



Anypoint Platform Development: API Design

Student Manual

July 30, 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 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 ...	25
Walkthrough 3-1: Explore Anypoint Platform	26
PART 2: DEFINING APIs WITH THE RESTFUL API MODELING LANGUAGE (RAML).....	30
MODULE 4: DEFINING API RESOURCES AND METHODS.....	31
Walkthrough 4-1: Create an API and define resources and methods in RAML 1.0	32
MODULE 5: SPECIFYING RESPONSES.....	42
Walkthrough 5-1: Add HTTP 2xx responses to resource methods	43
Walkthrough 5-2: Add responses bodies to return custom error information for client-side errors	49
Walkthrough 5-3: Add response bodies to return custom error information for server-side errors	53
Walkthrough 5-4: Add flexible content-type support to a resource method	57
Walkthrough 5-5: Add caching information to HTTP responses	62
MODULE 6: MODELLING DATA	64
Walkthrough 6-1: List datatypes and their attributes for an API	65
Walkthrough 6-2: Create datatype fragments	71
Walkthrough 6-3: Specify datatypes in resource methods	80
Walkthrough 6-4: Validate datatype attribute values using patterns	85
Walkthrough 6-5: Define example fragments for datatypes	89

MODULE 7: DOCUMENTING AND TESTING APIs	94
Walkthrough 7-1: Add a documentation fragment to a RAML API definition	95
Walkthrough 7-2: Add description nodes to a RAML API definition	101
Walkthrough 7-3: Use the mocking service in API Console to test an API	107
MODULE 8: MAKING APIS DISCOVERABLE	111
Walkthrough 8-1: Publish a RAML API fragment to Anypoint Exchange	112
Walkthrough 8-2: Publish a RAML API Specification to Anypoint Exchange.....	118
Walkthrough 8-3: Create and customize an API Portal.....	124
Walkthrough 8-4: Create a sample use case with API Notebook inside the API Portal.....	132
MODULE 9: REUSING PATTERNS	141
Walkthrough 9-1: Define and use a resource type for resources that perform operations on a collection	142
Walkthrough 9-2: Define and use a resource type for resources that perform operations on a member.....	148
Walkthrough 9-3: Define and use various traits for resources and methods	156
MODULE 10: MODULARIZING APIS.....	163
Walkthrough 10-1: Create and use a library of traits.....	164
Walkthrough 10-2: Internationalize documentation and resource description using an overlay	169
Walkthrough 10-3: Define and use API extensions to promote portability to test in multiple environments.....	172
MODULE 11: SECURING APIS	176
Walkthrough 11-1: Define a custom security scheme for an API.....	177
Walkthrough 11-2: Consume an OAuth2.0 security scheme for an API and secure API resources	182
MODULE 12: ENHANCING API RESPONSES USING HYPERMEDIA	186
Walkthrough 12-1: Modify an API definition to implement state-specific client responses ..	187
MODULE 13: VERSIONING APIS	193
Walkthrough 13-1: Add a new API version	194

Introducing the Course

The screenshot shows the MuleSoft Design Center interface. At the top, there's a navigation bar with a menu icon, the 'Design Center' logo, and a search bar labeled 'Search...'. To the right of the search bar are 'Get Started' and '+ Create' buttons. Below the navigation bar, the main area is titled 'Projects' and shows a list of projects:

Name	Project Type
Banking Data Types	API Fragment
ACME Banking API	API Specification

On the right side of the interface, the 'ACME Banking API' project is selected. It displays the following details:

- Version: v1
- Supported media type: application/json
- Supported protocols: HTTPS
- API base URI: <https://mocksvc.mulesoft.com/mocks/59a9cc34-171a-4825-a68d-7d0367cad03/mocks/5cf0cae2-02bc-47b1-a384-5a4676bb2228>
- API resources: /customers

The central part of the interface shows the RAML file content for the 'ACME Banking API' project. The code is as follows:

```
1  #%RAML 1.0
2  baseUri: https://mocksvc.mulesoft.com/mocks/59a9cc34-
3  version: v1
4  title: ACME Banking API
5  mediaType: application/json
6
7 documentation:
8   - !include documentation/acmeBankHeadline.raml
9   - !include documentation/acmeBankDoc.raml
10
11 types:
12   Customer: !include exchange_modules/b2447622-1656-4
13   Account: !include exchange_modules/b2447622-1656-4f
14   Transaction: !include exchange_modules/b2447622-165
15   CustomErrorMessage: !include datatypes/CustomErrorM
16
17 resourceTypes:
18   collection: !include resourceTypes/collection.raml
19   member: !include resourceTypes/member.raml
20   stateSpecific: !include resourceTypes/stateSpecific
21
22 uses:
23   Traits: libraries/TraitsLibrary.raml
24
```

Objectives:

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

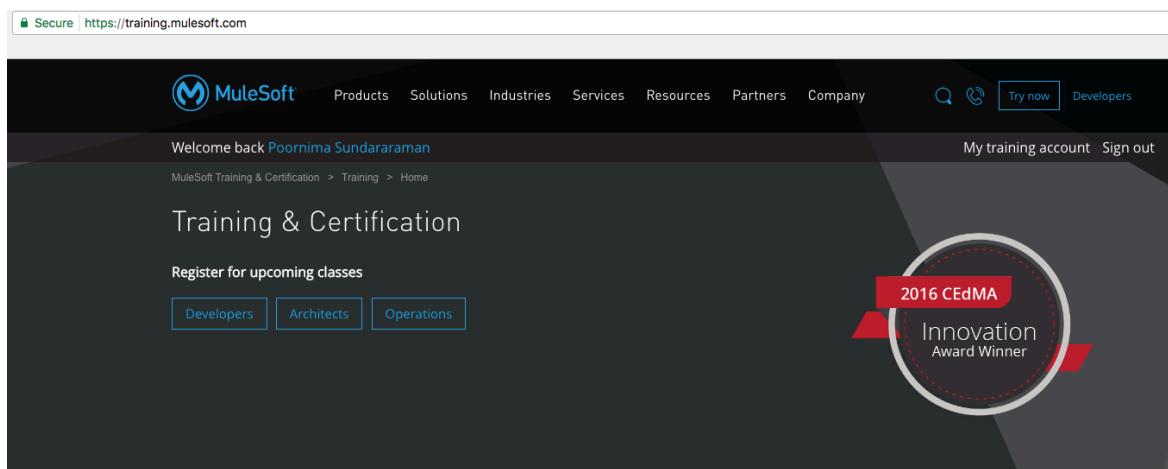
Walkthrough: Set up your computer for class

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.

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



This instructor-led course is for API designers, developers, and architects who want to get hands-on experience creating well-designed, modular API definitions using RAML 1.0 and Anypoint Platform. It includes a voucher code to take the [MuleSoft Certified Developer - API Design Associate](#) exam.

A downloadable datasheet for the course can be found [here](#).

FACILITIES

- Courseware

SETUP REQUIREMENTS

- A computer with a minimum screen resolution of 1024x768
- Unrestricted internet access to port 80 (with > 5Mbps download and > 2Mbps upload)
- An [Anypoint Platform](#) account

INSTRUCTORS

Grace Sakurada
Training Operations Lead at MuleSoft

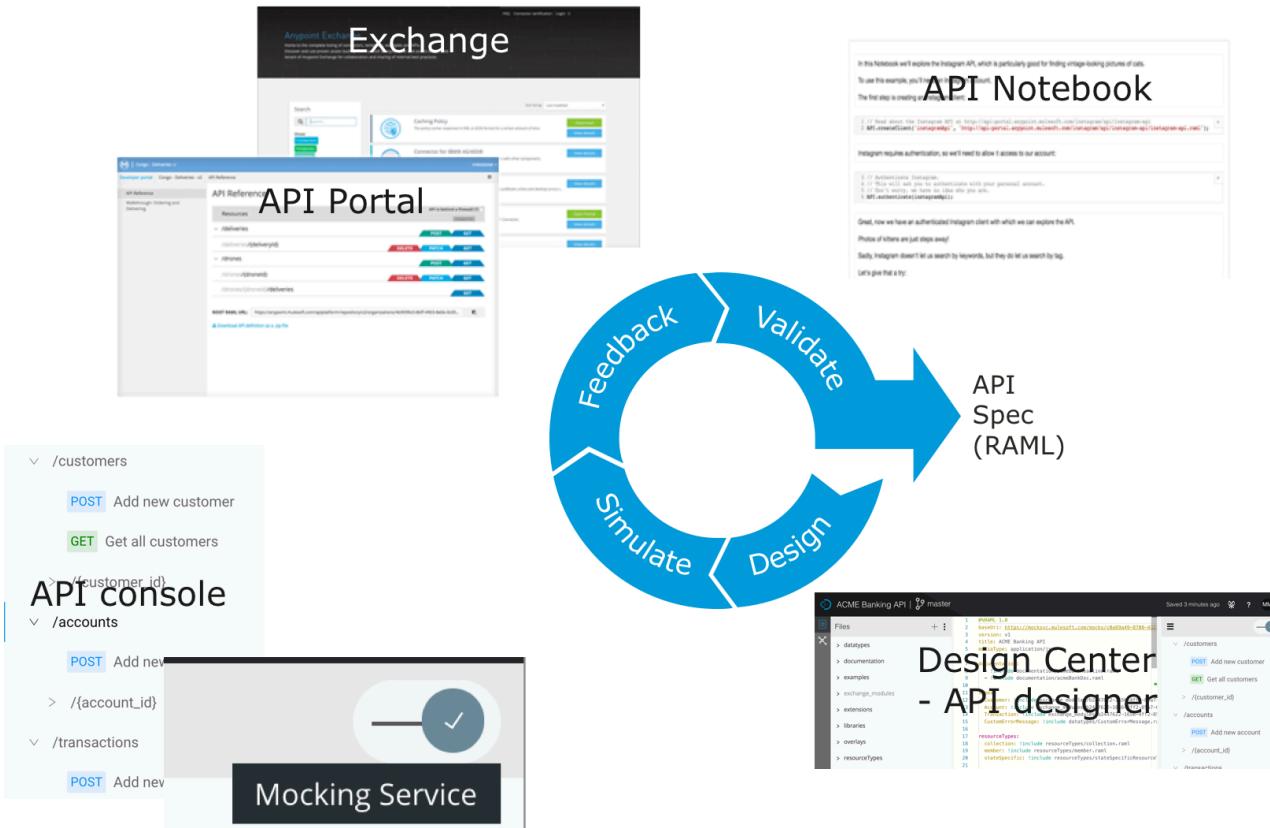
MATERIALS

- | | |
|--|-----------------------------------------------|
| | APDevApiDesign3.8 Student Manual (PDF) |
| | 49.92MB PDF |
| | APDevApiDesign3.8 Student Slides (ZIP) |
| | 18.89MB ZIP |
| | APDevApiDesign3.8 Student Files (ZIP) |
| | 246.51KB ZIP |

- 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/wsdl/soap/" targetNamespace="http://www.soaptraining.com"> <wsdl:types> <xsd:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" targetNamespace="http://soap.training" elementFormDefault="qualified"> <xsd:element name="findFlight" type="xsd:string"> <xsd:complexType> <xsd:sequence> <xsd:element name="flightId" type="xsd:int"/> </xsd:sequence> </xsd:complexType> </xsd:element> <xsd:element name="findFlightResponse" type="xsd:string"> <xsd:complexType> <xsd:sequence> <xsd:element name="flight" type="xsd:string"/> </xsd:sequence> </xsd:complexType> </xsd:element> </xsd:schema> </wsdl:types> <wsdl:message name="findFlightMessage"> <wsdl:part name="request" type="xsd:string"/> <wsdl:part name="response" type="xsd:string"/> </wsdl:message> <wsdl:message name="listAllFlightsMessage"> <wsdl:part name="request" type="xsd:string"/> <wsdl:part name="response" type="xsd:string"/> </wsdl:message> <wsdl:portType name="FlightServicePortType"> <wsdl:operation name="findFlight"> <wsdl:input message="findFlightMessage"/> <wsdl:output message="findFlightResponseMessage"/> </wsdl:operation> <wsdl:operation name="listAllFlights"> <wsdl:input message="listAllFlightsMessage"/> <wsdl:output message="listAllFlightsResponseMessage"/> </wsdl:operation> </wsdl:portType> <wsdl:binding name="FlightServiceSoapBinding" type="wsdl:PortType"> <wsdl:operation name="findFlight"> <wsdl:input> <soap:Envelope> <soap:Body> <findFlight> <flightId>1</flightId> </findFlight> </soap:Body> </soap:Envelope> </wsdl:input> <wsdl:output> <soap:Envelope> <soap:Body> <findFlightResponse> <flight>Flight A</flight> </findFlightResponse> </soap:Body> </soap:Envelope> </wsdl:output> </wsdl:operation> <wsdl:operation name="listAllFlights"> <wsdl:input> <soap:Envelope> <soap:Body> <listAllFlights> <flightId>1</flightId> </listAllFlights> </soap:Body> </soap:Envelope> </wsdl:input> <wsdl:output> <soap:Envelope> <soap:Body> <listAllFlightsResponse> <flights> <flight>Flight A</flight> <flight>Flight B</flight> </flights> </listAllFlightsResponse> </soap:Body> </soap:Envelope> </wsdl:output> </wsdl:operation> </wsdl:binding> <wsdl:service name="FlightService"> <wsdl:port name="FlightServicePort" binding="FlightServiceSoapBinding"> <wsdl:location href="http://www.soaptraining.com/FlightService.asmx" /> </wsdl:port> </wsdl:service> </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 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 displaying the www.webservicex.net/new/Home/Index page. The main heading is "Explore services for all devices". Below it, a sub-section titled "getAirportInformationByAirportCode" provides details about the operation, including parameters like "Get Airport Code, CityOrAirport Name, Country, Country Abbv, 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". A "Test" section allows users to invoke the operation with parameters like "Parameter: Value" and "airportCode: LHR". A "SOAP 1.1" section shows a sample request:

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

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 displaying the www.webservicex.net/new/Home/Index page. The main heading is "Explore services for all devices". Below it, a sub-section titled "Standards and Lookup Data" is highlighted. The "Webservices Directory" section lists several categories: Business and Commerce, Messaging, Standards and Lookup Data, Value Manipulation / Unit Converter, Graphics and Multimedia, Other Web Services, and Utilities. Each category has a corresponding icon.

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

The screenshot shows a web browser displaying a list of services under the 'Webservices' category. The services listed are:

- Country Details**: Get Currency, Currency code, International Dialing code, ISO country code for all countries
- 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.
- 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

View the Airport Information Webservice endpoint WSDL

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

The screenshot shows the 'Airport Information Webservice Detail' page. The 'Endpoint' section contains the URL:

```
http://www.webservicenet.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>
<wsdl:message name="GetAirportInformationByCountryHttpPostOut">
<wsdl:part name="Body" element="tns:string"/>
</wsdl:message>
<wsdl:message name="getAirportInformationByAirportCodeHttpPostIn">
<wsdl:part name="airportCode" type="xsd: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/">
```

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.

Demo

Click [here](#) for a complete list of operations.

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

Test

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

Parameter	Value
airportCode:	LHR

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.

www.webservicex.net/airport.asmx/getAirportInformationByAirportCode

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>
<LatitudePeerS>N</LatitudePeerS> <LongitudeDegree>0</LongitudeDegree> <LongitudeMinute>27</LongitudeMinute> <LongitudeSeconds>0</LongitudeSeconds>
<LongitudePeerW>W</LongitudePeerW> </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> <LatitudePeerS>N</LatitudePeerS> <LongitudeDegree>0</LongitudeDegree>
<LongitudeMinute>27</LongitudeMinute> <LongitudeSeconds>0</LongitudeSeconds> <LongitudePeerW>W</LongitudePeerW> </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.

Web Toolkit Online Useful Online Tools for Developers

» Home » Contact

CONVERTER

- » Hexadecimal decimal
- » Hexadecimal color
- » Less to CSS
- » CSV to XML

ENCODER / DECODER

- » Base64
- » URL

FORMATTER

- » CSS
- » JSON
- » SQL
- » XML

GENERATOR

- » Lorem Ipsum
- » QR Code
- » Password

MINIFIER

- » CSS
- » JSON
- » SQL
- » XML

SECURITY

- » MD5 Digest

TESTER

- » JavaScript
- » JSON

XML Formatter

XML Content

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

Format  **Clear** 

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 page. On the left, there's a sidebar with links like 'Start here', 'Building Slack apps', 'Recent updates', 'App features', 'Internal integrations', 'Incoming webhooks', 'Slash commands', 'Bot users', 'Enterprise Grid', 'Legacy custom integrations', 'Messages', 'Introduction', 'Basic formatting', 'Attaching content', 'Unfurling links', 'Threading messages', 'Interactive messages', 'Message guidelines', 'Message builder', and 'APIs' (with 'Web API' selected). The main content area is titled 'Slack Web API' and contains a brief introduction: 'The Slack Web API allows you to build applications that interact with Slack in more complex ways than the integrations we provide out of the box.' Below this is a 'Basics' section with a note about HTTP RPC-style methods and JSON responses. To the right, there's a 'Related Articles' section with a link to 'Building Slack Bots in Swift'. A large box on the right is titled 'channels.list' and contains tabs for 'Documentation' (which is active) and 'Tester'. The 'Documentation' tab shows examples of the JSON response for different cases: success ('ok: true, stuff: "This is good"'), failure ('ok: false, error: "something_bad"'), and warnings ('ok: true, warning: "something_problematic", stuff: "Your requested information"'). The 'Tester' tab has fields for 'token' (set to 'No token') and 'exclude_archived' (set to 'Optional, default=false'). There's also a 'Test Method' button. A dropdown menu at the top right says 'View another method...'.

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.

This screenshot shows the same Slack Web API documentation page as the previous one, but without the 'Tester' tab visible. The 'Documentation' tab is active, showing the same basic information and examples as the previous screenshot. The sidebar and overall layout are identical.

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.

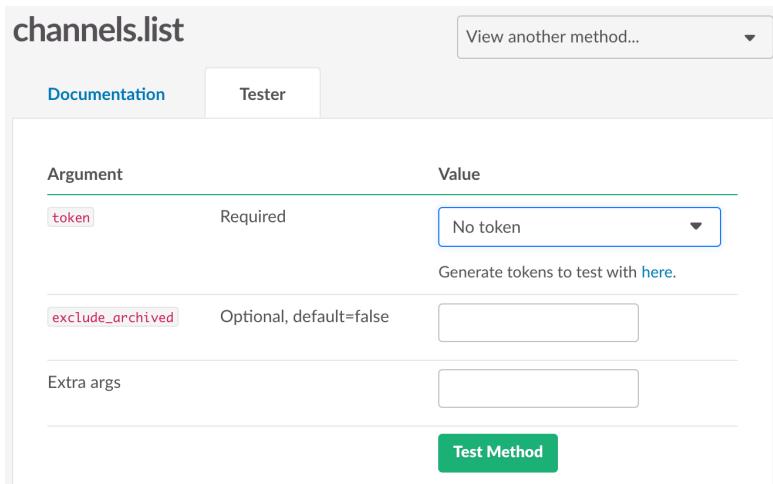
channels.list

View another method... ▾

Documentation Tester

Argument	Value
token	Required No token
Generate tokens to test with here .	
exclude_archived	Optional, default=false
Extra args	

Test Method



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 API 'Getting Started' page. On the left sidebar, under 'Getting Started', there are links for 'Get your API key', 'Register a new application', 'Get an access token', 'Make API calls to Vimeo', 'Method', 'Endpoint', 'Parameters', 'Set your API version', 'Upload a video', and 'Stay up to date'. The main content area is titled 'Getting Started' and contains two sections: '1. Get your API key' and '2. Get an access token'. Below these sections, there is a table for the 'categories' endpoint. The table has columns for 'Name', 'Value', and 'Description'. It includes rows for 'page' (int), 'per_page' (int), 'sort' (Choose one), and 'direction' (string). A note at the bottom says 'Use Application: API Playground'. At the bottom right is a yellow 'Make Call' button.

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

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 API 'Getting Started' page. The sidebar on the left is identical to the previous one, with links for 'Get your API key', 'Register a new application', 'Get an access token', 'Make API calls to Vimeo', 'Method', 'Endpoint', 'Parameters', 'Set your API version', 'Upload a video', and 'Stay up to date'. The main content area is titled 'Getting Started' and contains three sections: '1. Get your API key', '2. Get an access token', and '3. Make API calls to Vimeo'. The '2. Get an access token' section includes a note about OAuth 2 authentication and a link to 'See the authentication documentation'. The '3. Make API calls to Vimeo' section is partially visible at the bottom.

2. In the left navigation bar, click Playground.
3. On the API/Playground page, click (Empty...) on the left-hand side.

The screenshot shows the Vimeo developer API playground interface. At the top, there's a navigation bar with links for API, Player, Guidelines, My Apps, Help, and Jobs. Below the navigation bar, the title "vimeo developer();" is displayed. On the left, a sidebar lists several API endpoints: (Empty..), (Empty..), root, categories, channels, contentratings, creativecommons, documents, groups, and languages. The "categories" link is highlighted. The main content area shows a GET request to "https://api.vimeo.com/categories". Below the request, there's a button labeled "View all API Endpoints" and a dropdown menu for "Use Application: API Playground". A prominent orange "Make Call" button is located at the bottom right of the main panel.

