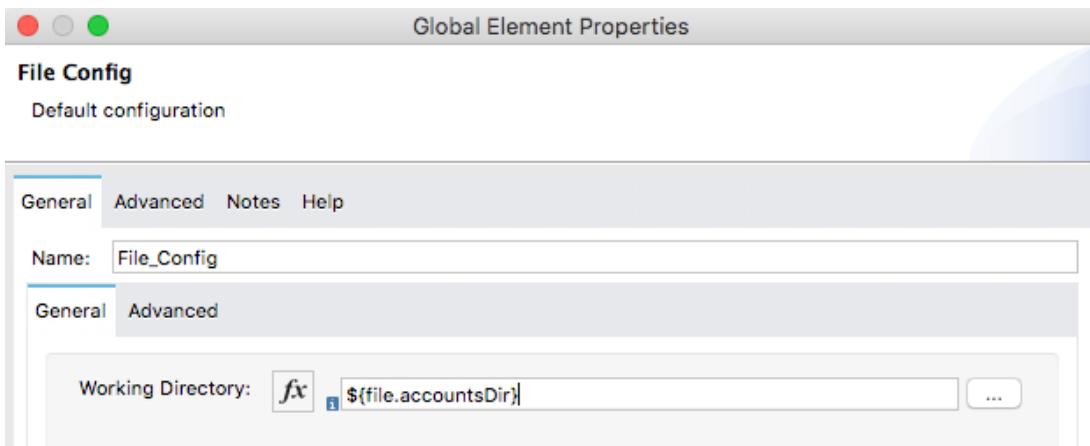
11. Open global.xml and switch to the Global Elements view.
12. Click Create.
13. In the Choose Global Type dialog box, select Connector Configuration > File Config and click OK.



14. In the Global Element Properties dialog box, click the browse button next to working directory.
15. Browse to the student files folder for the course.
16. Select the resources folder and click Open.



17. Select and cut the value of the working directory that got populated and replace it with a property \${file.accountsDir}.



18. In the Global Element Properties dialog box, click OK.
19. Open config.yaml in src/main/resources.
20. Create a new property file.accountsDir and set it equal to the value you cut.

```

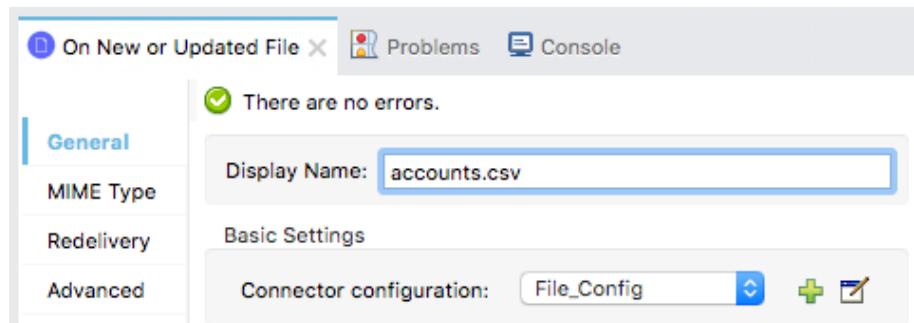
*accounts *global *config.yaml X
8
9 file:
10 accountsDir: "/Users/maxmule/Desktop/APDevFundamentals4.2_studentFiles/resources"

```

*Note: Users on all OSs, including Windows, should use forward slash characters as shown.*

## Configure the On New or Updated File operation

21. Return to accounts.xml.
22. In the On New or Updated File properties view, change the display name to accounts.csv.
23. Set the connector configuration to the existing File\_Config.

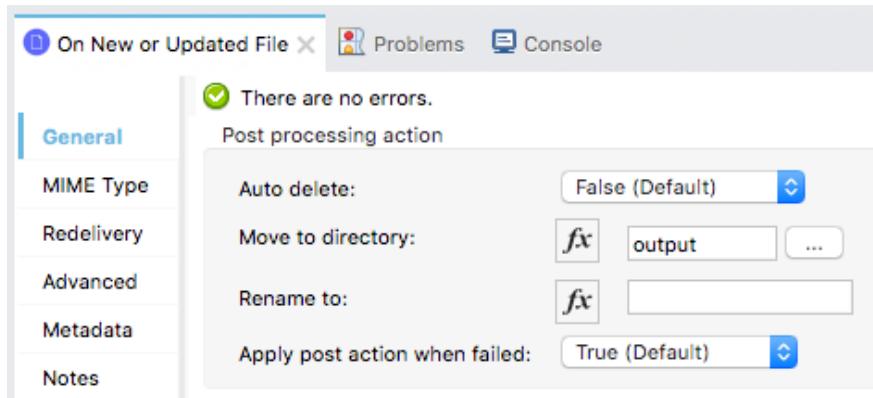


24. In the General section, set the directory to input.
25. Set the matcher to Edit inline.
26. Set filename pattern to \*.csv.

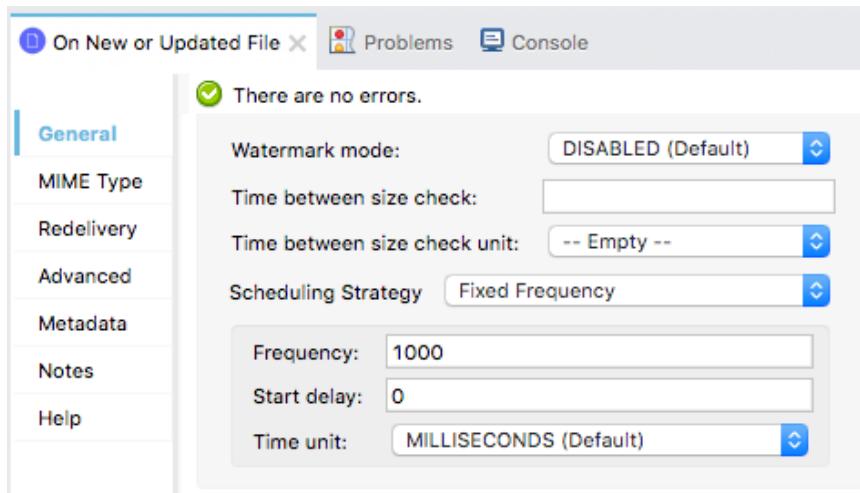
The screenshot shows the 'General' section of the 'On New or Updated File' properties view. It includes the following settings:

- Directory: input
- Matcher: Edit inline
- Created since:
- Created until:
- Updated since:
- Updated until:
- Accessed since:
- Accessed until:
- Not updated in the last:  10000
- Updated in the last:  10000
- Time unit:  MILLISECONDS (Default)
- Filename pattern:  \*.csv

27. In the Post processing action section, set the move to directory to output.



28. In the General section, review the default scheduling information; the endpoint checks for new files every 1000 milliseconds.



## Review event metadata in the DataSense Explorer

29. Select the Output tab in the DataSense Explorer; you should see no metadata about the structure of the CSV file.

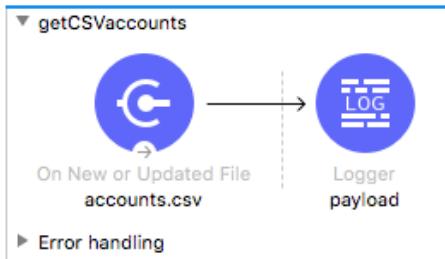
The screenshot shows the DataSense Explorer interface. The 'Output' tab is selected. The tree view shows a 'Mule Message' node expanded, revealing 'Payload' and 'Attributes' sub-nodes. The 'Payload' node contains an 'Any : Any' entry. The 'Attributes' node contains a 'LocalFileAttributes : Object' entry with two properties: 'lastModifiedTime : DateTime' and 'lastAccessTime : DateTime'.

## Display the file contents

30. Add a Logger to the flow.

31. In the Logger properties view, set the display name to payload and the message to display the payload.

```
[payload]
```



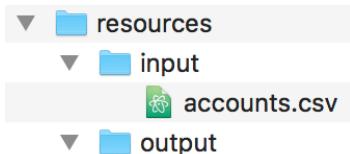
## Debug the application

32. Add a breakpoint to the Logger.

33. Save the files and debug the project.

34. In your computer's file browser, return to the resources folder in student files for the course.

35. Move the accounts.csv file to the input folder.



36. Return to the Mule Debugger and look at the payload.

The screenshot shows the Mule Debugger interface. On the left, a tree view displays the payload content:

- Logger = payload
- attributes = {LocalFileAttributes} LocalFileAttributes[lastModifiedTime=2019-06-10T11:24:45Z, mediaType=text/csv]
- correlationId = "3106c5f0-8b9b-11e9-99fe-f018983d93"
- payload = Billing Street,Billing City,Billing Country,Billing St
- vars = {Map} size = 0

To the right, the payload content is shown as a multi-line string:

```
BillingStreet,BillingCity,BillingCountry,BillingState,Name,Billing
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
```

Below the debugger, a flow diagram titled "getCSVAccounts" is visible. It shows a file icon labeled "On New or Updated File accounts.csv" connected to a logger icon labeled "Logger payload". A "Error handling" section is also present.

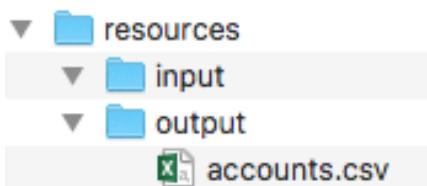
37. Expand Attributes and locate the fileName and path attributes.

The screenshot shows the Mule Debugger interface with the attributes expanded. The "fileName" attribute is highlighted with a blue selection bar:

- attributes = {LocalFileAttributes} LocalFileAttributes[lastModifiedTime=2019-06-10T11:24:45Z, mediaType=text/csv]
- creationTime = {LocalDateTime} 2019-06-10T10:24:45
- directory = false
- fileName = "accounts.csv"**
- lastAccessTime = {LocalDateTime} 2019-06-10T12:17:06
- lastModifiedTime = {LocalDateTime} 2019-06-10T10:28:34
- path = "/Users/maxmule/Desktop/APDevFundamentals4.2\_studentFiles/resources/

38. Step to the end of the application.

39. In your computer's file browser, return to the student files for the course; you should see accounts.csv has been moved to the output folder.



40. Return to Anypoint Studio and look at the console; you should see the file contents displayed.

```
INFO 2019-11-15 14:08:07,054 [[MuleRuntime].cpuLight.23: [apdev-examples].getCSVaccounts.CPU_LITE @672675c6] [event: 41b51480-07db-11ea-810e-f018983d9329] org.mule.runtime.core.internal.processor.LoggerMessageProcessor: BillingStreet,BillingCity,BillingCountry,BillingState,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
```

## Rename the file

41. Return to the accounts.csv properties view and in the Post processing action section, set the Rename to property to an expression that appends .backup to the original filename.

```
#[attributes.fileName ++ ".backup"]
```

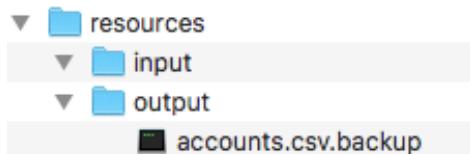


## Test the application

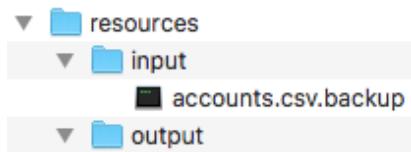
42. Save the file to redeploy the project.

43. Remove the breakpoint from the Logger.

44. In your computer's file explorer, 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.



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



46. Return to Anypoint Studio and switch perspectives.

47. Stop the project.

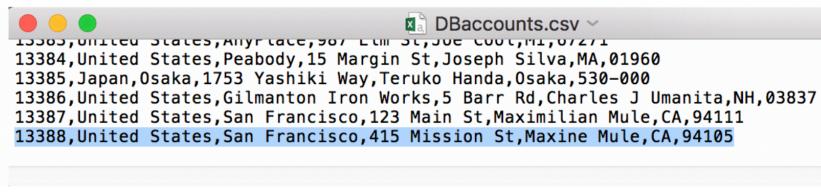
## Walkthrough 12-2: Trigger a flow when a new record is added to a database and use automatic watermarking

In this walkthrough, you work with the accounts table in the training database. You will:

- Add and configure a Database listener to check a table on a set frequency for new records.
- Use the listener's automatic watermarking to track the ID of the latest record retrieved and trigger the flow whenever a new record is added.
- Output new records to a CSV file.
- Use a form to add a new account to the table and see the CSV file updated.

### MUA Accounts

| accountID | name            | street         | city          |
|-----------|-----------------|----------------|---------------|
| 13388     | Maxine Mule     | 415 Mission St | San Francisco |
| 13387     | Maximilian Mule | 123 Main St    | San Francisco |



### View accounts data

1. Return to the course snippets.txt file and copy the text for the MUA Accounts URL.
2. In a web browser, navigate to the MUA accounts by pasting the URL.
3. Review the data and the names of the columns—which match those of a database table.

### MUA Accounts

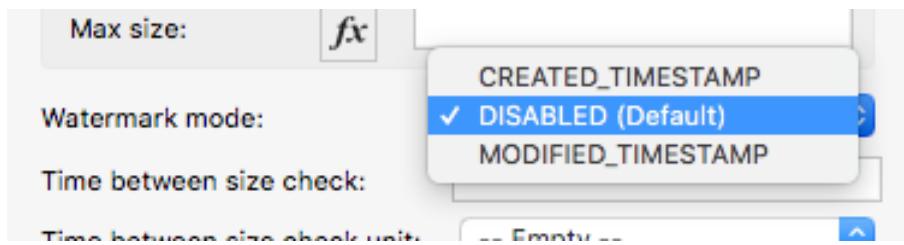
[Create More Accounts](#)

| accountID | name         | street       | city          | state | postal     | country       |
|-----------|--------------|--------------|---------------|-------|------------|---------------|
| 14589     | Maxine Mule  | 123 Main St  | San Francisco | CA    | 94111      | United States |
| 14588     | Joseph Silva | 15 Margin St | Peabody       | MA    | 99999-0107 | United States |

### Look at the File listener settings for watermarking

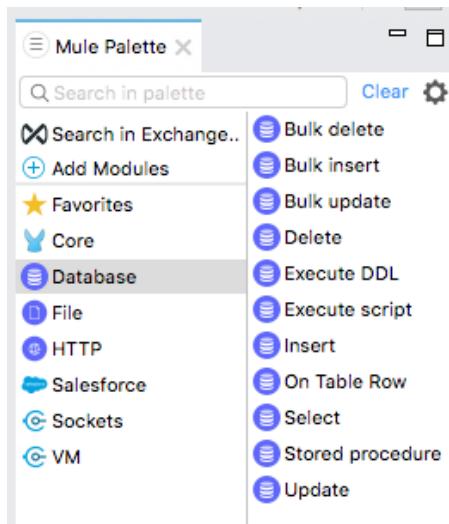
4. Return to accounts.xml in Anypoint Studio.
5. Return to the accounts.csv properties view for the listener in getCSVaccounts.

6. Locate the watermark mode setting in the General section and look at its possible values.



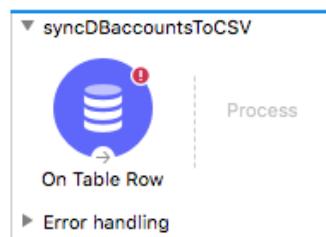
## Add the Database module to the project

7. In the Mule Palette, select Add Modules.
8. Select the Database connector in the right side of the Mule Palette and drag and drop it into the left side.
9. If you get a Select module version dialog box, select the latest version and click Add.



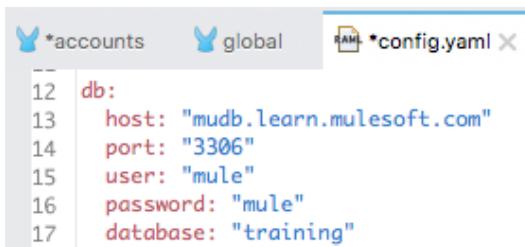
## Create a flow to monitor a database

10. Locate the On Table Row operation for the Database connector in the right side of the Mule Palette and drag and drop it at the top of the canvas to create a new flow.
11. Rename the flow to syncDBaccountsToCSV.



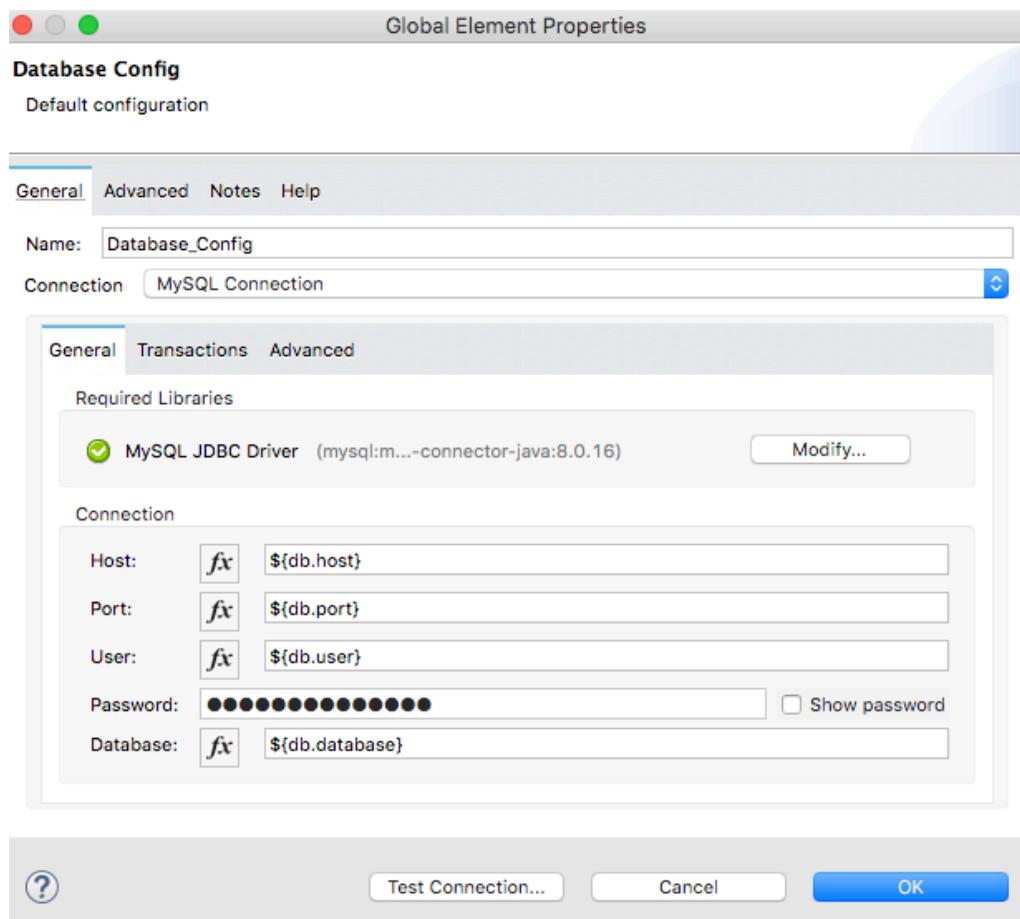
## Configure a Database connector

12. Return to the course snippets.txt file.
13. Locate and copy the MySQL (or Derby) database properties in the Module 4 section.
14. Return to config.yaml and paste the properties.

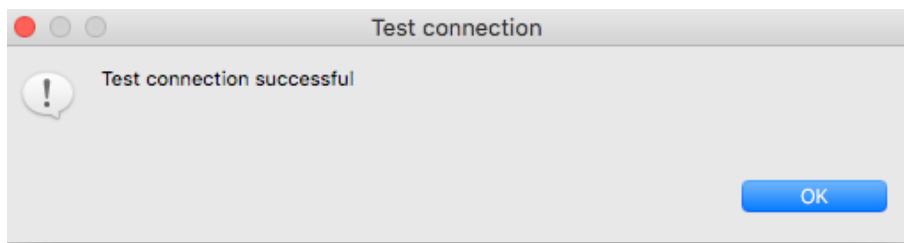


```
accounts global config.yaml
-- db:
12 host: "mudb.learn.mulesoft.com"
13 port: "3306"
14 user: "mule"
15 password: "mule"
16 database: "training"
```

15. Save the file.
16. Return to global.xml and create a new Database Config that uses these properties.  
*Note: If necessary, refer to the detailed steps in Walkthrough 4-2.*
17. Add the MySQL (or Derby) JDBC driver.



