



MuleSoft®

Anypoint Platform Development: API Design

Student Manual

May 5, 2017

Table of Contents

INTRODUCING THE COURSE.....	4
Walkthrough: Set up your computer for class	5
PART 1: DESIGNING APIs	7
MODULE 1: INTRODUCING RESTFUL API DESIGN	8
Walkthrough 1-1: Review and make a call to a Public SOAP API	9
Walkthrough 1-2: Review and make a call to an RPC API	13
Walkthrough 1-3: Review and make a call to a REST API	16
MODULE 2: TRANSLATING FUNCTIONAL REQUIREMENTS FOR APIs	18
Walkthrough 2-1: List the categories and actions for an API	19
Walkthrough 2-2: Translate categories and actions into resources and methods	21
MODULE 3: INTRODUCING API-LED CONNECTIVITY AND THE API LIFECYCLE ...	24
Walkthrough 3-1: Explore API Manager in Anypoint Platform	25
PART 2: DEFINING APIs WITH THE RESTFUL API MODELING LANGUAGE (RAML)	27
MODULE 4: DEFINING API RESOURCES AND METHODS.....	28
Walkthrough 4-1: Create an API and define resources in RAML 1.0	29
Walkthrough 4-2: Define methods for the resources	34
Walkthrough 4-3: Specify URI parameters for necessary resource methods.....	40
MODULE 5: SPECIFYING RESPONSES.....	44
Walkthrough 5-1: Add HTTP 2xx responses to resource methods.....	45
Walkthrough 5-2: Add responses bodies to return error status codes and messages for client-side errors.....	52
Walkthrough 5-3: Add response bodies to return error status codes and messages for server-side errors.....	57
Walkthrough 5-4: Add flexible content-types to HTTP responses	62
Walkthrough 5-5: Add caching information to HTTP methods.....	68
MODULE 6: MODELLING DATA	72
Walkthrough 6-1: List datatypes and their attributes for an API	73
Walkthrough 6-2: Create datatype fragments	79
Walkthrough 6-3: Validate datatype attribute values using patterns.....	89
Walkthrough 6-4: Specify datatypes in resource methods	94
Walkthrough 6-5: Define example fragments for datatypes.....	100

MODULE 7: DOCUMENTING AND TESTING APIS	108
Walkthrough 7-1: Add a documentation fragment to a RAML API definition	109
Walkthrough 7-2: Add description nodes to a RAML API definition.....	116
Walkthrough 7-3: Use the mocking service in API Console to test an API.....	124
MODULE 8: MAKING APIS DISCOVERABLE	130
Walkthrough 8-1: Create and customize an API Portal	131
Walkthrough 8-2: Create a sample use case with API Notebook inside the API Portal	140
Walkthrough 8-3: Publish a RAML API definition to Anypoint Exchange	151
MODULE 9: REUSING PATTERNS	156
Walkthrough 9-1: Define and use a resource type for resources that perform operations on a collection	157
Walkthrough 9-2: Define and use a resource type for resources that perform operations on a member referenced with an ID	163
Walkthrough 9-3: Refactor resources to use resource types.....	169
Walkthrough 9-4: Define and use various traits for resources and methods	172
MODULE 10: MODULARIZING APIS.....	179
Walkthrough 10-1: Create and use a library of traits	180
Walkthrough 10-2: Internationalize documentation and resource description using an overlay	188
Walkthrough 10-3: Define and use API extensions to promote portability to test in multiple environments	191
MODULE 11: SECURING APIS	195
Walkthrough 11-1: Define a custom security scheme for an API	196
Walkthrough 11-2: Define a security scheme for an API and secure API resources.....	203
MODULE 12: ENHANCING API RESPONSES USING HYPERMEDIA	213
Walkthrough 12-1: Modify an API definition to generate state-specific client responses	214
MODULE 13: VERSIONING APIS	222
Walkthrough 13-1: Add a new API version	223

Introducing the Course

The image shows two screenshots of the MuleSoft API Manager interface. The left screenshot displays the API Administration screen with a search bar and a button to 'Add new API'. Below it, a code editor shows RAML 1.0 code for the ACME Banking API, defining resources like /customers and /accounts with various HTTP methods. The right screenshot shows the API reference documentation for the ACME Banking API - 1.0, featuring sections for Documentation, Resources, and a detailed view of the /customers endpoint with its supported operations (POST, GET, DELETE, PATCH).

```
%RAML 1.0
title: ACME Banking API
version: 1.0

/customers:
  get:
  post:
    /{customer_id}:
      get:
      patch:
      delete:
        /accounts:
          get:
            /{account_id}:
              post:
                /{account_id}:
                  get:
```

Objectives:

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

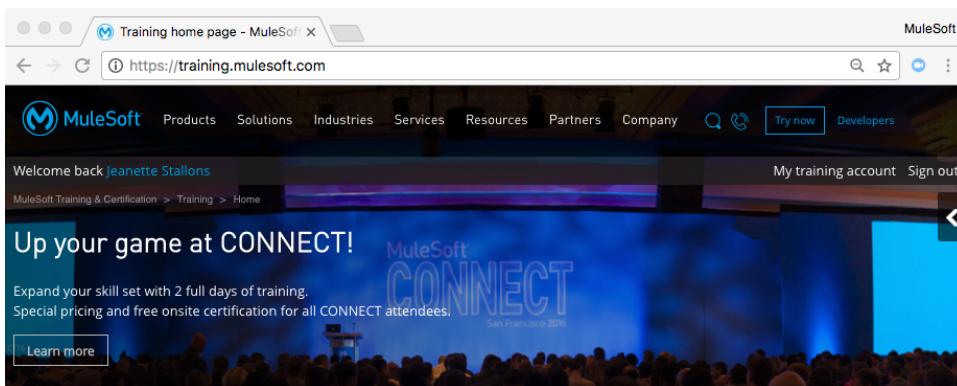
Walkthrough: Set up your computer for class

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.

Download student files

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



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 box 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.

5. Click the event's View Details button.

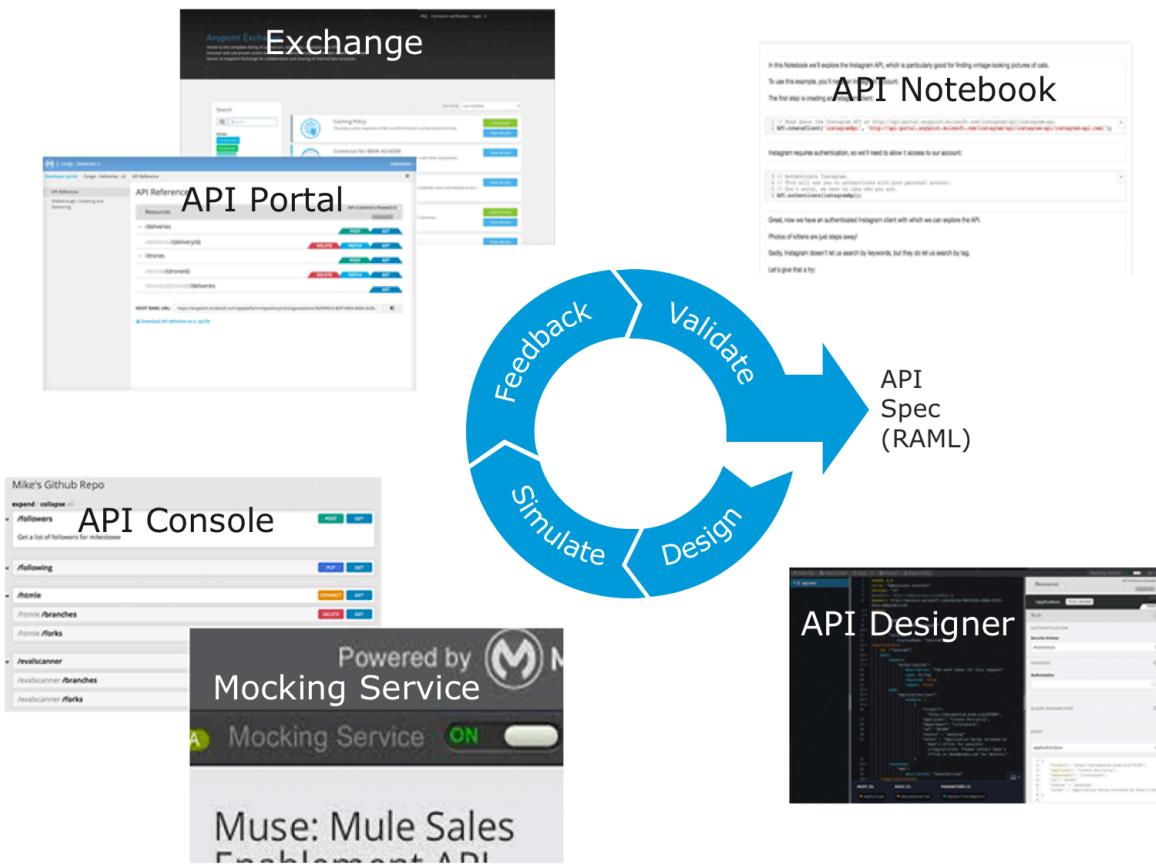
- Locate the list of course materials on the right-side of the page.

The screenshot shows a course page for 'CONNECT 2017 - API Design'. At the top, there's a navigation bar with links for Dashboard, Classes, Catalog, Support, Cart (with a red notification badge), Inbox, and a user profile icon. A green button labeled 'Confirmed' is visible in the top right. The main title 'CONNECT 2017 - API Design' is centered above a subtitle 'Apr 18, 8:00 AM - 5:00 PM PDT' and the location 'Marriott Marquis Hotel, 780 Mission St, San Francisco, US'. Below the title, a note states: 'This course is for developers, architects, and API designers who want to get hands-on experience designing APIs and defining them with RAML, the RESTful API Modeling Language. Students will:'. A bulleted list follows: 'Translate design requirements into API resources and methods.', 'Use API Designer to create API definitions.', 'Use RAML to define API resources, methods, parameters, and responses.', 'Minimize repetition in API definitions using resource types and traits.', 'Document and test APIs.' To the right, a box says 'No instructors assigned.' and another section titled 'MATERIALS' lists three files: 'CONNECT2017_ApiDesign3.8 Student Files (ZIP) 1.29MB ZIP', 'CONNECT2017_ApiDesign3.8 Student Slides (ZIP) 14.38MB ZIP', and 'CONNECT2017_ApiDesign3.8 Student Manual (PDF) 53.41MB PDF'.

- 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.

PART 1: Designing APIs



Objectives:

- Describe REST API architecture.
- Describe API development lifecycle.
- Translate functional requirements for APIs into resources and HTTP methods.
- Navigate Anypoint Platform.

Module 1: Introducing RESTful API Design

SOAP	RPC	REST
<pre><wsdl:definitions xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soap="http://schemas.xmlsoap.org/soap/envelope"> <wsdl:types> <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" targetNamespace="http://soap.training" elementFormDefault="qualified"> <xs:element name="findFlight" type="xs:string"/> <xs:element name="findFlightResponse" type="xs:string"/> <xs:element name="listAllFlights" type="xs:string"/> <xs:element name="listAllFlightsResponse" type="xs:string"/> </xs:schema> </wsdl:types> <wsdl:message name="findFlight"> <wsdl:part name="parameters" type="tns:findFlight"/> </wsdl:message> <wsdl:message name="findFlightResponse"> <wsdl:part name="parameters" type="tns:findFlightResponse"/> </wsdl:message> <wsdl:message name="listAllFlights"> <wsdl:part name="parameters" type="tns:listAllFlights"/> </wsdl:message> <wsdl:message name="listAllFlightsResponse"> <wsdl:part name="parameters" type="tns:listAllFlightsResponse"/> </wsdl:message> <wsdl:operation name="findFlight"> <wsdl:input message="findFlight"/> <wsdl:output message="findFlightResponse"/> </wsdl:operation> <wsdl:operation name="listAllFlights"> <wsdl:input message="listAllFlights"/> <wsdl:output message="listAllFlightsResponse"/> </wsdl:operation> </wsdl:definitions></pre>	<pre>POST /SendUserMessage HTTP/1.1 Host: api.example.com Content-type: application/json {"userId": 501, "message": "Hello!"}</pre>	<pre>%RAML 1.0 title: ACME Banking API version: 1.0 /customers: get: post: /{customer_id}: get: patch:</pre>

Objectives:

- Describe the common web API formats including SOAP, RPC and REST.
- Describe REST API architecture.
- List the rules for retaining REST principles in APIs.
- Describe design-first approach for REST APIs.

Walkthrough 1-1: Review and make a call to a Public SOAP API

In this walkthrough, you explore and understand a SOAP API. You will:

- Examine an example SOAP API.
- Make a call to a SOAP API endpoint to retrieve information.

The screenshot shows a web browser window with the URL www.webservicex.net/new/Home/Index. The page title is "WEBSERVICEX.NET". Below the title, there's a main heading "Explore services for all devices" with a subtext explaining the Webservicex.NET Data Protocol. A green "Explore" button is visible. To the right, there's a "Demo" section with a link to a complete list of operations, followed by a "getAirportInformationByAirportCode" operation. This section includes a description of the operation, parameters (like airportCode: LHR), and an "Invoke" button. Below this, there's a "Webservices Directory" section with icons for Business and Commerce, Messaging, Standards and Lookup Data, Value Manipulation / Unit Converter, and a Test section with a sample SOAP 1.1 request and response. The "Test" section shows a POST request to /airport.asmx with various headers and parameters.

Examine the Airline Information SOAP API

1. Return to the course snippets.txt file.
2. Copy the URL for the SOAP API: <http://www.webservicex.net/new/Home/Index>.
3. In a web browser, navigate to that URL
4. In the Webservices Directory, click Standards and Lookup Data.

The screenshot shows a web browser window with the URL www.webservicex.net/new/Home/Index. The page title is "WEBSERVICEX.NET". Below the title, there's a main heading "Explore services for all devices" with a subtext explaining the Webservicex.NET Data Protocol. A green "Explore" button is visible. To the right, there's a "Webservices Directory" section with icons for Business and Commerce, Messaging, Standards and Lookup Data, Value Manipulation / Unit Converter, Graphics and Multimedia, Other Web Services, and Utilities.

5. In the list of Webservices by category, click the Airport Information Webservice.

Country Details
Get Currency, Currency code, International Dialing code, ISO country code for all countries

Airport Information Webservice
Get Airport Code, CityOrAirport Name, Country, Country Abbrv, CountryCode,GMT Offset Runway Length in Feet, Runway Elevation in Feet, Latitude in Degree, Latitude in Minute Latitude in Second, Latitude in N/S, Longitude in Degree, Longitude in Minute, Longitude in Seconds and longitude E/W for an Airport

NAICS and SIC Code
Get NAICS United States Structure, Including Relationship to 1987 U.S. SIC

Periodic Table
Get Atomic Number, Element Name, Symbol, Atomic Weight, Boiling Point, Ionisation Potential, Electro-negativity, Atomic Radius, Melting Point, Density

USA Zip code Information
Get State Code,City,Area Code,Time Zone,Zip Code

ICD-9-CM
ICD-9-CM CLASSIFICATION OF DISEASES AND INJURIES - The ICD9 coding system is an international classification system which groups related disease entities and procedures for the purpose of reporting statistical information. The system is widely used to for medical billing.

View the Airport Information Webservice endpoint WSDL

6. In the Airport Information Webservice Detail page, copy the link specified for the Endpoint.

WEBSERVICEEX.NET Home Webservices Contact

Airport Information Webservice Detail

Get Airport Code, CityOrAirport Name, Country, Country Abbrv, CountryCode,GMT Offset Runway Length in Feet, Runway Elevation in Feet, Latitude in Degree, Latitude in Minute Latitude in Second, Latitude in N/S, Longitude in Degree, Longitude in Minute, Longitude in Seconds and longitude E/W for an Airport

Endpoint

<http://www.webservicex.net/airport.asmx?WSDL>

7. In a new tab in the web browser, navigate to the Endpoint URL.
8. Locate the operations and HTTP methods supported by the webservice.

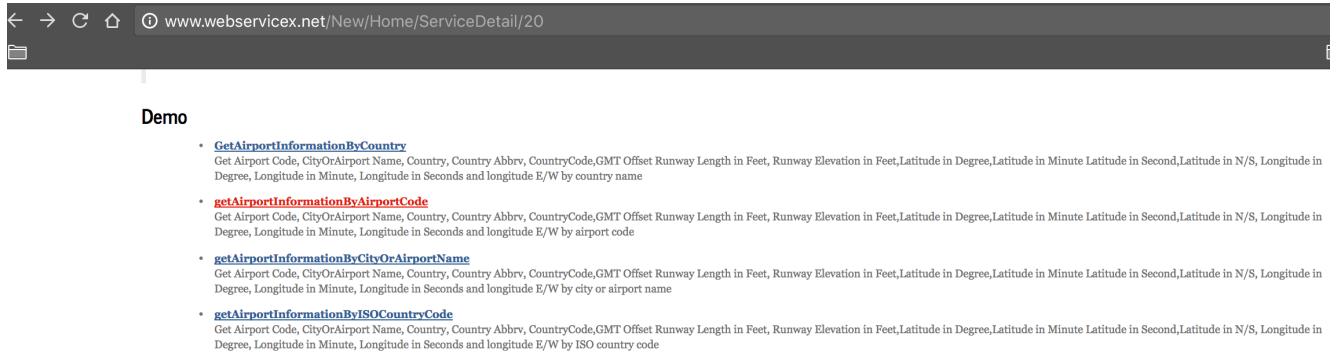
```

<wsdl:message name="GetAirportInformationByCountryHttpPostOut">
  <wsdl:part name="Body" element="tns:string"/>
</wsdl:message>
<wsdl:message name="GetAirportInformationByAirportCodeHttpPostIn">
  <wsdl:part name="airportCode" type="string"/>
</wsdl:message>
<wsdl:message name="GetAirportInformationByAirportCodeHttpPostOut">
  <wsdl:part name="Body" element="tns:string"/>
</wsdl:message>
<wsdl:portType name="airportSoap">
  <wsdl:operation name="getAirportInformationByISOCountryCode">
    <wsdl:documentation xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
      Get Airport Code, CityOrAirport Name, Country, Country Abbrv, CountryCode,GMT Offset Runway Length in Feet, Runway Elevation in Feet, Latitude in Degree, Latitude in Minute Latitude in Second, Latitude in N/S, Longitude in Degree, Longitude in Minute, Longitude in Seconds and longitude E/W by ISO country code
    </wsdl:documentation>
    <wsdl:input message="tns:getAirportInformationByISOCountryCodeSoapIn"/>
    <wsdl:output message="tns:getAirportInformationByISOCountryCodeSoapOut"/>
  </wsdl:operation>
  <wsdl:operation name="getAirportInformationByCityOrAirportName">
    <wsdl:documentation xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
      Get Airport Code, CityOrAirport Name, Country, Country Abbrv, CountryCode,GMT Offset Runway Length in Feet, Runway Elevation in Feet, Latitude in Degree, Latitude in Minute Latitude in Second, Latitude in N/S, Longitude in Degree, Longitude in Minute, Longitude in Seconds and longitude E/W by city or airport name
    </wsdl:documentation>
    <wsdl:input message="tns:getAirportInformationByCityOrAirportNameSoapIn"/>
    <wsdl:output message="tns:getAirportInformationByCityOrAirportNameSoapOut"/>
  </wsdl:operation>
  <wsdl:operation name="GetAirportInformationByCountry">
    <wsdl:documentation xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
      Get Airport Code, CityOrAirport Name, Country, Country Abbrv, CountryCode,GMT Offset Runway Length in Feet, Runway Elevation in Feet, Latitude in Degree, Latitude in Minute Latitude in Second, Latitude in N/S, Longitude in Degree, Longitude in Minute, Longitude in Seconds and longitude E/W by country name
    </wsdl:documentation>
    <wsdl:input message="tns:GetAirportInformationByCountrySoapIn"/>
    <wsdl:output message="tns:GetAirportInformationByCountrySoapOut"/>
  </wsdl:operation>
  <wsdl:operation name="GetAirportInformationByAirportCode">
    <wsdl:documentation xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
  </wsdl:operation>

```

Make a call to retrieve data from an Airport Information Webservice operation

9. Go back to the Airport Information Webservice Detail page.
10. In the list of operations under Demo, click getAirportInformationByAirportCode.

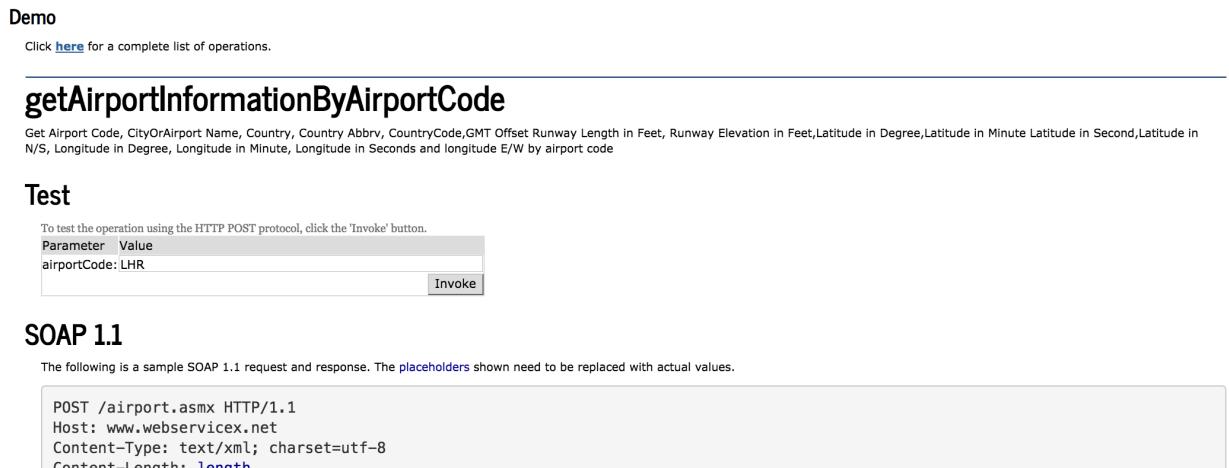


Demo

- **GetAirportInformationByCountry**
Get Airport Code, CityOrAirport Name, Country, Country Abbrv, CountryCode,GMT Offset Runway Length in Feet, Runway Elevation in Feet, Latitude in Degree, Latitude in Minute, Longitude in N/S, Longitude in Degree, Longitude in Minute, Longitude in Seconds and longitude E/W by country name
- **getAirportInformationByAirportCode**
Get Airport Code, CityOrAirport Name, Country, Country Abbrv, CountryCode,GMT Offset Runway Length in Feet, Runway Elevation in Feet, Latitude in Degree, Latitude in Minute, Latitude in Second, Latitude in N/S, Longitude in Degree, Longitude in Minute, Longitude in Seconds and longitude E/W by airport code
- **getAirportInformationByCityOrAirportName**
Get Airport Code, CityOrAirport Name, Country, Country Abbrv, CountryCode,GMT Offset Runway Length in Feet, Runway Elevation in Feet, Latitude in Degree, Latitude in Minute, Latitude in Second, Latitude in N/S, Longitude in Degree, Longitude in Minute, Longitude in Seconds and longitude E/W by city or airport name
- **getAirportInformationByISOCountryCode**
Get Airport Code, CityOrAirport Name, Country, Country Abbrv, CountryCode,GMT Offset Runway Length in Feet, Runway Elevation in Feet, Latitude in Degree, Latitude in Minute, Latitude in Second, Latitude in N/S, Longitude in Degree, Longitude in Minute, Longitude in Seconds and longitude E/W by ISO country code

11. In the Test parameter dialog box, type the airportCode parameter value as LHR.

12. Click Invoke.



Test

To test the operation using the HTTP POST protocol, click the 'Invoke' button.

Parameter	Value
airportCode:	LHR

Invoke

SOAP 1.1

The following is a sample SOAP 1.1 request and response. The **placeholders** shown need to be replaced with actual values.

```
POST /airport.asmx HTTP/1.1
Host: www.webservicex.net
Content-Type: text/xml; charset=utf-8
Content-Length: length
```

13. Verify that a new tab opens that displays the response received from the SOAP webservice operation.



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

```
<string xmlns="http://www.webservicex.NET">
<NewDataSet> <Table> <AirportCode>LHR</AirportCode> <CityOrAirportName>LONDON HEATHROW</CityOrAirportName> <Country>Great Britain (UK)</Country>
<CountryAbbrviation>GB</CountryAbbrviation> <CountryCode>493</CountryCode> <GMTOffset>0</GMTOffset> <RunwayLengthFeet>12802</RunwayLengthFeet>
<RunwayElevationFeet>80</RunwayElevationFeet> <LatitudeDegree>51</LatitudeDegree> <LatitudeMinute>28</LatitudeMinute> <LatitudeSecond>0</LatitudeSecond>
<LatitudeNpeers>N</LatitudeNpeers> <LongitudeDegree>0</LongitudeDegree> <LongitudeMinute>27</LongitudeMinute> <LongitudeSeconds>0</LongitudeSeconds>
<LongitudeNpeers>N</LongitudeNpeers> <LongitudeEperW>W</LongitudeEperW> </Table> <Table> <AirportCode>LHR</AirportCode> <CityOrAirportName>LONDON HEATHROW</CityOrAirportName> <Country>Great Britain (UK)</Country>
<CountryAbbrviation>GB</CountryAbbrviation> <CountryCode>493</CountryCode> <GMTOffset>0</GMTOffset>
<RunwayLengthFeet>12802</RunwayLengthFeet> <RunwayElevationFeet>80</RunwayElevationFeet> <LatitudeDegree>51</LatitudeDegree>
<LatitudeMinute>28</LatitudeMinute> <LatitudeSecond>0</LatitudeSecond> <LatitudeNpeers>N</LatitudeNpeers> <LongitudeDegree>0</LongitudeDegree>
<LongitudeMinute>27</LongitudeMinute> <LongitudeSeconds>0</LongitudeSeconds> <LongitudeEperW>W</LongitudeEperW> </Table> </NewDataSet>
</string>
```

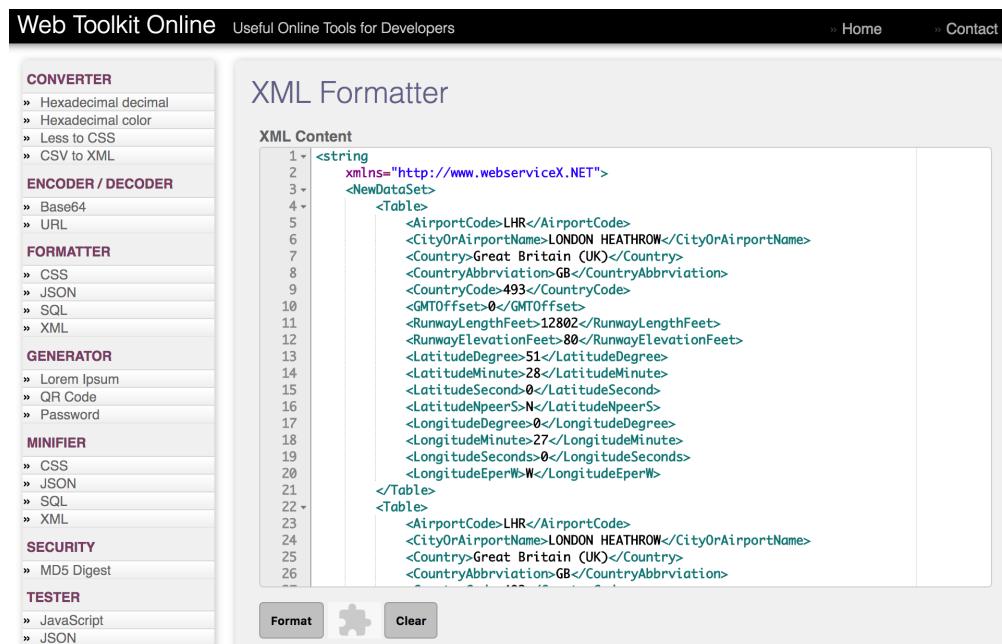
14. Copy the content after the horizontal line in the response webpage.

15. Return to the course snippets.txt file and copy the URL for the Web Toolkit Online
<http://www.webtoolkitonline.com/xml-formatter.html>.

16. In a new tab in the web browser, navigate to that URL.

17. Paste the content in the XML Content area.

18. Click Format.



The screenshot shows the 'XML Formatter' page of the Web Toolkit Online website. The top navigation bar includes links for Home and Contact. The left sidebar contains a menu with sections like CONVERTER, ENCODER / DECODER, FORMATTER, GENERATOR, MINIFIER, SECURITY, and TESTER, each with several sub-options. The main content area is titled 'XML Formatter' and contains a code editor labeled 'XML Content'. The code in the editor is:

```
1 <string
2   xmlns="http://www.webserviceX.NET"
3   <NewDataSet>
4     <Table>
5       <AirportCode>LHR</AirportCode>
6       <CityOrAirportName>LONDON HEATHROW</CityOrAirportName>
7       <Country>Great Britain (UK)</Country>
8       <CountryAbbreviation>GB</CountryAbbreviation>
9       <CountryCode>493</CountryCode>
10      <GMTOffset>-0</GMTOffset>
11      <RunwayLengthFeet>12802</RunwayLengthFeet>
12      <RunwayElevationFeet>80</RunwayElevationFeet>
13      <LatitudeDegree>51</LatitudeDegree>
14      <LatitudeMinute>28</LatitudeMinute>
15      <LatitudeSecond>0</LatitudeSecond>
16      <LatitudeNPeers>S</LatitudeNPeers>
17      <LongitudeDegree>0</LongitudeDegree>
18      <LongitudeMinute>27</LongitudeMinute>
19      <LongitudeSeconds>0</LongitudeSeconds>
20      <LongitudeEPerW>W</LongitudeEPerW>
21    </Table>
22  </Table>
23  <AirportCode>LHR</AirportCode>
24  <CityOrAirportName>LONDON HEATHROW</CityOrAirportName>
25  <Country>Great Britain (UK)</Country>
26  <CountryAbbreviation>GB</CountryAbbreviation>
```

Below the code editor are three buttons: 'Format' (highlighted in blue), 'Clear', and a puzzle-piece icon.

Walkthrough 1-2: Review and make a call to an RPC API

In this walkthrough, you explore an RPC API. You will:

- Examine the Slack RPC API.
- Make a call to an RPC endpoint.

The screenshot shows the Slack Web API documentation for the `channels.list` method. The left sidebar navigation includes sections like Start here, App features, Messages, and APIs (with Web API selected). The main content area is titled "Slack Web API" and describes the API's purpose. It shows examples of JSON responses for different success and failure cases. On the right, there's a "Related Articles" sidebar and a "Documentation" tab selected in a navigation bar. Below the main content, there's a form for testing the method, with fields for "token" (set to "No token") and "exclude_archived" (set to "Optional, default=false"). A "Test Method" button is at the bottom of the form.

Examine the Slack Web API

1. In a web browser, navigate to <https://api.slack.com/web>.

Note: This URL is also located in the course snippets.txt file.

The screenshot shows the Slack Web API documentation homepage. The left sidebar navigation includes sections like Start here, App features, Messages, and Unfurling links. The main content area is titled "Slack Web API" and describes the API's purpose. It shows examples of JSON responses for different success and failure cases. On the right, there's a "Related Articles" sidebar with a link to "Building Slack Bots in Swift" by Peter Zignego. A "Documentation" tab is selected in a navigation bar.

2. Click the HTTP RPC style methods link in the middle of the page.

3. Scroll down to the list of methods that describe the actions that can be performed with the channels.
4. Click the channels.list method.

channels.leave	Leaves a channel.
channels.list	Lists all channels in a Slack team.
channels.mark	Sets the read cursor in a channel.

5. View the documentation for the channels.list method with the arguments and response information.

Arguments

This method has the URL <https://slack.com/api/channels.list> and follows the [Slack Web API calling conventions](#).

Argument	Example	Required	Description
<code>token</code>	<code>xxxx-xxxxxxxxx-xxxx</code>	Required	Authentication token. Requires scope: <code>channels:read</code>
<code>exclude_archived</code>	<code>true</code>	Optional, default=false	Don't return archived channels.

Response

Returns a list of limited channel objects:

```
{
  "ok": true,
  "channels": [
    {
      "id": "C024BE91L",
      "name": "fun",
      "created": 1360782804.
```

Make a call to an endpoint

6. Scroll back up and click the Tester tab.

[channels.list](#) View another method... ▾

[Documentation](#) [Tester](#)

This method returns a list of all channels in the team. This includes channels the caller is in, channels they are not currently in, and archived channels but does not include private channels. The number of (non-deactivated) members in each channel is also returned.

To retrieve a list of private channels, use [groups.list](#).

7. In the Tester, select the No token or Invalid token option for the token attribute and click the Test Method button.

The screenshot shows the Slack API Tester interface for the `channels.list` method. At the top, there's a dropdown menu labeled "View another method...". Below it, two tabs are visible: "Documentation" (which is selected) and "Tester". The "Tester" tab has a form with the following fields:

Argument	Value
<code>token</code>	Required No token
<code>exclude_archived</code>	Optional, default=false
Extra args	

Below the form is a green "Test Method" button. A note below the first row says "Generate tokens to test with [here](#)".

8. Verify that you see the response as not authorized or invalid authorization.

URL

<https://slack.com/api/channels.list?pretty=1> (open raw response)

```
{  
  "ok": false,  
  "error": "not_authed"  
}
```

Note: To obtain a valid token, you need a Slack account and to belong to an organization with channels and members. If you do have an account, you can generate tokens by following the information in the link below the token attribute value in the tester.

Walkthrough 1-3: Review and make a call to a REST API

In this walkthrough, you explore a REST API. You will:

- Examine the Vimeo REST API.
- Make a call to a REST API endpoint to retrieve information.

The screenshot shows the Vimeo developer website's API playground. The URL is https://developer.vimeo.com/api/categories. The page title is "API / Playground". On the left, there's a "Vimeo API" logo and a cartoon illustration of small, colorful, worm-like creatures. In the center, there's a table for the "categories" endpoint. The table has columns for "Name", "Value", "Description", and "Required". The rows are:

Name	Value	Description	Required
page	int	The page number to show.	No
per_page	int	Number of items to show on each page. Max 100.	No
sort	Choose one	Technique used to sort the results.	No
direction	string	The direction that the results are sorted.	No

Below the table, there's a "Use Application:" dropdown set to "API Playground" and a "Make Call" button.

Examine the Vimeo REST API

1. In a web browser navigate to <https://developer.vimeo.com/api>.

Note: This URL is also located in the course snippets.txt file.

The screenshot shows the Vimeo developer website's main API page. The URL is https://developer.vimeo.com/api. The page title is "vimeo developer();". On the left, there's a "Vimeo API" logo and a cartoon illustration of small, colorful, worm-like creatures. On the right, there's a large illustration of a blue goat with long, curved horns. The page content includes sections for "API", "Endpoints", and "oEmbed".

API
The only way to perform authenticated read/write requests on videos, users, groups, channels, albums, or upload. The API uses OAuth 2 for authentication and you'll first need to register your app.
[Full documentation on the API](#)

Endpoints
Take a look at the full endpoint list to find out what's possible with the new API. If you want to see them in action, [check out our API playground](#).
[Full list of API Endpoints](#)

oEmbed
An open standard for embedding videos or images into a website. You can use the URL of any Vimeo video to easily get the embed code for it. You do not need to register your app to use oEmbed.
[Full documentation on oEmbed](#)

- Under the Endpoints heading on the Vimeo developer page, click check out our API playground.
- On the API/Playground page, click (Empty...) on the left-hand side.

The screenshot shows the Vimeo developer API/Playground interface. On the left, there is a sidebar with a list of resources: (Empty..), (Empty..), root, categories, channels, contentratings, creativecommons, documents, groups, and languages. The 'categories' item is selected. At the top right, there are links for API, Player, Guidelines, My Apps, Help, and Jobs. Below the sidebar, there is a main area with a 'View all API Endpoints' link, a dropdown for 'Use Application: API Playground', and a prominent yellow 'Make Call' button.

- In the expanded list of resources, click categories.
- Click the Make Call button.
- Verify the response has a status code 200.
- Scroll the page and view the data for the Animation and Arts & Design categories.

The screenshot shows the API response for the categories endpoint. The response header includes 'HTTP/1.1 200', 'Content-Type: application/vnd.vimeo.category+json', and 'Host: api.vimeo.com'. The JSON data returned is as follows:

```

{
  "total": 16,
  "page": 1,
  "per_page": 25,
  "paging": {
    "next": null,
    "previous": null,
    "first": "/categories?page=1",
    "last": "/categories?page=1"
  },
  "data": [
    {
      "uri": "/categories/animation",
      "name": "Animation",
      "link": "https://vimeo.com/categories/animation",
      "top_level": true,
      "pictures": {
        "uri": "/videos/203956723/pictures/618358902",
        "active": true,
        "type": "custom",
        "sizes": [
          {
            "width": 100,
            "height": 75,
            "link": "https://i.vimeocdn.com/video/618358902_100x75.jpg?r=pad",
            "link_with_play_button": "https://i.vimeocdn.com/filter/overlay?src0=https%3A%2F%2
          },
          {
            "width": 200,
            "height": 150,
            "link": "https://i.vimeocdn.com/video/618358902_200x150.jpg?r=pad",
            "link_with_play_button": "https://i.vimeocdn.com/filter/overlay?src0=https%3A%2F%2
        ]
      }
    }
  ]
}

```

Module 2: Translating Functional Requirements for APIs

ACME Banking API User Functionality

This document helps build API consumer stories to see how they will consume the API. API consumers can be involved at this stage to find new use cases for the API usage and prioritize features important to them.

ACME bank has customers(individuals, companies etc.) who want to perform actions with regards to their bank accounts. The accounts help them manage and handle their finances (using transactions).

As a developer, who is the API consumer at ACME Bank and is involved in building a web application for bankers to serve customers, what are the functions they should be able to perform with the API?

Categories:

CUSTOMERS – As a developer, one should be able to access and manipulate customer information.

- i. Get list of all customers in the bank
- ii. Register a new customer
- iii. Get customer information for a specific customer ID
- iv. Update customer information for a specific customer ID
- v. Delete a customer with a specific customer ID

ACCOUNTS – Since customers are associated with bank accounts, developers using the ACME Banking API should be able to work with account information.

- i. Get list of all account
- ii. Create a new account
- iii. Get account information
- iv. Delete an account with
- v. Update account informat

TRANSACTIONS – Bank accounts can perform actions on the transaction.

- i. Create a new transaction
- ii. Get transactions for a specific account
- iii. Get transaction informa

Translating categories and actions into resources and methods –

CUSTOMERS: Resource /customers

- i. Get list of all customers in the bank
- ii. Register a new customer
- iii. Get customer information for a specific customer ID
- iv. Update customer information for a specific customer ID
- v. Delete a customer with a specific customer ID

- Resource /customers
- Resource /customers
- Resource /customers/{customer_id}
- Resource /customers/{customer_id}
- Resource /customers/{customer_id}

- Method GET
- Method POST
- Method GET
- Method PATCH
- Method DELETE

ACCOUNTS: Resource /accounts

- i. Create a new account
- ii. Get account information for a specific account ID
- iii. Delete an account with a specific account ID
- iv. Update account information for a specific account ID
- ii. Get transactions for a specific account ID

- Resource /accounts
- Resource /accounts/{account_id}
- Resource /accounts/{account_id}
- Resource /accounts/{account_id}
- Resource /accounts/{account_id}/transactions

- Method POST
- Method GET
- Method DELETE
- Method PUT
- Method GET

TRANSACTIONS: Resource /transactions

- i. Create a new transaction
- iii. Get transaction information for a specific transaction ID

- Resource /transactions
- Resource /transactions/{transaction_id}

- Method POST
- Method GET

Objectives:

- Identify the different categories and actions for a REST API.
- Convert categories to resources.
- Select HTTP methods to support the actions on categories.

Walkthrough 2-1: List the categories and actions for an API

In this walkthrough, you list out the functional requirements for an API. You will:

- Identify the categories for a banking API.
- Define actions for the categories to decide how users will consume the API.

ACME Banking API User Functionality

This document helps build API consumer stories to see how they will consume the API. API consumers can be involved at this stage to find new use cases for the API usage and prioritize features important to them.

ACME bank has customers(individuals, companies etc.) who want to perform actions with regards to their bank accounts. The accounts help them manage and handle their finances (using transactions).

As a developer, who is the API consumer at ACME Bank and is involved in building a web application for bankers to serve customers, what are the functions they should have?

T0-DO 1: Identify the categories in the ACME Bank

CATEGORIES – As a developer, one should be able to access and manipulate customer information.

i. Get list of all customers in the bank
ii. Register a new customer
iii. Get customer information for a specific customer ID
iv. Update customer information for a specific customer ID
v. Delete a customer with a specific customer ID

ACCOUNTS – Since customers are associated with bank accounts, developers using the ACME Banking API should be able to work with account information.

i. Get list of all accounts for a specific customer ID
ii. Create a new account
iii. Get account information for a specific account ID
iv. Delete an account with a specific account ID
v. Update account information for a specific account ID

TRANSACTIONS – Bank accounts contain transactions and activities like withdrawal, deposit, interest earnings etc., and developers should be able to perform actions on the transaction information.

i. Create a new transaction
ii. Get transactions for a specific account ID
iii. Get transaction information for a specific transaction ID

Create user stories to define functional requirements for a banking API

1. Navigate to the studentFiles folder on your computer.
2. Navigate to the module02 directory and open the WT2-1 User functionality text file in a text editor.

ACME Banking API User Functionality

This document helps build API consumer stories to see how they will consume the API. API consumers can be involved at this stage to find new use cases for the API usage and prioritize features important to them.

ACME bank has customers(individuals, companies etc.) who want to perform actions with regards to their bank accounts. The accounts help them manage and handle their finances (using transactions).

As a developer, who is the API consumer at ACME Bank and is involved in building a web application for bankers to serve customers, what are the functions they should be able to perform with the API?

T0-DO 1: Identify the categories in the ACME Banking API

T0-DO 2: Add a one-line summary as to what an API consumer be able to do with these categories

T0-DO 3: Expand the one-line summary from T0-DO 2 into a detailed list of actions for each of the categories

3. Read the file.

Brainstorm for categories and actions

4. Brainstorm with the class about possible categories and actions.

Note: The instructor may break you out into smaller groups to first brainstorm with some of your peers.

List categories

5. In the WT 2-1 User functionality text file, add the following categories.

- CUSTOMERS
- ACCOUNTS
- TRANSACTIONS

List detailed actions

6. In the WT 2-1 User functionality text file, add a detailed list of actions that developers should be able to perform with the API for each of the categories.

CUSTOMERS –

- i. Get list of all customers in the bank
- ii. Get customer information for a specific customer ID
- iii. Register a new customer
- iv. Update customer information for a specific customer ID
- v. Delete a customer with a specific customer ID

ACCOUNTS –

- vi. Get list of all accounts for a specific customer ID
- vii. Get account information for a specific account ID
- viii. Create a new account
- ix. Update account information for a specific account ID
- x. Delete account with a specific account ID

TRANSACTIONS –

- xi. Get transactions for a specific account ID
- xii. Get transaction information for a specific transaction ID
- xiii. Create a new transaction

7. Save the file.

Walkthrough 2-2: Translate categories and actions into resources and methods

In this walkthrough, you translate the categories and actions identified in the last walkthrough into resources and methods. You will:

- Translate categories into resources.
- Identify the HTTP methods for actions.

Translating categories and actions into resources and methods -

CUSTOMERS: Resource _____	Resource _____	Method _____
i. Get list of all customers in the bank	Resource _____	Method _____
ii. Register a new customer	Resource _____	Method _____
iii. Get customer information for a specific customer ID	Resource _____	Method _____
iv. Update a customer information for a specific customer ID	Resource _____	Method _____
v. Delete a customer with a specific customer ID	Resource _____	Method _____

Translating categories and actions into resources and methods -		
ACCOUNTS: Resource _____	Resource /customers	Method GET
i. Get list of all accounts in the bank	Resource /customers	Method POST
ii. Create a new account	Resource /customers/{customer_id}	Method GET
iii. Get account information for a specific account ID	Resource /customers/{customer_id}	Method PATCH
iv. Update account information for a specific account ID	Resource /customers/{customer_id}	Method DELETE
v. Delete an account with a specific account ID	Resource /customers/{customer_id}/accounts	Method GET
TRANSACTIONS: Resource _____	Resource /accounts	Method POST
i. Create a new transaction	Resource /accounts/{account_id}	Method GET
ii. Get list of all transactions for a specific account ID	Resource /accounts/{account_id}	Method DELETE
iii. Get transaction information for a specific transaction ID	Resource /accounts/{account_id}/transactions	Method PUT
TRANSACTIONS: Resource /transactions	Resource /transactions	Method GET
i. Create a new transaction	Resource /transactions/{transaction_id}	Method POST
iii. Get transaction information for a specific transaction ID	Resource /transactions/{transaction_id}	Method GET

Translate categories into resources

1. Return to the module02 folder in the studentFiles folder on your computer.
2. Open the WT2-2 Resources and methods file in a text editor.

CUSTOMERS: Resource _____	Resource _____	Method _____
i. Get list of all customers in the bank	Resource _____	Method _____
ii. Get customer information for a specific customer ID	Resource _____	Method _____
iii. Register a new customer	Resource _____	Method _____
iv. Update a customer information for a specific customer ID	Resource _____	Method _____
v. Delete a customer with a specific customer ID	Resource _____	Method _____

ACCOUNTS: Resource _____	Resource _____	Method _____
i. Get list of all accounts for a specific customer ID	Resource _____	Method _____
ii. Get account information for a specific account ID	Resource _____	Method _____
iii. Create a new account	Resource _____	Method _____
iv. Delete an account with a specific account ID	Resource _____	Method _____
v. Update account information for a specific account ID	Resource _____	Method _____

TRANSACTIONS: Resource _____	Resource _____	Method _____
i. Get transactions for a specific account ID	Resource _____	Method _____
ii. Get transaction information for a specific transaction ID	Resource _____	Method _____
iii. Create a new transaction	Resource _____	Method _____

3. Review the file.

Brainstorm about possible resources and methods

4. Brainstorm with the class about possible resources and methods.

Note: The instructor may break you out into smaller groups to first brainstorm with some of your peers.

Specify resources

5. In the WT2-2 Resources and methods file, specify resources for the Customers entity.

- CUSTOMERS: Resource /customers
 - i. Get list of all customers - Resource /customers
 - ii. Register a new customer - Resource /customers
 - iii. Get customer information for a specific customer ID - Resource /customers/{customer_id}
 - iv. Update customer information for a customer ID - Resource /customers/{customer_id}
- Delete a customer with a specific customer ID - Resource /customers/{customer_id}

6. After a discussion, fill out the resources for Accounts entity.

- ACCOUNTS: Resource /accounts
 - i. Get list of all accounts for a specific customer ID – Resource /customers/{customer_id}/accounts
 - ii. Create a new account – Resource /accounts
 - iii. Get account information for a specific account ID – Resource /accounts/{account_id}
 - iv. Delete an account with a specific account ID – Resource /accounts/{account_id}
 - v. Update account information for a specific account ID – Resource /accounts/{account_id}

7. Specify resources for the Transactions entity.

- TRANSACTIONS: Resource /transactions
 - i. Create a new transaction – Resource /transactions
 - ii. Get transactions for a specific account ID – Resource /accounts/{account_id}/transactions
 - iii. Get transaction information for a specific transaction ID – Resource /transactions/{transaction_id}

Specify HTTP methods for the actions

8. Specify methods for the identified resources.
 - Get list of all customers – Method GET
 - Register a new customer – Method POST
 - Get customer information for a specific customer ID – Method GET
 - Update customer information for a customer ID – Method PATCH
 - Delete a customer with a specific customer ID – Method DELETE

 - Get list of all accounts for a specific customer ID – Method GET
 - Create a new account – Method POST
 - Get account information for a specific account ID – Method GET
 - Delete an account with a specific account ID – Method DELETE
 - Update account information for a specific account ID – Method PUT

 - Create a new transaction – Method POST
 - Get transactions for a specific account ID – Method GET
 - Get transaction information for a specific transaction ID – Method GET

Rearrange resources in the order of invocation

9. In the ACCOUNTS category, move the first line that contains the resource and method to get list of accounts for a specific customer inside the CUSTOMERS category.