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

The screenshot shows the API response for the categories endpoint. At the top, it says "Use Application: API Playground" and has an orange "Make Call" button. The response body starts with "HTTP/1.1 200" and "Content-Type: application/vnd.vimeo.category+json". It also shows the "Host: api.vimeo.com" header. The JSON response itself is displayed below:

```
{
  "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.

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	
TRANSACTIONS – Bank accounts can be used to perform actions on the transactional data.			
i. Create a new transaction	Resource /accounts	Method POST	
ii. Get transactions for a specific account ID	Resource /accounts/{account_id}	Method GET	
iii. Get transaction information for a specific transaction ID	Resource /accounts/{account_id}	Method DELETE	
ACCOUNTS: Resource /accounts	Resource /accounts	Method PUT	
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	
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			

Objectives:

- Identify the different categories and actions for REST APIs.
- 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 REST 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).

The API consumers at ACME Bank are the developers who will build a web application for bankers to serve customers. What are the functions they should be able to perform with the API?

Categories:

CUSTOMERS

1. List categories for the ACME Banking API

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

2. List the actions that a developer can perform with ACCOUNTS

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

Create user stories to define functional requirements for a banking API

1. Navigate to the APDevApiDesign3.8_studentFiles directory on your computer.
2. Navigate to 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).

The API consumers at ACME Bank are the developers who will build a web application for bankers to serve customers. What are the functions they should be able to perform with the API?

1. List categories for the ACME Banking API

2. List the actions that a developer can perform with 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 HTTP methods for the 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 _____
ACCOUNTS: Resource _____	Resource _____	Method _____
i. Get list of all accounts for a specific customer ID	Resource _____	Method _____
ii. Create a new account	Resource _____	Method _____
iii. Get account information for a specific account ID	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. Create a new transaction	Resource _____	Method _____
ii. Get transactions for a specific account ID	Resource _____	Method _____
iii. Get transaction information for a specific transaction ID	Resource _____	Method _____

Translate categories into resources

1. Return to the module02 directory in the studentFiles directory 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}
 - v. 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

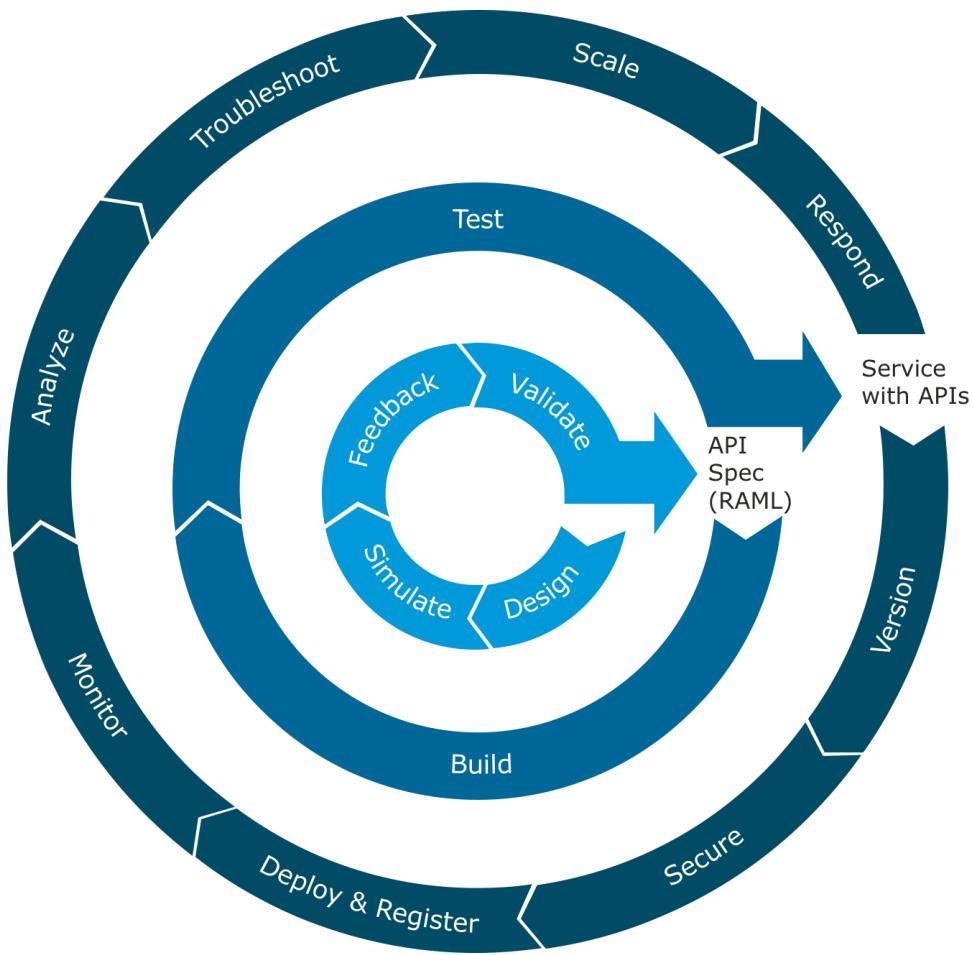
10. In the TRANSACTIONS category, move the second line that contains the resource and method to get transactions for a specific account inside the ACCOUNTS category.

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

TRANSACTIONS: Resource /transactions		
i. Create a new transaction	Resource /transactions	Method POST
ii. Get transaction information for a specific transaction ID	Resource /transactions/{transaction_id}	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.
- Describe the API development lifecycle with Anypoint Platform.
- Navigate Anypoint Platform.

Walkthrough 3-1: Explore Anypoint Platform

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

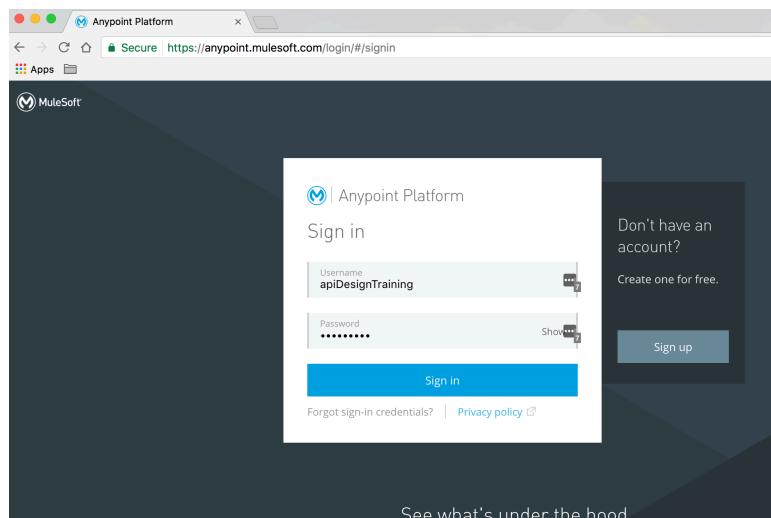
- Log in to Anypoint Platform.
- Explore the API entitlements in Anypoint Platform.



Log in to Anypoint Platform

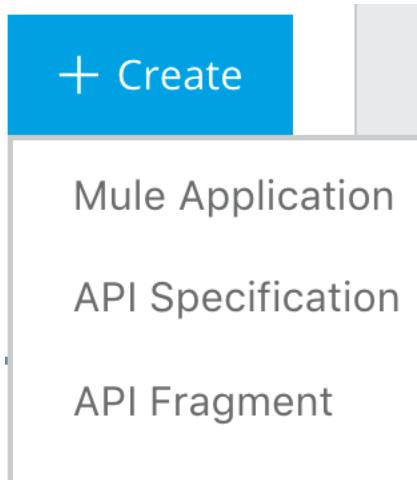
1. In a web browser, navigate to <https://anypoint.mulesoft.com>.
2. Log in to Anypoint Platform.

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 Design Center in Anypoint Platform

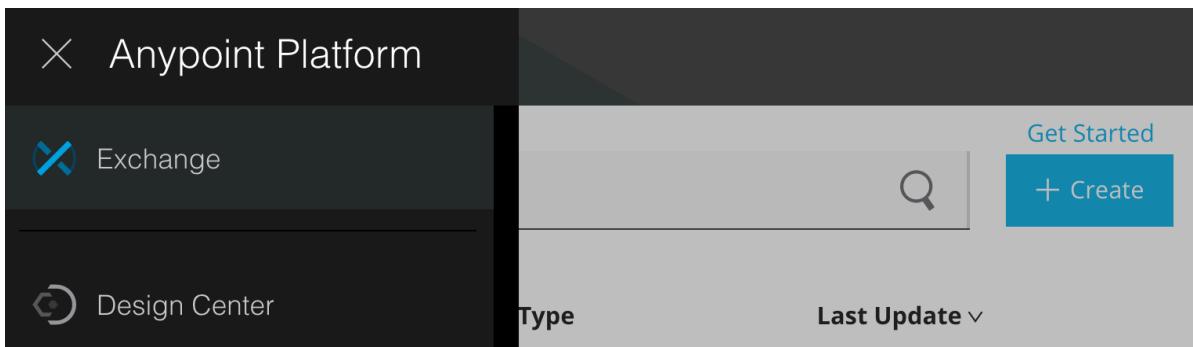
3. In the Anypoint Platform homepage, click the top left menu icon.
4. Click Design Center.
5. In the Design Center homepage, click Create in the top right corner.
6. Verify that you see the options to create API Specification, API Fragment and Mule application.



Note: In this class, we will be creating API Specification and API Fragment.

Explore Anypoint Exchange

7. Click the top left menu icon and select Exchange.



8. In the Exchange webpage, click the down arrow next to All types to the left of the search bar.

9. Click REST APIs.

The screenshot shows the 'All assets' page in the MuleSoft Anypoint Platform. On the left, a sidebar lists categories: All types, Connectors, Templates, Examples, REST APIs (which is highlighted with a blue border), SOAP APIs, RAML fragments, and Custom. The main area displays several cards for connectors and templates. One card for the 'Workday Connector' is shown, along with other cards for 'ServiceNow Connector' and 'MuleSoft'. A search bar and a 'New' button are at the top right.

10. Verify that you see all the public REST APIs listed.

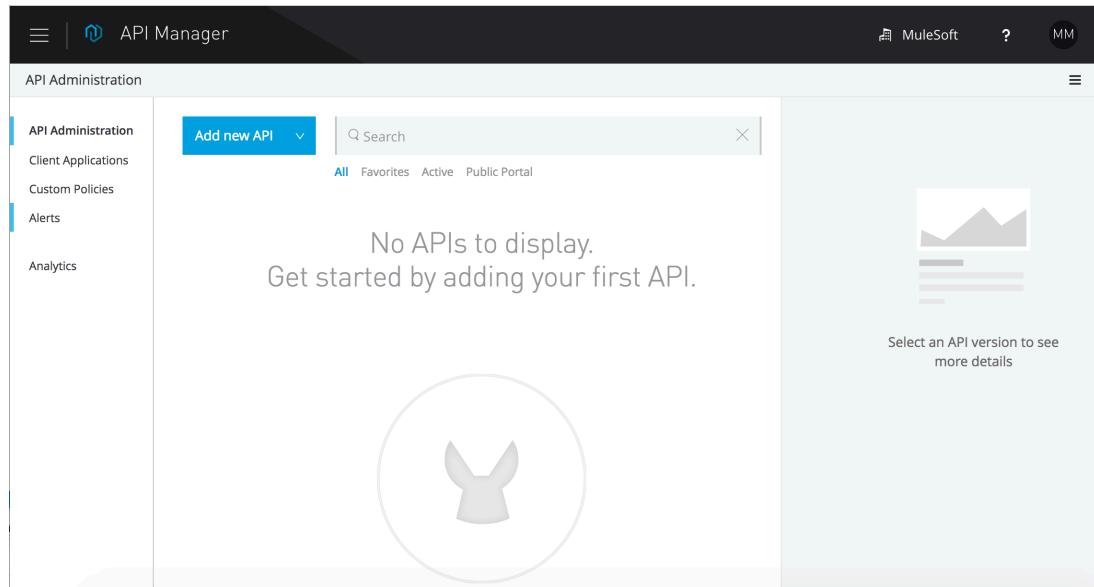
The screenshot shows the 'REST APIs' section of the MuleSoft Anypoint Platform. It lists several public REST APIs, each represented by a card with a green house icon. The APIs include 'Training: American Flights API', 'Optymize API', 'Pega API', 'Google Drive RAML', 'LinkedIn RAML', and 'Stripe RAML', all provided by MuleSoft. A 'Training' menu item is visible on the left, and a 'Go to old Exchange version' link is at the top right.

Explore API Manager in Anypoint Platform

11. Click the menu button located in the upper-left in the main menu bar.
12. From the menu, select API Manager.

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, updating terms and conditions for the APIs, and more.



Note: We will not be creating/designing new APIs in the API Manager. We will use Design Center in this class.

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

The screenshot shows the MuleSoft Anypoint Platform interface. On the left, there's a sidebar with a tree view of files: 'datatype', 'documentation', 'examples', 'exchange_modules', 'extensions', 'libraries', 'overlays', and 'resourceTypes'. The main area displays a RAML 1.0 file for the 'ACME Banking API'.

```
%RAML 1.0
baseUri: https://mocksvcs.mulesoft.com/mocks/c0a69a49-8708-432
version: v1
title: ACME Banking API
mediaType: application/json

documentation:
  - !include documentation/acmeBankHeadline.raml
  - !include documentation/ACME_Banking_API_1.0.raml

types:
  Customer: !include exchange_modules/Customer.raml
  Account: !include exchange_modules/Account.raml
  Transaction: !include exchange_modules/Transaction.raml
  CustomErrorMessage: !include exchange_modules/CustomErrorMessage.raml

resourceTypes:
  collection: !include resources/Collection.raml
  member: !include resources/Member.raml
  stateSpecific: !include resources/StateSpecific.raml
```

To the right, there's a preview of the developer portal. It shows a 'POST /customers' endpoint with a 'Add new customer' button. Below it, the 'Home' page of the developer portal lists various API operations:

- Customer: Add a new customer, Create a new account, Add a new bank account transaction
- Account: Delete a specific customer, Delete a specific account
- Transaction: Update a specific customer profile information, Update a specific bank account information

The developer portal also includes links for 'Developer portal' and 'API reference'.

Objectives:

- Create API definitions with RAML 1.0.
- Add documentation to RAML API definitions.
- Make APIs discoverable through API Portals and Anypoint Exchange.
- Test APIs through the API Console.
- Use patterns to refactor and modularize API definitions.
- Specify security schemes to secure resources in APIs.
- 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 Anypoint Platform Design Center interface. On the left, there's a sidebar with icons for files, projects, and settings. The main area has a title bar "ACME Banking API | master" and a status "Saved 3 minutes ago". Below the title bar, there are tabs for "Files" and "APIs". The "Files" tab is selected, showing a file named "acme-banking-api.raml". The content of the file is displayed as follows:

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

To the right of the code editor, there's a panel titled "ACME Banking API" which lists the API base URI and resources. The resources listed are "/customers", "/accounts", and "/transactions".

Objectives:

- Use Anypoint Platform Design Center to create API definitions with RAML 1.0.
- Define resources and methods in RAML API definitions.

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

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

- Create an API in Anypoint Platform Design Center.
- Define resources and nested resources identified for the API.
- Define HTTP methods for the resources.

```
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  
v. Get transactions for a specific account ID  
  
TRANSACTIONS: Resource /transactions  
i. Create a new transaction  
ii. Get transaction information for a specific transaction 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  
  
ACME Banking API | master  
Saved 3 minutes ago X ? MM  
Files + : acme-banking-api.raml  
1 #%RAML 1.0  
2 title: ACME Banking API  
3  
4 /customers:  
5 get:  
6 post:  
7 /{customer_id}:  
8 get:  
9 patch:  
10 delete:  
11 /accounts:  
12 get:  
13  
14 /accounts:  
15 post:  
16 /{accounts_id}:  
17 get:  
18 put:  
19 delete:  
20 /transactions:  
21 get:  
  
ACME Banking API  
API base URI  
  
API resources  
  
/customers  
  
/accounts  
  
/transactions
```

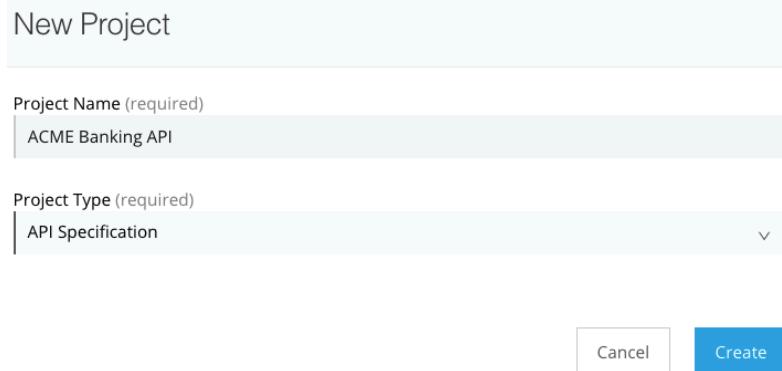
Create a new API in Anypoint Platform Design Center

1. Return to Design Center in Anypoint Platform.
2. Click the Create button next to the search text bar.

Name ^	Project Type	Last Update

3. In the New Project dialog box, enter the following information:

- Project Name: ACME Banking API
- Project Type: API Specification



4. Click Create.
5. In the API designer page that opens go to the file browser section, and click acme-banking-api.raml file name.

Review the resources and nested resources identified for the API

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

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
v. 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
ii. Get transaction information for a specific transaction ID	Resource /transactions/{transaction_id}	Method GET

7. Review the resources.

Define the resources

8. In the API designer, place the cursor after the title node and press the enter key two times to add two new lines.
9. Add the /customers resource.

/customers:

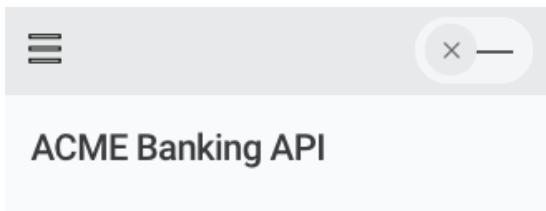
10. Press enter twice and add the /accounts resource.

11. Similarly, add the /transactions resource.

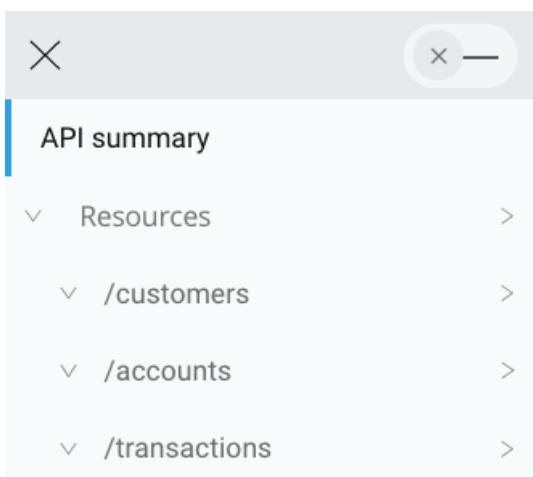


```
1  #%RAML 1.0
2  title: ACME Banking API
3
4  /customers:
5
6  /accounts:
7
8  /transactions:
```

12. In the top left corner of the API Console, click the menu icon.



13. Verify that you can view the Resources section with the three root level resources.



Define the nested resources

14. 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.

15. Indent by pressing the Tab key.
16. Add a nested resource called {customer_id} that will require a URI parameter called customer_id with a value to be passed to it.
17. On a new line, indent and add a nested resource called accounts.

```
1  #%RAML 1.0
2  title: ACME Banking API
3
4  /customers:
5      /{customer_id}:
6          /accounts:
7
```

18. Add a new line after the main /accounts resource.
19. Indent and add a nested resource called {account_id}.
20. Add a nested resource called transactions.

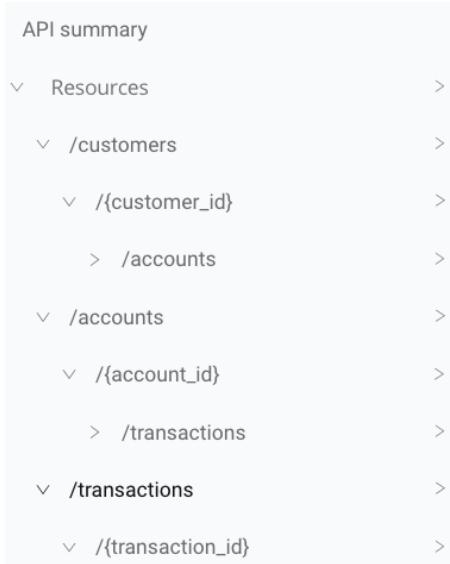
```
7
8  /accounts:
9      /{account_id}:
10     /transactions:
11
--
```

21. Go to a new line after the main /transactions resource.
22. Indent and add a nested resource called {transaction_id}.

```
11
12  /transactions:
13      /{transaction_id}:|
```

23. In the API Console, click the top left menu icon.

24. In the Resources section, click the arrow next to the /customers resource to see the nested resources when you expand all the resources.



Review the get methods identified for the API

25. Return to the WT 2-2 Resources and methods text file.

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
v. 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
ii. Get transaction information for a specific transaction ID	Resource /transactions/{transaction_id}	Method GET

26. Review the resources that have get methods.

Define HTTP methods for resources to retrieve collection and item data

27. Return to API designer.

28. Add a new line between the /customers and /{customer_id} resources.
29. Press tab to indent.
30. From the shelf, click get.

```

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

Parameters	Methods
uriParameters	get put

Note: Alternatively, you could press the control and spacebar keys, and click get from the drop-down box.

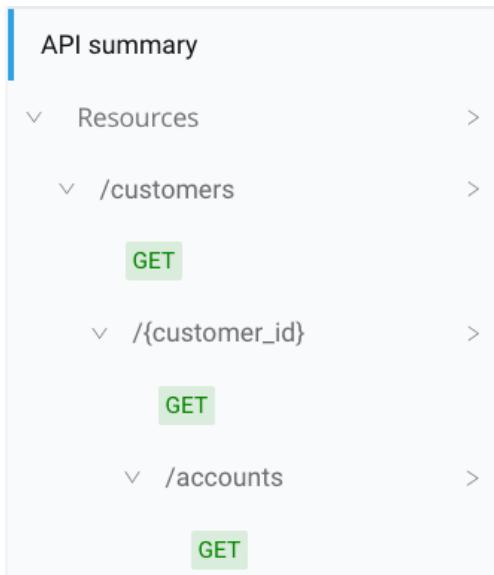
31. Delete the new line that is automatically added below the get method.
32. Similarly, add get methods to the nested /{customer_id} and /accounts resources.
33. Delete any extra empty lines.

```

1  #%RAML 1.0
2  title: ACME Banking API
3
4  /customers:
5      get:
6      /{customer_id}:
7          get:
8          /accounts:
9              get:
```

34. In the API Console, click the top left menu icon; you should see the links for the root resource methods in the Resources section.

35. Click the arrow next to `/{customer_id}` nested resource; you should see the links to the nested resource get methods.



36. Add get methods to the nested `/{account_id}` and `/transactions` resources.

37. For the `/transactions` resource, add a get method to the nested `/{transaction_id}` resource.

```
11 /accounts:  
12   /{account_id}:  
13     get:  
14   /transactions:  
15     get:  
16  
17   /transactions:  
18     /{transaction_id}:  
19       get:
```

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

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

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

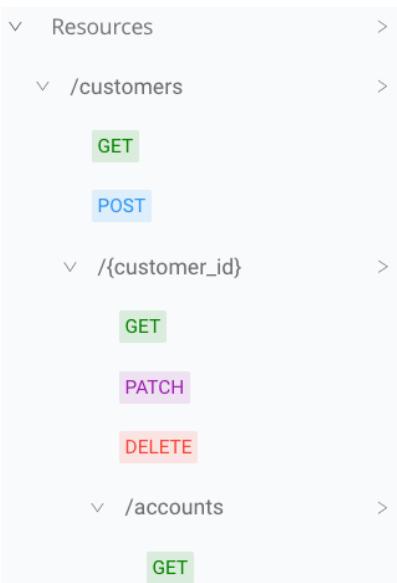
39. Review the methods for the `/customers` resource and nested resources.

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

40. Return to API designer.
41. In the API designer, type or use the shelf to add a post method to the /customers resource.
42. Add patch and delete methods to the /{customer_id} nested resource.

```
4  ⊞ /customers:  
5      get:  
6      post:  
7  ⊞  /{customer_id}:  
8      get:  
9      patch:  
10     delete:  
11  ⊞  /accounts:  
12      get:
```

43. In the API Console, click the top left menu icon; you should see the links for the new methods in the Resources section.



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

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

ACCOUNTS: Resource /accounts	Resource /accounts	Method POST
i. Create a new account		
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

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

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

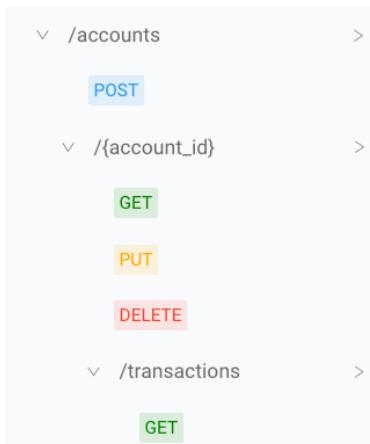
46. Return to API designer.

47. In the API designer, add a post method to the /accounts resource by using the auto-complete or by typing the node.

48. Add put and delete methods to the /{account_id} nested resource.

```
14  /accounts:  
15    post:  
16    /{account_id}:  
17      get:  
18      put:  
19      delete:  
20      /transactions:  
21        get:
```

49. In the API Console, click the top left menu icon; you should see the links for the new methods in the Resources section.



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

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

TRANSACTIONS: Resource /transactions	Resource /transactions	Method POST
i. Create a new transaction	Resource /transactions/{transaction_id}	Method GET
ii. Get transaction information for a specific transaction ID		

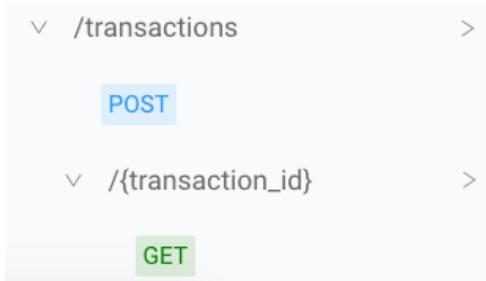
51. Review the methods for the /transactions resource and nested resources.

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

52. Return to API designer.
53. Add a post method to the /transactions resource.

```
23   /transactions:  
24     post:  
25    /{transaction_id}:  
26     get:
```

54. In the API Console, click the top left menu icon; you should see the links for the new methods in the Resources section.



Module 5: Specifying Responses

```
1  #%%RAML 1.0
2  title: ACME Banking API
3  mediaType: application/json
4
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:
18        404:
19          body:
20            properties:
21              statusCode: string
22              message: string
23        406:
24          body:
25            properties:
26              statusCode: string
27              message: string
28    post:
29      body:
30      responses:
31        201:
```

The screenshot shows a RAML 1.0 specification interface. On the left, the RAML code defines a GET and a POST method for the '/customers' endpoint. The POST method returns a 201 status code with a required 'Location' header and a body of type application/json. On the right, the interface displays the '/customers : post' endpoint. It shows the 'Request' (POST /customers), the 'Body' (Type application/json), the 'Response' (201, Type application/json), and the 'Response headers' (Location, string). A 'Try it' button is also visible.

/customers : post

Request

POST /customers

Body

Type application/json

Response

201

Response headers

Parameter	Type	Description
Location	string (required)	

Type application/json

Try it

Objectives:

- Create 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:

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

```
1  #%RAML 1.0
2  title: ACME Banking API
3  mediaType: application/json
4
5  /customers:
6    get:
7      headers:
8        Accept?:
9      responses:
10        200:
11          headers:
12            Cache-Control:
13            Expires:
14          body:
15            application/json:
16            application/xml:
17        404:
18          body:
19            properties:
20              statusCode: string
21              message: string
22        406:
23          body:
24            properties:
25              statusCode: string
26              message: string
27    post:
28      body:
29      responses:
30        201:
```

The screenshot shows the RAML API designer interface. On the left, the RAML code defines a resource at '/customers' with a 'post' method. The 'post' method has a 'body' node and a 'responses' node containing a single '201' entry. On the right, the interface displays the '/customers : post' endpoint. It shows a 'Request' section with a 'POST /customers' button. Below it is a 'Body' section with a 'Type application/json' dropdown. Underneath is a 'Response' section for the '201' status code, which includes a 'Response headers' table and a table for parameters. The parameter table has one row with 'Location' as the name, '(required)' as the type, and a 'Type application/json' dropdown. At the bottom right is a 'Try it' button.

Add a mediaType node to indicate the request and response body type

1. Return to API designer.
2. Add a new line after the title node at the beginning of the RAML definition.

3. From the shelf, add the mediaType node.

```
1  #%RAML 1.0
2  title: ACME Banking API
3
4
5  /customers:
6    get:
    | annotationTypes
```

Docs	Root
description	version
documentation	baseUri
	protocols
	mediaType

4. Add a space after the colon in the mediaType node.

5. From the shelf, click application/json.

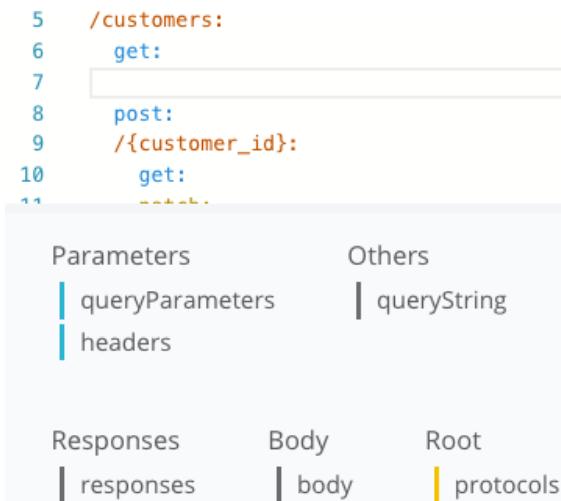
```
1  #%RAML 1.0
2  title: ACME Banking API
3  mediaType:
4
5  /customers:
6    get:
```

Others
application/json
application/xml
application/x-www-form-urlencoded
multipart/form-data

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

6. Add a new line under the /customers resource get method.
7. Press the tab key to indent.

8. From the shelf, add the responses node.



9. In the new line, add the HTTP status code 200 node.
10. Add a body node inside the HTTP 200 response.
11. Copy the lines from responses node till the body node.
12. Paste the copied lines in the get methods of the following resources:

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

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

13. Similarly, paste the copied lines inside the delete methods of the following resources:
- /customers/{customer_id}
 - /accounts/{account_id}

Add a request body to all POST methods

14. In the API designer, add a new line below the /customers post method.
15. Press the tab key to indent.

16. Add a body node.
17. Similarly, add the request body to the post methods of the following main resources:

- /accounts
- /transactions

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

18. Add a HTTP 201 response body to the post method of the /customers resource.
19. Add a new line above the 201-response body and indent to the same level.
20. In the shelf, click headers.

```
11      body:  
12      responses:  
13          201:  
14  
15          body:  
16          /{customer_id}:  
17              get:  
18          responses:  
19              200:  
  
Parameters Response  
| headers | body
```

21. In the shelf, scroll and click the Location header.
22. Delete the new line created below the Location line.
23. Copy the lines in the post method starting from the HTTP responses till the HTTP 201 response body.
24. Paste the copied lines in the following resources' post method:

- /accounts
- /transactions

Add a request body and a HTTP 200 and 201 response body for the PUT method

25. In the /{account_id} nested resource put method, add a request body.

```
46      put:  
47          body: |  
48          delete:
```

26. Below the request body, add a HTTP 200 response body.

```
46      put:  
47          body:  
48          responses:  
49              200:  
50                  body:  
51          delete:
```

27. Add a second status code to the response with a value of 201 with Location header in the response.

```
46      put:  
47          body:  
48          responses:  
49              200:  
50                  body:  
51              201:  
52                  headers:  
53                      Location:  
54                  body: |  
55          delete:
```

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

28. In the /{customer_id} nested resource patch method, add a request body.

```
22      patch:  
23          body: |  
24      delete:
```

29. Add a new line after the patch request body and indent to the same level as the body node.

30. In the shelf below, click responses.

31. In the shelf, click 204.

32. Delete the new line created below the HTTP status code 204 line.

View the API Console

33. In the API Console click the top left menu icon, and under the Resources section, locate the /accounts/{account_id} resource.

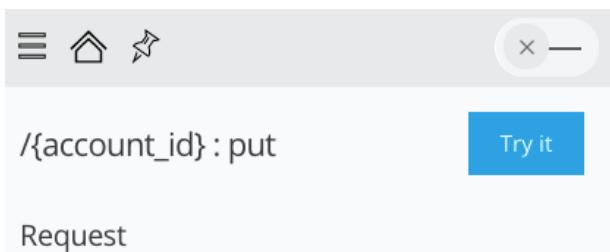
34. Click the PUT method link.