18. Test the connection and make sure it is successful.



## Configure the Database listener

19. Return to accounts.xml.

20. In the On Table Row properties view, set the following values:

- Display Name: accounts
- Connector configuration: Database\_Config
- Table: accounts
- Watermark column: accountID
- ID column: accountID
- Frequency: 10
- Time unit: SECONDS

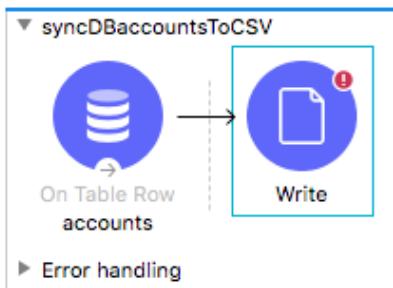
A screenshot of the MuleSoft Anypoint Studio interface. The left sidebar shows a tree structure with 'accounts' selected. The main panel shows the 'On Table Row' properties for the 'accounts' table. The 'General' tab is selected. The 'Basic Settings' section includes:

- Display Name: accounts
- Connector configuration: Database\_Config
- Table: accounts
- Watermark column: accountID
- Id column: accountID
- Scheduling Strategy: Fixed Frequency
- Frequency: 10
- Start delay: 0
- Time unit: SECONDS

The status bar at the top indicates 'There are no errors.'

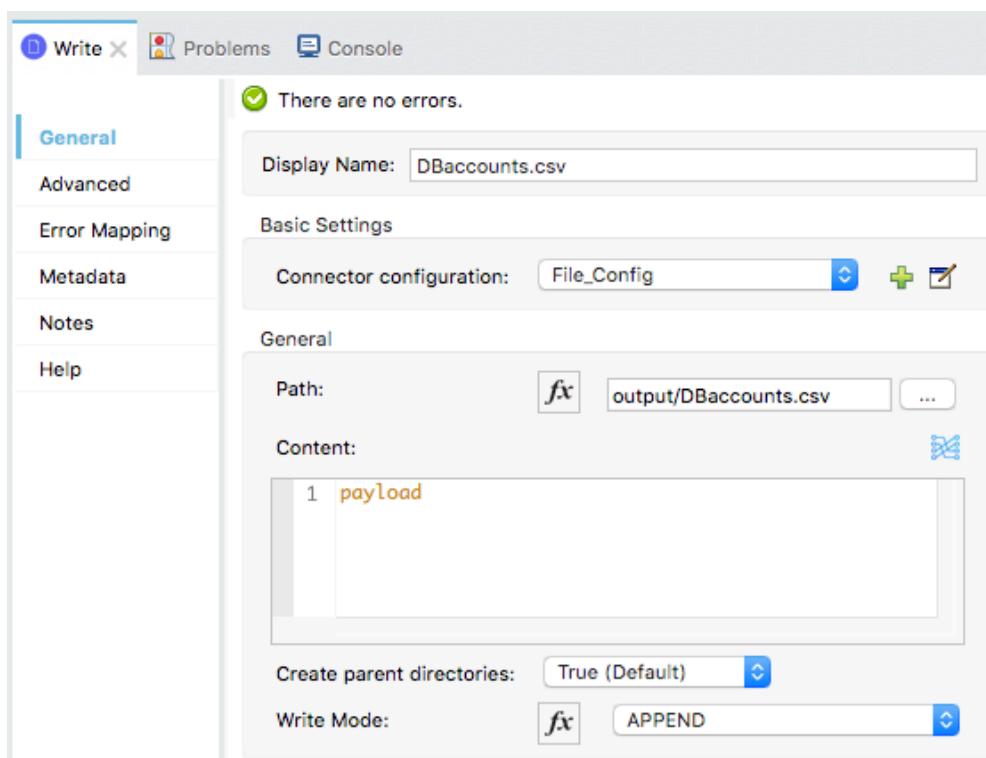
## Write new records to a CSV file

21. Add a File Write operation to the flow.



22. In the Write properties view, set the following values:

- Display Name: DBaccounts.csv
- Connector configuration: File\_Config
- Path: output/DBaccounts.csv
- Write mode: APPEND

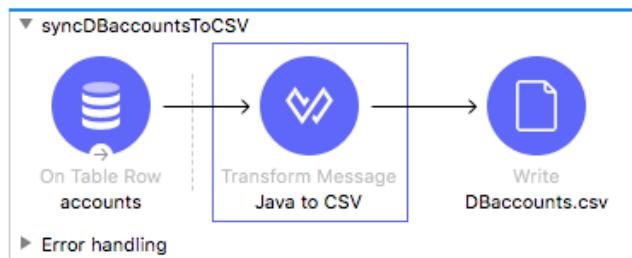


23. Expand Payload in the Input tab in the DataSense Explorer; you should see the operation will get an account with account information from the Database operation.

The screenshot shows the DataSense Explorer interface with the 'Input' tab selected. A search bar at the top contains the placeholder 'type filter text'. Below it, the 'Mule Message' structure is expanded, showing the 'Payload' section. The 'Payload' section is described as '(Actual) Object : Object' and contains fields: accountID (Number), name (String), street (String?), city (String?), country (String?), state (String?), and postal (String?). It also includes an '(Expected) Any : Any' section and an 'Attributes' section with a Void type. At the bottom, there is a 'Variables' section.

## Transform the records to CSV

24. Add a Transform Message component before the Write operation.  
25. Change the display name to Java to CSV.



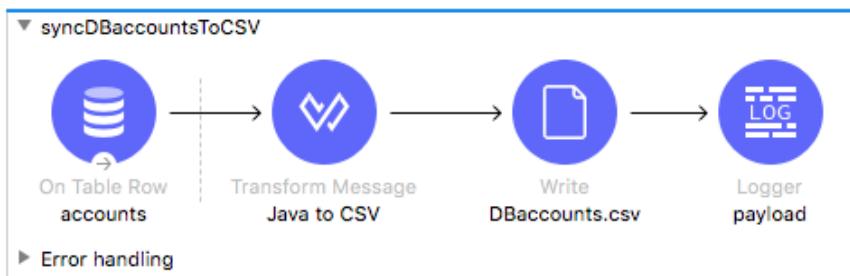
26. In the Transform Message properties view, change the output type to application/csv and add a header property equal to false.  
27. Set the body expression to [payload].

The screenshot shows the 'Properties' view for the 'Transform Message Java to CSV' component. At the top, there are tabs for 'Output', 'Payload' (which is currently selected), and 'Preview'. Below the tabs are icons for 'New', 'Edit', and 'Delete'. The 'Payload' tab displays the following code:

```
1 %dw 2.0
2 output application/csv header=false
3 ---
4 [payload]
```

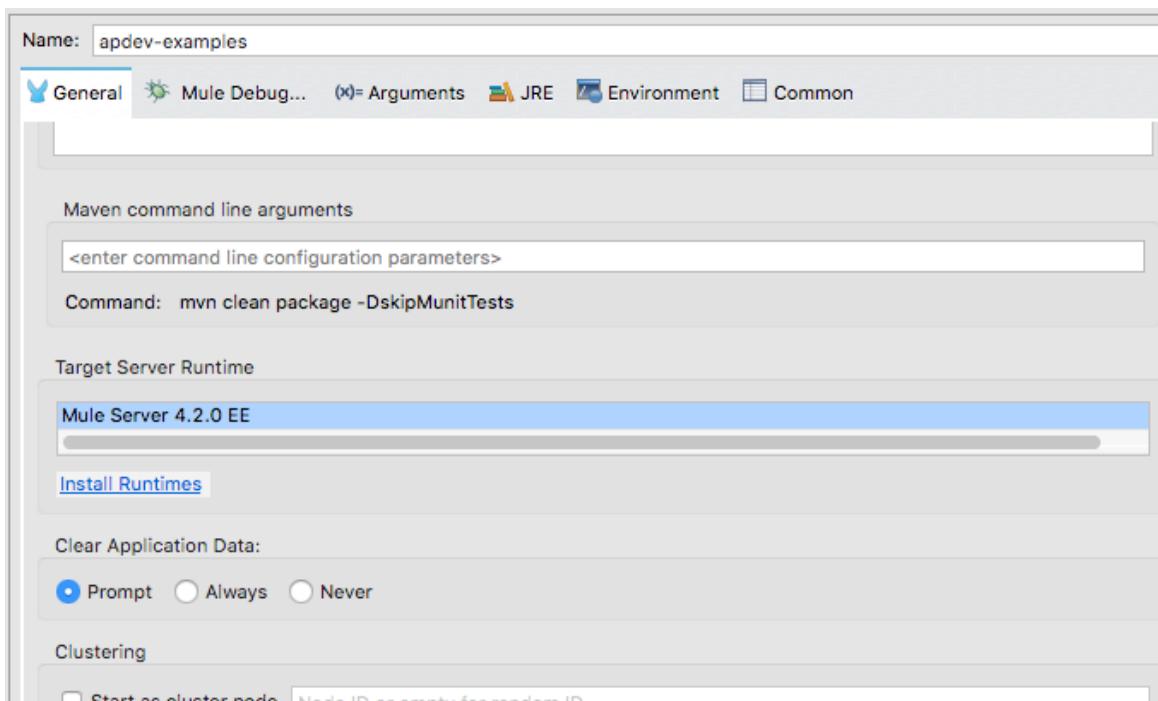
## Add a Logger to display the records

28. Add a Logger to the end of the flow.
29. Change the display name to payload.
30. Set the message to display the payload.



## Set the application to prompt to clear application data when it starts

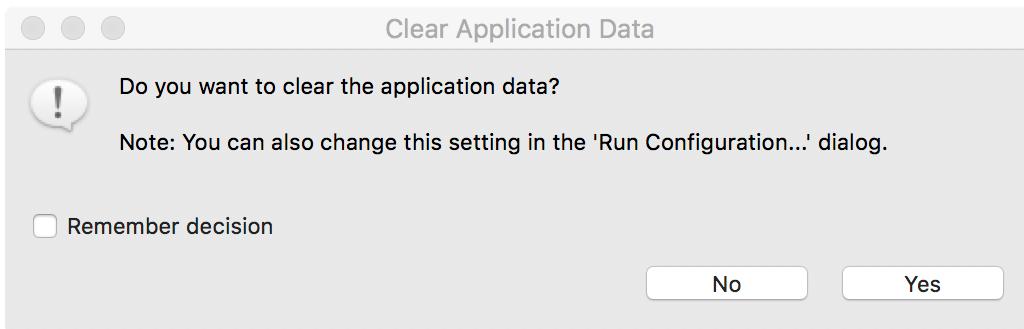
31. Add a breakpoint to the Transform Message component.
32. Save all files then, in the main menu, select Run > Debug Configurations.
33. In the Debug Configurations dialog box, locate the Clear Application Data section in the General tab and change it to Prompt.



*Note: You need to rerun or debug an application to get the prompt; you cannot just save to redeploy.*

## Debug the project

34. Click Debug.
35. In the Clear Application Data dialog box, click Yes.

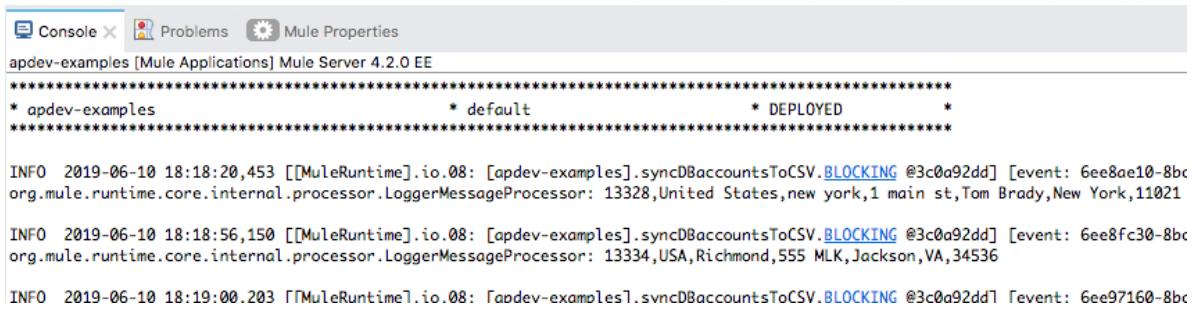


36. In the Mule Debugger, expand Payload.

The Mule Debugger interface. The top window shows the "payload" variable expanded, displaying a list of 7 entries (0-6) from an AbstractMap\$SimpleEntry, each containing accountID, country, city, street, name, state, and postal. An annotation indicates mediaType: application/java; charset=UTF-8. The bottom window shows the flow configuration for "syncDBaccountsToCSV". It consists of four components: "On Table Row accounts" (with "accounts" selected), "Transform Message Java to CSV" (highlighted with a red dot and a dashed box), "Write DBaccounts.csv", and "Logger payload". A "Error handling" section is also visible.

37. Click Resume and step through several of the records.

38. Look at the console; you should see the records displayed.



```
Console X Problems Mule Properties
apdev-examples [Mule Applications] Mule Server 4.2.0 EE

* apdev-examples * default * DEPLOYED *

INFO 2019-06-10 18:18:20,453 [[MuleRuntime].io.08: [apdev-examples].syncDBaccountsToCSV.BLOCKING @3c0a92dd] [event: 6ee8ae10-8bc
org.mule.runtime.core.internal.processor.LoggerMessageProcessor: 13328,United States,new york,1 main st,Tom Brady,New York,11021

INFO 2019-06-10 18:18:56,150 [[MuleRuntime].io.08: [apdev-examples].syncDBaccountsToCSV.BLOCKING @3c0a92dd] [event: 6ee8fc30-8bc
org.mule.runtime.core.internal.processor.LoggerMessageProcessor: 13334,USA,Richmond,555 MLK,Jackson,VA,34536

INFO 2019-06-10 18:19:00.203 [[MuleRuntime].io.08: [apdev-examples].syncDBaccountsToCSV.BLOCKING @3c0a92dd] [event: 6ee97160-8bc
```

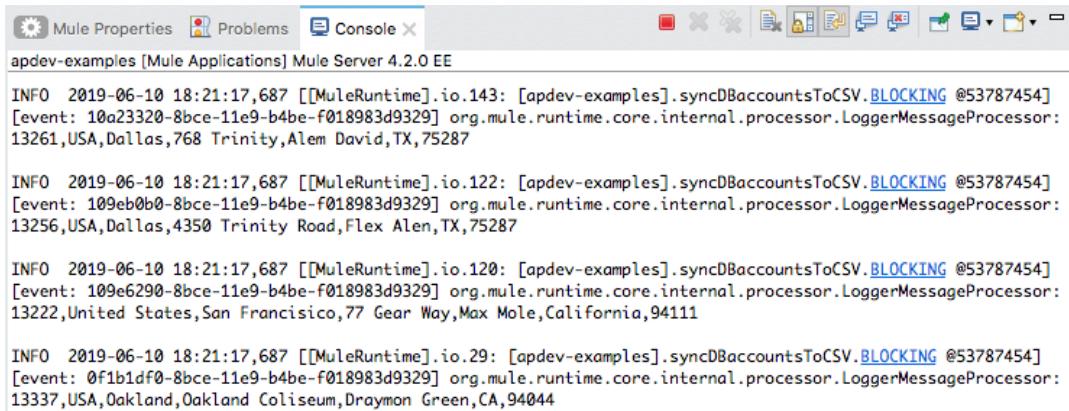
39. Stop the project and switch perspectives.

## Clear application data and test the application

40. Run the project.

41. In the Clear Application Data dialog box, click Yes.

42. Look at the console; you should see many records.



```
Mule Properties Problems Console
apdev-examples [Mule Applications] Mule Server 4.2.0 EE

INFO 2019-06-10 18:21:17,687 [[MuleRuntime].io.143: [apdev-examples].syncDBaccountsToCSV.BLOCKING @53787454]
[event: 10a23320-8bce-11e9-b4be-f018983d9329] org.mule.runtime.core.internal.processor.LoggerMessageProcessor:
13261,USA,Dallas,768 Trinity,Alem David,TX,75287

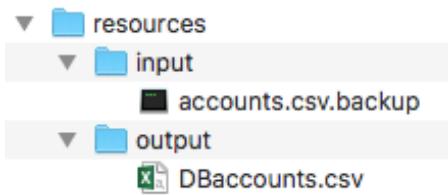
INFO 2019-06-10 18:21:17,687 [[MuleRuntime].io.122: [apdev-examples].syncDBaccountsToCSV.BLOCKING @53787454]
[event: 109eb0b0-8bce-11e9-b4be-f018983d9329] org.mule.runtime.core.internal.processor.LoggerMessageProcessor:
13256,USA,Dallas,4350 Trinity Road,Flex ALEN,TX,75287

INFO 2019-06-10 18:21:17,687 [[MuleRuntime].io.120: [apdev-examples].syncDBaccountsToCSV.BLOCKING @53787454]
[event: 109e6290-8bce-11e9-b4be-f018983d9329] org.mule.runtime.core.internal.processor.LoggerMessageProcessor:
13222,United States,San Francisco,77 Gear Way,Max Mole,California,94111

INFO 2019-06-10 18:21:17,687 [[MuleRuntime].io.29: [apdev-examples].syncDBaccountsToCSV.BLOCKING @53787454]
[event: 0f1b1df0-8bce-11e9-b4be-f018983d9329] org.mule.runtime.core.internal.processor.LoggerMessageProcessor:
13337,USA,Oakland,Oakland Coliseum,Draymon Green,CA,94044
```

43. Stop the project.

44. Return to the resources folder in your computer's file explorer; you should see a new DBaccounts.csv file appear in the output folder.

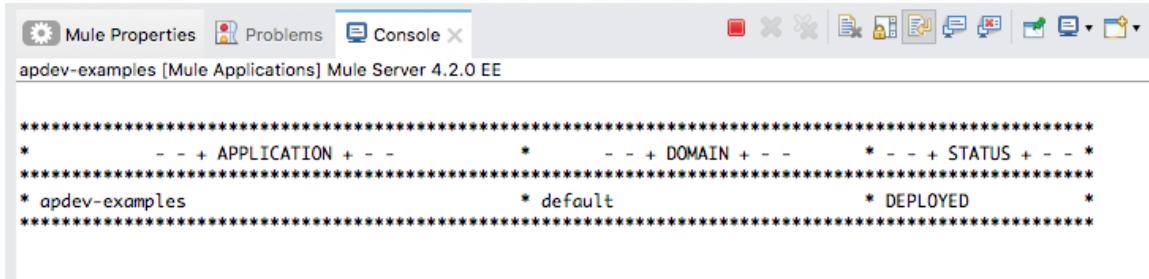


45. Open the DBaccounts.csv file with a text editor; you should see the records.

## Test the application again without clearing application data

46. Return to Anypoint Studio and run the project.

47. In the Clear Application Data dialog box, click No.
48. Look at the console; you should see no new records output.



```

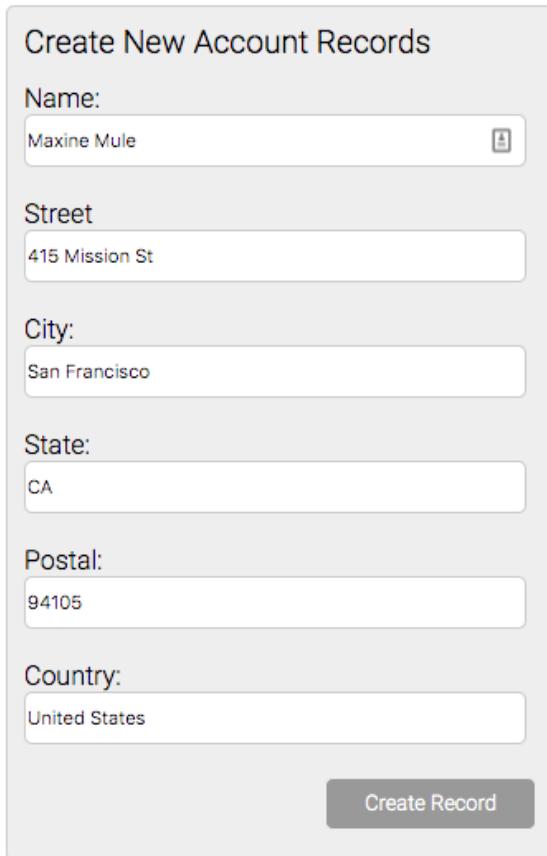
* - + APPLICATION + - - * - + DOMAIN + - - * - - + STATUS + - - *

* apdev-examples * default * DEPLOYED *

```

## Add a new account to the database

49. Return to the account data in the web browser at <http://mu.mulesoft-training.com/accounts/show>.
50. Click Create More Accounts.
51. Fill out the form with data and click Create Record.