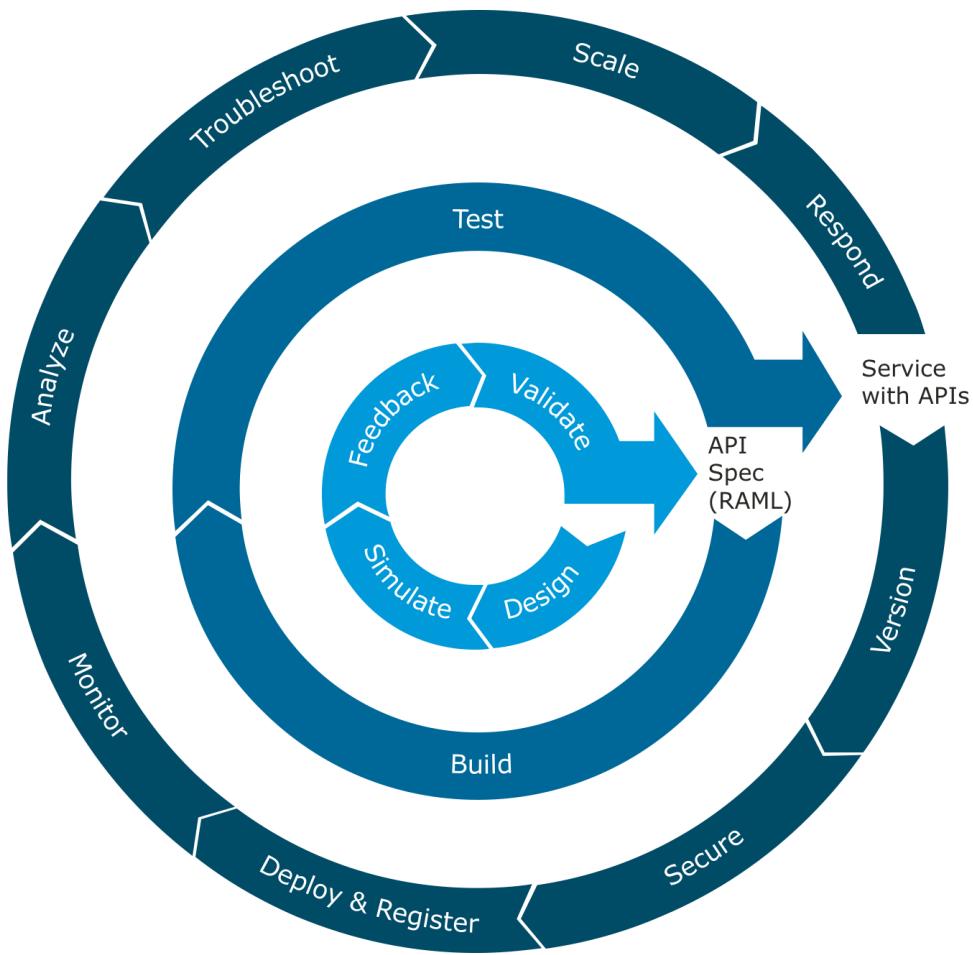
CUSTOMERS: Resource /customers		
i. Get list of all customers in the bank	Resource /customers	Method GET
ii. Register a new customer	Resource /customers	Method POST
iii. Get customer information for a specific customer ID	Resource /customers/{customer_id}	Method GET
iv. Update customer information for a specific customer ID	Resource /customers/{customer_id}	Method PATCH
v. Delete a customer with a specific customer ID	Resource /customers/{customer_id}	Method DELETE
vi. Get list of all accounts for a specific customer ID	Resource /customers/{customer_id}/accounts	Method GET

10. In the TRANSACTIONS category, move the second line that contains the resource and method to get transactions for a specific account inside the ACCOUNTS category.

ACCOUNTS: Resource /accounts		
i. Create a new account	Resource /accounts	Method POST
ii. Get account information for a specific account ID	Resource /accounts/{account_id}	Method GET
iii. Delete an account with a specific account ID	Resource /accounts/{account_id}	Method DELETE
iv. Update account information for a specific account ID	Resource /accounts/{account_id}	Method PUT
v. Get transactions for a specific account ID	Resource /accounts/{account_id}/transactions	Method GET

11. Save the text file.

Module 3: Introducing API-Led Connectivity and the API Lifecycle



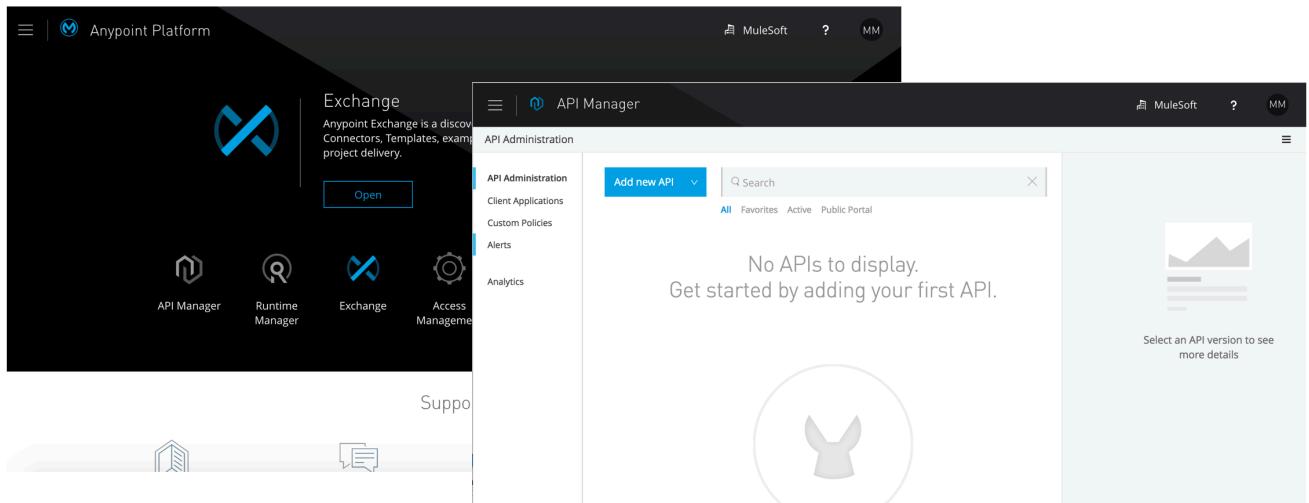
Objectives:

- Describe the API development lifecycle.
- Explain MuleSoft's API-led connectivity approach.
- Navigate Anypoint Platform.
- Describe the API development lifecycle with Anypoint Platform.

Walkthrough 3-1: Explore API Manager in Anypoint Platform

In this walkthrough, you get familiar with Anypoint Platform and its features. You will:

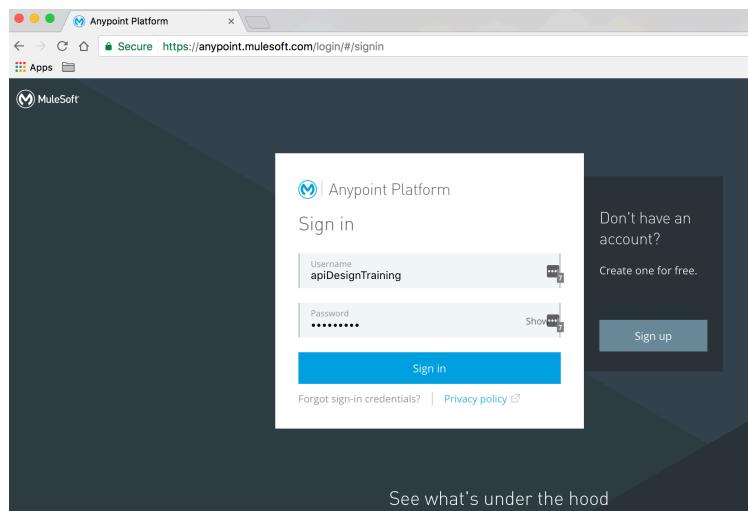
- Log in to Anypoint Platform.
- Explore API Manager in Anypoint Platform.



Log in to Anypoint Platform

1. In a web browser, navigate to <https://anypoint.mulesoft.com>.
2. Log in to Anypoint Platform.

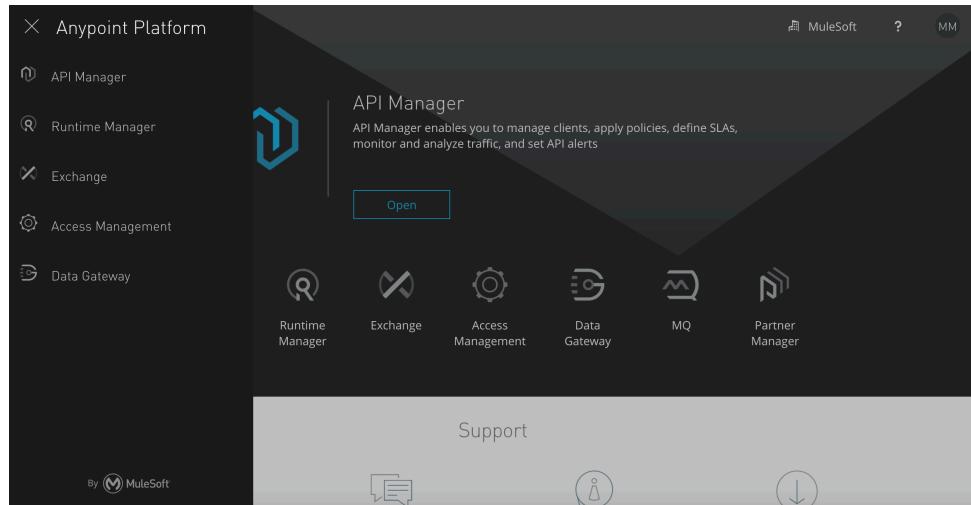
Note: You can use a trial account or your company account (if you already have one). If you do not have an account, sign up for a free, 30-day trial account now.



Explore API Manager in Anypoint Platform

3. Click the menu button located in the upper-left in the main menu bar.
4. Review the menu that appears that includes links to your Anypoint Platform entitlements.

Note: You can also open this menu by pressing your keyboard's Escape key.

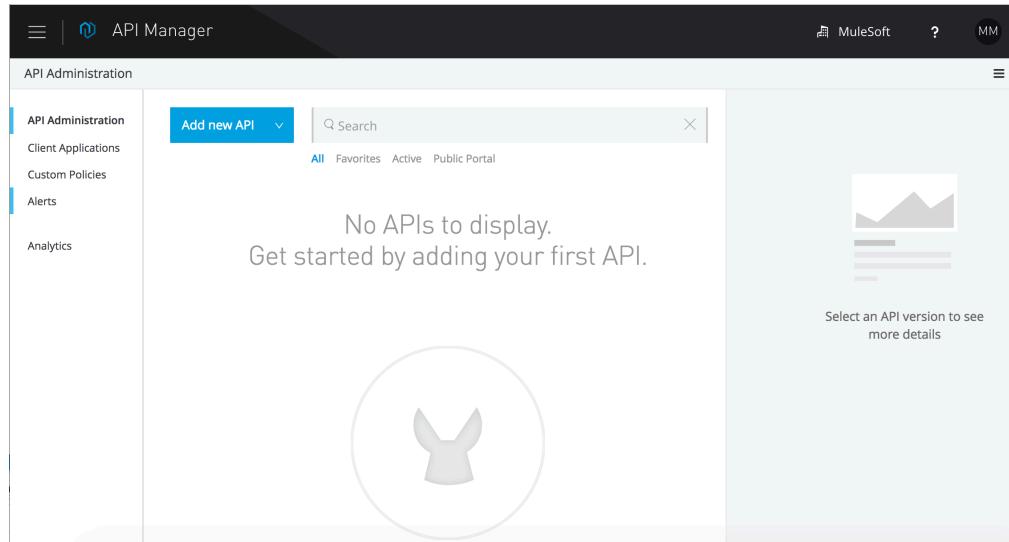


5. From the menu, select API Manager.

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

6. In the web page for API Manager, verify that you can see the following functionality:

- A Button to add A new API (with an option to import an API using the downward arrow next to it).
- A menu in the upper-right corner below the username initials to help work with developer portals for APIs, API Analytics, uploading custom policies, and more.



PART 2: Defining APIs with the RESTful API Modeling Language (RAML)

The screenshot displays the MuleSoft API Manager interface. On the left, the 'api.raml' file is open in the code editor, showing RAML 1.0 code defining resources for customers, accounts, and transactions. In the center, the 'Resources' panel shows the API structure with endpoints for POST, GET, DELETE, and PATCH methods. On the right, the developer portal for 'ACME Banking API 1.0' is shown, featuring a sidebar with links like Home, API reference, References, HTTP status codes, and API Notebook. The main content area lists functionality such as retrieving and manipulating customer, account, and transaction information.

Objectives:

- Create API definitions with RAML 1.0.
- Use patterns to refactor and modularize API definitions.
- Specify security schemes to secure resources in APIs.
- Add documentation to RAML API definitions.
- Make APIs discoverable through API Portals and Anypoint Exchange.
- Test APIs through the API Console.
- Add state specific responses to promote hypermedia.
- Learn when and how to version APIs.

Module 4: Defining API Resources and Methods

The screenshot shows the MuleSoft API Designer interface. On the left, a code editor displays the RAML 1.0 API definition for the ACME Banking API. The code includes sections for customers, accounts, and transactions, each with their own methods like get, post, patch, and delete, and associated URI parameters. On the right, the API resource details are shown for the /customers endpoint. It lists supported methods (POST, GET, DELETE, PATCH), describes the endpoint as '/customers/{customer_id}', and indicates that the 'Try-it' feature is disabled because 'baseUri' is not present. Below this, sections for 'Request', 'DESCRIPTION', 'URI PARAMETERS' (with 'customer_id' as a required string), and 'SECURITY SCHEMES' (listing 'Anonymous') are visible.

```
%RAML 1.0
title: ACME Banking API
version: 1.0

customers:
  get:
  post:
  /{customer_id}:
    uriParameters:
      | customer_id:
        get:
        patch:
        delete:
        /accounts:
          | get:

accounts:
  post:
  /{account_id}:
    uriParameters:
      | account_id:
        get:
        delete:
        put:
        /transactions:
          | get:

transactions:
  post:
  /{transaction_id}:
    uriParameters:
      | transaction_id:
        get:
```

Objectives:

- Use API Designer to create API definitions with RAML 1.0.
- Define resources and methods in RAML API definitions.
- Specify URI parameters for resource methods.

Walkthrough 4-1: Create an API and define resources in RAML 1.0

In this walkthrough, you create an API definition for the ACME Banking use case. You will:

- Create the ACME Banking API in API Designer.
- Define the resources and nested resources identified for the API.

Translating categories and actions into resources and methods -

CUSTOMERS: Resource /customers
i. Get list of all customers in the bank
ii. Register a new customer
iii. Get customer information for a specific customer ID
iv. Update customer information for a specific customer ID
v. Delete a customer with a specific customer ID
vi. Get list of all accounts for a specific customer ID

ACCOUNTS: Resource /accounts
i. Create a new account
ii. Get account information for a specific account ID
iii. Delete an account with a specific account ID
iv. Update account information for a specific account ID
ii. Get transactions for a specific account ID

TRANSACTIONS: Resource /transactions
i. Create a new transaction
iii. Get transaction information for a specific transaction ID

```
1  #RAML 1.0
2  title: ACME Banking API
3  version: 1.0
4
5  /customers:
6  | /{customer_id}:
7  | | /accounts:
8  | | | /{account_id}:
9  | | | | /transactions:
10 | | | | | /{transaction_id}:
11
12
13
14
```

Create a new API

1. Return to API Manager in Anypoint Platform.
2. Click the Add new API button.
3. In the Add API dialog box, enter the following information:
 - API name: ACME Banking API
 - Version name: 1.0
 - Description: This API provides endpoints to view and manipulate information related to customers, their accounts, and transaction activity.

Add API ORG MuleSoft X

API name *
ACME Banking API

Version name *
1.0

API endpoint
[Empty input field]

Description
ACME Banking API helps applications consume information related to bank customers, their accounts and transaction activity in these accounts.

Cancel Add

4. Click Add.
5. Look at the different sections and links for the API on the API version details page for ACME Banking API.

The screenshot shows the MuleSoft API Manager interface. At the top, there's a navigation bar with 'API Manager' and other icons. Below it, a breadcrumb navigation shows 'API administration / ACME Banking API - 1.0'. The main content area displays the 'ACME Banking API' version 1.0. It includes a section to 'Set the API URL...', a brief description of the API, and a button to 'ADD A TAG'. Below this are three cards: 'API Definition' (with a link to 'Define API in API designer'), 'API Portal' (showing 'No portal'), and 'API Status' (with a link to 'Configure endpoint'). At the bottom, there are tabs for 'Applications', 'Policies', 'SLA tiers', and 'Permissions', with 'Applications' being the active tab.

6. In the API Definition section, click the Define API in API designer link; the API Designer should open.
7. Examine the different parts of API Designer:
 - The file browser: The section on the left
 - The editor: The section in the middle with the RAML
 - The shelf: The gray section below the editor with tools to help write the RAML
 - The API Console: The section on the right for getting information about and testing APIs
8. In the editor, change the RAML version from 0.8 to 1.0.

The screenshot shows the MuleSoft API Manager API Designer interface. The title bar says 'API Administration / ACME Banking API (1.0) - Settings / Designer'. The menu bar includes 'Project', 'View', 'Help', and the file path '* /api.raml'. The main area is a code editor showing the following RAML code:

```

1  #%RAML 1.0
2  title: ACME Banking API
3  version: 1.0

```

Note: To change the background color from black to white, select View > Toggle Background Color or press Ctrl+Shift+T.

Review the resources and nested resources identified for the API

9. Return to the WT 2-2 Resources and methods file in a text editor.

Translating categories and actions into resources and methods -		
CUSTOMERS: Resource /customers		
i. Get list of all customers in the bank	Resource /customers	Method GET
ii. Register a new customer	Resource /customers	Method POST
iii. Get customer information for a specific customer ID	Resource /customers/{customer_id}	Method GET
iv. Update customer information for a specific customer ID	Resource /customers/{customer_id}	Method PATCH
v. Delete a customer with a specific customer ID	Resource /customers/{customer_id}	Method DELETE
vi. Get list of all accounts for a specific customer ID	Resource /customers/{customer_id}/accounts	Method GET
ACCOUNTS: Resource /accounts		
i. Create a new account	Resource /accounts	Method POST
ii. Get account information for a specific account ID	Resource /accounts/{account_id}	Method GET
iii. Delete an account with a specific account ID	Resource /accounts/{account_id}	Method DELETE
iv. Update account information for a specific account ID	Resource /accounts/{account_id}	Method PUT
ii. Get transactions for a specific account ID	Resource /accounts/{account_id}/transactions	Method GET
TRANSACTIONS: Resource /transactions		
i. Create a new transaction	Resource /transactions	Method POST
iii. Get transaction information for a specific transaction ID	Resource /transactions/{transaction_id}	Method GET

10. Review the resources.

Define the resources

11. Return to API Designer.
12. Place your cursor after the version number and press the Enter key two times to add two new lines.
13. Add the /customers resource.

/customers:

14. Press enter twice and add the /accounts resource.

15. Add the /transactions resource.

```
1  #%%RAML 1.0
2  title: ACME Banking API
3  version: 1.0
4
5  /customers:
6
7  /accounts:
8
9  /transactions:
```

16. Look at the API Console; you should see the three resources.

Note: If you do not see the API Console, click the arrow located in the middle of the right edge of the web browser window.

The screenshot shows the RAML 1.0 editor interface. On the left, the RAML code is displayed:

```
1  %%RAML 1.0
2  title: ACME Banking API
3  version: 1.0
4
5  /customers:
6
7  /accounts:
8
9  /transactions:
```

On the right, the 'Resources' section shows the generated API endpoints:

API is behind a firewall (?)
/customers
/accounts
/transactions

Define the nested resources

17. Go to the end of the /customers resource line, press enter to add a new line.

Note: To go to the end of a line, click anywhere in the line and press Ctrl+E.

18. Indent by pressing the Tab key.

19. Add a nested resource called {customer_id} that will require a URL parameter called customer_id to be pass do it.

Note: You will specify URI parameters in the next walkthrough.

```
4
5  /customers:
6    |  /{customer_id}:
```

20. On a new line, indent and add a nested resource called accounts.

```
4
5  /customers:
6    |  /{customer_id}:
7      |  |  /accounts:
```

21. Go to a new line after the main /accounts resource.

22. Indent and add a nested resource called {account_id}.

23. Add a nested resource called transactions.

```
5  /customers:
6    |  /{customer_id}:
7      |  |  /accounts:
8
9  /accounts:
10 |  /{account_id}:
11   |  |  /transactions:
```

24. Go to a new line after the main /transactions resource.

25. Indent and add a nested resource called {transaction_id}.

```
4
5 ▼ /customers:
6 ▼ |   /{customer_id}:
7 |     /accounts:
8
9 ▼ |   /accounts:
10 ▼ |     /{account_id}:
11 |       /transactions:
12
13 |     /transactions:
14 |       /{transaction_id}:
```

26. Look at the API Console; you should see the resources and nested resources displayed as noted in the WT 2-2 Resources and methods file.

The screenshot shows the API Console interface. On the left, there is a code editor window displaying the RAML 1.0 specification. On the right, there is a tree view of the generated resources and methods. The resources listed are /customers, /accounts, and /transactions. Under /customers, there is a method /customers/{customer_id}. Under /accounts, there is a method /accounts/{account_id}. Under /transactions, there is a method /transactions/{transaction_id}. A checkbox labeled "API is behind a firewall?" is visible at the top right of the tree view.

```
1 %RAML 1.0
2 title: ACME Banking API
3 version: 1.0
4
5 ▼ /customers:
6 ▼ |   /{customer_id}:
7 |     /accounts:
8
9 ▼ |   /accounts:
10 ▼ |     /{account_id}:
11 |       /transactions:
12
13 |     /transactions:
14 |       /{transaction_id}:
```

Resources	Methods
/customers	/customers/{customer_id}
/accounts	/accounts/{account_id}
/transactions	/transactions/{transaction_id}

27. From the Project menu in the upper-left, click Save.

The screenshot shows the MuleSoft Anypoint Studio interface. The Project menu is open, and the "Save" option is highlighted. The main workspace shows the RAML 1.0 file with its contents.

```
API Administration ▶ ACME Banking API (1.0) - Settings ▶ Designer
Project View Help * /api.raml
+ New File ▶
New Folder
Save
Save All
Import
Export files
1 %RAML 1.0
2 title: ACME Banking API
3 version: 1.0
4
5 ▼ /customers:
6 ▼ |   /{customer_id}:
7 |     /accounts:
8
9 ▼ |   /accounts:
10 ▼ |     /{account_id}:
```

Walkthrough 4-2: Define methods for the resources

In this walkthrough, you add methods to the defined resources to retrieve, modify, or delete data. You will:

- Define HTTP methods for resources to retrieve collection and item data.
- Define HTTP methods for resources to modify and delete data.

The screenshot shows the MuleSoft API Manager interface. On the left, the API RAML file is displayed in a code editor. The code defines an API with a base URL of `https://mocksvc.mulesoft.com/mocks/50c94779-b77b-4d4d-83f4-dca93696b9f0`, title "ACME Banking API", and version 1.0. It includes resources for customers, accounts, and transactions, each with GET, POST, PATCH, and DELETE methods. On the right, the generated API resources are listed with their corresponding URLs and HTTP methods. For example, the /customers resource has a POST method for creating a new customer and a GET method for retrieving all customers. The /accounts resource has a POST method for creating a new account and a GET method for retrieving all accounts for a specific customer. The /transactions resource has a POST method for creating a new transaction and a GET method for retrieving all transactions for a specific account.

Review the get methods identified for the API

1. Return to the WT 2-2 Resources and methods file in a text editor.

Translating categories and actions into resources and methods -			
CUSTOMERS: Resource /customers			
i. Get list of all customers in the bank	Resource /customers	Method GET	
ii. Register a new customer	Resource /customers	Method POST	
iii. Get customer information for a specific customer ID	Resource /customers/{customer_id}	Method GET	
iv. Update customer information for a specific customer ID	Resource /customers/{customer_id}	Method PATCH	
v. Delete a customer with a specific customer ID	Resource /customers/{customer_id}	Method DELETE	
vi. Get list of all accounts for a specific customer ID	Resource /customers/{customer_id}/accounts	Method GET	
ACCOUNTS: Resource /accounts			
i. Create a new account	Resource /accounts	Method POST	
ii. Get account information for a specific account ID	Resource /accounts/{account_id}	Method GET	
iii. Delete an account with a specific account ID	Resource /accounts/{account_id}	Method DELETE	
iv. Update account information for a specific account ID	Resource /accounts/{account_id}	Method PUT	
ii. Get transactions for a specific account ID	Resource /accounts/{account_id}/transactions	Method GET	
TRANSACTIONS: Resource /transactions			
i. Create a new transaction	Resource /transactions	Method POST	
iii. Get transaction information for a specific transaction ID	Resource /transactions/{transaction_id}	Method GET	

2. Review the resources that have get methods.

Define HTTP methods for resources to retrieve collection and item data

3. Return to API Designer.
4. Add a new line between the /customers and /{customer_id} resources.
5. In the shelf below the RAML editor, click the get method.

The screenshot shows the RAML editor interface. On the left, a code editor displays the following RAML code:

```
5 ▼ /customers:  
6  
7 ▼   /{customer_id}:  
8     | accounts:  
9  
10 ▼    /accounts:  
11 ▼      /{account_id}:  
12        | transactions:  
13  
14      /transactions:  
15      /{transaction_id}:
```

Below the code editor is a shelf titled "METHODS (7)" containing the following icons: get, put, post, delete, options, head, and patch. To the left of the shelf, under "DOCS (2)", are icons for "description" and "displayName".

6. Delete the new line that is automatically added below the get method.
7. Similarly, add get methods to the nested /{customer_id} and /accounts resources.
8. Delete any extra lines of code.

The screenshot shows the RAML editor with the following updated code:

```
5 ▼ /customers:  
6   get:  
7 ▼   /{customer_id}:  
8     get:  
9 ▼     /accounts:  
10       get:  
11  
12 ▼     /accounts:  
13     /{account_id}:  
14       /transactions:  
15  
16     /transactions:  
17     /{transaction_id}:
```

9. Look at the API Console; you should see tabs for the get methods.

The screenshot shows the API Console. The "Resources" section lists the following endpoints under the /customers resource:

- /customers (GET)
- /customers/{customer_id} (GET)
- /customers/{customer_id}/accounts (GET)

- For the /accounts resource in the editor, add get methods to the nested /{account_id} and /transactions resources.
- For the /transactions resource, add a get method to the nested /{transaction_id} resource.

The screenshot shows the MuleSoft API Designer interface. On the left, the 'api.raml' file is open, displaying RAML code for an API. The code defines a 'customers' resource with a 'get' method that returns a specific customer by ID. It also defines an 'accounts' resource under 'customers' with a 'get' method. Within the 'accounts' resource, there is another 'accounts' resource with a 'get' method that returns a specific account by ID. This nested 'accounts' resource has its own 'transactions' resource with a 'get' method. The right side of the interface shows the generated API resources and their corresponding HTTP methods. The resources listed are /customers, /customers/{customer_id}, /customers/{customer_id}/accounts, /accounts, /accounts/{account_id}, /accounts/{account_id}/transactions, and /transactions. Each resource entry includes a 'GET' button, indicating the available HTTP method for that endpoint.

```

2 title: ACME Banking API
3 version: 1.0
4
5 /customers:
6   get:
7     /{customer_id}:
8       get:
9         /accounts:
10        | get:
11
12 /accounts:
13   /{account_id}:
14     get:
15     /transactions:
16     | get:
17
18 /transactions:
19   /{transaction_id}:
20     get:
  
```

Resources	
collapse all	Mocking Service OFF
API is behind a firewall (?)	
/customers	GET
/customers/{customer_id}	GET
/customers/{customer_id}/accounts	GET
/accounts	
/accounts/{account_id}	GET
/accounts/{account_id}/transactions	GET
/transactions	
/transactions/{transaction_id}	GET

Review the methods identified for the /customers resource and nested resources

- Return to the WT 2-2 Resources and methods file in a text editor.

Translating categories and actions into resources and methods -		
CUSTOMERS: Resource /customers	Resource /customers	Method GET
i. Get list of all customers in the bank	Resource /customers	Method POST
ii. Register a new customer	Resource /customers/{customer_id}	Method GET
iii. Get customer information for a specific customer ID	Resource /customers/{customer_id}	Method PATCH
iv. Update customer information for a specific customer ID	Resource /customers/{customer_id}	Method DELETE
v. Delete a customer with a specific customer ID	Resource /customers/{customer_id}/accounts	Method GET
vi. Get list of all accounts for a specific customer ID		
ACCOUNTS: Resource /accounts	Resource /accounts	Method POST
i. Create a new account	Resource /accounts/{account_id}	Method GET
ii. Get account information for a specific account ID	Resource /accounts/{account_id}	Method DELETE
iii. Delete an account with a specific account ID	Resource /accounts/{account_id}	Method PUT
iv. Update account information for a specific account ID	Resource /accounts/{account_id}/transactions	Method GET
ii. Get transactions for a specific account ID		
TRANSACTIONS: Resource /transactions	Resource /transactions	Method POST
i. Create a new transaction	Resource /transactions/{transaction_id}	Method GET
iii. Get transaction information for a specific transaction ID		

- Review the methods for the /customers resource and nested resources.

Define methods for the /customers resource to add, modify, and delete data

- Return to API Designer.
- Type or use the shelf to add a post method to the /customers resource.

16. Add patch and delete methods to the /{customer_id} nested resource.

```
1  #%%RAML 1.0
2  title: ACME Banking API
3  version: 1.0
4
5  ▼ /customers:
6      get:
7      post:
8  ▼ {customer_id}:
9      get:
10     patch:
11     delete:
12  ▼ /accounts:
13      get:
14
15  ▼ /accounts:
16      post:
17      |
18  ▼ {account_id}:
19      get:
```

17. Look at the API Console; you should see tabs for the new methods.

```
1  #%%RAML 1.0
2  title: ACME Banking API
3  version: 1.0
4
5  ▼ /customers:
6      get:
7      post:
8  ▼ {customer_id}:
9      get:
10     patch:
11     delete:
12  ▼ /accounts:
13      get:
14
```

The screenshot shows the API Console interface. At the top right, there is a note: "API is behind a firewall (?)". Below it is a "Collapse All" button. The main area is titled "Resources". Under the "/customers" resource, there are three tabs: "POST" (highlighted in blue), "GET" (highlighted in green), and "DELETE" (highlighted in red). Below the "/customers" resource, there are two more tabs: "PATCH" (highlighted in blue) and "GET" (highlighted in green). At the bottom, there is another tab labeled "GET".

Review the methods identified for the /accounts resource and nested resources

18. Return to the WT 2-2 Resources and methods file in a text editor.

Translating categories and actions into resources and methods -		
CUSTOMERS: Resource /customers	Resource /customers	Method GET
i. Get list of all customers in the bank	Resource /customers	Method POST
ii. Register a new customer	Resource /customers/{customer_id}	Method GET
iii. Get customer information for a specific customer ID	Resource /customers/{customer_id}	Method PATCH
iv. Update customer information for a specific customer ID	Resource /customers/{customer_id}	Method DELETE
v. Delete a customer with a specific customer ID	Resource /customers/{customer_id}/accounts	Method GET
vi. Get list of all accounts for a specific customer ID		
ACCOUNTS: Resource /accounts	Resource /accounts	Method POST
i. Create a new account	Resource /accounts/{account_id}	Method GET
ii. Get account information for a specific account ID	Resource /accounts/{account_id}	Method DELETE
iii. Delete an account with a specific account ID	Resource /accounts/{account_id}	Method PUT
iv. Update account information for a specific account ID	Resource /accounts/{account_id}/transactions	Method GET
ii. Get transactions for a specific account ID		
TRANSACTIONS: Resource /transactions	Resource /transactions	Method POST
i. Create a new transaction	Resource /transactions/{transaction_id}	Method GET
iii. Get transaction information for a specific transaction ID		

19. Review the methods for the /accounts resource and nested resources.

Define methods for the /accounts resource to add, modify, and delete data

20. Return to API Designer.
21. Type or use the shelf to add a post method to the /accounts resource.
22. Add put and delete methods to the /{account_id} nested resource.
23. Look at the API Console; you should see tabs for the new methods.

```
15 ▼  /accounts:  
16    post:  
17 ▼  /{account_id}:  
18    get:  
19    put:  
20    delete:  
21 ▼  /transactions:  
22    /{transaction_id}:  
23    get:  
24 ▼  /transactions:  
25    /{transaction_id}:  
26    get:
```

Review the methods identified for the /transactions resource and nested resources

24. Return to the WT 2-2 Resources and methods file in a text editor.

Translating categories and actions into resources and methods -			
CUSTOMERS: Resource /customers			
i. Get list of all customers in the bank	Resource /customers	Method GET	
ii. Register a new customer	Resource /customers	Method POST	
iii. Get customer information for a specific customer ID	Resource /customers/{customer_id}	Method GET	
iv. Update customer information for a specific customer ID	Resource /customers/{customer_id}	Method PATCH	
v. Delete a customer with a specific customer ID	Resource /customers/{customer_id}	Method DELETE	
vi. Get list of all accounts for a specific customer ID	Resource /customers/{customer_id}/accounts	Method GET	
ACCOUNTS: Resource /accounts			
i. Create a new account	Resource /accounts	Method POST	
ii. Get account information for a specific account ID	Resource /accounts/{account_id}	Method GET	
iii. Delete an account with a specific account ID	Resource /accounts/{account_id}	Method DELETE	
iv. Update account information for a specific account ID	Resource /accounts/{account_id}	Method PUT	
ii. Get transactions for a specific account ID	Resource /accounts/{account_id}/transactions	Method GET	
TRANSACTIONS: Resource /transactions			
i. Create a new transaction	Resource /transactions	Method POST	
iii. Get transaction information for a specific transaction ID	Resource /transactions/{transaction_id}	Method GET	

25. Review the methods for the /transactions resource and nested resources.

Define methods for the /transactions resource to add, modify and delete data

26. Return to API Designer.
27. Type or use the shelf to add a post method to the /transactions resource.
28. Save the project.

29. Look at the API Console; you should see tabs for the new methods.

```
1  #%RAML 1.0
2  title: ACME Banking API
3  version: 1.0
4
5  ▼ /customers:
6      get:
7      post:
8  ▼ {customer_id}:
9      get:
10     patch:
11     delete:
12  ▼ /accounts:
13      |  get:
14
15  ▼ /accounts:
16      post:
17  ▼ {account_id}:
18      get:
19      put:
20      delete:
21  ▼ /transactions:
22      |  get:
23
24  ▼ /transactions:
25      post:
26  ▼ {transaction_id}:
27      |  get:
```

API is behind a firewall (?)

Resources

▼ /customers

POST GET

/customers/{customer_id}

DELETE PATCH GET

/customers/{customer_id}/accounts

▼ /accounts

POST

/accounts/{account_id}

DELETE PUT GET

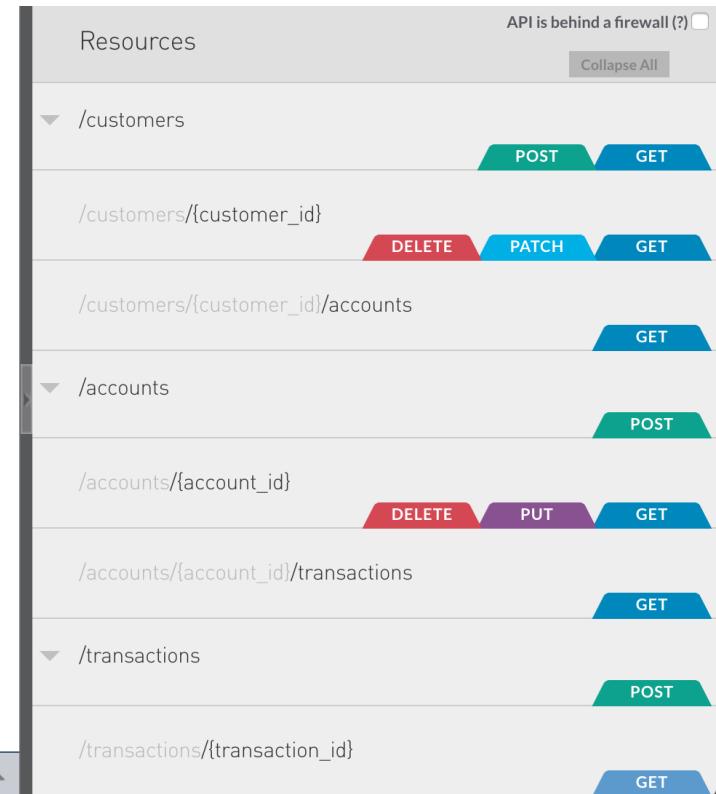
/accounts/{account_id}/transactions

▼ /transactions

POST

/transactions/{transaction_id}

GET



Walkthrough 4-3: Specify URI parameters for necessary resource methods

In this walkthrough, you specify URI parameters for resources that require it. You will:

- Identify resources that require URI parameters.
- Define URI parameters in the RAML definition.

The screenshot shows a RAML file on the left and a Mocking Service interface on the right. The RAML file defines an API with resources for customers, accounts, and transactions. The 'customers' resource has a 'get' method with a parameter '{customer_id}'. The 'accounts' resource has a 'post' method with a parameter '{account_id}'. The 'transactions' resource has a 'post' method with a parameter '{transaction_id}'. The Mocking Service interface shows the 'customers' resource with a 'GET' method highlighted. A red bar at the bottom states 'Try-it is disabled because baseUrl is not present'. The 'URI PARAMETERS' section shows 'customer_id required string'. The 'SECURITY SCHEMES' section shows 'Anonymous'.

Identify resources that require URI parameters

1. Return to the WT2-2 Resources and methods file in a text editor.
2. Identify the nested resources that require an ID passed to the URI as a parameter.

Translating categories and actions into resources and methods -		
CUSTOMERS: Resource /customers		
i. Get list of all customers in the bank	Resource /customers	Method GET
ii. Register a new customer	Resource /customers	Method POST
iii. Get customer information for a specific customer ID	Resource /customers/{customer_id}	Method GET
iv. Update customer information for a specific customer ID	Resource /customers/{customer_id}	Method PATCH
v. Delete a customer with a specific customer ID	Resource /customers/{customer_id}	Method DELETE
vi. Get list of all accounts for a specific customer ID	Resource /customers/{customer_id}/accounts	Method GET
ACCOUNTS: Resource /accounts		
i. Create a new account	Resource /accounts	Method POST
ii. Get account information for a specific account ID	Resource /accounts/{account_id}	Method GET
iii. Delete an account with a specific account ID	Resource /accounts/{account_id}	Method DELETE
iv. Update account information for a specific account ID	Resource /accounts/{account_id}	Method PUT
ii. Get transactions for a specific account ID	Resource /accounts/{account_id}/transactions	Method GET
TRANSACTIONS: Resource /transactions		
i. Create a new transaction	Resource /transactions	Method POST
iii. Get transaction information for a specific transaction ID	Resource /transactions/{transaction_id}	Method GET

Note: The {customer_id}, {account_id}, and {transaction_id} resources are the ones that require URI parameters with the respective values. These URLs are invalid when called without an ID value in the URI.

Define a customer_id URI parameter

3. Return to API Designer.
4. Add a new line below the /{customer_id} nested resource above the get method.
5. Make sure the cursor in the new line is aligned to the get method in the line below.
6. Locate uriParameters in the shelf and click it.

Note: Make sure you have the correct indentation. If you don't you will not see the correct keywords at that level of indentation.

7. Make sure the cursor is indented on the new line below uriParameters.
8. In the shelf, click customer_id:.

```
4
5 ▼ /customers:
6   |  get:
7   |  post:
8 ▼   |  /{customer_id}:
9   |    |  uriParameters:
10  |    |  |
11  |    |  get:
12  |    |  patch:
13  |    |  delete:
14 ▼   |    |  /accounts:
15   |    |    |  get:
16
17 ▼   |  /accounts:
18   |    |  post:
19 ▼   |    |  /{account_id}:
```

OTHERS (1)

customer_id:

9. Delete the new line that is automatically added below the customer_id parameter.

```
5 ▼ /customers:
6   |  get:
7   |  post:
8 ▼   |  /{customer_id}:
9   |    |  uriParameters:
10  |    |    |  customer_id:
11  |    |    |  get:
12  |    |    |  patch:
13  |    |    |  delete:
14 ▼   |    |    |  /accounts:
15   |    |    |    |  get:
```

Define account_id and transaction_id URI parameters

10. Similarly, add a uriParameters parameter to the nested /{account_id} resource of /accounts and name it account_id.

```
17 ▼ | /accounts:  
18 |   post:  
19 ▼ |   /{account_id}:  
20 |     uriParameters:  
21 |       account_id:  
22 |       get:  
23 |       put:  
24 |       delete:  
25 ▼ |       /transactions:  
26 |         get:
```

11. Add a uriParameters parameter to the nested /{transaction_id} resource of /transactions and name it transaction_id.

```
28 ▼ | /transactions:  
29 |   post:  
30 ▼ |   /{transaction_id}:  
31 |     uriParameters:  
32 |       transaction_id:  
33 |       get:
```

12. Save the project.

View the generated API documentation

13. In the API Console, locate the /customers/{customer_id} resource and click its GET tab.

The screenshot shows the Mule API Console interface. At the top, there is a checkbox labeled "API is behind a firewall (?)". Below that, a "Resources" section is shown with a "Collapse All" button. Under the resources, a "/customers" endpoint is listed with a dropdown arrow. Below the endpoint, there are four colored tabs: POST (green), GET (blue), DELETE (red), and PATCH (light blue). The "GET" tab is currently selected, indicated by a white background and blue border. The URL for the resource is also visible as "/customers/{customer_id}".

14. Locate customer_id listed under URI PARAMETERS.

The screenshot shows the MuleSoft Anypoint Studio interface. In the top left, there's a 'Resources' section with a 'POST' button highlighted in green. Below it, a modal window is open for the endpoint '/customers/{customer_id}'. The modal has four buttons: 'DELETE' (red), 'PATCH' (blue), 'GET' (white), and 'CLOSE X' (top right). A red bar at the bottom of the modal contains the text 'Try-it is disabled because baseUrl is not present'. The main interface below the modal shows sections for 'DESCRIPTION' and 'URI PARAMETERS', where 'customer_id' is listed as a required string.

Note: We will be passing in values to the URI parameter and testing the methods in the later modules after we input examples in the API definition. URI parameters are necessary part of the resource being called.

15. In the upper-right corner, click CLOSE X.

Module 5: Specifying Responses

```
#%RAML 1.0
title: ACME Banking API
version: 1.0

/customers:
  get:
    headers:
      Accept?:
    responses:
      200:
        headers:
          Cache-Control:
          Expires:
          | type: datetime-only
        body:
          application/json:
          application/xml:
      404:
        body:
          application/json:
          properties:
            statusCode: string
            message: string
```

Objectives:

- Describe the structure of HTTP method responses.
- Use status codes in HTTP responses.
- Add error handling and caching information to HTTP responses.
- Select and specify the types of content returned in HTTP responses.

Walkthrough 5-1: Add HTTP 2xx responses to resource methods

In this walkthrough, you add HTTP 2xx response bodies to all resource methods. You will:

- Add an HTTP 200 response body to all GET methods indicating success of the HTTP request.
- Add an HTTP 201 response body to all POST methods indicating success of the HTTP request.
- Add an HTTP 200 response body to all DELETE methods indicating success of the HTTP request.
- Add a response body with both HTTP 200 and 201 status codes to the PUT method.
- Add an HTTP 204 response body to PATCH methods indicating request success.

```
1  #%RAML 1.0
2  title: ACME Banking API
3  version: 1.0
4
5  ▼ /customers:
6  ▾ |   get:
7  ▾ |   |   responses:
8  ▾ |   |   |   200:
9  ▾ |   |   |   |   body:
10 |   |   |   |   |   application/json:
11 ▾ |   post:
12 ▾ |   |   responses:
13 ▾ |   |   |   201:
14 |   |   |   |   headers:
15 |   |   |   |   |   Location:
16 |   |   |   |   body:
17 |   |   |   |   |   application/json:
18 ▾ |   |   |   /{customer_id}:
19 |   |   |   |   uriParameters:
20 |   |   |   |   |   customer_id:
21 ▾ |   |   |   get:
22 ▾ |   |   |   |   responses:
23 |   |   |   |   |   200:
24 |   |   |   |   |   body:
25 |   |   |   |   |   |   application/json:
26 ▾ |   |   patch:
27 |   |   |   responses:
28 |   |   |   |   204:
29 ▾ |   |   delete:
30 ▾ |   |   |   responses:
31 |   |   |   |   |   200:
32 |   |   |   |   |   body:
33 |   |   |   |   |   application/json:
```

Add an HTTP 200 response body to GET methods indicating request success

1. Return to API Designer.
2. In the editor, add a new line under the get method of the /customers resource.
3. Press the tab key to indent.