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

Response

HTTP Status Code	Description
200	No description provided.
201	Type: application/json <i>This object doesn't have a type description.</i>

36. Click the menu icon in the top left corner of API Console to go back to the API summary.



The screenshot shows the top navigation bar of the API Console. On the left is a menu icon (three horizontal lines). Next to it are icons for home and a star. To the right is a circular button with an 'X' and a minimize/maximize window icon. Below the bar, the URL '{account_id} : put' is displayed, followed by a blue 'Try it' button. At the bottom left, the word 'Request' is visible.

Walkthrough 5-2: Add responses bodies to return custom error information for client-side errors

In this walkthrough, you add response bodies to return a custom error information objects for client-side errors. You will:

- Add HTTP 4xx status code responses to all GET and DELETE methods.
- Create a custom error information object to be returned in the response body.

The screenshot shows the API designer interface. On the left, a code editor displays the following JSON schema for a DELETE method:

```
83  delete:
84    responses:
85      200:
86        body:
87      404:
88        body:
89          properties:
90            statusCode: string
91            message: string
```

On the right, a "Response" panel is open, showing the 404 response details:

- 200**: No description provided.
- 404**: Type: application/json

The 404 response body is defined as:

```
{
  "statusCode": "string",
  "message": "string"
}
```

Parameter	Type	Description
statusCode*	string	
message*	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 after HTTP 200 response body in the get method.
3. Press backspace or delete to go back a tab space, 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.

```
5  /customers:
6    get:
7      responses:
8        200:
9          body:
10         4 ~
11        post:
12          body:
13        responses:
| 401
| 402
| 403
| 404
```

5. In the shelf, click body.

```
9      body:  
10     404:  
11  
12     post:  
13       body:  
14       responses:  
15         201:  
  
Parameters Response  
headers   body
```

Create a custom object to display an error message for the response

6. In the new line, type:

```
properties:
```

7. Press enter and press tab to indent the line of code.

8. In the new line below properties, type:

```
statusCode:
```

9. Press spacebar, and scroll down in the shelf and click string.

```
11      body:  
12        properties:  
13          statusCode: |  
14      post:  
15      body:  
  
union  
object  
string  
boolean
```

10. Press enter and type:

```
message: string  
10      404:  
11        body:  
12          properties:  
13            statusCode: string  
14            message: string |  
15      post:
```

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

11. Copy the HTTP 404 response body lines.

```
7     responses:
8       200:
9         body:
10      404:
11        body:
12          properties:
13            statusCode: string
14            message: string
15      post:
```

12. Add the HTTP 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}

Note: Make sure to fix the indentation after pasting these lines under the resource method.

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

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

```
83  delete:
84    responses:
85      200:
86        body:
87      404:
88        body:
89          properties:
90            statusCode: string
91            message: string|
```

View the API Console

14. In the API Console click the top-left menu icon, and in the Resources section, click the GET for the /customers resource.

15. In the Response section click the 404 tab; you should see the custom error message properties listed under the application/json object.

Response

200	No description provided.	
404	Type: application/json	
{ "statusCode": "string", "message": "string" }		
Parameter	Type	Description
statusCode*	string	
message*	string	

16. Go back to the list of resources by clicking the top-left menu icon.

Walkthrough 5-3: Add response bodies to return custom error information 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 HTTP 5xx status code responses to all PATCH, PUT and POST methods.
- Create a custom error message object to be returned in the response body.

```
79      put:
80          body:
81          responses:
82              200:
83                  body:
84              201:
85                  headers:
86                      Location:
87                  body:
88              501:
89                  body:
90                      properties:
91                          statusCode: st
92                          message: strin
```

Parameter	Type	Description
statusCode*	string	
message*	string	

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 HTTP 204 status code in the patch method.
3. Type 5, then scroll through the shelf to locate 501 and click it.

```
32      patch:
33          body:
34          responses:
35              204:
36              5|
37          delete:
```

426
500
501
502

4. In the shelf, click body.

Create a custom object to display an error message for the response

5. In the new line, type:

```
properties:
```

6. In the new line below properties, type:

```
statusCode:
```

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

```
38      properties:  
39      statusCode: |  
40    delete:  
41    responses:  
| union  
| object  
| string  
| boolean
```

8. Press enter and type:

```
message: string
```

Add the HTTP 501 response body to the PUT method

9. Copy the HTTP 501 response body lines.

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

```
79    put:  
80      body:  
81      responses:  
82        200:  
83          body:  
84        201:  
85          headers:  
86            Location:  
87          body:  
88        501:  
89          body:  
90            properties:  
91              statusCode: string  
92              message: string |
```

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

11. In the /customers resource, add a new line under HTTP 201 response body in the post method.
12. Press backspace or delete to go back one tab space, so the cursor is at the same indent level as the HTTP 201 status code.
13. Type 5, then click 503 in the shelf.

```
20      Location:  
21      body:  
22      5  
23  /{customer_id}:  
24    get:  
25    responses:  
26      200.  
      501  
      502  
      503
```

14. In the shelf, click body.

15. In the new line, type:

properties:

```
20      Location:  
21      body:  
22      503:  
23      body:  
24      properties:  
25  /{customer_id}:  
      501  
      502  
      503
```

16. Press enter and press tab to add an extra indentation in the new line and type:

statusCode:

17. Press spacebar and scroll down the shelf and select string.

18. Press enter and type:

message: string

```
22      503:  
23      body:  
24      properties:  
25      statusCode: string  
26      message: string  
27  /{customer_id}:  
      501  
      502  
      503
```

Add the HTTP 503 response body to all POST methods

19. Copy the HTTP 503 response body lines.
20. Add the HTTP 503 response body to the post methods of the following resources:
 - /accounts
 - /transactions

View the API Console

21. In the API Console, go to the API summary by clicking the menu icon in the top left corner.
22. Click the POST method link for the /customers resource.
23. Scroll down to the Response section and click the 503 tab.

Note: You should see the custom error message properties listed under the application/json object.

Response

201	No description provided.	
503	Type: application/json	
{ "statusCode": "string", "message": "string" }		
Parameter	Type	Description
statusCode*	string	
message*	string	

24. Go back to the API summary by clicking on the top-left menu icon.

Walkthrough 5-4: Add flexible content-type support to a resource method

In this walkthrough, you add flexible content-type to HTTP response of a resource method.

You will:

- Add XML body type to the HTTP 200 response of a resource method.
- Add an optional accept header to the request to specify the type of response accepted by the client.
- Add a relevant HTTP status code for client-side error when an unsupported type is requested.

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

The screenshot shows the API designer interface. On the left, there's a code editor with the following snippet:

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

On the right, there's a detailed view of the 200 response. It shows a JSON schema with two fields: "statusCode" and "message". Below the schema, there's a table with columns "Parameter" and "Type". The "statusCode*" row has "string" in the Type column. The "message*" row also has "string" in the Type column.

A context menu is open over the 200 response area. The menu has a "Body type" submenu with "application/json" and "application/xml" options. "application/xml" is highlighted, indicating it's the current selection.

Add XML response media type to the HTTP 200 responses body for /customers GET method

1. Return to API designer.
2. In the /customers resource, add a new line below HTTP 200 response body.
3. Press tab and in the shelf, click application/json.
4. In the new line, press the backspace/delete key to go back one tab space.
5. In the shelf, click application/xml.
6. Delete the new line created below application/xml.

```
8      200:  
9        body:  
10       application/json:  
11       application/xml:  
12      404:
```

View the API Console

7. In the API Console, click the top-left menu icon to open the list of resources.
8. Click the GET method for the /customers resource.

The screenshot shows the 'API summary' section of an API console. Under the 'Resources' category, there is a node for '/customers'. Below this node, two methods are listed: 'GET' (highlighted in green) and 'POST' (highlighted in blue).

9. Scroll down to the Response section and in the HTTP status 200 body type click the drop-down menu next to application/json; you should see both application/json and application/xml listed as possible return types.

The screenshot shows the 'Response' section for a 200 status code. The description says 'No description provided.' Below it, there is a dropdown menu for 'Body type' with 'application/json' selected. Another option, 'application/xml', is also visible. A note at the bottom states 'This object doesn't have a type description.'

Add an optional accept header to specify the media type of response body accepted by the client

10. In the editor, go to the /customers resource and add a new line below the get node.

11. Press tab in the new line and in the shelf, click headers.

```
5   /customers:  
6     get:  
7       |  
8       responses:  
9         200:  
10        body:
```

Parameters

```
queryParameters  
headers
```

12. Scroll through the shelf, and click 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. It also indicates that the client accepts all media type responses.

View the API Console

15. In the API Console, click the top-left menu icon.

16. In the Resources section, click GET for the /customers resource.

17. In the Headers section for the request, locate the Accept header.

Request

GET /customers

Headers

Hide ^

Parameter	Description
Accept	string

Add a HTTP 406 response to the resource method to indicate error when an unsupported type is requested

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

406:

```
16      properties:  
17          statusCode: string  
18          message: string  
19      406:  
20      post:
```

21. Press enter and press tab.
22. In the shelf, click body.
23. In the new line type:

properties:

24. Press enter and press tab.
25. In the new line, add a property called statusCode of type string.
26. Press enter.
27. In the new line, add a property called message of type string.

```
19      406:  
20          body:  
21              properties:  
22                  statusCode: string  
23                  message: string  
24      post:
```

View the API Console

28. In API Console, click the top-left menu icon
29. In the Resources section, click GET for the /customers resource.
30. In the Response section, click the 406 tab to view the HTTP 406 response body.

Response

Parameter	Type	Description
statusCode*	string	
message*	string	

31. Go back to the list of resources by clicking on the top-left menu icon.

Walkthrough 5-5: Add caching information to HTTP responses

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.

```
1  #%RAML 1.0
2  title: ACME Banking API
3  mediaType: application/json
4
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:
18        404:
19          body:
20            properties:
21              statusCode: string
22              message: string
23        406:
```

Others
| Define Inline
| array
| union
| object
| string

The screenshot shows the MuleSoft API Designer interface. On the left, the RAML code defines a GET method for the '/customers' endpoint. In the middle, the 'Headers' section of the API configuration is shown, with an 'Accept' parameter defined. On the right, the 'Response' section shows response headers for status codes 200, 404, and 406. For status code 200, the 'Cache-Control' header is defined as a required string, and the 'Expires' header is defined as a required datetime.

Parameter	Type	Description
Accept	string	

Response headers	Parameter	Type	Description
200			
404			
406	Cache-Control (required)	string	
	Expires (required)	datetime	

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 with HTTP 200 node.
3. Press tab and in the shelf, click headers.

```
10  200:
11
12    body:
13      application/json:
14      application/xml:
```

Parameters Response Docs
 headers body description

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

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

5. In the editor, go to the new line after the Cache-Control header.
6. In the new line, align the cursor below the start of the Cache-Control line.

```
10      200:  
11          headers:  
12              Cache-Control:  
13  
14      body:
```

7. In the shelf, click Expires.
8. In the shelf, click type.

```
12      Cache-Control:  
13          Expires:  
14  
15      body:
```

Docs	Others
displayName	facets
example	type

9. In the shelf, scroll down and click datetime.

```
11          headers:  
12              Cache-Control:  
13                  Expires:  
14                      type: datetime  
15          body:
```

Others
datetime
datetime-only

View the API Console

10. In the API Console, click the top-left menu icon.
11. Under the Resources section, click GET for the /customers resource.
12. In the Response section, locate the Expires and Cache-Control headers in the HTTP Status 200 response.
13. Go back to the list of resources by clicking on the top-left menu icon in the API Console.

Module 6: Modelling Data

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

```
Properties
1
2  accountBalance(required)  accountBalance
3
4
5  accountBalance. currency(required)
6
7    Validation pattern: ^[A-Z]{3,3}$
8
9
10   accountBalance. amount(required)
11
12    Validation pattern: ^[+-]?\d*\.\d{2}$
13
14   accountNumber(required)  accountNumber
15
16   accountOwner(required)   accountOwner
17
18
19  accountOwner. customerID(required)
```

```
1  #%RAML 1.0 NamedExample
2  value:
3    accountID: '12345'
4    accountType: Savings
5    accountNumber: '1234567890'
6    accountOwner:
7      - #item 1
8        customerID: 8f19cb50-3f57-4d38
9        displayName: John Doe
10       ssn: 123-456-7890
11
12       accountBalance:
13         currency: USD
14         amount: '8457.90'
15
16       IBAN: GB29NWBK60161331926820
17
18       bank:
19         bankCode: NWBKGB2L
20         bankName: ACME Bank
21         routingNumber: '432159876'
22
23       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 datatype fragments.

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

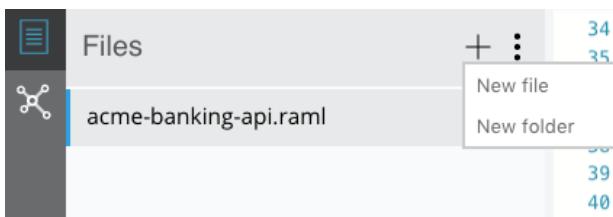
In this walkthrough, you define datatype fragments for the identified datatypes for the ACME Banking API. You will:

- Create datatype fragment files for the identified datatypes.
- Define the datatypes with required and optional attributes.
- Include the datatype fragments in the main RAML API definition.

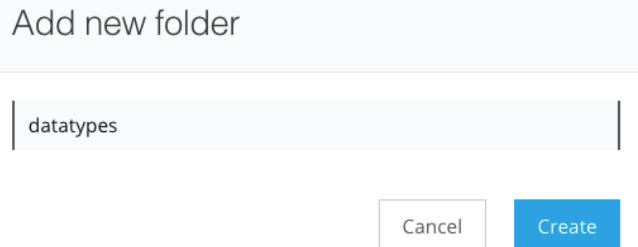
```
1  %%RAML 1.0 DataType          1  %%RAML 1.0
2  type: object                2  title: ACME Banking API
3  properties:                 3  mediaType: application/json
4  customerID: string          4
5  prefix?: string              5  types:
6  firstName: string            6    Customer: !include datatypes/Customer.raml
7  lastName: string              7    Address: !include datatypes/Address.raml
8  suffix?: string              8    Account: !include datatypes/Account.raml
9  displayName: string           9    AccountOwner: !include datatypes/AccountOwner.raml
10 address: Address             10   Bank: !include datatypes/Bank.raml
11 phone: string                11   Money: !include datatypes/Money.raml
12 email: string                12   Transaction: !include datatypes/Transaction.raml
13 ssn: string                  13   CustomErrorMessage: !include datatypes/CustomErrorMessage.raml
14 dateOfBirth: date-only
```

Refactor the custom error message object to use a datatype

1. Return to API designer.
2. In the file browser section, click the + icon in the Files header section.
3. In the drop-down menu click New folder.



4. In the Add new folder dialog box, set the name to datatypes.



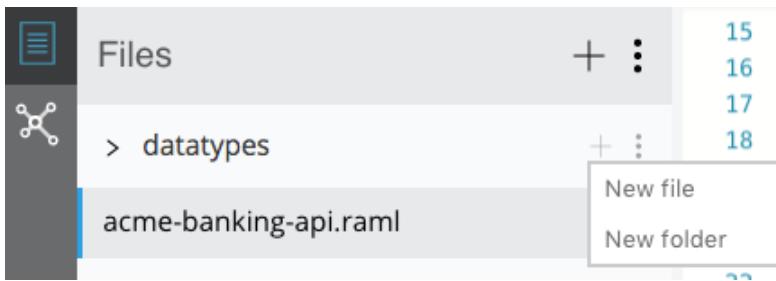
5. Click Create.
6. In the file browser, click acme-banking-api.raml.
7. Go to the /customers resource get method HTTP 404 response body and copy the lines that define the properties of the custom error message.

```

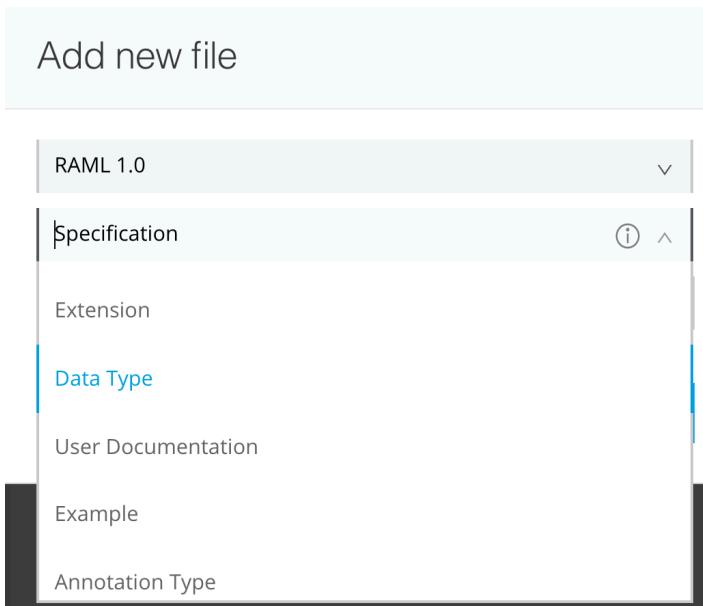
19      404:
20          body:
21              properties:
22                  statusCode: string
23                  message: string
24      406:

```

8. In the file browser section, hover the mouse pointer along the datatypes folder; you should see an + icon and a menu icon to the right of the folder name.
9. Click the + icon and click New file from the drop-down menu.



10. In the Add new file dialog box, click the name drop-down menu next to Specification textbox.
11. Scroll down and click Data Type.



12. Rename the file as CustomErrorMessage.raml.

13. Click Create.

14. Go to the end of the first line in the CustomErrorMessage file and press enter.

15. Type the following:

```
type: object
```

16. Press enter and paste the copied lines below.

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

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

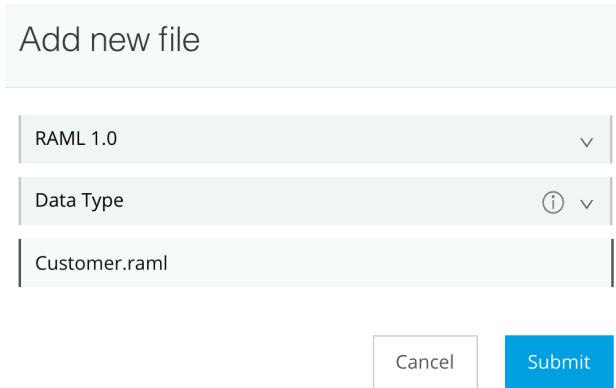
```
1  #%RAML 1.0 DataType
2  type: object
3  properties:
4      statusCode: string
5      message: string
```

Create a Customer datatype fragment

18. In the file browser section, click the + icon next to the datatypes folder and click New file.

19. In the Add new file dialog box, set the value of the field which reads Specification to Data Type.

20. Rename the file as Customer.raml.



21. Click Create.

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

23. In the new line add the type node with the value object.

```
1  #%RAML 1.0 DataType  
2  type: object
```

24. In the shelf, click properties.

```
1  #%RAML 1.0 DataType  
2  type: object  
3
```

Docs	Others
displayName	facets
example	examples
description	xml
	isAnnotation
	properties
	minProperties

25. Return to the WT 6-1 Datatypes and attributes file in your computer.

26. Return to API designer.

27. 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.

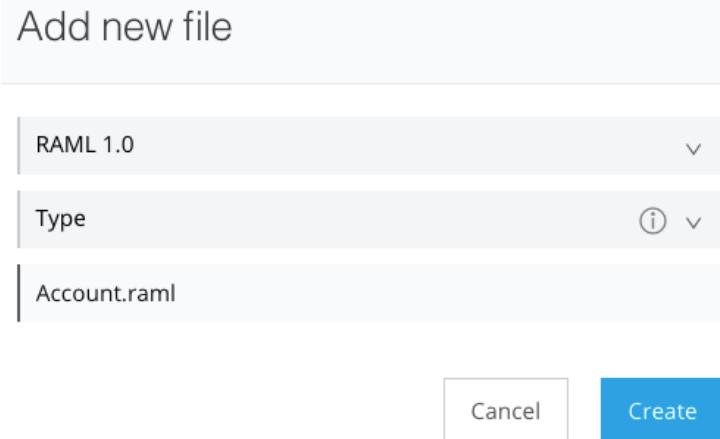
28. Add a ? after the prefix and suffix attribute names to specify them as optional attributes.

```
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 bar in the right side of 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.

Create an Account datatype fragment

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



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

- Use the enum datatype to specify the accountType that has possible values of Checking, Savings, Overdraft Savings, Credit Card.

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

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

Create a Transaction datatype fragment

31. Similarly, create another new Type file called Transaction.raml and type the required and optional attributes for the Transaction datatype.

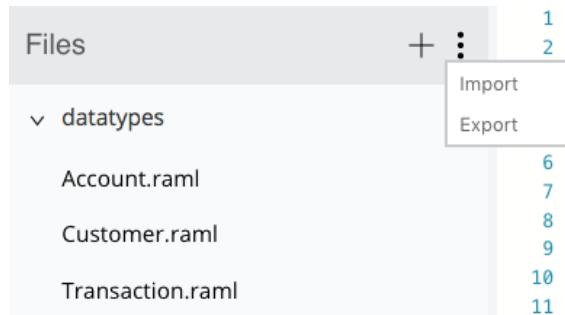
32. Use the enum datatype to specify the transactionType has possible values of atm, deposit, cashWithdrawal, sepa, and online.

```
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.

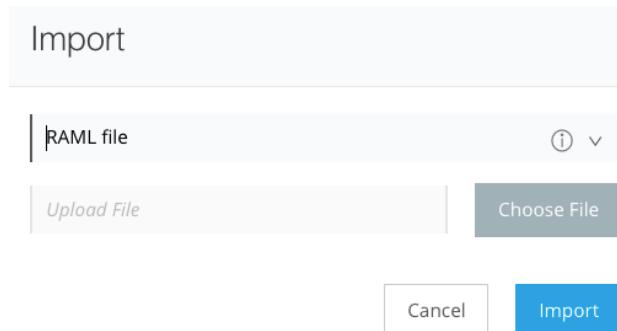
Import other datatype fragment files into API Designer

33. In the file browser section, click the menu icon on the top right next to the + icon in the Files menu bar.



34. Click Import.

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



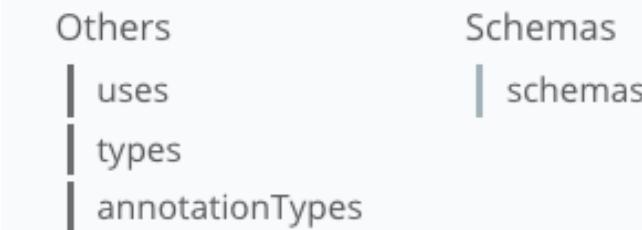
36. In the file chooser, locate the studentFiles directory, open the module06 folder, and double-click Address.raml.
37. Click Import.
38. In the left-side navigation, click Address.raml and drag-and-drop it inside the datatypes folder.
39. Similarly, import Money.raml, AccountOwner.raml and Bank.raml and drag them into the datatypes directory.
40. Verify the project structure has the datatype files inside the datatypes folder, and the acme-banking-api.raml file in the root location.

Include the datatype fragments in the main RAML API definition

41. In the file browser section, select acme-banking-api.raml.
42. In the RAML editor, go to the end of the line that contains the mediaType node and add two new lines below it.
43. Go to the second new line and in the shelf, click types.

```

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



44. In the new line type,

Customer: !

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

45. In the shelf, click include.

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

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

```
6  types:  
7      Customer: !include datatypes/  
8  
9  /customers:  
10     get:  
11         headers:
```

Others

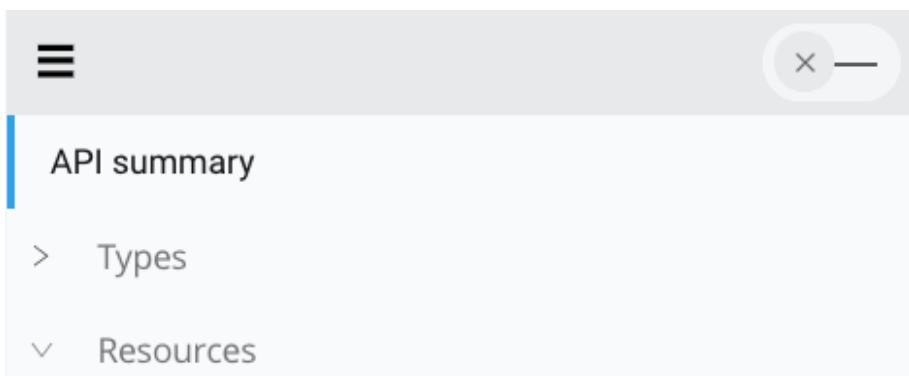
- Account.raml
- Customer.raml
- Transaction.raml

48. Similarly, include rest of the datatypes in the acme-banking-api.raml file.

View the API Console

49. In the API Console, click the top-left menu icon.

50. Click the arrow in the Types section to expand it.



51. Verify that you see all the datatypes listed.

The screenshot shows a sidebar titled "API summary" with a blue vertical bar on the left. Below it is a list of data types:

- ✓ Types
 - Customer
 - Address
 - Account
 - AccountOwner
 - Bank
 - Transaction
 - Money
 - CustomErrorMessage

52. Click the Customer datatype to view its properties.

The screenshot shows a modal window for the "Customer" type. At the top is a header with three horizontal bars. Below it is a title "Type Customer". The main content area contains a JSON object representing the Customer type:

```
{  
    "suffix": "string",  
    "address": {  
        "addressLine1": "string",  
        "addressLine2": "string",  
        "city": "string",  
        "zipCode": "string",  
        "state": "string",  
        "country": "string"  
    },  
    "email": "string",  
    "displayName": "string",  
    "phone": "string",  
    "ssn": "string",  
    "firstName": "string",  
    "prefix": "string",  
    "dateOfBirth": "date-only",  
    "lastName": "string",  
    "customerID": "string"  
}
```

Below the JSON is a section titled "Properties" with a table:

suffix	string
address(required)	address

Walkthrough 6-3: 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 response bodies.

```
120      put:  
121          body:  
122          responses:  
123              200:  
124                  body:  
125                      type: Account
```

The screenshot shows the API designer interface. On the left, there is a code editor with the following JSON-like code:

```
200  No description provided.  
404  Type: Account  
  
{  
    "accountBalance": {  
        "currency": "string",  
        "amount": "string"  
    },  
    "accountNumber": "string",  
    "accountOwner": [  
        {  
            "customerID": "string",  
            "ssn": "string",  
            "displayName": "string"  
        }  
    ],  
    "interestRate": "number",  
}
```

On the right, there is a 'Response' panel with two tabs: '200' and '404'. The '200' tab is selected and contains the text 'No description provided.' and 'Type: Account'. The '404' tab is also visible. Below the tabs, there is a preview of the JSON structure.