Create New Account Records

Name:

Street

City:

State:

Postal:

Country:

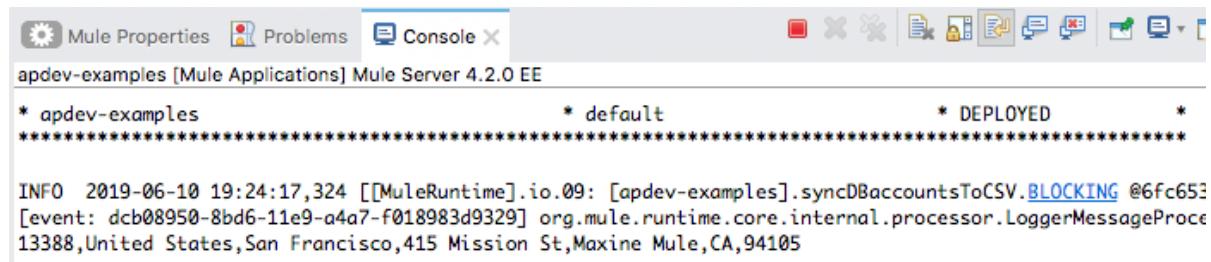
*Note: Set the postal code to a specific value. In the next walkthrough, you will retrieve only new records with this specific postal code, so you do not get all of the records.*

52. Click View Existing Accounts; you should see your new account.

## MUA Accounts

| accountID | name            | street         | city          |
|-----------|-----------------|----------------|---------------|
| 13388     | Maxine Mule     | 415 Mission St | San Francisco |
| 13387     | Maximilian Mule | 123 Main St    | San Francisco |

53. Return to the console in Anypoint Studio; you should see your new record.



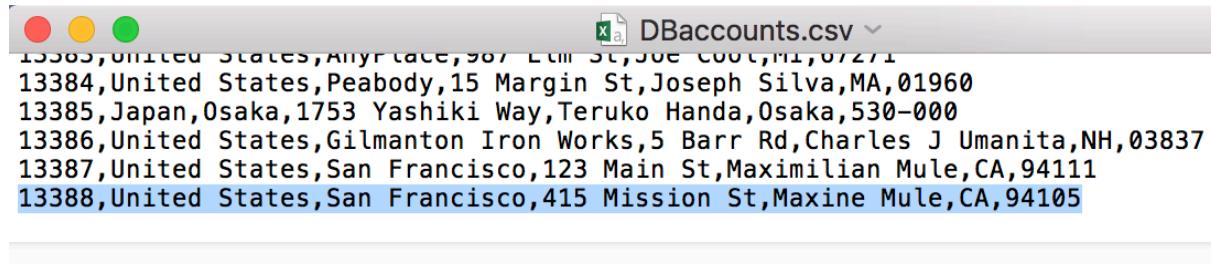
The screenshot shows the Anypoint Studio interface with the 'Console' tab selected. The title bar indicates 'apdev-examples [Mule Applications] Mule Server 4.2.0 EE'. The deployment status is shown as '\* apdev-examples \* default \* DEPLOYED \*'. Below the status, a log message is displayed:

```
INFO 2019-06-10 19:24:17,324 [[MuleRuntime].io.09: [apdev-examples].syncDBaccountsToCSV.BLOCKING @6Fc653
[event: dcb08950-8bd6-11e9-a4a7-f018983d9329] org.mule.runtime.core.internal.processor.LoggerMessageProce
13388,United States, San Francisco, 415 Mission St, Maxine Mule, CA, 94105
```

54. Stop the project.

55. Return to your computer's file explorer; the modified date on the DBAccounts.csv file should have been updated.

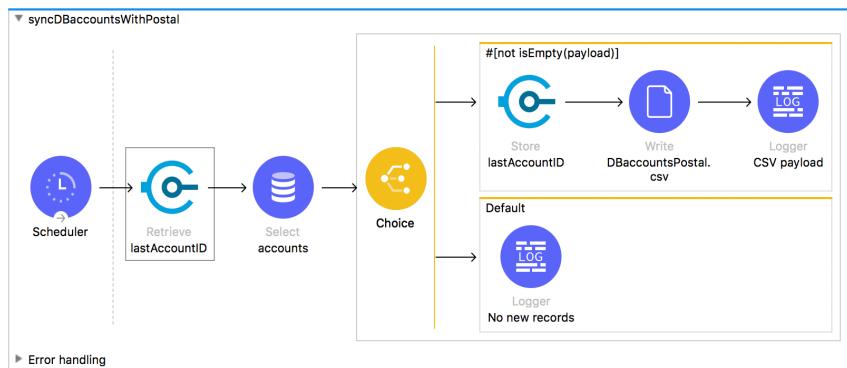
56. Open DBaccounts.csv and locate your new record at the end of the file.



## Walkthrough 12-3: Schedule a flow and use manual watermarking

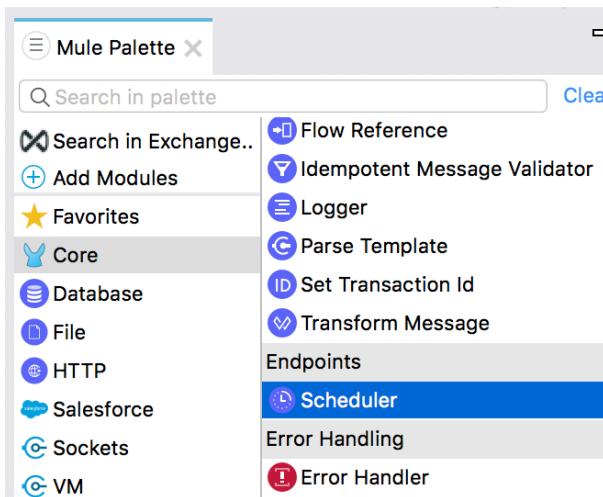
In this walkthrough, you continue to work with the accounts table in the training database. You will:

- Use the Scheduler component to create a new flow that executes at a specific frequency.
- Retrieve accounts with a specific postal code from the accounts table.
- Use the Object Store component to store the ID of the latest record and then use it to only retrieve new records.



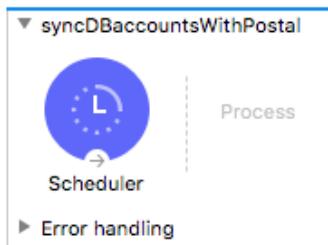
### Create a flow that executes at a specific frequency

1. Return to accounts.xml in Anypoint Studio.
2. Locate the Scheduler component in the Core section of the Mule Palette.

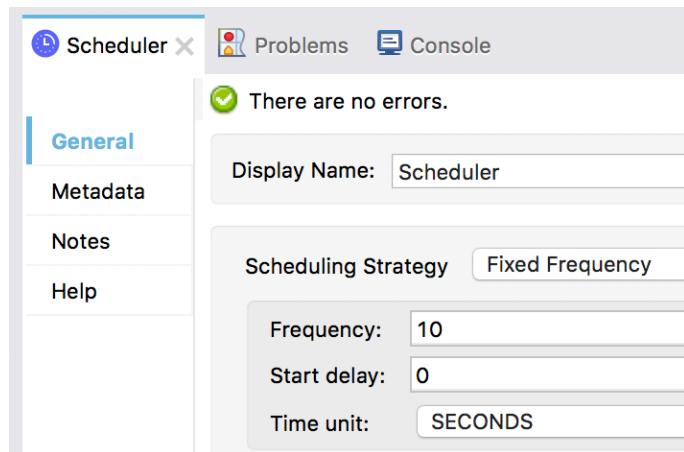


3. Drag a Scheduler component from the Mule Palette and drop it at the top of the canvas to create a new flow.

4. Change the name of the flow to syncDBaccountsWithPostal.

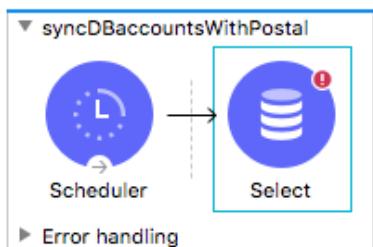


5. In the Scheduler properties view, set the frequency to 10 seconds.



## Retrieve records with a specific postal code from the database

6. From the Mule Palette, drag a Database Select operation and drop it in the flow.



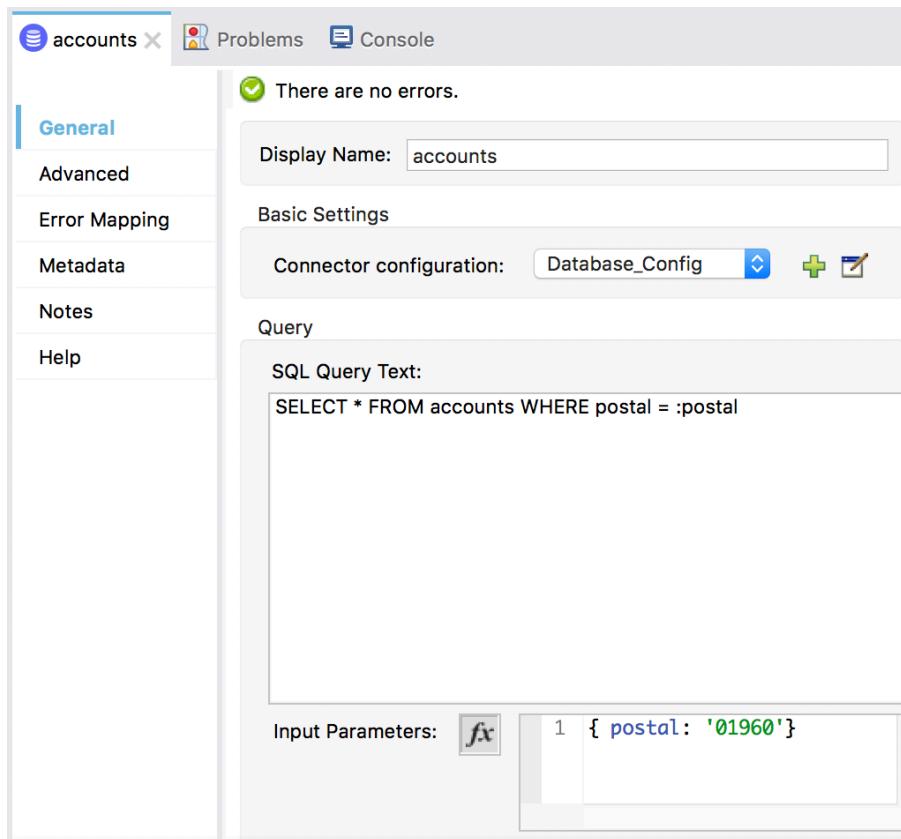
7. In the Select properties view, set the following:

- Display Name: accounts
- Connector configuration: Database \_Config
- SQL query text: SELECT \* FROM accounts WHERE postal = :postal

- In the Input Parameters section, use expression mode to create a postal input parameter with the specific postal code you used in the previous walkthrough.

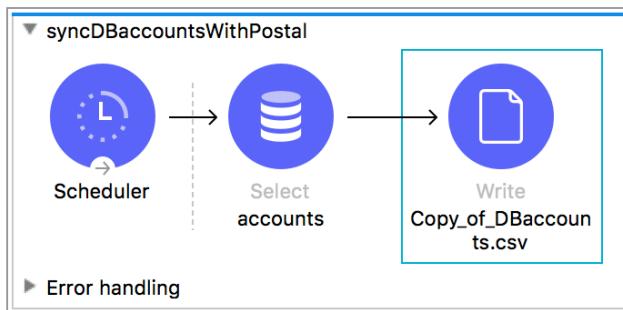
```
{ postal: 'yourPostalValue' }
```

*Note: If you want, you can store the postal code as a property in config.yaml and then reference it here in the DataWeave expression as { postal: p('propertyName') }.*



## Output the records to a CSV file

- Copy the Write operation in syncDBaccountsToCSV and paste it at the end of syncDBaccountsWithPostal.

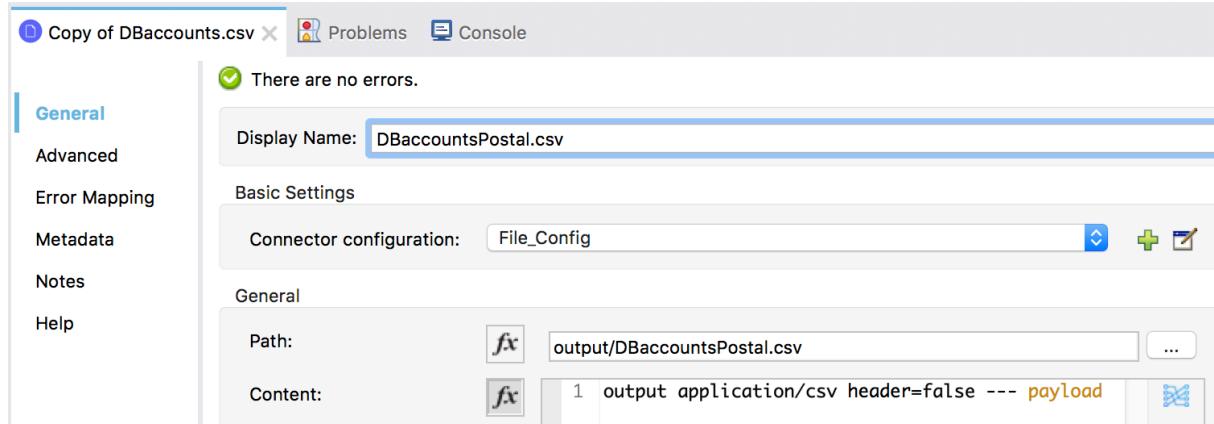


10. In the properties view for the Write operation, set the following values:

- Display Name: DBaccountsPostal.csv
- Path: output/DBaccountsPostal.csv

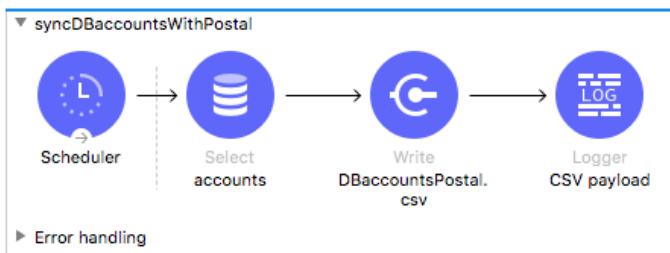
11. Change the content to output the payload as application/csv with a header property equal to false.

```
output application/csv header=false --- payload
```

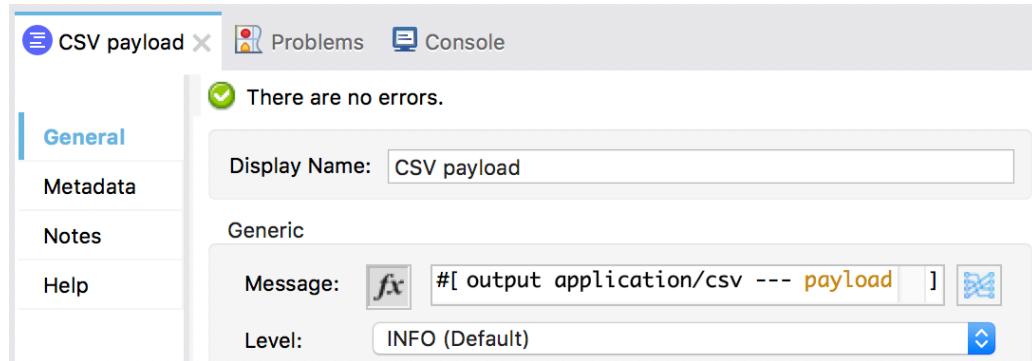


## Log the payload

12. Add a Logger at the end of the flow and change the display name to CSV payload.

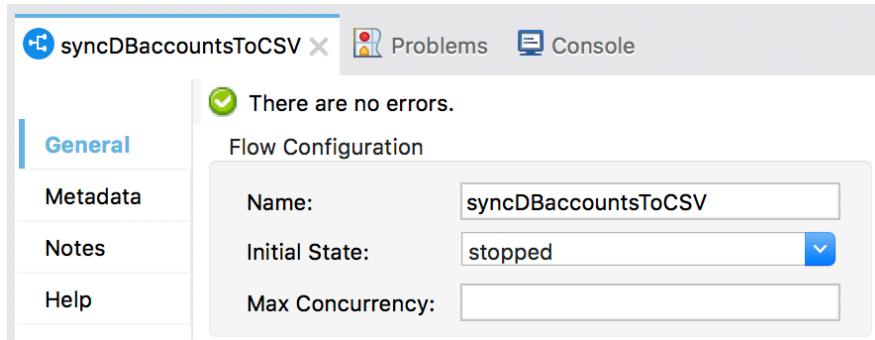


13. Set the message to the payload as type application/csv.



## Stop the syncDBaccountsToCSV flow so it does not run

14. In the properties view for the syncDBaccountsToCSV flow, set the initial state to stopped.



## Test the application

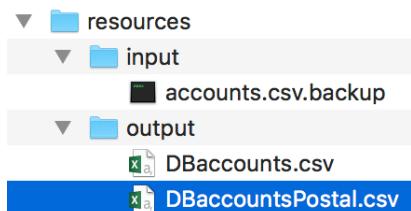
15. Save the file and run the project.
16. In the Clear Application Data dialog box, click Yes.
17. Watch the console, you should see the same records displayed every 10 seconds.

```
INFO 2019-06-10 22:02:03,758 [[MuleRuntime].io.10: [apdev-examples].sync
e78a6420-8bec-11e9-b90c-f018983d9329] org.mule.runtime.core.internal.proc
accountID,country,city,street,name,state,postal
13388,United States,San Francisco,415 Mission St,Maxine Mule,CA,94105
13389,United States,San Francisco,415 Mission St,Clark Kent,CA,94105
13390,United States,San Francisco,415 Mission St,Lois Lane,CA,94105

INFO 2019-06-10 22:02:13,792 [[MuleRuntime].io.09: [apdev-examples].sync
ed804520-8bec-11e9-b90c-f018983d9329] org.mule.runtime.core.internal.proc
accountID,country,city,street,name,state,postal
13388,United States,San Francisco,415 Mission St,Maxine Mule,CA,94105
13389,United States,San Francisco,415 Mission St,Clark Kent,CA,94105
13390,United States,San Francisco,415 Mission St,Lois Lane,CA,94105
```

*Note: Right now, all records with matching postal code are retrieved – over and over again. Next, you will modify this so only new records with the matching postal code are retrieved.*

18. Stop the project.
19. Return to the resources folder in your computer's file browser; you should see the new DBaccountsPostal.csv file in the output folder.



20. Open the file and review the contents.

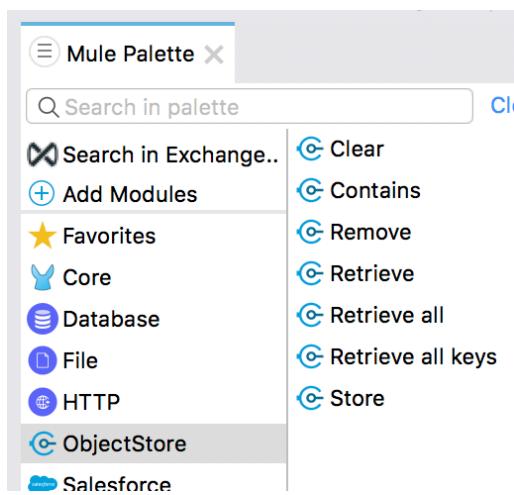
21. Delete the file.

## Add the ObjectStore module to the project

22. In the Mule Palette, select Add Modules.

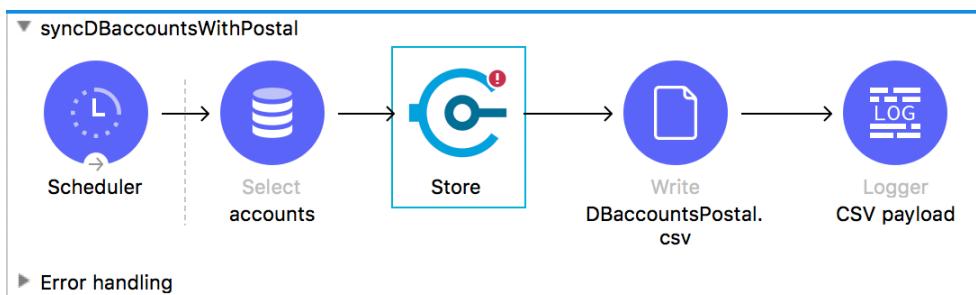
23. Select the ObjectStore connector in the right side of the Mule Palette and drag and drop it into the left side.