4. In the shelf, click responses.

```
5 ▼ /customers:  
6   |  get:  
7   |  post:  
8   |  /{customer_id}:  
9   |    uriParameters:  
10  |      |  customer_id:  
11  |      get:  
12  |      patch:  
13  |      delete:  
14  |      /accounts:  
15 ▼ |      get:  
16
```

RESPONSES (1) SECURITY (1)

responses **securedBy**

5. In the shelf, click 200.

```
5 ▼ /customers:  
6 ▼ |  get:  
7   |   responses:  
8   |   post:  
9   |   /{customer_id}:  
10  |     uriParameters:  
11  |     |  customer_id:  
12  |     get:  
13  |     patch:  
14  |     delete:  
15  |     /accounts:  
16 ▼
```

OTHERS (55)

100
 101
 200
 201

6. In the shelf, click body.

```
5 ▼ /customers:  
6 ▼ |  get:  
7 ▼ |   responses:  
8   |     200:  
9   |   post:  
10  |   /{customer_id}:  
11 ▼ |     uriParameters:  
12  |     |  customer_id:  
13  |     get:  
14  |     patch:  
15  |     delete:
```

DOCS (1) PARAMETERS (1) RESPONSE (1)

description **headers** **body**

7. In the shelf, click application/json .

```
5 ▼ /customers:  
6 ▼   | get:  
7 ▼     |   responses:  
8 ▼       |     200:  
9         |       | body:  
10        |       |  
11       post:  
12 ▼     |   /{customer_id}:  
13       |   uriParameters:  
14         |     customer_id:  
15       get:  
16       patch:
```

BODY (2)

 application/json

 application/xml

8. Delete the new line created below the application/json line.

```
5 ▼ /customers:  
6 ▼   | get:  
7 ▼     |   responses:  
8 ▼       |     200:  
9         |       | body:  
10        |       | application/json:  
11       post:
```

9. Similarly, in the /{customer_id} nested resource, add an HTTP 200 response body to the get method.

```
12 ▼   |   /{customer_id}:  
13     |   | uriParameters:  
14       |   | customer_id:  
15 ▼     |   | get:  
16 ▼       |   | responses:  
17 ▼         |   |   200:  
18           |   |   | body:  
19             |   |   | application/json:
```

10. Repeat the process to add the same HTTP 200 response body to the get methods of the following resources:

- /customers/{customer_id}/accounts
- /accounts/{account_id}
- /accounts/{account_id}/transactions
- /transactions/{transaction_id}

```
5 ▼  /customers:
6 ▼    get:
7 ▶      responses: +
11   post:
12 ▼     /{customer_id}:
13       uriParameters:
14         | customer_id:
15 ▼       get:
16 ▼         responses:
17 ▼           200:
18             | body:
19               | application/json:
20             patch:
21             delete:
22 ▼     /accounts:
23 ▼       get:
24 ▼         responses:
25 ▼           200:
26 ▼             | body:
27               | application/json:
28
29 ▼   /accounts:
30     post:
31 ▼   /{account_id}:
32     uriParameters:
33       | account_id:
34 ▼     get:
35 ▼       responses:
36 ▼         200:
37           | body:
38             | application/json:
39     delete:
40     put:
41 ▼   /transactions:
42 ▼     get:
43 ▼       responses:
44 ▼         200:
45 ▼           | body:
46             | application/json:
47
48 ▼   /transactions:
49     post:
50 ▼   /{transaction_id}:
51     uriParameters:
52       | transaction_id:
53 ▼     get:
54 ▼       responses:
55 ▼         200:
56           | body:
57             | application/json:
```

Add an HTTP 201 response body to POST methods indicating request success

11. Add an HTTP 201 response body to the post method of the /customers resource.

```
5 ▼ | /customers:  
6 ▼ |   get:  
7 ▼ |     responses:  
8 ▼ |       200:  
9 |         body:  
10 |           application/json:  
11 ▼ |   post:  
12 ▼ |     responses:  
13 ▼ |       201:  
14 |         body:  
15 |           application/json:
```

12. Add a new line above the body and indent to the same level as body.

13. In the shelf, click the headers parameter.

14. In the shelf, scroll and click the Location header.

15. Delete the new line created below the Location line.

```
11 ▼ |   post:  
12 ▼ |     responses:  
13 ▼ |       201:  
14 |         headers:  
15 |           Location:  
16 |         body:  
17 |           application/json:
```

16. Repeat the process to add the same HTTP 201 response body to the post methods of the following resources:

- /accounts
- /transactions

```
35 ▼ | /accounts:  
36 ▼ |   post:  
37 ▼ |     responses:  
38 ▼ |       201:  
39 |         headers:  
40 |           Location:  
41 |         body:  
42 |           application/json:  
43 ▼ |          /{account_id}:  
60 ▼ | /transactions:  
61 ▼ |   post:  
62 ▼ |     responses:  
63 ▼ |       201:  
64 |         headers:  
65 |           Location:  
66 |         body:  
67 |           application/json:
```

Add an HTTP 200 response body to the DELETE methods indicating request success

17. Add an HTTP 200 response body to the delete methods of the following resources:

- /customers/{customer_id}
- /accounts/{account_id}

```
18 ▼ |  /{customer_id}:           47 ▼ |  /{account_id}:          ..  
19 |  uriParameters:         48 |  uriParameters:  
20 |    customer_id:         49 |  
21 ▼ |  get:                  50 ▼ |  get:  
22 ▼ |    responses:        51 ▼ |    responses:  
23 ▼ |      200:              52 ▼ |      200:  
24 |        body:            53 |        body:  
25 |          application/json: 54 |          application/json:  
26 |      patch:             55 ▼ |  
27 ▼ |      delete:            56 ▼ |  delete:  
28 ▼ |    responses:          57 ▼ |    responses:  
29 ▼ |      200:              58 |      200:  
30 |        body:            59 |        body:  
31 |          application/json: 60 ▼ |          application/json:  
put:
```

Add a response body with both HTTP 200 and 201 responses to the PUT method

18. Add an HTTP 200 response body to the put method of the /{account_id} nested resource.

19. Add a second status code to the response with a value of 201.

```
60 ▼ |  put:  
61 ▼ |    responses:  
62 ▼ |      200:  
63 |        body:  
64 |          application/json:  
65 ▼ |      201:  
66 |        body:  
67 |          application/json:
```

Add an HTTP 204 response body to PATCH methods indicating request success

20. Add a new line of code below the patch method of the /customers resource and indent.

21. In the shelf below, click responses.

22. In the shelf, click 204.

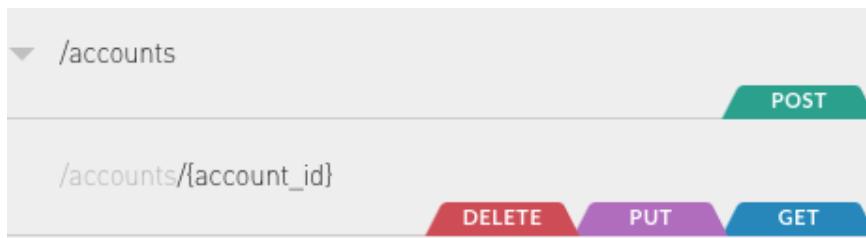
23. Delete the new line created below the 204 status code line.

```
25 | | | | application/json:  
26 ▼ | | | patch:  
27 | | responses:  
28 | | | 204:  
29 | | delete:
```

24. Save the project.

View the generated API documentation

25. In the API Console, locate the /accounts/{account_id} resource and click its PUT tab.



26. Locate the two HTTP status code responses under Response.

The screenshot shows the 'Response' section of the API documentation. It lists two status codes: 'STATUS 200' and '201'. Under '200', it says 'Body application/json' and 'application/json object'. A green box highlights the '200' status code.

27. Scroll up to the /accounts/{account_id} resource name and click the CLOSE X button.

Walkthrough 5-2: Add responses bodies to return error status codes and messages for client-side errors

In this walkthrough, you add response bodies to return error status codes and error messages for client-side errors. You will:

- Add HTTP 4xx status code responses to all GET and DELETE methods.
- Specify error messages to be returned.

The screenshot shows the API Designer interface. On the left is the RAML 1.0 code editor with the following content:

```
1  #%RAML 1.0
2  title: ACME Banking API
3  version: 1.0
4
5  /customers:
6  |   get:
7  |   |       responses:
8  |   |       |   200:
9  |   |       |       body:
10 |   |       |       |   application/json:
11 |   |       |       |       properties:
12 |   |       |       |       statusCode: string
13 |   |       |       |       message: string
14 |
15 |
16 |
17 |   post:
18 |   |       responses:
19 |   |       |   201:
20 |   |       |       headers:
```

On the right is the Response tab, which includes the SECURITY SCHEMES section (Anonymous) and the Response section. The Response section shows two status codes: 200 and 404. The 200 status code has a green background, while the 404 status code has an orange background. The 404 status code is currently selected, and its details are shown in the STATUS 404 panel:

- Body: application/json object
- statusCode required string
- message required string

Add a 404 Not Found HTTP status code response to a GET method

1. Return to API Designer.
2. In the /customers resource, add a new line under application/json in the get method.
3. Press backspace or delete two times to go back two tab spaces so the cursor is at the same indent level as the 200.
4. Type 4, then scroll through the shelf to locate 404 and click it.

The screenshot shows the RAML 1.0 code editor with the following content. A red X icon is placed next to line 11, indicating an error or warning:

```
5  /customers:
6  |   get:
7  |   |       responses:
8  |   |       |   200:
9  |   |       |       body:
10 |   |       |       |   application/json:
11 |   |       |       |       4
12 |   |       |       |       post:
13 |   |       |       |       responses:
14 |   |       |       |       |   201:
15 |   |       |       |       |       headers:
16 |   |       |       |       |       Location:
17 |   |       |       |       |       body:
18 |   |       |       |       |       application/json:
19 |   |       |       |       |       /{customer_id}:
```

Below the code editor is a shelf containing icons for various HTTP status codes: 401, 402, 403, 404, and 405. The 404 icon is highlighted, indicating it is the selected status code for addition.

- In the shelf, click body.
- In the shelf, click application/json.

```

5 ▼ | /customers:
6 ▼ |   get:
7 ▼ |     responses:
8 ▼ |       200:
9 |         body:
10 |           application/json:
11 ▼ |       404:
12 ▼ |         body:
13 |           application/json:
14 |
15 ▼ |   post:

```

Add a custom error message for the response

- In the new line below application/json, type the letter p and in the shelf, click properties.

```

11 ▼ |   404:
12 ▼ |     body:
13 |     application/json:
14 |       p|
15 ▼ |   post:
16 ▼ |     responses:
17 ▼ |       201:
18 |         headers:
19 |           Location:
20 |

```

OTHERS (1)

 properties

- In the new line below properties, type statusCode:
- Press spacebar and scroll down in the shelf and click string.

```

11 ▼ |   404:
12 ▼ |     body:
13 ▼ |     application/json:
14 ▼ |     properties:
15 |       statusCode: string
16 |

```

- Press enter and type message: in the new line.

11. Press spacebar and scroll down in the shelf and click string.

```
5 ▼ /customers:  
6 ▼   get:  
7 ▼     responses:  
8 ▼       200:  
9         body:  
10        | application/json:  
11 ▼       404:  
12 ▼         body:  
13 ▼           application/json:  
14 ▼             properties:  
15               statusCode: string  
16               message: string
```

Add the HTTP 404 response body to all GET and DELETE methods

12. Copy the 404 response body you just created.

```
5 ▼ /customers:  
6 ▼   get:  
7 ▼     responses:  
8 ▼       200:  
9         body:  
10        | application/json:  
11 ▼       404:  
12 ▼         body:  
13 ▼           application/json:  
14 ▼             properties:  
15               statusCode: string  
16               message: string
```

13. Add the 404 response body to the get methods of the following resources:

- /customers/{customer_id}
- /accounts
- /accounts/{account_id}
- /accounts/{account_id}/transactions
- /transactions/{transaction_id}

```
24 ▼   /{customer_id}:  
25     uriParameters:  
26       customer_id:  
27 ▼     get:  
28 ▼       responses:  
29 ▼         200:  
30           body:  
31           | application/json:  
32 ▼         404:  
33 ▼           body:  
34 ▼             application/json:  
35 ▼               properties:  
36                 statusCode: string  
37                 message: string
```

14. Add the 404 response body to the delete methods of the following resources:

- /customers/{customer_id}
- /accounts/{account_id}

```
24 ▼  |  /{customer_id}:  
25    |  |  uriParameters:  
26    |  |  |  customer_id:  
27 ▼  |  |  get:  
28 ▼  |  |  |  responses:  
29 ▼  |  |  |  |  200:  
30    |  |  |  |  |  body:  
31    |  |  |  |  |  |  application/json:  
32 ▼  |  |  |  |  |  404:  
33 ▼  |  |  |  |  |  body:  
34 ▼  |  |  |  |  |  |  application/json:  
35 ▼  |  |  |  |  |  properties:  
36    |  |  |  |  |  |  statusCode: string  
37    |  |  |  |  |  |  message: string  
38 ▼  |  |  patch:  
39    |  |  |  responses:  
40    |  |  |  |  204:  
41 ▼  |  |  |  delete:  
42 ▼  |  |  |  |  responses:  
43 ▼  |  |  |  |  |  200:  
44    |  |  |  |  |  |  body:  
45    |  |  |  |  |  |  |  application/json:  
46 ▼  |  |  |  |  |  |  404:  
47 ▼  |  |  |  |  |  |  body:  
48 ▼  |  |  |  |  |  |  |  application/json:  
49 ▼  |  |  |  |  |  |  properties:  
50    |  |  |  |  |  |  |  statusCode: string  
51    |  |  |  |  |  |  |  message: string
```

View the API documentation

15. In the API Console, click the GET for the /customers resource.

16. Scroll down to the Response section and click the 404 tab; you should see the custom error message properties listed under the application/json object.

Response

STATUS 404

200

404

Body application/json

application/json object

statusCode required string

message required string



17. Scroll up and click CLOSE X.

Walkthrough 5-3: Add response bodies to return error status codes and messages for server-side errors

In this walkthrough, you add response bodies to return error status codes and error messages for server-side errors. You will:

- Add necessary HTTP 5xx status code responses to all PATCH, PUT and POST methods.
- Specify error messages to be returned.

The screenshot shows the MuleSoft API Designer interface. On the left, the API configuration for the `/accounts` endpoint's `post` method is displayed. Lines 77-80 define the `responses` block, which includes `201` and `503` status codes. The `503` response has a `body` section with `statusCode` and `message` properties. On the right, the generated API documentation shows a `201` response and a `503` response. The `503` response is highlighted with a red border. The documentation details the `statusCode` and `message` properties.

Add a 501 Not Implemented HTTP status code response to the PATCH method

1. Return to API Designer.
2. In the `/customers` resource, add a new line under 204 in the patch method.
3. Type 5, then scroll through the shelf to locate 501 and click it.

The screenshot shows the MuleSoft API Designer interface. The configuration for the `/customers` resource's `PATCH` method is shown. Lines 38-40 define the `responses` block for `204`. Line 41 adds a new line for a `5` status code. The `5` status code is selected from a shelf on the right, which also contains `500`, `501`, and `502`.

4. In the shelf, click body.

5. In the shelf, click application/json.

```
38 ▼ | patch:      +-----+
39 ▼ |   responses:  |
40 |     204:      |
41 ▼ |     501:      |
42 ▼ |       body:   |
43 |         application/json: |
44 |           +-----+
45 ▼ | delete:    |
```

Add a custom error message for the response

6. In the new line below application/json, type the letter p and in the shelf, click properties.
7. In the new line below properties, type statusCode::
8. Press spacebar and scroll down in the shelf and click string.

```
38 ▼ | patch:      +-----+
39 ▼ |   responses:  |
40 |     204:      |
41 ▼ |     501:      |
42 ▼ |       body:   |
43 ▼ |         application/json: |
44 |           properties: |
45 |             statusCode: string
```

9. Press enter and type message: in the new line.
10. Press spacebar and scroll down in the shelf and click string.

```
38 ▼ | patch:      +-----+
39 ▼ |   responses:  |
40 |     204:      |
41 ▼ |     501:      |
42 ▼ |       body:   |
43 ▼ |         application/json: |
44 |           properties: |
45 |             statusCode: string
46 |             message: string
```

Add the HTTP 501 response body to the PUT method

11. Copy the 501 response body you just created.

12. Add the 501 response body to the put method of the /accounts/{account_id} resource.

```
98 ▼ |  put:
99 ▼ |    responses:
100 ▼ |      200:
101 |        body:
102 |          application/json:
103 ▼ |      201:
104 |        body:
105 |          application/json:
106 ▼ |      501:
107 ▼ |        body:
108 ▼ |          application/json:
109 ▼ |            properties:
110 |              statusCode: string
111 |              message: string
112 ▼ |  /transactions:
```

Add the 503 Service Unavailable HTTP status code response to a POST method

13. In the /customers resource, add a new line under application/json in the post method.
14. Press backspace or delete two times to go back two tab spaces so the cursor is at the same indent level as the 201.
15. Type 5, then click 503 in the shelf.

```
17 ▼ |  post:
18 ▼ |    responses:
19 ▼ |      201:
20 |        headers:
21 |          Location:
22 |        body:
23 |          application/json:
24 |          5|  
25 ▼ |        /{customer_id}:
26 |          uriParameters:
```

OTHERS (9)

-  500
-  501
-  502
-  503

16. In the shelf, click body.
17. In the shelf, click application/json.
18. In the shelf, click properties.
19. In the new line below the properties line, type statusCode:..

20. Press spacebar and scroll down the shelf and select string.
21. In the new line press enter and type message:.
22. Press spacebar and scroll down the shelf and select string.
23. Press enter and type message: in the new line.
24. Press spacebar and scroll down in the shelf and click string.

```

17 ▼ | post:
18 ▼ |   responses:
19 ▼ |     201:
20 |       headers:
21 |         Location:
22 |       body:
23 |         application/json:
24 ▼ |     503:
25 ▼ |       body:
26 ▼ |         application/json:
27 ▼ |           properties:
28 |             statusCode: string
29 |             message: string

```

Add the HTTP 503 response body to all POST methods

25. Copy the 503 response body you just created.
26. Add the 503 response body to the post methods of the following resources:

- /accounts
- /transactions

<pre> 77 ▼ /accounts: 78 ▼ post: 79 ▼ responses: 80 ▼ 201: 81 headers: 82 Location: 83 body: 84 application/json: 85 ▼ 503: 86 ▼ body: 87 ▼ application/json: 88 ▼ properties: 89 statusCode: string 90 message: string </pre>	<pre> 137 ▼ /transactions: 138 ▼ post: 139 ▼ responses: 140 ▼ 201: 141 headers: 142 Location: 143 body: 144 application/json: 145 ▼ 503: 146 ▼ body: 147 ▼ application/json: 148 ▼ properties: 149 statusCode: string 150 message: string </pre>
--	--

27. Save the project.

View the API documentation

28. In the API Console, click POST for the /customers resource.

29. Scroll down to the Response section and click the 503 tab; you should see the custom error message properties listed under the application/json object.

Response

STATUS 503

Body application/json object

statusCode required string

message required string

30. Scroll up and click CLOSE X.

Walkthrough 5-4: Add flexible content-types to HTTP responses

In this walkthrough, you add flexible content-types to the HTTP responses for retrieving the customer information including accounts and transactions. You will:

- Add XML body type to the HTTP 200 responses for all GET methods.
- Add an optional accept header to specify the type of response body requested.
- Add HTTP status code errors for when unsupported content types are requested.

The screenshot shows a code editor on the left and a 'Response' configuration panel on the right. The code editor contains several lines of JSON-like configuration, with line numbers 181 through 187 visible. The configuration includes sections for transaction ID, uriParameters, get method, headers (Accept?), and responses (200, 404, 406). The 'Response' panel shows a 'STATUS 406' configuration with fields for 'Body' (application/json), 'statusCode' (required string), and 'message' (required string).

Add XML body type to the HTTP 200 responses body for all GET methods

1. Return to API Designer.
2. In the /customers resource, add a new line below application/json in the 200 response of the get method.
3. In the shelf, click application/xml.
4. Delete the new line created below application/xml.

```
5 ▼ /customers:  
6 ▼   get:  
7 ▼     responses:  
8 ▼       200:  
9 ▼         body:  
10          application/json:  
11          application/xml:  
12 ▼
```

5. Add the application/xml body type to the HTTP 200 responses of the get methods of the following resources:

- /customers/{customer_id}
- /customers/{customer_id}/accounts
- /accounts/{account_id}
- /accounts/{account_id} /transactions
- /transactions/{transaction_id}

View the API documentation

6. In the API Console, click GET for the /customers resource.
7. Scroll down to the Response section and click the 503 tab; you should see both application/json and application/xml listed as possible return types.
8. Click application/XML.

Response

STATUS 200

200	Body	application/json	application/xml
404	Body	application/xml	object

9. Scroll up and click CLOSE X.

Add an optional accept header to specify the type of response body requested

10. In the /customers resource, add a new line below the get line of the get method.
11. In the shelf, click headers.

```
5 ▼  /customers:  
6 ▼    get:  
7  
8 ▼      responses:  
9 ▼        200:  
10 ▼          body:  
11            application/json:  
12            application/xml:  
13 ▼        404:  
14 ▼          body:  
15            application/json:  
16            properties:  
17              statusCode: string  
18              message: string  
19 ▼    post:
```

ROOT (1) DOCS (2) PARAMETERS (2)

protocols description queryParameters
displayName headers

12. In the shelf, click Accept.

```
5 ▼ | /customers:
6 ▼ |   get:
7 |     headers:
8 |
9 ▼ |     responses:
10 ▼ |       200:
11 ▼ |         body:
12 |           application/json:
13 |           application/xml:
14 ▼ |       404:
15 ▼ |         body:
16 ▼ |           application/json:
17 ▼ |             properties:
18 |               statusCode: string
19 |               message: string
```

IT - Range

◀ If-Unmodified-Since

◀ Accept

13. Delete the new line created below the Accept header property.

14. Go to the line with the Accept header and change it to an optional parameter by adding a question mark at the end of it.

```
5 ▼ | /customers:
6 ▼ |   get:
7 |     headers:
8 |     |   Accept?:
9 ▼ |     responses:
10 ▼ |       200:
```

Note: When you make the Accept header optional, the API can be implemented to return the response body in JSON by default when the header is not specified.

15. Add the optional Accept header to the get methods of the following resources:

- /customers/{customer_id}
- /customers/{customer_id}/accounts
- /accounts/{account_id}
- /accounts/{account_id}/transactions
- /transactions/{transaction_id}

View the API documentation

16. In the API Console, click GET for the /customers/{customer_id}/accounts resource.
17. In the Headers section for the request, locate the Accept header.

The screenshot shows a modal window for a GET request to the endpoint /customers/{customer_id}/accounts. The 'Request' tab is selected. A red bar at the top of the request body area contains the message 'Try-it is disabled because **baseUri** is not present'. Below this, the 'DESCRIPTION' and 'URI PARAMETERS' sections are visible, along with a parameter 'customer_id' with the type 'string'. The 'HEADERS' section shows an 'Accept' header with the type 'string'.

18. Click CLOSE X.

Add HTTP status code errors for when unsupported content types are requested

19. In the /customers resource, add a new line below the message property in the HTTP 404 response body.
20. Press backspace or delete four times to go four tab spaces back.

21. Type 4 and in the shelf, scroll down and click 406.

```
5 ▼ /customers:  
6 ▼   get:  
7     |   headers:  
8     |     Accept?:  
9     |   responses:  
10    |     200:  
11     |       body:  
12       |         application/json:  
13       |         application/xml:  
14     |     404:  
15     |       body:  
16       |         application/json:  
17       |           properties:  
18           |             statusCode: string  
19           |             message: string  
20  
21     post:  
22     |   responses:  
23       201  
24  
25
```



22. In the shelf, click body.

23. In the shelf, click application/json.

24. In the shelf, click properties.

25. Add a property called statusCode of type string.

26. Add a property called message of type string.

```
5 ▼ /customers:  
6 ▼   get:  
7     |   headers:  
8     |     Accept?:  
9     |   responses:  
10    |     200:  
11     |       body:  
12       |         application/json:  
13       |         application/xml:  
14     |     404:  
15     |       body:  
16       |         application/json:  
17       |           properties:  
18           |             statusCode: string  
19           |             message: string  
20     406:  
21     |       body:  
22       |         application/json:  
23       |           properties:  
24           |             statusCode: string  
25           |             message: string
```

27. Copy the 406 response body you just created.

28. Add the 406 response body to the post methods of the following resources:

- /customers/{customer_id}
- /customers/{customer_id}/accounts
- /accounts/{account_id}
- /accounts/{account_id}/transactions
- /transactions/{transaction_id}

29. Save the project.

View the API documentation

30. In API Console, click GET for the /customers resource.

31. In the Response section, click the 406 tab to view the HTTP 406 response body.

The screenshot shows the 'Response' section of the API Console. On the left, there is a vertical menu with three tabs: '200' (green), '404' (orange), and '406' (orange, highlighted with a red border). To the right of the tabs, the word 'STATUS' is followed by '406'. Below this, the word 'Body' is followed by 'application/json'. Under 'application/json', the word 'object' is shown. There are two input fields: one for 'statusCode' (type 'string') and one for 'message' (type 'string').

32. Click CLOSE X.

Walkthrough 5-5: Add caching information to HTTP methods

In this walkthrough, you indicate that a resource is cacheable by using cache-control headers. You will:

- Add a Cache-Control header to a GET response to enable caching.
- Add an Expires header to a GET response to set the date when the cached resource becomes invalid.

The screenshot shows the API Designer interface. On the left is a code editor with the following JSON configuration for the /customers resource:

```
5 ▼ /customers:
6 ▼   get:
7     headers:
8       Accept?:
9     responses:
10    200:
11      headers:
12        Cache-Control:
13        Expires:
14          type: datetime
15      body:
16        application/json:
17        application/xml:
```

On the right is a "Response" panel with the following details:

STATUS	CODE
200	
404	
406	

Below the status codes, the panel lists:

- Headers: Cache-Control required string
- Expires required datetime
- Body: application/json application/xml
application/json object

Add a Cache-Control header to a GET response to enable caching

1. Return to API Designer.
2. In the get method for the /customers resource, add a new line below the line of code with 200.
3. In the shelf, click headers.

The screenshot shows the API Designer interface with the following updated code for the /customers resource:

```
5 ▼ /customers:
6 ▼   get:
7     headers:
8       Accept?:
9     responses:
10    200:
11
12    body:
13      application/json:
14      application/xml:
15    404:
16    body:
17      application/json:
18      properties:
19        statusCode: string
```

At the bottom, there are three navigation tabs: "DOCS (1)", "PARAMETERS (1)", and "RESPONSE (1)". Below these tabs are three buttons: "description", "headers", and "body". The "headers" button is highlighted.

4. In the shelf, scroll down and click Cache-Control.

```
5 ▼ /customers:  
6 ▼   get:  
7     headers:  
8       Accept?:  
9     responses:  
10    200:  
11      headers:  
12  
13      body:  
14        application/json:  
15        application/xml:  
16    404:  
17      body:  
18        application/json:  
19        properties:  
...  
Content-Type  
Content-Length  
Content-Range  
Transfer-Encoding  
Cache-Control  
Expect
```

View the API documentation

5. In the API Console, click GET for the /customers resource.
6. In Response section, locate the Cache-Control header .

Response	
	STATUS 200
200	Headers
404	Cache-Control <i>required string</i>
406	Body <i>application/json</i> <i>application/xml</i>
	application/json <i>object</i>

Add an Expires headers to a GET response to set the date when the cached resource becomes invalid

7. In the editor, go to the new line after the Cache-Control header.

8. Align the cursor below the start of the Cache-Control line.

```
5 ▼ | /customers:  
6 ▼ |   get:  
7 |     headers:  
8 |       Accept?:  
9 ▼ |     responses:  
10 ▼ |       200:  
11 ▼ |         headers:  
12 |           Cache-Control:  
13 |           |
```

9. In the shelf, scroll down and click Expires.
10. In the shelf, click type in the OTHERS section.
11. In the shelf, scroll down and click datetime.

```
5 ▼ | /customers:  
6 ▼ |   get:  
7 |     headers:  
8 |       Accept?:  
9 ▼ |     responses:  
10 ▼ |       200:  
11 ▼ |         headers:  
12 |           Cache-Control:  
13 |           Expires:  
14 |             type: datetime  
15 ▼ |         body:  
16 |           application/json:  
17 |           application/xml:
```

12. Save the Project.

View the API documentation

13. In the API Console, click GET for the /customers resource.

14. In the Response section, locate the Expires header for the HTTP Status 200 response.

Response

STATUS 200

200

404

406

Headers

Cache-Control *required string*

Expires *required datetime*

Body

application/json

application/xml

application/json object

15. Scroll up and click CLOSE X.

Module 6: Modelling Data

The screenshot shows a file structure on the left and a code editor on the right. The file structure includes 'datatypes' (Account.raml, AccountOwner.raml, Address.raml, Bank.raml, Customer.raml, Money.raml, Transaction.raml), 'documentation' (acmeBankDoc.raml, acmeBankHeadline...), and 'api.raml'. The 'api.raml' file is highlighted with a blue bar at the bottom.

The code editor displays RAML 1.0 code. A red circle with a minus sign is placed over line 10, indicating a problem with that line. The code is as follows:

```
#%RAML 1.0 DataType
type: object
properties:
  customerID: string
  prefix?: string
  firstName: string
  lastName: string
  suffix?: string
  displayName: string
  address: Address
  phone: string
  email: string
  ssn: string
  dateOfBirth: date-only
```

To the right of the code, there is a detailed example object:

```
#%RAML 1.0 NamedExample
accountID: '12345'
accountType: Savings
accountNumber: '1234567890'
accountOwner:
  - #item 1
    customerID: 8f19cb50-3f57-4d38,
    displayName: John Doe,
    ssn: 123-456-7890,
  accountBalance:
    currency: USD
    amount: '8457.90'
  IBAN: GB29NWBK60161331926820
  bank:
    bankCode: NWBKGB2L
    bankName: ACME Bank
    routingNumber: '432159876'
    createdAt: 2012-03-07T00:00:00.001Z
```

Objectives:

- List datatypes and their attributes to be returned from or sent to resource methods.
- Create datatype fragments.
- Set request and response body types to datatypes.
- Create examples for datatype fragments.
- Include examples in request and response bodies.

Walkthrough 6-1: List datatypes and their attributes for an API

In this walkthrough, you identify datatypes and their attributes to be used in HTTP request and response bodies for the ACME Banking API. You will:

- List the datatypes required for the resource methods.
- Identify the attributes for each datatype.
- Create necessary additional datatypes to simplify the identified datatypes.
- Identify optional attributes in datatypes.

Attributes for Customer datatype along with each attribute type:

- customerID	type: string
- prefix?	type: string
- firstName	type: string
- middleName?	type: string
- lastName	type: string
- suffix?	type: string
- displayName	type: string
- address	type: Address
- phone	type: string
- email	type: string
- ssn	type: string
- dateOfBirth	type: date-only

Attributes for Account datatype along with each attribute type:

- accountID	type: string
- accountType	type: enum[Checking, Savings, Overdraft Savings, Credit Card]
- accountNumber	type: string
- accountOwner	type: AccountOwner[]
- accountBalance	type: Money (since it should include currency and an amount)
- IBAN	type: string
- bank	type: Bank
- interestRate?	type: number
- createdAt	type: datetime
- modifiedAt?	type: datetime

Attributes for Transaction datatype along with each attribute type:

- transactionID	type: string
- fromAccount	type: Account
- toAccount	type: Account
- transactionType	type: enum[atm, check, deposit, cashWithdrawal, onlineTransfer, sepa]
- transactionName?	type: string
- transactionAmount	type: Money
- newAccountBalance	type: Money
- postedAt	type: datetime
- completedAt?	type: datetime

Attributes for Address datatype:

- addressLine1	type: string
- addressLine2?	type: string
- city	type: string
- state	type: string
- country	type: string
- zipCode	type: string

List out the base datatypes required for the resource methods

1. Navigate to the studentFiles folder on your computer.
2. Navigate to the module08 directory and open the WT8-1 Datatypes and attributes text file in a text editor.

Identify the attributes for each datatype

3. Brainstorm with the class about possible attributes for each datatype.

Note: The instructor may break you out into smaller groups to first brainstorm with some of your peers.

4. After a brief discussion, type the following attributes and their types for the Customer datatype:

- customerID type: string
- prefix type: string
- firstName type: string
- middleName type: string
- lastName type: string
- suffix type: string
- displayName type: string
- addressLine1 type: string
- addressLine2 type: string
- city type: string
- state type: string
- country type: string
- zipCode type: string
- phone type: string
- email type: string
- ssn type: string
- dateOfBirth type: date-only

5. After a brief discussion, type the following attributes and their types for the Account datatype:

- accountID type: string
- accountType type: enum [Checking, Savings, Overdraft Savings, Credit Card]
- accountNumber type: string
- accountOwner type: Customer[]
- accountBalance type: Money (since it should include currency and an amount)
- IBAN type: string
- bankCode type: string
- routingNumber type: string
- bankName type: string

- swiftBIC type: string
- interestRate type: number
- createdAt type: datetime
- modifiedAt type: datetime

6. After a brief discussion, type the following attributes and their types for the Transaction datatype:

- transactionID type: string
- fromAccount type: Account
- toAccount type: Account
- transactionType type: enum [atm, check, deposit, cashWithdrawal, online, sepa]
- transactionName type: string
- transactionAmount type: Money
- newAccountBalance type: Money
- postedAt type: datetime
- completedAt type: datetime

Create necessary additional user-defined datatypes as per the attribute definition

7. In the Customer datatype, select attributes from addressLine1 to zipCode and cut the lines and paste them at the end of the document.

- lastName type: string
- suffix type: string
- phone type: string
- email type: string
- ssn type: string
- dateOfBirth type: date-only

Attributes for Account datatype along with each attribute type:

- ID type: string
- accountType type: enum[Checking, Savings, Overdraft Savings, Credit Card]
- accountNumber type: string
- accountOwner type: Customer[]
- accountBalance type: Money (since it should include currency and an amount)
- IBAN type: string
- bankID type: string
- swiftBIC type: string
- interestRate type: number
- createdAt type: datetime
- modifiedAt type: datetime

Attributes for Transaction datatype along with each attribute type:

- ID type: string
- fromAccount type: Account
- toAccount type: Account
- transactionType type: enum[atm, check, deposit, cashWithdrawal, onlineTransfer]
- transactionName type: string
- transactionAmount type: Money
- newAccountBalance type: Money
- postedAt type: datetime
- completedAt type: datetime

- addressLine1 type: string
- addressLine2 type: string
- city type: string
- state type: string
- country type: string
- zipCode type: string

8. In the empty line, in between the Customer datatype attributes, type:

- address type: Address

9. For the attributes that were to the end of the document, add a header to indicate them as attributes for Address datatype.

- completedAt type: ~~datetime~~

Attributes for Address datatype:

- addressLine1 type: string
- addressLine2 type: string
- city type: string
- state type: string
- country type: string
- zipCode type: string

10. Similarly, create a new datatype Money with the following attributes:

- currency type: string
- amount type: string

11. In the Account datatype, go to the line that contains the accountOwner attribute and change the type to AccountOwner[].

Attributes for Account datatype along with each attribute type:

- ID type: string
- accountType type: enum[Checking, Savings, Overdraft Savings, Credit Card]
- accountNumber type: string
- accountOwner type: AccountOwner[]
- accountBalance type: Money (since it should include currency and an amount)
- IBAN type: string

Note: Although using Customer data type for accountOwner attribute is valid, we create an AccountOwner datatype because you only need a subset of information (name, ID and ssn) about the customer to add them as an account owner. This way the entire customer datatype and their attributes are not necessary.

12. Create a new datatype AccountOwner with the following attributes:

- customerID type: string
- ssn type: string
- displayName type: string

13. In the Account datatype, select the attributes bankCode, bankName and swiftBIC and move them to end of the document and name them as attributes for Bank datatype.

Attributes for Account datatype along with each attribute type:

- ID	type: string
- accountType	type: enum[Checking, Savings, Overdraft Savings, Credit Card]
- accountNumber	type: string
- accountOwner	type: AccountOwner[]
- accountBalance	type: Money (since it should include currency and an amount)
- IBAN	type: string
- bankCode	type: string
- bankName	type: string
- swiftBIC	type: string
- interestRate	type: number
- createdAt	type: datetime
- modifiedAt	type: datetime

14. Reference the bank datatype within the Account datatype as:

- bank type: Bank.

Identify optional attributes in the datatypes

15. Brainstorm with the class about the attributes that could be optional for each datatype.

16. Identify the optional attributes and add a ? right after the attribute name.

Attributes for Customer datatype along with each attribute type:

- ID	type: string
- prefix?	type: string
- firstName	type: string
- middleName?	type: string
- lastName	type: string
- suffix?	type: string
- displayName	type: string
- address	type: Address
- phone	type: string
- email	type: string
- ssn	type: string
- dateOfBirth	type: date-only

Attributes for Account datatype along with each attribute type:

- ID	type: string
- accountType	type: enum[Checking, Savings, Overdraft Savings, Credit Card]
- accountNumber	type: string
- accountOwner	type: AccountOwner[]
- accountBalance	type: Money (since it should include currency and an amount)
- IBAN	type: string
- bank	type: Bank
- interestRate?	type: number
- createdAt	type: datetime
- modifiedAt?	type: datetime

Attributes for Transaction datatype along with each attribute type:

- ID	type: string
- fromAccount	type: Account
- toAccount	type: Account
- transactionType	type: enum[atm, check, deposit, cashWithdrawal, onlineTransfer, sepa]
- transactionName?	type: string
- transactionAmount	type: Money
- newAccountBalance	type: Money
- postedAt	type: datetime
- completedAt?	type: datetime

Attributes for Address datatype:

- addressLine1	type: string
- addressLine2?	type: string
- city	type: string
- state	type: string
- country	type: string
- zipCode	type: string

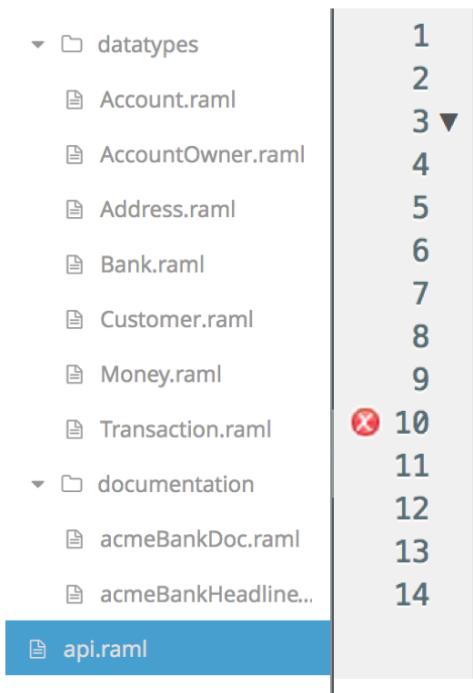
Attributes for the Bank datatype:

- bankCode	type: string
- bankName	type: string
- swiftBIC?	type: string
- routingNumber?	type: string

Walkthrough 6-2: Create datatype fragments

In this walkthrough, you define datatype fragments for the identified datatypes for the ACME Banking API. You will:

- Create individual datatype fragment files for the identified datatypes.
- Define the datatypes with required and optional attributes.



```
1  %%RAML 1.0 DataType
2  type: object
3  ▼ properties:
4    customerID: string
5    prefix?: string
6    firstName: string
7    lastName: string
8    suffix?: string
9    displayName: string
10   address: Address
11   phone: string
12   email: string
13   ssn: string
14   dateOfBirth: date-only
```

Refactor the custom error message object to use a datatype

1. Return to API Designer.
2. Click the Project menu and select New Folder.
3. In the Add a new folder dialog box, set the name to datatypes.

Add a new folder

Input a name for your new folder:

Name *

datatype

Cancel

OK

4. Click OK.

5. In the file browser, click api.raml and go to the /customers resource get method HTTP 404 response body and copy the lines from properties up until the message string.

```
16 |           application/json:  
17 |           application/xml:  
18 ▼         404:  
19 ▼           body:  
20 ▼             application/json:  
21 ▼               properties:  
22                 statusCode: string  
23                 message: string
```

6. In the file browser section, right-click the datatypes folder and select New File > RAML 1.0 > Type.
7. In the Add new Type file dialog box, set the name to CustomErrorMessage.raml and click OK.
8. Go to the end of the first line in the new file and press enter.
9. Type the following:

```
type: object
```

10. Press enter and paste the copied lines below.

```
1  #%%RAML 1.0 DataType  
2  type: object  
3 ▼   properties:  
4    |   statusCode: string  
5    |   message: string
```

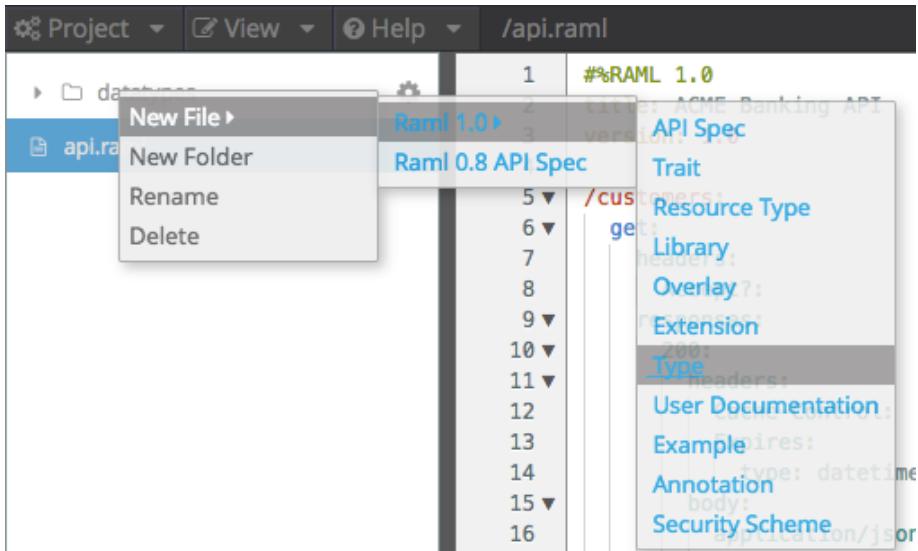
11. Select the statusCode and message properties line and press Shift+Tab to indent the lines properly.

```
1  #%%RAML 1.0 DataType  
2  type: object  
3 ▼   properties:  
4     statusCode: string  
5     message: string
```

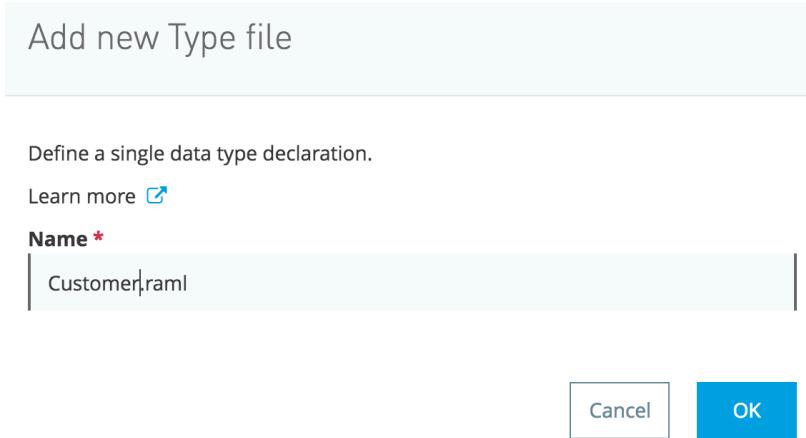
12. Save the file.

Create a Customer datatype fragment

13. In the file browser section, right-click the datatypes folder and select New File > RAML 1.0 > Type.



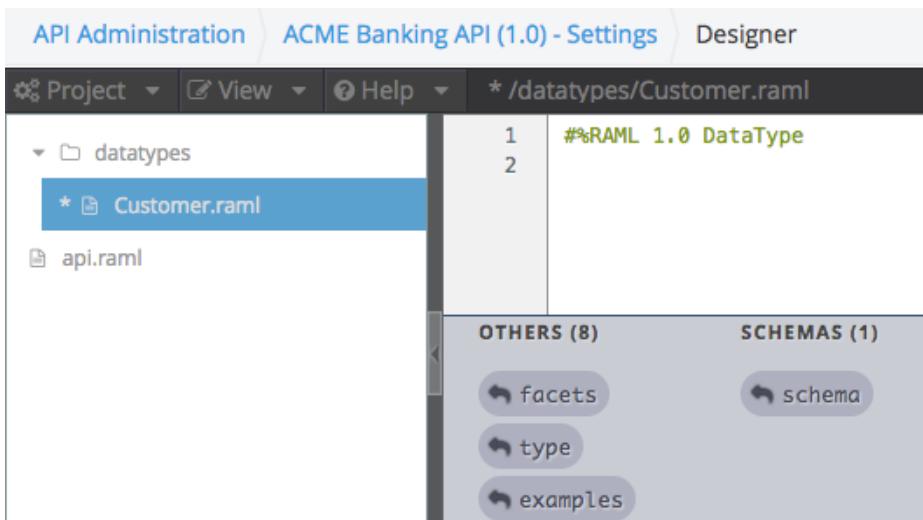
14. In the Add new type dialog box, set the name to Customer.raml.



15. Click OK.

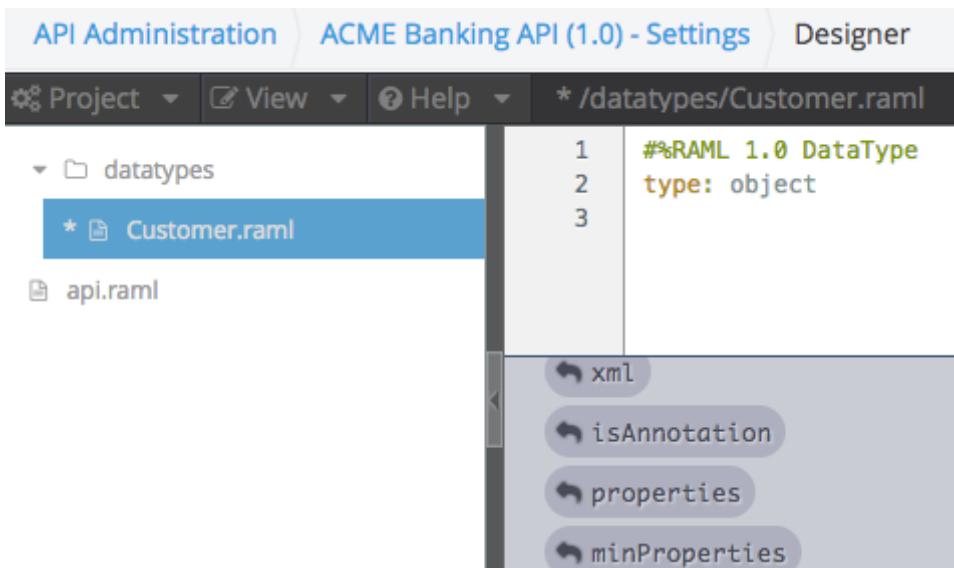
16. In the RAML editor, go to the end of the first line and press enter to add a new line.

17. In the shelf, click type.



18. In the shelf, click object and press enter.

19. In the shelf, click properties.



20. In your text editor, return to the WT 6-1 Datatypes and attributes file.

21. Review the attributes of the Customer datatype.

22. Return to API Designer.

23. Add the attributes for the Customer one below the other; for each, specify the attribute name, followed by a colon, and then the attribute type.

24. Add a ? after the prefix and suffix attribute names to specify them as optional attributes.

```
1  #%RAML 1.0 DataType
2  type: object
3 ▼ properties:
4    customerID: string
5    prefix?: string
6    firstName: string
7    lastName: string
8    suffix?: string
9    displayName: string
10   address: Address
11   phone: string
12   email: string
13   ssn: string
14   dateOfBirth: date-only
```

Note: The red cross in the address attribute line can be ignored. While linking the datatype fragments to the api.raml the errors won't persist in the main RAML definition or the API Console.

25. Save the project.

Create an Account datatype fragment

26. Similarly, add another new type file and set its name to Account.raml.

Add new Type file

Define a single data type declaration.

Learn more 

Name *

Account.raml

Cancel

OK

27. In the RAML editor, add required and optional attributes for the Account datatype.

```
1  #RAML 1.0 DataType
2  type: object
3 ▼ properties:
4    accountID: string
5    accountType:
6      | enum: [Checking, Savings, Overdraft Savings, Credit Card]
7    accountNumber: string
8    accountOwner: AccountOwner[]
9    accountBalance: Money
10   IBAN: string
11   bank: Bank
12   interestRate?: number
13   createdAt: datetime
14   modifiedAt?: datetime |
```

Note: Ignore the red errors in the three lines that inherit types from user-defined attributes that are yet to be defined.