Add type parameters for the appropriate method requests and responses

1. Return to acme-banking-api.raml in API designer.
2. Locate the /customers resource and in the get method HTTP 200 response body, add a new line below the response body type application/json.
3. Press tab and in the shelf, click type.

```
25          type: datetime  
26          body:  
27              application/json:  
28  
29          application/xml:  
30          application/raml:
```

Docs	Others
displayName	facets
example	type
description	examples

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

```
26      body:  
27          application/json:  
28              type:  
29          application/xml:  
30      ...
```

Others

Customer
Address

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

```
21      200:  
22          headers:  
23              Cache-Control:  
24              Expires:  
25                  type: datetime  
26          body:  
27              application/json:  
28                  type: Customer[]  
29              application/xml:  
30                  type: Customer[]
```

8. 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.

```
81      /accounts:           139      /transactions:  
82          get:             140          get:  
83          responses:       141          responses:  
84              200:            142          200:  
85                  body:        143          body:  
86                  type: Account[] 144          type: Transaction[]  
87          404:             145          404:
```

9. In the /customers resource post method, add a new line below the request body media type.

10. Press tab and type:

```
type: Customer  
41   post:  
42     body:  
43       type: Customer  
44     responses:  
45       201:  
46         headers:  
47           Location:  
48         body:
```

11. Add the request body for the /accounts post method, and the /transactions post method with the respective datatypes in the request.

```
94  /accounts:  
95    post:          152  /transactions:  
96      body:          153  post:  
97        type: Account 154  body:  
98      responses:    155  type: Transaction  
99        201:          156  responses:  
100       headers:     157  201:
```

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

13. Add a new line below the HTTP 200 response body media type.

14. Press tab and type:

```
type: Customer  
54  /{customer_id}:  
55    get:  
56      responses:  
57        200:  
58          body:  
59            type: Customer
```

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

```
109  /{account_id}:          168  /{transaction_id}:  
110    get:                  169  get:  
111      responses:        170  responses:  
112        200:              171  200:  
113          body:           172  body:  
114            type: Account 173  type: Transaction
```

17. Add the type node for the /{account_id} put method request body with the value as the Account datatype.

```
120      put:  
121          body:  
122          responses:  
123              200:  
124                  body:  
125                      type: Account
```

View the API Console

16. In the API Console, click the top left menu icon to view the list of resources.
17. In the Resources section, click the GET method for the /accounts/{account_id} resource.
18. Verify that you can see the Account object listed in the 200 Response section of the resource.

The screenshot shows the API Console interface. At the top, there's a navigation bar with a menu icon and close/minimize buttons. Below it, a sidebar on the left is labeled "Response". The main content area displays a 200 response with the following details:

- 200**: No description provided.
- 404**: Type: Account
- The response body is a JSON schema for an Account object, starting with a brace { and containing fields like accountBalance, accountNumber, accountOwner, and interestRate.

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

19. In the RAML editor, go to the /customers resource get method HTTP 404 response body and delete the lines that contain the custom error object properties.

20. Set the datatype to CustomErrorMessage.

```
type: CustomErrorMessage  
31      404:  
32          body:  
33              type: CustomErrorMessage  
34      406:  
35          body:
```

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

Walkthrough 6-4: 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}$
```

Properties	
accountBalance(required)	accountBalance
accountBalance.currency(required)	string
Validation pattern:	^[A-Z]{3,3}\$
accountBalance.amount(required)	string
Validation pattern:	^[+ -]?\d*\.\d{2}\$
accountNumber(required)	string

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.

```
10  IBAN:
11
12  bank: Bank
13  interestRate?: number
14  createdAt: datetime
15  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
```

Docs	Others
displayName	facets
example	examples
description	xml
	isAnnotation
	uses
	pattern

7. Return to or open the course snippets.txt file.
8. Locate the Module 6 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}$  
13     bank: Bank
```

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

type: number

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. In the file browser section, click Money.raml.
15. Add a new line below the currency type node line.

```
1  %%RAML 1.0 DataType  
2  type: object  
3  properties:  
4    currency:  
5      type: string  
6      |  
7    amount:  
8      type: string
```

16. In the shelf, click pattern.

```
1  #%RAML 1.0 DataType
2  type: object
3  properties:
4    currency:
5      type: string
6
7    amount:
8      type: string
```

displayName	facets
example	examples
description	xml
	isAnnotation
	uses
	pattern

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

```
4  currency:
5    type: string
6    pattern: ^[A-Z]{3,3}$
7  amount:
```

18. Add a new line below the amount type node line.

19. In the shelf, click pattern.

20. 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}$
```

View the API documentation

21. In the file browser section, click acme-banking-api.raml.

22. In the API Console, click the top-left menu icon.

23. Expand the Types section; you should see all the datatypes listed.

The screenshot shows a sidebar menu with the following structure:

- API summary
- Types
 - Customer
 - Address
 - Account
 - AccountOwner
 - Bank
 - Transaction
 - Money
 - CustomErrorMessage
- Resources

24. Click the Account datatype and scroll down to the Properties section.

25. Verify that you can view the patterns enforced for the accountBalance currency and amount properties.

The screenshot shows the properties section for the Account datatype, specifically focusing on the accountBalance property.

Properties

accountBalance(required) **accountBalance**

accountBalance. currency(required) **string**

Validation pattern: `^[A-Z]{3,3}$`

accountBalance. amount(required) **string**

Validation pattern: `^[-+]?[0-9]*\.[0-9]{2}$`

accountNumber(required) **string**

accountOwner(required) **AccountOwner[object]**

accountOwner. customerID(required) **string**

26. Click the top left menu icon in the API Console to view the list of datatypes and resources.

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 response bodies.

```
1  #%RAML 1.0 NamedExample
2  value:
3      transactionID: b05f550d-1915-4def
4      fromAccount:
5          accountID: '12345'
6          accountType: Savings
7          accountNumber: '1234567890'
8          accountOwner:
9              -
10                 customerID: 8f19cb50-3f57-4d38
11                 displayName: John Doe
12                 ssn: 123-456-7890
13             accountBalance:
14                 currency: USD
15                 amount: '8457.90'
16             IBAN: GB29NWBK60161331926820
17             bank:
18                 bankCode: NWBKGB2L
19                 bankName: ACME Bank
20                 routingNumber: '432159876'
21             createdAt: 2012-03-07T00:00:00.001Z
22             toAccount:
23                 accountID: '56437'
24                 accountType: Credit Card
25                 accountNumber: '4321987650'
26                 accountOwner:
```

Response

200	No description provided.
404	Type: Transaction
Type Examples	
 	
transactionID	b05f550d-1915-4def
fromAccount(Object)	
accountID	12345
accountType Savings	
accountNumber	1234567890
accountOwner(Array 1)	
customerID	8f19cb50-3f57-4d38
displayName	John Doe
ssn	123-456-7890

Create example fragment for the Customer datatype

1. Return to API designer.
2. In the file browser section, click the + icon next to the Files section header and select New folder.
3. In the Add a new folder dialog box, set the name of the folder to examples.

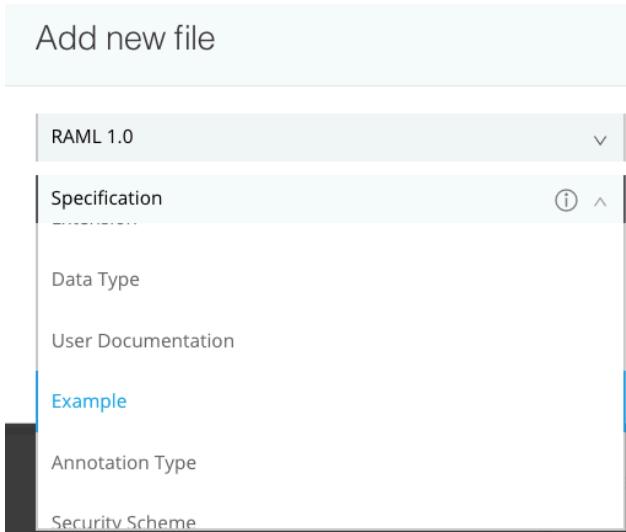
Add new folder

examples

Cancel

Create

4. Click Create.
5. Hover over the examples folder in the file browser section, and click the + icon
6. In the drop-down menu, select New file.
7. In the Add new file dialog box, click the drop-down arrow next to the field that contains the value Specification and select Example.



8. Set the name to CustomerExample.raml and click Create.
9. In the CustomerExample.raml file, add a new line below the value node and press tab.
10. Return to the course snippets.txt file and copy the snippet for the CustomerExample.raml content values and paste it in the new line.

```

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

```

Note: Fix the indentation to make sure all the attributes are one tab level inside the value node.

Create example fragment for the Account datatype

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

```
1  #%RAML 1.0 NamedExample
2  value:
3      accountID: '12345'
4      accountType: Savings
5      accountNumber: '1234567890'
6      accountOwner:
7          - #item 1
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: GB29NWBK60161331926820
15             bank:
16                 bankCode: NWBKGB2L
17                 bankName: ACME Bank
18                 routingNumber: '432159876'
19             createdAt: 2012-03-07T00:00:00.001Z
```

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  value:
3      transactionID: b05f550d-1915-4def
4      fromAccount:
5          accountID: '12345'
6          accountType: Savings
7          accountNumber: '1234567890'
8          accountOwner:
9              -
10             customerID: 8f19cb50-3f57-4d38
11             displayName: John Doe
12             ssn: 123-456-7890
13             accountBalance:
14                 currency: USD
15                 amount: '8457.90'
16             IBAN: GB29NWBK60161331926820
17             bank:
18                 bankCode: NWBKGB2L
19                 bankName: ACME Bank
20                 routingNumber: '432159876'
21             createdAt: 2012-03-07T00:00:00.001Z
22             toAccount:
23                 accountID: '56437'
24                 accountType: Credit Card
25                 accountNumber: '4321987650'
26                 accountOwner:
```

Include example fragments in the datatype fragments

13. In the file browser section, click acme-banking-api.raml.
14. Go to the /{customer_id} resource.
15. In the get method, add a new line below the HTTP 200 response body type node.
16. Type:

```
example: !include examples/CustomerExample.raml
49   /{customer_id}:
50     get:
51       responses:
52         200:
53           body:
54             type: Customer
55             example: !include examples/CustomerExample.raml
```

17. Similarly, include the AccountExample.raml example in the /{account_id} resource get method.

```
95   /{account_id}:
96     get:
97       responses:
98         200:
99           body:
100          type: Account
101          example: !include examples/AccountExample.raml
102          404:
```

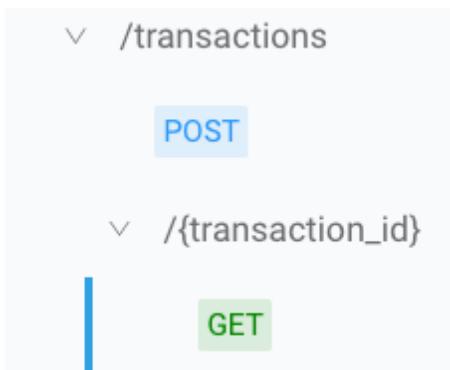
18. Similarly, add the TransactionExample.raml example in the /{transaction_id} resource get method.

```
147   /{transaction_id}:
148     get:
149       responses:
150         200:
151           body:
152             type: Transaction
153             example: !include examples/TransactionExample.raml
```

View the API Console

19. In the API Console, click the top left menu icon to go to the list of resources.

20. In the `/{transaction_id}` resource, expand the list of methods to click the GET method link.



21. In the Response section, click the Examples tab and verify that you see the TransactionExample.

Response

The screenshot shows the response details for the `GET /{transaction_id}` method. On the left, there are two status code sections: one for `200` (No description provided) and one for `404` (Type: Transaction). To the right of these are two tabs: `Type` and `Examples`. The `Examples` tab is currently selected. Below this, there is a preview area with icons for copy and download. Under the `transactionID` example, the value `b05f550d-1915-4def` is shown. The `fromAccount` example is expanded, showing fields `accountID` (value `12345`) and `accountType` (value `Savings`). The `accountNumber` example shows the value `1234567890`. The `accountOwner` example is expanded, showing a table with columns `customerID`, `displayName`, and `ssn`. One row in the table has values `8f19cb50-3f57-4d38`, `John Doe`, and `123-456-7890`.

22. Click the top left menu icon to go back to the list of resources.

Module 7: Documenting and Testing APIs

The screenshot shows a user interface for API documentation. On the left, there's a sidebar with navigation links: 'API summary', 'Documentation' (which is expanded), 'ACME Bank Headline' (highlighted with a blue vertical bar), 'ACME Banking API Home', 'Types' (which is collapsed), and 'Resources' (which is collapsed). The main content area has a header 'ACME Banking API' with a description: 'enables developers to build applications that make use of the information from resource methods implemented in the API.' Below this, under 'Documentation', is a section for the '/customers' endpoint. It includes a 'Description' box stating 'This API contains functionality that allows developers to retrieve and manipulate customer, account and transaction information.' and a code block for the 'get' method:

```
Expires:  
  type: datetime  
  description: |  
    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 be ignored.  
example: Tue, 17 Jul 2017 09:30:41 GMT  
format: rfc2616  
12  /customers:  
13    get:  
14      description: Retrieve a list of customers  
15      displayName: Get all customers  
16      headers:
```

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 API designer interface. On the left, there is a code editor with the following RAML 1.0 code:

```
1 %%RAML 1.0 DocumentationItem
2 title: ACME Banking API Home
3 content: |
4   **ACME Banking API** enables developers to build :
```

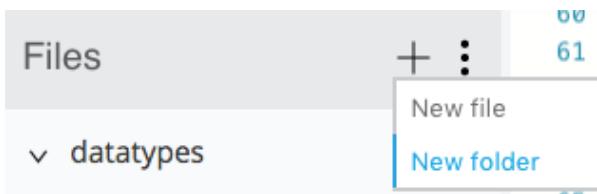
On the right, there is a detailed documentation fragment:

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

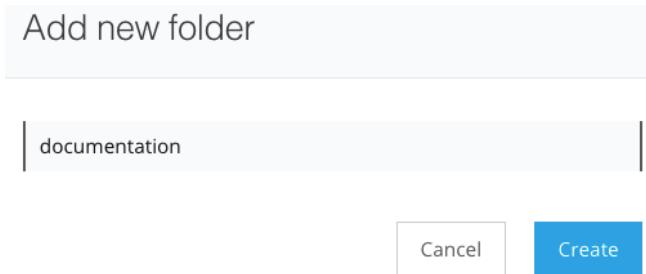
This API contains functionality that allows developers to retrieve and manipulate *customer*, *account* and *transaction* information. Check out the [API Portal](#) for more details.

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

1. Return to API designer.
2. In the file browser section, click the + icon in the Files menu bar.
3. Select New folder.



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



5. Click Create.
6. In the file browser section, click the + icon next to the documentation folder and select New file.
7. In the Add new file dialog box, click the drop-down arrow next to the Specification field.
8. Scroll down and select User Documentation.
9. Set the file name to acmeBankDoc.raml.
10. Click Create.
11. In the RAML editor, set the title property to ACME Banking API Home.

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

```

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

```

12. Add a space after the colon in the content node line and type a | (the pipe operator).

Note: The pipe operator indicates the value of the node is accepted as multi-line input.

13. Return to the course snippets.txt file.
14. 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

```

15. Return to API designer.
22. In acmeBankDoc.raml, press enter after the pipe operator in the content node line.
16. Press tab and paste the copied lines.

```

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

```

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

```

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 -

```

18. Return to API designer and add two new lines after the first snippet documentation.

19. Paste the copied text in the second new line.

```
1  #%RAML 1.0 DocumentationItem
2  title: ACME Banking API Home
3  content: |
4    **ACME Banking API** enables developers to build applications that make use of the information from resource
5
6    This API contains functionality that allows developers to retrieve and manipulate _customer_, _account_ and
```

Link the documentation fragment to the main RAML definition file

20. In the file browser section, click acme-banking-api.raml.

21. Go to the line of code before the types node and press enter to add a new line.

22. In the shelf, click documentation.

```
5  mediaType: application/json
6
7  |
8  types:
9    Customer: !include datatypes/Customer.raml
10   Address: !include datatypes/Address.raml
11   Account: !include datatypes/Account.raml
```

Security

securitySchemes
securedBy

Docs

description
documentation

Parameters

baseUriParameters

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

- !include

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

24. In the shelf, click documentation.

25. After the word documentation word, type /.

26. In the shelf, click acmeBankDoc.raml.

```
7  documentation:
8    - !include documentation/
9  types:
10   Customer: !include datatypes/Customer.raml
11   Address: !include datatypes/Address.raml
```

Others

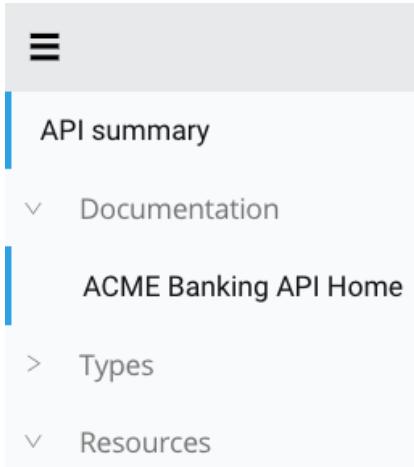
acmeBankDoc.raml

25. Add a new line.

View the API Console

26. In the API Console, click the top left menu icon.

27. Locate the Documentation section and click ACME Banking API Home.



28. Verify that you can view the ACME Banking API Home contents in the API Console.

A screenshot of the 'ACME Banking API Home' page. The page title is 'ACME Banking API'. Below it, a paragraph states: 'ACME Banking API enables developers to build applications that make use of the information from resource methods implemented in the API.' Another paragraph below says: 'This API contains functionality that allows developers to retrieve and manipulate customer, account and transaction information. Check out the [API Portal](#) for more details.' There are also 'X' and '-' buttons in the top right corner of the page area.

Create another documentation fragment

29. In the file browser section, click the + icon next to the documentation folder and select New file.
30. In the Add new file dialog box, click the drop-down arrow next to the Specification field and select User Documentation.
31. Set the file name to acmeBankHeadline.raml.
32. Click Create.
33. In the editor, set the type node value to ACME Bank Headline.

33. In the line that contains the content node, press spacebar after the colon and type | (pipe operator)

```
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 API designer, press enter and press tab.

36. Paste the copied lines.

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

Link the documentation fragment to the main RAML definition file

37. In the file browser section, click acme-banking-api.raml.

38. Add a new line below the documentation node definition.

39. Press tab and add an include statement to add the acmeBankHeadline.raml file:

```
- !include documentation/acmeBankHeadline.raml  
6  documentation:  
7      - !include documentation/acmeBankHeadline.raml  
8      - !include documentation/acmeBankDoc.raml
```

View the API Console

40. In the API Console, click the top left menu icon.

41. In the Documentation section, click ACME Bank Headline.

The screenshot shows a sidebar menu for API documentation. At the top is a header with three horizontal bars. Below it is a list of sections: 'API summary', 'Documentation' (which is expanded, showing a sub-section 'ACME Bank Headline' with a blue vertical bar to its left), 'ACME Banking API Home', 'Types' (preceded by a right-pointing arrow), and 'Resources' (preceded by a downward-pointing arrow). The 'ACME Bank Headline' section is currently active.

42. View the documentation content in the API Console.

The screenshot shows the main content area of the API console. At the top is a header with three horizontal bars on the left and a close button ('X') and a minimize/maximize button on the right. Below the header is the text: 'ACME Bank is a *multinational* banking and financial services organization.'

43. Click the top left menu icon to go back to the list of documentation, types, and resources.

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

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

```
12 /customers:  
13   get:  
14     description: Retrieve a list of customers  
15     displayName: Get all customers  
16     headers:  
17       Accept?:
```

To the right of the code editor is a preview panel. It includes a toolbar with a menu icon, a close button, and a "Try it" button. The main area of the preview panel displays the following information:

/customer_id : get

Retrieve a customer with a specific customer ID

Request

GET /customers/{customer_id}

Add a description to a get method

1. Return to API designer.
2. In the /customers resource, create a new line below the line that contains the get method.
3. Press tab and in the shelf, click description.

The screenshot shows the RAML API designer interface. The code editor has the following RAML code:

```
20 /customers:  
21   get:  
22  
23   headers:
```

Below the code editor is a shelf with several categories: Body, Root, Docs, and protocols. The "description" node is highlighted under the "Docs" category. A tooltip or callout box is visible over the "description" node, indicating it is selected.

4. Set the description node value to:

Retrieve a list of customers

```
20  /customers:  
21    get:  
22      description: Retrieve a list of customers  
23      headers:
```

Add descriptions to other methods

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

- /customers post 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

Add displayName nodes to resource methods

6. In the /customers resource, add a new line below the get method description.
7. In the shelf, click displayName.
8. Type the value of the displayName node as:

Get all customers

```
12  /customers:  
13    get:  
14      description: Retrieve a list of customers  
15      displayName: Get all customers  
16      headers:  
17      Accept?:
```

Note: The displayName node helps in better readability of the API functionality in the REST Connect functionality in Flow Designer.

9. Similarly add some or all the following displayName nodes to the following resource methods:

- /customers post Add new customer
- /customers/{customer_id} get Get a customer by customer ID
- /customers/{customer_id} patch Update a customer by customer ID
- /customers/{customer_id} delete Delete a customer by customer ID

View the API Console

10. In the API Console, click the top left menu icon.
11. Under the Resource section, locate the `/{customer_id}` nested resource and expand it to view the methods.
12. Click the GET method link.



13. View the Description above the Request section of the GET method.

The screenshot shows the detailed view for the `/{customer_id}` GET method. At the top, there's a navigation bar with a menu icon and a close button. Below that, the method name `/{customer_id} : get` is displayed next to a `Try it` button. A descriptive text below the method says "Retrieve a customer with a specific customer ID". The word "Request" is followed by the HTTP verb "GET" and the endpoint `/customers/{customer_id}`.

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

14. In the `/customers` resource, locate the optional Accept header and add a new line below it.
15. Press tab in the new line and add a description with the value:

description: Specify the media type of the response to be returned.

```
24      Accept?:  
25          description: Specify the media type of the response to be returned  
26      responses:  
27          200:
```

16. Add a new line and click example in the shelf.

17. Set the example to application/xml.

headers:

Accept?:

description: Specify the media type of the response to be returned

example: application/xml

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

19. In the description node line, type | and press enter.

20. Return to the course snippets.txt file.

21. Locate the Description for Cache-Control header under Module 7 and copy the lines of text.

*****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:

22. Return to API designer, press tab in the new line and paste the text in the new empty line.

30 **Cache-Control:**

31 **description:** |

32 Activates caching and defines cache behavior through cache response directives.

33 Usually defines public or private (cacheable by proxy or not) and max-age for resource.

34 See <http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html> for more information.

35 **Expires:**

Note: Fix the indentation for the last two lines of the description, by going to the beginning of the line and pressing tab to align it with the first line.

23. Add a new line below the description node.

24. Press backspace or delete to go back one tab space.

25. In the shelf, click example.

```
31      description: |
32          Activates cac
33          Usually defin
34          See http://ww
35
36      Expires:
37          type: datetime
```

Docs	Others
displayName	facets
example	type

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

27. 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.

```
Expires:
type: datetime
description: |
    Sets a date in RFC 1123 format from which the cached resource should no longer be consi
    If both the Expires header and max-age in the Cache-Control header are set, max-age wil
example: Tue, 17 Jul 2017 09:30:41 GMT
format: rfc2616
```

Add documentation to other request and response headers

28. In the /customers resource post method, add a description to the Location headers for the \

```
description: URL of the new customer information
59      201:
60          headers:
61              Location:
62                  description: URL of the new customer information
63          body:
```

29. Press enter to add a new line and add an example node.

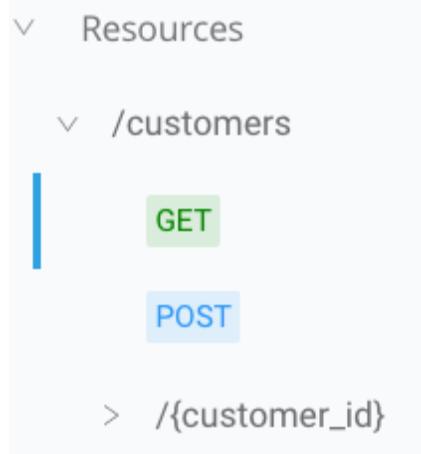
30. Type the following value for the example node:

/customers/8f19cb50-3f57-4d38

View the API Console

31. In the API Console, click the top left menu icon.

32. In the /customers resource click the GET method link.



33. In the Request section of the resource method, verify that you can view the Accept header description and example data.

A screenshot of the API Console's 'Request' section for the GET /customers method. The 'Headers' table is displayed, showing one row for the 'Accept' header. The 'Parameter' column shows 'Accept', the 'Type' column shows 'string', and the 'Description' column contains the text: 'Specify the media type of the response to be returned'. Below the table, it says 'Example value: application/xml'. There is also a 'Hide ^' button in the top right corner of the table area.

34. Click the top left menu icon to go back to the list of resources

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. On the left, there's a sidebar with a menu icon, 'API summary', and a 'Mocking Service' toggle button. The main area displays a successful response: '200 OK' with a response time of '334.72 ms'. The response body is shown in a table-like format:

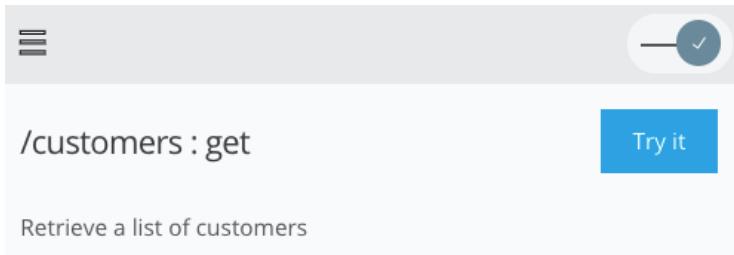
customerID	8f19cb50-3f57-4d38
firstName	John
lastName	Doe
displayName	John Doe
address(Object)	
addressLine1	1234 Lane
addressLine2	Apt.#620
city	San Francisco
state	California

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.

The screenshot shows the API Console interface. The 'Mocking Service' toggle button is highlighted with a red box. The rest of the interface is identical to the previous screenshot, showing the sidebar and the successful response details.

3. Verify that the toggle button now contains a check mark.

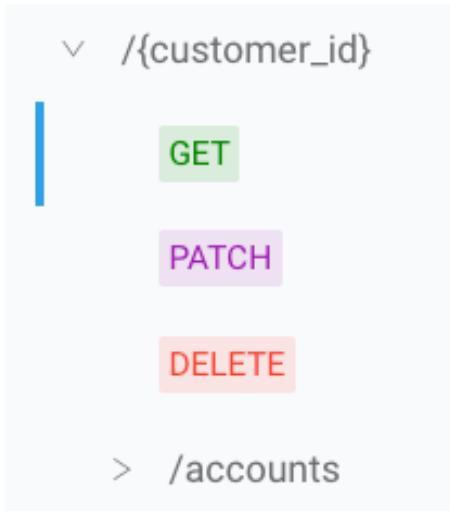


4. Look at acme-banking-api.raml in the RAML editor; you should now see a new baseUrl parameter was added above the title node.

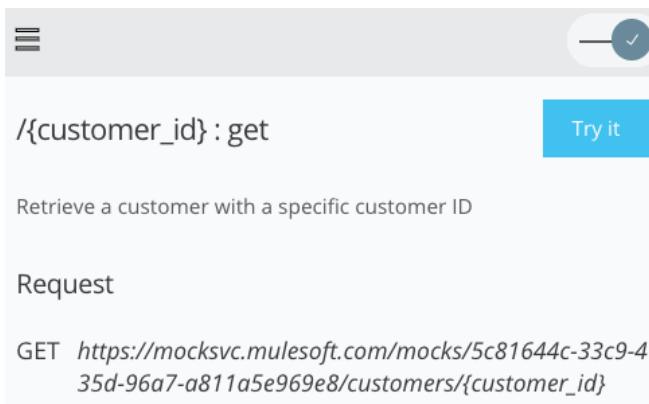
```
1  #%RAML 1.0
2  baseUri: https://mocksvc.mulesoft.com/mocks/5c81644c-33c9-435d-96a7-a811a5e969e8
3  version: 1.0
4  title: ACME Banking API
5  mediaType: application/json
```

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

5. In the API Console, click the top left menu icon.
6. Go to the /{customer_id} resource, expand it (to view the methods) and click the GET method link.