24. If you get a Select module version dialog box, select the latest version and click Add.



## Store the ID of the last record retrieved

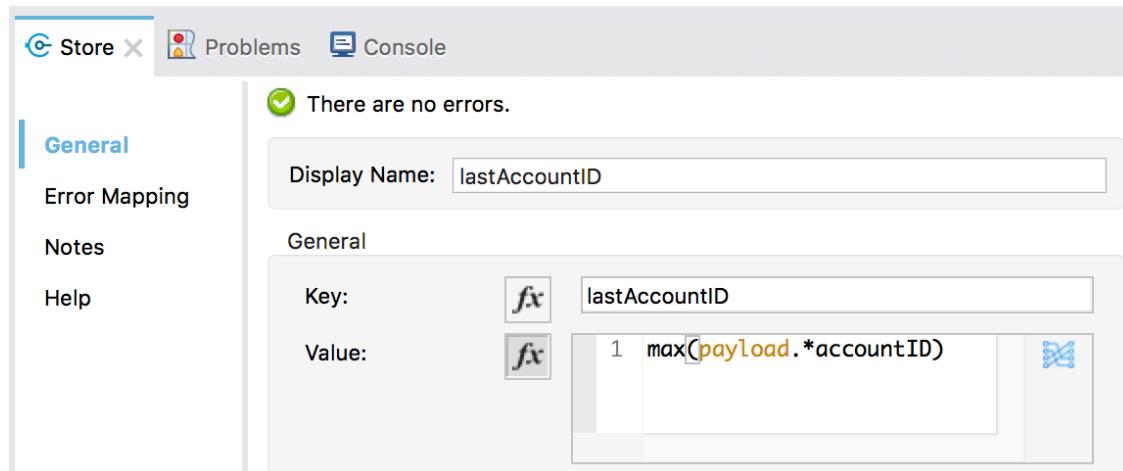
25. Drag the Store operation for the ObjectStore connector from the Mule Palette and drop it after the Database Select operation.



26. In the Store properties view, set the display name and key to lastAccountId.

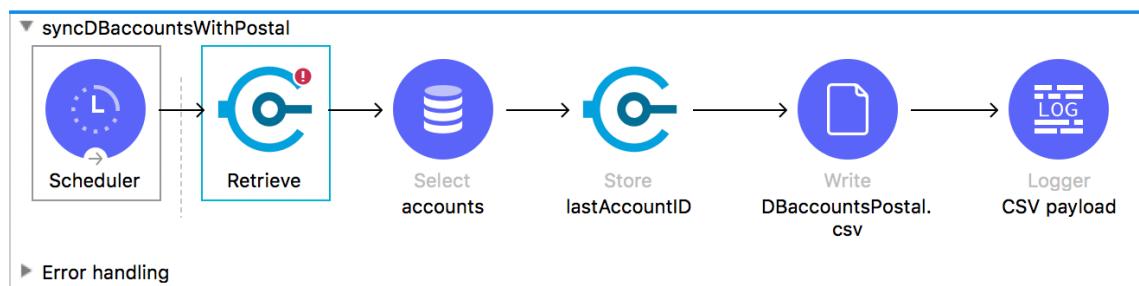
27. Set the value to an expression for the maximum value of lastAccountID in the payload containing the retrieved records.

```
max(payload.*accountID)
```



### Before the query, retrieve the ID of the last record retrieved

28. Drag the Retrieve operation for the ObjectStore connector from the Mule Palette and drop it before the Database Select operation.



29. In the Retrieve properties view, set the following values:

- Display Name: lastAccountID
- Key: lastAccountID
- Default value: 0

30. Select the Advanced tab and set the target variable to lastAccountID so the value is stored in a variable called lastAccountID.

## Modify the query to only retrieve new records with a specific postal code

31. In the accounts Select properties view, add a second query input parameter called lastAccountID that is equal to the watermark value.

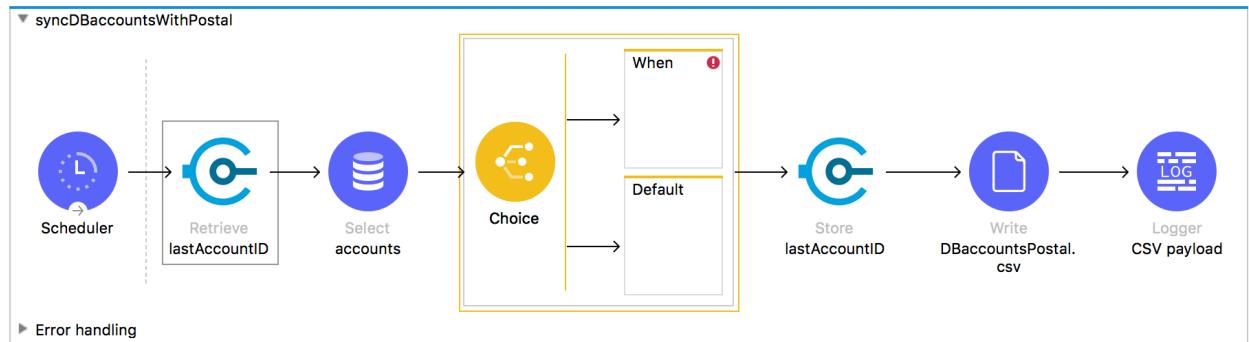
```
{postal: 'yourValue', lastAccountID: vars.lastAccountID}
```

32. Modify the SQL query text to use this parameter to only retrieve new records.

```
SELECT * FROM accounts WHERE postal = :postal AND accountID > :lastAccountID
```

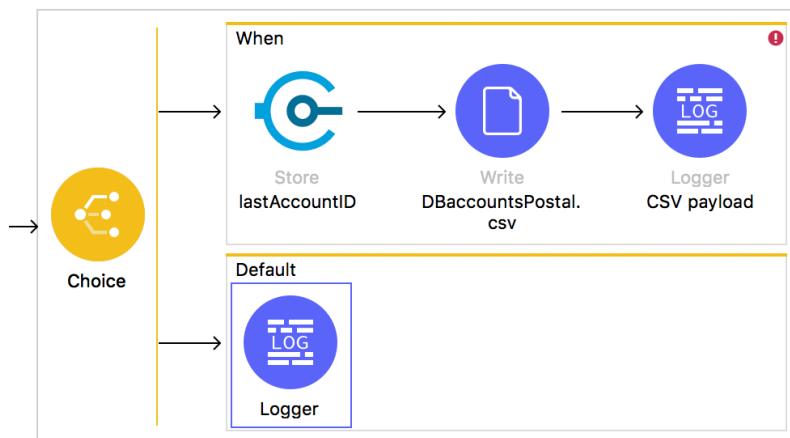
## Only store the value if new records were retrieved

33. Add a Choice router after the Select operation.

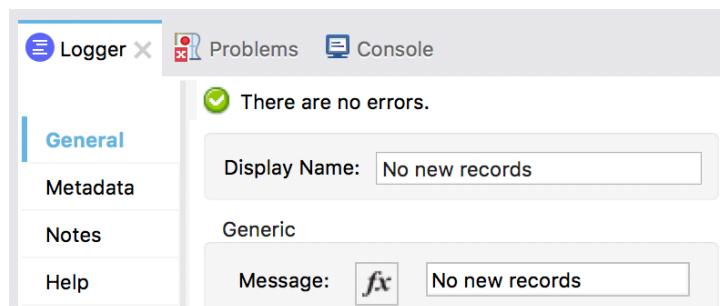


34. Move the Store, Write, and Logger operations into the when branch of the router.

35. Add a Logger to the default branch.

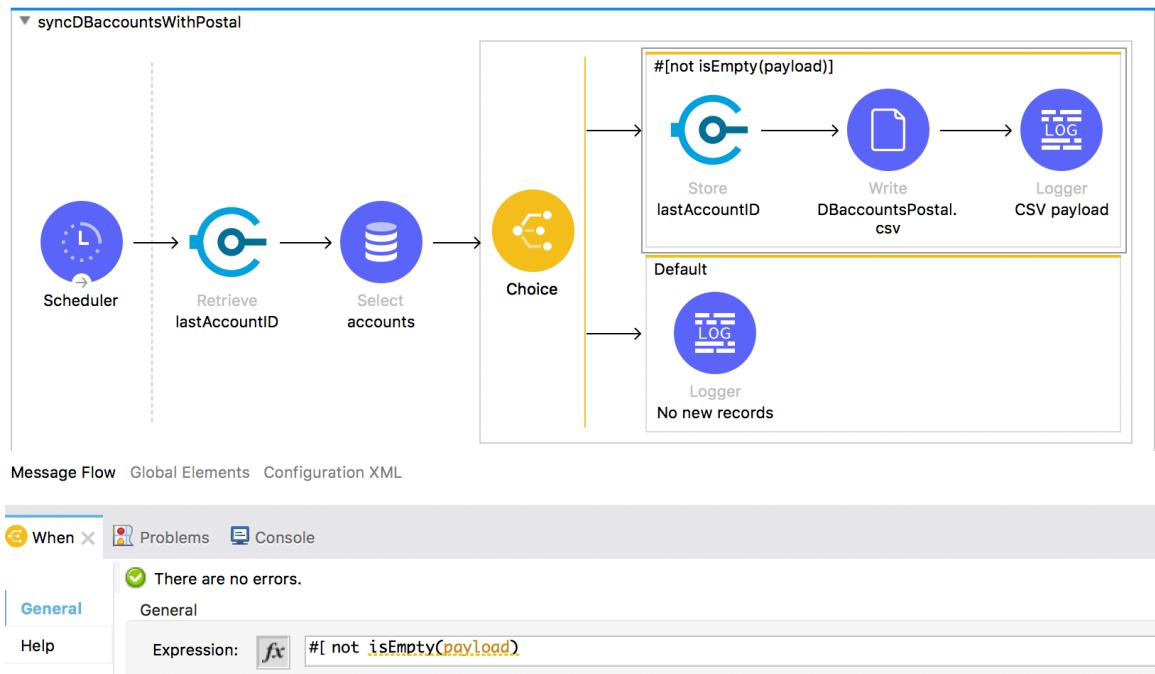


36. In the default branch Logger properties view, set the display name and message to No new records.



37. In the Choice router, click the when scope and in the properties view, add an expression to route the event when the payload is not empty.

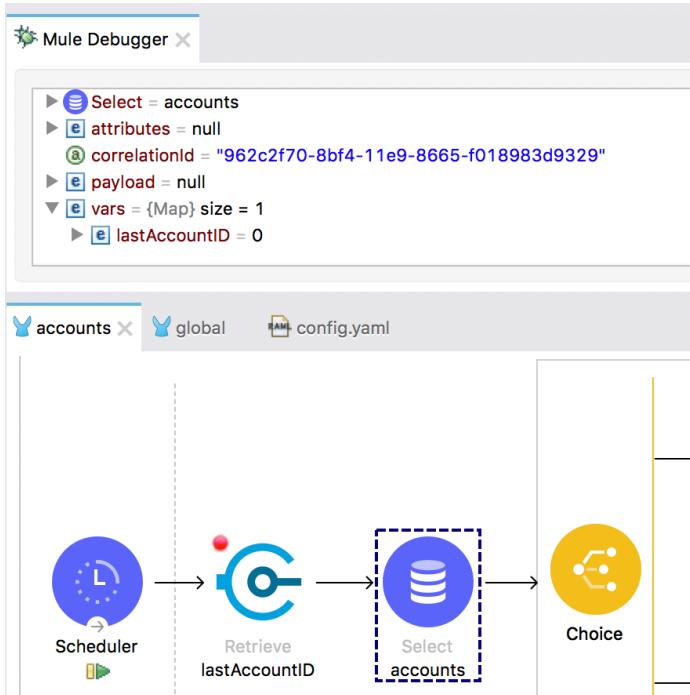
```
##[not isEmpty(payload)]
```



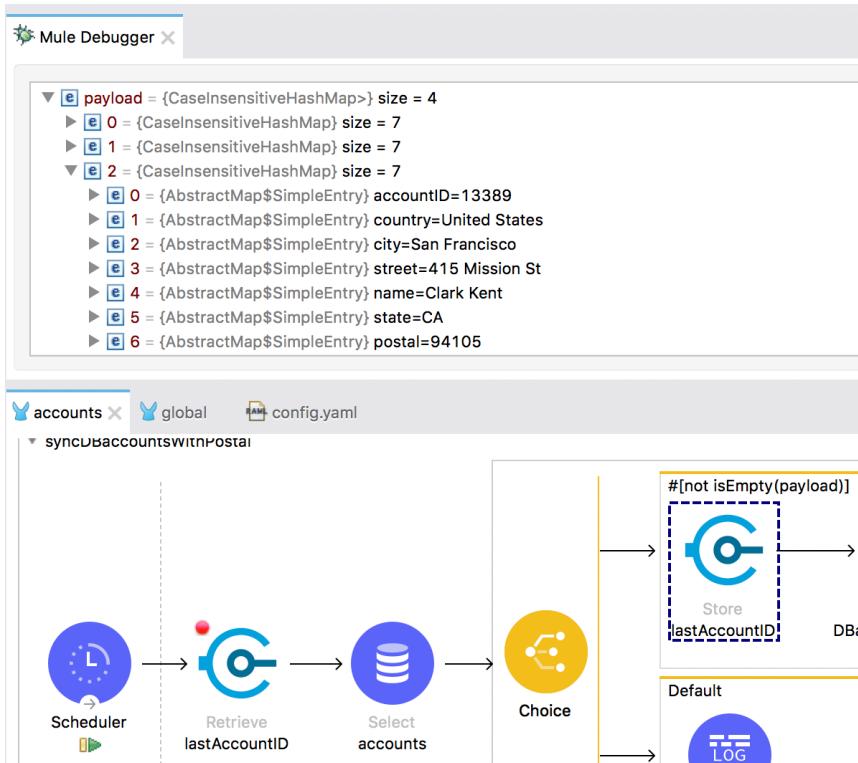
## Clear the application data and debug the application

38. Add a breakpoint to the Retrieve operation.
39. Save the file and debug the project.
40. In the Clear Application Data dialog box, click Yes.

41. In the Mule Debugger, step to the Select operation; you should see a lastAccountID variable with a value of 0.

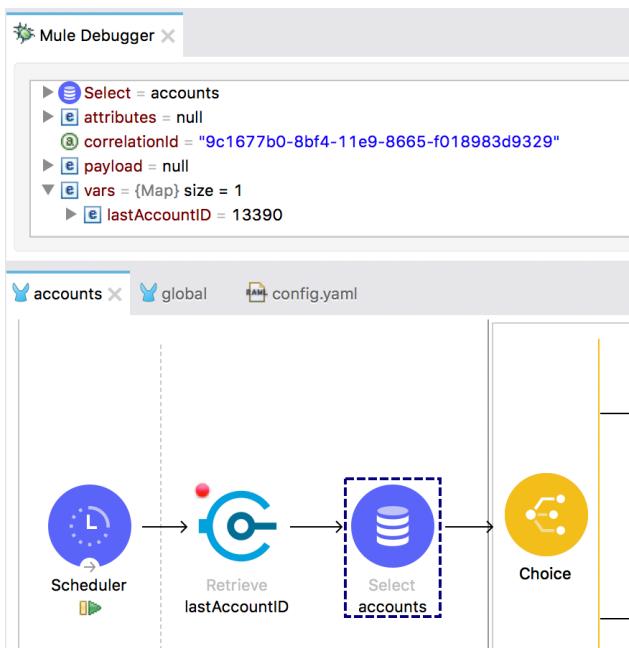


42. Step into the Choice router; you should see record(s) were retrieved from the database.

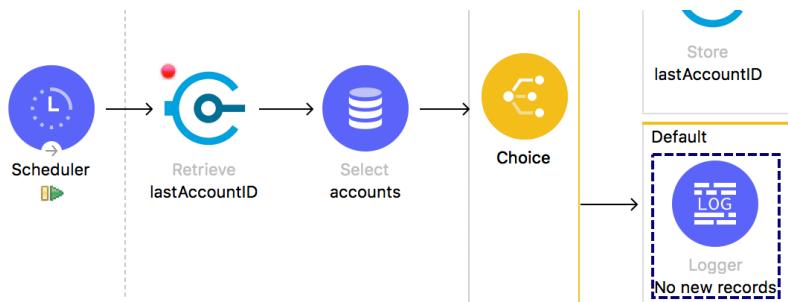


43. Click resume; the flow should be executed again, and execution should be stopped back at the beginning of the flow.

44. Step to the Select operation; you should see the lastAccountID variable now has a non-zero value.



45. Step into the Choice router; you should see no records were retrieved from the database (unless someone else just added one with the same postal code!).



46. Stop the project and switch perspectives.

47. Return to your computer's file explorer and locate and open the new DBaccountsPostal.csv file.

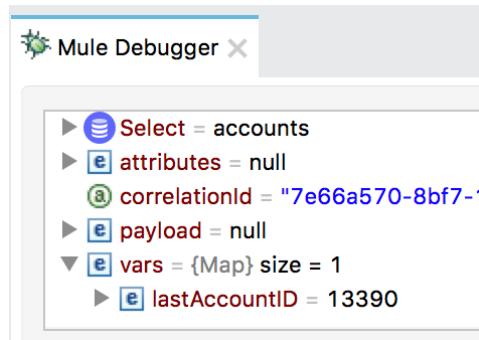
*Note: If you see the same records more than once, it was because during debugging, the flow was executed again before the watermark was stored.*

## Debug the application and do not clear application data

48. Return to Anypoint Studio and debug the project.

49. In the Clear Application Data dialog box, click No.

50. In the Mule Debugger, step to the Select operation; you should see a lastAccountID variable with the same non-zero value.



51. Step into the Choice router; no new records should have been returned.  
52. Remove the breakpoint from the Retrieve operation.  
53. Make sure there are no other breakpoints in the flow then click Resume to continue application execution.

## Add a new account to the database

54. Return to the account data in the web browser at <http://mu.mulesoft-training.com/accounts/show>.  
55. Click Create More Accounts.  
56. Fill out the form with data and use the same postal code.

Create New Account Records

Name:

Street

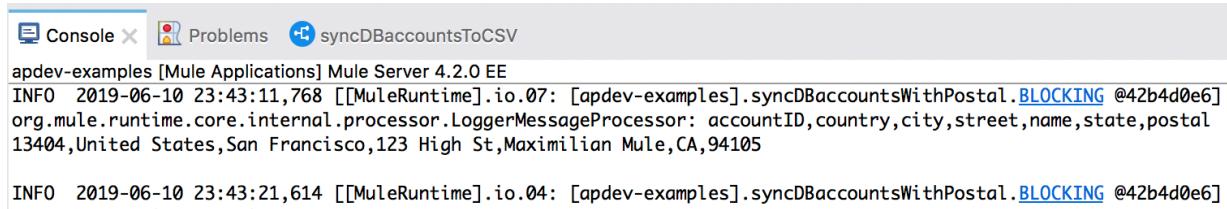
City:

State:

Postal:

Country:

57. Click Create Record.
58. Click View Existing Accounts; you should see your new account.
59. Return to Anypoint Studio; you should see your new record in the console.



```
Console × Problems syncDBaccountsToCSV
apdev-examples [Mule Applications] Mule Server 4.2.0 EE
INFO 2019-06-10 23:43:11,768 [[MuleRuntime].io.07: [apdev-examples].syncDBaccountsWithPostal.BLOCKING @42b4d0e6]
org.mule.runtime.core.internal.processor.LoggerMessageProcessor: accountID,country,city,street,name,state,postal
13404,United States,San Francisco,123 High St,Maximilian Mule,CA,94105

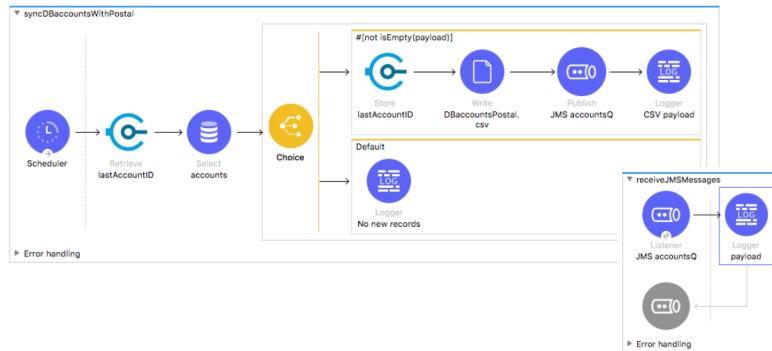
INFO 2019-06-10 23:43:21,614 [[MuleRuntime].io.04: [apdev-examples].syncDBaccountsWithPostal.BLOCKING @42b4d0e6]
```

60. Return to your computer's file explorer; the modified date on the DBAccountsPostal.csv file should have been updated.
61. Open DBaccountsPostal.csv and locate your new record at the end of the file.
62. Return to Anypoint Studio and switch perspectives.
63. Stop the project.