28. Save the project.

Create a Transaction datatype fragment

29. Similarly, create another new Type file called Transaction.raml and type the required and optional attributes for the Transaction datatype.
30. Use the enum datatype to specify the transactionType has possible values of atm, deposit, cashWithdrawal, sepa, and online.

Note: The enum datatype should be defined in a line below the attribute as a key-value pair.

```
1  #RAML 1.0 DataType
2  type: object
3 ▼ properties:
4    transactionID: string
5    fromAccount: Account
6    toAccount: Account
7    transactionType:
8      | enum: [atm, deposit, cashWithdrawal, sepa, online]
9    transactionName?: string
10   transactionAmount: Money
11   newAccountBalance: Money
12   postedAt: datetime
13   completedAt?: datetime|
```

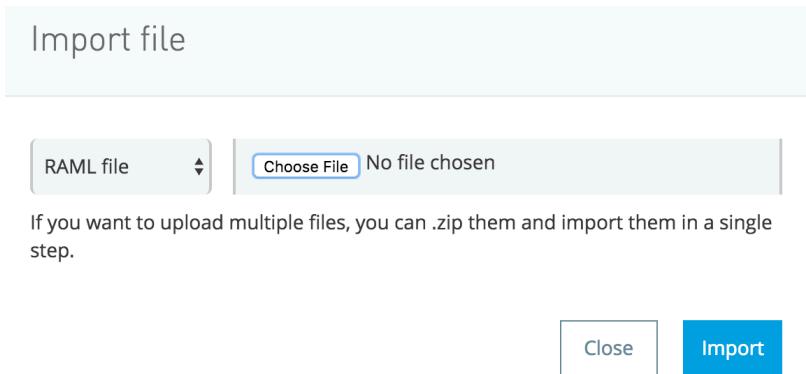
Note: Ignore the red errors in some of the user-defined datatypes reference. Once the files are referenced in the main RAML file, the errors won't affect the API design specification.

31. Save the project.

Import other datatype fragment files into API Designer

32. In API Designer, click the Project menu and click Import.

33. In the Import file dialog box, click the Choose File button.



34. In the File browser, locate the studentFiles directory, open the module08 folder, and double-click Address.raml.

35. Click Import.

36. In the left-side navigation, click Address.raml and drag-and-drop it inside the datatypes folder.

37. Similarly, import Money.raml, AccountOwner.raml and Bank.raml and drag them into the datatypes directory.

38. Verify the project structure has the datatype files inside the datatypes folder, the documentation files in the documentation folder, and the api.raml file in the root location.

Include the datatype fragments in the main RAML API definition

39. In the file browser section, select api.raml.

40. In the RAML editor, go to the end of the documentation line that includes the acmeBankDoc.raml documentation file and press enter to add a new line below it.

41. Press enter and place the cursor at the beginning of the line.

42. In the shelf, scroll down and click types.

```
1  #%RAML 1.0
2  title: ACME Banking API
3  version: 1.0
4
5  |
6  ▼ /customers:
7  ▼   get:
8    |   headers:
9    |     Accept?:
10  ▼    responses:
```

RESOURCES (1) **OTHERS (3)**

- New Resource
- uses
- types**
- annotationTypes

43. In the new line type,

Customer: !

Note: Make sure there is a space between the colon after Customer and the !.

44. In the shelf, click include.

45. Press spacebar and in the shelf, click datatypes.

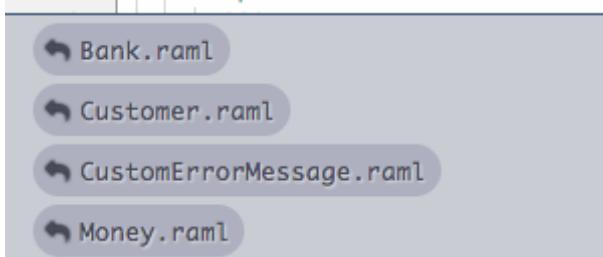
```
5  types:
6  |   Customer: !include
7  ▼ /customers:
8  ▼   get:
9    |   headers:
10   |     Accept?:
```

OTHERS (2)

- datatypes
- api.raml

46. Type / and in the shelf, click Customer.raml.

```
5 ▼ types:  
6   | Customer: !include datatypes/  
7  
8 ▼ /customers:  
9 ▼   get:  
10    | headers:  
11    |   Accept?:  
12 ▼   responses:
```



47. Similarly, include rest of the datatypes in the api.raml file.

48. Press enter to add an empty line in between the types and the /customers resource.

```
1  #%RAML 1.0  
2  title: ACME Banking API  
3  version: 1.0  
4  
5 ▼ types:  
6   | Customer: !include datatypes/Customer.raml  
7   | Account: !include datatypes/Account.raml  
8   | Transaction: !include datatypes/Transaction.raml  
9   | Address: !include datatypes/Address.raml  
10  | AccountOwner: !include datatypes/AccountOwner.raml  
11  | Money: !include datatypes/Money.raml  
12  | Bank: !include datatypes/Bank.raml  
13  | CustomErrorMessage: !include datatypes/CustomErrorMessage.raml  
14  
15 ▼ /customers:
```

49. Save the project.

View the documentation

50. In the API Console, click the arrow in the Types section to expand it.

The screenshot shows the API Console interface. At the top, there is a header with the text "Mocking Service" and a toggle switch labeled "OFF". Below this, a sidebar on the left has a section titled "Types" with a right-pointing arrow. The main content area is titled "Resources" and contains a checkbox labeled "API is behind a firewall (?)". A "Collapse All" button is also present. Under "Resources", there is a collapsed section for "/customers". When expanded, it shows three endpoints: "POST" (green), "GET" (blue), and "DELETE" (red). Below these, another endpoint "PATCH" (blue) is shown, followed by "GET" (blue) for the path "/customers/{customer_id}/accounts".

51. Verify that you see all the datatypes listed.

The screenshot shows the API Console interface with the "Types" section expanded. It lists several data types as objects, each preceded by a right-pointing arrow: "Customer object", "Account object", "Transaction object", "Address object", "AccountOwner object", "Money object", "Bank object", and "CustomErrorMessage object".

Walkthrough 6-3: Validate datatype attribute values using patterns

In this walkthrough, you add patterns and format values to validate necessary datatype attributes. You will:

- Specify attribute patterns and format values for certain datatype attributes.

```
1  #%RAML 1.0 DataType
2  type: object
3  ▼ properties:
4  ▼   currency:
5    |   type: string
6    |   pattern: ^[A-Z]{3,3}$
7  ▼   amount:
8    |   type: string
9    |   pattern: ^[+|-]?\d*\.\d{2}$
```

▼ Account object

IBAN required, matching ^[A-Z]{2,2}[0-9]{2,2}[a-zA-Z0-9]{1,30}\$ string

accountBalance required Money

accountID required string

accountNumber required string

accountOwner required AccountOwner[]

accountType required, one of [Checking, Savings, Overdraft Savings, Credit Card] string

bank required Bank

createdAt required datetime

Specify attribute patterns and format values for necessary datatype attributes

- Return to API Designer.
- In the file browser section, expand the datatypes folder and select Account.raml.
- Go to the line that contains the IBAN attribute and delete the type string from the line.
- Press enter and press tab.

```
accountNumber: string
accountOwner: AccountOwner[]
accountBalance: Money
IBAN:
|
|   bank: Bank
|   interestRate?: number
|   createdAt: datetime
|   modifiedAt?: datetime
```

- Type:

```
type: string
```

Note: Make sure you add a space between the colon after type and the value string.

6. In the shelf, click pattern.

```
10 ▼ | IBAN:  
11 |   type: string  
12 |  
13 |   bank: Bank  
14 |   interestRate?: number  
15 |   createdAt: datetime  
16 |   modifiedAt?: datetime
```

example required examples
description minLength xml
maxLength isAnnotation
enum uses
pattern

7. Return to or open the course snippets.txt file.
8. Locate the Module 8 section and copy the pattern for IBAN.
9. Return to API Designer and paste the text in the value of the pattern node.

```
10 ▼ | IBAN:  
11 |   type: string  
12 |   pattern: ^[A-Z]{2,2}[0-9]{2,2}[a-zA-Z0-9]{1,30}$
```

10. Go to the line that contains the interestRate attribute and delete number attribute type from the line.
11. Press enter and press tab.
12. Set the type to number and press enter.

```
type: number  
13 |   bank: Bank  
14 |   interestRate?:  
15 |     type: number  
16 |   createdAt: datetime  
17 |   modifiedAt?: datetime
```

13. In the shelf, click format and specify the format value as double.

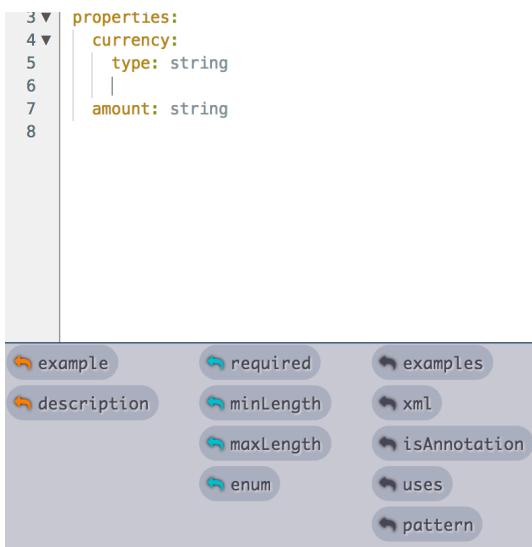
```
14 ▼ | interestRate?:  
15 |   type: number  
16 |   format: double  
17 |   createdAt: datetime  
18 |   modifiedAt?: datetime
```

14. Save the project.

15. In the file browser section, select Money.raml inside the datatypes folder.
16. In the editor, go to the line that contains currency attribute and delete the attribute type string from the line.
17. Press enter and press tab.
18. Set the type to string and press enter.

```
type: string
```

19. In the shelf, click pattern.



20. Return to the course snippets.txt file and copy the pattern value for currency and paste it in the editor.

```
properties:
  currency:
    type: string
    pattern: ^[A-Z]{3,3}$
  amount: string
```

21. Go to the line that contains amount attribute and delete the attribute type string from the line.
22. Press enter and press tab.
23. Set the type to string and press enter.

```
type: string
```

24. In the shelf, click pattern.

25. Return to the course snippets.txt file and copy the line that contains the pattern value for amount and paste it in the RAML editor.

```
1  #%RAML 1.0 DataType
2  type: object
3 ▼ properties:
4 ▼   currency:
5     type: string
6     pattern: ^[A-Z]{3,3}$
7 ▼   amount:
8     type: string
9     pattern: ^[+|-]?\d*\.\d{2}$
```

26. Save the project.

View the API documentation

27. In the API Console, expand the Types section; you should see all the datatypes listed.

The screenshot shows a user interface for an API console. At the top, there is a navigation bar with several icons. Below the navigation bar, there is a search bar and a dropdown menu labeled "Types". The main content area is a list of datatypes, each preceded by a black triangle icon and followed by a horizontal line. The datatypes listed are: Customer, Account, Transaction, Address, AccountOwner, Money, and Bank.

- Customer *object*
- Account *object*
- Transaction *object*
- Address *object*
- AccountOwner *object*
- Money *object*
- Bank *object*

28. Expand the Account datatype and see the attributes of the Account datatype listed.

▼ Account *object*

IBAN *required, matching ^[A-Z]{2,2}[0-9]{2,2}[a-zA-Z0-9]{1,30}\$ string*

accountBalance *required Money*

accountID *required string*

accountNumber *required string*

accountOwner *required AccountOwner []*

accountType *required, one of (Checking, Savings, Overdraft Savings, Credit Card) string*

bank *required Bank*

createdAt *required datetime*

29. Click the blue AccountOwner datatype link in the accountOwner attribute and verify that you can see the attributes of the AccountOwner datatype in a pop-up dialog box.

accountOwner *required AccountOwner []*

AccountOwner

object

customerID required string

displayName required string

ssn required string

interestRate *number*

30. Click the Types section to collapse the list of datatypes.

Walkthrough 6-4: Specify datatypes in resource methods

In this walkthrough, you specify the type of data to be sent in the request or received from the response of a resource method. You will:

- Add the type parameter with a reference to the datatype for all resource method request and response bodies.

The screenshot shows a code editor on the left and a detailed API response schema on the right. The code editor contains a snippet of RAML 1.0 code for a PUT method. The response schema is for a 200 status code, showing a complex object structure with properties like IBAN, accountBalance, accountID, accountNumber, accountOwner, and accountType, each with their respective types and descriptions.

```
181 ▼
182
183 ▼
184
185
186 ▼
187 ▼
188
189
190
```

```
put:
  description: Update a specific account information
  body:
    application/json:
      type: Account
  responses:
    201:
      body:
        application/json:
    204:
```

Response

STATUS 200

Body application/json

Account object

200

404

IBAN required, matching ^[A-Z]{2,2}[0-9]{2,2}[a-zA-Z0-9]{1,30}\$ string

accountBalance required Money

accountID required string

accountNumber required string

accountOwner required AccountOwner []

accountType required, one of {Checking, Savings, Quardant Savings, Credit Card}, string

Add type parameters for the appropriate method requests and responses

1. Return to API Designer.
2. In the file browser section, click api.raml.
3. Locate the /customers resource and in the get method HTTP 200 response body, add a new line below the response body type application/json.
4. Press tab and in the shelf, click type.

The screenshot shows a code editor with a shelf at the bottom containing buttons for displayName, default, facets, example, type, description, and examples. The code editor contains a snippet of RAML 1.0 code for a GET method, showing the addition of a new line under the application/json type definition.

```
42 ▼
43
44
45
46 ▼
47 ▼
48 ▼
49
```

```
body:
  application/json:
    application/xml:
      404:
        body:
          application/json:
```

displayName default facets

example type

description examples

5. In the shelf, click Customer and add [] (square brackets) to denote an array of Customer objects as the response body.

```
40 |           example: Tue, 18 Ap
41 |           format: rfc2616
42 ▼         body:
43 |             application/json:
44 |               type: Customer[]
45 |             application/xml:
46 |
47 ▼         404:
```

6. Copy the line type: Customer[] and go to the end of the next line that contains the application/xml response body type.
7. Press enter and press tab.
8. Paste the copied line.

```
42 ▼         body:
43 |             application/json:
44 |               type: Customer[]
45 |             application/xml:
46 |               type: Customer[]
47 ▼         404:
```

9. Similarly, in the /accounts nested resource get method and the /transactions resource get method, add the respective Account and Transaction array datatypes in the HTTP 200 response body.

```
118 ▼       responses:
119 ▼         200:
120 ▼           body:
121 |             application/json:
122 |               type: Account[]
123 |             application/xml:
124 |               type: Account[]
125 ▼         404:
197 ▼       /transactions:
198 ▼         get:
199 |           description: Retrieve a list of tra
200 |             account
201 ▼           responses:
202 ▼             200:
203 |               body:
204 |                 application/json:
205 |                   type: Transaction[]
```

10. In the /customers resource post method, add a new line below the post method node.

11. In the shelf, scroll down and click body.

```
42 ▼ | post:  
43 |  
44 ▼ |   responses:  
45 ▼ |     201:  
46 |       headers:  
47 |         Location:  
48 |       body:  
  
OTHERS (1)           BODY (1)  
  
queryString          body
```

12. In the shelf, click application/json.

13. In the shelf, click type.

14. In the shelf, click Customer.

```
42 ▼ | post:  
43 ▼ |   body:  
44 |     application/json:  
45 |       type: Customer  
46 ▼ |   responses:  
47 ▼ |     201:  
48 |       headers:
```

15. Add the request body for the /accounts post method, /{account_id} put method and the /transactions post method with the respective datatypes in the request.

```
124
125 ▼ /accounts:
126 ▼   post:
127 ▼     body:
128       application/json:
129         type: Account
130 ▼     responses:
176 ▼       put:
177 ▼         body:
178           application/json:
179             type: Account
180 ▼         responses:
217 ▼ /transactions:
218 ▼   post:
219 ▼     body:
220       application/json:
221         type: Transaction
222 ▼     responses:
223 ▼       201:
```

16. Go to the /{customer_id} nested resources get method.

17. Add a new line below the HTTP 200 response body type application/json.

18. Press tab and type:

type: Customer

19. Add a new line below the application/xml body type and type:

type: Customer

```
61 ▼   get:
62     headers:
63       Accept?:
64 ▼     responses:
65 ▼       200:
66 ▼         body:
67           application/json:
68             type: Customer
69           application/xml:
70             type: Customer
```

20. Add the type node for the /{account_id} and /{transaction_id} get method with the value as the respective datatypes.

```
144 ▼  /{account_id}:
145    uriParameters:
146      account_id:
147 ▼       get:
148         headers:
149           | Accept?:
150 ▼         responses:
151 ▼           200:
152 ▼             body:
153               application/json:
154                 | type: Account
155               application/xml:
156                 | type: Account
157 ▼           404:
238 ▼  /{transaction_id}:
239    uriParameters:
240      transaction_id:
241 ▼       get:
242         headers:
243           | Accept?:
244 ▼         responses:
245 ▼           200:
246 ▼             body:
247               application/json:
248                 | type: Transaction
249               application/xml:
250                 | type: Transaction
251 ▼           404:
```

21. Save the project.

View the documentation

22. In the API Console, click the GET tab for the /accounts/{account_id} resource.

23. Verify that you can see the Account object listed in the 200 Response section of the resource.

Response

STATUS 200

200

Body application/json

Account object

IBAN required, matching ^[A-Z]{2,2}[0-9]{2,2}[a-zA-Z0-9]{1,30}\$ string

accountBalance required Money

accountID required string

accountNumber required string

accountOwner required AccountOwner []

accountType required, one of (Checking, Savings, Overdraft Savings, Credit Card) string

404

24. Click CLOSE X.

Reference the custom error message datatype in the necessary resource method responses

25. Go to the /customers resource get method HTTP 404 response body and delete the lines that contain the properties.

26. Set the datatype to CustomErrorMessage.

```
type: CustomErrorMessage
```

27. Modify all the other error response bodies in the RAML definition to be of this datatype.

```
30 ▼ 404:  
31 ▼   body:  
32     application/json:  
33       type: CustomErrorMessage  
34 ▼ 406:  
35 ▼   body:  
36     application/json:  
37       type: CustomErrorMessage|
```

28. Save the project.

Walkthrough 6-5: Define example fragments for datatypes

In this walkthrough, you create example fragments for each of the datatypes. You will:

- Create example fragments for the datatypes.
- Include example fragments in request and response bodies.

The screenshot shows the RAML 1.0 NamedExample editor interface. On the left, there is a code editor with the following RAML code:

```
#%RAML 1.0 NamedExample
accountID: '12345'
accountType: Savings
accountNumber: '1234567890'
accountOwner:
  - #item 1
    customerID: 8f19cb50-3f57-4d38,
    displayName: John Doe,
    ssn: 123-456-7890,
  accountBalance:
    currency: USD
    amount: '8457.90'
  IBAN: GB29NWBK60161331926820
  bank:
    bankCode: NWBKGB2L
    bankName: ACME Bank
    routingNumber: '432159876'
  createdAt: 2012-03-07T00:00:00.001Z
```

On the right, there is a preview area labeled "BODY" with the media type "application/json". It shows an example object:

```
{
  "transactionID": "b05f550d-1915-4def",
  "fromAccount": {
    "accountID": "12345",
    "accountType": "Savings",
    "accountNumber": "1234567890",
    "accountOwner": [
      {
        "customerID": "8f19cb50-3f57-4d38",
        "displayName": "John Doe",
        "ssn": "123-456-7890"
      }
    ],
    "accountBalance": {
      "currency": "USD",
      "amount": "8457.90"
    }
},
```

A button labeled "Examples: Example" is visible above the preview area.

Create example fragment for the Customer datatype

1. Return to API Designer.
2. Click the Project menu and select New Folder.
3. In the Add a new folder dialog box, set the name of the folder to examples.

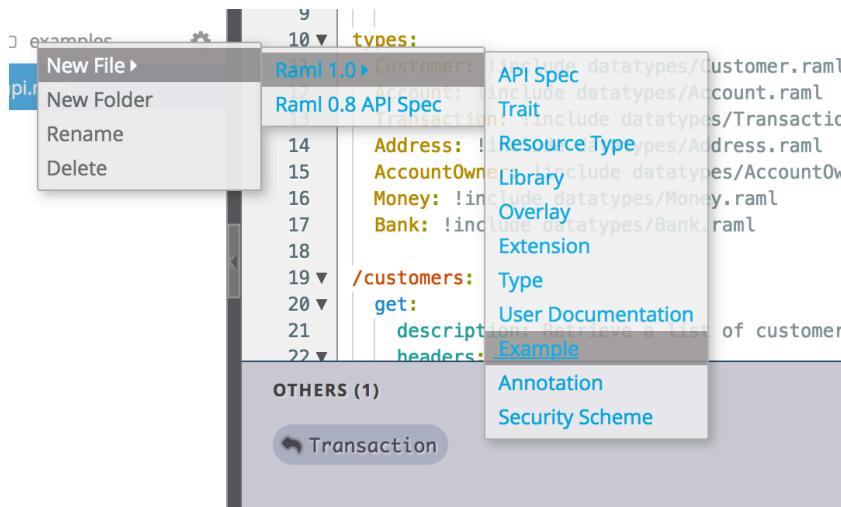
Add a new folder

Input a name for your new folder:

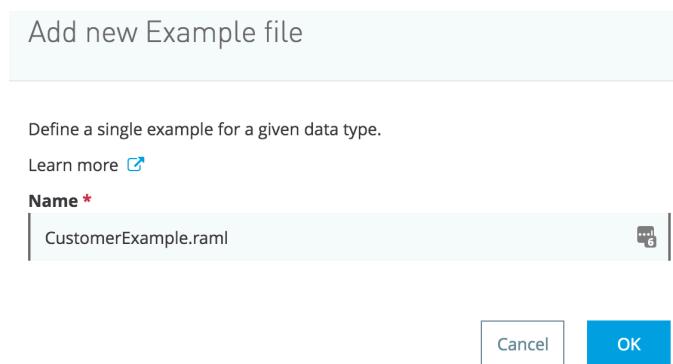
Name *

 6

4. Click OK.
5. Right-click the examples folder in the file browser section and select New File > RAML 1.0 > Example.



6. In the Add new Example file dialog box, set the name to CustomerExample.raml and click OK.



7. In the CustomerExample.raml file, delete the line that contains the value node.

8. Return to the course snippets.txt file and copy the snippet for the CustomerExample.raml content values and paste it in the second line.

```
1  %%RAML 1.0 NamedExample
2  customerID: 8f19cb50-3f57-4d38
3  firstName: John
4  lastName: Doe
5  displayName: John Doe
6 ▼ address:
7    addressLine1: 1234 Lane
8    addressLine2: Apt.#620
9    city: San Francisco
10   state: California
11   zipCode: 94108
12   phone: 415-000-0000
13   email: john doe@example.com
14   ssn: 123-456-7890
15   dateOfBirth: 1983-01-01
```

9. Save the project.

Create example fragment for the Account datatype

10. Create a new example file AccountExample.raml and copy the example data from the course snippets.txt file.

```
1  %%RAML 1.0 NamedExample
2  accountID: 12345
3  accountType: Savings
4  accountNumber: 1234567890
5 ▼ accountOwner:
6    customerID: 8f19cb50-3f57-4d38
7    displayName: John Doe
8    ssn: 123-456-7890
9 ▼ accountBalance:
10   currency: USD
11   amount: 8457.90
12   IBAN: B29NWBK60161331926820
13 ▼ bank:
14   bankCode: NWBKG2L
15   bankName: ACME Bank
16   routingNumber: 432159876
17   createdAt: 2012-03-07T00:00:00.001Z
```

11. Save the project.

Create example fragment for the Transaction datatype

12. Create a new example file called TransactionExample.raml and copy the example data from the course snippets.txt file.

```
1  %%RAML 1.0 NamedExample
2  transactionID:
3 ▼  fromAccount:
4    accountID: 12345
5    accountType: Savings
6    accountNumber: 1234567890
7 ▼  accountOwner:
8    customerID: 8f19cb50-3f57-4d38
9    displayName: John Doe
10   ssn: 123-456-7890
11 ▼  accountBalance:
12   currency: USD
13   amount: 8457.90
14   IBAN: B29NWBK60161331926820
15 ▼  bank:
16   bankCode: NWBKGB2L
17   bankName: ACME Bank
18   routingNumber: 432159876
19   createdAt: 2012-03-07T00:00:00.001Z

20 ▼  toAccount:
21   accountID: 56437
22   accountType: Credit Card
23   accountNumber: 4321987650
24 ▼  accountOwner:
25   customerID: 8f19cb50-3f57-4d38
26   displayName: John Doe
27   ssn: 123-456-7890
28   IBAN: B29NWBK60161331926820
29 ▼  bank:
30   bankCode: NWBKGB2L
31   bankName: ACME Bank
32   routingNumber: 432159876
33   createdAt: 2012-03-07T00:00:00.001Z
34   transactionType: onlineTransfer
35 ▼  transactionAmount:
36   currency: USD
37   amount: 70.00
38 ▼  newAccountBalance:
39   currency: USD
40   amount: 8387.90
```

13. Save the project.

Include example fragments in request and response bodies

14. In the file browser section, select api.raml.
15. In the /customers resource post method, add a new line below the line that contains the HTTP request body Customer datatype.

```
38 ▼ |   post:  
39 ▼ |     body:  
40 ▼ |       application/json:  
41 |         type: Customer  
42 |  
43 ▼ |       responses:
```

16. Press enter and in the shelf, click example.

```
38 ▼ |   post:  
39 ▼ |     body:  
40 ▼ |       application/json:  
41 |         type: Customer  
42 |  
43 ▼ |       responses:  
44 ▼ |         201:  
45 |           headers:  
  
DOCS (3)          PARAMETERS (1)  
  
↳ displayName    ↳ default  
↳ example  
↳ description
```

17. In the example node, type the value as:

```
!include examples/CustomerExample.raml  
  
38 ▼ |   post:  
39 ▼ |     body:  
40 ▼ |       application/json:  
41 |         type: Customer  
42 |         example: !include examples/CustomerExample.raml  
43 ▼ |       responses:
```

18. Copy the example node line.
19. In the /{customer_id} get method response body, add a new line below the line that contains the type node Customer.

20. Paste the copied line for both media types (application/json and application/xml).

```
53 ▼ |  /{customer_id}:
54 |    uriParameters:
55 |      customer_id:
56 ▼ |    get:
57 |      headers:
58 |        Accept?:
59 ▼ |      responses:
60 ▼ |        200:
61 ▼ |          body:
62 ▼ |            application/json:
63 |              type: Customer
64 |              example: !include examples/CustomerExample.raml
65 ▼ |            application/xml:
66 |              type: Customer
67 |              example: !include examples/CustomerExample.raml
68 ▼ |        404:
```

21. Similarly, add the AccountExample.raml file example node in the /accounts resource post request body as well as in /{account_id} nested resource get and put method response body.

```
128 ▼ |  /{account_id}:
129 |    uriParameters:
130 |      account_id:
131 ▼ |    get:
132 |      headers:
133 |        Accept?:
134 ▼ |      responses:
135 ▼ |        200:
136 ▼ |          body:
137 ▼ |            application/json:
138 |              type: Account
139 |              example: !include examples/AccountExample.raml
140 ▼ |            application/xml:
141 |              type: Account
142 |              example: !include examples/AccountExample.raml
143 ▼ |        404:
160 ▼ |    put:
161 ▼ |      body:
162 ▼ |        application/json:
163 |          type: Account
164 |          example: !include examples/AccountExample.raml
165 ▼ |      responses:
166 ▼ |        200:
```

22. Similarly, add the TransactionExample.raml file example node in the /transactions resource post request body as well as in /{transaction_id} nested resource get method response body.

```
196 ▼  /transactions:  
197 ▼    post:  
198 ▼      body:  
199 ▼        application/json:  
200          type: Transaction  
201          example: !include examples/TransactionExample.raml  
202 ▼        responses:  
212 ▼    /{transaction_id}:  
213      uriParameters:  
214        transaction_id:  
215 ▼      get:  
216        headers:  
217        Accept?:  
218 ▼      responses:  
219 ▼        200:  
220 ▼          body:  
221 ▼            application/json:  
222              type: Transaction  
223              example: !include examples/TransactionExample.raml  
224 ▼            application/xml:  
225              type: Transaction  
226              example: !include examples/TransactionExample.raml  
227 ▼          404:
```

23. Save the project.

View the documentation

24. In the API Console, click the POST tab for the /transactions resource.
25. Verify that you see the TransactionExample in the request body of the method.

BODY

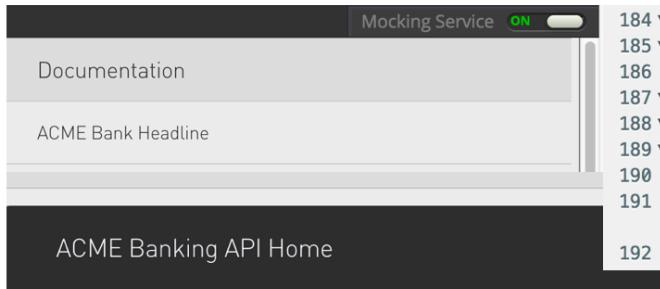
application/json

Examples: [Example](#)

```
{  
  "transactionID": "b05f550d-1915-4def",  
  "fromAccount": {  
    "accountID": "12345",  
    "accountType": "Savings",  
    "accountNumber": "1234567890",  
    "accountOwner": [  
      {  
        "customerID": "8f19cb50-3f57-4d38",  
        "displayName": "John Doe",  
        "ssn": "123-456-7890"  
      }  
    ],  
    "accountBalance": {  
      "currency": "USD",  
      "amount": "8457.90"  
    },  
    "IBAN": "GB29NWBK60161331926820",  
    "bank": {  
      "name": "Westpac",  
      "branch": "Sydney",  
      "address": "123 Main Street, Sydney, NSW 2000",  
      "zip": "2000",  
      "city": "Sydney",  
      "state": "NSW",  
      "country": "Australia",  
      "phone": "+61 2 1234 5678",  
      "fax": "+61 2 1234 5679",  
      "email": "info@westpac.com.au",  
      "url": "http://www.westpac.com.au"  
    }  
  }  
}
```

26. Scroll back up and click CLOSE X.

Module 7: Documenting and Testing APIs



The screenshot shows the API Mocking Service interface. On the left, there's a sidebar with 'Mocking Service' turned 'ON'. Below it are sections for 'Documentation', 'ACME Bank Headline', and 'ACME Banking API Home'. The main area displays RAML code for a 'transactions' resource. The code includes a 'post' method with a description and response status '201'. The response body is described as containing a 'Location' header with a URL for the new bank account transaction information.

```
Mocking Service ON
Documentation
ACME Bank Headline
ACME Banking API Home
184 ▼ /transactions:
185 ▼ 184 ▼ post:
186 185 ▼ description: Create a new transaction
187 ▼ 186 ▼ responses:
188 ▼ 187 ▼ 186 ▼ 201:
189 ▼ 188 ▼ 187 ▼ 186 ▼ headers:
190 189 ▼ 188 ▼ 187 ▼ 186 ▼ Location:
191 190 ▼ 189 ▼ 188 ▼ 187 ▼ 186 ▼ description: URL of the new bank account transaction|
192 191 ▼ 189 ▼ 188 ▼ 187 ▼ 186 ▼ information
body:
```

ACME Banking API enables developers to build applications that make use of the information from resource methods implemented in the API.

This API contains functionality that allows developers to retrieve and manipulate *customer*, *account* and *transaction* information. Check out the [API Portal](#) for more details.

Objectives:

- Add documentation and description nodes to a RAML definitions.
- Use the mocking service to create API endpoints.
- Use the API Console to test API endpoints.

Walkthrough 7-1: Add a documentation fragment to a RAML API definition

In this walkthrough, you add documentation for the ACME Banking API RAML definition. You will:

- Create a documentation fragment.
- Link the documentation fragment to the main RAML definition file.
- Link multiple documentation fragments to the documentation node in the RAML API definition.

The screenshot shows the MuleSoft API Designer interface. The left sidebar shows a project structure with a 'documentation' folder containing 'acmeBankDoc.raml'. The main editor window displays RAML code. A context menu is open over the code at line 3, with options like 'Documentation' and 'ACME Banking API Home'. The right panel shows a preview of the generated documentation pages.

```
#%RAML 1.0 DocumentationItem
title: ACME Banking API Home
content: |
    **ACME Banking API** enables developers to build applications that make use of the information from resource methods implemented in the API.

This API contains functionality that allows developers to retrieve and manipulate customer, account and transaction information. Check out the API Portal for more details.
```

Documentation

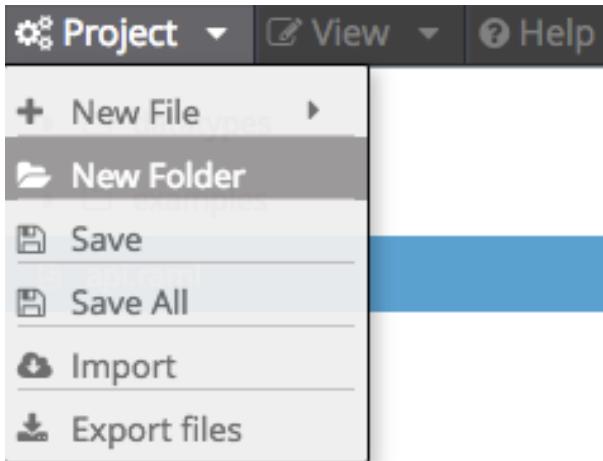
ACME Banking API Home

ACME Banking API enables developers to build applications that make use of the information from resource methods implemented in the API.

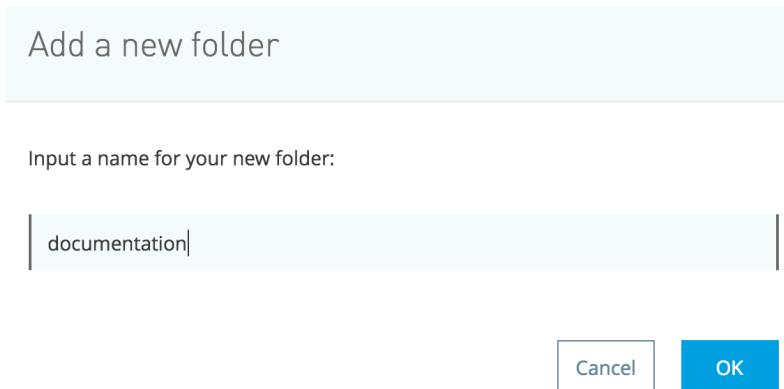
This API contains functionality that allows developers to retrieve and manipulate customer, account and transaction information. Check out the [API Portal](#) for more details.

Create a documentation fragment and link it to the main RAML definition file

1. Return to API Designer.
2. Click the Project menu and select New Folder.

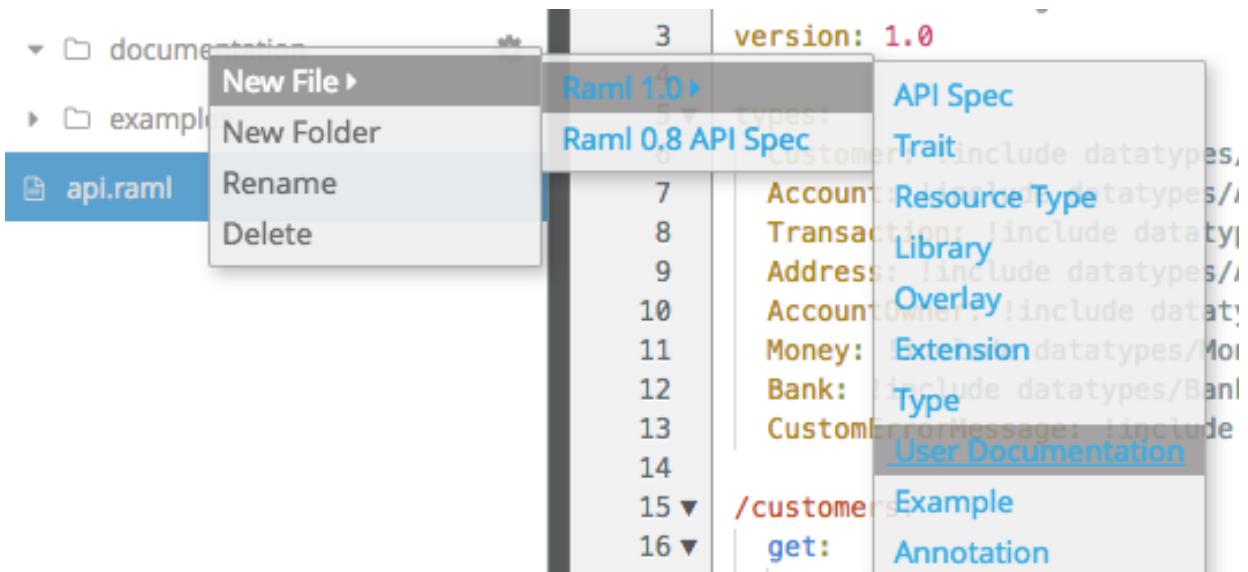


3. In the Add a new folder dialog box, set the folder name to documentation.

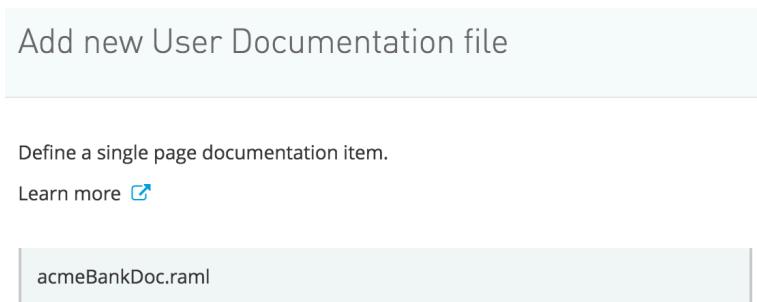


4. Click OK.

5. In the file browser section, right-click the documentation folder and select New File > RAML 1.0 > User Documentation.



6. In the Add a new User Documentation dialog box, set the file name to acmeBankDoc.raml.



7. Click OK.
8. In the editor, set the title property to ACME Banking API Home.

Note: Be sure to put a space after the colon.

```

1  %%RAML 1.0 DocumentationItem
2  title: ACME Banking API Home
3  content:

```

9. After the colon for the content property, press the spacebar and add a | (the pipe operator).
10. Return to the course snippets.txt file.
11. Locate Module 7 and copy the text under Snippet #1.

```

-----Module 7-----
Documentation for acmeBankDoc.raml file -

Snippet #1
**ACME Banking API** enables developers to build applications that make use of the information
from resource methods implemented in the API.

Snippet #2
This API contains functionality that allows developers to retrieve and manipulate _customer_,
_account_ and _transaction_ information. Check out the [API Portal]() for more details.

Documentation for acmeBankHeadline.raml file -

```

12. Return to API Designer.
13. In acmeBankDoc.raml, on a new line after the content property, paste the text.

```

API administration > ACME Banking API - 1.0 > Designer
Project View Help * /documentation/acmeBankDoc.raml

1  %%RAML 1.0 DocumentationItem
2  title: ACME Banking API Home
3  content: |
4  | **ACME Banking API** enables developers to build applications
5  | that make use of the information from resource methods
| implemented in the API.

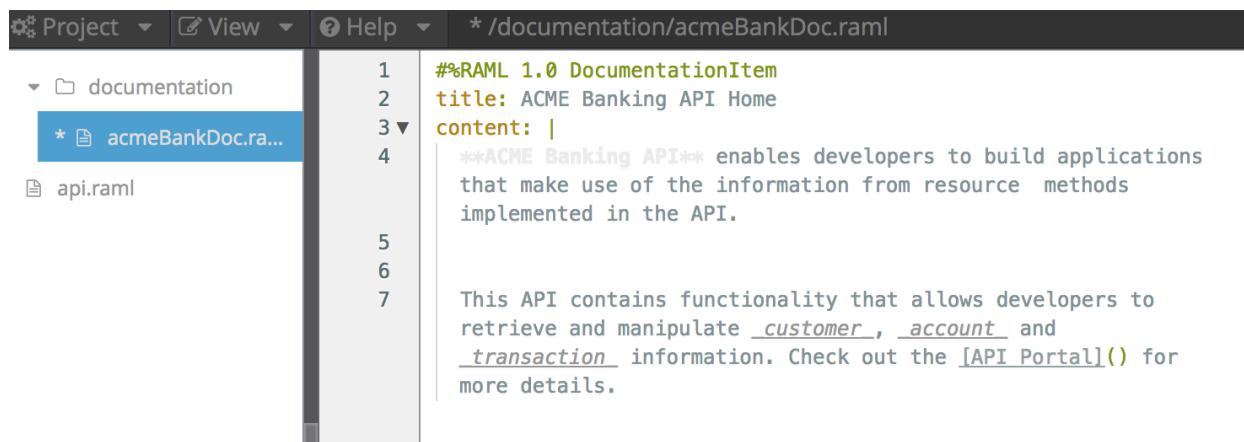
```

14. Press enter two times to add new empty lines.

15. Return to the course snippets.txt file and copy Snippet #2.

```
-----Module 7-----  
Documentation for acmeBankDoc.raml file -  
  
Snippet #1  
**ACME Banking API** enables developers to build applications that make use of the information from resource methods implemented in the API.  
  
Snippet #2  
This API contains functionality that allows developers to retrieve and manipulate customer, account and transaction information. Check out the [API Portal]() for more details.  
  
Documentation for acmeBankHeadline.raml file -
```

16. Return to acmeBankDoc.raml in API Designer and paste the text.



```
#%RAML 1.0 DocumentationItem  
title: ACME Banking API Home  
content:  
  **ACME Banking API** enables developers to build applications  
  that make use of the information from resource methods  
  implemented in the API.  
  
  This API contains functionality that allows developers to  
  retrieve and manipulate customer, account and  
  transaction information. Check out the [API Portal]() for  
  more details.
```

17. Save the project.

18. In the API Designer file browser, click api.raml.

19. In api.raml, go to the line of code before the types node and press enter to add a new line.

20. In the shelf, click documentation.

21. In the new line created below the documentation node, type the following:

```
- !include
```

Note: Be sure to put a space between the - and the !.

22. Stay on the same line and click documentation in the shelf.

23. After the word documentation word, type /.

```
5  documentation:  
6    - !include documentation/  
7  types:  
8    Customer: !include datatypes/Customer.raml  
9    Account: !include datatypes/Account.raml
```

24. In the shelf, click acmeBankDoc.raml.

25. Add a new line.

```
5 ▼ documentation:  
6   - !include documentation/acmeBankDoc.raml  
7  
8 ▼ types:  
9   Customer: !include datatypes/Customer.raml  
10  Account: !include datatypes/Account.raml
```

View the API documentation

26. In the API Console, locate the new Documentation section and click ACME Banking API Home.

The screenshot shows the API Console interface. On the left, there's a sidebar with sections like 'Documentation', 'ACME Banking API Home', 'Types', and 'Resources'. The 'ACME Banking API Home' section is currently selected, indicated by a darker background. On the right, there's a main content area with a heading 'API is behind a firewall (?)' and a checkbox. Below that is a 'Collapse All' button.

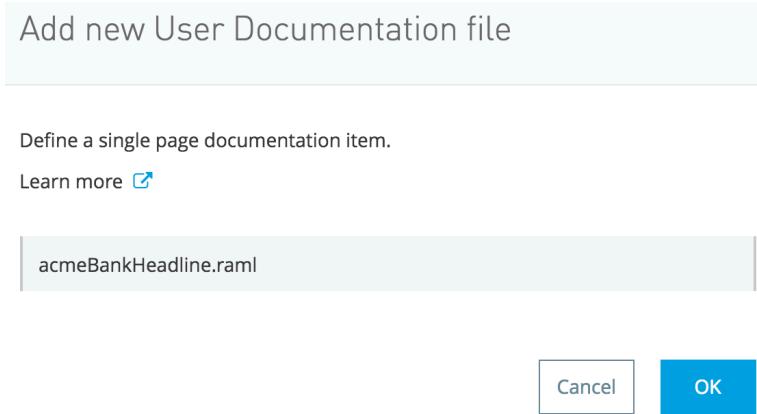
27. Browse the documentation content for the ACME Banking API Home in the console.

The screenshot shows the 'ACME Banking API Home' documentation page. At the top, there's a header with a 'CLOSE X' button. The main content area contains text about the API's purpose and functionality, mentioning 'customer', 'account', and 'transaction' information, and a link to the 'API Portal'.

28. Click CLOSE X in the ACME Banking API Home header.

Link multiple documentation fragments to the documentation node

29. In the API Designer file browser, right-click the documentation folder and select New File > RAML 1.0 > User Documentation.
30. In the Add new User Documentation file dialog box, set the file name to acmeBankHeadline.raml.



31. Click OK.
32. In the editor, set the type property to ACME Bank Headline.
33. In the line that contains the content property, press spacebar after the colon and type | (pipe operator) and press enter.

The screenshot shows the API Designer file browser. On the left, there is a tree view with nodes: datatypes, documentation (which is expanded), and acmeBankDoc.raml. Below the tree, a list of files is shown, with 'acmeBankHeadline.raml' highlighted by a blue bar and marked with an asterisk (*).

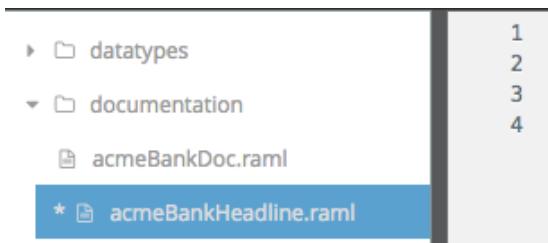
1	#%RAML 1.0 DocumentationItem
2	title: ACME Bank Headline
3	content:

34. Return to the course snippets.txt file and copy Snippet #3.

```
Documentation for acmeBankHeadline.raml file -  
Snippet #3  
**ACME Bank** is a _multinational_ banking and financial services organization.
```

35. Return to acmeBankHeadline.raml and paste the text on the new line.
36. Save the project.

37. In the API Designer file browser, click api.raml to open the file in the editor.



```
1 %%RAML 1.0 DocumentationItem
2 title: ACME Bank Headline
3 content: |
4 | **ACME Bank** is a multinational banking and financial
| services organization.|
```

* acmeBankHeadline.raml

38. At the end of the line that defines the documentation node, press enter.

39. Add code to include the acmeBankHeadline.raml file:

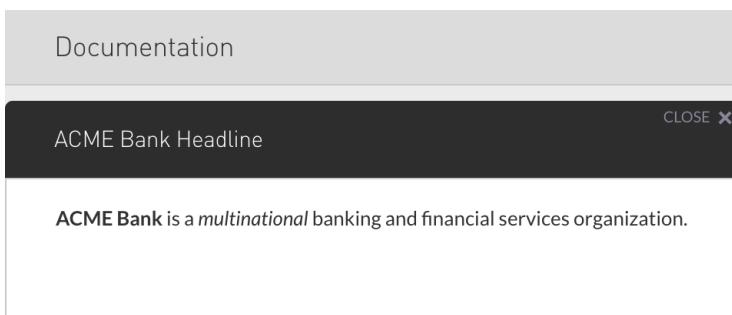
```
- !include documentation/acmeBankHeadline.raml
5 ▼ documentation:
6   - !include documentation/acmeBankHeadline.raml
7   - !include documentation/acmeBankDoc.raml
8
9 ▼ types:
```

40. Save the project.

View the API documentation

41. In the Documentation section of API Console, click ACME Bank Headline.

42. View the documentation content in the Console window.



43. Click CLOSE X in the ACME Bank Headline header.

Walkthrough 7-2: Add description nodes to a RAML API definition

In this walkthrough, you add documentation for the ACME Banking API RAML definition. You will:

- Add descriptions to resource methods.
- Add descriptions to URI parameters.
- Add description and example nodes to request and response headers.