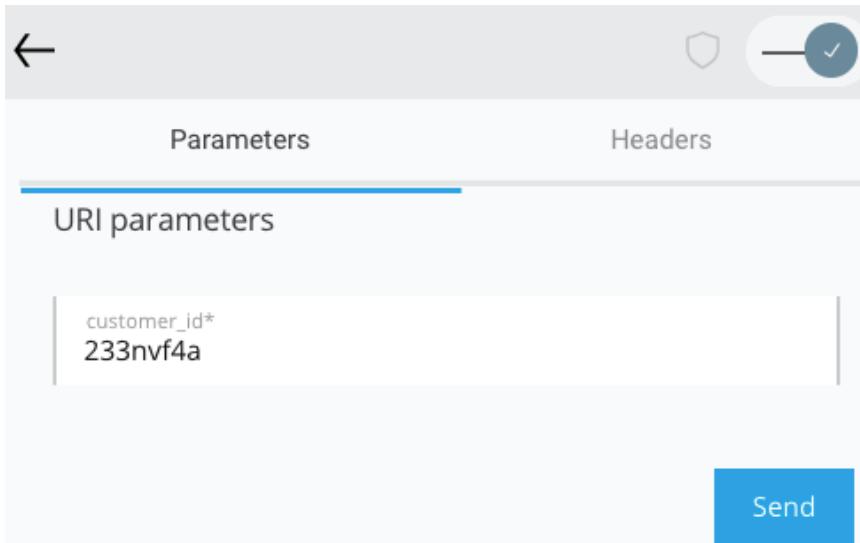


7. Click Try it.



The screenshot shows a user interface element for a REST API call. At the top right is a blue button labeled "Try it" with a checkmark icon. To its left is a URL placeholder: "/{customer_id} : get". Below the URL is a brief description: "Retrieve a customer with a specific customer ID".

8. In the URI parameters field, enter any value as customer_id and click Send.



The screenshot shows a "Send" dialog. At the top are navigation icons: a back arrow, a shield icon, and a blue "Send" button with a checkmark. Below these are tabs for "Parameters" and "Headers", with "Parameters" being active. A section titled "URI parameters" contains a text input field with the value "customer_id* 233nfv4a". At the bottom right is a large blue "Send" button.

9. Verify that you can view the Response status and information.

The screenshot shows a mobile application interface. At the top, there is a grey header bar with a back arrow icon. Below it is a green button-like bar containing the text "200 OK" and "334.72 ms". Underneath these are several icons: a square with a minus sign, a heart, a double arrow, and a vertical ellipsis. The main content area displays a JSON response. It includes fields like "customerID" (value: 8f19cb50-3f57-4d38), "firstName" (value: John), "lastName" (value: Doe), and "displayName" (value: John Doe). A section titled "address(Object)" contains fields for "addressLine1" (value: 1234 Lane), "addressLine2" (value: Apt.#620), "city" (value: San Francisco), and "state" (value: California).

```
customerID: 8f19cb50-3f57-4d38
firstName: John
lastName: Doe
displayName: John Doe
addressLine1: 1234 Lane
addressLine2: Apt.#620
city: San Francisco
state: California
```

Module 8: Making APIs Discoverable

The screenshot shows the Anypoint Exchange interface. At the top, there's a navigation bar with 'Training' and other options. Below it, the main area is titled 'All assets' with a search bar and a 'New' button. It displays three connectors: 'Salesforce Connector' (MuleSoft logo), 'Workday Connector' (WPS logo), and 'ServiceNow Connector' (ServiceNow logo). A banner at the bottom indicates 'ACME Bank'. On the left, a sidebar lists 'Home', 'API reference', and 'API Notebook' (which is selected). The right side features an 'API Notebook' section with code snippets and a preview pane.

Objectives:

- Publish API specifications and fragments to the Anypoint Exchange for discovery.
- 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.
- Gather feedback from API consumers.

Walkthrough 8-1: Publish a RAML API fragment to Anypoint Exchange

In this walkthrough, you promote reusability of an API fragment by publishing it to the private Anypoint Exchange. You will:

- Create an API Fragment project in Design Center.
- Publish an API Fragment to Anypoint Exchange.
- Consume the fragment from Anypoint Exchange.

The screenshot shows the Anypoint Platform Design Center interface. The top navigation bar has the project name 'Banking Data Types' and a 'master' branch indicator. On the left, there's a sidebar with icons for files, projects, and more. The main content area shows a list of files under 'datatypes': Account.raml, AccountOwner.raml, Address.raml, Bank.raml, Customer.raml (which is selected), Money.raml, and Transaction.raml. To the right of the files, there's a code editor window displaying the following RAML code:

```
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
```

Below the code editor, there's a 'Consume API Fragment' button. Further down, there's a search bar labeled 'Search for fragments' and a list of published fragments:

Name	Date Modified	Rating	Created By
<input checked="" type="checkbox"/> Banking datatypes	Jul 18, 2017	★★★★★ (0 votes)	MM Max Mule
<input type="checkbox"/> REST Connect Library	Jul 18, 2017	★★★★★ (0 votes)	CA Connectivity Account

Create an API fragment project in Anypoint Platform Design Center

1. If you are inside the ACME Banking API project, click the Design Center icon in top left corner; this will navigate to the list of projects page.

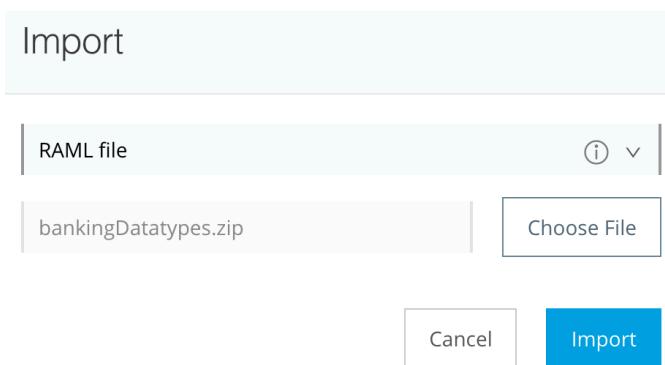
The screenshot shows the Anypoint Platform Design Center interface with the project name 'ACME Banking API' and 'master' branch. The sidebar on the left has icons for files and projects. The main content area shows a list of categories under 'Files': 'datatypes', 'documentation', and 'examples'.

2. In the Design Center homepage, click the Create button next to the search bar.

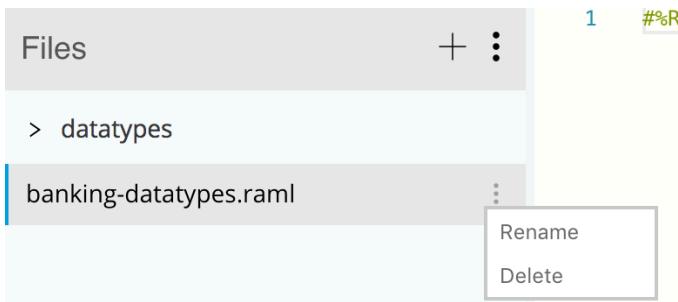
- In the New Project dialog box, type the Project name as Banking Data Types.
- In the Project Type field, click the drop-down menu and select API Fragment.
- In the Fragment Type field, click the drop-down menu and select Data Type.
- Click Create.
- In API designer, click the menu icon in the Files section header.



- Click Import.
- In the Import dialog box, click Choose File.
- Navigate to the APDevApiDesign3.8_studentFiles folder and select bankingDatatypes.zip file.
- Click Import.



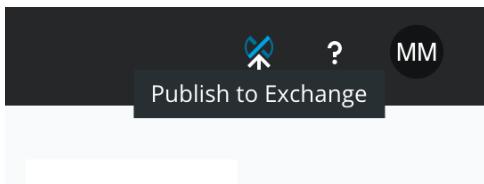
- In the file browser section, click the menu icon next to banking-datatatypes.raml.



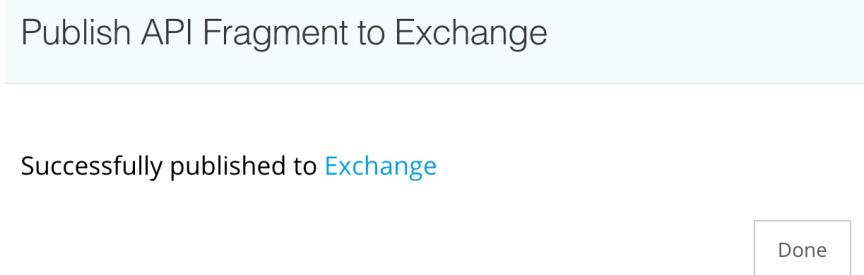
- Click Delete.

Publish an API Fragment to Anypoint Exchange

14. In the top right corner of the Design Center webpage, click the Publish to Exchange icon.



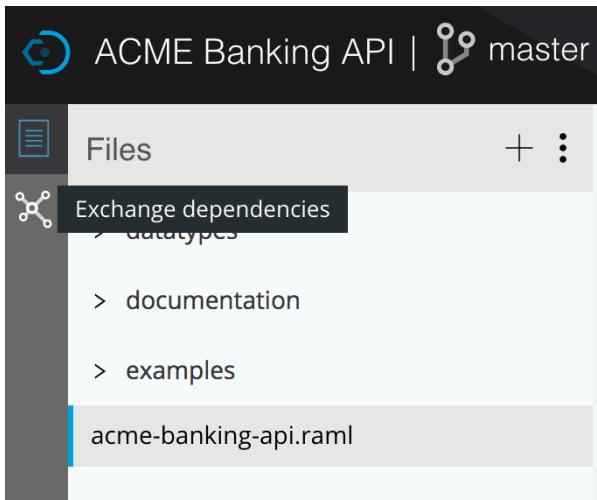
15. In the Publish API Fragment to Exchange dialog box, click Publish.
16. Verify that you see a dialog box with a message that the Banking datatypes project was successfully published to Exchange.



17. Click Done.

Consume an API Fragment from Anypoint Exchange

18. In the API designer page, click the Design Center icon in the top left corner next to the Banking datatypes project name.
19. In the Design Center homepage, click the ACME Banking API project.
20. In the file browser section, click the Exchange dependencies icon below the file icon.



21. Click the + icon in the Dependencies header section.
22. In the Consume API Fragment dialog box, check the box for Banking datatypes API fragment.

The screenshot shows a dialog box titled "Consume API Fragment". At the top is a search bar labeled "Search for fragments" with a magnifying glass icon. Below the search bar is a table with four columns: "Name", "Date Modified", "Rating", and "Created By". There are two rows in the table:

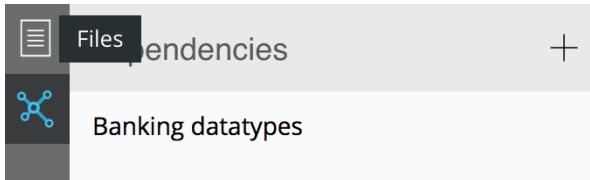
Name	Date Modified	Rating	Created By
<input checked="" type="checkbox"/> Banking datatypes	Jul 18, 2017	★★★★★ (0 votes)	Max Mule
<input type="checkbox"/> REST Connect Library	Jul 18, 2017	★★★★★ (0 votes)	Connectivity Account

23. Click the Add 1 Dependency button.
24. Verify that you can view the path to the asset added as dependency from Exchange and click Ok.

The screenshot shows the same "Consume API Fragment" dialog box. A message at the top states: "Your new dependency is at the 'exchange_modules/b2447622-1656-4ff2-85a7-6a4627f47b7f/banking-datatypes/1.0.0/' folder." Below this message is a link: "You can manage your dependencies from the Dependencies panel .

In the bottom right corner of the dialog box, there is a button labeled "Ok".

25. Click the Files icon to the top-left next to the Dependencies section header.



26. In the file browser section, click the acme-banking-api.raml file.
27. In the RAML editor, go to the lines that contain the types definition.
28. Remove the lines that include Money, AccountOwner, Bank, Address datatype fragments.

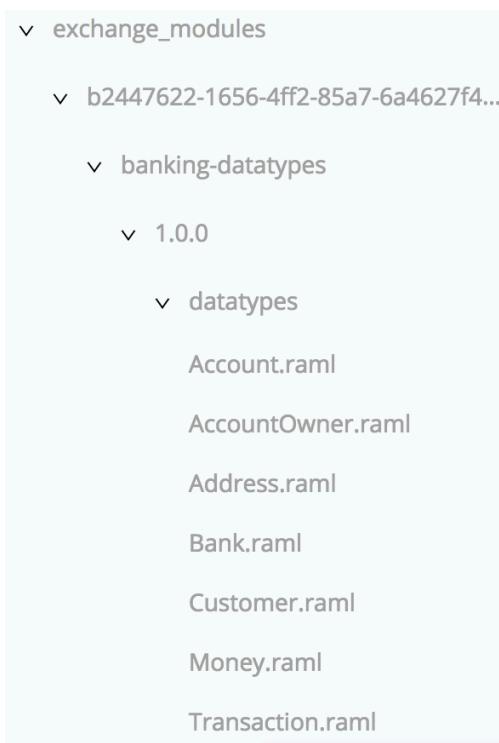
```

11  types:
12  Customer: !include datatypes/Customer.raml
13  Account: !include datatypes/Account.raml
14  Transaction: !include datatypes/Transaction.raml
15  CustomErrorMessage: !include datatypes/CustomErrorMessage.raml
-- 

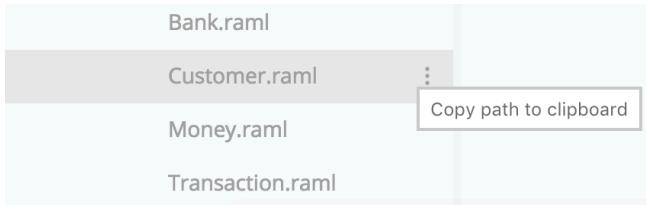
```

Note: The errors are because of the removal of the datatype fragment files. We will be getting rid of those errors in the next steps.

29. In the file browser section, expand the exchange_modules folder to view the datatype fragment files.



30. Hover over the Customer.raml file and click the menu icon that appears next to it.



31. Click Copy path to clipboard.

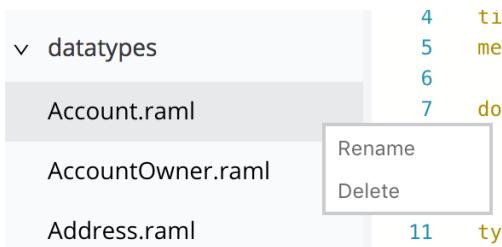
32. In the acme-banking-api.raml file in the editor, replace the include path for the Customer datatype with the path that you copied.

```
11  types:
12  Customer: !include exchange_modules/b2447622-1656-4ff2-85a7-6a4627f47b7f/banking-datatypes/1.0.0/datatypes/Customer.raml
13  Account: !include datatypes/Account.raml
14  Transaction: !include datatypes/Transaction.raml
15  CustomErrorMessage: !include datatypes/CustomErrorMessage.raml
```

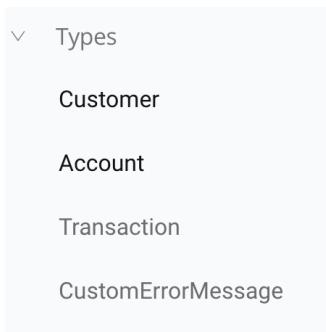
33. Similarly, repeat the steps to change the include path for Account and Transaction datatypes.

```
11  types:
12  Customer: !include exchange_modules/b2447622-1656-4ff2-85a7-6a4627f47b7f/banking-datatypes/1.0.0/datatypes/Customer.raml
13  Account: !include exchange_modules/b2447622-1656-4ff2-85a7-6a4627f47b7f/banking-datatypes/1.0.0/datatypes/Account.raml
14  Transaction: !include exchange_modules/b2447622-1656-4ff2-85a7-6a4627f47b7f/banking-datatypes/1.0.0/datatypes/Transaction.raml
15  CustomErrorMessage: !include datatypes/CustomErrorMessage.raml
```

34. In the file browser section, expand the datatypes folder.
35. Click the menu icon next to the Account.raml file and click Delete.



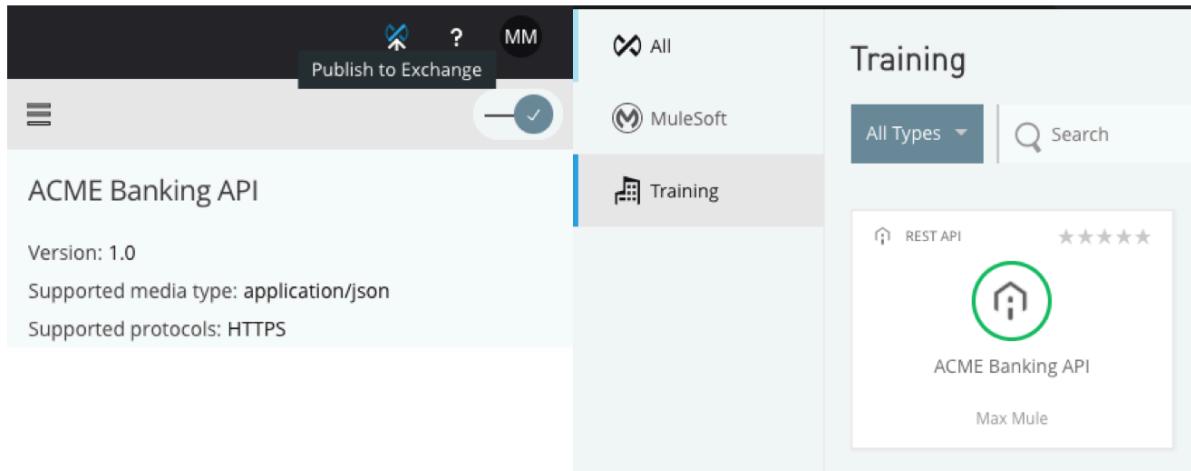
36. Similarly, delete the rest of the datatype files except the CustomErrorMessage.raml file.
37. In the API Console, click the top left menu icon and expand the Types section.
38. Verify that you see the Customer, Account and Transaction datatypes from exchange dependencies listed.



Walkthrough 8-2: Publish a RAML API Specification to Anypoint Exchange

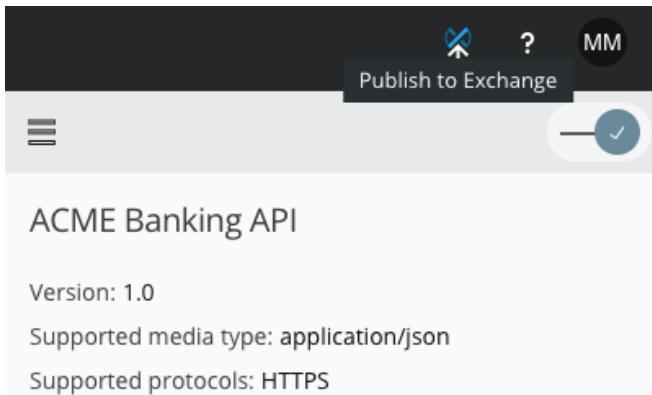
In this walkthrough, you enhance the discoverability of an API by adding it to the private Anypoint Exchange. You will:

- Publish a RAML API Specification to private Exchange.
- Add documentation to an Exchange asset to improve usability.



Publish a RAML API Specification to private Exchange

1. Return to API designer.
2. In the top right corner, next to the username initials, locate and click the Publish to Exchange button.



3. In the Publish API Specification to Exchange, check the option Also import into API Manager.

Publish API Specification to Exchange

Name (required)	ACME Banking API
Main file (required)	acme-banking-api.raml
	API version ⓘ (required) 1.0 <small>Valid RAML</small>
Tags	Tag... <input type="button" value="Add"/>
Show advanced >	
<input checked="" type="checkbox"/> Also import into API Manager	<input type="button" value="Cancel"/> <input type="button" value="Publish"/>

4. Click Publish.

Note: If you get errors while publishing to Exchange, click the Show advanced link to make sure the group_id is unique (public group_id is com.mulesoft), rename it to com.mulesoft.training or anything that is not the same as the public group_id.

5. Verify that you see the message in the dialog box that indicates the API Specification has been successfully published to Exchange and API Manager.

Publish API Specification to Exchange

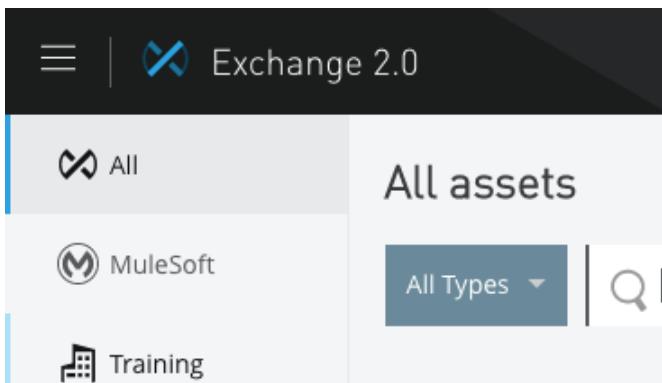
Successfully published to [API Manager](#) and [Exchange](#)

6. Click Done.

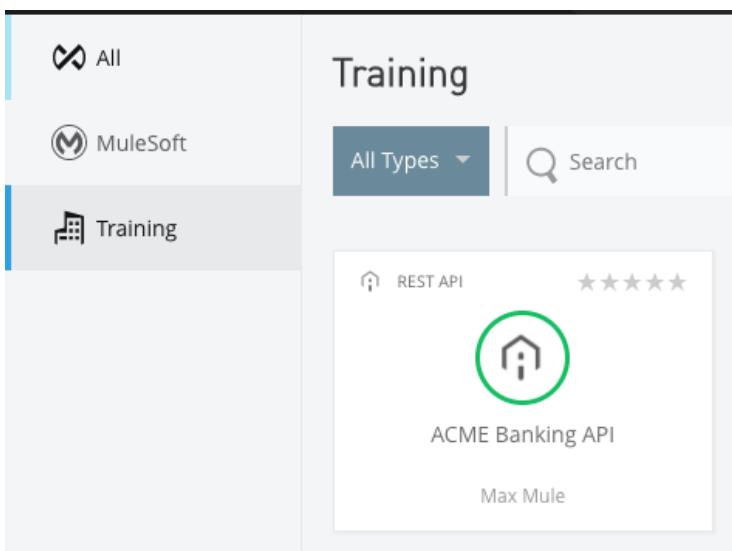
View the asset in Exchange

7. In API designer, click the top left Design Center icon.
8. Click the top left menu icon to open the Navigation menu.
9. Click Exchange 2.0.

10. In Exchange 2.0 webpage, click your business group name in the left navigation panel.



11. In the list of assets in your business group, click ACME banking API.



Add documentation to an Exchange asset

12. In the ACME Banking API asset page, click Edit in the center of the page to add information about the asset.

The screenshot shows the ACME Banking API asset page. On the left, there's a sidebar with a back arrow labeled "Assets list" and a home icon labeled "ACME Banking API". The main area has a house icon in a green circle, followed by the text "ACME Banking API" and a pencil icon. Below that is a star rating of 0 votes and a "Rate and review" link. A large, semi-transparent gray silhouette of a person's head and shoulders is centered on the page. At the bottom, it says "ACME Banking API does not contain content. Click edit to add information about your asset." To the right of this text is a blue "Edit" button with a pencil icon.

13. In the top right corner of the ACME Banking API content editor, change the tab from Markdown to Visual.

The screenshot shows the ACME Banking API content editor. At the top, there's a toolbar with various icons: bold (B), italic (I), code (code block), list (list), list (list), heading (H), horizontal line (—), quote (‘‘), and image (image). Below the toolbar, there are two tabs: "Markdown" and "Visual". The "Visual" tab is highlighted with a blue background. At the bottom of the editor, there are two buttons: "Discard changes" and "Save as draft".

14. Go to APDevApiDesign3.8_snippets.txt.
15. In Module 8 section, copy the content under ACME Banking API Exchange content.
16. Return to Exchange 2.0 webpage and paste the content.

ACME Banking API

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

17. Select the lines that list out the API functionality and click the numbered bullets icon on top to add numbering.

ACME Banking API contains functionality that allows developers to retrieve and manipulate customer, account and transaction information. The API currently supports the following functionality:

1. Retrieve a list of customers
2. Retrieve a customer with a specific customer ID
3. Retrieve a list of customer accounts
4. Retrieve an account with a specific account ID
5. Retrieve a list of bank account transactions
6. Retrieve a transaction with a specific transaction ID
7. Add a new customer
8. Create a new account
9. Add a new bank account transaction
10. Delete a specific customer
11. Delete a specific account
12. Update a specific customer profile information
13. Update a specific bank account information

18. Select the ACME Banking API text at the beginning and click the B icon to bold the text.
19. Similarly, make the API Reference text below the numbered list bold.

20. In the bottom left corner of the editor, click the Save as draft button.

Check out the [API Reference](#) page to learn more about the resources and methods.

[Discard changes](#) [Save as draft](#)

21. In the Exchange asset details panel on the left, click the Publish button to add the content information to the asset.

This screenshot shows the Exchange asset details panel. At the top, there is a message: "This is a draft that has not been published yet." Below it are buttons for "Exit Draft" and "Publish". The "Publish" button is highlighted with a blue background. To the right, there is an "Edit" button and a three-dot menu icon. The main content area displays the asset details for "ACME Banking API". It includes a thumbnail, the asset name, a star rating of 5 stars (0 votes), and a brief description: "ACME Banking API contains functionality that allows developers to retrieve and manipulate customer, account and transaction information. The API currently supports the following functionality:". Below the description is a numbered list of 4 items. On the right side, there is an "Overview" section with details: Type (REST API), Created By (Max Mule), and Published On (Jul 19, 2017).

22. Verify that you are redirected to the Asset homepage, with the information about the asset listed in the center panel.

This screenshot shows the Asset homepage. It features the same "ACME Banking API" asset details as the previous screenshot. The asset thumbnail, name, star rating, and description are all present. Below the description is a numbered list of 9 items. To the right, there is an "Overview" section with details: Type (REST API), Created By (Max Mule), and Published On (Jul 19, 2017). Further down, there is a "Versions" section.