## Walkthrough 12-4: Publish and listen for JMS messages

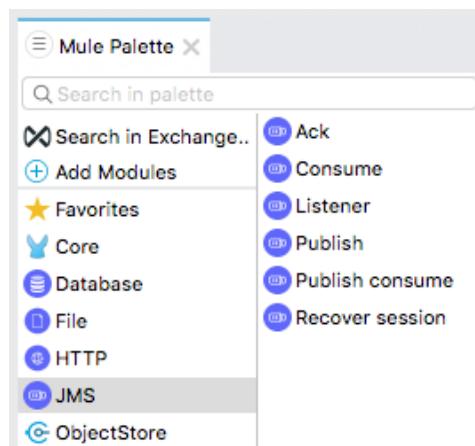
In this walkthrough, you send a JMS message for each new database record so it can be processed asynchronously. You will:

- Add and configure a JMS connector for ActiveMQ (that uses an in-memory broker).
- Send messages to a JMS queue.
- Listen for and process messages from a JMS queue.



### Add the JMS module to the project

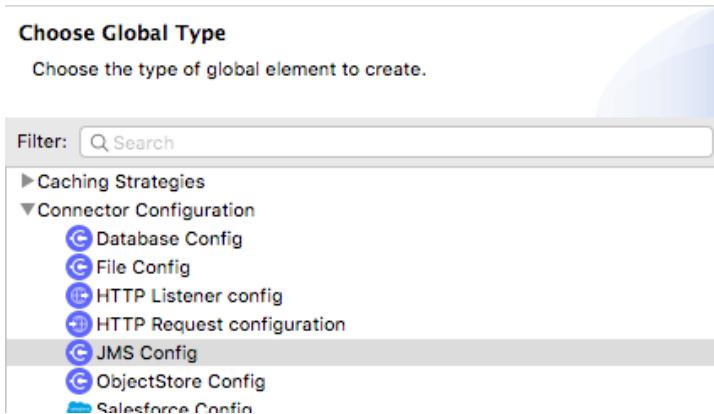
1. Return to accounts.xml.
2. In the Mule Palette, select Add Modules.
3. Select the JMS connector in the right side of the Mule Palette and drag and drop it into the left side.
4. If you get a Select module version dialog box, select the latest version and click Add.



### Configure the JMS connector

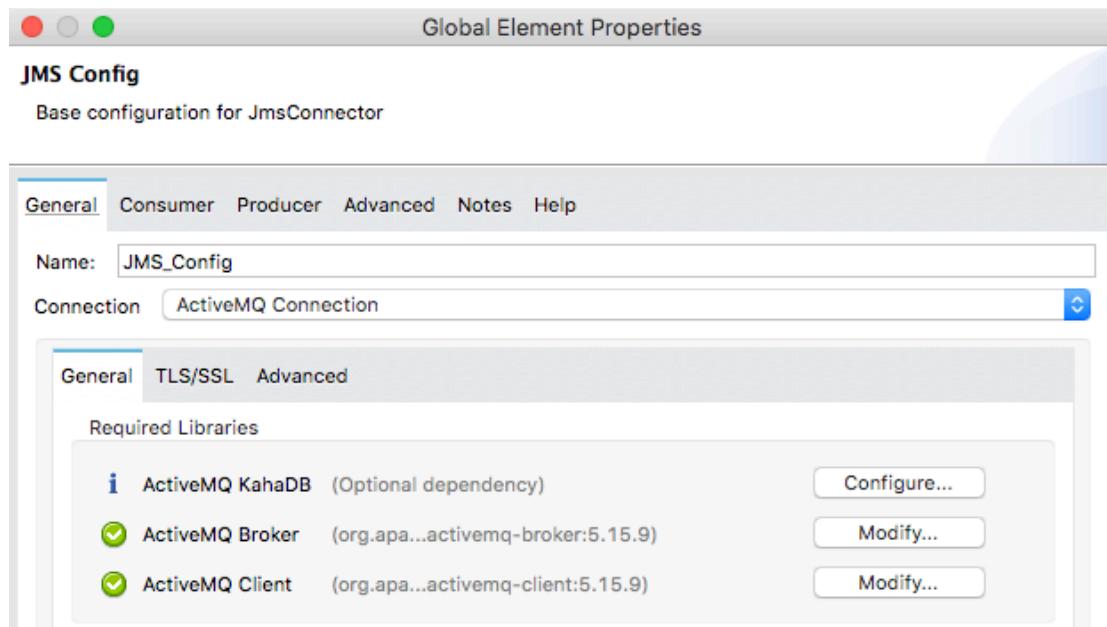
5. Return to the Global Elements view of global.xml.

6. Click Create.
7. In the Choose Global Type dialog box, select Connector Configuration > JMS Config and click OK.



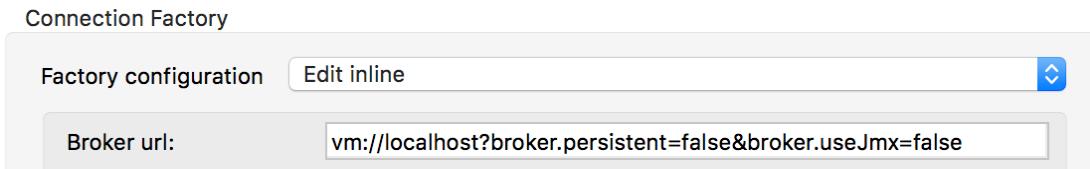
8. In the Global Element Properties dialog box, make sure the connection is set to ActiveMQ Connection.
9. Click the Configure > Add Maven dependency option for the ActiveMQ Broker.
10. In the Maven dependency dialog box, enter activemq-broker in the Search Maven Central field.
11. Select the org.apache.activemq:activemq-broker entry and select Finish.

*Note: There are local versions of the drivers available in the student files.*



12. Scroll down and locate the Connection Factory section.

13. For Factory configuration, select Edit inline; the Broker url should already be populated.

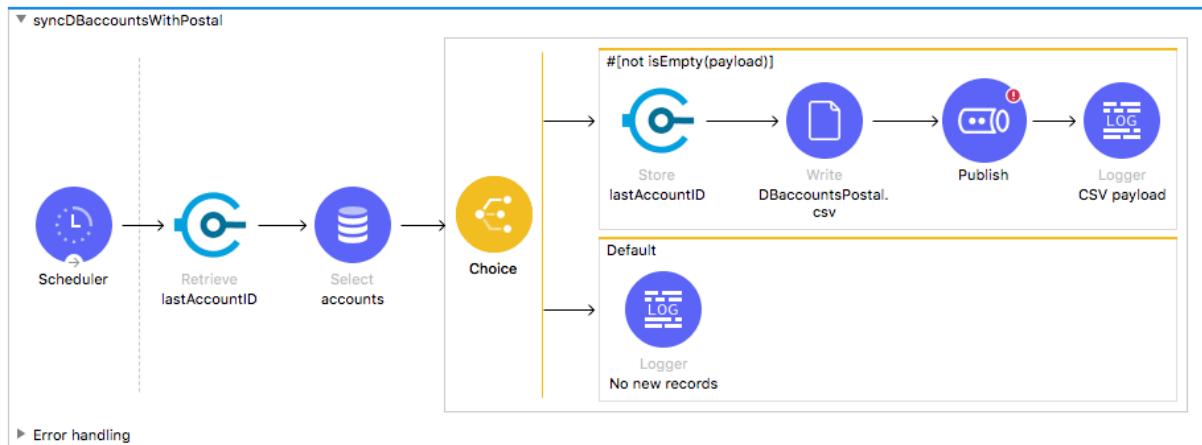


14. Click OK.

## Use the JMS Publish operation to send a message to a JMS queue

15. Return to accounts.xml.

16. Drag out a JMS Publish operation from the Mule Palette and drop it after the Write operation in syncDBaccountsWithPostal.



17. In the Publish properties view, set the following values:

- Display Name: JMS accountsQ
- Connector configuration: JMS\_Config
- Destination: accountsQ
- Destination type: QUEUE

The screenshot shows the Mule Studio interface with the 'Publish' tab selected. In the left sidebar, 'General' is the active category. On the right, under 'Basic Settings', the 'Display Name' is set to 'JMS accountsQ'. Under 'General', the 'Destination' is 'accountsQ', 'Destination type' is 'QUEUE (Default)', and 'Send correlation id' is '-- Empty --'.

18. Modify the message body to be of type application/json.

```
output application/json --- payload
```

The screenshot shows the Mule Studio interface with the 'Publish' tab selected. In the left sidebar, 'General' is the active category. On the right, under 'Message', the 'Body' field contains the expression 'output application/json --- payload'.

19. Scroll down and locate the User Properties field.

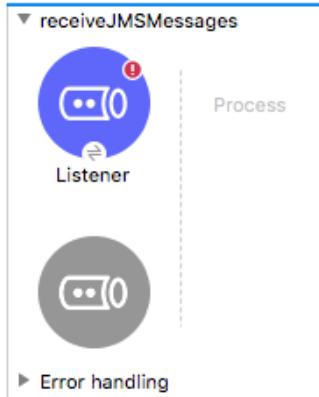
20. Use expression mode and set it equal to an object with a key called publisher and a value of training.

```
{"publisher":"training"}
```

The screenshot shows the Mule Studio interface with the 'Publish' tab selected. In the left sidebar, 'General' is the active category. On the right, under 'User Properties', the field contains the expression '{\"publisher\":\"training\"}'.

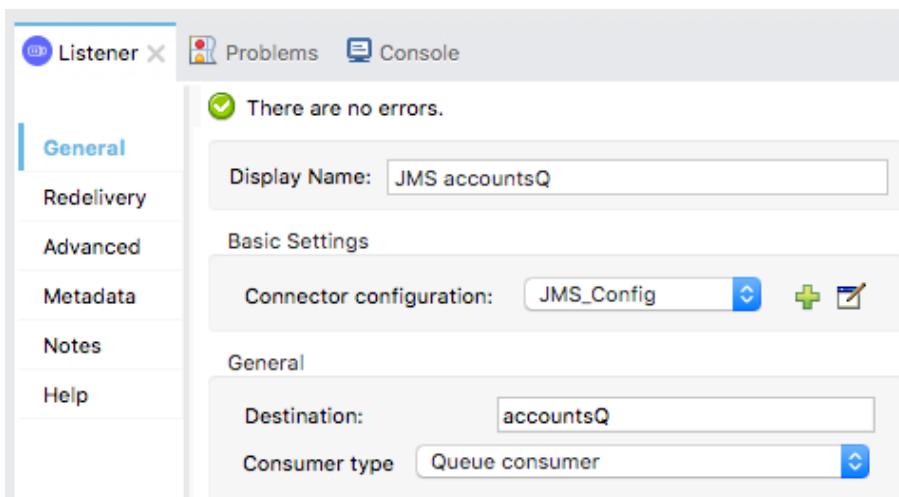
## Create a flow to listen to a JMS topic

21. Drag out the JMS Listener operation from the Mule Palette and drop it in the canvas to create a new flow.
22. Give the flow a new name of receiveJMSMessages.



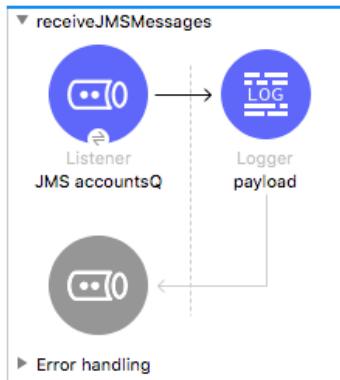
23. In the Listener properties view, set the following values:

- Display Name: JMS accountsQ
- Connector configuration: JMS\_Config
- Destination: accountsQ
- Consumer type: Queue consumer



24. Add a Logger to the flow.

25. Set its display name to payload and its message to #[payload].



## Clear application data and debug the application

26. Add a breakpoint to the Logger in receiveJMSmessages.
27. Save the files and debug the project.
28. In the Clear Application Data dialog box, click Yes.
29. Wait until the project deploys; application execution should stop in receiveJMSMessages.
30. In the Mule Debugger view, look at the value of the payload.

The screenshot shows the 'Mule Debugger' view. At the top, there is a tree view with nodes: 'Logger = payload', 'attributes = {JmsAttributes} org.mule.extensions.jms.api.message.JmsAttrib...', 'correlationId = "2cd41350-8c4c-11e9-95f6-f018983d9329"', and 'payload = [{"accountID": 13404, "country": "United States"}]'. The 'payload' node is expanded, showing its content: 'mediaType = application/json; charset=UTF-8' and 'vars = {Map} size = 0'. To the right of the tree view, the actual JSON payload is displayed in a code editor:

```
[{"accountID": 13404, "country": "United States"}]
```

Below the tree view, there are tabs for 'accounts', 'global', and 'config.yaml'. At the bottom, the Mule flow diagram is shown again, identical to the one in the previous step, with the 'Logger payload' component highlighted by a dashed blue border.

31. Expand the Attributes and locate the publisher property in the userProperties.

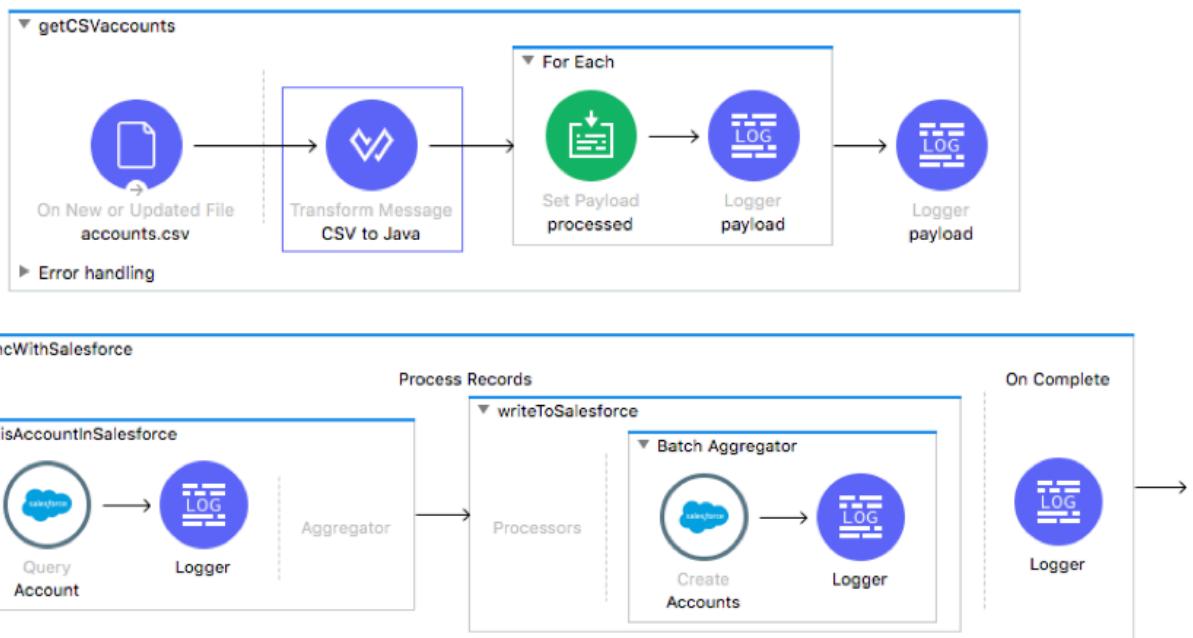
```
▼ e attributes = {JmsAttributes} org.mule.extensions.jms.api.message.JmsAttrib
 ⑧ ^mediaType = application/java
 ⑧ ackId = "null"
▶ e headers = {JmsHeaders} org.mule.extensions.jms.api.message.JmsHead
▼ e properties = {JmsMessageProperties} org.mule.extensions.jms.api.messa
 ⑧ JMSX_PREFIX = "JMSX"
 ⑧ JMS_PREFIX = "JMS"
▶ e all = {RegularImmutableMap} size = 3
 ⑧ jmsProperties = {HashMap} size = 0
▶ e jmsxProperties = {JmsxProperties} org.mule.jms.commons.api.message.J
▼ e userProperties = {HashMap} size = 3
 ▶ e 0 = {HashMap$Node} MM_MESSAGE_ENCODING=UTF-8
 ▶ e 1 = {HashMap$Node} MM_MESSAGE_CONTENT_TYPE=application/
 ▶ e 2 = {HashMap$Node} publisher=training
 ⑧ serialVersionUID = -8148917084189760450
```

32. Step through the rest of the application; you should see the JSON message in the console.

*Note: If you want to test further, return to the account data in the web browser at <http://mu.mulesoft-training.com/accounts/show> and add a new record with the same postal code.*

33. Stop the project and switch perspectives.

# Module 13: Processing Records



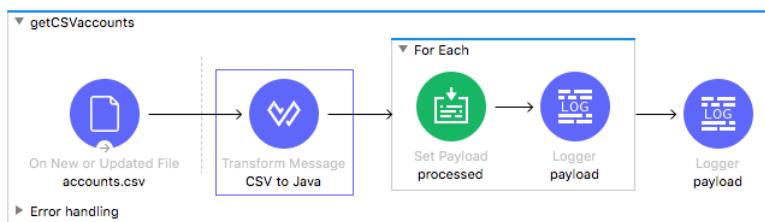
At the end of this module, you should be able to:

- Process items in a collection using the For Each scope.
- Process records using the Batch Job scope.
- Use filtering and aggregation in a batch step.

## Walkthrough 13-1: Process items in a collection using the For Each scope

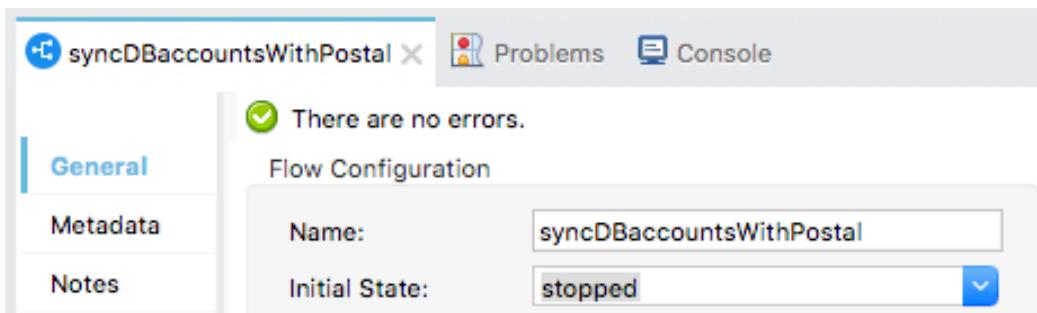
In this walkthrough, you split a collection and process each item in it. You will:

- Use the For Each scope element to process each item in a collection individually.
- Change the value of an item inside the scope.
- Examine the payload before, during, and after the scope.
- Look at the thread used to process each item.



### Stop the syncDBaccountsWithPostal flow so it does not run

1. Return to accounts.xml.
2. In the properties view for the syncDBaccountsWithPostal flow, set the initial state to stopped.

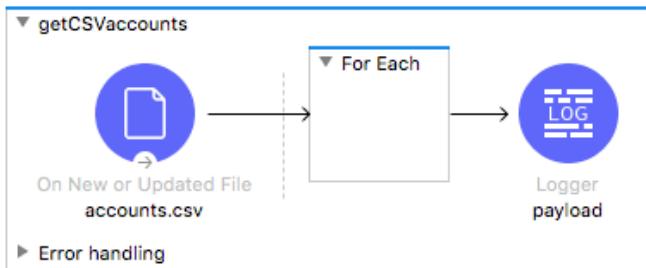


3. Right-click in the canvas and select Collapse All.

### Add a For Each scope

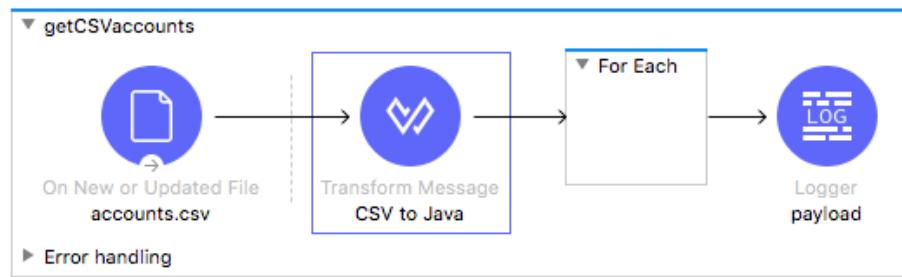
4. Expand getCSVaccounts.
5. In the Mule Palette, select Core.

- Locate the For Each scope and drag and drop it before the Logger.



## Transform the input to a collection

- Add a Transform Message component before the For Each scope.
- Set its display name to CSV to Java.



- In the Transform Message properties view, leave the output type set to java and set the expression to payload.