The screenshot shows the API Designer interface with a RAML code editor and a detailed description panel. The code editor highlights a 'description' node under a 'post' method. The description text is: 'Create a new transaction'. To the right, a 'Response' panel is open, showing a 201 status code. The detailed description for 201 includes 'Headers' (Location), 'Body' (application/json object), and examples for 'Location' and 'Body'. A red box highlights the 'description' node in the code editor.

```
184 ▼ /transactions:  
185 ▼   post:  
186     description: Create a new transaction  
187     responses:  
188       201:  
189         headers:  
190           Location:  
191             description: URL of the new bank account information  
192             body:
```

Response

STATUS 201

Headers

Location required string

201

503

URL of the new bank account transaction information

Body application/json

application/json object

Add a description to a get method

1. Return to api.raml in API Designer.
2. In the /customers resource, create a new line after the line that contains the get method.
3. In the shelf, click description.

The screenshot shows the API Designer interface with a RAML code editor. A 'description' node is added to the 'get' method of the '/customers' resource. The description text is: 'Accept?'. The code editor shows the following snippet:

```
7 ▼ /customers:  
8 ▼   get:  
9  
10     description:  
11     Accept?:  
12     responses:  
13       200:  
14         headers:  
15           Cache-Control:  
16           Expires:  
17             type: datetime-local  
18         body:  
19           application/json:
```

ROOT (1) DOCS (2)

protocols description displayName

- Set the description to Retrieve a list of customers.

```
7 ▼ | /customers:  
8 ▼ |   get:  
9 |     description: Retrieve a list of customers  
10 |    headers:
```

Add descriptions to other methods

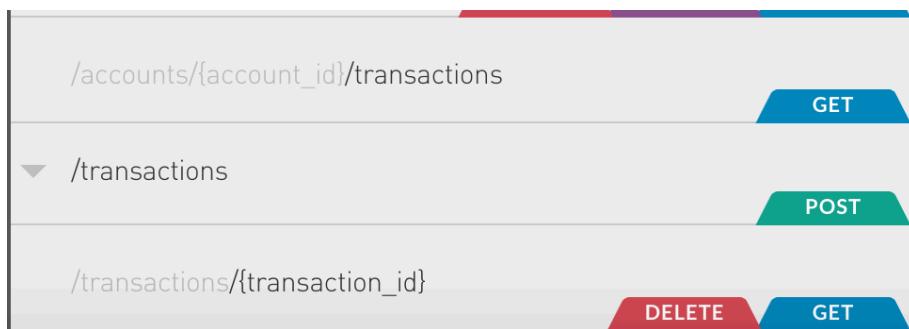
- Add some or all the following descriptions to the specified methods for the following resources:

• /customers	get	Add a new customer
• /customers/{customer_id}	get	Retrieve a customer with a specific customer ID
• /customers/{customer_id}	patch	Update a specific customer info
• /customers/{customer_id}	delete	Delete a specific customer
• /accounts	get	Retrieve a list of accounts for a specific customer
• /accounts	post	Create a new bank account
• /accounts/{account_id}	get	Retrieve an account info with a specific account ID
• /accounts/{account_id}	delete	Delete an account with a specific account ID
• /accounts/{account_id}	put	Update a specific account info
• /transactions	get	Retrieve a list of transactions for a specific account
• /transactions	post	Create a new transaction
• /transactions/{transaction_id}	get	Retrieve a transaction with a specific transaction ID

- Save the project.

View the API documentation

- In the API Console, click GET for the /transactions/{transaction_id} resource.



8. View the Description in the Request section of the GET method.

The screenshot shows a REST API interface. At the top, there is a URL bar with the path `/transactions/{transaction_id}`. Below the URL, there are two buttons: **DELETE** and **GET**. A red banner across the buttons displays the message **Try-it is disabled because baseUri is not present**. Below the buttons, the word **Request** is displayed. Under the **DESCRIPTION** section, the text **Retrieve a transaction with a specific transaction ID** is shown. In the **URI PARAMETERS** section, there is a single parameter `transaction_id` with the type *required string*. Under the **SECURITY SCHEMES** section, the option **Anonymous** is selected. At the top right of the interface, there is a **CLOSE X** button.

9. Click CLOSE X.

Add description nodes to URI parameters

10. In the /customers resource, locate the customer_id URI parameter.
11. Add a description to it and set it to ID of the customer.

```
48 ▼ |  /{customer_id}:  
49 ▼ |    uriParameters:  
50 |      customer_id:  
51 |        description: ID of the customer|  
52 ▼ |    get:
```

12. Similarly, add a description for the account_id URI parameter equal to ID of the account.
13. Add a description for the transaction_id URI parameter equal to ID of the transaction.

Add description and example nodes to a method's request and response headers

14. In the /customers resource, locate the optional Accept header.

15. Add a description to it and set it to Specify the media type of the response.

```
8 ▼ | /customers:  
9 ▼ |   get:  
10 |     description: Retrieve a list of customers  
11 ▼ |     headers:  
12 ▼ |       Accept?:  
13 |         description: Specify the media type of the response  
14 |       |  
15 - |
```

16. Go to a new line and click example in the shelf.

17. Set the example to application/xml.

```
8 ▼ | /customers:  
9 ▼ |   get:  
10 |     description: Retrieve a list of customers  
11 ▼ |     headers:  
12 ▼ |       Accept?:  
13 |         description: Specify the media type of the response  
14 |         example: application/xml  
15 ▼ |       responses:
```

18. Similarly, add the description and example nodes to the Accept headers for the following resource methods:

- /customers/{customer_id} get method
- /customers/{customer_id}/accounts get method
- /accounts/{account_id} get method
- /accounts/{account_id}/transactions get method
- /transactions/{transaction_id} get method

19. Similarly, add a description node to the Cache-Control header of the response.

20. On the description node line, type | and press enter.

21. Return to the course snippets.txt file.

22. Locate the Description for Cache-Control header and copy the lines of text.

```
Documentation for acmeBankDoc.raml file -  
  
Snippet #1  
**ACME Banking API** enables developers to build applications that make use of the information from resource methods implemented in the API.  
  
Snippet #2  
This API contains functionality that allows developers to retrieve and manipulate _customer_, _account and _transaction_ information. Check out the [API Portal]() for more details.  
  
Documentation for acmeBankHeadline.raml file -  
  
Snippet #3  
**ACME Bank** is a _multinational_ banking and financial services organization.  
  
Description for Cache-Control header:  
  
Activates caching and defines cache behavior through cache response directives.  
Usually defines public or private (cacheable by proxy or not) and max-age for resource.  
See http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html for more information.  
  
Description for Expires header:  
  
Sets a date in RFC 1123 format from which the cached resource should no longer be considered valid.  
If both the Expires header and max-age in the Cache-Control header are set, max-age will take precedence.
```

23. Return to api.raml.

24. Paste the text in the new empty line.

```
15 ▼ | responses:  
16 ▼ | | 200:  
17 ▼ | | | headers:  
18 ▼ | | | | Cache-Control:  
19 | | | | | description: |  
② 20 | | | | | Activates caching and defines cache behavior through  
| | | | | cache response directives.  
21 | | | | Usually defines public or private (cacheable by proxy or not) and  
| | | | max-age for resource.  
22 ▼ | | | | See http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html for more  
| | | | information.
```

25. Go to the line below the line number that contains the red cross error.

26. Select the lines to before the Expires header and press tab 7 times to indent it in line with the description node.

```
18 ▼ | | | | Cache-Control:  
19 | | | | | description: |  
② 20 | | | | | Activates caching and defines cache behavior through  
| | | | | cache response directives.  
21 | | | | Usually defines public or private (cacheable by proxy or not) and max-  
| | | | age for resource.  
22 ▼ | | | | See http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html for more  
| | | | information.  
② 23 ▼ | | | | | Expires:
```

27. Go to the end of the last line of the description node and press enter.
28. Press backspace or delete to go back one tab space.
29. In the shelf, click example.

```

18 ▼ | Cache-Control:
19 ▼ |   description: |
20 |     Activates caching and defines cache behavior through
|     cache response directives.
21 |     Usually defines public or private (cacheable by proxy or
|     not) and max-age for resource.
22 |     See http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html for more information.
23 |
24 |   Expires:
25 |     type: datetime

```

30. Set the example node to private, max-age=31536000.

```

18 ▼ | Cache-Control:
19 ▼ |   description: |
20 |     Activates caching and defines cache behavior through
|     cache response directives.
21 |     Usually defines public or private (cacheable by proxy or
|     not) and max-age for resource.
22 |     See http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html for more information.
23 |     example: private, max-age=31536000
24 |   Expires:
25 |     type: datetime

```

31. Similarly, set the following documentation nodes for the Expires header:

- description: The text for Description for Expires header text in the course snippets.txt file
- example: Tue, 18 Apr 2017 09:30:41 GMT
- format: rfc2616

Note: The default datetime format follows rfc3339 specification.

```

24 ▼ |   Expires:
25 ▼ |     description: |
26 |       Sets a date in RFC 1123 format from which the cached
|       resource should no longer be considered valid.
27 |       If both the Expires header and max-age in the Cache-
|       Control header are set, max-age will take precedence.
28 |     type: datetime
29 |     example: Tue, 18 Apr 2017 09:30:41 GMT
30 |     format: rfc2616

```

Add documentation to other request and response headers

32. Add some or all of the following descriptions to the Location headers for the specified resource methods:

- /customers post URL of the new customer information
- /accounts post URL of the new bank account information
- /transactions post URL of the new bank account transaction information

```
184 ▼ | /transactions:  
185 ▼ |   post:  
186 |     description: Create a new transaction  
187 ▼ |     responses:  
188 ▼ |       201:  
189 ▼ |         headers:  
190 |           Location:  
191 |             description: URL of the new bank account transaction|  
192 |               information  
    body:
```

33. Scroll up to the get method of the /customers resource and copy the Accept header's description and example lines.

```
8 ▼ | /customers:  
9 ▼ |   get:  
10 |     description: Retrieve a list of customers  
11 ▼ |     headers:  
12 ▼ |       Accept?:  
13 |         description: Specify the media type of the response  
14 |           example: application/xml  
15 ▼ |     responses:  
16 ▼ |       200:
```

34. Scroll down to the get method of the /accounts nested resource add the description and example to the optional Accept header.

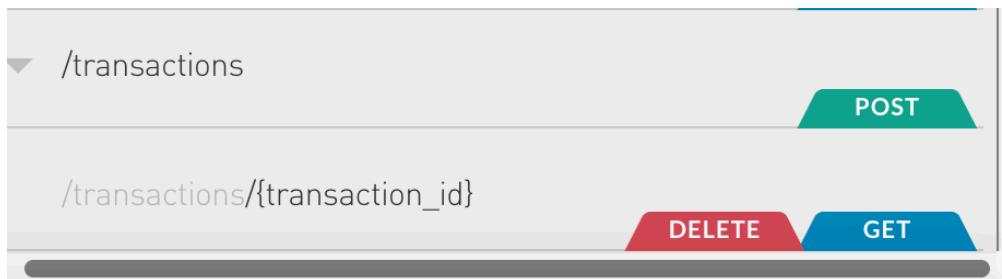
```
94 ▼ | /accounts:  
95 ▼ |   get:  
96 |     description: Retrieve a list of accounts for a specific  
97 ▼ |     customer  
98 |     headers:  
99 |       Accept?:  
100 |         description: Specify the media type of the response  
101 ▼ |           example: application/xml  
      responses:
```

35. Indent the example node.

36. Save the project.

View the API documentation

37. In the API Console, click POST for the /transactions resource.



38. Locate the Location header description in the 201 Response section.

Response

201	STATUS 201
503	Headers Location <i>required string</i> URL of the new bank account transaction information
	Body <i>application/json</i> <i>application/json object</i>

39. In the /transactions resource banner, click CLOSE X.

Walkthrough 7-3: Use the mocking service in API Console to test an API

In this walkthrough, you will test your API by enabling mocking service. You will:

- Enable the mocking service in the API Console.
- Test a few methods in the API Console to see the response information.

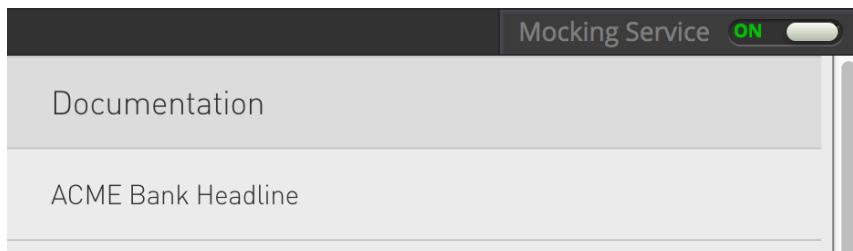
The screenshot shows the API Console interface. At the top, there is a dark header bar with the text "Mocking Service" followed by a toggle switch that is currently set to "ON". Below this, the main content area has two sections: "Documentation" and "ACME Bank Headline". To the right of these sections is a vertical sidebar containing a "Response" section. This section includes a "Status" field showing "200", a "Headers" section listing "Content-Type: application/xml; charset=utf-8", "Date: Tue, 28 Mar 2017 20:29:36 GMT", "Server: nginx", and "Vary: Accept-Encoding"; and a "Body" section containing the text "1 <response>".

Enable the mocking service in the API Console

1. Return to API Designer.
2. In the top-right corner of the API Console, click the Mocking Service toggle button.

This screenshot shows the API Console with the "Mocking Service" toggle switch set to "OFF". The main content area contains three sections: "Documentation", "ACME Bank Headline", and "ACME Banking API Home". At the bottom of the screen, there is a "Resources" section which includes a note "API is behind a firewall (?) " and a "Collapse All" button.

- Verify that the toggle button now contains green ON text.



- Look at api.raml; you should now see a new baseUrl parameter was added above the title node.

```
1  #%RAML 1.0
2  baseUrl: https://mocksvc.mulesoft.com/mocks/2dd91fa5-7a3b-49a7-
3    8926-4adb73dcbe3c
4  title: ACME Banking API
5  version: 1.0
6 ▼ documentation:
```

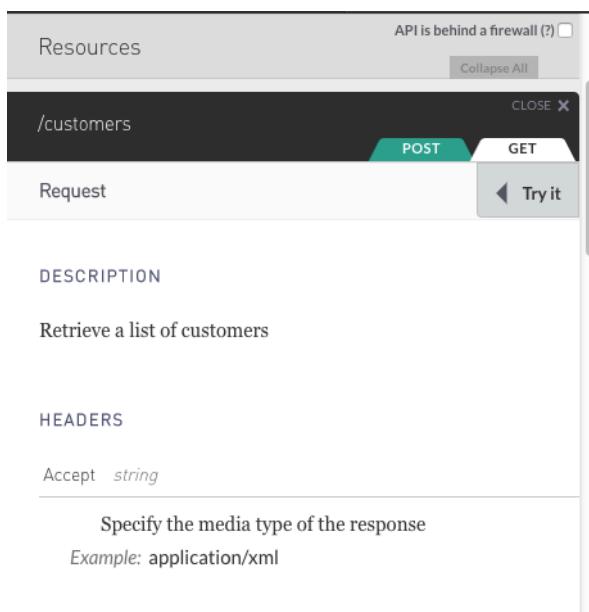
- Save the project.

Test a method in the API Console to see the response information

- In the API Console, click GET the /customers resource.



- Click Try It.



8. In the Try It section, scroll down to locate the GET button and click it.

The screenshot shows the 'Try It' interface for a '/customers' endpoint. The URL is displayed as '/customers'. Below the URL, there are two buttons: 'POST' (highlighted in green) and 'GET'. A 'Try it' button is located above the configuration area. The configuration area is divided into sections: 'AUTHENTICATION', 'HEADERS', and 'QUERY PARAMETERS'. Under 'AUTHENTICATION', it says 'Security Scheme: Anonymous'. Under 'HEADERS', there is a dropdown for 'Accept' with 'application/xml' selected. Under 'QUERY PARAMETERS', there is a plus sign icon. At the bottom of the interface are three buttons: 'GET' (highlighted in blue), 'Clear', and 'Reset'.

9. Verify that you can see the Request information (Request URL and the Accept header information).

The screenshot shows the 'Request' section. It has a header 'Request ▲'. Below it, under 'Request URL', is the URL: 'https://anypoint.mulesoft.com/apiplatform/proxy/https://mocksvc.mulesoft.com/mocks/2dd91fa5-7a3b-49a7-8926-4adb73dcbe3c/customers'. Under 'Headers', there is a table with one row: 'Accept: application/xml'.

Header	Value
Accept	application/xml

10. Review the Response status and information displayed right below the request information.

Response ▲

Status
200

Headers

O:
 private, max-age=31536000

1:
 Tue, 18 Apr 2017 09:30:41 GMT

access-control-allow-origin:
 *

connection:
 keep-alive

content-length:
 99

content-type:
 application/xml; charset=utf-8

date:
 Tue, 28 Mar 2017 20:29:36 GMT

server:
 nginx

vary:
 Accept-Encoding

Body

```
1 <response>
2   <message>RAML had no response information for app
3 </response>
```

11. Scroll up and click CLOSE X.

Test another method in the API Console to see the response information

12. Click the GET tab for the /accounts/{account_id}/transactions resource.
13. Click Try It.

14. In the Try It section, scroll down to locate the GET button and click it.

URI PARAMETERS

GET /accounts/{account_id}/transactions

account_id * string

HEADERS

QUERY PARAMETERS

GET **Clear** **Reset**

15. Verify that the account_id URI parameter has a red banner denoting it is required before calling the GET method.

URI PARAMETERS

GET /accounts/{account_id}/transactions

account_id * string

Required

HEADERS

QUERY PARAMETERS

Force GET **Clear** **Reset**

16. Enter an account_id of 2 and click the GET button again.

URI PARAMETERS

GET /accounts/2/transactions

account_id * string

2

17. Scroll down and review the Response status and information displayed right below the request information.

Response ▲

Status
200

Headers

```
access-control-allow-origin: *
connection: keep-alive
content-length: 720
content-type: application/xml; charset=utf-8
date: Tue, 02 May 2017 22:05:12 GMT
server: nginx
vary: Accept-Encoding
```

Body

```
1 <response>
2   <accountID>12345</accountID>
3   <accountType>Savings</accountType>
4   <accountNumber>1234567890</accountNumber>
5   <accountOwner>
6     <accountOwner>
7       <customerID>8f19cb50-3f57-4d38,</customerID>
8       <displayName>John Doe,</displayName>
9       <ssn>123-456-7890,</ssn>
10      </accountOwner>
11    </accountOwner>
12    <accountBalance>
13      <currency>USD</currency>
14      <amount>8457.90</amount>
15    </accountBalance>
16    <IBAN>GB29NWBK60161331926820</IBAN>
```

18. Click CLOSE X in the /accounts/{account_id}/transactions banner.

Module 8: Making APIs Discoverable

The screenshot shows the Anypoint API Notebook interface. At the top, there are three buttons: 'Private' (with a lock icon), 'Themes' (with a paint palette icon), and 'Live portal' (with a monitor icon). Below the header, the title 'API Notebook' is displayed, along with a 'Save' button. A message at the top of the notebook area says, 'For more information, check out examples at <http://api-notebook.anypoint.mulesoft.com/>'. The main area contains two code cells. The first cell contains the following code:

```
1 API.createClient('client', 'https://anypoint.mulesoft.com/apiplatform/repository/v2/organizations/fe9154b4-8772-434c-be4-8326cec38d2e/public/apis/12364526/versions/173187/definition');
```

The second cell contains the result of the previous command:

```
$0= "Create a new code cell and type \"client.\"" to explore this API.
```

Below the results, the output is shown as:

```
2 client.customers.get()
$1= ▼Object {"body": Object, "status": 200, "headers": Object}
  ▼body: Object
    message: "RAML had no response information for application/json"
  ▼headers: Object
    content-type: "application/json; charset=utf-8"
  status: 200
```

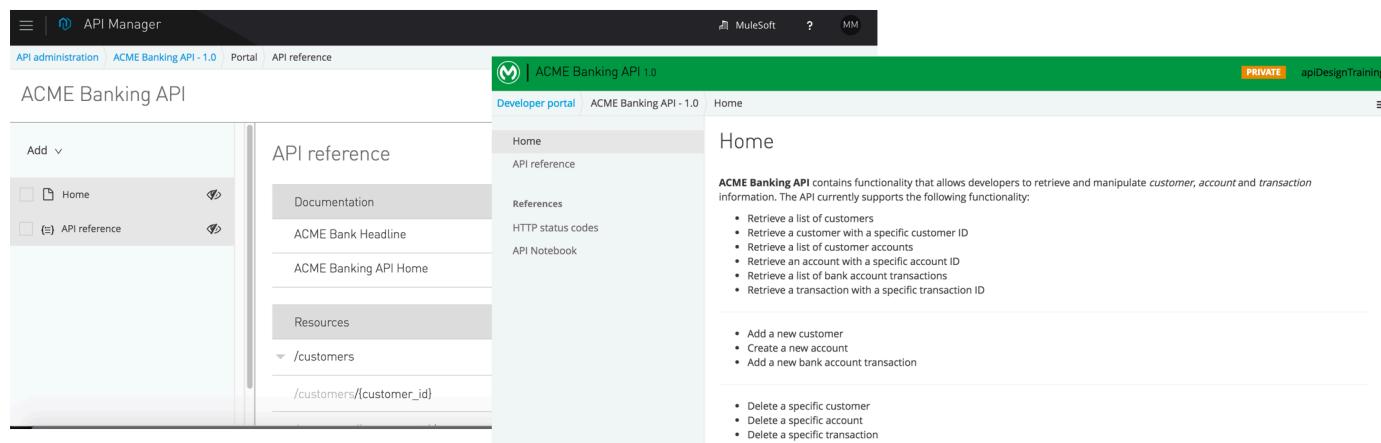
Objectives:

- Create API Portals for learning about and testing APIs.
- Customize API Portals with themes.
- Create a sample use case with API Notebook inside the API Portal.
- Publish API definitions to the Anypoint Exchange for discovery.
- Gather feedback from API consumers.

Walkthrough 8-1: Create and customize an API Portal

In this walkthrough, you will create an API Portal documenting information about the ACME Banking API. You will:

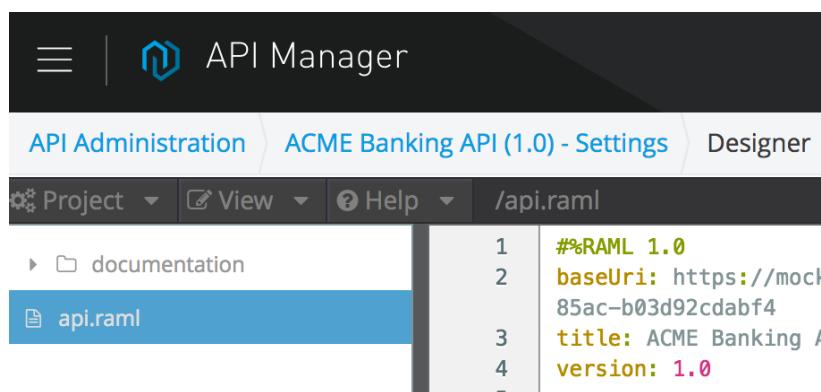
- Create an API Portal.
- Add various types of content to an API Portal.
- Define a color scheme.
- Publish and view the resulting API Portal.



The screenshot shows the MuleSoft API Manager interface. On the left, there's a sidebar with 'Add' and 'API reference' sections. The 'API reference' section contains links to 'Documentation', 'ACME Bank Headline', 'ACME Banking API Home', 'Resources', and a 'customers' resource. The main content area is titled 'ACME Banking API 1.0' and shows the 'Home' page. It includes a 'Developer portal' link, a 'PRIVATE' status indicator, and a 'apiDesignTraining' tag. The 'Home' page content describes the API's functionality for retrieving and manipulating customer, account, and transaction information. It lists several endpoints and their descriptions.

Create an API Portal

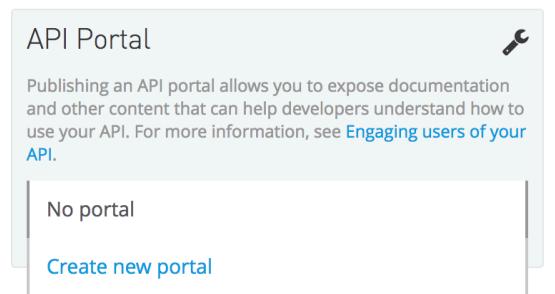
1. Return to API Designer.
2. Click the ACME Banking API (1.0) - Settings link above the editor.



The screenshot shows the 'ACME Banking API (1.0) - Settings' page in the API Manager. The top navigation bar includes 'API Administration', 'ACME Banking API (1.0) - Settings', and 'Designer'. Below the navigation is a toolbar with 'Project', 'View', 'Help', and the file path '/api.raml'. The main area displays the RAML code for the 'api.raml' file. The code defines the API version, base URI, title, and version.

```
#%RAML 1.0
baseUri: https://mock
85ac-b03d92cdabf4
title: ACME Banking A
version: 1.0
```

- In the API Portal section on the API version details page, click the drop-down arrow and select Create new portal from the drop-down menu; the API Portal Designer should open.



- In the API Portal Designer, locate the two pages that were automatically created: Home and API reference.
- In the left-side navigation, click the Home page link.
- Examine the Home page editor.

Note: The Home page is empty and has icons in the editor to add text that supports markdown, include images and files.

- In the left-side navigation, click the API reference link.
- Examine the API reference page.

Note: The API reference page is only populated if the API definition has a base URI specified.

The screenshot shows the MuleSoft API Manager interface. The top navigation bar includes "API Manager", "MuleSoft", and "MM". The sub-navigation bar shows "API administration", "ACME Banking API - 1.0", "Portal", and "API reference".

The main content area displays the "ACME Banking API". On the left, there's a sidebar with an "Add" dropdown and two items: "Home" and "API reference".

The right side shows the "API reference" page. It features a "Documentation" section with "ACME Bank Headline" and a "ACME Banking API Home" link. Below that is a "Resources" section for the "/customers" endpoint. It lists three methods: "POST", "DELETE", "PATCH", and "GET". The "POST" and "GET" buttons are highlighted in green, while "DELETE" and "PATCH" are in red. A note "API is behind a firewall (?)" with a checkbox is visible.

Use Markdown to add content to a page

9. In the left-side navigation, click the Home page link.
10. In the Home page editor, click the letter A to begin adding content.



11. Return to the course snippets.txt file.
12. Locate the Module 8 section and copy the ACME Banking API Home content.

-----Module 8-----

ACME Banking API Home content -

ACME Banking API contains functionality that allows developers to retrieve and manipulate _customer_, _account_ and _transaction_ information. The API currently supports the following functionality:

- Retrieve a list of customers
- Retrieve a customer with a specific customer ID
- Retrieve a list of customer accounts
- Retrieve an account with a specific account ID
- Retrieve a list of bank account transactions
- Retrieve a transaction with a specific transaction ID

- Add a new customer
- Create a new account
- Add a new bank account transaction

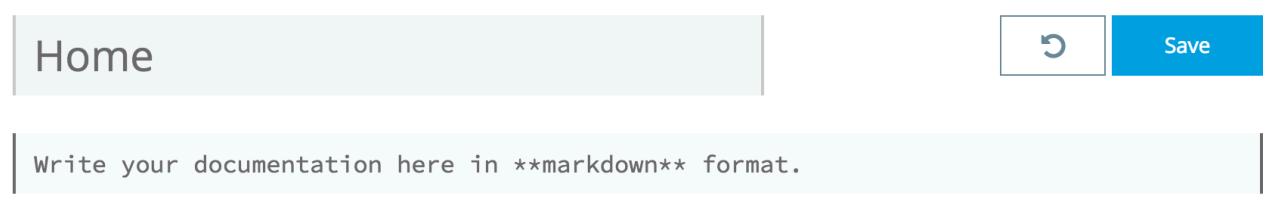
- Delete a specific customer
- Delete a specific account
- Delete a specific transaction

- Update a specific customer profile information
- Update a specific bank account information

Check out the **API Reference** page to learn more about the resources and methods.

HTTP status codes URL

13. Return to the API Portal Designer and click the box with the text Write your documentation here in **markdown** format.



14. Paste the copied lines here.

Home

Save

ACME Banking API contains functionality that allows developers to retrieve and manipulate _customer_, _account_ and _transaction_ information. The API currently supports the following functionality:

- Retrieve a list of customers
- Retrieve a customer with a specific customer ID
- Retrieve a list of customer accounts
- Retrieve an account with a specific account ID
- Retrieve a list of bank account transactions
- Retrieve a transaction with a specific transaction ID

- Add a new customer
- Create a new account
- Add a new bank account transaction

- Delete a specific customer

15. Click the Save button.

Add external links

16. Click the Add button located above the left-side navigation and in the drop-down menu, select External link.

API administration ACME Banking API - 1.0 F

ACME Banking API

Add   

-  Page
-  API Notebook
-  API Reference
-  Header
-  External link

17. Click the title External Link in the editor and type HTTP status codes.

HTTP status codes

http://example.com

Save

18. Change the URL from http://example.com to
<http://www.restapitutorial.com/httpstatuscodes.html>.

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

HTTP status codes

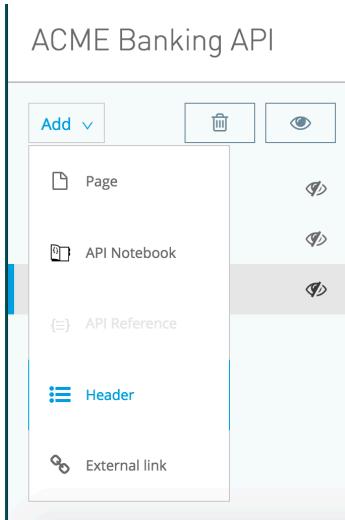
http://www.restapitutorial.com/httpstatuscodes.html

Save

19. Click the Save button.

Add a header

20. Click the Add button again and select Header in the drop-down menu.



21. Click the title Header in the editor and replace the text with References.

References

Save

22. Click the Save button.
23. In the left-side navigation, click and drag the References header and drop it above the HTTP status codes external link.

The screenshot shows the 'ACME Banking API' administration interface. The top navigation bar includes 'API administration', 'ACME Banking API - 1.0', 'Portal', and 'Header'. On the right, there are 'Private', 'Themes', and 'Live portal' buttons. The main area displays a card titled 'References' with the following items:

- Home
- (≡) API reference
- References
- HTTP status codes

At the bottom right of the card are 'Cancel' and 'Save' buttons.

Publish the API Portal

24. In the left-side navigation, select the checkboxes for all the components.

The screenshot shows the same interface as before, but now all components in the left navigation menu have their checkboxes selected:

- Home
- (≡) API reference
- References
- HTTP status codes

25. Click the eye button to publish the selected pages.

Note: Anypoint Platform will not publish a page with zero content. Once you click the eye icon, it should turn green for all the selected pages.

The screenshot shows the interface after publishing. The eye icons next to each component are now solid green, indicating they are published:

- Home
- (≡) API reference
- References
- HTTP status codes

26. Click the Live portal button in the upper-right corner of the API Portal Designer.

The screenshot shows the 'ACME Banking API' portal designer interface. At the top, there's a navigation bar with 'API administration', 'ACME Banking API - 1.0', 'Portal', and 'Home'. In the top right corner, there are three buttons: 'Private' (with a lock icon), 'Themes' (with a blue circular icon), and 'Live portal' (with a monitor icon). Below the navigation bar is a toolbar with 'Add', a trash bin icon, an eye icon, and a search bar labeled 'Home'. To the right of the search bar are two more buttons: a refresh icon and a 'Saved' button with a checkmark.

27. Examine the generated API Portal.

The screenshot shows the generated 'ACME Banking API 1.0' developer portal. At the top, it says 'PRIVATE apiDesignTraining v'. On the left is a sidebar with 'Developer portal', 'ACME Banking API - 1.0', 'Home' (which is selected and highlighted in grey), 'API reference', 'References', and 'HTTP status codes'. The main content area has a title 'Home'. It contains a paragraph about the ACME Banking API's functionality and a list of operations:

- Retrieve a list of customers
- Retrieve a customer with a specific customer ID
- Retrieve a list of customer accounts
- Retrieve an account with a specific account ID
- Retrieve a list of bank account transactions
- Retrieve a transaction with a specific transaction ID

Below this is another list:

- Add a new customer
- Create a new account
- Add a new bank account transaction

At the bottom is a third list:

- Delete a specific customer
- Delete a specific account
- Delete a specific transaction

28. Close the tab with containing the Live Portal and return to the tab with the API Portal Designer.

Define a color scheme for the API Portal

29. In the right corner of the API Portal Designer, click Themes.

The screenshot shows the 'ACME Banking API' portal designer interface. At the top, there's a navigation bar with 'API administration', 'ACME Banking API - 1.0', 'Portal', and 'API reference'. In the top right corner, there are three buttons: 'Private' (with a lock icon), 'Themes' (with a blue circular icon), and 'Live portal' (with a monitor icon). The 'Themes' button is highlighted with a blue border.

30. In the API portal theme settings dialog box, go to the Background color option and click the text area that mentions Use Default.

31. Select a color from the drop-down graph or type a color's hex value.

The screenshot shows the 'Logo image' section with a placeholder 'Drop logo file here' and a 'Browse images' button. Below it, there are five color selection fields: 'Logo link' (gray), 'Background color' (blue), 'Text color' (white), 'Button color' (dark blue), and 'Button text color' (white). The 'Text color' field has a color picker open, showing a gradient from black to white with a central gray circle. The 'Text color' label is bolded.

Setting	Current Value
Logo link	Gray
Background color	Blue
Text color	White (with color picker open)
Button color	Dark Blue
Button text color	White

32. Change the text color, button color and button text color using the same approach of click over the Use Default text box and picking a desired color.

The screenshot shows the same portal settings page after changes. The 'Text color' field now contains the hex value '#030202'. The 'Button color' field contains '#33c5cf'. The 'Badge color' field contains '#e68718'. The other settings remain the same: 'Logo link' (gray), 'Background color' (green), 'Button text color' (white), and 'Badge color' (orange).

Setting	Current Value
Logo link	Gray
Background color	Green
Text color	#030202
Button color	#33c5cf
Button text color	White
Badge color	#e68718

33. Scroll down and click the Update button.



34. Click the Live Portal button again; the live portal should open and you should see your changes.

A screenshot of the ACME Banking API 1.0 developer portal. The top navigation bar includes a logo, the title 'ACME Banking API 1.0', a 'PRIVATE' status indicator, and a user dropdown. The left sidebar has links for 'Developer portal', 'ACME Banking API - 1.0', 'Home', 'API reference', 'References', 'HTTP status codes', and 'API Notebook'. The main content area is titled 'Home' and contains a brief description of the API's functionality. It lists several operations under different sections:

- Retrieve a list of customers
- Retrieve a customer with a specific customer ID
- Retrieve a list of customer accounts
- Retrieve an account with a specific account ID
- Retrieve a list of bank account transactions
- Retrieve a transaction with a specific transaction ID

- Add a new customer
- Create a new account
- Add a new bank account transaction

- Delete a specific customer
- Delete a specific account
- Delete a specific transaction

- Update a specific customer profile information

35. Close the Live Portal and return to the API Portal Designer.

Walkthrough 8-2: Create a sample use case with API Notebook inside the API Portal

In this walkthrough, you create an API Notebook inside the API Portal. You will:

- Add an API Notebook in the API Portal Designer.
- Perform string manipulation on the datatypes.
- Create and use JavaScript functions inside API Notebook.
- View the API Notebook in the Live Portal.

The screenshot shows the API Notebook interface. On the left, a sidebar navigation includes Home, API reference, References, HTTP status codes, and API Notebook (which is selected). The main area is titled "API Notebook" and contains a code editor with two snippets. The first snippet is titled "Create ACME Banking API Client to access it's methods and resources" and contains the following code:

```
1 API.createClient('acmeBankclient',
  'https://anypoint.mulesoft.com/apiplatform/repository/v2/organizations/a6fb4480-6e30-4248-ac23-6e7a751b5606/public/apis/11698133/versions/162026/definition');
```

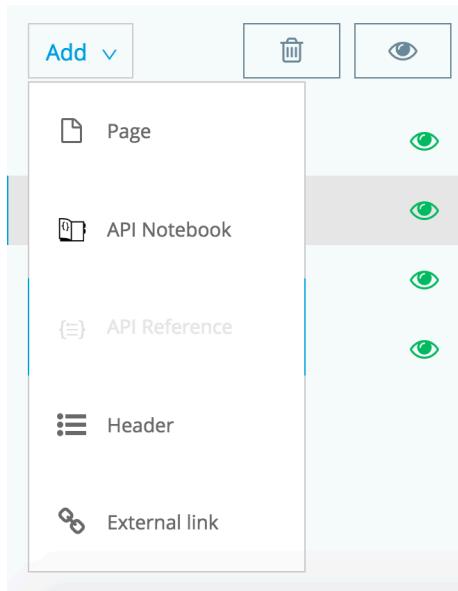
The second snippet is titled "Retrieve a list of customers" and contains the following code:

```
2 acmeBankclient.customers.get();
```

At the bottom, it says "Hosted by API Notebook". There are buttons for "Make your own" and "Play notebook".

Add an API Notebook in the API Portal Designer

1. Return to the API Portal Designer.
2. In the left-side navigation, click Add and select API Notebook in the drop-down menu.



- In the API Notebook editor, change the first parameter in the createClient function from 'client' to 'acmeBankclient'.

API Notebook Save

For more information, check out examples at <http://api-notebook.anypoint.mulesoft.com/>

```
1 API.createClient('acmeBankclient',
  'https://anypoint.mulesoft.com/apiplatform/repository/v2/organizations/a6fb4480-6e30-4248-ac23-
  6e7a751b5606/public/apis/11698133/versions/162026/definition');

createClient(alias, url, options?, cb?)

string
```

Powered by MuleSoft™

- Click the Play notebook button to play the Notebook.
- Verify you get this message below the code cell: Create a new node cell and type \' "acmeBankClient.\'" to explore this API.

API Notebook Save

For more information, check out examples at <http://api-notebook.anypoint.mulesoft.com/>

```
1 API.createClient('acmeBankclient',
  'https://anypoint.mulesoft.com/apiplatform/repository/v2/organizations/a6fb4480-6e30-4248-ac23-
  6e7a751b5606/public/apis/11698133/versions/162026/definition');

$0= "Create a new code cell and type \"acmeBankclient.\" to explore this API."
```

Powered by MuleSoft™

Insert a code cell to explore the ACME Banking API

- Hover over the small circle that is in the middle underneath the createClient code cell.

```
1 API.createClient('acmeBankclient',
  'https://anypoint.mulesoft.com/apiplatform/repository/v2/organizations/a6fb4480-6e30-4248-ac23-
  6e7a751b5606/public/apis/11698133/versions/162026/definition');

$0= "Create a new code cell and type \"acmeBankclient.\" to explore this API."
```

 Insert Text Cell Insert Code Cell Insert API Client

Powered by MuleSoft™

7. Click Insert Code Cell.
8. In the new code cell, type a and press the right arrow key.

```
2 acmeBankclient
  acmeBankclient  function
    addEventListener  function
    alert  function
    applicationCache  object
    async  function
    atob  function
```

Powered by  MuleSoft™

9. Type . after acmeBankclient and select customers in the drop-down menu.

```
2 acmeBankclient.
  customers
    accounts  object
    customers  object
    transactions  object
```

Powered by  MuleSoft™

10. Type . after customers and click on get from the drop-down menu.
11. Add opening and closing parenthesis and a semicolon after the get method; the expression should look like this:

```
2 acmeBankclient.customers.get();
```

12. Press enter to execute the code cell.
13. Verify that you see the status code 200 returned in the result below the code cell.

```
2 acmeBankclient.customers.get();

$1= ▼Object {"body": Object, "status": 200, "headers": Object}
  ►body: Object
  ►headers: Object
  status: 200
```

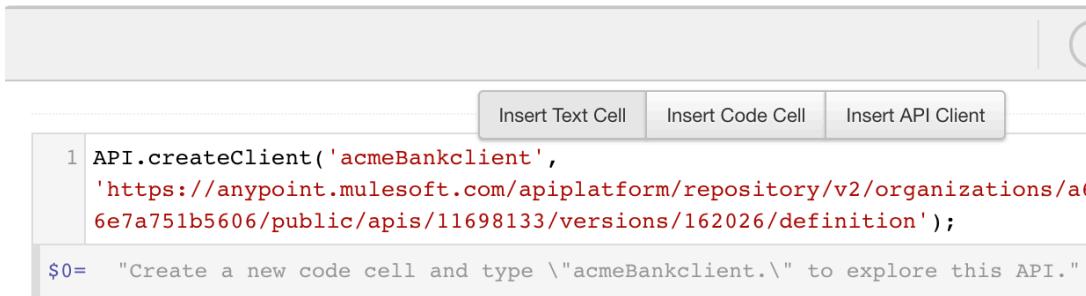
Note: When you expand the body, you can see the customer example returned.

14. Click the menu icon in the right corner of code cell 3 and click Delete from the drop-down menu.

Insert text cells to make the API Notebook readable

15. Locate the circle in the middle above code cell 1 and hover over it.

16. Click Insert Text cell.



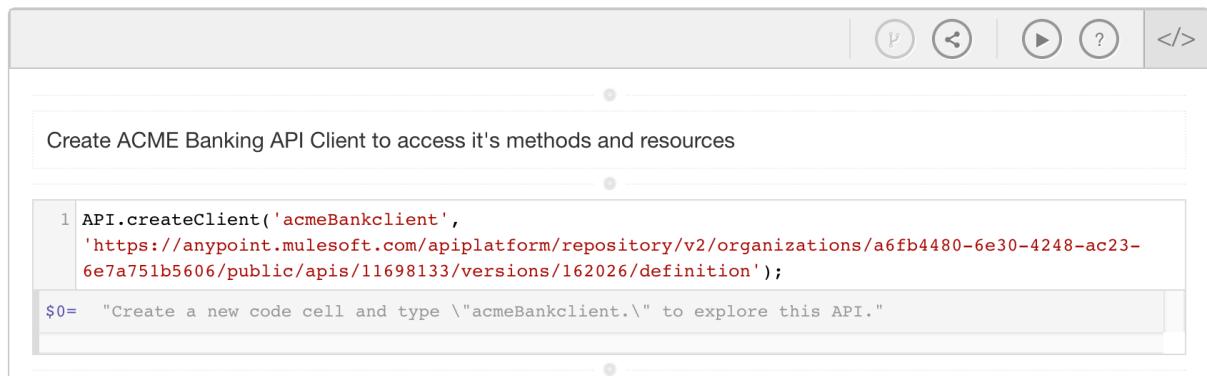
The screenshot shows a code editor interface. At the top, there are three buttons: "Insert Text Cell" (highlighted in grey), "Insert Code Cell", and "Insert API Client". Below these buttons is a code cell containing Java-like pseudocode for creating an API client:

```
1 API.createClient('acmeBankclient',
  'https://anypoint.mulesoft.com/apiplatform/repository/v2/organizations/a6
  6e7a751b5606/public/apis/11698133/versions/162026/definition');
```

Below the code cell is a text cell placeholder with the following content:

```
$0= "Create a new code cell and type \"acmeBankclient.\" to explore this API."
```

17. Click inside the new text cell and type Create ACME Banking API client to access its methods and resources.



The screenshot shows a text cell containing the text: "Create ACME Banking API Client to access it's methods and resources". Below this text cell is a code cell containing Java-like pseudocode for creating an API client:

```
1 API.createClient('acmeBankclient',
  'https://anypoint.mulesoft.com/apiplatform/repository/v2/organizations/a6fb4480-6e30-4248-ac23-
  6e7a751b5606/public/apis/11698133/versions/162026/definition');
```

Below the code cell is a text cell placeholder with the following content:

```
$0= "Create a new code cell and type \"acmeBankclient.\" to explore this API."
```

18. Locate the circle in the middle above code cell 2 and hover over it.

19. Click Insert Text Cell.

20. Click inside the new text cell and type retrieve a list of customers.



The screenshot shows a text cell containing the text: "Retrieve a list of customers". Below this text cell is a code cell containing Java-like pseudocode for retrieving a list of customers:

```
2 acmeBankclient.customers.get();
```

Powered by  MuleSoft™

21. Click Save.

Access the customer datatype example and its attributes and nested datatypes attributes in the API Notebook

22. Type the following inside the text cell: Retrieve a specific customer information.

```
Retrieve a specific customer information
```

23. Return to the course snippet.txt file and copy the JavaScript code snippet for the Code cell value to retrieve a specific customer in Module 8.

24. Return to API Notebook and press enter and in the code cell, paste the Javascript code snippet.

```
Retrieve a specific customer information
```

```
3 acmeBankclient.customers.customer_id("8f19cb50-3f57-4d38").get();
```

25. Press enter inside the code cell to execute the line of code.

26. In the response, click the arrow before the body Object.

```
3 acmeBankclient.customers.customer_id("8f19cb50-3f57-4d38").get();
$2= ▼Object {"body": Object, "status": 200, "headers": Object}
  ▼body: Object
    ► address: Object
      customerID: "8f19cb50-3f57-4d38"
      dateOfBirth: "1983-01-01"
      displayName: "John Doe"
      email: "johndoe@example.com"
      firstName: "John"
      lastName: "Doe"
      phone: "415-000-0000"
      ssn: "123-456-7890"
    ►headers: Object
      status: 200
```

27. Click the arrow before the address object.

```
3 acmeBankclient.customers.customer_id("8f19cb50-3f57-4d38").get();

$2= ▼Object {"body": Object, "status": 200, "headers": Object}
  ▼body: Object
    ▼address: Object
      addressLine1: "1234 Lane"
      addressLine2: "Apt.#620"
      city: "San Francisco"
      country: "United States"
      state: "California"
      zipCode: "94108"
      customerID: "8f19cb50-3f57-4d38"
      dateOfBirth: "1983-01-01"
      displayName: "John Doe"
      email: "john doe@example.com"
      firstName: "John"
      lastName: "Doe"
      phone: "415-000-0000"
      ssn: "123-456-7890"
    ▶headers: Object
    status: 200
```

28. In the same code cell, place the cursor at the beginning of the code cell and type CustomerA=.

```
3 CustomerA=acmeBankclient.customers.customer_id("8f19cb50-3f57-4d38").get();
```

29. Go to the end of the code and press enter to re-execute the line.

30. In the new code cell generated below, type CustomerA.body.displayName; and press enter.

```
4 CustomerA.body.displayName;
```

```
$3= "John Doe"
```

31. In the new code cell generated below, type CustomerA.body.address.city; and press enter.

```
5 CustomerA.body.address.city;
```

```
$4= "San Francisco"
```

Perform string manipulation on the datatypes

32. Hover over the circle in the middle before the new code cell and click Insert New Text Cell.

33. Type Retrieve a specific account information.

34. Return to the course snippets.txt and copy the code from Module 8 with the heading Code cell value to retrieve a specific account information.