Walkthrough 8-3: Create and customize an API Portal

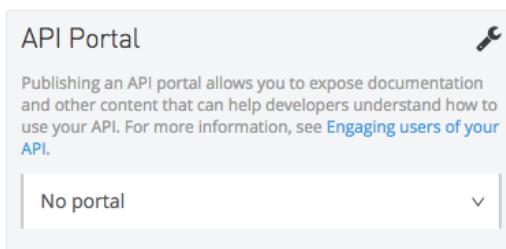
In this walkthrough, you will create an API Portal documenting information about the ACME Banking API. You will:

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

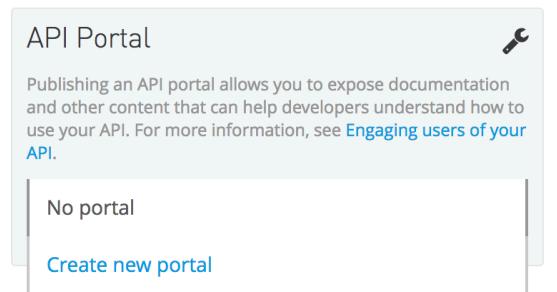
The screenshot shows two main windows. On the left is the 'API Manager' interface, displaying a list of APIs including 'ACME Banking API'. On the right is the 'ACME Banking API 1.0' portal, which includes sections for 'Documentation' (with links to 'ACME Bank Headline' and 'ACME Banking API Home'), 'Resources' (with links to '/customers' and '/customers/{customer_id}'), and a 'Home' section detailing API functionality. The portal is marked as 'PRIVATE' and has a 'apiDesignTraining' tag.

Create an API Portal

1. Return to Anypoint Platform webpage and click the top left menu to open the navigation panel.
2. Click API Manager.
3. In the list of APIs in API Manager, click the version number 1.0 under ACME Banking API.
4. In the API version details page, locate the API Portal section and click the drop-down next to the No portal field.



5. Select Create new portal from the drop-down menu; the API Portal Designer should open.



6. In the API Portal Designer, locate the two pages that were automatically created: Home and API reference.
7. In the left-side navigation, click the Home page link.
8. 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.

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

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

A screenshot of the API reference page in the API Portal Designer. On the left, there is a sidebar with an "Add" button, a trash bin icon, and an eye icon. Below that are links for "Home" and "API reference", where "API reference" is selected and highlighted with a blue border. The main content area is titled "API reference". It contains sections for "Documentation" (with "ACME Bank Headline" and "ACME Banking API Home"), "Types" (with a plus sign icon), and "Resources". Under "Resources", there is a section for "/customers" with endpoints: "/customers/{customer_id}" (with "DELETE", "PATCH", and "GET" buttons), "/customers/{customer_id}/accounts" (with "GET" and "POST" buttons), and a section for "/accounts" (with a "POST" button). There are also buttons for "API is behind a firewall (?)" and "Collapse All".

Use Markdown to add content to a page

11. In the left-side navigation, click the Home page link.

12. In the Home page editor, click the letter A to begin adding content.

The screenshot shows a 'Home' page editor interface. At the top right are two buttons: a white square with an eye icon and a blue button with a checkmark and the text 'Saved'. Below these are three icons: a magnifying glass over a document, a camera, and a document. A large text input field is centered, containing the letter 'A'.

13. Return to the course snippets.txt file.

14. 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

15. Return to the API Portal Designer and click the box with the text Write your documentation here in **markdown** format.

The screenshot shows the 'API Portal Designer' interface. At the top right is a blue 'Save' button with a circular arrow icon. Below it is a text input field containing the placeholder text 'Write your documentation here in **markdown** format.'

16. 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

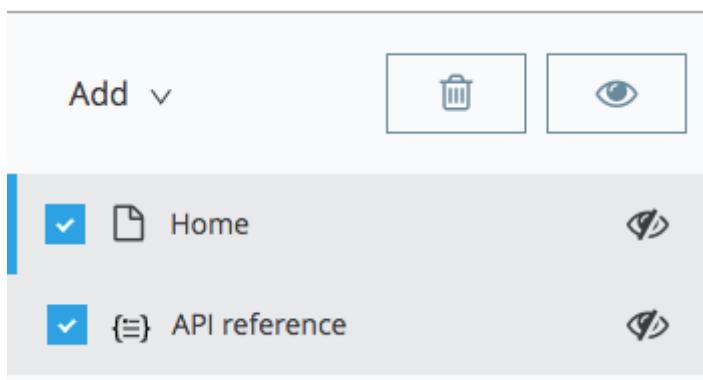
- Add a new customer
- Create a new account
- Add a new bank account transaction

- Delete a specific customer

17. Click the Save button.

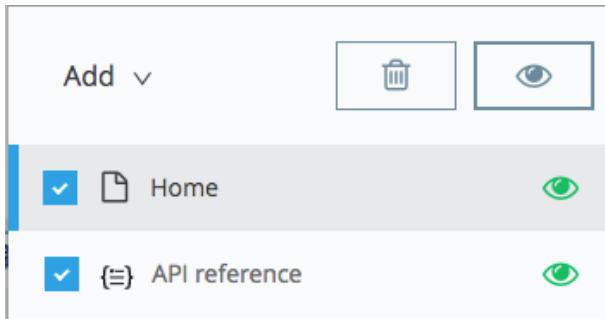
Publish the API Portal

18. In the left-side navigation, select the checkboxes for all the components.



19. Click the eye button to set the selected pages to be visible in the API Portal.

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.



20. Click the Live portal button in the upper-right corner of the API Portal Designer.

21. Examine the generated API Portal.

22. 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

23. In the right corner of the API Portal Designer, click Themes.

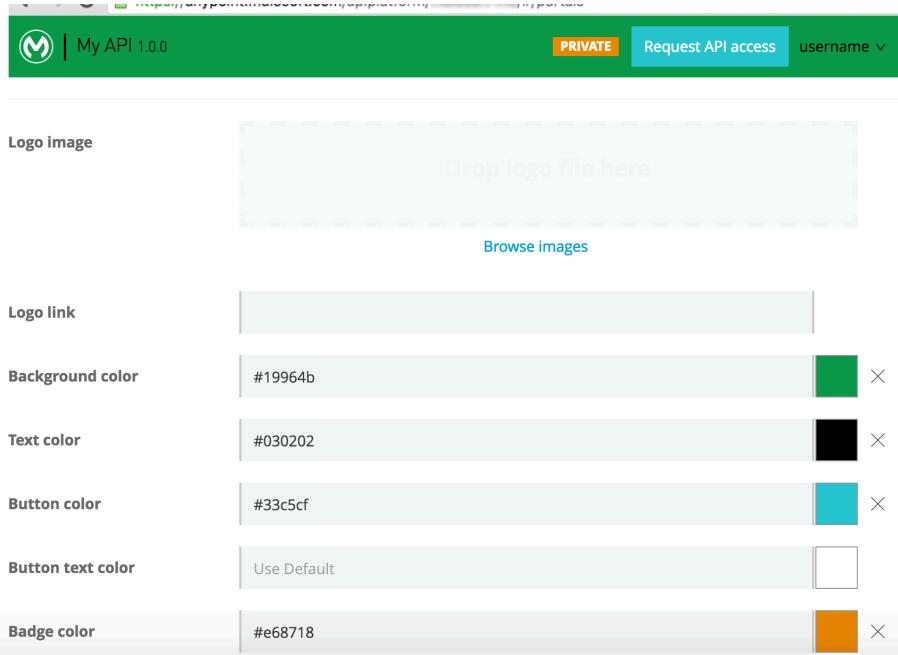


24. In the API portal theme settings dialog box, go to the Background color option and click the text area that mentions Use Default.

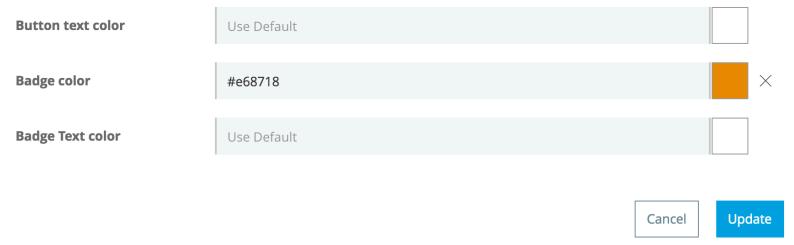
25. Select a color from the drop-down graph or type a color's hex value.

A screenshot of the 'My API 1.0.0' theme settings page. At the top, it shows the URL 'https://anypoint.mulesoft.com/apiplatform/.../#/portals'. Below that is a header with a logo, the API name, a 'PRIVATE' button, a 'Request API access' button, and a user dropdown. The main content area has sections for 'Logo image' (with a placeholder 'Drop logo file here' and a 'Browse images' button), 'Logo link' (a text input field), and several color selection fields. The 'Background color' field has a 'Use Default' button and a color picker with a blue square preview. The 'Text color' field has a color picker with a white square preview. The 'Button color' field has a color picker with a dark blue square preview. The 'Button text color' field has a 'Use Default' button and a color picker with a white square preview.

26. Change the text color, button color and button text color using the same approach of clicking over the Use Default text box and picking a desired color.



27. Scroll down and click the Update button.



28. Click the Live Portal button again; the live portal should open and you should see your changes.

The screenshot shows the ACME Banking API 1.0 Home page. The top navigation bar includes a logo, the text "ACME Banking API 1.0", and a "PRIVATE" badge. Below the navigation is a breadcrumb trail: "Developer portal / ACME Banking API (1.0) - Home". The left sidebar has two items: "Home" (which is selected and highlighted in grey) and "API reference". The main content area is titled "Home" and contains the following text:
ACME Banking API contains functionality that allows developers to retrieve and manipulate *customer*, *account* and *transaction*. 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

- 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.

29. Close the Live Portal and return to the API Portal Designer.

Walkthrough 8-4: 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 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 its 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" and has 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.
3. In the API Notebook editor, change the first parameter in the createClient function from 'client' to 'acmeBankclient'.

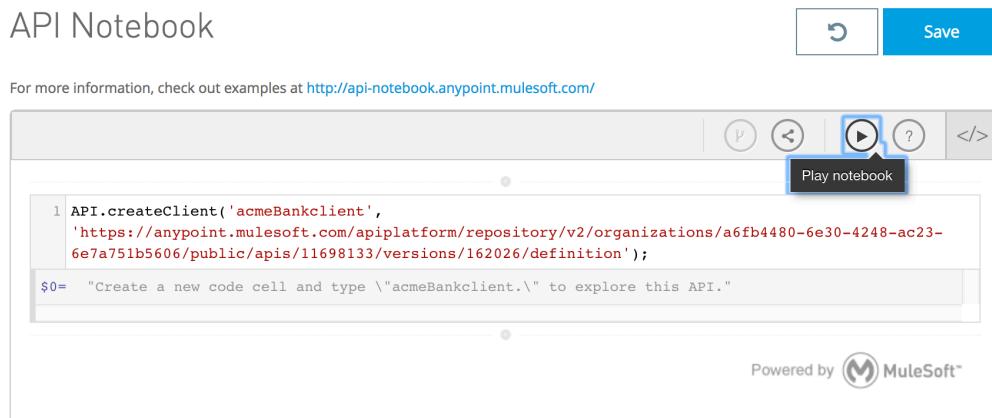
The screenshot shows the API Notebook editor. At the top, there's a title bar "API Notebook" and a "Save" button. Below that is a message: "For more information, check out examples at <http://api-notebook.anypoint.mulesoft.com/>". The main area has a toolbar with icons for back, forward, search, and help. A code editor window displays 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');
createClient(alias, url, options?, cb?)
string
```

At the bottom right, it says "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.”



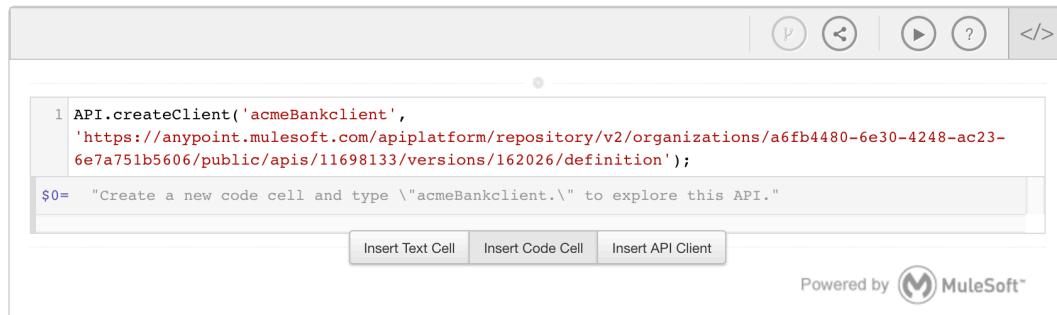
The screenshot shows the MuleSoft API Notebook interface. At the top, there's a navigation bar with icons for back, forward, search, and help, followed by a 'Save' button. Below the bar, a message reads: "For more information, check out examples at <http://api-notebook.anypoint.mulesoft.com/>". The main area contains a code cell with the following content:

```
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, a message says: "\$0= "Create a new code cell and type \"acmeBankclient.\" to explore this API."'. At the bottom right of the interface, it says "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.

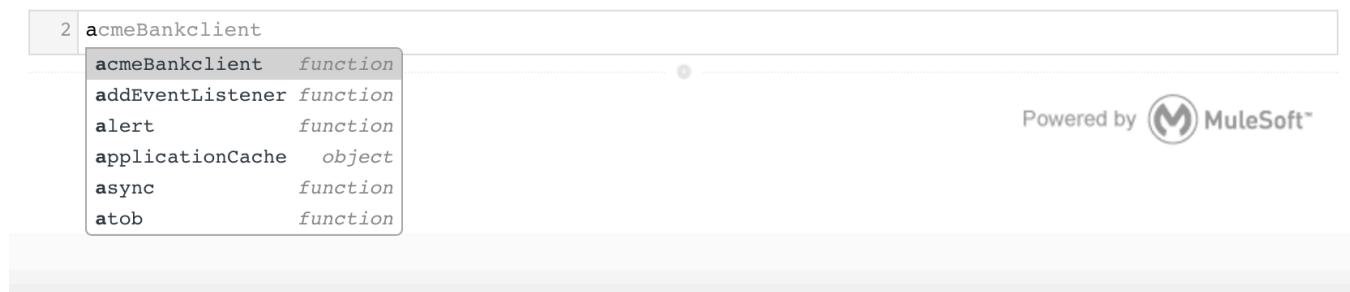


The screenshot shows the MuleSoft API Notebook interface. A code cell contains the following code:

```
1 API.createClient('acmeBankclient',
  'https://anypoint.mulesoft.com/apiplatform/repository/v2/organizations/a6fb4480-6e30-4248-ac23-
  6e7a751b5606/public/apis/11698133/versions/162026/definition');
```

Below the code cell, a message says: "\$0= "Create a new code cell and type \"acmeBankclient.\" to explore this API."'. At the bottom, there are three buttons: "Insert Text Cell", "Insert Code Cell", and "Insert API Client". At the very bottom right, it says "Powered by MuleSoft".

- Click Insert Code Cell.
- In the new code cell, type the letter a and press the right arrow key.

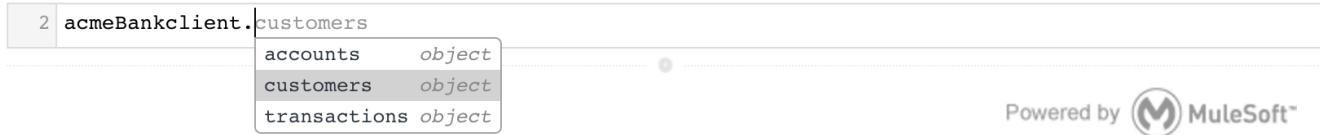


The screenshot shows the MuleSoft API Notebook interface. A code cell has the number "2" next to it. The cell content is "acmeBankclient". A dropdown menu is open, listing several suggestions:

- acmeBankclient function
- addEventListener function
- alert function
- applicationCache object
- async function
- atob function

At the bottom right of the interface, it says "Powered by MuleSoft".

9. Type a .(period) after acmeBankclient and select customers in the drop-down menu.



Powered by  MuleSoft™

10. Type .(period) 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
```

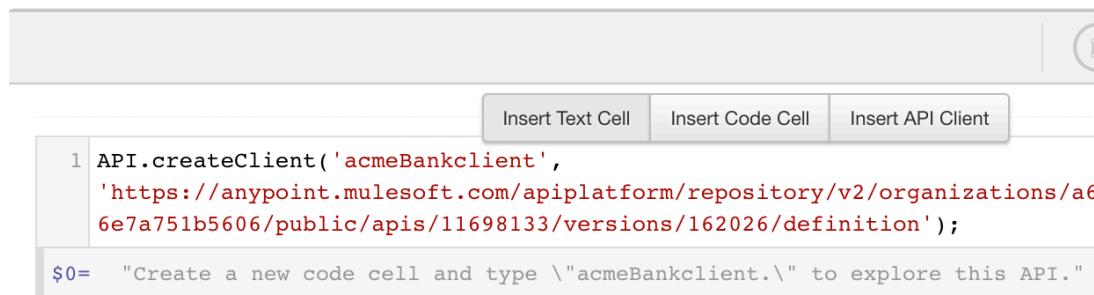
Note: When you expand the body, you can see the customer example returned.

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.



1 API.createClient('acmeBankclient',
'https://anypoint.mulesoft.com/apiplatform/repository/v2/organizations/a6
6e7a751b5606/public/apis/11698133/versions/162026/definition');

\$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.

Create ACME Banking API Client to access it's methods and resources

```
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."
```

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.

Retrieve 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: "johndoe@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 rewards awarded based on the account balance.
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.
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.
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. At the top, there is a header bar with the title 'Create ACME Banking API Client to access it's methods and resources'. Below the header, there are two code cells. The first code cell 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');
```

Below the code, a note says '\$0= "Create a new code cell and type \"acmeBankclient.\" to explore this API."'. The second code cell is titled 'Retrieve a list of customers' and contains the following code:

```
2 acmeBankclient.customers.get();
```

Below the code, the result is shown as a JSON object:

```
$1= ▼Object {"body": Object, "status": 200, "headers": Object}
▶body: Object
▶headers: Object
status: 200
```

58. Return to the API Portal and close it.

Module 9: Reusing Patterns

```
1  #%%RAML 1.0 ResourceType
2  post?:
3      description: Add a new <<resourcePathName | !singularize>>
4      displayName: Add new <<resourcePathName | !singularize>>
5      body:
6          type: <<reso 30      get:
7      responses: 31          description: Retrieve a list of customers
8                      is:
9                          - cacheable
10                         - hasAcceptHeader:
11                             customErrorDataType: CustomErrorMessage
12                         responses:
13                             200:
```

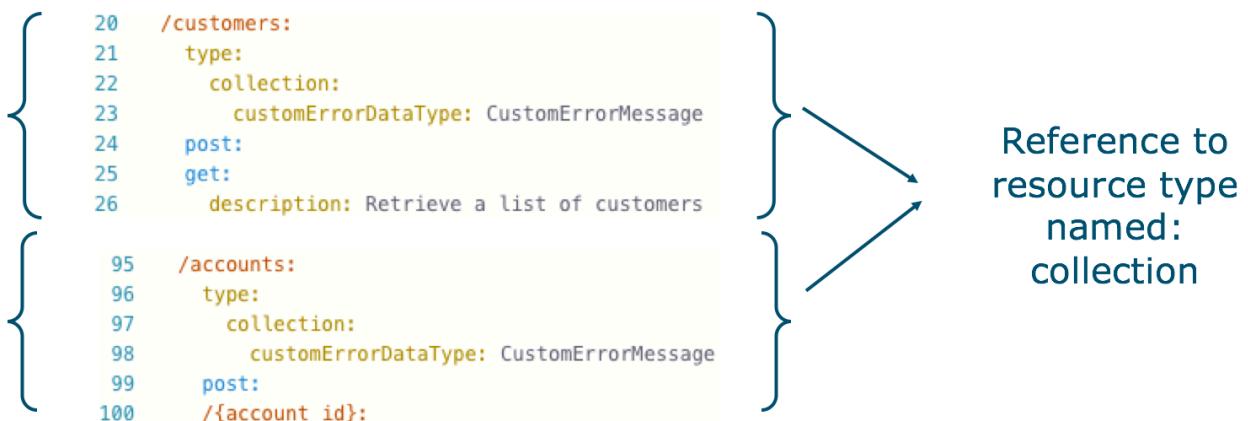
Objectives:

- Create and reference resource type patterns for reusability.
- Use traits to modularize methods.

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.

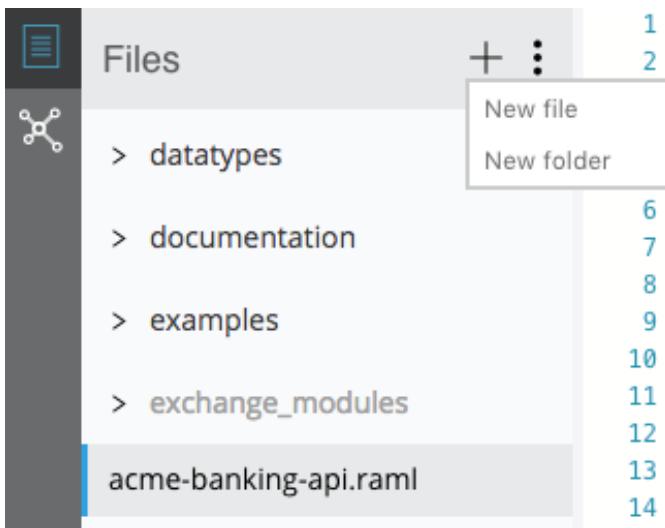


Define a collection resource type fragment

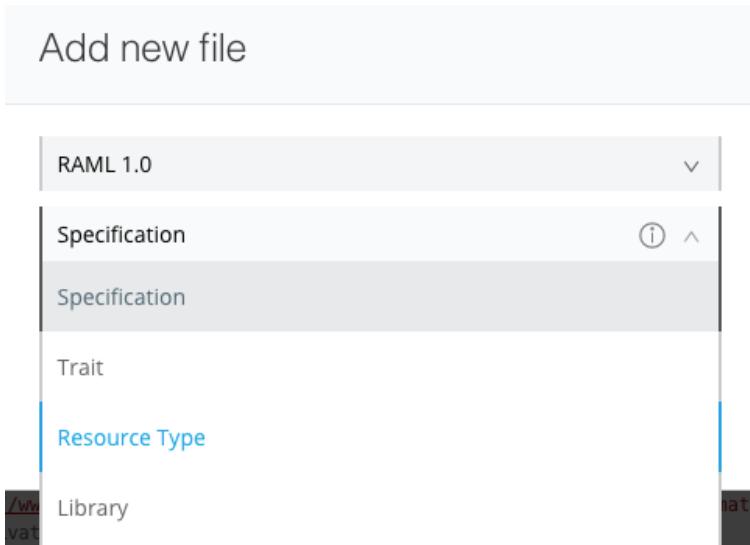
1. In the Anypoint Platform webpage, click the top-left menu icon.
2. Select Design Center.
3. In the Design Center projects page, click ACME Banking API.

Name	Project Type
Banking datatypes	API Fragment
ACME Banking API	API Specification

4. In the file browser section, click the + icon next to the Files header and select New folder.



5. In the Add new folder dialog box, set the name of the folder to resourceTypes.
6. Click Create.
7. In the file browser, hover over the resourceTypes folder and click the + icon.
8. Select New file.
9. In the Add new file dialog box, click the drop-down box next to the Specification field and select Resource Type.



10. Rename the file as collection.raml.
11. Click Create.
12. In the file browser section, click acme-banking-api.raml to open the RAML definition file.
13. In the /customers resource, cut the post method definition.
14. In the file browser, expand the resourceTypes folder and click collection.raml.

15. Paste the lines after the first line in collection.raml.

Note: 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

16. In the collection.raml resource type file, go to the line that contains the description for the post method.

17. Replace the word customer with <<resourcePathName | !singularize>>

```
1  %%RAML 1.0 ResourceType
2  post:
3    description: Add a new <<resourcePathName | !singularize>>
4    body:
```

18. Similarly, in the displayName node, replace the word customer with <<resourcePathName | !singularize>>.

19. Similarly, in the description node for the Location header, replace the word customer with <<resourcePathName | !singularize>>.

20. Similarly, in the example for the location header, replace customers with <<resourcePathName>>.

21. Locate the line of code that contains the post request body type Customer.

22. Delete the Customer datatype and set it to the following:

```
<<resourcePathName | !singularize | !uppercase>>
```

23. Go to the line that contains the HTTP 503 response body type CustomErrorMessage and replace the type node value as <<customErrorDataType>>.

```
11      body:
12      503:
13      body:
14      type: <<customErrorDataType>>|
```

24. Make the post node an optional method.

```
1  #%RAML 1.0 ResourceType
2  post?:
3    description: Add a new <>resourcePathName | !singularize>>
4    displayName: Add new <>resourcePathName | !singularize>>
5    body:
6      type: <>resourcePathName | !singularize | !uppercase>>
7    responses:
```

Reference the resource type in the RAML API definition

25. In the file browser section, select acme-banking-api.raml.

26. Add a new line before the /customers resource and add the resourceTypes node:

resourceTypes:

```
17  resourceTypes:
18  |
19  /customers:
20  get:
21    description: Retrieve a list of customers
```

27. In the new line, include the collection resourcetype by typing:

```
collection: !include resourceTypes/collection.raml
```

28. Press enter.

```
17  resourceTypes:
18  collection: !include resourceTypes/collection.raml
19  |
20  /customers:
21  get:
```

29. In the /customers resource, add a new line before the get method.

Note: Indent the new line at the same tab level as the get method.

30. Reference the type collection and add the mapping values by typing:

```
type:
  collection:
    customErrorDataType: CustomErrorMessage
```

39. Add the post method in the line before the get method definition.

```
20   /customers:  
21     type:  
22       collection:  
23         customErrorDataType: CustomErrorMessage  
24     post:  
25     get:  
26       description: Retrieve a list of customers
```

31. Similarly, go to the /accounts main resource and delete the post method definition.

32. In the /accounts main resource, in the empty line, reference the collection resource type:

```
type:  
collection:  
  customErrorDataType: CustomErrorMessage
```

33. Add the post method node below the reference to the collection resource type.

```
95   /accounts:  
96     type:  
97       collection:  
98         customErrorDataType: CustomErrorMessage  
99     post:  
100    /{account_id}:  
101
```

34. Similarly, in the /transactions main resource, delete the post method definition and add a reference to the collection resource type.

```
140   /transactions:  
141     type:  
142       collection:  
143         customErrorDataType: CustomErrorMessage  
144     post:  
145
```

View the documentation

35. In the API Console, click the top left menu icon.

36. Locate the /customers resource.

37. Click the POST method for the /customers resource and verify that you see the method details.

The screenshot shows a REST API documentation interface. At the top, there is a header with a menu icon and a checkmark icon. Below the header, the URL '/customers : post' is displayed next to a 'Try it' button. A descriptive text 'Add a new customer' follows. The word 'Request' is bolded. Below 'Request', a 'POST' verb is listed with a specific URL: `https://mocksvc.mulesoft.com/mocks/ed87b212-aa47-410b-9e19-56f864b5bf09/mocks/5c81644c-33c9-435d-96a7-a811a5e969e8/customers`. A 'Body' section is expanded, showing the schema for a 'Customer' type. The schema is defined as:

```
{  
  "suffix": "string",  
  "address": {  
    "addressLine1": "string",  
    "addressLine2": "string",  
    "city": "string",  
    "zipCode": "string",  
    "state": "string",  
    "country": "string"  
},
```

Walkthrough 9-2: Define and use a resource type for resources that perform operations on a member

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 mappings to pass parameter values to the resource type.
- Reference the resource type in the RAML API definition.

```
59      /{customer_id}:
60          type:
61              member:
62                  exampleValue: !include examples/CustomerExample.raml
63                  customErrorDataType: CustomErrorMessage
64          get:
65          patch:
66          delete: | 82      /{account_id}:
83              type:
84                  member:
85                      exampleValue: !include examples/AccountExample.raml
86                      customErrorDataType: CustomErrorMessage
87              get:
88              put:
89              delete:
```

Define a member resource type fragment

1. Return to API designer.
2. In the file browser, hover over the resourceTypes folder and click the + icon.
3. Select New file
4. In the Add new file dialog box, select the file type as Resource Type and rename it as member.raml.
5. Click Create.
6. In the file browser, select acme-banking-api.raml.