Output Payload ▾

---

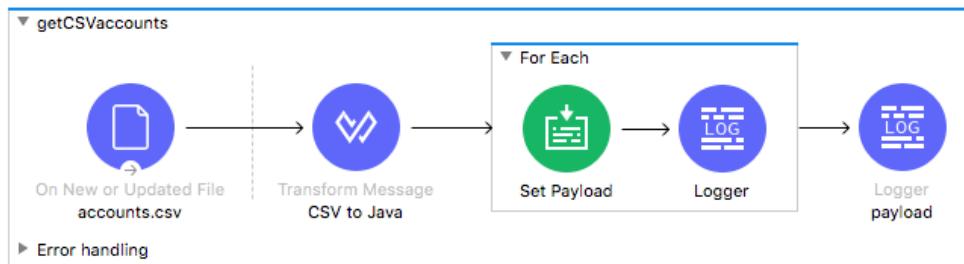
```

1 %dw 2.0
2 output application/java
3 ---
4 payload

```

## Process each element in the For Each scope

- Add a Set Payload transformer and a Logger to the For Each scope.



11. In the Set Payload properties view, set the display name and value to processed.

The screenshot shows the 'Set Payload' properties view. The 'General' tab is selected. The 'Display Name' field contains 'processed'. The 'Value' field contains 'fx' followed by a placeholder '#[ payload ]'. A message at the top says 'There are no errors.'

12. Set the Logger display name to payload and have it display the payload.

The screenshot shows the 'Logger' properties view. The 'General' tab is selected. The 'Display Name' field contains 'payload'. In the 'Generic' section, the 'Message' field contains 'fx' followed by a placeholder '#[ payload ]'. A message at the top says 'There are no errors.'

## Change the File listener so it does not rename files

13. In the accounts.csv properties view, delete the Rename to value.

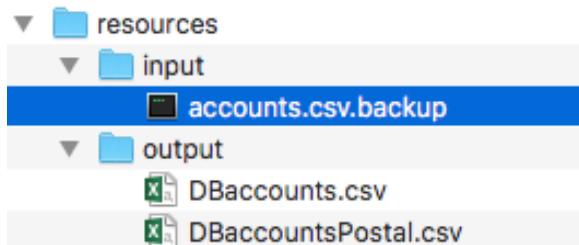
The screenshot shows the 'Post processing action' settings. Under 'Rename to:', there is a 'fx' button and an empty input field.

## Debug the application

14. Add a breakpoint to the Transform Message component.

15. Save the file, debug the project, and do not clear application data.

16. Return to the student files in your computer's file browser.



17. Rename accounts.csv.backup to accounts.csv in the input folder.

18. Return to Anypoint Studio; application execution should have stopped at the Transform Message component.

19. In the Mule Debugger view, look at the payload type and value.

The image shows the Anypoint Studio interface. The top half displays the 'Mule Debugger' view, which shows the payload of a 'Transform CSV to Java' component. The payload is a list of billing addresses:

```
BillingStreet,BillingCity,BillingCountry,BillingState,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
```

The bottom half shows the 'Message Flow' for the 'getCSVaccounts' application. It starts with an 'On New or Updated File accounts.csv' event, followed by a 'Transform Message CSV to Java' component (which is currently highlighted with a red dot). This is followed by a 'For Each' loop. Inside the loop, the 'Set Payload processed' processor is shown, along with two 'Logger payload' processors that log the processed payload. The right side of the interface shows the 'Global Elements' palette with three checked items: 'accounts [processors:0] - receiveJM', 'accounts [processors:0] - getCSVac.', and 'accounts [processors:0] - syncDBAc.'

20. Step to the For Each scope; the payload should now be an ArrayList of LinkedHashMaps.

The screenshot shows the Mule Debugger interface. The top window displays variable information for the current scope:

- Foreach = For Each**
- attributes = {LocalFileAttributes} LocalFileAttributes[{lastModifiedTime=2019-11-01T10:30:00.000Z, correlationId="3294ee30-07a9-11ea-92b0-f018983d9329"}]**
- payload = {ArrayList} size = 3** (highlighted in blue)
- vars = {Map} size = 0**

The payload variable contains three elements (size = 3):

- 0 = {LinkedHashMap} size = 6**
- 1 = {LinkedHashMap} size = 6**
- 2 = {LinkedHashMap} size = 6**

Each element has entries for BillingStreet, BillingCity, BillingCountry, BillingState, Name, and BillingPostalCode. A mediaType entry is also present.

The bottom window shows the Mule flow configuration for the 'getCSVaccounts' endpoint:

```
graph LR
 OnNewFile[On New or Updated File accounts.csv] --> Transform[Transform Message CSV to Java]
 Transform --> ForEachScope[For Each]
 ForEachScope --> SetPayload[Set Payload processed]
 SetPayload --> Logger1[Logger payload]
 SetPayload --> Logger2[Logger payload]
```

The 'Set Payload processed' step is highlighted with a dashed blue box.

21. Step into the Set Payload in the For Each scope; the payload should now be a LinkedHashMap.

22. Expand Variables; you should see a counter variable.

The screenshot shows the Mule Debugger interface. The top window displays variable information for the current scope:

- Set Payload = processed**
- attributes = null**
- correlationId = "c75b65d0-8c57-11e9-9185-f018983c"**
- payload = {LinkedHashMap} size = 6**
- vars = {Map} size = 2**
- counter = 1**
- rootMessage = {MessageImplementation} \norg.mule..**

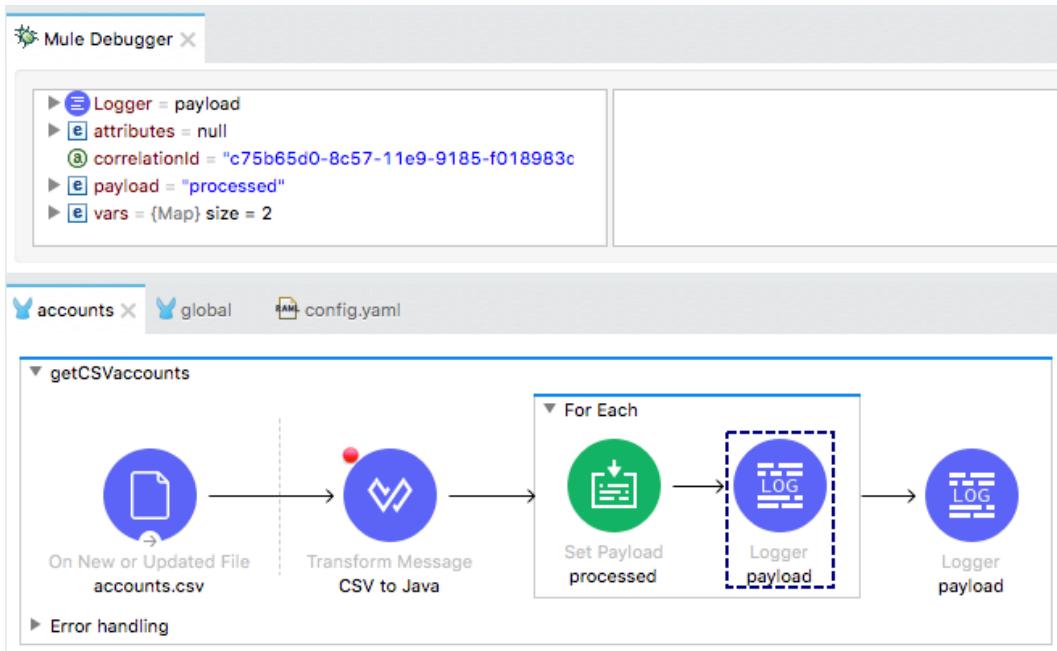
The 'Set Payload' step is highlighted with a dashed blue box.

The bottom window shows the Mule flow configuration for the 'getCSVaccounts' endpoint:

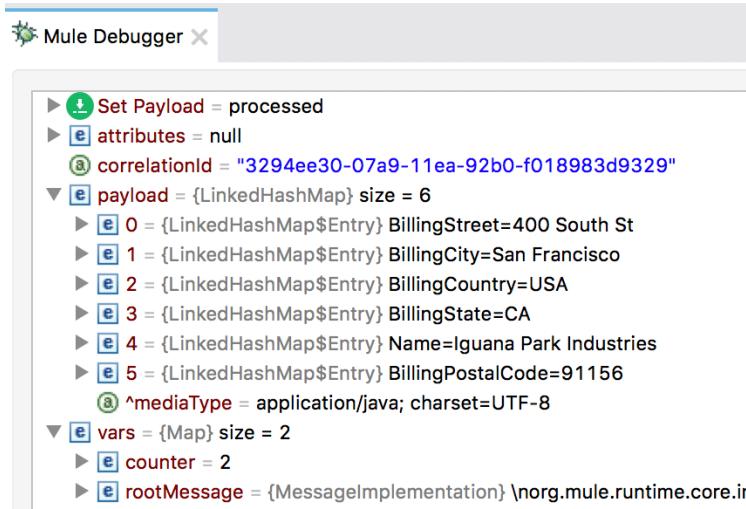
```
graph LR
 OnNewFile[On New or Updated File accounts.csv] --> Transform[Transform Message CSV to Java]
 Transform --> ForEachScope[For Each]
 ForEachScope --> SetPayload[Set Payload processed]
 SetPayload --> Logger1[Logger payload]
 SetPayload --> Logger2[Logger payload]
```

The 'Set Payload processed' step is highlighted with a dashed blue box.

23. Step again; you should see the payload for this record inside the scope has been set to the string, processed.



24. Step again and look at the payload and counter for the second record.



25. Step through the application to the Logger after the For Each scope; the payload should be equal to the original ArrayList of HashMaps and not a list of processed strings.

The screenshot shows the Mule Debugger interface with two main sections. The top section displays variable states:

- Logger = payload
- attributes = {LocalFileAttributes} LocalFileAttributes[lastModifiedTime=2019-11-15T08:40:23.817Z]
- correlationId = "3294ee30-07a9-11ea-92b0-f018983d9329"
- payload = {ArrayList} size = 3** (highlighted in blue)
- 0 = {LinkedHashMap} size = 6
- 1 = {LinkedHashMap} size = 6
- 2 = {LinkedHashMap} size = 6
  - 0 = {LinkedHashMap\$Entry} BillingStreet=777 North St
  - 1 = {LinkedHashMap\$Entry} BillingCity=San Francisco
  - 2 = {LinkedHashMap\$Entry} BillingCountry=USA
  - 3 = {LinkedHashMap\$Entry} BillingState=CA
  - 4 = {LinkedHashMap\$Entry} Name=Cat Park Industries
  - 5 = {LinkedHashMap\$Entry} BillingPostalCode=91156
- mediaType = application/java; charset=UTF-8
- vars = {Map} size = 0

The right side of the debugger shows the value `size = 3`. The bottom section shows the Mule flow diagram:

```
graph LR; Start(()) --> FileIn[On New or Updated File accounts.csv]; FileIn --> Transform[Transform Message CSV to Java]; Transform --> ForEach[For Each]; subgraph ForEachScope [For Each]; SetPayload[Set Payload processed] --> Log1[Logger payload]; end; Log1 --> Log2[Logger payload];
```

The flow starts with an "On New or Updated File" connector for "accounts.csv", followed by a "Transform Message" component ("CSV to Java"). This is followed by a "For Each" scope. Inside the scope, the payload is set to "processed". After the scope, there is a "Logger payload" component, which is highlighted with a dashed blue box. A second "Logger payload" component is shown to its right.

26. Step to the end of the application.

27. Stop the project and switch perspectives.

## Look at the processing threads

28. In the console, locate the thread number used to process each item in the collection; the same thread should be used for each: `cpuIntensive.01` in the following screenshot.

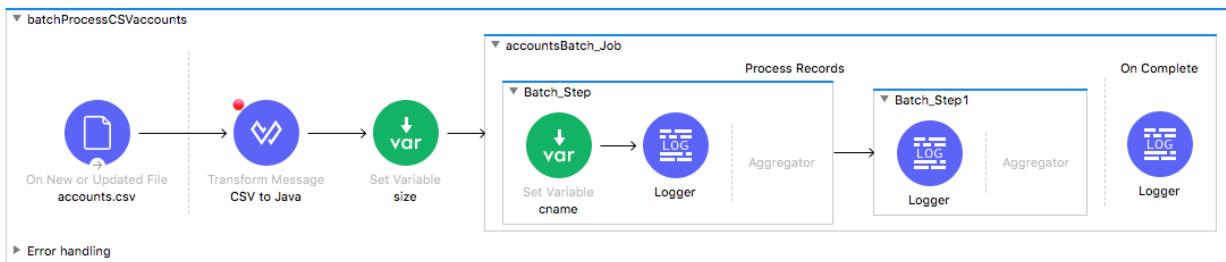
The screenshot shows the Eclipse IDE interface with the Mule Server 4.2.2 EE perspective. The top bar includes tabs for "Console", "Problems", and "Mule Properties". The toolbar has various icons for file operations and Mule specific functions. The main area is the "Console" view, which displays the following log output:

```
apdev-examples [Mule Applications] Mule Server 4.2.2 EE
INFO 2019-11-15 08:40:23,817 [[MuleRuntime].cpuIntensive.01]: [apdev-examples].getCSVaccounts.CPU_INTENSIVE @6c254540
INFO 2019-11-15 08:40:23,819 [[MuleRuntime].cpuIntensive.01]: [apdev-examples].getCSVaccounts.CPU_INTENSIVE @6c254540
INFO 2019-11-15 08:40:23,820 [[MuleRuntime].cpuIntensive.01]: [apdev-examples].getCSVaccounts.CPU_INTENSIVE @6c254540
INFO 2019-11-15 08:40:23,824 [[MuleRuntime].cpuIntensive.01]: [apdev-examples].getCSVaccounts.CPU_INTENSIVE @6c254540
```

## Walkthrough 13-2: Process records using the Batch Job scope

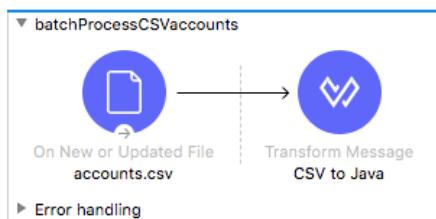
In this walkthrough, you create a batch job to process the records in a CSV file. You will:

- Use the Batch Job scope to process items in a collection.
- Examine the payload as it moves through the batch job.
- Explore variable persistence across batch steps and phases.
- Examine the payload that contains information about the job in the On Complete phase.
- Look at the threads used to process the records in each step.



### Create a new flow to read CSV files

1. Return to accounts.xml.
2. Drag a Flow scope from the Mule Palette and drop it above getCSVaccounts.
3. Change the flow name to batchProcessCSVaccounts.
4. Select the On New or Updated File operation in getCSVaccounts and copy it.
5. Click the Source section of batchProcessCSVaccounts and paste it.
6. Modify the display name to accounts.csv.
7. Add a Transform Message component to the flow.
8. Set the display name to CSV to Java.

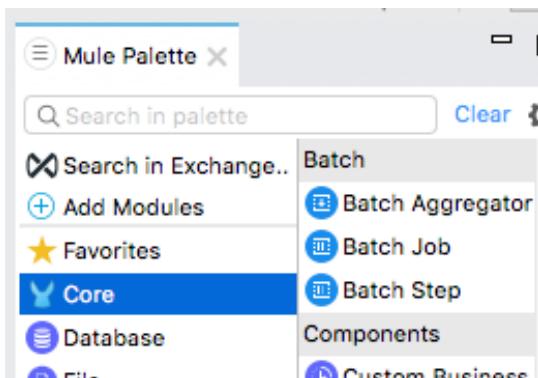


9. In the Transform Message properties view, change the body expression to payload.

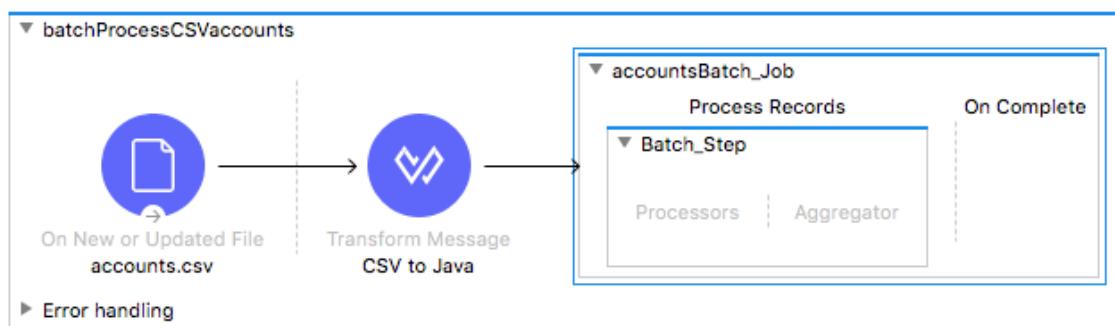
```
Output Payload ▾ + ✎ ─
1 %dw 2.0
2 output application/java
3 ---
4 payload
```

## Add a Batch Job scope

10. In the Mule Palette, select Core and locate the Batch elements.

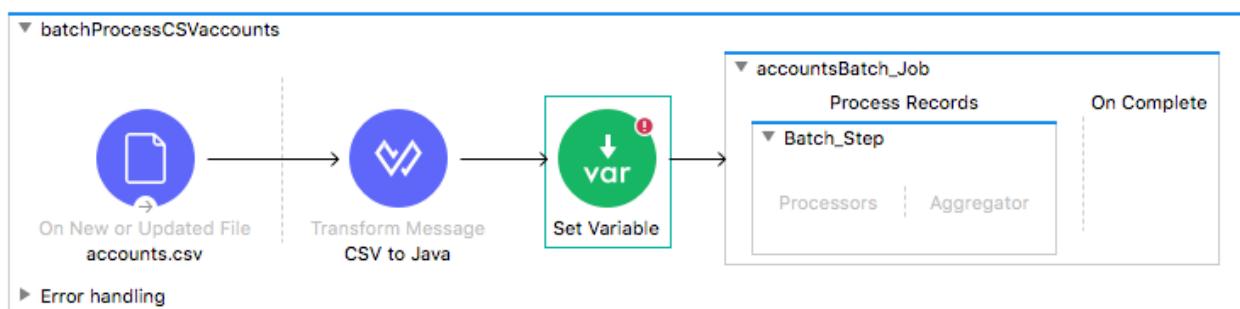


11. Drag a Batch Job scope and drop it after the Transform Message component.



## Set a variable before the Batch Job scope

12. Add a Set Variable transformer before the Batch Job scope.

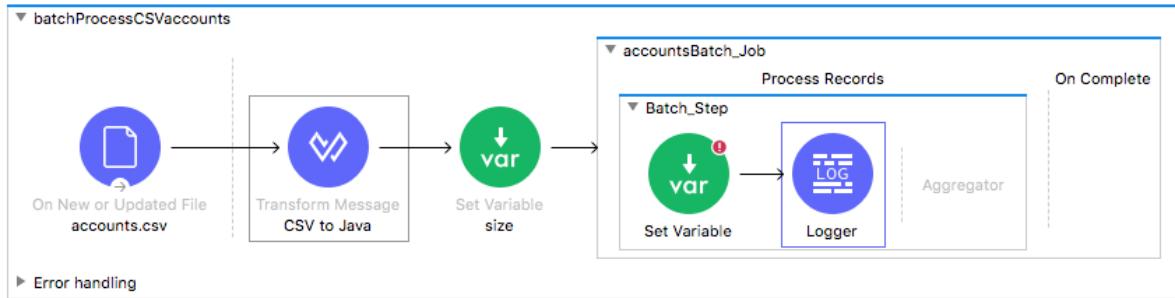


13. In the Set Variable properties view, set the following:

- Display Name: size
- Name: size
- Value: #[sizeOf(payload)]

## Set a variable inside the Batch Job scope

14. Add a Set Variable transformer to the processors phase of the batch step.
15. Add a Logger component after the Set Variable transformer.

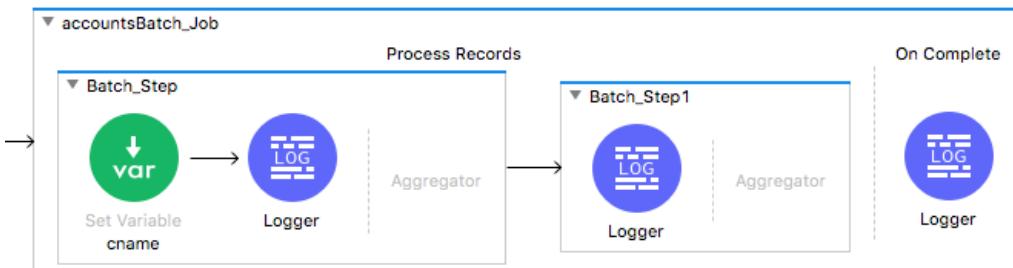


16. In the Set Variable properties view, set the following:

- Display Name: cname
- Name: cname
- Value: #[payload.Name]

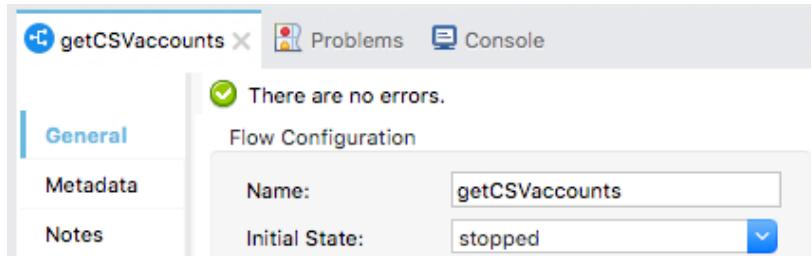
## Create a second batch step

17. Drag a Batch Step scope from the Mule Palette and drop it in the process records phase of the Batch Job scope after the first batch step.
18. Add a Logger to the processors phase of the second batch step.
19. Add a Logger to the On Complete phase.