35. Return to API Notebook and paste it in the code cell and press enter.

```
6 accountA=acmeBankclient.accounts.account_id("12345").get();
```

36. In the response returned, click the arrow before body to observe the structure of the account information.

37. Expand the accountBalance, accountOwner, and the bank objects to see the underlying values.

```
▼body: Object
  IBAN: "GB29NWBK60161331926820"
  ▼accountBalance: Object
    amount: "8457.90"
    currency: "USD"
    accountID: "12345"
    accountNumber: "1234567890"
  ▼accountOwner: Array[1]
    ▼0: Object
      customerID: "8f19cb50-3f57-4d38,"
      displayName: "John Doe,"
      ssn: "123-456-7890,"
      length: 1
      accountType: "Savings"
    ▼bank: Object
      bankCode: "NWBKGB2L"
      bankName: "ACME Bank"
      routingNumber: "432159876"
      createdAt: "2012-03-07T00:00:00.001Z"
```

38. Hover over the circle between the new code cell and previous code cell and click Insert Text Cell.

39. Type the value of the text cell as Convert the accountBalance amount into a number from string.

```
createdAt: "2012-03-07T00:00:00.001Z"
▼headers: Object
  content-type: "application/json; charset=utf-8"
  status: 200
```

Convert the accountBalance amount attribute into a number from string

40. Return to the course snippets.txt and copy the JavaScript snippet from Module 8 with the heading Code cell value to convert accountBalance amount into number from string.

41. Return to API Notebook and in the new code cell paste the snippet and press enter.

Convert the accountBalance amount attribute into a number from string

```
7 accountBalanceNumber=Number(accountA.body.accountBalance.amount);  
$6= 8457.9
```

42. Hover over the circle between the new code cell and previous code cell and click Insert Text Cell.

43. Type the value of the text cell as: Write a conditional statement to print the rewards awarded based on the account balance.

Write a conditional statement to print that 1.5x rewards are awarded when the balance in the account is greater than 5000 or 1x rewards are awarded

44. Return to the course snippets.txt and copy the JavaScript snippet from Module 8 with the heading Code cell value for a conditional statement.

45. Return to API Notebook and in the new code cell paste the snippet and press enter.

```
8 if ( accountBalanceNumber > 5000.00 ) {  
9   text = "You will earn 1.5x times the reward for every dollar you spend"; }  
10 else {  
11   text = "You will earn 1x times the reward for every dollar you spend";  
12 }  
13 text;  
  
$7= "You will earn 1.5x times the reward for every dollar you spend"
```

Create and use JavaScript functions inside the API Notebook

46. Hover over the circle between the new code cell and previous code cell and click Insert Text Cell.

47. Type the value of the text cell as: Write a JavaScript function to concatenate the currency and amount attributes to display the amount in one string.

Write a JavaScript function to concatenate the currency and amount attributes in accountBalance to display the amount in one string

48. Return to the course snippets.txt file and copy the JavaScript snippet from Module 8 with the heading Code cell value for a function to concatenate and display currency and amount.
49. Return to API notebook and in the new code cell, paste the snippet and press enter.

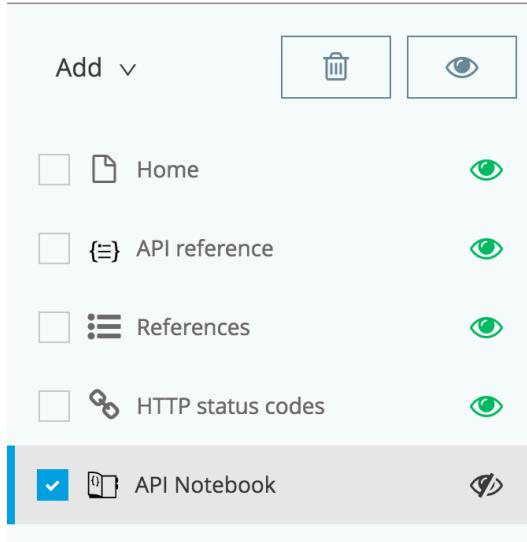
```
14 function concatAccountBalance(accountA) {  
15     return "The accountBalance is " + accountA.body.accountBalance.currency.concat(" "+  
16         accountA.body.accountBalance.amount);  
17 }  
17 concatAccountBalance(accountA);  
  
$8= "The accountBalance is USD 8457.90"
```

50. Scroll up in the API Notebook area and click Save.
51. Click the link ACME Banking API (1.0) – Settings to go back to the API version details page.
52. In the API version details page, click Edit API in API Designer inside the API Definition panel.

View the API Notebook in the Live Portal

53. In the left-side navigation, check the box to select the API Notebook and click the eye button at the top of the section.

ACME Banking API



54. Click the Live Portal button in the top-right corner of API Portal Designer.

55. In the left -side navigation in the API Portal, select API Notebook.

The screenshot shows the API Notebook interface. On the left, a sidebar menu includes Home, API reference, References, HTTP status codes, and API Notebook (which is selected). The main area is titled "API Notebook" and contains two code cells. The first cell, titled "Create ACME Banking API Client to access it's methods and resources", contains the following code:

```
1 API.createClient('acmeBankclient',
  'https://anypoint.mulesoft.com/apiplatform/repository/v2/organizations/a6fb4480-6e30-4248-ac23-
  6e7a751b5606/public/apis/11698133/versions/162026/definition');
```

The second cell, titled "Retrieve a list of customers", contains the following code:

```
2 acmeBankclient.customers.get();
```

At the bottom right of the main area, there are buttons for "Hosted by API Notebook", "Make your own", and "Play notebook".

56. Click the Play notebook icon at the bottom to play all the code cells in the API Notebook.

57. Verify that you see the results returned below each cell in the API Notebook.

API Notebook

The screenshot shows the API Notebook interface after executing the code. The first cell, "Create ACME Banking API Client to access it's methods and resources", now displays the result of the command \$0=:

```
$0= "Create a new code cell and type \"acmeBankclient.\" to explore this API."
```

The second cell, "Retrieve a list of customers", now displays the result of the command acmeBankclient.customers.get();. It shows the response object with expanded properties:

```
2 acmeBankclient.customers.get();
$1= ▼Object {"body": Object, "status": 200, "headers": Object}
▶body: Object
▶headers: Object
status: 200
```

58. Go to the Home page in the API Portal and copy the Portal URL.

59. Return to the course snippets.text file.

60. Locate the Module 8 section and paste API Portal URL in the designated section.

API Portal URL: <https://anypoint.mulesoft.com/apiplatform/mulesoft-1581/#/portals/organizations/a6fb4480-6e30-4248-ac23-6e7a751b5606/apis/11698133/versions/162026/pages/225830>

RAML URL:

61. Return to the API Portal.

62. Select the API Reference page.

63. Scroll down to the bottom of the API Console and click the Copy button next to the Root RAML URL.

The screenshot shows the Anypoint API Platform API Console. At the top, there's a tree view with a node expanded to show '/transactions'. Below this, there are two main sections: one for the POST method and one for the /transactions/{transaction_id} endpoint. The /transactions/{transaction_id} section contains a DELETE method and a GET method. To the right of the /transactions/{transaction_id} section, there is a 'Copy' button. Below the main sections, there is a 'ROOT RAML URL:' field containing a long URL, followed by a 'Download API definition as a .zip file' link.

64. Return to the course snipepets.txt file.
65. Paste the RAML URL in the designated section.
66. Return to the API Portal and close it.

Walkthrough 8-3: Publish a RAML API definition to Anypoint Exchange

In this walkthrough, you enhance the discoverability of an API by adding it to the private Anypoint Exchange. You will:

- Give yourself permission to publish items on the Exchange.
- Add a new RAML API to the private Exchange.
- Submit an item for approval.
- Approve and publish an item to the private Exchange.

The screenshot shows the Anypoint Exchange interface. On the left, there's a sidebar with navigation links like 'Connectors', 'Templates', 'Examples', 'REST APIs', 'SOAP APIs', and 'Others'. The main area has a search bar and a list of items. One item is highlighted: 'Querying a MySQL Data' with a rating of 5 stars. To the right, a modal window titled 'New REST API' is open, showing fields for 'Item Id' (set to 'acme-banking-rest-api'), 'Icon URL', 'Summary' (set to 'Summary'), and 'Author' (set to 'Max Mule'). Buttons for 'Save new item' and 'Discard' are at the bottom right of the modal.

View Exchange roles

1. In the API Portal Designer, select Access Management from the main menu.
2. In the left-side navigation, click Roles.
3. View the Exchange roles.

<input type="radio"/>	Cloudhub Support (Production)	0	Cloudhub (Production) Support users
	Exchange Administrators	1	Exchange Administrators
	Exchange Contributors	0	Exchange Contributors
	Exchange Viewers	0	Exchange Viewers
	Organization Administrators	1	Organization Administrators
	Portals Viewer	0	Viewer of all portals in the organization

4. Click Exchange Administrators.

- Click Exchange Administrators; you should see that you are already an Exchange Administrator.
- Locate the add button that can be used to add additional people.

The screenshot shows the 'Access Management' interface with the 'Exchange Administrators' page selected. On the left, there's a sidebar with 'ACCESS MANAGEMENT' and 'SETTINGS' sections. The main area displays a table with columns 'Name', 'Username', and 'Email'. A search bar at the top contains 'Max Mule'. A blue '+' button is visible in the bottom right corner of the table area.

Create a new item to add to the Exchange

- In the main menu, select Exchange.
- In the Exchange, click the Add item button and select REST API.

The screenshot shows the 'Exchange' interface with the 'REST API' section selected. On the left, there's a sidebar with 'Show content from:' set to 'Everywhere' and a 'Show:' dropdown containing 'Connectors', 'Templates', 'Examples', 'REST APIs', 'SOAP APIs', and 'Others'. Below that is an 'Edit terms' button. The main area has a search bar and a 'Sort by: Rating' dropdown. It lists three items: 'Querying a MySQL Database' (with a 'Download' and 'View details' button), 'Workday Connector' (with a 'View details' button), and 'Proxying a REST API' (with a 'Download' and 'View details' button). At the bottom, there's a link to 'Salesforce to Salesforce and Database Account Broadcast' and a 'Download' button.

9. Set the item name to ACME Banking REST API.

The screenshot shows the 'New REST API' creation screen. The 'Item Id' field contains 'acme-banking-rest-api'. On the right, there are 'Save new item' and 'Discard' buttons. Below the form fields are 'Add tags group' and 'Add links group' buttons.

Item Id:	acme-banking-rest-api
Icon URL:	[Icon URL]
Summary:	Summary
Author:	Max Mule
Author Icon URL:	[Author Icon URL]

10. Scroll down to the versions section and click the Add version button.

Versions

REST API Spec Version	API Version
+ Add version	

11. Enter the following information:

- REST API Spec version: 1.0
- API version: 1.0
- API Portal URL: *Use the URL for your portal that you saved in the course snippets.txt file*
- RAML URL: *Use the URL for your RAML that you saved in the course snippets.txt file*

Versions

RAML version	API version
RAML version *	API version
1.0	1.0
API Portal URL	<input type="text" value="https://anypoint.mulesoft.com/apiplatform/organization-05/#/portals/1"/>
RAML URL	<input type="text" value="https://anypoint.mulesoft.com/apiplatform/repository/v2/organization"/>
Documentation URL	<input type="text"/>

[Done](#) [Discard](#)

12. Click Done.

13. Scroll up and click the Save new item button.

Locate the item

14. Click the Back to list button.

The screenshot shows the details of the 'ACME Banking REST API' item. At the top, there's a 'MuleSoft' logo. Below it, the title 'ACME Banking REST API' is displayed, along with the author 'By: Max Mule'. A 'MuleSoft' badge is also present. To the right, there's a vertical column of buttons: 'Open Portal' (blue), 'Share URL' (white), 'Edit' (blue), 'Clone me!' (white), 'Status: Work in progress' (yellow), 'Request for publish' (white), and 'Delete' (red). Below these buttons, a note says 'Please open the Exchange from Anypoint Studio to rate content.' At the bottom right, there's a 'Back to list' button.

15. Click the REST APIs button in the upper-left corner of the Exchange.

The screenshot shows the Exchange interface with a sidebar on the left. The sidebar has a heading 'Show content from:' followed by a dropdown menu set to 'Everywhere'. Below this, there's a 'Show:' section with several buttons: 'Connectors' (blue), 'Templates' (green), 'Examples' (teal), 'REST APIs' (black), 'SOAP APIs' (purple), and 'Others' (dark grey). At the bottom of the sidebar, there are two sections: 'Horizontal Functions' and 'Collaboration', each with a single blue link.

16. In the Status menu beneath the Add item button, select Waiting for approval; you should see nothing listed.

17. Change the Status menu to Work in progress; you should see the ACME Banking REST API.

The screenshot shows the Exchange interface with the following elements:

- Add item** button (blue)
- Search** input field
- X** button
- Status dropdown**: Set to **Work in progress**.
- Sort by**: Set to **Last modified**.
- ACME Banking REST API** item listed, created by **MuleSoft**.
- Action buttons** for the item: **Open Portal** (blue) and **View details** (white).
- Status dropdown menu** open, showing options: All, Published, Waiting for approval, and **Work in progress**.

Publish the item

18. Click the View details button for the ACME Banking REST API.
19. Locate the Work in progress status.
20. Click the drop-down button on the Publish button and select your organization.

Note: Exchange contributors would have a Request to Publish instead of a Publish button and then an Exchange administrator would have to approve the item.

View the new item in the Exchange

21. In the Status menu beneath the Add item button, select Published; you should see the ACME Banking REST API listed.
22. Change the Status menu to All; you should see your RAML listed.

Module 9: Reusing Patterns

```
traits:  
  cacheable: !include traits/cacheable.raml  
  hasAcceptHeader: !include traits/hasAcceptHeader.raml  
  |  
resourceTypes:  
  collection: !include resourceTypes/collection.raml  
  member: !include resourceTypes/member.raml  
  |  
/accounts:  
  type: { collection:  
    |  { postRequestBodyDatatype: Account,  
        postRequestBodyExample: !include examples/AccountExample.json,  
        customErrorMessageType: CustomErrorMessage  
    }  
  }  
  |  
  /{account_id}:  
    type: { member:  
      |  { uriParameterName: account_id,  
          getResponseBodyDataType: Account,  
          getResponseBodyExample: !include examples/AccountExample.json,  
          customErrorMessageType: CustomErrorMessage  
      }  
    }  
  }  
}|
```

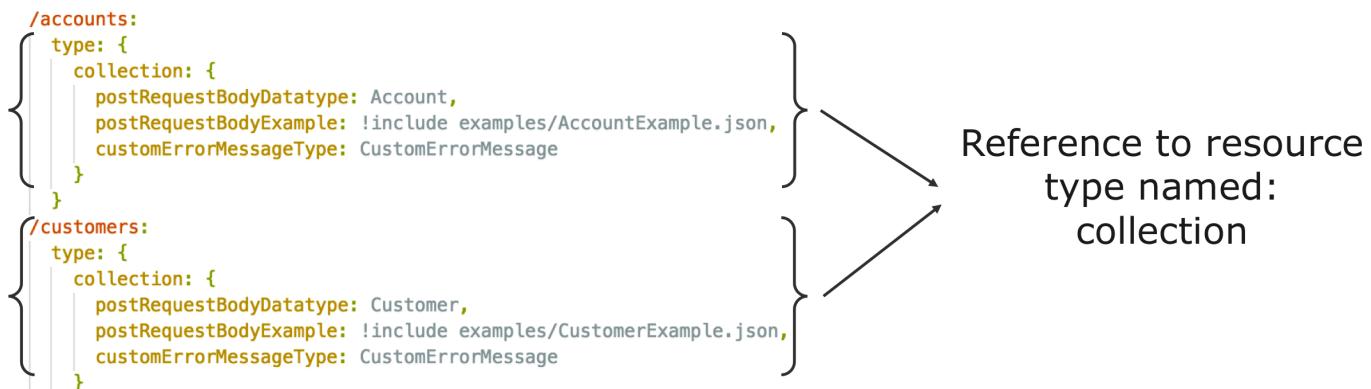
Objectives:

- Use resource types to reuse parts of method definitions.
- Use traits to enhance responses.
- Use the Concurrent Modification API design pattern when using shared resources.
- Use the Asynchronous API design pattern to handle and track concurrent requests.

Walkthrough 9-1: Define and use a resource type for resources that perform operations on a collection

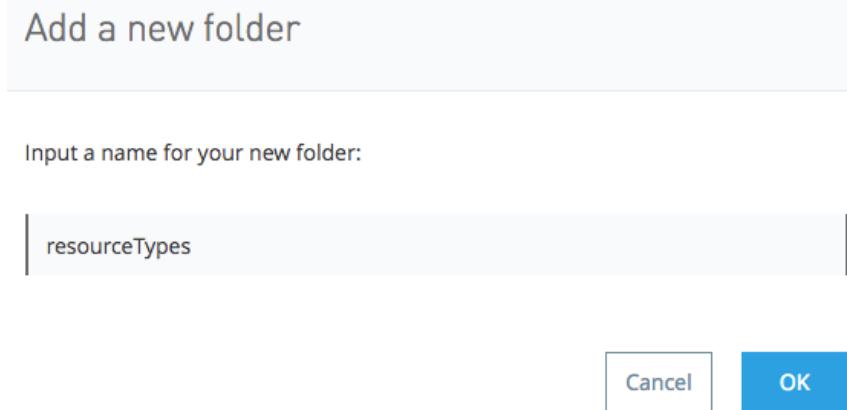
In this walkthrough, you define a collection resource type to reuse some parts of the method definition for resources that perform operations on a collection. You will:

- Define a collection resource type fragment.
- Use a mapping to pass parameter values to a resource type.
- Reference the resource type in the RAML API definition.



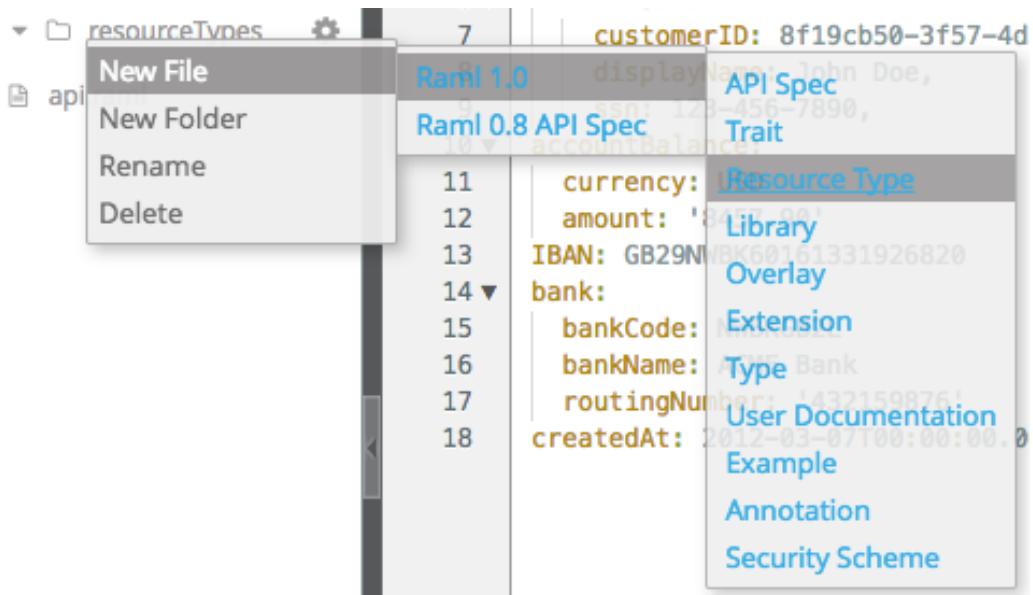
Define a collection resource type fragment

1. Return to API Designer.
2. Right-click inside the file browser section, and select New Folder.
3. In the Add a new folder dialog box, set the name of the folder to resourceTypes.

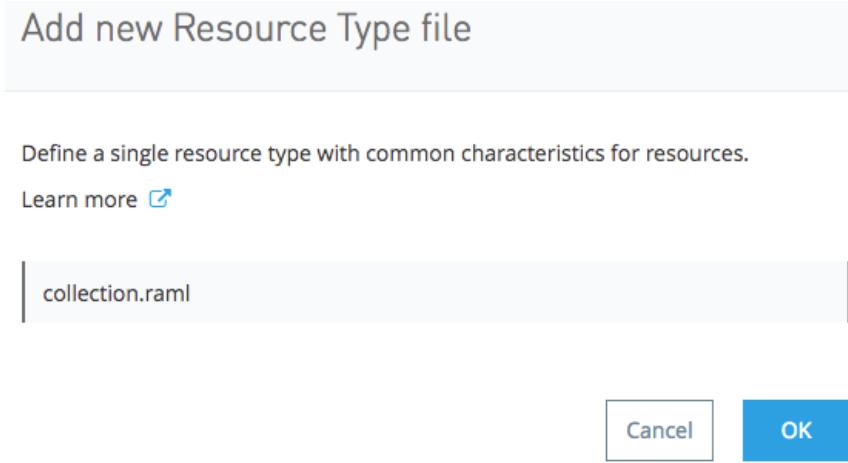


4. Click OK.

5. In the file browser, right-click the resourceTypes folder and select New File > RAML 1.0 > Resource Type.



6. In the Add new Resource Type file dialog box, rename the file as collection.raml.



7. Click OK.
8. In the file browser, select api.raml to open the RAML definition file.

9. In the /customers resource, select and copy the post method definition.

```
59 ▼  post:
60    description: Add a new customer
61    ▼ body:
62      application/json:
63        type: Customer
64        example: !include examples/CustomerExample.raml
65    ▼ responses:
66      201:
67        headers:
68          Location:
69            description: URL of the new customer information
70        body:
71          application/json:
72      503:
73        body:
74          application/json:
75            properties:
76              statusCode: string
77              message: string
```

10. In the file browser, expand the resourceTypes folder and click collection.raml.

11. Paste the lines after the first line in collection.raml.

```
1 ▼ #%RAML 1.0 ResourceType
2 ▼ post:
3   description: Add a new customer
4   ▼ body:
5     application/json:
6       type: Customer
7       example: !include examples/CustomerExample.raml
8   ▼ responses:
9     201:
10    headers:
11      Location:
12        description: URL of the new customer information
13    body:
14      application/json:
15    503:
16    body:
17      application/json:
18        properties:
19          statusCode: string
20          message: string
```

12. Select the pasted lines and press Shift + Tab to move a tab space to the left and properly indent the lines.

Use mappings to pass parameter values to resource types

13. In the collection.raml resource type file, go to the line that contains the description for the post method.

14. Replace the word customer with <<resourcePathName | !singularize>>

```
1  #%RAML 1.0 ResourceType
2 ▼ post:
3   | description: Add a new <<resourcePathName | !singularize>>
4 ▼   | body:
5 ▼     | application/json:
```

15. Similarly, in the description node for the Location header, replace the word customer with <<resourcePathName | !singularize>>.

16. Locate the line of code that sets the post request body type to Customer.

17. Delete the Customer datatype and set it to the following mapping:

```
<<resourcePathName | !singularize | !uppercase>>
```

18. Go to the next line that defines the example for the post method's request body datatype and delete the example value and type <<postRequestBodyExample>>.

```
1  #%RAML 1.0 ResourceType
2 ▼ post:
3   | description: Add a new <<resourcePathName | !singularize>>
4 ▼   | body:
5 ▼     | application/json:
6       | type: <<postRequestBodyDatatype>>
7       | example: <<postRequestBodyExample>>
8 ▼   | responses:
9 ▼     | 201:
```

19. Go to the line that contains the HTTP 503 response body type CustomErrorMessage and replace the type node value as <<customErrorMessageType>>.

```
15 ▼   | 503:
16 ▼     |   | body:
17     |   |   | application/json:
18     |   |   |   | type: <<customErrorMessageType>>
```

20. Save the project.

Reference the resource type in the RAML API definition

21. In the file browser section, select api.raml.

22. In the /customers resource, go to the post method definition and delete it.

23. Add a new line before the /customers resource and add the resourceTypes node:

resourceTypes:

```
19
20   | resourceTypes:
21   |
22 ▼   | /customers:
```

24. Add a new line below and press tab.

25. Include the collection resourcetype by typing:

collection: !include resourceTypes/collection.raml

```
18   |   | CustomErrorMessage: !include datatypes/CustomErrorMessage.raml
19
20   |   | resourceTypes:
21   |   |   | collection: !include resourceTypes/collection.raml
22 ▼   |   | /customers:
```

26. Press enter.

27. In the /customers resource, add a new line before the get method.

```
20 ▼   | /customers:
21
22 ▼   |   | get:
23   |   |   | description: Retrieve a list of customers
24 ▼   |   |   | headers:
```

28. Reference the type collection and add the mapping values by typing:

```
type: { collection: { postRequestBodyExample: !include
..examples/CustomerExample.raml, customErrorMessageType:
CustomErrorMessage } }
```

```
23 ▼   | /customers:
24   |   | type: { collection: { postRequestBodyExample: !include
..examples/CustomerExample.raml, customErrorMessageType:
CustomErrorMessage } }
25 ▼   |   |   | get:
```

Note: Ignore the warning in the line that references the resource type collection. The warning is an issue in the parser caused because of including the root in the path for the example.

View the documentation

29. In the API Console, verify that you see the resource type listed across the /customers resource.

The screenshot shows the 'Resources' section of the API Console. A grey bar at the top has a checkbox labeled 'API is behind a firewall (?)'. Below it is a 'Collapse All' button. Underneath, a list item for '/customers' is shown. To its right is a button labeled 'Type: collection'. Below this are two buttons: a green 'POST' button and a blue 'GET' button. The '/customers' item has a downward arrow icon to its left.

30. Click the POST method for the /customers resource and verify the example for the post request body is in JSON format.

BODY

application/json

Examples: [Example](#)

```
{  
    "customerID": "8f19cb50-3f57-4d38",  
    "firstName": "John",  
    "lastName": "Doe",  
    "displayName": "John Doe",  
    "address": {  
        "addressLine1": "1234 Lane",  
        "addressLine2": "Apt.#620",  
        "city": "San Francisco",  
        "state": "California",  
        "zipCode": "94108",  
        "country": "United States"  
    },  
    "phone": "415-000-0000",  
    "email": "johndoe@example.com",  
    "ssn": "124-456-7890",  
    "dateOfBirth": "1983-01-01"  
}
```

31. Click CLOSE X in the /customers resource.

Walkthrough 9-2: Define and use a resource type for resources that perform operations on a member referenced with an ID

In this walkthrough, you define a member resource type to reuse some parts of the method definition for resources that perform operations on a member. You will:

- Define a member resource type fragment.
- Use a mapping to pass parameter values to the resource type.
- Reference the resource type in the RAML API definition.

The screenshot shows the API Designer interface. On the left, there is a code editor with RAML code. The code defines a member resource type fragment and a collection resource. The collection resource has a member resource type defined for it. On the right, the API interface is displayed. It shows a collection resource at `/customers` (Type: collection) with a POST button. Below it is a member resource at `/customers/{customer_id}` (Type: member) with DELETE, PATCH, and GET buttons.

```
68 |   |   |   | description: ID of the customer
69 |   |   |   | type: { member: { getResponseBodyDataType: Customer,
70 |   |   |   |   | getResponseBodyExample: !include
71 |   |   |   |   | examples/CustomerExample.json,
72 |   |   |   |   | customErrorMessageType: CustomErrorMessage } }
70 ▼
71
72 ▼
|   |   | patch:
|   |   |   | description:
|   |   |   | information
|   |   |   | responses:
```

Resources

API is behind a firewall (?)

collapse All

/customers Type: collection

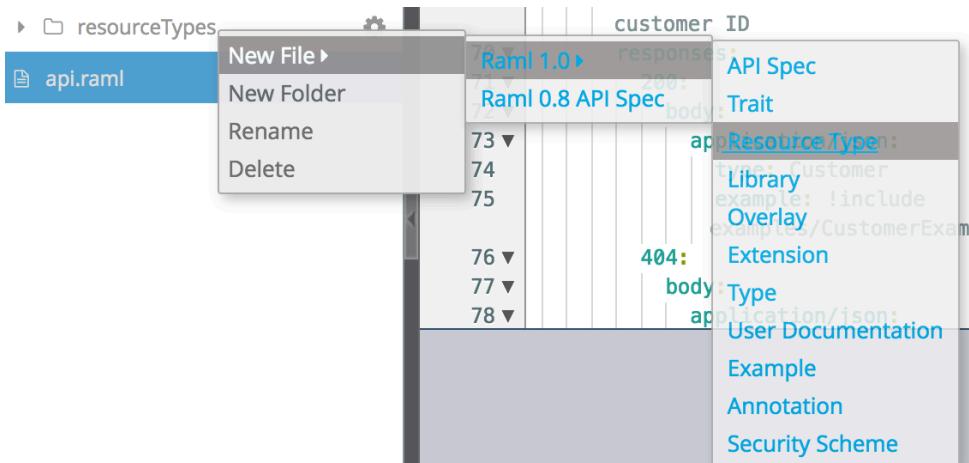
POST GET

/customers/{customer_id} Type: member

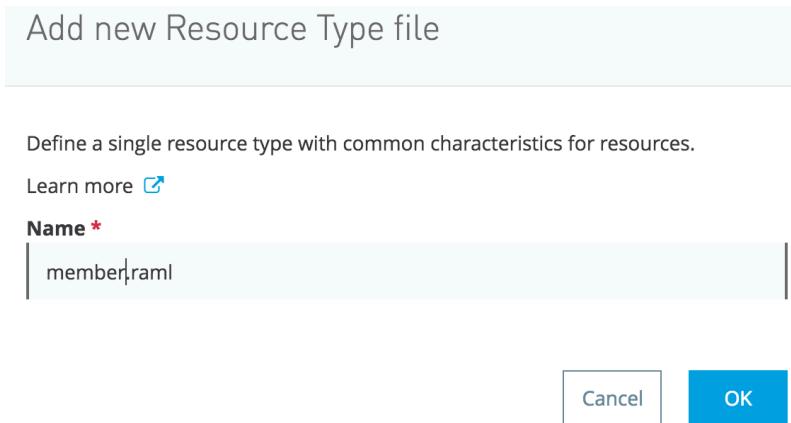
DELETE PATCH GET

Define a member resource type fragment

1. Return to API Designer.
2. In the file browser, right-click the resourceTypes folder and select New File > RAML 1.0 > Resource Type.



3. In the Add new Resource Type file dialog box, set the name to member.raml and click OK.



4. In the file browser, select api.raml.
5. Locate the /{customer_id} resource and select the lines from the uriParameters to the end of the get method definition.

```
60 ▼  |  /{customer_id}:  
61 ▼  |  |  uriParameters:  
62  |  |  |  customer_id:  
63  |  |  |  |  description: ID of the customer  
64 ▼  |  |  |  get:  
65  |  |  |  |  description: Retrieve a customer with a specific customer ID  
66 ▼  |  |  |  |  headers:  
67 ▼  |  |  |  |  |  Accept?:  
68  |  |  |  |  |  |  description: Specify the media type of the response  
69  |  |  |  |  |  |  example: application/xml  
70 ▼  |  |  |  |  responses:  
71 ▼  |  |  |  |  |  200:  
72 ▼  |  |  |  |  |  |  body:  
73 ▼  |  |  |  |  |  |  |  application/json:  
74  |  |  |  |  |  |  |  |  type: Customer  
75  |  |  |  |  |  |  |  |  example: !include examples/CustomerExample.raml  
76 ▼  |  |  |  |  |  |  |  application/xml:  
77  |  |  |  |  |  |  |  |  type: Customer  
78  |  |  |  |  |  |  |  |  example: !include examples/CustomerExample.raml  
79 ▼  |  |  |  |  | 404:  
80 ▼  |  |  |  |  |  |  body:  
81  |  |  |  |  |  |  |  application/json:  
82  |  |  |  |  |  |  |  |  type: CustomErrorMessage  
83 ▼  |  |  |  |  | 406:  
84 ▼  |  |  |  |  |  |  body:  
85  |  |  |  |  |  |  |  application/json:  
86  |  |  |  |  |  |  |  type: CustomErrorMessage
```

6. In the file browser, click the resourceTypes/member.raml file.
7. Paste the copied lines after the first line in member.raml.

- Select the pasted lines and press Shift+tab to indent the lines properly.

```

1  %%RAML 1.0 ResourceType
2 ▼ get:
3   description: Retrieve a customer with a specific
4   customer ID
5   responses:
6     200:
7       body:
8         application/json:
9           type: Customer
10          example: !include examples/CustomerExample.raml
11
12     404:
13       body:
14         application/json:
15           properties:
16             statusCode: string
17             message: string

```

Use mappings to pass parameter values to resource types

- In the member.raml file, go to the line that contains the uriParameter customer_id and replace the variable with <><uriParameterName>>.

```

2 ▼ uriParameters:
3   | <><uriParameterName>>:
4   |   description: ID of the customer
5 ▼ get:

```

- In the next line that contains the description, replace customer with <><resourcePathName | !singularize>>.

```

1  %%RAML 1.0 ResourceType
2 ▼ uriParameters:
3   | <><uriParameterName>>:
4   |   description: ID of the <><resourcePathName | !singularize>>
5 ▼ get:

```

- Go to the line that contains the get method description and replace the word customer with <><resourcePathName | !singularize>>.

```

2 ▼ get:
3   description: Retrieve a <><resourcePathName |
4   !singularize>> with a specific <><resourcePathName |
5   !singularize>> ID
6   responses:

```

- Go to the line that contains the HTTP 200 response body type Customer and delete the type value Customer, and replace it with <><resourcePathName | !singularize | !uppercase>>.
- Go to the next line and replace the value of the example node with <><getResponseBodyExample>>.

```

4 ▼ | responses:
5 ▼ |   200:
6 ▼ |     body:
7 ▼ |       application/json:
8 |         type: <>getResponseBodyDataType>>
9 |         example: <>getResponseBodyExample>>

```

14. Go to the HTTP 404 response body and replace the CustomErrorMessage datatype with <<customErrorMessageType>>.
15. Similarly, in the HTTP 406 response body change the CustomErrorMessage datatype with the mapping

```

10 ▼ |   404:
11 ▼ |     body:
12 |       application/json:
13 |         type: <<customErrorMessageType>>
14 ▼ |   delete:

```

16. Save the project.

Reference the member resource type fragment in the RAML API definition

17. In the file browser, select api.raml.
18. Go the /{customer_id} resource and delete the lines from the uriParameters to the get method definition.

```

64 ▼ |   /{customer_id}:
65 ▼ |     uriParameters:
66 |       customer_id:
67 |         description: ID of the customer
68 ▼ |     patch:
69 |       description: Update a specific customer
information
70 ▼ |     responses:
71 |       204:
72 ▼ |       501:
73 ▼ |         body:
74 ▼ |           application/json:
75 ▼ |             properties:
76 |               statusCode: string
77 |               message: string

```

19. Go to the resource type definition node above the /customers resource and add a new line below the collection resource type.

20. In the new line type:

```
member: !include resourceTypes/member.raml
```

```
20 ▼ | resourceTypes:  
21 | | collection: !include resourceTypes/collection.raml  
22 | | member: !include resourceTypes/member.raml  
--
```

Note: Make sure the indentation aligns with the previous collection resource type reference.

21. In the /{customer_id} resource go to the end of the line that contains the description for the URI parameter and press enter to add a new line below it.

22. Press backspace or delete two time to align the indentation with the next line and type the following (or copy from the snippets.txt):

```
type: { member: { getResponseBodyExample: !include  
..examples/CustomerExample.raml, customErrorMessageType:  
CustomErrorMessage } }
```

```
61 ▼ | /{customer_id}:  
62 | | type: { member: { uriParameterName: customer_id,  
getResponseBodyExample: !include ..examples/CustomerExample.raml,  
customErrorMessageType: CustomErrorMessage } }  
63 ▼ | patch:  
64 | | description: Update a specific customer information
```

23. Save the project.

24. In API Console, verify that the /customers/{customer_id} resource contains a type: member tag.

The screenshot shows the Mule API Console interface. At the top, there's a checkbox for "API is behind a firewall (?)". Below that is a "Resources" section with a "Collapse All" button. Under "Resources", there's a tree view with a minus sign next to "/customers" and a plus sign next to "/customers/{customer_id}". The "/customers" node has a tooltip "Type: collection". Below the tree, there are three horizontal buttons for methods: "POST" (green), "DELETE" (red), "PATCH" (blue), and "GET" (light blue). The "/customers/{customer_id}" node also has a tooltip "Type: member".

25. Click GET and scroll down and verify that you see the HTTP 200 response message example.

Response

STATUS 200

200

Body application/json

Examples: Example

404

406

```
{  
    "customerID": "8f19cb50-3f57-4d38",  
    "firstName": "John",  
    "lastName": "Doe",  
    "displayName": "John Doe",  
    "address": {  
        "addressLine1": "1234 Lane",  
        "addressLine2": "Apt.#620",  
        "city": "San Francisco",  
        "state": "California",  
        "zipCode": "94108",  
        "country": "United States"  
    }  
}
```

Walkthrough 9-3: Refactor resources to use resource types

In this walkthrough, you refactor the ACME Banking API. You will:

- Refactor the resource methods by referencing the already created resource types.

The screenshot shows the API Designer interface. On the left, the code editor displays the following Raml code for the /accounts resource:

```
94 ▼ /accounts:
95   type: { collection: { postRequestBodyDatatype: Account,
96     postRequestBodyExample: !include examples/AccountExample.raml,
97     customErrorMessageType: CustomErrorMessage } }
98   ▼ {account_id}:
99     type: { member: { uriParameterName: account_id, $ref: '#/types/Account',
100       getResponseBodyExample: !include examples/GetAccountExample.raml,
101       customErrorMessageType: CustomErrorMessage } }
102     put:
103       description: Update a specific account information
104       body:
105         application/json:
106           type: Account
```

On the right, the API resources panel lists the following endpoints:

- /customers (Type: collection)
 - POST
 - GET
- /customers/{customer_id} (Type: member)
 - DELETE
 - PATCH
 - GET
- /customers/{customer_id}/accounts (Type: collection)
 - GET
- /accounts (Type: collection)
 - POST
- /accounts/{account_id} (Type: member)
 - DELETE
 - PUT
 - GET
- /accounts/{account_id}/transactions (Type: collection)
 - GET
- /transactions (Type: collection)
 - POST

Refactor /accounts resource

- Return to API Designer.
- In api.raml, go to the /accounts main resource and delete the post method definition.
- Return to the course snippets.text file and copy the lines for the Collection resource type for /accounts resource.

The screenshot shows the API Designer interface. The code editor now contains the following Raml code for the /accounts resource, reflecting the changes made in the walkthrough:

```
108 ▼ /accounts:
109   type: { collection: { postRequestBodyExample: !include
110     ../examples/AccountExample.raml, customErrorMessageType:
111     CustomErrorMessage } }
112   ▼ {account_id}:
```

Refactor /accounts/{account_id} resource

4. In the /{account_id} nested resource, delete the lines starting from the uriParameters to the end of the get method definition.

```
106 ▼ |  /{account_id}:  
107 ▼ |    uriParameters:  
108 |      account_id:  
109 |      description: ID of the account  
110 ▼ |    get:  
111 |      description: Retrieve an account information with a specific account ID  
112 ▼ |      responses:  
113 ▼ |        200:  
114 ▼ |          body:  
115 ▼ |            application/json:  
116 |              type: Account  
117 |              example: !include examples/AccountExample.raml  
118 ▼ |        404:  
119 ▼ |          body:  
120 ▼ |            application/json:  
121 ▼ |              properties:  
122 |                statusCode: string  
123 |                message: string  
124 ▼ |    delete:  
125 |      description: Delete an account with a specific account ID  
126 ▼ |      responses:  
127 |        204:  
128 ▼ |        200:  
129 |          body:  
130 |            application/json:  
131 ▼ |  put:
```

5. Copy the lines from the course snippets.txt file for the Member resource type reference code for /{account_id} resource.

```
110 ▼ |  /{account_id}:  
111 |    type: { member: { uriParameterName: account_id,  
112 |      getResponseBodyExample: !include ../examples/AccountExample.raml,  
113 |      customErrorMessageType: CustomErrorMessage } }  
114 ▼ |  delete:
```

Refactor /transactions resource

6. Go to the /transactions main resource and delete the post method definition.
7. Copy the lines from the course snippets.txt for the Collection resource type reference code for /transactions resource.

```
162 ▼ |  /transactions:  
163 |    type: { collection: { postRequestBodyExample: !include  
164 |      ../examples/TransactionExample.raml, customErrorMessageType:  
165 |      CustomErrorMessage } }  
166 ▼ |  /{transaction_id}:
```

Refactor /transactions/{transaction_id} resource

8. In the /{transaction_id} nested resource, delete the lines starting from the uriParameters to the end of the get method definition.

```
141 ▼  /{transaction_id}:  
142 ▼    uriParameters:  
143      transaction_id:  
144      | description: ID of the transaction  
145 ▼    get:  
146      | description: Retrieve a transaction with a specific transaction ID  
147      | responses:  
148 ▼        200:  
149      |           body:  
150 ▼            application/json:  
151          |               type: Transaction  
152          |               example: !include examples/TransactionExample.raml  
153 ▼        404:  
154      |           body:  
155 ▼            application/json:  
156      |               properties:  
157          |                 statusCode: string  
158          |                 message: string  
159 ▼    delete:  
160      | description: Delete a specific transaction  
161      | responses:  
162          204:  
163 ▼        200:  
164      |           body:  
165          |             application/json:
```

9. Copy the lines from the course snippets.txt file for the Member resource type reference code for /{transaction_id} resource.

```
164      |           customErrorMessage } ,  
165    /{transaction_id}:  
166      | type: { member: { uriParameterName: transaction_id,  
167      |           getResponseBodyExample: !include  
168      |             ../examples/TransactionExample.raml, customErrorMessageType:  
169      |               CustomErrorMessage } }
```

10. Save the project.

Walkthrough 9-4: Define and use various traits for resources and methods

In this walkthrough, you refactor the API definition by creating reusable traits for resources and methods. You will:

- Define a cacheable trait to be applied to the /customers resource get method.
- Define a flexible content type trait to be applied to resource methods with an Accept header in the request.
- Refactor the resource methods to use the trait.

The screenshot shows the API Designer interface for the '/customers' resource. The 'Traits' field is set to 'cacheable, hasAcceptHeader'. The 'Request' tab is selected, showing the GET method with the following details:
- Description: Retrieve a list of customers
- Headers: Accept string
- Responses: 200

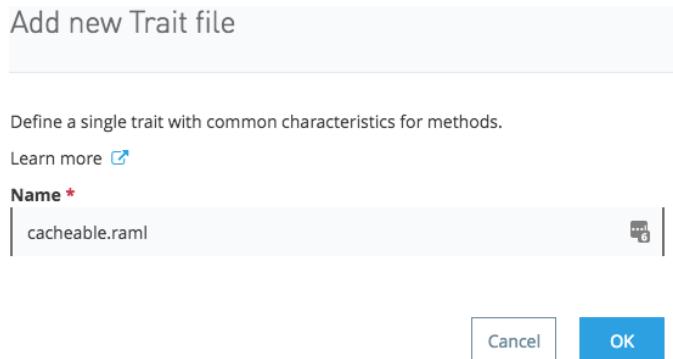
Define a cacheable trait to be applied to the /customers resource get method

1. Return to API Designer.
2. From the Project menu, select New Folder.
3. In the Add new folder dialog box, set the name to traits and click OK.

The screenshot shows the 'Add a new folder' dialog box. The 'Name' field is filled with 'traits'. At the bottom right are 'Cancel' and 'OK' buttons.

4. In the file browser, right-click the traits folder and select New File > RAML 1.0 > Trait.

- In the Add new Trait file dialog box, set the name to cacheable.raml and click OK.



- In the file browser, select api.raml.
- Go to the /customers resource get method HTTP responses line.
- Select the lines from responses to the Expires header format example line and copy them.

```

32 ▼ |   responses:
33 ▼ |     200:
34 ▼ |       headers:
35 ▼ |         Cache-Control:
36 ▼ |           description: |
37 |             Activates caching and defines cache behavior through cache response directives.
38 |             Usually defines public or private (cacheable by proxy or not) and max-age for resource.
39 |             See http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html for more information.
40 |             example: private, max-age=31536000
41 ▼ |           Expires:
42 ▼ |             description: |
43 |               Sets a date in RFC 1123 format from which the cached resource should no longer be considered
44 |               valid.
45 |               If both the Expires header and max-age in the Cache-Control header are set, max-age will take
46 |               precedence.
47 |               type: datetime
        example: Tue, 18 Apr 2017 09:30:41 GMT
        format: rfc2616

```

- In the file browser, select traits/cacheable.raml.
- Paste the copied lines, in the second line and correct the indentation for the lines.

```

1  #%RAML 1.0 Trait
2  responses:
3  200:
4  |   headers:
5  |     Cache-Control:
6  |       description: |
7 |         Activates caching and defines cache behavior through cache response directives.
8 |         Usually defines public or private (cacheable by proxy or not) and max-age for resource.
9 |         See http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html for more information.
10 |         example: private, max-age=31536000
11 |       Expires:
12 |         description: |
13 |           Sets a date in RFC 1123 format from which the cached resource should no longer be considered valid.
14 |           If both the Expires header and max-age in the Cache-Control header are set, max-age will take
15 |           precedence.
16 |           type: datetime
17 |           example: Tue, 18 Apr 2017 09:30:41 GMT
        format: rfc2616

```

- Go to the beginning of the responses line and add a press enter to add a new line above it.
- Go to the new line and In the shelf, click usage.