7. Locate the /{customer_id} resource and cut all the lines for the get, patch and delete methods.

```
58  /{customer_id}:
59    get:
60      description: Retrieve a customer with a specific customer ID
61      responses:
62        200:
63          body:
64            type: Customer
65            example: !include examples/CustomerExample.raml
66        404:
67          body:
68            type: CustomErrorMessage
69    patch:
70      description: Update a specific customer info
71      body:
72      responses:
73        204:
74        501:
75          body:
76            type: CustomErrorMessage
77    delete:
78      description: Delete a specific customer
79      responses:
80        200:
81          body:
82        404:
83          body:
84            type: CustomErrorMessage
85  /accounts:
```

8. In the file browser, click the resourceTypes/member.raml file.
9. Paste the copied lines after the first line in member.raml.
10. Select the pasted lines and press Shift+tab two times to indent the lines properly.

```
1  #%%RAML 1.0 ResourceType
2  get:
3    description: Retrieve a customer with a specific customer ID
4    responses:
5      200:
6        body:
7          type: Customer
8          example: !include examples/CustomerExample.raml
9        404:
10       body:
11         type: CustomErrorMessage
12    patch:
13      description: Update a specific customer info
14      body:
15      responses:
16        204:
17        501:
18          body:
19            type: CustomErrorMessage
20    delete:
21      description: Delete a specific customer
22      responses:
23        200:
24          body:
25        404:
26          body:
27            type: CustomErrorMessage
```

11. Go back to acme-banking-api.raml and locate the /{account_id} method.

12. Cut the lines for the put method definition.

```
85   put:
86     body:
87     responses:
88       200:
89         body:
90           type: Account
91       201:
92         headers:
93           Location:
94         body:
95       501:
96         body:
97           type: CustomErrorMessage
```

13. Go to member.raml and paste the put method below the delete method definition.

Note: Make sure to fix the indentation.

```
26   body:
27     type: CustomErrorMessage
28 put:
29   body:
30   responses:
31     200:
32       body:
33         type: Account
34     201:
35       headers:
36         Location:
37       body:
38     501:
39       body:
40         type: CustomErrorMessage
```

14. In member.raml file, add the optional attribute to all the methods.

```
1  #%RAML 1.0 ResourceType
2  get?:
3    description: Retrieve a customer !
4    responses:
5      200:
6        body:
7          type: Customer
8          example: !include examples/
9      404:
10        body:
11          type: CustomErrorMessage
12 patch?:
```



MuleSoft

Use mappings to pass parameter values to resource types

15. In the get method, go to the line that contains the description, replace the value as:

Retrieve a/an <>resourcePathName | !singularize<> with a specific
<>resourcePathName | !singularize >>

```
1  #%RAML 1.0 ResourceType
2  get?:
3      description: Retrieve a <>resourcePathName | !singularize<> with a specific <>resourcePathName | !singularize<>
4      responses:
5          200:
6              body:
```

16. Similarly, replace the displayName node value with:

Get a/an <>resourcePathName | !singularize<> by <>resourcePathName | !singularize >> ID

17. Go to the line that contains the HTTP 200 response body type Customer.

18. Delete the type value Customer, and replace it with <>resourcePathName | !singularize | !uppercasecase >>.

19. In the next line, replace the value of the example node with <>exampleValue >>.

```
responses:
  200:
    body:
      type: <>resourcePathName | !singularize | !uppercasecase >>
      example: <>exampleValue >>
```

20. Go to the HTTP 404 response body and replace the CustomErrorMessage datatype with <>customErrorDataType >>.

21. Similarly, replace the patch and delete method description, displayName, and error response body types with similar mappings.

```
12  patch?:
13    description: Update a specific <>resourcePathName | !singularize>> info
14    body:
15    responses:
16      204:
17      501:
18        body:
19          type: <<customErrorDataType>>
20  delete?:
21    description: Delete a specific cus<<resourcePathName | !singularize>>
22    responses:
23      200:
24        body:
25      404:
26        body:
27          type: <<customErrorDataType>>
```

22. In the put method definition replace the 200-response body type with an in-built mapping and the 501-response body type with a custom mapping.

```
28  put?:
29    body:
30    responses:
31      200:
32        body:
33          type: <<resourcePathName | !singularize | !uppercase>>
34      201:
35        headers:
36          Location:
37        body:
38      501:
39        body:
40          type: <<customErrorDataType>>
```

Reference the member resource type fragment in the RAML API definition

23. In the file browser, select acme-banking-api.raml.
24. Go to the resource type definition node above the /customers resource and add a new line below the collection resource type.
25. In the new line type:

member: !include resourceTypes/member.raml

```
17  resourceTypes:
18    collection: !include resourceTypes/collection.raml
19    member: !include resourceTypes/member.raml
20
21  /customers:
```

26. Go the /{customer_id} resource and add lines to reference the member resource type with the get, patch and delete methods.

type:

member:

exampleValue: !include examples/CustomerExample.raml

customErrorDataType: CustomErrorMessage

get:

patch:

delete:

59 /{customer_id}:

60 type:

61 member:

62 exampleValue: !include examples/CustomerExample.raml

63 customErrorDataType: CustomErrorMessage

64 get:

65 patch:

66 delete:|

27. Similarly, in the /{account_id} nested resource, delete the get and delete methods.

28. Reference the member resource type with the custom mappings for example, custom error message and include the get, put and delete methods.

type:

member:

exampleValue: !include examples/AccountExample.raml

customErrorMessageType: CustomErrorMessage

get:

put:

delete:

82 /{account_id}:

83 type:

84 member:

85 exampleValue: !include examples/AccountExample.raml

86 customErrorDataType: CustomErrorMessage

87 get:

88 put:

89 delete:|

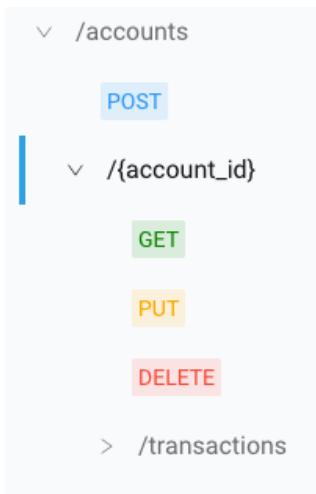
29. Similarly, reference the member resource type for the `/{transaction_id}`: nested resource with custom mappings and include only the get method.

```
105     /{transaction_id}:
106         type:
107             member:
108                 exampleValue: !include examples/TransactionExample.raml
109                 customErrorHandlerType: CustomErrorMessage
110             get:
```

View the API Console

30. In API Console, click the top left menu icon to view the list of resources.

31. Locate and expand the `/accounts/{account_id}` resource.



32. Click the GET method link.

33. Scroll down and verify that you see the HTTP 200 response body contains the Account datatype.

The screenshot shows a section of a Swagger UI interface. At the top, there's a navigation bar with three horizontal lines on the left and a circular icon with a checkmark on the right. Below this is a header labeled "Response".

Under the "Response" header, there are two tabs: "200" and "404". The "200" tab is active, showing the following description: "No description provided." Below this, it says "Type: Account".

Below the "Type" label, there are two buttons: "Type" and "Examples". The "Type" button is underlined, indicating it is selected. To the right of these buttons is a small square button with an upward-pointing arrow.

The main content area displays the JSON schema for the "Account" type:

```
{  
    "accountBalance": {  
        "currency": "string",  
        "amount": "string"  
    },  
    "accountNumber": "string",  
    "accountOwner": [  
        {  
            "customerID": "string",  
            "ssn": "string",  
            "displayName": "string"  
        }  
    ],  
    "interestRate": "number",  
    "bank": {  
        "bankCode": "string",  
        "bankName": "string",  
        "routingNumber": "string",  
        "swiftBIC": "string"  
    },  
    "createdAt": "datetime",  
    "IBAN": "string",  
    "accountID": "string",  
    "accountType": "string",  
    "modifiedAt": "datetime"  
}
```

Walkthrough 9-3: 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:

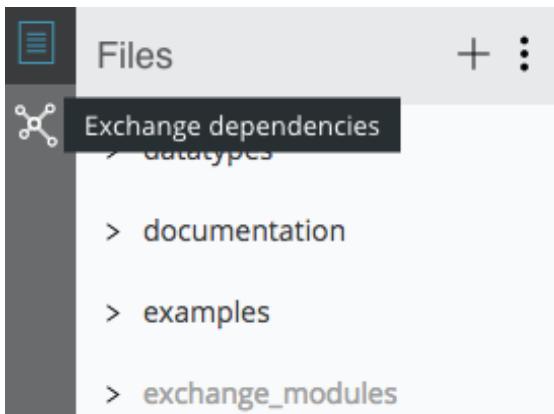
- Consume a cacheable trait asset from Anypoint Exchange.
- Define a flexible content type trait to be applied to the resource method with an Accept header in the request.
- Refactor the resource methods to use these traits.

```
30     get:  
31         description: Retrieve a list of customers  
32         is:  
33             - cacheable  
34             - hasAcceptHeader:  
35                 customErrorDataType: CustomErrorMessage  
36         responses:  
37             200:
```

The screenshot shows the Anypoint Exchange interface. On the left, there is a code editor window displaying the JSON configuration for a trait asset. The configuration includes a 'get' method with a description of 'Retrieve a list of customers'. It uses two traits: 'cacheable' and 'hasAcceptHeader'. The 'hasAcceptHeader' trait specifies a custom error message type. The 'responses' section contains a single '200' entry. On the right, there is a preview pane for the '/customers : get' endpoint. The preview shows the endpoint URL as `GET https://mocksvcs.mulesoft.com/mocks/ed87b212-aa47-410b-9e19-56f864b5bf09/mocks/5c81644c-33c9-435d-96a7-a811a5e969e8/customers`. Below the URL, there is a 'Request' section with a 'Headers' table. The table has one row with 'Accept' as the parameter, 'string' as the type, and 'Specify the media type of the response to be returned' as the description. A 'Try it' button is located at the top right of the preview pane.

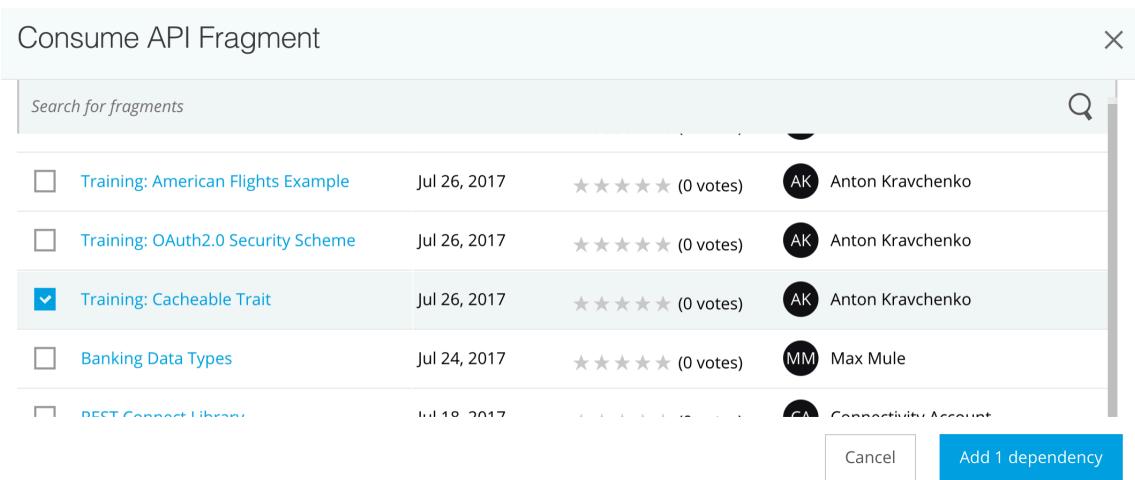
Consume a cacheable trait asset from Anypoint Exchange 2.0

1. Return to API designer.
2. In the file browser section, click the Exchange dependencies icon.



3. Click the + icon next to the Dependencies header.

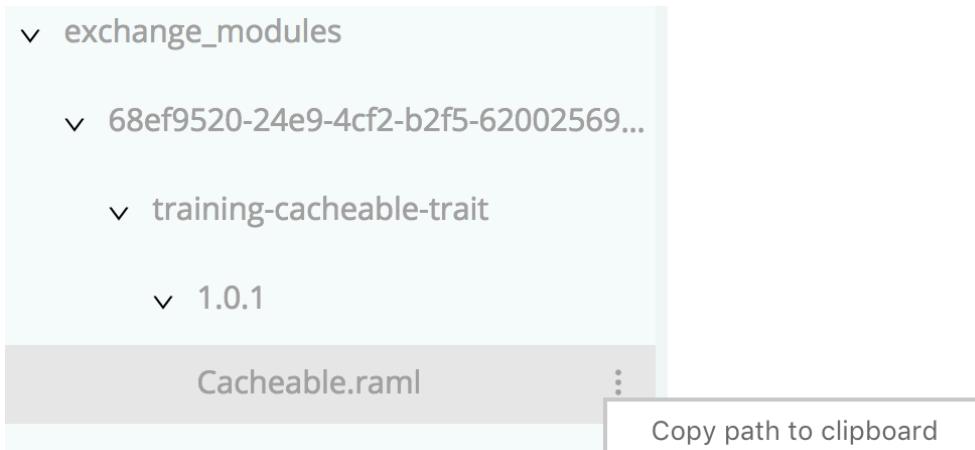
- In the Consume API Fragment dialog box, locate Training: Cacheable Trait and check the box.



- Click Add Dependency.

Reference the cacheable trait inside a RAML API Specification

- In the file browser section, click the files menu icon.
 - In the file browser section, click acme-banking-api.raml ;if the file is not open in the RAML editor.
 - Locate the line that includes the member resource type file and add two new lines below it.
 - Remove indentation in the second new line created.
 - From the shelf, click traits.
 - In the new line, type:
- ```
cacheable: !include
```
- In the file browser section, expand the exchange\_modules folder to locate the Cacheable.raml trait file.
  - Click the menu icon next to the file name and click Copy path to clipboard.



14. In the RAML editor, paste the path to the traits file after the !include keyword.

*Note: If you imported the Cacheable.raml file from the studentFiles folder into a traits folder, include the path as traits/Cacheable.raml.*

## Refactor a resource method to use the cacheable trait

15. In the /customers resource get method 200 response, delete the lines that define the response headers.

```
35 responses:
36 200:
37 headers:
38 Cache-Control:
39 description: |
40 Activates caching and defines cache behavior through cache response directives.
41 Usually defines public or private (cacheable by proxy or not) and max-age for res
42 See http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html for more information.
43 example: private, max-age=31536000
44 Expires:
45 type: datetime
46 description: |
47 Sets a date in RFC 1123 format from which the cached resource should no longer be
48 If both the Expires header and max-age in the Cache-Control header are set, max-a
49 example: Tue, 17 Jul 2017 09:30:41 GMT
50 format: rfc2616|
```

16. Add a new line below the line that contains the displayName for the get method.

17. In the shelf below, click is.

18. In the shelf, click cacheable.

```
29 get:
30 description: Retrieve a list of customers
31 displayName: Get all customers
32 is: |
33 headers:
```

Others

| cacheable

19. In the API Console, if you see an error that the 'Property is should be an array', change the is node value to the following:

```
is:
 - cacheable
29 get:
30 description: Retrieve a list of customers
31 displayName: Get all customers
32 is:
33 - cacheable
34 headers:
```

## Define an Accept header trait to apply to resource methods to support flexible content types

20. In the file browser, click the + icon in the Files header.

21. Select New folder.

22. In the Add new folder dialog box, type the name as traits.

*Note: If you have already created a traits folder, do not recreate another one.*

23. In the file browser, locate the traits folder and click the + icon.

24. Select New file.

25. In the Add new file dialog box, set the type of file to Trait.

26. Rename the file to hasAcceptHeader.raml and click Create.

27. In the file browser, click acme-banking-api.raml.

28. Go to the /customers resource get method.

29. Select the lines from headers to the example line for the Accept header and copy them.

```
29 get:
30 description: Retrieve a list of customers
31 is:
32 - cacheable
33 headers:
34 Accept?:
35 description: Specify the media type of the response to be returned
36 example: application/xml
37 responses:
38 200:
```

30. In the file browser, click hasAcceptHeader.raml and paste the lines in the second line.

*Note: Fix the indentation.*

```
1 #%RAML 1.0 Trait
2 headers:
3 Accept?:
4 description: Specify the media type of the response to be returned
5 example: application/xml
6 responses:
```

31. In the file browser, click acme-banking-api.raml.

32. In the /customers get method responses, copy the HTTP 406 response lines.

```
45 404:
46 body:
47 type: CustomErrorMessage
48 406:
49 body:
50 type: CustomErrorMessage
51 /{customer_id}:
```

33. In the file browser section, click hasAcceptHeader.raml.

34. Paste the copied lines below the responses node.

*Note: Fix the indentation.*

35. Go to the line in the 406 response body that contains the type and replace the type value with a custom mapping parameter.

```
7 406:
8 body:
9 type: <<customErrorDataType>>|
```

36. In the file browser, click acme-banking-api.raml.

37. Locate the line that includes the cacheable trait and add a new line below it.

38. Include the hasAcceptHeader trait.
39. Go to the /customers resource get method and delete the lines that define the Accept header and the HTTP 406 response.

```
30 get:
31 description: Retrieve a list of customers
32 is:
33 - cacheable
34 responses:
35 200:
36 body:
37 application/json:
38 type: Customer[]
39 application/xml:
40 type: Customer[]
41 404:
42 body:
43 type: CustomErrorMessage
```

40. In the /customers resource get method, go to the line that contains the cacheable trait applied and change the line to:

```
is:
- cacheable
- hasAcceptHeader:
 customErrorDataType: CustomErrorMessage
30 get:
31 description: Retrieve a list of customers
32 is:
33 - cacheable
34 - hasAcceptHeader:
35 customErrorDataType: CustomErrorMessage
36 responses:
37 200:
```

## View the API Console

41. In the API Console, click the top left menu icon.
42. Locate the /customers resource and click the GET method link.

43. Verify that you can view Traits applied at the top of the resource method details page.

/customers : get

Traits: cacheable, hasAcceptHeader

Retrieve a list of customers

Request

GET <https://mocksvc.mulesoft.com/mocks/ed87b212-aa47-410b-9e19-56f864b5bf09/mocks/5c81644c-33c9-435d-96a7-a811a5e969e8/customers>

Headers

Hide ^

| Parameter | Type   | Description                                                                             |
|-----------|--------|-----------------------------------------------------------------------------------------|
| Accept    | string | Specify the media type of the response to be returned<br>Example value: application/xml |

# Module 10: Modularizing APIs

```
1 #%RAML 1.0 Library
2 usage: Use the following traits request and response headers and response status
3
4 traits:
5 cacheable: !include ../exchange_modules/68ef9520-24e9-4cf2-b2f5-620025690913/
6 hasAcceptHeader: !include ../traits/hasAcceptHeader.raml
7 hasGetResponse:
8 responses:
9 200:
10 body:
11 type: <>resourcePathName | !singular<>
12 404:
13 body:
14 type: <>customErrorDataType>>
15
16 #%RAML 1.0 Overlay
17 extends: ../acme-banking-api.raml
18 usage: Spanish localization
19
20 documentation:
21 - title: ACME API Bancario Home
22 content: |
23 **ACME Banca API** permite a los desarrolladores crear aplicaciones que hacen uso de la i
24 Esta API contiene una funcionalidad que permite a los desarrolladores recuperar y manipul
25 - title: Banco ACME Headline
26 content: |
27
28 #%RAML 1.0 Extension
29 extends: ../acme-banking-api.raml
30 baseUri: <>insert prod specific implementation URL>>
```

## 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 the following traits request and response headers and response status
3
4 traits:
5 cacheable: !include ../exchange_modules/68ef9520-24e9-4cf2-b2f5-620025690913/
6 hasAcceptHeader: !include ../traits/hasAcceptHeader.raml
7 hasGetResponse:
8 responses:
9 200: 31 is:
10 | body: 32 - Traits.cacheable
11 | type: <<resourcePathName | !singl 33 - Traits.hasAcceptHeader:
12 | 404: 34 customErrorDataType: CustomErrorMessage
13 | body: 35 responses:
```

### Create a library file

1. Return to API designer.
2. In the file browser, click the + icon in the Files header section.
3. Select New folder.
4. In the Add new folder dialog box, type the name of the folder as libraries.
5. Click Create.
6. In the file browser, locate the library folder and click the + icon.
7. Select New file.
8. In the Add new file dialog box, select the file type as Library.
9. Rename the file as TraitsLibrary.raml
10. In the RAML editor, go to the usage node and type the value as:

Use the following traits to define request and response headers and response status information

```
1 #%RAML 1.0 Library
2 usage: Use the following traits to define request and response headers and response status information
```

## Define traits for response information in the library file

11. In the traitsLibrary.raml file, add two new lines.

12. In the shelf, click traits.

13. In the new line type:

```
hasGetResponse:
```

14. Add a new line below.

15. In the file browser, select acme-banking-api.raml.

16. Go to the /accounts nested resource get method, and cut the lines from the responses to the HTTP 404 response body type CustomErrorMessage.

```
54 /accounts:
55 get:
56 responses:
57 200:
58 body:
59 type: Account[]
60 404:
61 body:
62 type: CustomErrorMessage
63
```

17. In file browser, select TraitsLibrary.raml and press tab in the new line.

18. Paste the copied lines.

*Note: Fix the indentation.*

```
4 hasGetResponse:
5 responses:
6 200:
7 body:
8 type: Account[]
9 404:
10 body:
11 type: CustomErrorMessage
```

19. Go to the line that contains the HTTP 200 response body type Account[] and replace it with:

```
type: <>resourcePathName | !singularize | !uppercase>>[]
6 200:
7 body:
8 type: <>resourcePathName | !singularize | !uppercase>>[]
9 404:
```

20. In the HTTP status 404 body type, delete the reference to CustomErrorMessage datatype, and type <>customErrorDataType>>.

```
9 404:
10 body:
11 type: <>customErrorDataType>>|
```

## Apply the traits in the main RAML API definition

21. In the file browser, select acme-banking-api.raml.  
22. Cut the cacheable and hasAcceptHeader reference lines and delete them from acme-banking-api.raml.

*Note: Delete the traits node entirely in the acme-banking-api.raml file.*

23. In the file browser, select traitsLibrary.raml and add the copied lines before the hasGetResponse trait definition.  
24. To get rid of the error in lines that include the traits fragment, add .. / before the trait's path in the include statement.  
25. In the file browser, select acme-banking-api.raml.  
26. Go to the empty line after the resourceTypes definition and add a new line.  
27. In the shelf, click uses.

```
17 resourceTypes:
18 collection: !include resourceTypes/collection.raml
19 member: !include resourceTypes/member.raml
20
21 |
22 /customers:
23 type:
```

| Others          | Schemas | Types And Traits |
|-----------------|---------|------------------|
| uses            | schemas | traits           |
| types           |         | resourceTypes    |
| annotationTypes |         |                  |

28. In the new line type:

Traits: libraries/traitsLibrary.raml.

*Note: Add empty lines below the Traits definition and above the uses node to increase readability.*

29. In the /customers resource get method, go to the line that applies the traits cacheable and hasAcceptHeader and add Traits. before the traits name:

```
is:
 - Traits.cacheable
 - Traits.hasAcceptHeader
 customErrorMessageType: CustomErrorMessage
31 is:
32 - Traits.cacheable
33 - Traits.hasAcceptHeader:
34 customErrorDataType: CustomErrorMessage
35 responses:
```

30. In the /accounts nested resource get method.

```
is:
 - Traits.hasGetResponse:
 customErrorDataType: CustomErrorMessage
55 is:
56 - Traits.hasGetResponse:
57 customErrorDataType: CustomErrorMessage
```

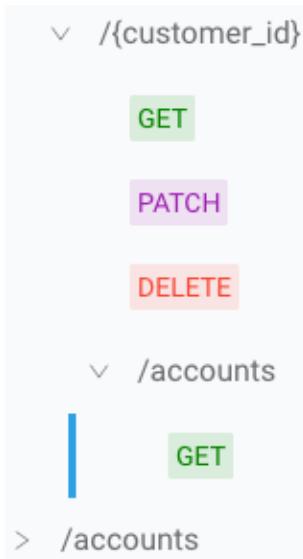
31. Similarly, in the /transactions nested resource get method, delete the lines that define the get method response and apply the hasGetResponse trait.

```
72 /transactions:
73 get:
74 is:
75 - Traits.hasGetResponse:
76 customErrorDataType: CustomErrorMessage
77
78 /transactions:
```

## View the API Console

32. In API Console, click the top left menu icon to open the list of resources.

33. Locate and expand the /customers/{customer\_id}/accounts resource and click GET.



34. Verify that the hasGetResponse trait has been applied to the resource.

The screenshot shows the detailed configuration for the '/accounts : get' resource. At the top, there's a 'Try it' button. Below it, under 'Request', is a 'GET' method with the URL `https://mocksvc.mulesoft.com/mocks/ed87b212-aa47-410b-9e19-56f864b5bf09/mocks/5c81644c-33c9-435d-96a7-a811a5e969e8/customers/{customer_id}/accounts`. Under 'Parameters', there's a 'customer\_id' parameter listed as 'string (required)'. In the 'Response' section, the '200' status code is shown with the note 'No description provided.' and a 'Type: application/json' entry. A JSON snippet is partially visible: `[ { "accountBalance": { "currency": "string", "amount": "string" } }`.

## 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.

The screenshot shows a code editor with a RAML 1.0 Overlay file. The code includes documentation for an API endpoint, with some text highlighted in red. To the right, a preview pane shows the generated API documentation in Spanish, including a title, content, and a detailed description of the API's functionality.

```
1 #%RAML 1.0 Overlay
2 extends: ../acme-banking-api.raml
3 usage: Spanish localization
4
5 documentation:
6 - title: ACME API Bancario Home
7 content: |
8 **ACME Banca API** permite a los desarrolladores crear aplicaciones que ...
9
10 Esta API contiene una funcionalidad que permite a los desarrolladores ...
11 - title: Banco ACME Headline
12 content: |
13 **Banco ACME** es una _multinacional de servicios bancarios_ y ...
14
```

ACME Banca API permite a los desarrolladores crear aplicaciones que ...  
Esta API contiene una funcionalidad que permite a los desarrolladores ...  
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. In the file browser section, click the + icon next to the Files header and select New folder.
3. In the Add new folder dialog box, set the name to overlays and click Create.
4. In the file browser, click the + icon next to the overlays folder and select New file.
5. In the Add new file dialog box, select the file type as Overlay and set the name to internationalization.raml and click Create.

The dialog box has the following fields:  
Type: RAML 1.0  
Specification: ⓘ ▾  
Name: internationalization.raml  
Buttons: Cancel, Create

## Add documentation and description nodes to support Spanish localization

6. In the second line that contains the extends node, type the value as

```
./acme-banking-api.raml
1 #%RAML 1.0 Overlay
2 extends: ./acme-banking-api.raml
```

7. Enter a new line before the extends node and type:

usage: Spanish localization.

8. Return to the APDevApiDesign3.8\_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 extends: ./acme-banking-api.raml
3 usage: Spanish localization
4
5 documentation:
6 - title: ACME API Bancario Home
7 content: |
8 **ACME Banca API** permite a los desarrolladores crear aplicaciones que hacen uso de la i
9
10 Esta API contiene una funcionalidad que permite a los desarrolladores recuperar y manipul
11 - title: Banco ACME Headline
12 content: |
13 **Banco ACME** es una _multinacional de servicios bancarios_ y financieros de la organiza
14
```

10. From the APDevApiDesign3.8\_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.

*Note: If you observe errors that mention that the mentions invalid media type at..., there is an issue with the RAML JS parser. The fix will be available soon. Till then, you could add the media type application/json with every request and response body in the acme-banking-api.raml file.*

## View the documentation

13. In API Console, click the top-left menu icon.
14. Verify that the documentation, resources, and types show up like the acme-banking-api.raml API Console.