## Stop the other flow watching the same file directory from being executed

20. In the properties view for the getCSVaccounts flow, set the initial state to stopped.



## Debug the application

21. In batchProcessCSVaccounts, add a breakpoint to the Transform Message component.
22. Save the file, debug the project, and do not clear application data.
23. Return to the student files in your computer's file browser and move accounts.csv from the output folder to the input folder.
24. In the Mule Debugger view, watch the payload as you step to the Batch Job scope; it should go from CSV to an ArrayList.
25. In the Mule Debugger view, expand Variables; you should see the size variable.

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

- Mule Debugger View:** Shows the payload variable expanded to an ArrayList of 6 entries, each a LinkedHashMap. The first entry has keys: BillingStreet=111 Boulevard Hausmann, BillingCity=Paris, BillingCountry=France, BillingState=, Name=Dog Park Industries, and BillingPostalCode=75008. It also shows a mediaType of application/java; charset=UTF-8.
- accounts Project View:** Shows the batchProcessCSVaccounts flow. The flow starts with an 'On New or Updated File accounts.csv' component, followed by a 'Transform Message CSV to Java' component with a red breakpoint icon, and a 'Set Variable size' component. The flow then enters a 'accountsBatch\_Job' scope, which contains a 'Batch\_Step' component with a 'Set Variable cname' action, a 'Logger' component, and an 'Aggregator' component. The 'Batch\_Step1' component is shown as an aggregating logger.

26. Step to the Logger in the first batch step.
27. In the Mule Debugger view, look at the value of the payload; it should be a HashMap.
28. Expand Variables; you should see the size variable and the cname variable specific for that record.

The screenshot shows the Mule Debugger window at the top, displaying the state of the payload variable. Below it is the Mule Studio interface, specifically the 'batchProcessCSVaccounts' flow. The flow starts with an 'On New or Updated File accounts.csv' connector, followed by a 'Transform Message CSV to Java' component, and a 'Set Variable size' step. This is followed by the 'accountsBatch\_Job' job, which contains a 'Batch\_Step'. Inside the 'Batch\_Step', there is a 'Set Variable cname' step and a 'Logger' step, which is highlighted with a dashed box. The 'Logger' step is connected to an 'Aggregator' and then to a final 'Batch' connector.

```

Mule Debugger
Logger = Logger
attributes = null
correlationId = "756e5970-8c71-11e9-8d01-f018983d9329"
payload = {HashMap} size = 6
vars = {Map} size = 4
_mule_batch_INTERNAL_record = {Record} com.mulesoft.mule.runtime.module.batch.api.record.Record@8f06430
batchJobInstanceId = "e545a2d0-8c71-11e9-8d01-f018983d9329"
cname = "Dog Park Industries"
size = 3

accounts X global config.yaml

batchProcessCSVaccounts
accountsBatch_Job
 Process Records
 Batch_Step
 Set Variable cname
 LOG
 Aggregator
 Batch

```

29. Step through the rest of the records in the first batch step and watch the payload and the cname variable change.

The screenshot shows the Mule Debugger window again, but now the payload variable has changed. It is still a HashMap, but its size is now 4. The 'vars' map also contains four entries: '\_mule\_batch\_INTERNAL\_record', 'batchJobInstanceId', 'cname', and 'size'. The 'cname' entry is now set to "Iguana Park Industries".

```

Logger = Logger
attributes = null
correlationId = "756e5970-8c71-11e9-8d01-f018983d9329"
payload = {HashMap} size = 6
vars = {Map} size = 4
_mule_batch_INTERNAL_record = {Record} com.mulesoft.mule.runtime.n
batchJobInstanceId = "e545a2d0-8c71-11e9-8d01-f018983d9329"
cname = "Iguana Park Industries"
size = 3

```

30. Step into the second batch step and look at the payload and the cname variable; you should see the cname variable is defined and has a value.

The screenshot shows the Mule Debugger interface. In the top window, a variable named 'cname' is highlighted in blue, containing the value "Dog Park Industries". Below it, the variable 'size' is shown with a value of 3. The bottom window displays a process flow diagram for an 'accounts' application. The flow starts with a 'Transform Message CSV to Java' component, followed by a 'Set Variable size' component. This leads into the 'accountsBatch\_Job' job, which contains a 'Batch\_Step' and a 'Batch\_Step1'. The 'Batch\_Step' includes a 'Set Variable cname' component and a 'Logger' component. The 'Batch\_Step1' also includes a 'Logger' component. The flow continues through an 'Aggregator' and ends with an 'On Complete' phase, which also features a 'Logger' component.

31. Step through the rest of the records in the second batch step and watch the value of cname.

32. Step into the On Complete phase; you should see the payload is an ImmutableBatchJobResult.

33. Expand the payload and locate the values for totalRecords, successfulRecords, and failedRecords.

The screenshot shows the Mule Debugger interface. The payload structure is expanded, showing fields like mediaType, batchJobInstanceId, elapsedTimeInMillis, failedOnCompletePhase, failedOnInputPhase, failedOnLoadingPhase, failedRecords, inputPhaseException, loadedRecords, loadingPhaseException, onCompletePhaseException, processedRecords, serialVersionUID, stepResults, successfulRecords, and totalRecords. The totalRecords value is highlighted. Below the debugger, a process flow diagram titled 'accounts' is shown. It starts with an 'Aggregator' component, followed by a 'Batch\_Step1' component containing a 'Logger' node. An 'On Complete' section follows, also containing an 'Aggregator' component with a 'Logger' node.

34. Step through the rest of the application and switch perspectives.

35. Stop the project.

## Look at the processing threads

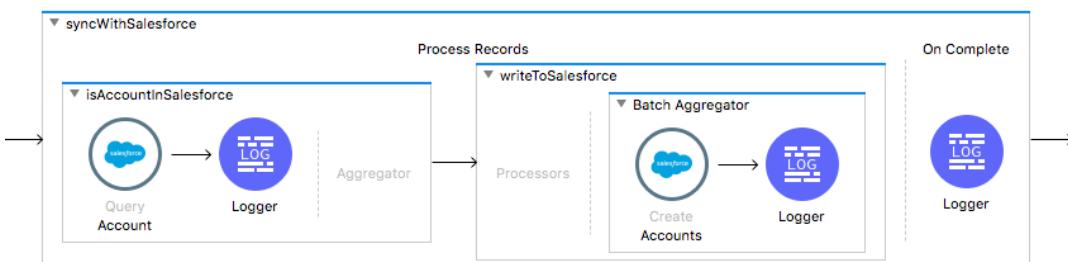
36. In the console, locate the thread number used to process each record in the collection in each step of the batch process; you should see more than one thread used.

The screenshot shows the Mule Server 4.2.0 EE console. It displays log messages from the 'apdev-examples' application. The first message is an INFO log at 2019-06-11 15:55:10,558, indicating the start of execution for the 'batchProcessCSVaccounts' step. The second message is another INFO log at 2019-06-11 15:55:10,596, showing the 'batch-job-accountsBatch' processor starting execution.

## Walkthrough 13-3: Use filtering and aggregation in a batch step

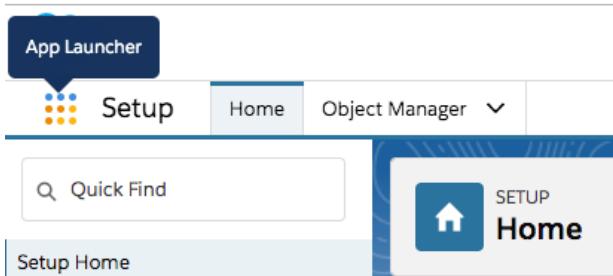
In this walkthrough, you use a batch job to synchronize account database records to Salesforce. You will:

- Use a batch job to synchronize database records (with your postal code) to Salesforce.
- In a first batch step, check to see if the record already exists in Salesforce.
- In a second batch step, add the record to Salesforce.
- Use a batch step filter so the second batch step is only executed for specific records.
- Use a Batch Aggregator scope to commit records in batches.

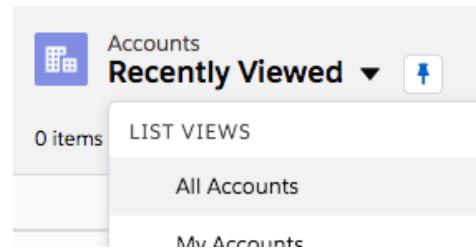


### Look at existing Salesforce account data

1. In a web browser, navigate to <http://login.salesforce.com/> and log in with your Salesforce Developer account.
2. Click the App Launcher then click the Accounts link



3. In the view drop-down menu, select All Accounts.

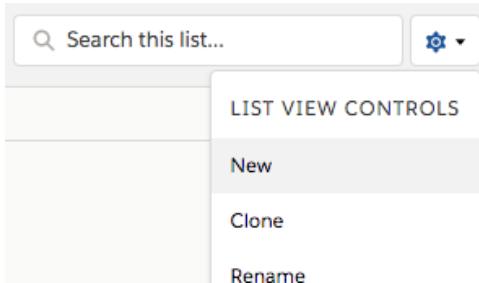


4. Look at the existing account data; a Salesforce Developer account is populated with some sample data.

The screenshot shows a list view of accounts. At the top, there's a header with a grid icon, the word "Accounts", a dropdown menu set to "All Accounts", and a search icon. Below the header, it says "12 items · Sorted by Account Name · Filtered by all accounts · Updated a minute ago". The main area is a table with four columns: ACCOUNT NAME, ACCOUNT SITE, BILLING STATE/PROVINCE, and PHONE. The data rows are:

|   | ACCOUNT NAME ↑                                      | ACCOUNT SITE | BILLING STATE/PROVINCE | PHONE          |
|---|-----------------------------------------------------|--------------|------------------------|----------------|
| 1 | <a href="#">Burlington Textiles Corp of America</a> |              | NC                     | (336) 123-4567 |
| 2 | <a href="#">Dickenson plc</a>                       |              | KS                     | (785) 123-4567 |
| 3 | <a href="#">Edge Communications</a>                 |              | TX                     | (512) 123-4567 |

5. Notice that countries and postal codes are not displayed by default.
6. In the List View Controls drop-down menu, select New.

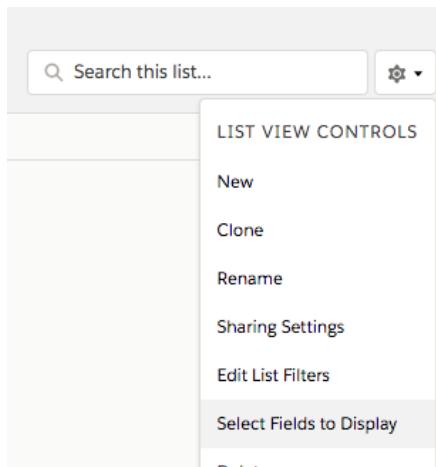


7. In the New List View dialog box, set the list name to All Accounts with Postal Code.
8. Select All users can see this list view.

The screenshot shows the "New List View" dialog box. It has two main sections: "List Name" and "List API Name". The "List Name" field contains "All Accounts with Postal Code". The "List API Name" field contains "All\_Accounts\_with\_Postal\_Code". Below these, there's a section titled "Who sees this list view?" with three radio button options: "Only I can see this list view", "All users can see this list view", and "Share list view with groups of users". The second option is selected. At the bottom right of the dialog are "Cancel" and "Save" buttons.

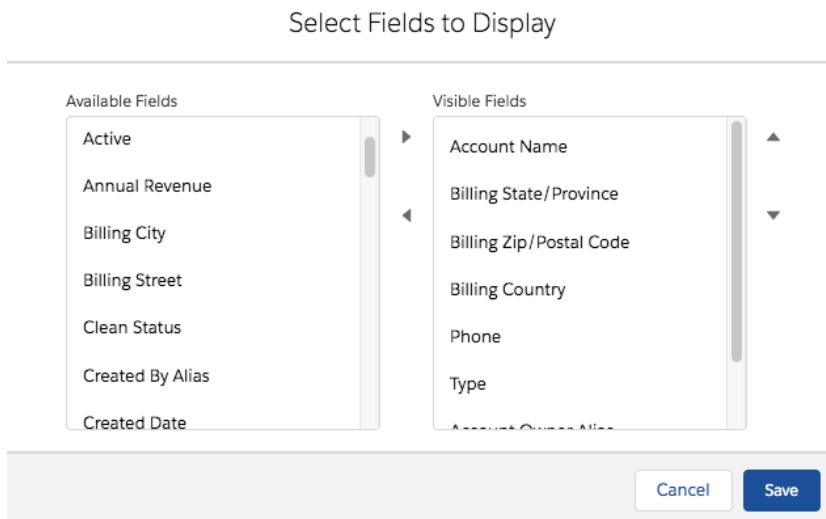
9. Click Save.

10. In the List View Controls drop-down menu, select Select Fields to Display.



11. In the Select Fields to Display dialog box, select Billing Zip/Postal Code as the available field and click the Move selection to Visible Fields button.

12. Select Billing Country as the available field and click the Move selection to Visible Fields button.
13. Use the Up and Down buttons to order the fields as you prefer.



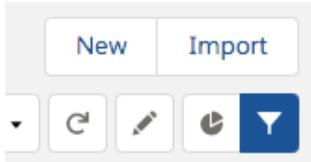
14. Click Save; you should now see all the accounts with postal codes and countries.

15. Adjust the column widths as you prefer.

|   | ACCOUNT NAME                        | BILLING STATE... | BILLING ZIP... | BILLING CO... | PHONE       |
|---|-------------------------------------|------------------|----------------|---------------|-------------|
| 1 | Burlington Textiles Corp of America | NC               | 27215          | USA           | (336) 222-7 |
| 2 | Dickenson plc                       | KS               | 66045          | USA           | (785) 241-6 |
| 3 | Edge Communications                 | TX               |                |               | (512) 757-6 |
| 4 | Express Logistics and Transport     | OR               |                |               | (503) 421-7 |
| 5 | GenePoint                           | CA               |                |               | (650) 867-3 |

## Add an account to Salesforce with a name matching one of the database records

16. Click the New button.



17. In the New Account dialog box, enter an account name (one that matches one of the accounts you added with your postal code to the database) and click Save.

New Account

Account Information

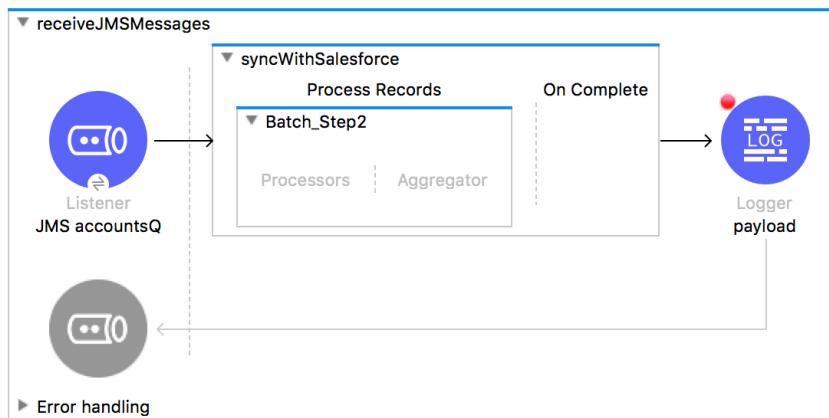
|                                   |                    |
|-----------------------------------|--------------------|
| Account Owner<br>Max Mule         | Rating<br>--None-- |
| * Account Name<br>Maximilian Mule | Phone              |
| Parent Account                    | Fav                |

18. Leave this browser tab open.

The screenshot shows the Salesforce interface with the 'Accounts' tab selected. The account record for 'Maximilian Mule' is displayed, showing basic information like Type, Phone, Website, Account Owner (Max Mule), Account Site, and Industry. Below the main details, there are tabs for 'Related', 'Details', and 'News'. Under the 'Related' tab, it says 'We found no potential duplicates of this account.' and lists 'Contacts (0)' and 'Opportunities (0)'.

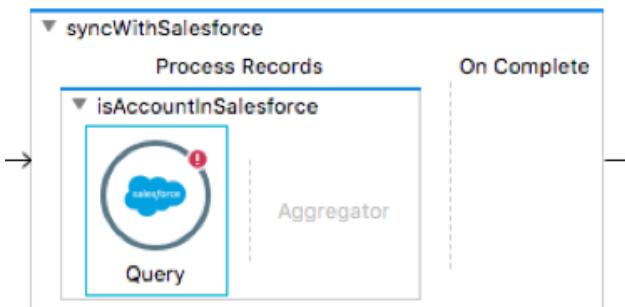
## Add a Batch Job scope to the receiveJMSmessages flow

19. Return to accounts.xml in Anypoint Studio.
20. Drag a Batch Job scope from the Mule Palette and drop it before the Logger in the receiveJMSmessages flow.
21. Change the display name of the batch job to syncWithSalesforce.



## Query Salesforce to see if an account already exists

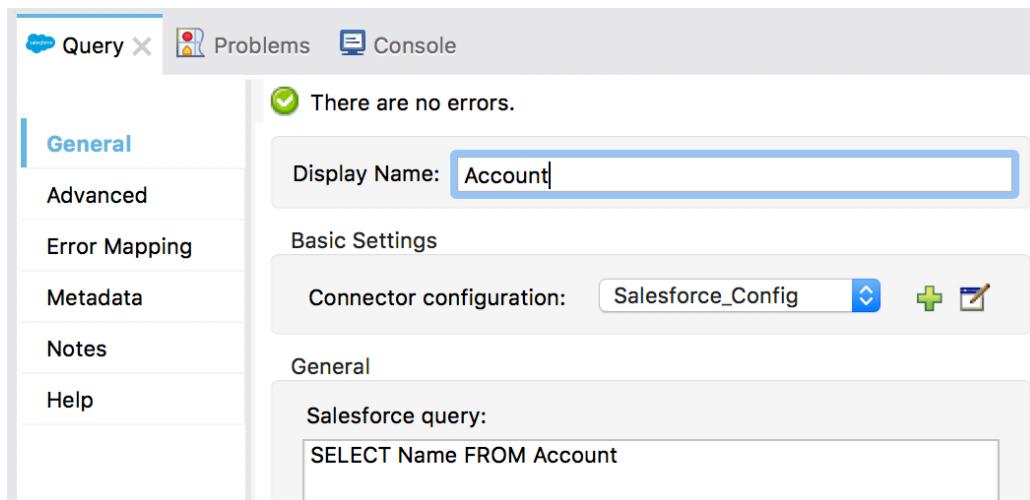
22. Change the name of the batch step inside the batch job to isAccountInSalesforce.
23. Drag a Salesforce Query operation from the Mule Palette and drop it in the processors phase in the batch step.