13. Type the value for the usage node as: Apply this trait to any GET method that supports caching control.

```

1  #%RAML 1.0 Trait
2  usage: Apply this trait to any GET method that supports caching control.
3 ▼ responses:
4 ▼   200:
5   |   headers:
6   |   |   Cache-Control:
7   |   |   description: |
8   |   |   Activates caching and defines cache behavior through cache response directives.
9   |   |   Usually defines public or private (cacheable by proxy or not) and max-age for resource.
10  |   |   See http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html for more information.
11  |   |   example: private, max-age=31536000
12 ▼   Expires:
13 ▼     description: |
14     Sets a date in RFC 1123 format from which the cached resource should no longer be considered valid.
15     If both the Expires header and max-age in the Cache-Control header are set, max-age will take precedence.
16     type: datetime
17     example: Tue, 18 Apr 2017 09:30:41 GMT
18     format: rfc2616

```

14. Save the project.

Refactor the /customers get method to apply the cacheable trait

15. In the file browser, select api.raml.
 16. In the RAML editor, add a new line above the resourceTypes declaration.
 17. In the shelf, click traits.

```

18  |   CustomErrorMessage: !include datatypes/C
19  |
20  |
21 ▼ resourceTypes:
22   |   collection: !include resourceTypes/collection.ram
23   |   member: !include resourceTypes/member.ram
24
25 ▼ /customers:
26   |   type: { collection: { postRequestBodyData:
27   |   |   examples/CustomerExample.json, customError:
28   |   |   get:
29   |   |   description: Retrieve a list of customers
30   |   |   headers:
31   |   |   |   Accept?:
32   |   |   |   |   description: Specify the media type
33   |   |   |   |   example: application/xml
34   |   |   responses:
35   |   |   200:
36   |   |   |   headers:
37   |   |   |   |   Cache-Control:
38   |   |   |   |   description: |
39   |   |   |   |   Activates caching and defines cache behavior through cache response directives.
40   |   |   |   |   Usually defines public or private (cacheable by proxy or not)
41   |   |   |   |   See http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html for more information.
42   |   |   |   |   example: private, max-age=31536000
43   |   |   |   Expires:

```



18. In the new line type the following to add the cacheable trait to the definition:

```
cacheable: !include traits/cacheable.raml  
20  traits:  
21  | cacheable: !include traits/cacheable.raml|  
22 ▼ resourceTypes:  
23  | collection: !include resourceTypes/collection.raml  
24  | member: !include resourceTypes/member.raml
```

19. Press enter.

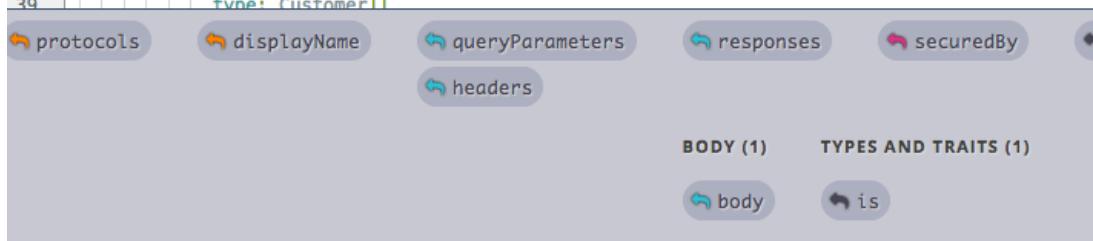
20. In the /customers resource get method HTTP 200, select lines from headers to the line before the body definition and delete them.

```
34 ▼ | responses:  
35 ▼ | | 200:  
36 ▼ | | | headers:  
37 ▼ | | | | Cache-Control:  
38 ▼ | | | | | description: |  
39 | | | | | | Activates caching and defines cache behavior through cache response directives.  
40 | | | | | | Usually defines public or private (cacheable by proxy or not) and max-age for resource.  
41 | | | | | | See http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html for more information.  
42 | | | | | | example: private, max-age=31536000  
43 ▼ | | | | Expires:  
44 ▼ | | | | | description: |  
45 | | | | | | Sets a date in RFC 1123 format from which the cached resource should no longer be considered  
46 | | | | | | valid.  
47 | | | | | | If both the Expires header and max-age in the Cache-Control header are set, max-age will take  
48 | | | | | | precedence.  
49 | | | | | type: datetime  
50 | | | | | example: Tue, 18 Apr 2017 09:30:41 GMT  
50 ▼ | | | | | format: rfc2616  
50 ▼ | | | | body:
```

21. Add a new line below the line that contains the description for the get method.

22. In the shelf below, click is.

```
26 ▼ | /customers:  
27 | | type: { collection: { postRequestBodyDatatype: Customer, postRequestBodyExample: !include  
examples/CustomerExample.json, customErrorMessageType: CustomErrorMessage } }  
28 ▼ | | get:  
29 | | | description: Retrieve a list of customers  
30 | | |  
31 ▼ | | | headers:  
32 ▼ | | | | Accept?:  
33 | | | | | description: Specify the media type of the response  
34 | | | | | example: application/xml  
35 ▼ | | | responses:  
36 ▼ | | | | 200:  
37 ▼ | | | | | body:  
38 | | | | | | application/json:  
39 | | | | | | type: Customer[]
```



The screenshot shows the MuleSoft Anypoint Studio interface. The code editor displays the RAML configuration for the /customers resource. Below the editor, a shelf bar contains several tabs: 'protocols', 'displayName', 'queryParameters', 'responses', 'securedBy', 'headers' (which is highlighted in blue), and 'body'. At the bottom of the interface, there are two buttons: 'BODY (1)' and 'TYPES AND TRAITS (1)'. Under 'BODY (1)', there is a button labeled 'body'. Under 'TYPES AND TRAITS (1)', there is a button labeled 'is'.

23. In the shelf, click cacheable.

```
28 ▼ | get:
29 |   description: Retrieve a list of customers
30 |   is: cacheable
31 ▼ |   headers:
32 ▼ |     Accept?:
33 |       description: Specify the media type of the response
34 |       example: application/xml
35 ▼ |   responses:
36 ▼ |     200:
37 ▼ |       body:
38 |         application/json:
39 |           type: Customer[]
```

OTHERS (1)

cacheable

Define a flexible content type trait to be applied to resource methods with an Accept header in the request

24. In the file browser, right-click the traits folder and select New File > RAML 1.0 > Trait.
25. In the Add new trait file dialog box, set the name to hasAcceptHeader.raml and click OK.
26. In the file browser, select api.raml.
27. Go to the /customers resource get method.
28. Select the lines from headers to the example line for the Accept header and copy them.

```
32 ▼ |   headers:
33 ▼ |     Accept?:
34 |       description: Specify the media type of the response
35 |       example: application/xml
36 ▼ |   responses:
```

29. In the file browser, select hasAcceptHeader.raml and paste the lines in the second line.

```
1  #%RAML 1.0 Trait
2 ▼ headers:
3 ▼   Accept?:
4 |     description: Specify the media type of the response
5 |     example: application/xml
```

30. Fix the indentation and save the file.
31. In the file browser, select api.raml
32. Add a new line after the line that includes the cacheable trait.

33. Include the hasAcceptHeader trait.

```
20 ▼ traits:
21   | cacheable: !include traits/cacheable.raml
22   | hasAcceptHeader: !include traits/hasAcceptHeader.raml
23   |
24 ▼ resourceTypes:
25   | collection: !include resourceTypes/collection.raml
26   | member: !include resourceTypes/member.raml
```

34. Go to the /customers resource get method and delete the lines that define the Accept header for the request.

```
27 ▼ /customers:
28   | type: { collection: { postRequestBodyData:
29     | examples/CustomerExample.json, customExample:
29 ▼   | get:
30     |   description: Retrieve a list of customers
31     |   is: cacheable
32 ▼   |   responses:
33 ▼     |     200:
34 ▼       |       body:
```

35. In the /customers resource get method, go to the line that contains the cacheable trait applied and change the line to:

```
is: [cacheable, hasAcceptHeader]
```

```
30 ▼   -----, -----
31   |   get:
32     |     description: Retrieve a list of customers
33     |     is: [cacheable, hasAcceptHeader]
34 ▼       |       responses:
34 ▼         |         200:
```

36. Similarly, in the /accounts nested resource get method, delete the Accept header definition lines and add the trait.

```
58 ▼   -----, -----
59 ▼   |   /accounts:
60     |     get:
61       |       description: Retrieve a list of accounts for a specific customer
62     |       is: hasAcceptHeader|
62 ▼       |       responses:
```

37. Similarly, in the /transactions nested resource get method, delete the Accept header definition lines and add the trait.

```
123 ▼ | /transactions:  
124 ▼ |   get:  
125 |     description: Retrieve a list of  
126 |     transactions for a specific account  
127 |     is: hasAcceptHeader  
      responses:
```

38. Save the project.

View the documentation

39. In the API Console, click GET for the /customers resource.

The screenshot shows the API Console interface. At the top, it says "Resources" and "API is behind a firewall (?)". Below that, the "/customers" resource is listed as a "collection" with traits "cacheable, hasAcceptHeader". There are two buttons: "POST" and "GET". Underneath, there's a "Request" section with a "Try it" button.

40. Verify that you can view the Accept header in the request and the cache-control headers that were added using traits in the HTTP 200 response.

The screenshot shows the API Console interface for the "/customers" resource. It includes sections for "HEADERS" (with "Accept" set to "string"), "SECURITY SCHEMES" (with "Anonymous"), and "Response". The "Response" section shows a "STATUS 200" block with "Headers" (Cache-Control: "required string") and an "Expires" header (described as "Sets a date in RFC 1123 format from which the cached resource should no longer be considered valid"). Other status codes like 404 and 406 are also listed.

41. Click CLOSE X in the /customers resource.

Module 10: Modularizing APIs

```
1  %%RAML 1.0 Library
2  usage: Use these traits to define request and re
put and patch methods
3 ▼ traits:
4    cacheable: !include ../traits/cacheable.raml
5    hasAcceptHeader: !include ../traits/hasAccepth
6 ▼ hasResponseItems:
7 ▼   responses:
8     200:
9       body:
10      application/json:
11        type: <<typeName>> []
12      application/xml:
13        type: <<typeName>> []
14 ▼   404:
15     body:
16       application/json:
17         type: <<customErrorMessageDataType>>
18 ▼   406:
19     body:
20       application/json:
21         type: <<customErrorMessageDataType>>
22 ▼ hasPutOrPatchResponseItems:
23   responses:
24     204:
25     501:
26     body:
27       application/json:
28         type: <<customErrorMessageDataType>>

1  %%RAML 1.0 Overlay
2  usage: Spanish localization
3  extends: ../api.raml
4 ▼ documentation:
5   - title: ACME API Bancario Home
6     content: |
7       **ACME Banca API** permite a los desarrolladores crear aplicaciones que hacen uso de la información del recurso a los métodos implementados en la API.
8
9       Esta API contiene una funcionalidad que permite a los desarrolladores recuperar y manipular el _cliente_, la _cuenta_ y la información de la _transacción_. Echa un vistazo a la API [Portal]() para más detalles.
10  - title: Banco ACME Headline
11    content: |
12      **Banco ACME** es una _multinacional de servicios bancarios_ y financieros de la organización.
13
14  /customers:
15    get:
16      description: Recuperar una lista de clientes

1  %%RAML 1.0 Extension
2  baseUri: https://mocksvc.mulesoft.com/mocks/c7eb6e39-a79f-435b-9488-
14f45b3276f4<<insert staging link here>>
3  extends: ../api.raml
```

Objectives:

- Use libraries for greater API composability.
- Override resource information using overlays.
- Use overlays to internationalize resources.
- Use extensions to enhance resources.

Walkthrough 10-1: Create and use a library of traits

In this walkthrough, you will create a library file with several traits listed in them. You will:

- Create a library fragment file.
- Refactor and move existing traits inside the library file.
- Create and define new resource method request and response traits.
- Refactor the API definition to reuse the traits from the library.

```
1  #RAML 1.0 Library
2  usage: Use these traits to define request ar 20 ▾
3    put and patch methods
4    traits: 21
5      cacheable: !include ../traits/cacheable.raml
6      hasAcceptHeader: !include ../traits/hasAc 22
7      hasResponseItems: 23 ▾
8        responses: 24
9          200: 25
10            body: 26
11              application/json: 27 ▾
12                type: <<typeName>>[]
13              application/xml: 28
14                type: <<typeName>>[]
15            404: 29 ▾
16              body: 30
17                application/json: 31
18                  type: <<customErrorMessageDataTy
19            406: 30
20              body: 31
21                application/json:
22                  type: <<customErrorMessageDataTy
23    hasPutOrPatchResponseItems: 32
24      responses: 33
25        204:
26        501:
27          body: 34
28            application/json:
29              type: <<customErrorMessageDataTy
```

Create a library file

1. Return to API Designer.
2. From the Project menu, select New Folder.
3. In the Add a new folder dialog box, set the name of the folder to libraries and click OK.

Add a new folder

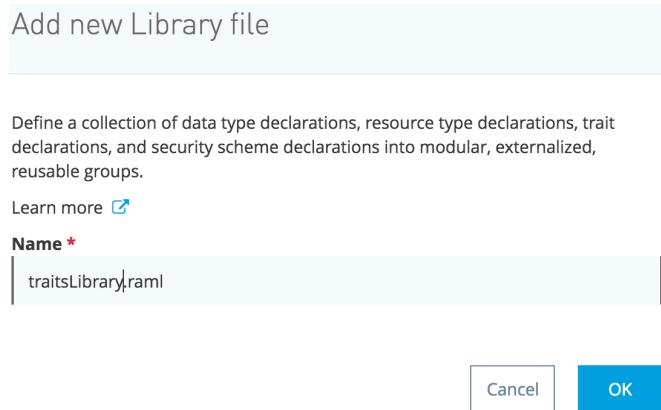
Input a name for your new folder:

Name *

Cancel OK

4. In the file browser, right-click the libraries folder and select New File > RAML 1.0 > Library.

5. In the Add new Library file dialog box, set the name to traitsLibrary.raml and click OK.



6. In the RAML editor, go to the usage node and type the value as: Use these traits to define response structures for get and delete methods.

```
1  %%RAML 1.0 Library
2  usage: Use these traits to define request and response structures for get and delete methods
```

Define traits for response and request structures in the library file

7. Add a new line and in the shelf, click traits.
8. In the new line type hasResponseItems: and press enter.

```
1  %%RAML 1.0 Library
2  usage: Use these traits to define request and response structures
   for get, put and patch methods
3 ▼ traits:
4   | hasResponseItems:
5   | | | |
```

9. In the file browser, select api.raml.

10. Go to the /customers get method, and copy the lines from the responses to the HTTP 406 response application/json type CustomErrorMessage.

```
30 ▼      get:
31        description: Retrieve a list of customers
32        is: [cacheable, hasAcceptHeader]
33 ▼      responses:
34 ▼        200:
35 ▼          body:
36            application/json:
37              type: Customer[]
38            application/xml:
39              type: Customer[]
40 ▼        404:
41 ▼          body:
42            application/json:
43              type: CustomErrorMessage
44 ▼        406:
45 ▼          body:
46            application/json:
47              type: CustomErrorMessage
48 ▼      /{customer_id}:
```

11. In file browser, select traitsLibrary.raml and paste the lines in the new line added to the end of the file.

```
2  usage: Use these traits to define request and response structures
3  for get, put and patch methods
3 ▼  traits:
4 ▼    hasResponseItems:
5 ▼      responses:
6 ▼        200:
7 ▼          body:
8            application/json:
9              type: Customer[]
10             application/xml:
11               type: Customer[]
12 ▼        404:
13 ▼          body:
14            application/json:
15              type: CustomErrorMessage
16 ▼        406:
17 ▼          body:
18            application/json:
19              type: CustomErrorMessage
```

Note: If necessary, correct the indentation in the pasted lines.

12. Go to the first line with the red error that contains the Customer array type.

13. Delete the value Customer[] and type <<resourcePathName | !singularize | !uppercasecase>>[].
14. Replace the type Customer[] in the line under application/xml with << resourcePathName | !singularize | !uppercasecase>>[].

```
hasResponseItems:  
  responses:  
    200:  
      body:  
        application/json:  
          type: <<resourcePathName | !singularize | !uppercasecase>>[]  
        application/xml:  
          type: <<resourcePathName | !singularize | !uppercasecase>>[]
```

15. In the HTTP status 404 and 406 body type, delete the reference to CustomErrorMessage datatype, and type <<customErrorMessageDataType>>.

```
404:  
  body:  
    application/json:  
      type: <<customErrorMessageDataType>>  
406:  
  body:  
    application/json:  
      type: <<customErrorMessageDataType>>
```

16. Add a new line below the HTTP 406 body definition, to define a new trait.
17. In the new line, type:

```
hasDeleteResponseItems:  
  18 ▼  406:  
  19 ▼    body:  
  20      application/json:  
  21      type: <<customErrorMessageDataType>>  
  22 ▼    hasDeleteResponseItems:
```

18. In the file browser, select api.raml.
19. Locate the /{customer_id} nested resource delete method definition and copy the responses section.

20. In the file browser, select traitsLibrary.raml and paste the lines below the hasDeleteResponseItems trait.

```
22 ▼ | hasDeleteResponseItems:  
23 ▼ |   responses:  
24 ▼ |     200:  
25 |       body:  
26 |         application/json:  
27 ▼ |     404:  
28 ▼ |       body:  
29 |         application/json:  
30 |         type: customErrorMessage
```

21. Replace the last line containing the CustomErrorMessage datatype with <><customErrorMessageDataType><>.

Apply the traits in the main RAML API definition

22. In the file browser, select api.raml.
23. Copy the cacheable and hasAcceptHeader reference lines and delete them from api.raml.
24. In the file browser, select traitsLibrary.raml and add the copied lines before the hasResponseItems trait definition.

```
3 ▼ | ---  
3 ▼ | traits:  
4 |   cacheable: !include traits/cacheable.raml  
5 |   hasAcceptHeader: !include traits/hasAcceptHeader.raml  
6 ▼ |   hasResponseItems:
```

Note: There can only be one traits node in the api.raml file. Because the library file requires a traits node, we should move the other traits references over here.

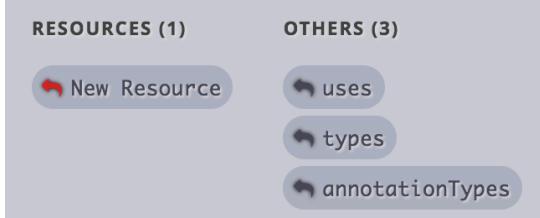
25. To get rid of the error in lines that include the traits fragment, add .. / before the trait's path in the include statement.

```
3 ▼ | ---  
3 ▼ | traits:  
4 |   cacheable: !include ../traits/cacheable.raml  
5 |   hasAcceptHeader: !include ../traits/hasAcceptHeader.raml  
6 ▼ |   hasResponseItems:  
7 ▼ |     responses:  
8 |       200:
```

26. Save the project.
27. In the file browser, select api.raml.

28. Go to the line before the resourceTypes definition and in the shelf, click uses.

```
19
20
21
22 ▼ resourceTypes:
23   collection: !include resourceTypes
24   member: !include resourceTypes/member
25
26 ▼ /customers:
27   type: { collection: { postRequestExample: !include ...
   customErrorMessageType: CustomError
```



29. In the new line type:

Traits: libraries/traitsLibrary.raml.

```
20 ▼ uses:
21   Traits: libraries/traitsLibrary.raml
22
23 ▼ resourceTypes:
24   collection: !include resourceTypes/collection.raml
25   member: !include resourceTypes/member.raml
26
```

30. In the /customers resource get method, go to the line that applies the traits cacheable and hasAcceptHeader and change the value to:

```
is: [Traits.cacheable, Traits.hasAcceptHeader,
Traits.hasResponseItems: { customErrorMessageDataType:
CustomErrorMessage} ]
29 ▼   get:
30     description: Retrieve a list of customers
31     is: [Traits.cacheable, Traits.hasAcceptHeader, Traits.hasResponseItems : {
customErrorMessageDataType: CustomErrorMessage}]
```

31. Delete the responses structure in the get method.

```
27 ▼ /customers:  
28   type: { collection: { postRequestBodyExample: !include examples/CustomerExample.json,  
29     customErrorMessageType: CustomErrorMessage } }  
30   get:  
31     description: Retrieve a list of customers  
32     is: [Traits.cacheable, Traits.hasAcceptHeader, Traits.hasResponseItems : {  
33       customErrorMessageDataType: CustomErrorMessage}]  
34     /{customer_id}:  
35       type: { member: { uriParameterName: customer_id, getResponseBodyExample: !include  
36         examples/CustomerExample.json, customErrorMessageType: CustomErrorMessage } }  
37     patch:
```

32. Similarly, in the /accounts and /transactions resources, delete the get method responses, and change the traits application line to:

```
is: [Traits.hasAcceptHeader, Traits.hasResponseItems: {  
  customErrorMessageDataType: CustomErrorMessage} ]
```

```
52 ▼   /accounts:  
53     get:  
54       description: Retrieve a list of accounts for a specific customer  
55       is: [Traits.hasAcceptHeader, Traits.hasResponseItems: { customErrorMessageDataType:  
56         CustomErrorMessage} ]  
57 ▼   /accounts:  
58     /transactions:  
59       get:  
60         description: Retrieve a list of transactions for a specific account  
61         is: [Traits.hasAcceptHeader, Traits.hasResponseItems: { customErrorMessageDataType:  
62           CustomErrorMessage } ]  
63  
64
```

33. In the /{customer_id} resource delete method, delete the lines that define the method responses.

34. In the new line below the description node, type the following:

```
is: [Traits.hasDeleteResponseItems : {customErrorMessageDataType:  
  CustomErrorMessage}]
```

```
42 ▼   delete:  
43     description: Delete a specific customer  
44     is: [Traits.hasDeleteResponseItems : {customErrorMessageDataType: CustomErrorMessage}]  
45 ▼   /accounts:  
46     get:
```

35. Similarly, in the /{account_id} resource delete method, delete the lines that define the responses and apply the hasDeleteResponseItems trait.

```
52 ▼ |  /{account_id}: ...
53 |    type: { member: { uriParameterName: account_id, getResponseBodyExample: !include
54 ▼ |      examples/AccountExample.json, customErrorMessageType: CustomErrorMessage } }
55 |    delete:
56 |      description: Delete an account with a specific account ID
|      is: [Traits.hasDeleteResponseItems : {customErrorMessageDataType: CustomErrorMessage}]
```

36. Save the project.

View the documentation

37. In API Console, scroll down to the /accounts/{account_id} and click DELETE.

38. Verify that the hasDeleteResponseItems trait has been applied to the resource.

The screenshot shows the Mule API Console interface. At the top, there's a header with 'CLOSE X'. Below it, the endpoint path is shown as '/accounts/{account_id}' with a tooltip 'Type: member'. A callout box indicates the 'Traits' are 'Traits.hasDeleteResponseItems'. Below the path, there are three buttons: 'DELETE' (gray), 'PUT' (purple), and 'GET' (blue). To the right of these buttons is a 'Try it' button with a left arrow icon. On the left side of the main content area, the word 'Request' is visible. The main content area is titled 'DESCRIPTION' and contains the text 'Delete an account with a specific account ID'.

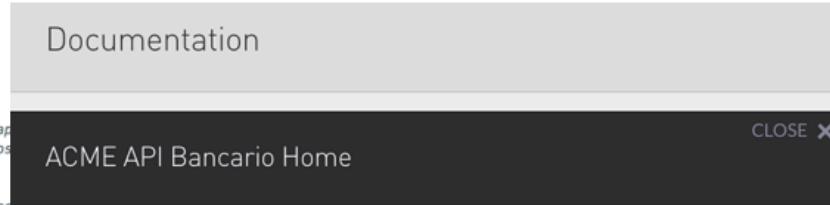
39. Click CLOSE X.

Walkthrough 10-2: Internationalize documentation and resource description using an overlay

In this walkthrough, you internationalize the ACME Banking API using overlays. You will:

- Create a RAML overlay fragment file.
- Add documentation and resource method description nodes in the overlay file to support Spanish localization.

```
1  #RAML 1.0 Overlay
2  usage: Spanish localization
3  extends: ../api.raml
4  documentation:
5    - title: ACME API Bancario Home
6    - content: |
7      **ACME Banca API** permite a los desarrolladores crear aplicaciones que hacen uso de la información del recurso a los métodos implementados en la API.
8
9      Esta API contiene una funcionalidad que permite a los desarrolladores recuperar y manipular el _cliente_, la _cuenta_ y la información de la _transacción_. Echa un vistazo a la API [Portal]() para más detalles.
10   - title: Banco ACME Headline
11   - content: |
12     **Banco ACME** es una multinacional de servicios financieros de la organización.
13
14  /customers:
15    get:
16      description: Recuperar una lista de clientes
17      post:
```

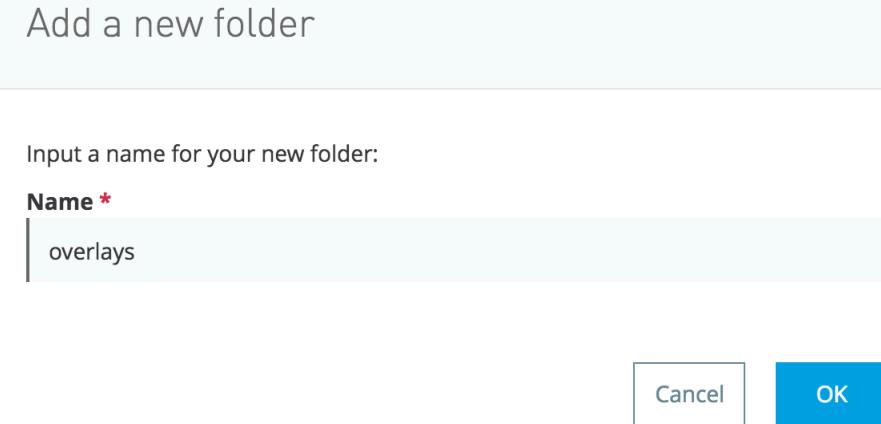


ACME Banca API permite a los desarrolladores crear aplicaciones que hacen uso de la información del recurso a los métodos implementados en la API.

Esta API contiene una funcionalidad que permite a los desarrolladores recuperar y manipular el `cliente`, la `cuenta` y la información de la `transacción`. Echa un vistazo a la API `Portal` para más detalles.

Create a RAML overlay file

1. Return to API Designer.
2. From the Project menu, select New Folder.
3. In the Add a new folder dialog box, set the name to overlays and click OK.



4. In the file browser, right-click the overlays folder and select New File > RAML 1.0 > Overlay.
5. In the Add a new overlay file dialog box, set the name to internationalization.raml and click OK.

Add documentation and description nodes to support Spanish localization

6. In the second line that contains the extends node, type the value as

```
..../api.raml
```

```
1  #%RAML 1.0 Overlay  
⚠+ 2  extends: ../api.raml
```

7. Enter a new line before the extends node and type:

usage: Spanish localization.

8. Return to the course snippets.txt file.
9. Copy the text from the Module 10, Spanish documentation snippet and paste it in internationalization.raml.

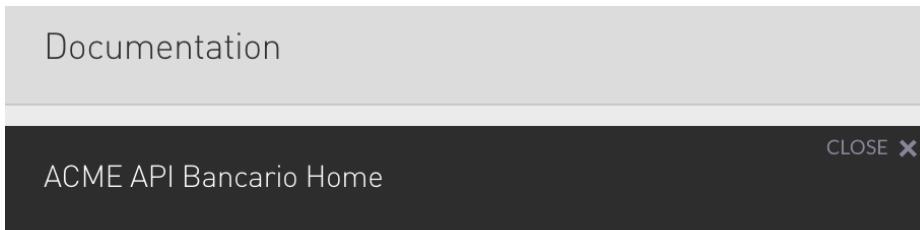
```
1  #%RAML 1.0 Overlay  
2  usage: Spanish localization  
⚠+ 3  extends: ../api.raml  
4  documentation:  
5  └─ title: ACME API Bancario Home  
6  └─ content: |  
7    | **ACME Banca API** permite a los desarrolladores crear  
8    | aplicaciones que hacen uso de la información del recurso a  
9    | los métodos implementados en la API.  
10 ─ content: |  
11   | Esta API contiene una funcionalidad que permite a los  
12   | desarrolladores recuperar y manipular el _cliente_, la  
13   | _cuenta_ y la información de la _transacción_. Echa un  
14   | vistazo a la API [Portal]() para más detalles.  
15 ─ content: |  
16   | **Banco ACME** es una _multinacional de servicios bancarios_  
17   | y financieros de la organización.
```

10. From the course snippets.txt file, copy the lines under the Spanish localization for method description heading.
11. Return to API Designer and enter a new line below the documentation node and place the cursor at the beginning of the line.
12. Add a new line and paste the copied text.

```
11 ─ content: |  
12   | **Banco ACME** es una _multinacional de servicios bancarios_  
13   | y financieros de la organización.  
14 ─ /customers:  
15   | get:  
16   |   description: Recuperar una lista de clientes  
17   | post:  
18   |   description: Agregue un nuevo cliente  
19 ─ /{customer_id}:  
20   | get:  
21   |   description: Recuperar un cliente con una ID de cliente  
22   |   específico  
23   | delete:  
24   |   description: Eliminar un cliente específico
```

View the documentation

13. In API Console, verify that the documentation, resources, and types show up like the api.raml API Console.
14. Go to the Documentation section and expand the documentation title to view the Spanish localization of the document.



ACME Banca API permite a los desarrolladores crear aplicaciones que hacen uso de la información del recurso a los métodos implementados en la API.

Esta API contiene una funcionalidad que permite a los desarrolladores recuperar y manipular el *cliente*, la *cuenta* y la información de la *transacción*. Echa un vistazo a la API [Portal](#) para más detalles.

15. Click CLOSE X.
16. Go to the /customers resource and click GET.
17. Verify you can view the GET method description in Spanish.

A screenshot of the API Console's Resources section. It shows a list of resources, with "/customers" being selected. The resource details show it is a "Type: collection" with traits: Traits.cacheable, Traits.hasAcceptHeader, Traits.hasResponseItems. Below this, there are "POST" and "GET" buttons. A "Request" button is on the left and a "Try it" button is on the right.

DESCRIPTION

Recuperar una lista de clientes

18. Click CLOSE X.
19. Save the project.

Walkthrough 10-3: Define and use API extensions to promote portability to test in multiple environments

In this walkthrough, you will define an extension. You will:

- Create API extension files.
- Add a baseUri parameter to the extension files and enter a placeholder for environment specific implementation URL.



The screenshot shows the MuleSoft API Designer interface. The left sidebar displays the project structure with a 'extensions' folder containing 'stagingExtension.raml'. The main editor window shows the following RAML code:

```
#%RAML 1.0 Extension
baseUri: https://mocksvc.mulesoft.com/mocks/c7eb6e39-a79f-435b-9488-14f45b3276f4<<insert staging link here>>
extends: ../api.raml
```

The right panel shows documentation sections: 'Documentation', 'ACME Bank Headline', 'ACME Banking API Home', and 'Types'.

Create API extension files

1. Return to API Designer.
2. In the file browser, create a new folder.
3. In the Add a new folder dialog box, set the name to extensions and click OK.

Add a new folder

Input a name for your new folder:

Name *

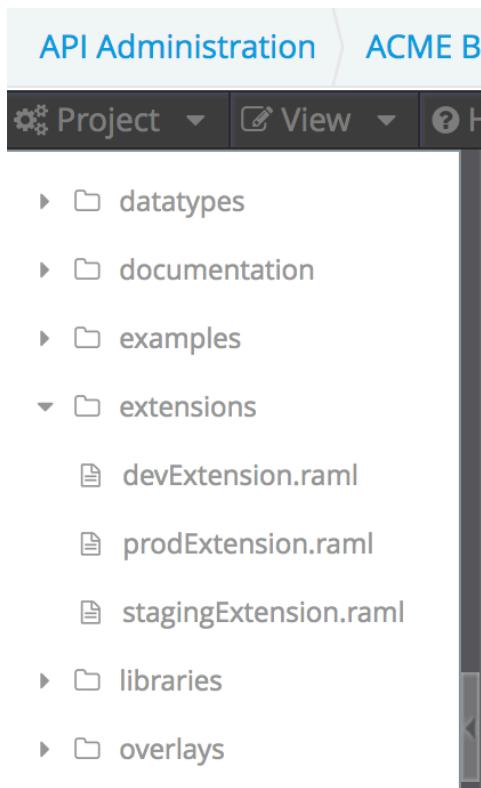
extensions

Cancel

OK

4. Right-click the extensions folder and select New File > RAML 1.0 > Extension.
5. In the Add new Extension file dialog box, set the name to stagingExtension.raml and click OK.

6. Create two more extensions in the extensions folder called devExtension.raml and prodExtension.raml.



Add the baseUri node to the extension files and enter a placeholder for environment specific implementation URL

7. In the file browser, select api.raml.
8. Go to the line that defines the baseUri at the top and delete it.

Note: The baseUri parameter cannot be overridden if declared in the root RAML file. We will be declaring it in the individual extension files.

9. Save the project.
10. In the file browser, select stagingExtension.raml.
11. Go to the line that contains the extends node and type the value as:

```
./api.raml
```

```
* /extensions/stagingExtension.raml (2 warnings)
```

1	#%RAML 1.0 Extension
⚠+ 2	extends: ./api.raml

12. In API Console, turn on the mocking service at the top right corner.

13. Verify that you see the baseUri added to the extension file.



```
#%RAML 1.0 Extension
baseUri: https://mocksvc.mulesoft.com/mocks/c7eb6e39-a79f-435b-9488-14f45b3276f4<<insert staging link here>>
extends: ../api.raml
```

Note: You can replace the Mock service URL with a staging environment specific implementation URL.

14. Save the project.

15. In API Console, click GET for the /accounts/{account_id} resource.

16. Click Try It.

17. Enter the account_id value as 12345 and click GET.

18. Verify that you see the response structure of the get method.

```
content-type:
application/json; charset=utf-8
date:
Wed, 19 Apr 2017 18:35:58 GMT
server:
nginx
vary:
Accept-Encoding
```

Body

```
1 {
2   "accountID": "12345",
3   "accountType": "Savings",
4   "accountNumber": "1234567890",
5   "accountOwner": [
6     {
7       "customerID": "8f19cb50-3f57-4d38",
8       "displayName": "John Doe",
9       "ssn": "123-456-7890"
10      }
11    ],
12   "accountBalance": {
13     "currency": "USD",
14     "amount": "8457.90"
```

19. Similarly, go to the devExtension.raml file and include the api.raml file in the extends node.

20. Add a new line before the extends node, and type the following:

```
baseUri: <<insert dev environment specific implementation URL>>
```

```
* /extensions/devExtension.raml (2 warnings)
```

```
  1  #%%RAML 1.0 Extension  
  2  baseUri: <<insert dev| environment specific implementation URL>>  
⚠+ 3  extends: ../api.raml
```

21. Save the project.

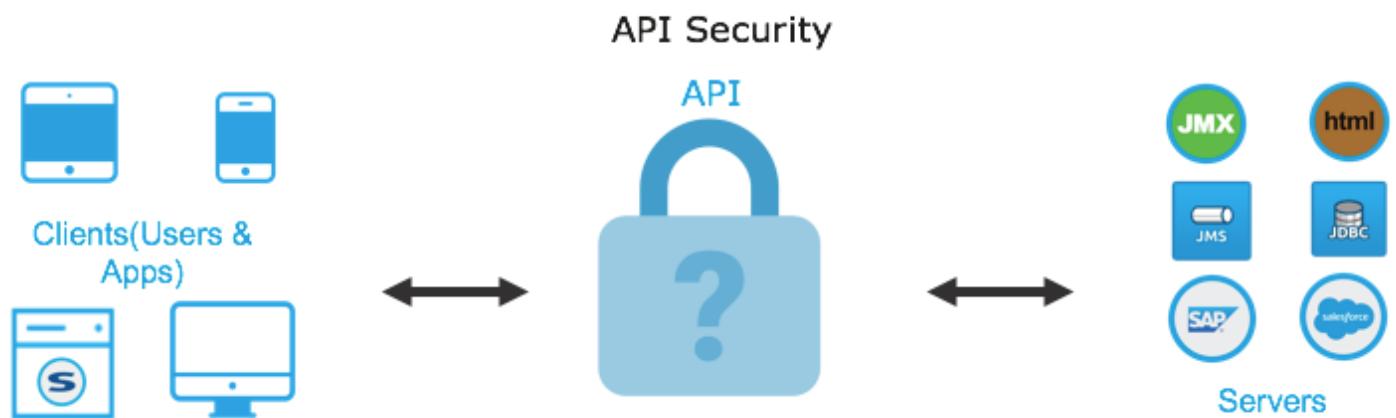
22. Similarly, in the prodExtension.raml file, add in a placeholder for the baseUri and reference api.raml in the extends node.

```
* /extensions/prodExtension.raml (2 warnings)
```

```
  1  #%%RAML 1.0 Extension  
  2  baseUri: <<insert prod environment specific implementation URL>>  
⚠+ 3  extends: ../api.raml
```

23. Save the project.

Module 11: Securing APIs



Objectives:

- Define API security requirements.
- Implement basic authentication for a resource URL using traits.
- Use RAML to define an API-to-OAuth2.0 policy.

Walkthrough 11-1: Define a custom security scheme for an API

In this walkthrough, you will define a custom security scheme for your API. You will:

- Create a custom security scheme file.
- Reference the custom security scheme in the main RAML API definition.
- Apply the security scheme to certain resource methods.

```
1  #%RAML 1.0 SecurityScheme
2  type: x-customToken
3  description: This security scheme validates a requ
token provided that should be provided in the head
4 ▼ describedBy:
5 ▼   headers:
6 ▼     Authorization:
7       description: This header should contain the
8       type: string

23 ▼   securitySchemes:
24     customTokenSecurity: !include
securitySchemes/customTokenSecurity.raml
25
26 ▼   resourceTypes:
```

SECURITY SCHEMES

x-customToken

This security scheme validates a request to the API using a token provided that should be provided in the header with the request

HEADERS

Authorization *required* string

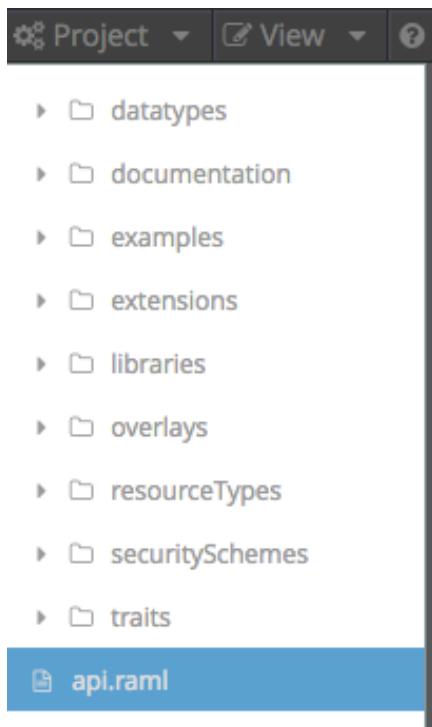
This header should contain the security token

Response

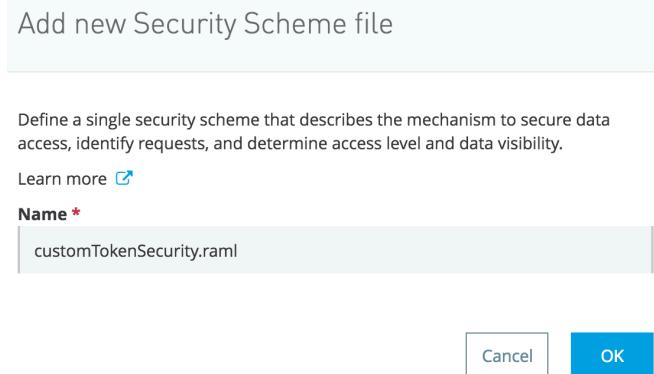
Create a security scheme file

1. Return to API Designer.

2. In the file browser, create a new folder called securitySchemes.



3. Right-click the securitySchemes folder and select New File > RAML 1.0 > Security Scheme.
4. In the Add new Security Scheme file dialog box, set the name to customTokenSecurity.raml.



Define a custom security scheme

5. In the RAML editor, go to the line that contains type node and press space bar after the colon.

6. In the shelf below the editor, click x-{other}.

The screenshot shows a code editor with two lines of RAML code:

```
1 %%RAML 1.0 SecurityScheme
2 type:
```

Below the editor is a shelf with two categories: "SECURITY (4)" and "OTHERS (3)". The "OTHERS" category contains four items: "Define Inline", "Pass Through", "x-{other}" (which is highlighted), and "Digest Authentication".

7. Replace {other} with customToken.

The screenshot shows the RAML code with the {other} placeholder replaced by "customToken".

```
* /securitySchemes/customTokenSecurity.raml
1 %%RAML 1.0 SecurityScheme
2 type: x-customToken
```

8. Add a new line below the type node.
9. In the shelf, click description.
10. Type the value of the description node as:

This security scheme validates the request to the API using a token provided in the header with the request.

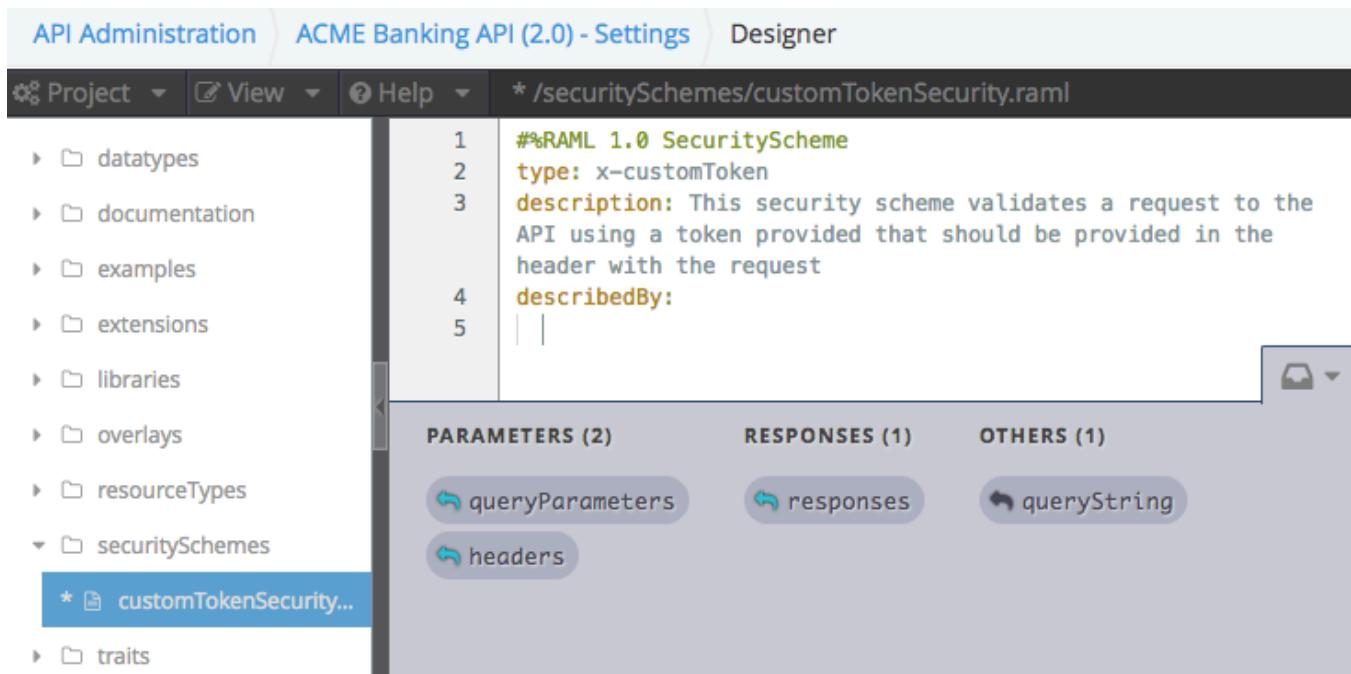
11. Press enter to add a new line.

The screenshot shows the RAML code with a new line added after the type node, containing the description text.

```
1 %%RAML 1.0 SecurityScheme
2 type: x-customToken
3 description: This security scheme validates a request to the API
   using a token provided that should be provided in the header with
   the request
4
```

12. In the shelf, click describedBy.

13. In the shelf, click headers.



The screenshot shows the API Administration interface with the title "ACME Banking API (2.0) - Settings" and the path "/securitySchemes/customTokenSecurity.raml". The left sidebar contains a navigation tree with items like "datatypes", "documentation", "examples", "extensions", "libraries", "overlays", "resourceTypes", "securitySchemes", and "traits". The "securitySchemes" item has a sub-item "customTokenSecurity..." which is currently selected. The main pane displays the Raml code:

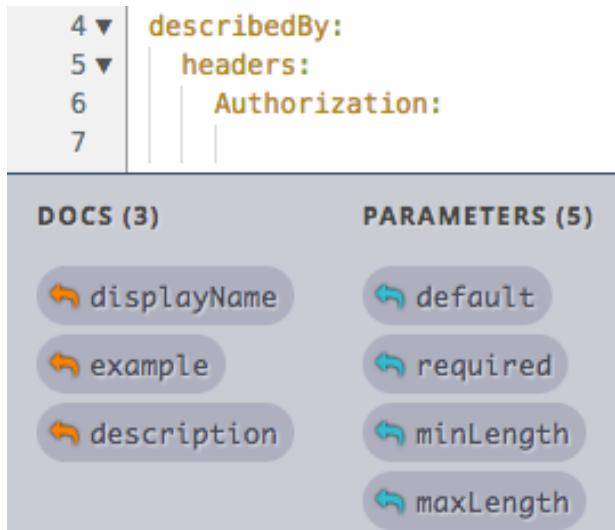
```
1  #%RAML 1.0 SecurityScheme
2  type: x-customToken
3  description: This security scheme validates a request to the
4  header with the request
5  | |

```

Below the code, there are three tabs: "PARAMETERS (2)", "RESPONSES (1)", and "OTHERS (1)". Under "PARAMETERS (2)", there are two items: "queryParameters" and "headers". The "headers" item is highlighted with a blue background.

14. In the shelf, click Authorization.

15. In the shelf, click description.



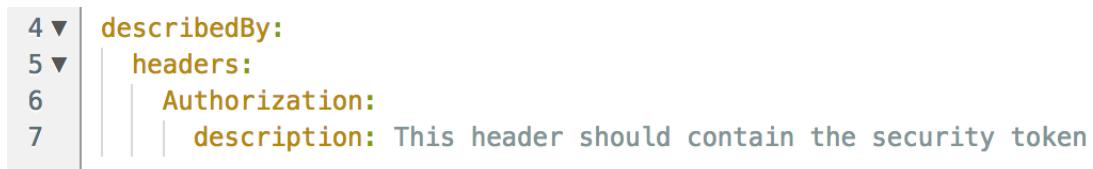
The screenshot shows the Raml code with the "headers" node expanded:

```
4  describedBy:
5  |   headers:
6  |   |   Authorization:
7  |   |   |

```

Below the code, there are two sections: "DOCS (3)" and "PARAMETERS (5)". The "DOCS (3)" section contains "displayName", "example", and "description". The "PARAMETERS (5)" section contains "default", "required", "minLength", and "maxLength". The "description" node from the Raml code is also listed under "PARAMETERS (5)".

16. Type the value of the description node as: This header should contain the security token.



The screenshot shows the Raml code with the "description" node value changed:

```
4  describedBy:
5  |   headers:
6  |   |   Authorization:
7  |   |   |   description: This header should contain the security token

```

17. Press enter to add a new line.

18. In the shelf, click type.

19. In the shelf, click string.



20. Save the project.

Reference the custom security scheme file in the main API definition

21. In the file browser, select api.raml.

22. Go to the empty line after the line that references the traitsLibrary file.

23. Add a new line below it.

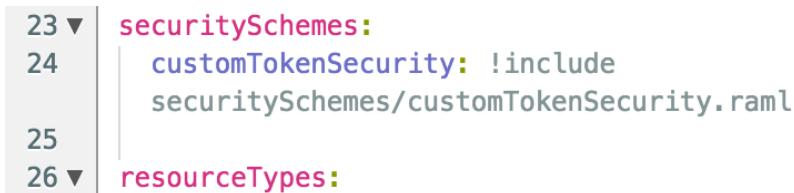
24. In the shelf, click securitySchemes.



25. In the new line type:

```
customTokenSecurity: !include securitySchemes/customTokenSecurity.raml
```

26. Press enter to add a new line.



Apply the custom security scheme to all the resource methods in the API

27. Go to the empty line before the /customers resource and press enter.
28. In the shelf, click securedBy.

```
29
30
31 ▼ /customers:
32   type: { collection: { postRequestBodyDatatype: Customer,
33     postRequestBodyExample: !include examples/CustomerExample.raml,
34     customErrorMessageType: CustomErrorMessage } }
35   get:
36     description: Retrieve a list of customers
37     is: [Traits.cacheable, Traits.hasAcceptHeader,
38       Traits.hasResponseItems : {typeName: Customer,
39         customErrorMessageDataType: CustomErrorMessage }]
36 ▼ /{customer_id}:
37   type: { member: { uriParameterName: customer_id,
38     getResponseBodyDataType: Customer, getResponseBodyExample: !include examples/Customer.raml }
```

mediaType documentation

SECURITY (2)

securitySchemes securedBy

29. In the shelf, click customTokenSecurity.

30. Press enter to add a new line.

```
26 ▼ resourceTypes:
27   collection: !include resourceTypes/collection.raml
28   member: !include resourceTypes/member.raml
29
30   securedBy: customTokenSecurity
31
32 ▼ /customers:
33   type: { collection: { postRequestBodyDatatype: Customer,
```

31. Save the project.

View the documentation

32. In API Console, click GET for the /customers resource.

33. Scroll down to the Headers section of the Request and verify that you can see the security scheme information below.

SECURITY SCHEMES

x-customToken

This security scheme validates a request to the API using a token provided that should be provided in the header with the request

HEADERS

Authorization *required string*

This header should contain the security token

Response

Note: Go to any other resource method and notice the security scheme header added to all the resource methods by adding to the root of the RAML definition. Custom security schemes like the customTokenSecurity is not supported for testing using the Try It option.

34. Click CLOSE X.

Walkthrough 11-2: Define a security scheme for an API and secure API resources

In this walkthrough, you define a security scheme to secure API resources using OAuth2.0. You will:

- Create a security scheme fragment file.
- View the OAuth2.0 protocol security scheme snippet from the policy details.
- Define the OAuth2.0 protocol in the security scheme fragment.
- Reference the OAuth2.0 security scheme in the RAML API definition.
- Apply the security scheme in the API resource methods.

```
securitySchemes:  
  - oauth_2_0:  
      description: |  
        Dropbox supports OAuth 2.0 for authenticating all API requests.  
      type: OAuth 2.0  
      describedBy:  
        headers:  
          Authorization:  
            description: |  
              Used to send a valid OAuth 2 access token. Do not use  
              with the "access_token" query string parameter.  
            type: string  
        queryParameters:  
          access_token:  
            description: |  
              Used to send a valid OAuth 2 access token. Do not use together with  
              the "Authorization" header  
            type: string  
      responses:  
        401:  
          description: |  
            Bad or expired token. This can happen if the user or Dropbox  
            revoked or expired an access token. To fix, you should re-  
            authenticate the user.  
        403:  
          description: |  
            Bad OAuth request (wrong consumer key, bad nonce, expired  
            timestamp...). Unfortunately, re-authenticating the user won't help  
      settings:  
        authorizationUri: https://www.dropbox.com/1/oauth2/authorize  
        accessTokenUri: https://api.dropbox.com/1/oauth2/token  
        authorizationGrants: [ code, token ]
```

AUTHENTICATION

Security Scheme
Custom Security Schemes are not supported in Try It

OAuth 2.0

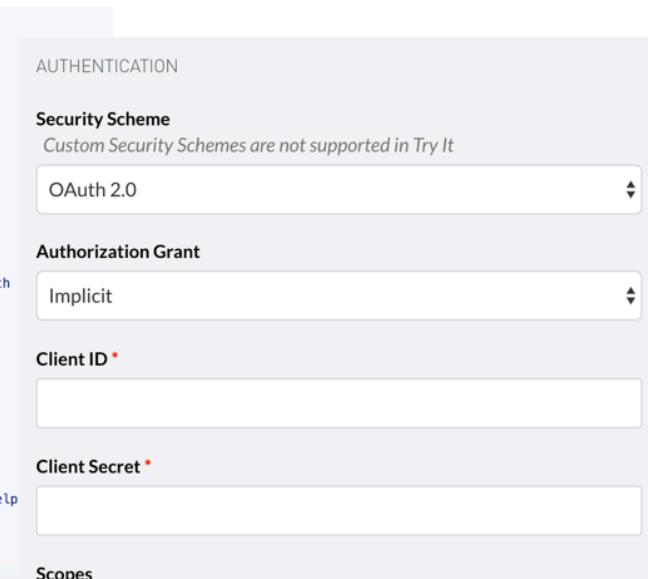
Authorization Grant

Implicit

Client ID*

Client Secret*

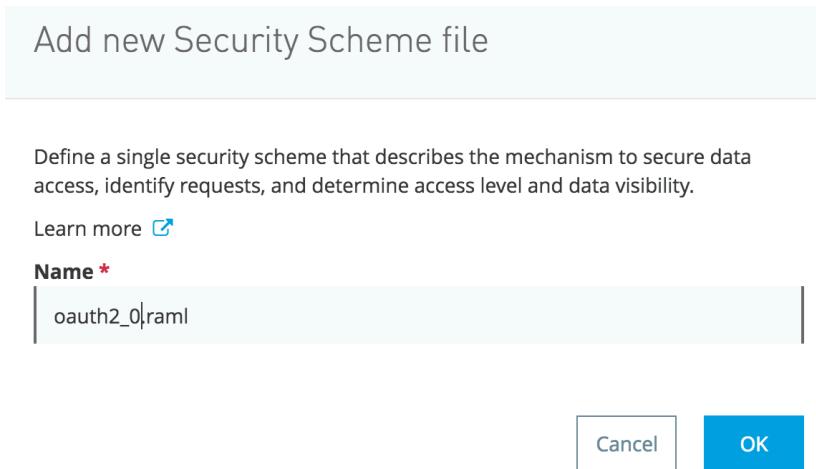
Scopes



Create a security scheme fragment file

1. In the file browser, right-click the securitySchemes folder and select New File > RAML 1.0 > Security Scheme.

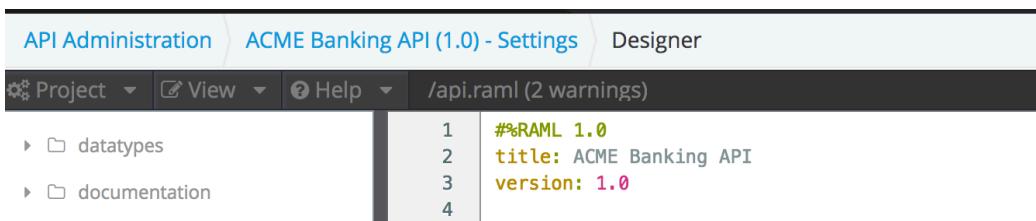
2. In the Add new Security Scheme file dialog box, set the name to oauth2_0.raml.



3. Save the project.

View the security scheme snippet from the OAuth2.0 policy details

4. Click the ACME Banking API (1.0) – Settings link above the API Designer menu.



5. On the API version details page, select Policies in left-side navigation; you should not see a list of policies.

Note: To see the list of policies, you need to set the API URL.

The screenshot shows the "ACME Banking API (1.0) - Settings" page in the API Administration interface. The left sidebar has links for "API Administration", "Applications", "Permissions", "Policies", "SLA Tiers", and "Settings". The main area shows the API title "ACME Banking API" and status "Unregistered". A note says "Set the API URL...".

6. in the left-side navigation, select Settings.

7. On the API version details page, click Configure endpoint in the API Status section.
8. In the Configure endpoint dialog box, select the Basic endpoint radio button.



Use a **basic endpoint** if you are implementing your API in the MuleSoft API Gateway. Otherwise, choose an **endpoint with a proxy** for an existing API.

- Basic endpoint
 Endpoint with a proxy

Endpoint settings

Type

9. In the Endpoint settings, select the Type as HTTP URL and type the Implementation URI as <http://localhost:8081/>.



Use a **basic endpoint** if you are implementing your API in the MuleSoft API Gateway. Otherwise, choose an **endpoint with a proxy** for an existing API.

- Basic endpoint
 Endpoint with a proxy

Endpoint settings

Type

HTTP URL



Implementation URI



[Get from RAML](#)

[Advanced Options >](#)



Note: This step is carried out to set the API URL to view the list of policies. We are not deploying any applications with these values. To understand more about implementing your APIs or proxies, please take the Anypoint Platform Operations: API Management course.

10. Click Save.
11. In the left-side navigation, select Policies.

12. Click Apply New Policy.

The screenshot shows the ACME Banking API dashboard. At the top, it displays "ACME Banking API" and "1.0". Below that, it says "API Status: Unregistered". On the right, there are "Export" and "More" buttons. A prominent blue button labeled "Apply New Policy" is visible. Below it, a message box states "There are no applied policies."

13. In the Select Policy dialog box, click the All Categories drop-down bar on top and click Security.
14. View the list of security policies.
15. Scroll down and select the radio button for OAuth 2.0 access token enforcement using external provider policy and click Configure Policy.
16. In the Apply OAuth 2.0 access token enforcement using external provider policy, click the link text 'here' to obtain the RAML snippet.

Apply OAuth 2.0 access token enforcement using external provider policy

Enforces use of an OAuth 2.0 access token issued through an OAuth 2.0 external provider.

This policy will require updates to the RAML definition in order to function. You can obtain the RAML snippet and learn more [here](#).

Scopes

A space-separated list of supported scopes

Access Token validation endpoint url *

The url of the Access Token validation endpoint of the External OAuth2 Provider.

Cancel Apply

17. Review the code snippet with the OAuth 2.0 security scheme definition.

```
securitySchemes:
  - oauth_2_0:
      description: |
        Dropbox supports OAuth 2.0 for authenticating all API requests.
      type: OAuth 2.0
      describedBy:
        headers:
          Authorization:
            description: |
              Used to send a valid OAuth 2 access token. Do not use with the "access_token" query string parameter.
            type: string
        queryParameters:
          access_token:
            description: |
              Used to send a valid OAuth 2 access token. Do not use together with the "Authorization" header
            type: string
        responses:
          401:
            description: |
              Bad or expired token. This can happen if the user or Dropbox revoked or expired an access token. To fix, you should re-authenticate the user.
          403:
            description: |
              Bad OAuth request (wrong consumer key, bad nonce, expired timestamp...). Unfortunately, re-authenticating the user won't help here.
      settings:
        authorizationUri: https://www.dropbox.com/1/oauth2/authorize
        accessTokenUri: https://api.dropbox.com/1/oauth2/token
        authorizationGrants: [ code, token ]
```

Add the OAuth2.0 security scheme definition

18. Return to the course snippets.txt file.

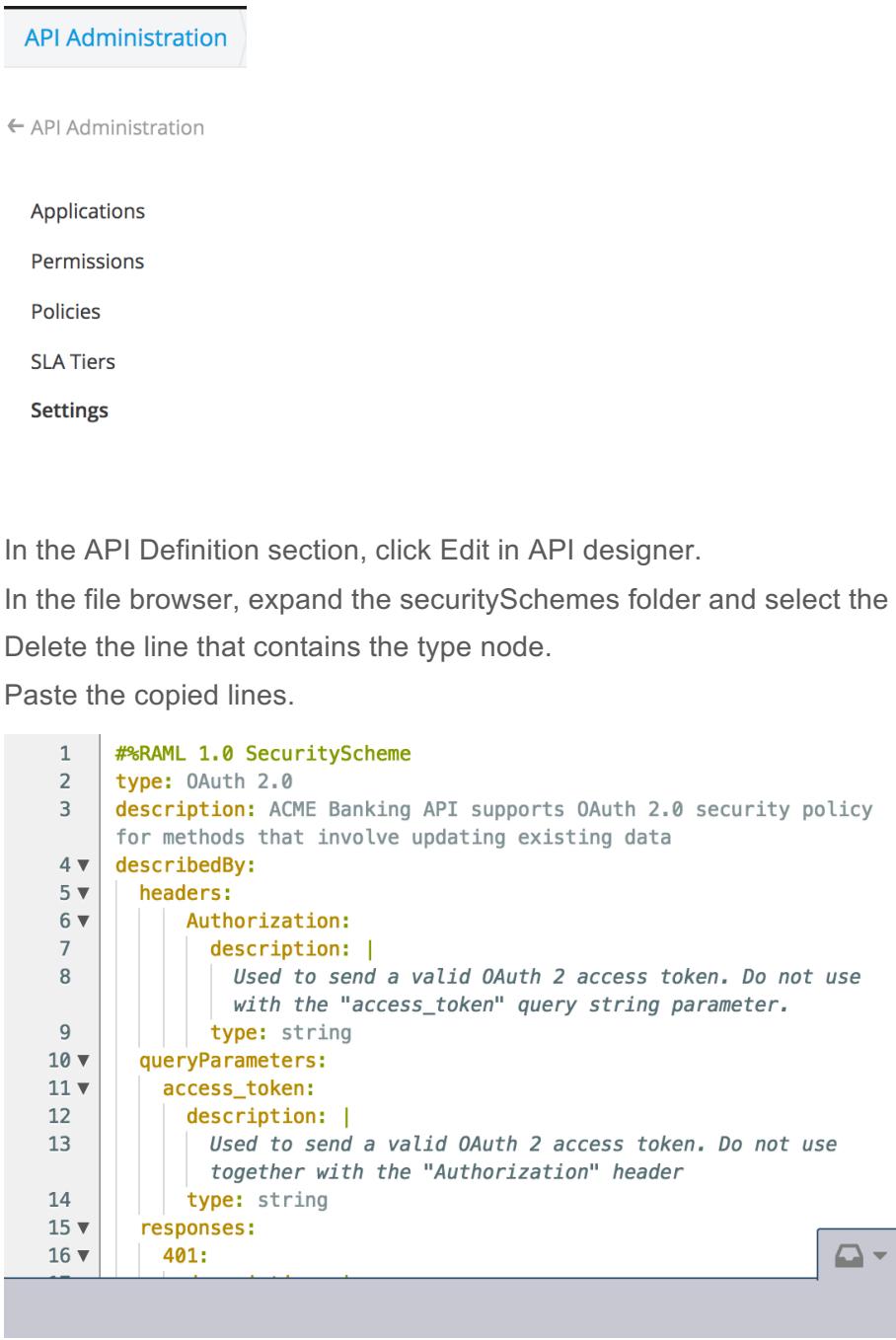
19. Copy the lines in Module 11 under the OAuth2.0 security scheme definition.

```
-----Module 11-----
OAuth 2.0 security scheme definition:

type: OAuth 2.0
description: ACME Banking API supports OAuth 2.0 security policy for methods that involve updating existing data
describedBy:
  headers:
    Authorization:
      description: |
        Used to send a valid OAuth 2 access token. Do not use with the "access_token" query string parameter.
      type: string
  queryParameters:
    access_token:
      description: |
        Used to send a valid OAuth 2 access token. Do not use together with the "Authorization" header
      type: string
  responses:
    401:
      description: |
        Bad or expired token. This can happen if the API consumer uses a revoked or expired access token. To fix, you should re-authenticate the user.
    403:
      description: |
        Bad OAuth request (wrong consumer key, bad nonce, expired timestamp...). Unfortunately, re-authenticating the user won't help here.
settings:
  authorizationUri: https://placeholder.com/oauth2/authorize
  accessTokenUri: https://placeholder.com/oauth2/access_token
  authorizationGrants: implicit
```

20. Return to Anypoint Platform.

21. Click Cancel to cancel applying the OAuth2.0 policy.
22. In the left-side navigation, select Settings to return to the API version details page.



The screenshot shows the 'API Administration' interface. On the left, there is a navigation sidebar with the following items:

- API Administration** (selected)
- Applications
- Permissions
- Policies
- SLA Tiers
- Settings** (selected)

Below the sidebar, there is a large, empty gray area which is likely the main content area of the API designer.

23. In the API Definition section, click Edit in API designer.
24. In the file browser, expand the securitySchemes folder and select the oauth2_0.raml file.
25. Delete the line that contains the type node.
26. Paste the copied lines.

```

1  %%RAML 1.0 SecurityScheme
2  type: OAuth 2.0
3  description: ACME Banking API supports OAuth 2.0 security policy
   for methods that involve updating existing data
4 ▼ describedBy:
5 ▼   headers:
6 ▼     Authorization:
7     description: |
      | Used to send a valid OAuth 2 access token. Do not use
      | with the "access_token" query string parameter.
9     type: string
10 ▼  queryParameters:
11 ▼    access_token:
12    description: |
13    | Used to send a valid OAuth 2 access token. Do not use
14    | together with the "Authorization" header
15    type: string
16 ▼  responses:
17    401:

```

Note: The authorizationUri, accessTokenUri and the authorizationGrants value should be the URI in the external OAuth2.0 provider that you configure. For now, we use placeholder values.

27. Save the project.

Reference the OAuth2.0 security scheme in the main API definition

28. In the left-side navigation, select api.raml.
29. Go to the end of the line that contains the customTokenSecurity scheme and add a new line below it.

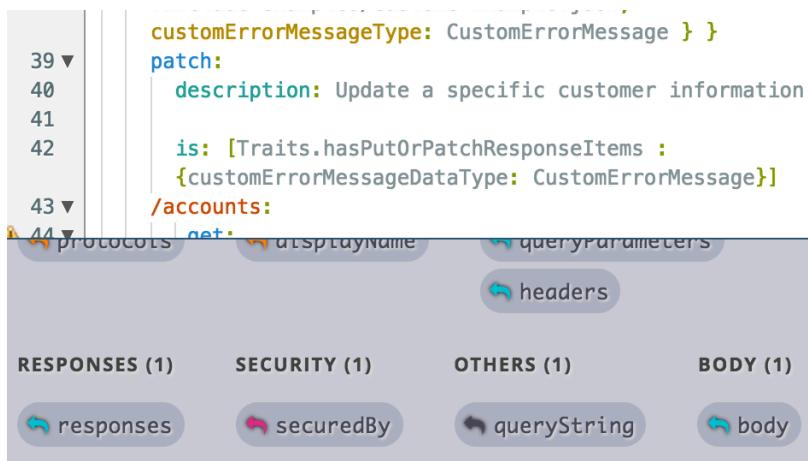
```
22 ▼ | securitySchemes:  
23 |   customTokenSecurity: !include  
|     securitySchemes/customTokenSecurity.raml  
24 |  
25 |  
26 ▼ | resourceTypes:
```

30. In the new line type:

```
oauth2_0: !include securitySchemes/oauth2_0.raml.  
22 ▼ | securitySchemes:  
23 |   customTokenSecurity: !include  
|     securitySchemes/customTokenSecurity.raml  
24 |   oauth2_0: !include securitySchemes/oauth2_0.raml  
25 |  
26 ▼ | resourceTypes:
```

Secure the resource methods that update bank customer and account information with the OAuth2.0 security scheme

31. In the /{customer_id} nested resource patch method, add a new line below the line that contains the description node.
32. In the shelf, click securedBy.



33. In the shelf, click oauth2_0.

34. In the /{account_id} nested resource put method, add a new line below the line that contains the description node.
35. In the shelf, click securedBy.
36. In the shelf, click oauth2_0.

```
52 ▼ |  put:  
53 |   description: Update a specific account information  
54 |   securedBy:  
55 |     is: [Traits.hasPutOrPatchResponseItems :  
56 |       {customErrorMessageDataType: CustomErrorMessage}]  
57 |     body:  
58 |       application/json:  
      |         type: Account
```

OTHERS (3)

- customTokenSecurity
- oauth2_0
- null

37. Save the project.
38. In the file browser, expand the extensions folder and click stagingExtension.raml file.

Note: Because the mocking service is turned on for the extension RAML file, the baseUri gets added in the RAML editor multiple times. This can indicate an error in the API Console. To avoid the error, delete duplicate baseUri nodes that get created.

39. In API Console, click the PUT tab for the /accounts/{account_id} resource.
40. Scroll to the Security Schemes header under the Request section.

41. Review the details for the OAuth2.0 security scheme.

SECURITY SCHEMES

OAuth 2.0

ACME Banking API supports OAuth 2.0 security policy for methods that involve:

HEADERS

Authorization *required string*

Used to send a valid OAuth 2 access token. Do not use with the `access_token` parameter.

QUERY PARAMETERS

access_token *required string*

Used to send a valid OAuth 2 access token. Do not use together with the Authorization header

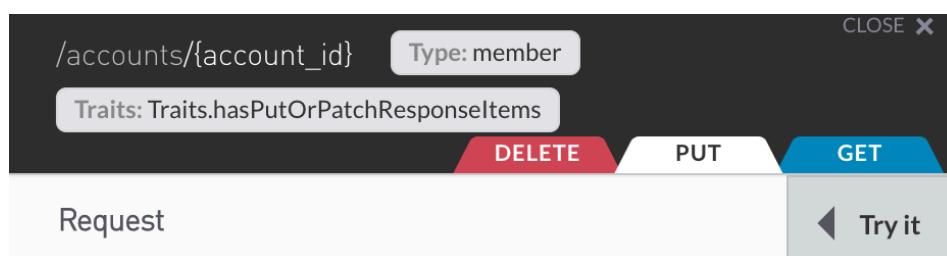
RESPONSES

401

Bad or expired token. This can happen if the API consumer

Note: Even though we applied the customToken security scheme at the root level, observe that resource method level security takes precedence when both are specified.

42. Scroll back up and click Try it across the Request section header.



43. Locate the required input Client ID and Client Secret parameters for the OAuth2.0 Authentication.

AUTHENTICATION

Security Scheme
Custom Security Schemes are not supported in Try It

OAuth 2.0

Authorization Grant

Implicit

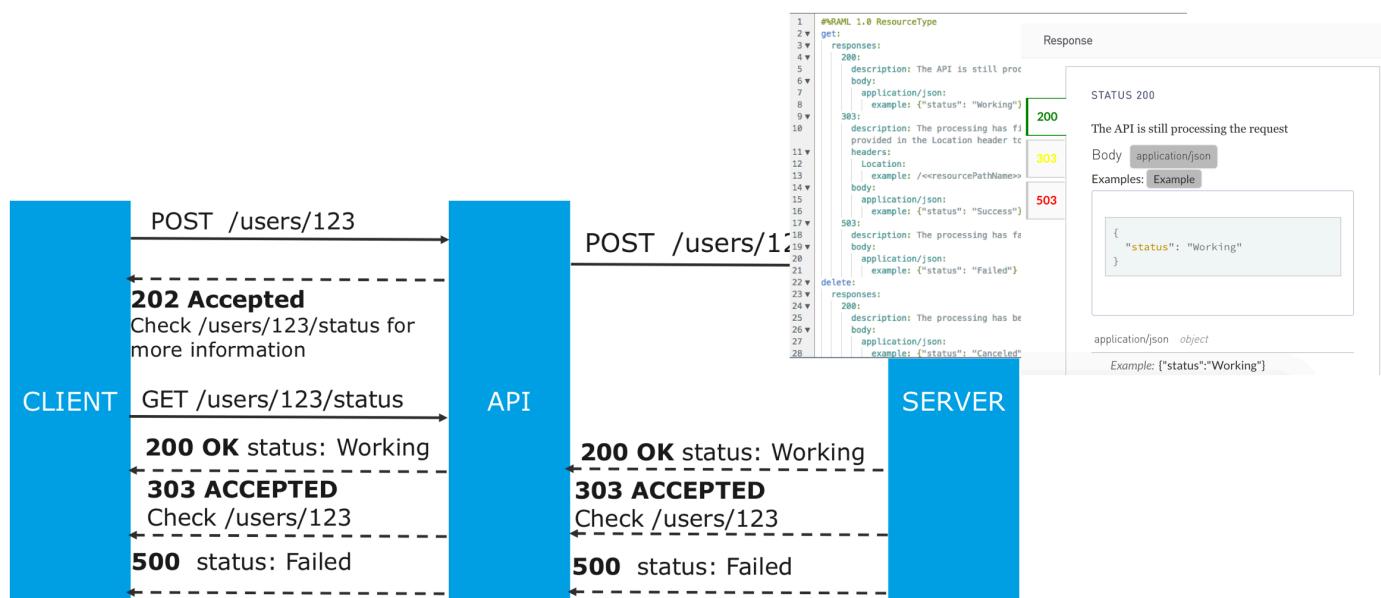
Client ID *

Client Secret *

Scopes

Note: We carried out this step using the extension file because a baseUri is required to use the Try It option in API Console.

Module 12: Enhancing API Responses using Hypermedia



Objectives:

- Describe hypermedia.
- Simplify API discoverability using hypermedia.
- Use hypermedia to enhance API responses.
- Modify an API definition to generate state-specific client responses.

Walkthrough 12-1: Modify an API definition to generate state-specific client responses

In this walkthrough, you add state specific responses for methods that add data into the API implementation and go through multiple stages of processing information. You will:

- Create an HTTP 202 post body response with necessary headers.
- Add a new resource to track the state of post requests that are under processing.
- Add necessary HTTP methods with responses to give meaningful information back to the API consumers.

The screenshot shows a RAML 1.0 API definition on the left and its corresponding generated response details on the right. The API definition includes a 'get' method with three response status codes: 200, 303, and 503. The 200 response has a description and an example JSON object. The 303 response has a description and an example JSON object. The 503 response has a description and an example JSON object. The generated response details on the right show the STATUS 200 response with the description 'The API is still processing the request'. It also shows the Body as application/json and Examples with the JSON object: { "status": "Working" }. The 303 and 503 responses are also listed with their descriptions and examples.

```
1  #%RAML 1.0 ResourceType
2  ▼ get:
3  ▼   responses:
4  ▼     200:
5     |   description: The API is still processing the request
6     |   body:
7     |     application/json:
8     |       example: {"status": "Working"}
9  ▼     303:
10    |   description: The processing has finished successfully
11    |   provided in the Location header to obtain the actual
12    |   headers:
13    |     Location:
14    |       example: /<>resourcePathName<>/1234
15    |   body:
16    |     application/json:
17    |       example: {"status": "Success"}
18  ▼     503:
19    |   description: The processing has failed
20    |   body:
21    |     application/json:
22    |       example: {"status": "Failed"}
22  ▼ delete:
23  ▼   responses:
24  ▼     200:
25    |   description: The processing has been canceled
26    |   body:
27    |     application/json:
28    |       example: {"status": "Canceled"}
```

Response

STATUS 200

The API is still processing the request

Body application/json

Examples: Example

{ "status": "Working" }

application/json object

Example: {"status":"Working"}

Create an HTTP 202 post body response with necessary headers

1. Return to API Designer.
2. In the file browser, expand the resourceTypes folder and select collection.raml.
3. Go to the line that contains the HTTP response status 503 and add a new line above it.

```
11  |   |   |   |   Location:
12  |   |   |   |   |   description: URL of the new <>resourcePathName<> |
13  |   |   |   |   |   !singularize>> information
13  |   |   |   |   body:
14  |   |   |   |   |   application/json:
15
16  |   |   |   503:
17  |   |   |   |   body:
18  |   |   |   |   application/json:
```

4. Align the cursor with the start of the next line and type

202:

5. Add a new line below the response status line.

6. In the shelf, click description.

7. Type | and add a new line.

8. In the new line, type the following sentence:

The request has been accepted for processing. Use the URI provided in the Location header in the response to monitor the status.

```
15 ▼ 202:  
16   description: |  
17     The request has been accepted for processing. Use the URI  
18     provided in the Location header as a part of the response  
19     to monitor the status of the processing.  
20 ▼ 503:  
21   body:
```

9. Press enter and press backspace to go one tab space behind.

10. In the shelf, click headers.

11. In the shelf, scroll down and click Location.

```
18   headers:  
19   |  
20 ▼ 503:  
21   body:  
  
  ↳ Expires  
  ↳ Date  
  ↳ Location  
  ↳ Retry-After  
  ↳ Tk
```

12. In the shelf click example.

13. Type the value of the example node as:

/<>resourcePathName>>/1234/status

```
15 ▼ 202:  
16   description: |  
17     The request has been accepted for processing. Use the URI  
18     provided in the Location header as a part of the response  
19     to monitor the status of the processing.  
20   headers:  
21   |  Location:  
22     example: /<>resourcePathName>>/1234/status
```

14. Save the project.

Add a new resource to track the state of post requests that are under processing

15. Go to the empty line before the /accounts main resource and press enter.

16. Press tab to align the cursor with the /accounts nested resource beginning and type:

```
/status:
```

```
    47     |    CUSTOMER
    |     is: [Traits.hasAcceptHeader, Traits.hasRe-
    |     {typeName: Account, customErrorMessageData-
    |     CustomErrorMessage} ]
    48     |     /status:
    49
    50 ▼  /accounts:
```

17. Save the project.

Add necessary methods and HTTP response statuses to give meaningful information back to the API consumers

18. In the file browser, right-click the resourceTypes folder and select New File > RAML 1.0 > Resource Type.

19. In the Add new Resource Type file dialog box, set the name to stateSpecificResourceType.raml and click OK.

20. In stateSpecificResourceType.raml, add a new line below the first line.

21. In the shelf, click get.

```
1  #%RAML 1.0 ResourceType
2

DOCS (3)          METHODS (7)
description      get put
displayName      post delete
usage           options head
                  patch
```

22. In the shelf, click responses.
23. In the shelf, click 200.
24. In the shelf, click description.
25. Type the value of the description node as: The API is still processing the request.

```

1  %%RAML 1.0 ResourceType
2 ▼ get:
3 ▼   responses:
4   |  200:
5   |    | description: The API is still processing the request

```

26. Add a new line and in the shelf, click body.
27. In the shelf, click application/json.

```

1  %%RAML 1.0 ResourceType
2 ▼ get:
3 ▼   responses:
4 ▼     200:
5     |   description: The API is st.
6     |   body:
7

```

BODY (2)

- application/json
- application/xml

28. In the shelf, click example, and type the value as:

```

{ "status": "Working"}

1  %%RAML 1.0 ResourceType
2 ▼ get:
3 ▼   responses:
4 ▼     200:
5     |   description: The API is still processing the request
6 ▼     |   body:
7     |       application/json:
8     |         example: {"status": "Working"}

```

29. Press enter to add a new line.
30. Copy the lines containing the details for the 200 response paste it in the new empty line.

31. Press enter and paste the lines again.

```
8   |     example: {"status": "Working"}  
x 9   |     200:  
x 10  |       description: The API is still processing the request  
x 11 ▼ |       body:  
x 12 ▼ |         application/json:  
13   |           example: {"status": "Working"}  
x 14   |           200:  
x 15   |             description: The API is still processing the request  
x 16 ▼ |             body:  
x 17   |               application/json:  
18   |                 example: {"status": "Working"}
```

32. In the first set of pasted lines, replace 200 with 303 and correct the indentation.

```
6 ▼ |     body:  
7   |       application/json:  
8   |         example: {"status": "Working"}  
9 ▼ |     303:  
10  |       description: The API is still processing the  
11 ▼ |       body:  
12 ▼ |         application/json:
```

33. Edit the description node value to:

The processing has finished successfully. Use the URI provided in the Location header to obtain the actual result.

```
303:  
  description: The processing has finished successfully. Use  
  the URI provided in the Location header to obtain the actual  
  result  
  body:  
    application/json:  
      example: {"status": "Working"}
```

34. Press enter after the description to add a new line.

35. In the shelf, click headers.

36. In the shelf, click Location.

37. In the shelf, click example, type the value as:

/<>resourcePathName>>/1234

```
9 ▼ |     303:  
10  |       description: The processing has finished successfully. Use  
11  |       the URI provided in the Location header to obtain the actual  
12  |       result  
13  |       headers:  
14  |         Location:  
15  |           example: /<>resourcePathName>>/1234  
16  |       body:
```

Note: Always add a space between the node and the value

38. In the response JSON example, change the status value from Working to Success.
39. In the second set of pasted lines, change the status code from 200 to 503 and correct the indentation.

```
9 ▼ | 303:  
10 |   description: The processing has finished successfully. Use  
     the URI provided in the Location header to obtain the actu  
     result  
11 ▼ |   headers:  
12 |     Location:  
13 |       example: /<<resourcePathName>>/1234  
14 ▼ |   body:  
15 |     application/json:  
16 |       example: {"status": "Success"}  
17 ▼ | 503:  
18 |   description: The API is still processing the request  
19 -
```

40. In the description node below the 503 status code line, change the value of the node to: The processing has failed.
41. In the response JSON example, change the status value from Working to Failed.

```
16 |   example: {"status": "Success"}  
17 ▼ | 503:  
18 |   description: The processing has failed  
19 ▼ |   body:  
20 |     application/json:  
21 |       example: {"status": "Failed"}
```

42. Add a new line at the end and place the cursor at the beginning of the line.
43. In the shelf, click delete.
44. In the shelf, click responses.
45. In the shelf, click 200.
46. In the shelf, click description.

```
22 ▼ | delete:  
23 ▼ | responses:  
24 |   200:  
25 |     description:
```

47. Type the value of the description node as: The processing has been canceled.
48. Press enter and copy the lines for the HTTP response body and paste it in the HTTP 200 response body.

49. In the response JSON example, change the status value from Failed to Canceled.

```
22 ▼ | delete:
23 ▼ |   responses:
24 ▼ |     200:
25 |       description: The processing has been canceled
26 ▼ |       body:
27 |         application/json:
28 |           example: {"status": "Canceled"}
```

50. Save the project.

51. In the file browser, select api.raml.

52. Go to the resourceType references at the root and add a new line below the member resourceType.

53. Indent the line and type:

```
stateSpecificResourceType: !include resourceTypes/stateSpecificResourceType.raml
```

```
27 ▼ | resourceTypes:
28 |   collection: !include resourceTypes/collection.raml
29 |   member: !include resourceTypes/member.raml
30 |   stateSpecificResourceType: !include
31 |     resourceTypes/stateSpecificResourceType.raml
```

54. Add a new line below the line that contains /status nested method and press enter.

55. Press tab and type:

```
type: stateSpecificResourceType
```

```
49 ▼ |   /status:
50 |     type: stateSpecificResourceType
51 |
```

56. Copy the lines that define the status resource.

```
80 ▼ |   /status:
81 |     type: stateSpecificResourceType
82 |
83 ▼ |   /transactions:
```

57. Add the /status resource as a nested resource for /accounts and /transactions main resource.

```
85 |   /{transaction_id}:
86 |     type: { member: { uriParameterName: transaction_id,
87 |       getResponseBodyDataType: Transaction, getResponse:
88 |         !include examples/TransactionExample.json,
89 |         customErrorMessageType: CustomErrorMessage } }
```

58. Save the project.

59. In API Console, click the GET tab for the /customers/{customer_id}/status resource.

60. Scroll down to the response and verify that you can see the various status codes for the responses.

Response

STATUS 200

200 The API is still processing the request

303 Body application/json

503 Examples: Example

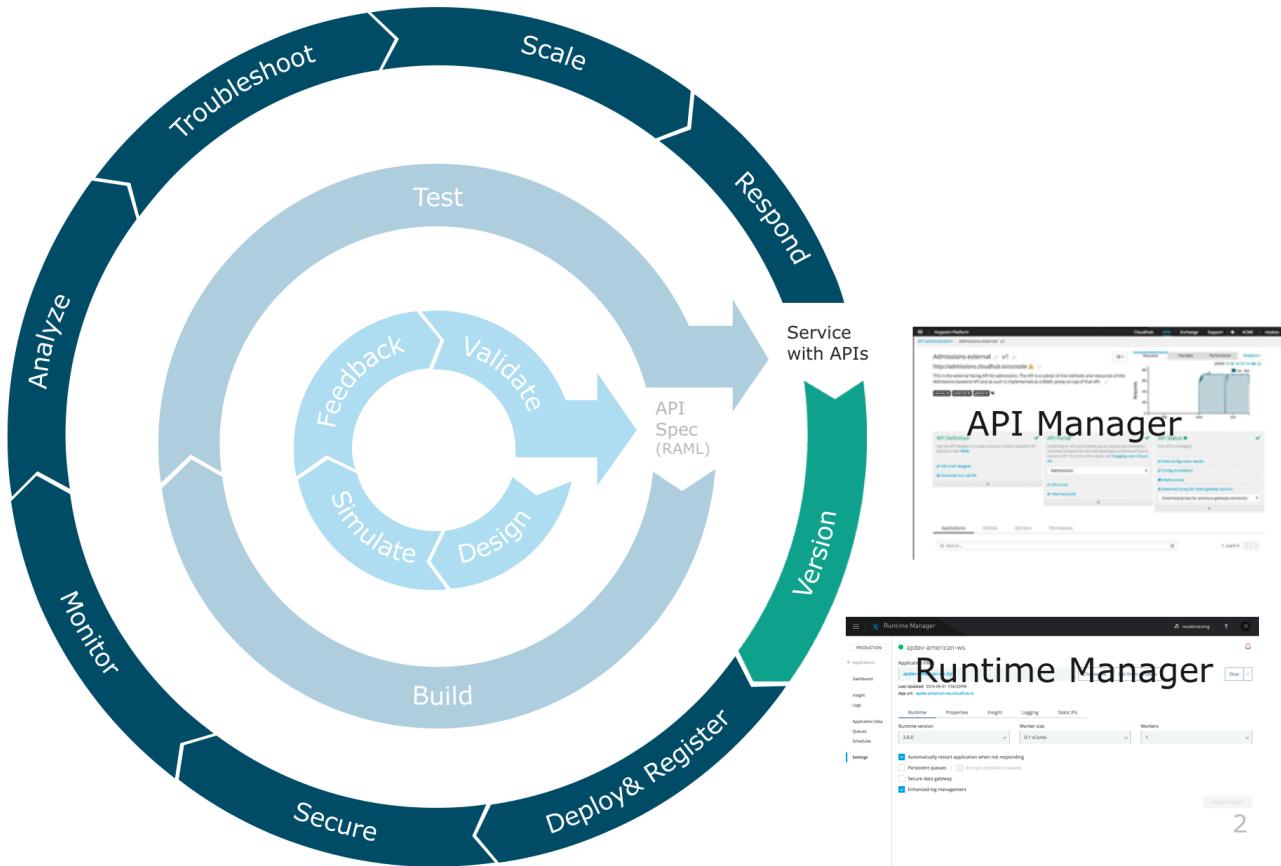
{
 "status": "Working"
}

application/json object

Example: {"status":"Working"}

61. Click CLOSE X.

Module 13: Versioning APIs



Objectives:

- Explain when and when not to version APIs.
- Describe the methods for versioning APIs.
- Document changes in new API versions using shared API Portals.
- Deprecate older versions of APIs.

Walkthrough 13-1: Add a new API version

In this walkthrough, you create a new version of the ACME Banking API. You will:

- Create a new API version in API Manager.
- Share the older version API Portal for the new version of the API.
- Compare the API Portal items in the two versions.
- Deprecate the old version of the API.

The screenshot shows the API Administration interface with two API versions listed: 'ACME Banking API' (2.0) and 'ACME Banking API' (1.0). The left sidebar includes links for Applications, Permissions, Policies, SLA Tiers, and Settings. The right pane displays details for each API, including API Status (Unregistered), Set the API URL..., and Add a description... buttons. Below the APIs are sections for 'API Definition' and 'API Portal'. The 'API Definition' section has a link to 'Define API in API designer'. The 'API Portal' section shows 'No portal' selected. A 'DEPRECATED' button is visible at the top right of the 1.0 API panel.

Add a new version of API definition in the API Designer

1. Return to API Designer.
2. Click the API Administration link above the API Designer menu.

The screenshot shows the API Administration menu with the path: API Administration > ACME Banking API (1.0) - Settings > Designer. The menu bar includes Project, View, Help, and the current page path /api.raml (2 warnings). The main area shows the RAML code for version 1.0, which includes '#%RAML 1.0' and 'baseUri: https://mocksvc.mule'. A 'datatypes' folder is also visible.

3. On the API administration page, click the Add Version button to the right of the ACME Banking API.

The screenshot shows the API Administration page for the ACME Banking API. The '1.0' version is listed. To its right is a 'Private port' section with a 'New Version' button. A dropdown menu is open over the 'New Version' button, showing options: 'Add Version', 'New Version', and 'Import'.

4. In the Add version drop-down menu, select New Version.

5. In the Add API version dialog box, set the Version name to 2.0.

Add API version

API name
ACME Banking API

Version name *
2.0

API endpoint

Description

Cancel Add

6. Click Add.
7. In the API Definition section, click Define API in API Designer.

← API Administration

ACME Banking API / 2.0 ✓ /

API Status: Unregistered

Set the API URL...

Add a description...

ADD A TAG

API Definition

Use the API Designer to create a concise, human-readable API definition with [RAML](#).

[Define API in API designer](#)

API Portal

Publishing an API portal allows you to expose documentation and other content that can help developers understand how to use your API. For more information, see [Engaging users of your API](#).

No portal

8. In the RAML editor, place the cursor at the end of the line that defines the version and press enter two times.
9. Type `/{version}:` and press enter.

```

1  #%RAML 0.8
2  title: ACME Banking API
3  version: 2.0
4
5  /{version}:

```

10. Press tab and type /customers::

```
1  #%RAML 0.8
2  title: ACME Banking API
3  version: 2.0
4
5  /{version}:
6    | /customers:
```

11. Save the project.

Note: Versioning is done only when APIs are live and when the previous version is deployed and being used by API consumers. We do not fill in the RAML definition for v2.0 since versioning is not necessary in this scenario.

Link multiple versions of the API in the shared portal

12. In API Designer, click ACME Banking API (2.0) - Settings to go to the API version details page.
13. In the API Portal section, click the drop-down arrow next to the No portal option.

ACME Banking API 2.0

API Status: Unregistered

Set the API URL...

Add a description...

ADD A TAG

API Definition

Use the API Designer to create a concise, human-readable API definition with [RAML](#).

[Edit in API designer](#)

[Download as a .zip file](#)

API Portal

Publishing an API portal allows you to expose documentation and other content that can help developers understand how to use your API. For more information, see [Engaging users of your API](#).

No portal

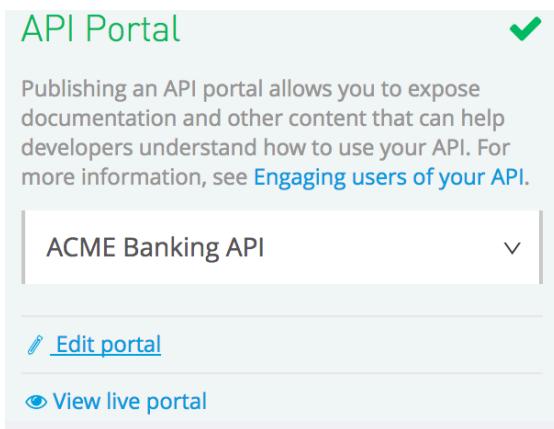
Create new portal

Active portals

[ACME Banking API](#)

14. Select the ACME Banking API active portal from the list.

15. Click View live portal.



Compare the non-identical portal items

16. In the left-side navigation, select the API reference page.

17. Verify that you see the version 2.0 RAML API Definition resources in API Console.

A screenshot of the API Console interface. The top bar is green with the MuleSoft logo, the API name "ACME Banking API 2.0", and status indicators like "PRIVATE" and "apiDesignTraining". The main area shows the "Developer portal" section with "ACME Banking API (2.0) - API reference". On the left, there's a navigation sidebar with "Home", "API reference" (which is selected and highlighted in grey), "References", "HTTP status codes", and "API Notebook". The right side shows the "API reference" content. It includes a "Resources" section with endpoints like "{version}" and "{version}/customers". There's also a note about "API is behind a firewall (?)" with a checkbox and a "Collapse All" button. At the bottom, there's a "ROOT RAML URL:" field containing the URL "https://anypoint.mulesoft.com/apiplatform/repository/v2/organizations/a6fb4480-6e30-4248-ac23-6e7a7...". Below it are download links for ".zip" and ".yaml" files.

18. Copy the link of the API portal URL for ACME Banking API 2.0 and paste it in the course snippets.txt file in Module 13 under the Version 2.0 API Portal URL heading.

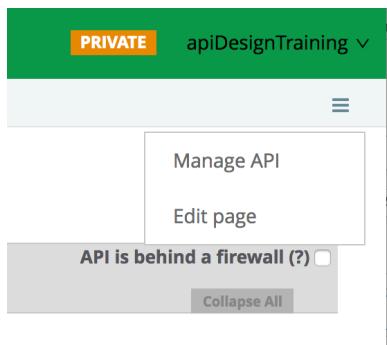
-----Module 13-----

Version 2.0 API Portal URL:

<https://anypoint.mulesoft.com/apiplatform/mulesoft-1581/#/portals/organizations/a6fb4480-6e30-4248-ac23-6e7a751b5606/apis/11698133/versions/178909/pages/225831>

Version 1.0 API Portal URL:

19. Click the menu icon in the right top corner and select Manage API.



20. Click the version 2.0 drop-down in the top center and select 1.0.

A screenshot of the ACME Banking API configuration page. At the top, it says 'ACME Banking API' and has a version dropdown set to '2.0'. Below the dropdown is a status indicator 'API Status: Unregistered'. There are two input fields: 'Set the API URL...' and 'Add a description...'. A 'ADD A TAG' button is located below the description field. The version dropdown has a list of options: '1.0' (which is highlighted with a blue border), '2.0', and '3.0'.

21. In the API Portal section, click View live portal.

22. Copy the link to the API portal URL for ACME Banking API 1.0 and paste it in the course snippets.txt file.

23. Verify that the version ids in the URL are different for the different versions.

Version 2.0 API Portal URL:

`https://anypoint.mulesoft.com/apis/11698133/versions/178909|`

Version 1.0 API Portal URL:

`https://anypoint.mulesoft.com/apis/11698133/versions/162026|`

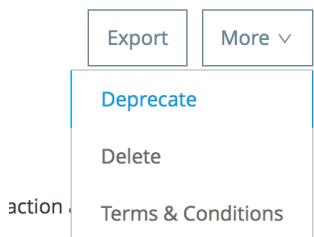
Deprecate the older version of the API

24. Click the menu icon in the top-right corner of the API Portal and select Manage API.

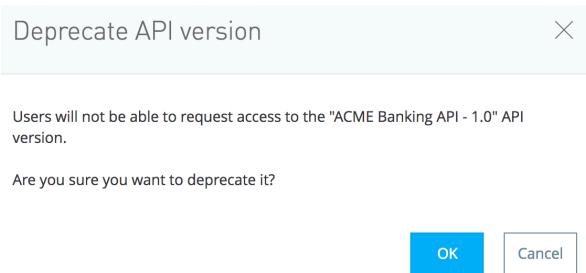
25. Click the More button to the top right corner in the API version details page.

The screenshot shows the API version details page for 'ACME Banking API' version '1.0'. At the top right, there is a 'More' button with a dropdown arrow. Below it are buttons for 'Export' and 'More'. The main content area includes fields for 'Set the API URL...' and a description: 'ACME Banking API helps applications consume information related to bank customers, their accounts and transaction activity in these accounts.' There is also a 'ADD A TAG' button at the bottom.

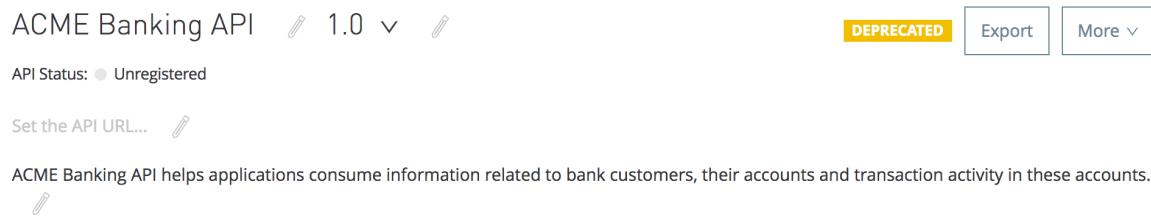
26. Select Deprecate.



27. In the dialog box that appears, click OK.



28. Verify that you see a yellow Deprecated tag next to the API name and version.



Note: It is not a best practice to deprecate APIs that are not being consumed. This is just a walkthrough to demonstrate the process of versioning and deprecation.