24. In the Query properties view, set the following values:

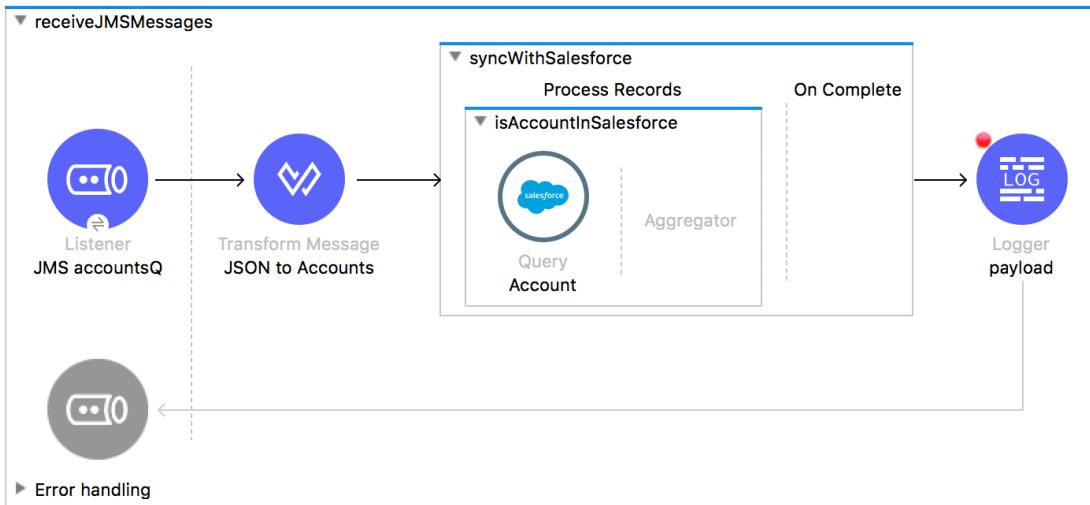
- Display Name: Account
- Connector configuration: Salesforce\_Config
- Salesforce query: SELECT Name FROM Account

*Note: You will add to this query shortly.*

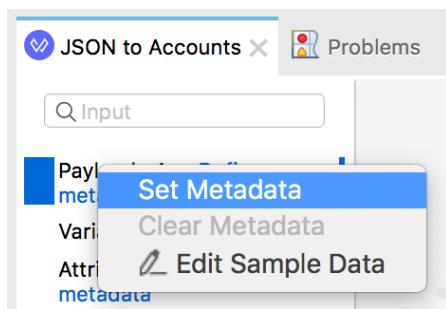


## Transform the input JSON data to Salesforce Account objects

25. Add a Transform Message component before the Batch Job.
26. Change the display name to JSON to Accounts.

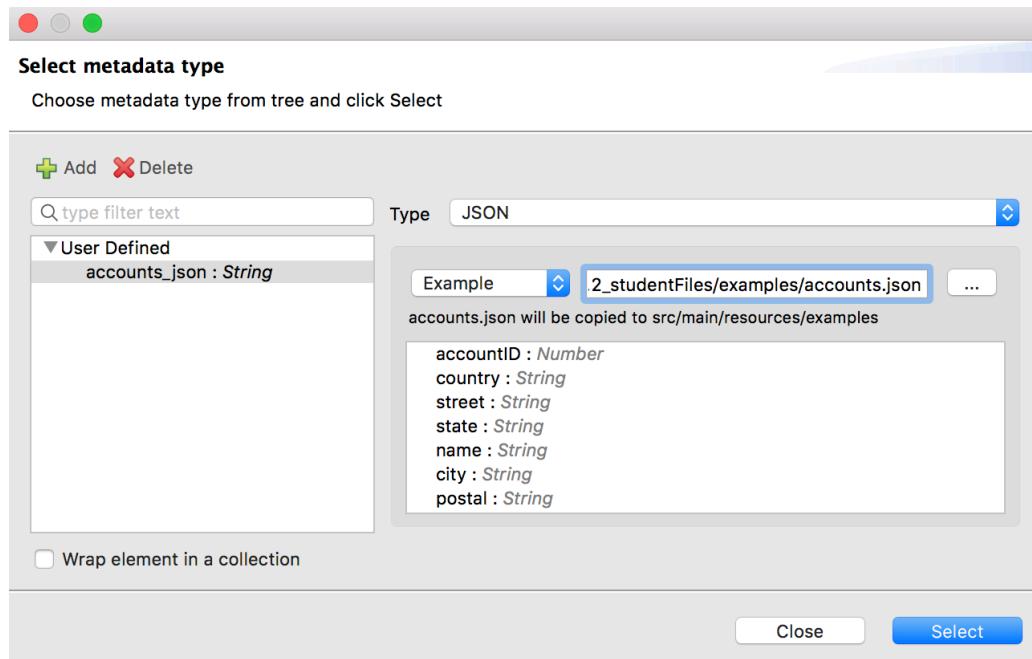


27. In the Transform Message properties view, look at the metadata in the input section.
28. Right-click Payload and select Set Metadata.



29. Create a new metadata type with the following information:

- Type id: accounts\_json
- Type: JSON
- Example: Use the accounts.json file in the resources/examples folder of the student files



30. In the Transform Message properties view, map the following fields:

- name: Name
- street: BillingStreet
- city: BillingCity
- state: BillingState
- postal: BillingPostalCode
- country: BillingCountry

*Note: If you do not get Salesforce metadata for the Account object, you can copy the DataWeave transformation from the course snippets.txt file.*

The screenshot shows the 'JSON to Accounts' DataWeave transformation. At the top, there are tabs for 'JSON to Accounts' (selected), 'Problems', and 'Console'. The main area has two sections: 'Input' on the left and 'Output' on the right. The 'Input' section shows a tree view of 'Payload : Array<Object>' with fields: 'accountID : Number', 'country : String', 'street : String', 'state : String', 'name : String', and 'city : String'. The 'Output' section shows the resulting structure: 'Undefined Define metadata'. Below the output is the DataWeave code:

```
1@dw 2.0
2 output application/java
3 ---
4@ payload map (payload01 , indexOfPayload01) -> {
5@ Name: payload01.name,
6@ BillingStreet: payload01.street,
7@ BillingCity: (payload01.city default ""),
8@ BillingState: payload01.state,
9@ BillingPostalCode: payload01.postal,
10@ BillingCountry: payload01.country
11 }
```

## Finish the batch step to check if an account already exists in Salesforce

31. In the Query properties view, add an input parameter named cname.
32. Set the parameter value to payload.Name, ensure it is a String, and give it a default value.

```
payload.Name default "" as String
```

33. Modify the Salesforce query to look for accounts with this name.

```
SELECT Name FROM Account
WHERE Name= ':cname'
```

Account X Problems Console

General

Salesforce query:

```
SELECT Name FROM Account
WHERE Name= ':cname'
```

Parameters

| Name    | Value                             |
|---------|-----------------------------------|
| "cname" | payload.Name default "" as String |

There are no errors.

## Store the result in a variable instead of overriding the payload

34. Select the Advanced tab.
35. Set the target variable to exists.
36. Set the target value to

```
#[(sizeOf(payload as Array) > 0)]
```

Account X Problems Console

General

Advanced

Error Mapping

Metadata

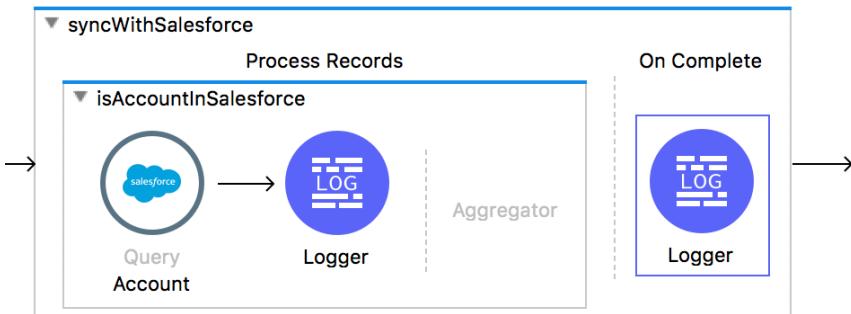
Output

Target Variable: exists

Target Value: #[(sizeOf(payload as Array) > 0)]

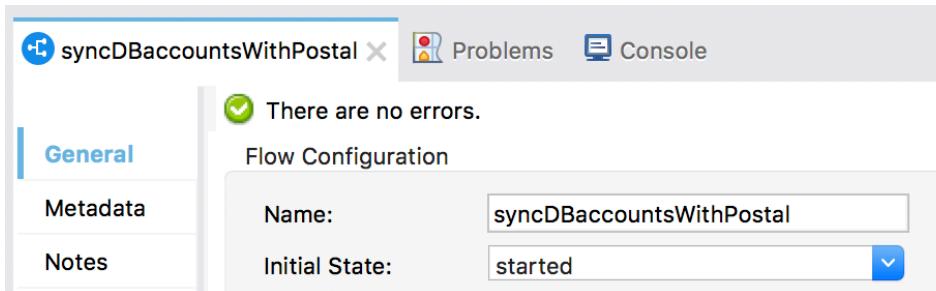
There are no errors.

37. Add a Logger after the Query operation.
38. Add a Logger to the On Complete phase.



## Change the initial state of the flow

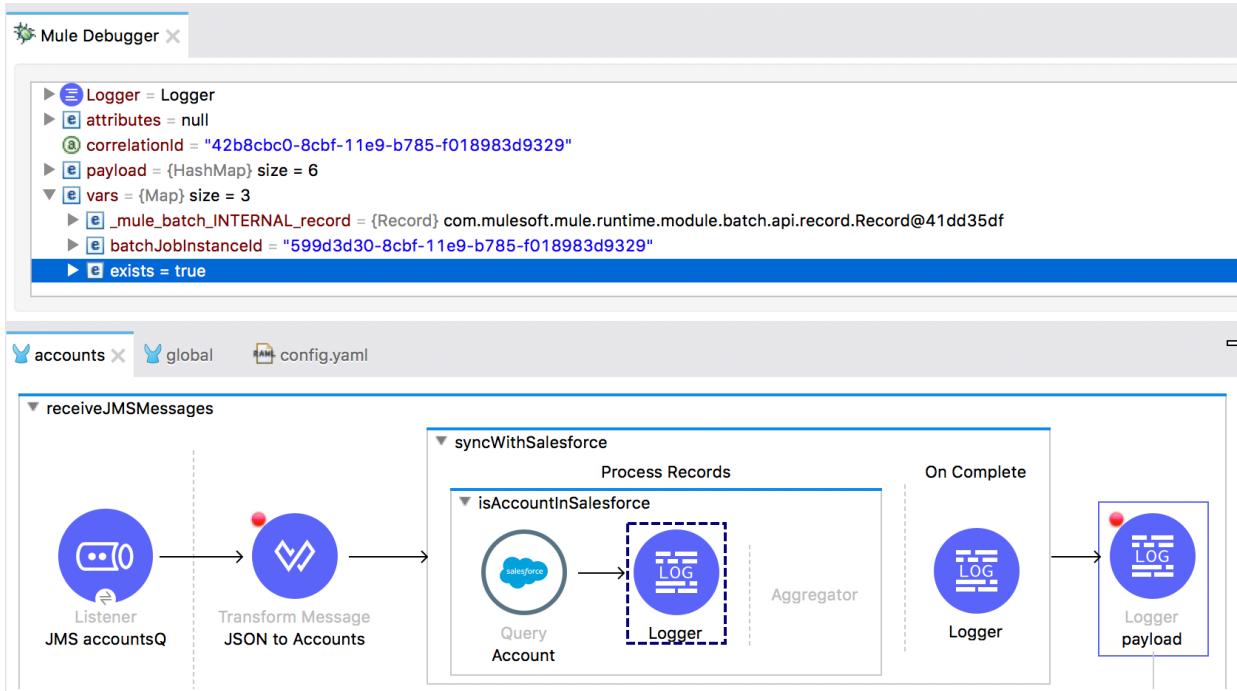
39. In the properties view for the syncDBaccountsWithPostal flow, change the initial state to started.



## Debug the application

40. Add a breakpoint to the Transform Message component before the batch job.
41. Save the file and debug the project.
42. Clear the application data.
43. In the Mule Debugger, wait until application execution stops in the receiveJMSmessages flow.

44. Step to the Logger in the first batch step and expand Variables; you should see the exists variable set to true or false.



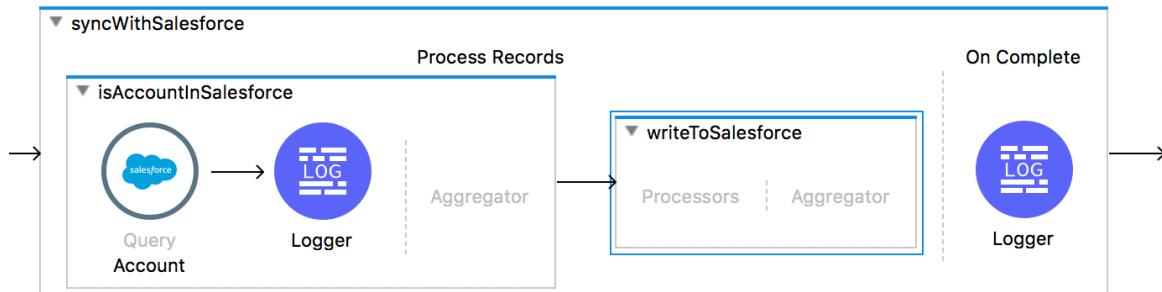
45. Step through the application; you should see the exists variable set to false for records with names that don't exist in Salesforce and true for those that do.

46. Stop the project and switch perspectives.

## Set a filter for the insertion step

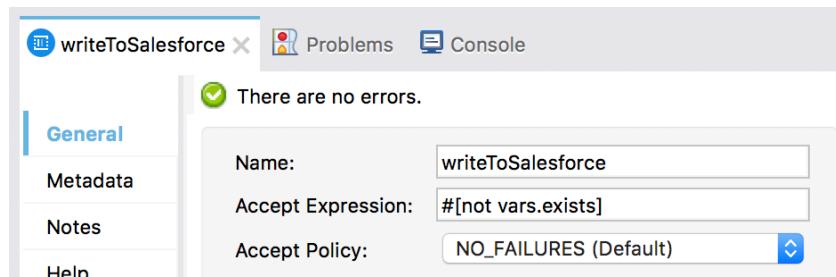
47. Add a second batch step to the batch job.

48. Change its display name to writeToSalesforce.



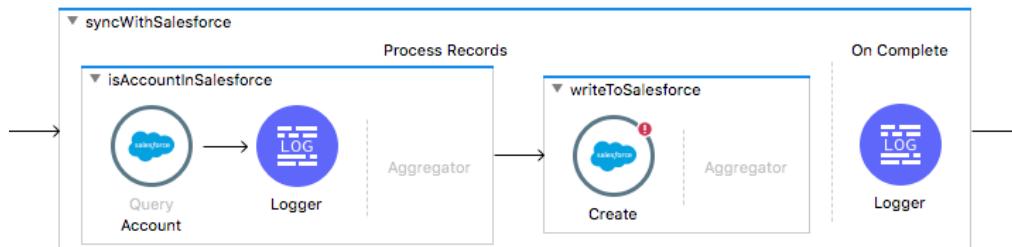
49. In the properties view for the second batch step, set the accept expression so that only records that have the variable exists set to false are processed.

```
##[not vars.exists]
```



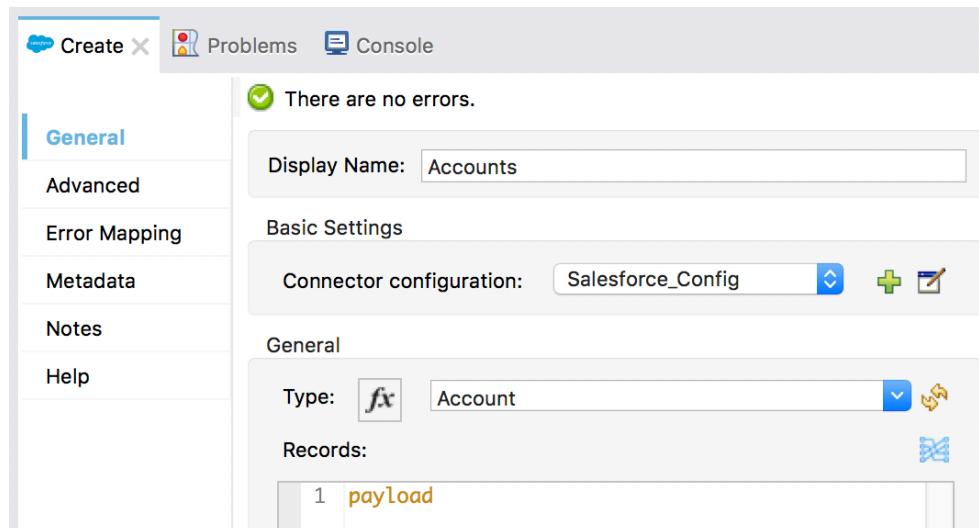
## Use the Salesforce Create operation to add new account records to Salesforce

50. Add Salesforce Create operation to the second batch step in the processors phase.



51. In the Create properties view, set the following:

- Display Name: Accounts
- Connector configuration: Salesforce\_Config
- Type: Account
- Records: payload



52. Select the Input tab in the DataSense Explorer; you should see this operation expects an Array of Account objects – not just one.

Input   Output

Q type filter text

▼ Mule Message

  ▼ Payload

    ► **(Actual)** Object : Object

    ► **(Expected)** Array<Object> : Array<Object>

  ▼ Attributes

    Void : Void

  ▼ Variables

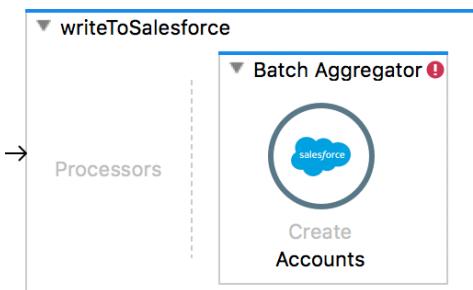
    ▼ exists

      Boolean : Boolean

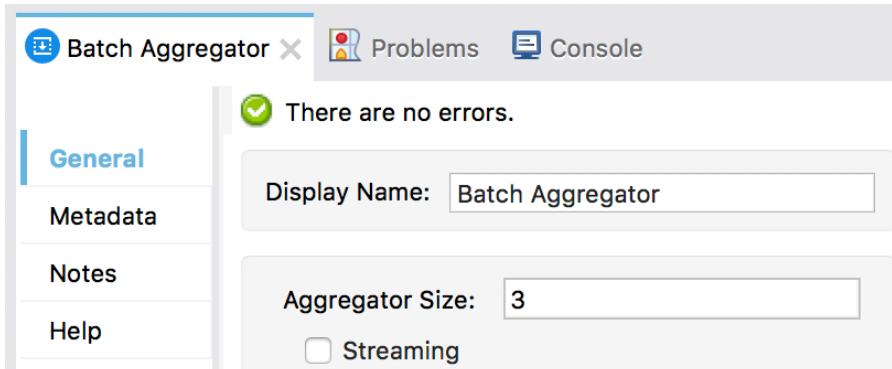
### Create the Array of objects that the operation is expecting

53. Drag a Batch Aggregator scope from the Mule Palette and drop it in the aggregator phase of the second batch step.

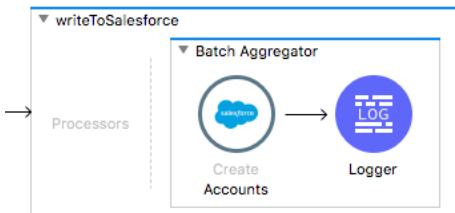
54. Move the Salesforce Create operation into the Batch Aggregator.



55. In the Batch Aggregator properties view, set the aggregator size to 3.



56. Add a Logger after the Create operation.



## Test the application

57. Save the file, debug the project, and clear the application data.

58. Step through the application until you step into the Batch Aggregator.

59. Expand Payload.

The screenshot shows the Mule Debugger interface. The top window displays the payload structure:

```
payload = {ImmutableRecordAwareList} size = 3
 0 = {HashMap} size = 6
 1 = {HashMap} size = 6
 2 = {HashMap} size = 6
 0 = {HashMap$Node} BillingCountry=United States
 1 = {HashMap$Node} BillingStreet=415 Mission St
 2 = {HashMap$Node} BillingCity=San Francisco
 3 = {HashMap$Node} BillingPostalCode=94105
 4 = {HashMap$Node} BillingState=CA
 5 = {HashMap$Node} Name=Lois Lane
```

The bottom window shows the Mule flow diagram:

```
graph LR; subgraph writeToSalesforce [writeToSalesforce]; subgraph BatchAggregator [Batch Aggregator]; Create[Create Accounts] --> Logger[Logger]; end;
```

60. Step through the rest of the flow.
61. Return to Salesforce and look at the accounts; you should see the records from the legacy MySQL database are now in Salesforce.

*Note: You could also check for the records by making a request to <http://localhost:8081/sfdc>.*

|   | ACCOUNT NAME             | BILLING STATE... | BILLING Z... | BILLING CO... | PE  |
|---|--------------------------|------------------|--------------|---------------|-----|
| 1 | Lois Lane                | CA               | 94105        | United States |     |
| 2 | Clark Kent               | CA               | 94105        | United States |     |
| 3 | Maxine Mule              | CA               | 94105        | United States |     |
| 4 | sForce                   | CA               | 94087        | US            | (4) |
| 5 | Pyramid Construction Inc |                  | 75251        | France        | (0) |

62. Run the application again but do not clear the application data; no records should be processed.
63. Return to salesforce.com and locate your new record(s); they should have been inserted only once.
64. Return to Anypoint Studio and stop the project.