15. Go to the Documentation section and expand the documentation title to view the Spanish localization of the document.
16. Go to the /customers resource and click GET.
17. Verify you can view the GET method description in Spanish.
18. Click the top-left menu icon to go one level back.

## Walkthrough 10-3: Define and use API extensions to promote portability to test in multiple environments

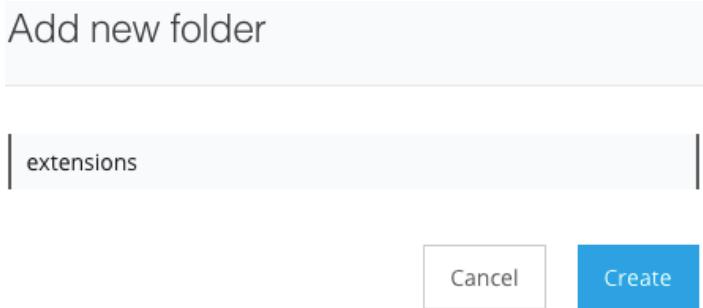
In this walkthrough, you will define an extension. You will:

- Create API extension files.
- Add a baseUri parameter to the extension files and enter a placeholder for environment specific implementation URL.



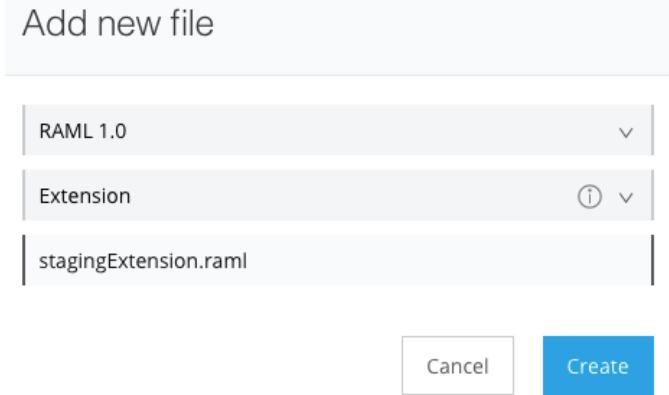
### Create API extension files

1. Return to API designer.
2. In the file browser, click the + icon in the Files header section and select New folder.
3. In the Add new folder dialog box, set the name to extensions and click Create.

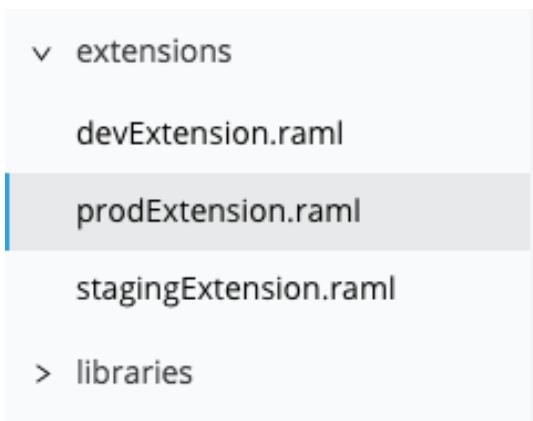


4. Click the + icon next to the extensions folder and select New file.

5. In the Add new file dialog box, set the file type to Extension and set the name to stagingExtension.raml and click Create.



6. Create two more extension files 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, locate the extensions folder and select stagingExtension.raml.
8. Go to the line that contains the extends node and type the value as:

```
./acme-banking-api.raml
1 #%RAML 1.0 Extension
2 extends: ./acme-banking-api.raml
```

9. In API Console, turn on the mocking service at the top right corner.

10. Verify that you see the baseUri added to the extension file.

```
1 #%RAML 1.0 Extension
2 baseUri: https://mocksvc.mulesoft.com/mocks/eecb5519-8cb0-4a1c-a3e6-fc66d26ca719 #
3 extends! ../acme-banking-api.raml
```

ACME Banking API

*Note: You can replace the Mock service URL with a staging environment specific implementation URL.*

11. In API Console, click the top left menu icon and click GET for the /accounts/{account\_id} resource.
12. Click Try It.
13. Enter the account\_id value as 12345 and click GET.
14. Verify that you see the response structure of the get method.

The screenshot shows the Mule API Console interface. At the top, there's a navigation bar with a back arrow, a shield icon, and a checkmark icon. Below it, the 'Request URL' field contains the URL <https://mocksvc.mulesoft.com/mocks/5cf0cae2-02bc-47b1-a>. The main area has tabs for 'Parameters' and 'Headers', with 'Parameters' being active. Under 'Parameters', there's a section for 'URI parameters' with a single entry: 'account\_id\*' set to '12345'. At the bottom right of this section is a 'Send' button. Below the parameters, the response status is shown as '200 OK' with a time of '363.39 ms'. The response body is displayed as a JSON object:

```
{
 "accountID": "12345",
 "accountType": "Savings",
 "accountNumber": "1234567890",
 "accountOwner": [Array[1]
 -0: {
 "customerID": "8f19cb50-3f57-4d38",
 "displayName": "John Doe",
 "ssn": "123-456-7890"
 }
],
 "accountBalance": {
 "currency": "USD",
 "amount": "8457.90"
 },
 "IBAN": "GB29NWBK60161331926820",
 "bank": {
 "bankCode": "NWBKG2L",
 "branchCode": "1234567890",
 "name": "National Westminster Bank PLC",
 "address": "1 Queen Victoria Street, London EC4R 1QS, United Kingdom",
 "zip": "EC4R 1QS",
 "city": "London",
 "state": "Greater London",
 "country": "United Kingdom",
 "phone": "+44 20 7631 1234",
 "fax": "+44 20 7631 1234",
 "email": "info@nwbk.co.uk",
 "url": "http://www.nwbk.co.uk",
 "notes": "This is a test bank account.",
 "status": "Active",
 "type": "Commercial",
 "category": "Bank",
 "subcategory": "Current Account",
 "parent": null
 }
}
```

15. Similarly, go to the devExtension.raml file and include the acme-banking-api.raml file in the extends node.

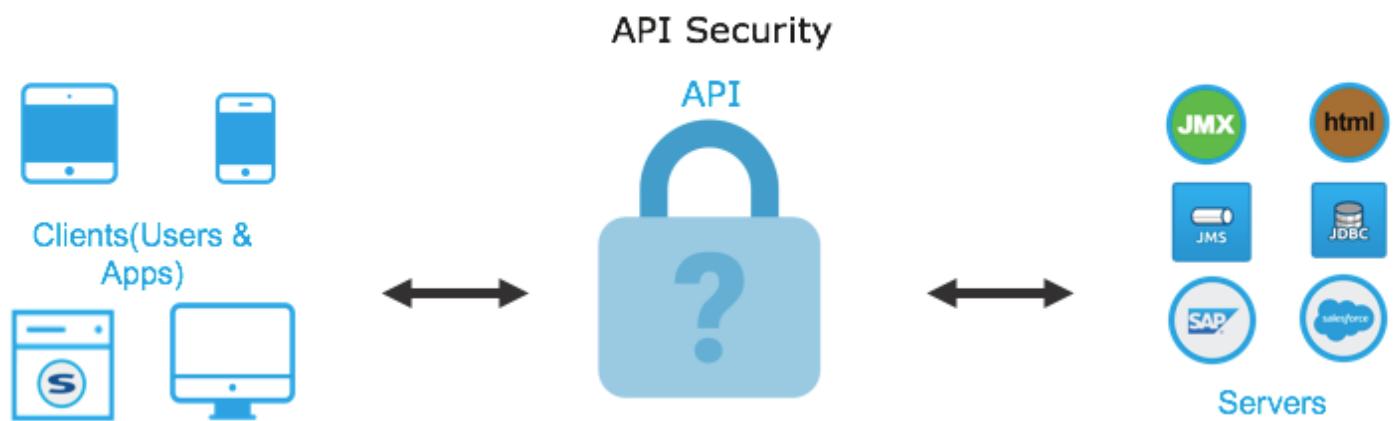
16. Add a new line before the extends node, and type the following:

```
baseUri: <<insert dev environment specific implementation URL>>
1 #%RAML 1.0 Extension
2 extends: ../acme-banking-api.raml
3 baseUri: <<insert dev environment specific implementation URL>>
```

17. Similarly, in the prodExtension.raml file, add in a placeholder for the baseUri and reference acme-banking-api.raml in the extends node.

```
1 #%RAML 1.0 Extension
2 extends: ../acme-banking-api.raml
3 baseUri: <<insert prod specific implementation URL>>
```

# Module 11: Securing APIs



## Objectives:

- Define API security requirements.
- Use security schemes to apply resource and method level policies.
- Define custom security scheme for APIs.
- Apply an OAuth2.0 external provider policy to resource methods.

## 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.

The screenshot shows the API designer interface. On the left, a code editor displays a portion of a RAML file:

```
24 securitySchemes:
25 customTokenSecurity: !include securitySchemes/customTo
26
27 securedBy: customTokenSecurity
28
29 /customers:
30 type:
31 collection:
32 customErrorDataType: CustomErrorMessage
```

To the right of the code editor, the API interface is displayed. It includes:

- A header bar with a toggle icon.
- A URL field: `/customers : Get all customers`.
- A "Try it" button.
- A description area: "Traits: Traits.cacheable, Traits.hasAcceptHeader" and "Retrieve a list of customers".
- A "Request" section showing a GET method: `GET https://mocksvc.mulesoft.com/mocks/5cf0cae2-02bc-47b1-a384-5a4676bb2228/customers`.
- A "Headers" section with a table:

| Parameter     | Type              | Description                                                                             |
|---------------|-------------------|-----------------------------------------------------------------------------------------|
| Accept        | string            | Specify the media type of the response to be returned<br>Example value: application/xml |
| Authorization | string (required) | This header should contain a valid security token                                       |

### Create a security scheme file

1. Return to API designer.
2. In the file browser section, click the + icon in the Files header section.
3. Select New folder and in the Add new folder dialog box, enter the name as `securitySchemes`.
4. Click Create.
5. Click the + icon next to the `securitySchemes` folder and select New file.
6. In the Add new file dialog box, select the file type as Security Scheme and rename the file as `customTokenSecurity.raml`.

The dialog box has the following fields:

- A title bar: "Add new file".
- A dropdown menu: "RAML 1.0".
- A dropdown menu: "Security Scheme".
- A text input field: "customTokenSecurity.raml".
- Buttons at the bottom: "Cancel" and "Create".

7. Click Create.

## Define a custom security scheme

8. In the RAML editor, go to the line that contains type node and press space bar after the colon.
9. In the shelf below the editor, click x-{other}.

```
1 #%RAML 1.0 SecurityScheme
2 type: |
```

| Others        | Security                                      |
|---------------|-----------------------------------------------|
| Define Inline | OAuth 2.0                                     |
| Pass Through  | OAuth 1.0                                     |
| x-{other}     | Basic Authentication<br>Digest Authentication |

10. Replace {other} with customToken.
11. Add a new line below the type node.
12. In the shelf, click description.
13. Type the value of the description node as:

This security scheme validates requests to the API using a token provided in the request header

14. Press enter to add a new line.

```
1 #%RAML 1.0 SecurityScheme
2 type: x-customToken
3 description: This security scheme validates requests to the API using
4 |
```

15. In the shelf, click describedBy.

16. In the shelf, click headers.

```
1 #%RAML 1.0 SecurityScheme
2 type: x-customToken
3 description: This security scheme
4 describedBy:
5
```

The screenshot shows a RAML editor interface. At the top, there is a code editor with the following content:

```
1 #%RAML 1.0 SecurityScheme
2 type: x-customToken
3 description: This security scheme
4 describedBy:
5
```

Below the code editor is a navigation bar with tabs: "Parameters", "Others", "Responses", and "responses". The "headers" node is currently selected, indicated by a blue vertical bar on its left. The "Responses" tab is also highlighted with a blue bar.

17. In the shelf, click Authorization.

18. In the shelf, click description.

```
4 describedBy:
5 headers:
6 Authorization:
7
```

The screenshot shows a RAML editor interface. At the top, there is a code editor with the following content:

```
4 describedBy:
5 headers:
6 Authorization:
7
```

Below the code editor is a navigation bar with tabs: "Docs", "Others", and "xml". The "description" node is currently selected, indicated by a blue vertical bar on its left. The "xml" tab is also highlighted with a blue bar.

19. Type the value of the description node as: This header should contain a valid security token.

20. Press enter to add a new line.

21. In the shelf, click type.

22. In the shelf, click string.

## Reference the custom security scheme file in the main API definition

23. In the file browser, select acme-banking-api.raml.

24. Go to the empty line after the line that references the traitsLibrary file.

25. Add two new lines below it.

26. Remove indentation in the second new line created.

27. From the shelf, click securitySchemes.

```
21 uses:
22 Traits: libraries/TraitsLibrary.raml
23
24
25
--
```

| Others          | Schemas | Types And Traits | Security        |
|-----------------|---------|------------------|-----------------|
| uses            | schemas | traits           | securitySchemes |
| types           |         | resourceTypes    | securedBy       |
| annotationTypes |         |                  |                 |

28. In the new line, type:

```
customTokenSecurity: !include securitySchemes/customTokenSecurity.raml
```

```
21 uses:
22 Traits: libraries/TraitsLibrary.raml
23
24 securitySchemes:
25 customTokenSecurity: !include securitySchemes/customTokenSecurity.raml
26
27 /customers:
```

## Apply the custom security scheme to all the resource methods in the API

29. Go to the empty line before the /customers resource and press enter.

30. In the shelf, click securedBy.

```
27 |
28 /customers:
29 type:
30 collection:
|-----
| type
| annotationTypes
```

| Security        |
|-----------------|
| securitySchemes |
| securedBy       |

31. In the shelf, click customTokenSecurity.

32. Press enter to add a new line.

```
24 ⊞ securitySchemes:
25 customTokenSecurity: !include securitySchemes/customTokenSecurity.raml
26
27 ⊞ securedBy: customTokenSecurity
28 |
29 ⊞ /customers:
30 ⊞ type:
31 ⊞ collection:
32 customErrorDataType: CustomErrorMessage
```

## View the documentation

33. In API Console, click the top left menu icon.

34. Click GET for the /customers resource.

35. Locate the Headers section of the Request and verify that you can see the Authorization header field listed.

The screenshot shows the API Console interface. At the top, there's a navigation bar with a menu icon and a search/filter icon. Below it, the URL `/customers : Get all customers` is displayed, along with a blue "Try it" button. Underneath, the description reads: "Traits: Traits.cacheable, Traits.hasAcceptHeader" and "Retrieve a list of customers". The "Request" section shows a GET method pointing to `https://mocksvc.mulesoft.com/mocks/5cf0cae2-02bc-47b1-a384-5a4676bb2228/customers`. The "Headers" section is expanded, showing two parameters:

| Parameter     | Type              | Description                                                                             |
|---------------|-------------------|-----------------------------------------------------------------------------------------|
| Accept        | string            | Specify the media type of the response to be returned<br>Example value: application/xml |
| Authorization | string (required) | This header should contain a valid security token                                       |

*Note: Go to any other resource method and notice the Authorization header added to all the resource methods by adding to the root of the RAML definition. Custom security schemes like the customTokenSecurity are not supported for testing using the Try It option.*

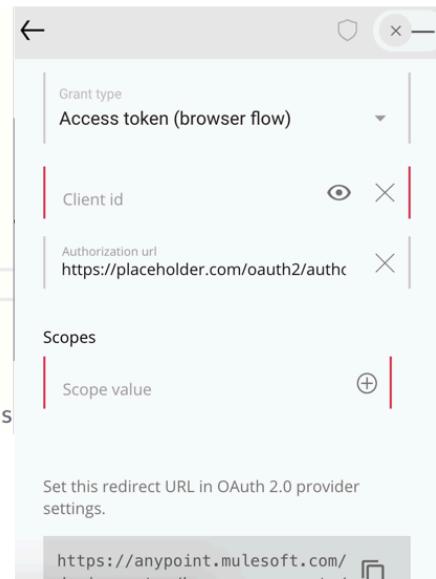
## Walkthrough 11-2: Consume an OAuth2.0 security scheme for an API and secure API resources

In this walkthrough, you define a security scheme to secure API resources using OAuth2.0.

You will:

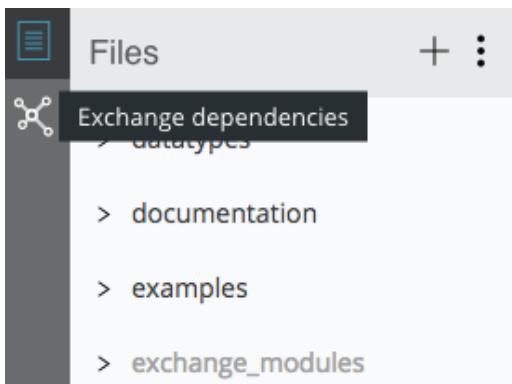
- Consume an OAuth2.0 security scheme fragment file.
- Reference the OAuth2.0 security scheme in the RAML API definition.
- Apply the security scheme in the API resource methods.

```
31 /customers:
32 type:
33 collection:
34 customErrorDataType: CustomErrorMessage
35 post:
36 get:
37 description: Retrieve a list of customers
38 displayName: Get all customers
39 securedBy: oauth2_0
40 is:
41 - Traits.cacheable
42 - Traits.hasAcceptHeader:
43 customErrorDataType : CustomErrorMess
```



### Consume an OAuth2.0 security scheme fragment file

1. Return to API designer.
2. In the file browser section, click the Exchange dependencies icon.



3. Click the + icon next to the Dependencies header.

- In the Consume API Fragment dialog box, locate Training: OAuth2.0 Security Scheme and check the box.

| Name                                                                   | Date Modified | Rating            | Created By          |
|------------------------------------------------------------------------|---------------|-------------------|---------------------|
| <input type="checkbox"/> Training: American Flight Data Type           | Jul 26, 2017  | ★ ★ ★ ★ (0 votes) | AK Anton Kravchenko |
| <input type="checkbox"/> Training: American Flights Example            | Jul 26, 2017  | ★ ★ ★ ★ (0 votes) | AK Anton Kravchenko |
| <input checked="" type="checkbox"/> Training: OAuth2.0 Security Scheme | Jul 26, 2017  | ★ ★ ★ ★ (0 votes) | AK Anton Kravchenko |
| <input type="checkbox"/> Training: Cacheable Trait                     | Jul 26, 2017  | ★ ★ ★ ★ (0 votes) | AK Anton Kravchenko |

Cancel Add 1 dependency

- Click Add Dependency.

## Reference the OAuth 2.0 security scheme inside a RAML API Specification

- In the file browser section, click acme-banking-api.raml ;if the file is not open in the RAML editor.
- Locate the customTokenSecurity include statement and add a new line below it.
- In the new line, type:

```
oauth2_0: !include
```

- In the file browser section, expand the exchange\_modules folder to locate the OAuth2.raml security scheme file.
- Click the menu icon next to the file name and click Copy path to clipboard.

```

 exchange_modules
 68ef9520-24e9-4cf2-b2f5-62002569...
 training-oauth20-security-scheme
 1.0.1
 OAuth2.raml
 ...
 Copy path to clipboard
 b2447622-1656-4ff2-85a7-6a4627f4...
 extensions
 libraries

```

11. In the RAML editor, paste the path to the traits file after the !include keyword.

*Note: If you imported the OAuth2.raml file from the studentFiles folder into a traits folder, include the path as securitySchemes/OAuth2.raml.*

## Secure the resource methods that update bank customer and account information with the OAuth2.0 security scheme

12. In the /{customer\_id} nested resource patch method, add a new line below the line that contains the displayName node.
13. In the shelf, click securedBy.

```
34 get:
35 description: Retrieve
36 displayName: Get all
37 []
38 is:
39 - Traits.cacheable
40 - Traits.hasAcceptHeader
41 customErrorData:
 | headers | us
```

|           |      |
|-----------|------|
| Responses | Body |
| responses | body |

Security  
| securedBy

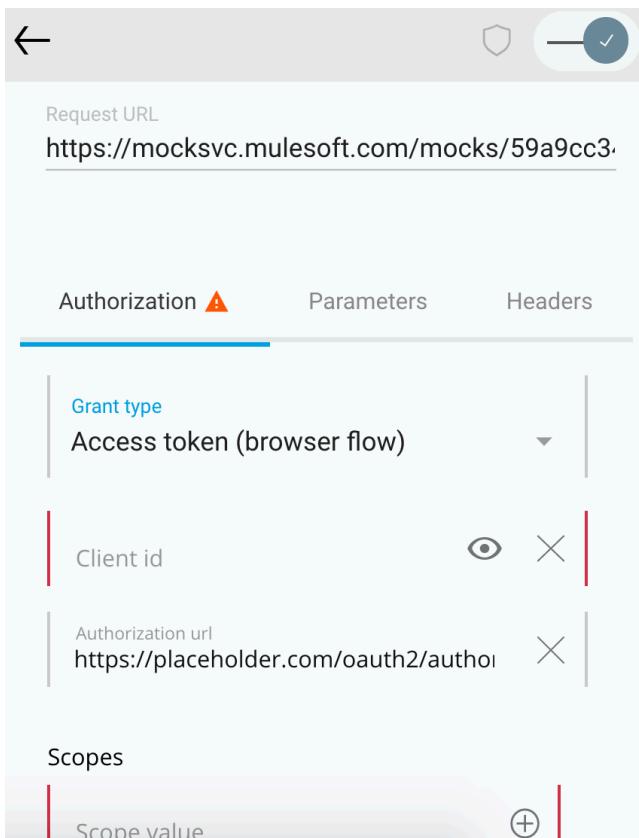
14. In the shelf, click oauth2\_0.
15. In API Console, click the top left menu icon.
16. Click the GET method link for the /customers resource.

17. Scroll to the Request Headers section and verify that you can see an Authorization header.

| Headers                  |        |                                                                                                        |
|--------------------------|--------|--------------------------------------------------------------------------------------------------------|
| Parameter                | Type   | Description                                                                                            |
| Accept                   | string | Specify the media type of the response to be returned<br><br>Example value: application/xml            |
| Authorization (required) | string | Used to send a valid OAuth 2 access token. Do not use with the "access_token" query. string parameter. |

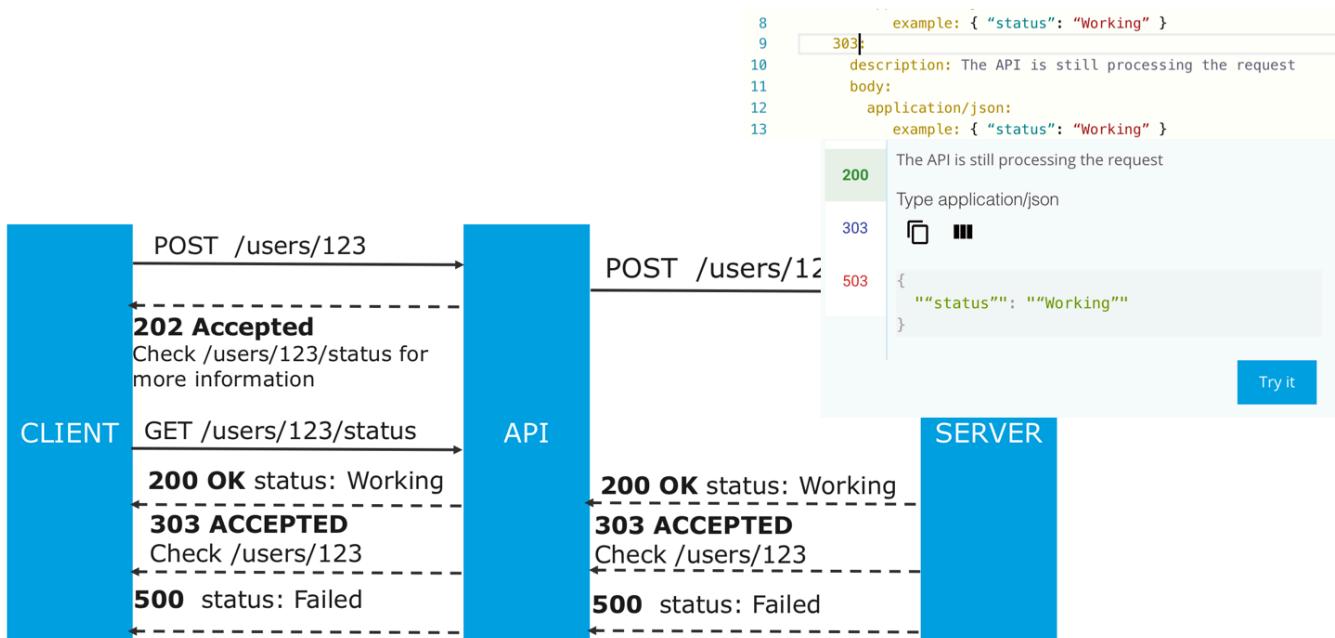
18. Click the Try it button.

19. Select the Authorization and verify that the you need to fill the Auth data to send a request.



The screenshot shows the 'Try it' dialog for a API call. The 'Request URL' is set to <https://mocksvc.mulesoft.com/mocks/59a9cc3>. The 'Authorization' tab is selected, indicated by a blue underline. Below it, there are fields for 'Grant type' (set to 'Access token (browser flow)'), 'Client id', 'Authorization url' (set to <https://placeholder.com/oauth2/autho>), and 'Scopes'. Each field has a delete icon (X) next to it.

# Module 12: Enhancing API Responses using Hypermedia



## Objectives:

- Describe hypermedia.
- Simplify API discoverability using hypermedia.
- Use hypermedia to enhance API responses.
- Modify API definitions to generate state-specific client responses.

## Walkthrough 12-1: Modify an API definition to implement 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 an API designer interface. On the left, there is a code editor with Raml code. Lines 8-13 are shown:

```
8 example: { "status": "Working" }
9
10 303:
11 description: The API is still processing the re
12 body:
13 application/json:
14 example: { "status": "Working" }
```

To the right of the code editor is a 'Response' panel. It contains three sections corresponding to the status codes in the code:

- 200**: The API is still processing the request
- 303**: Type application/json  
A small icon of a document with a progress bar is shown.
- 503**: {  
 "status": "Working"  
}

A 'Try it' button is located at the bottom right of the response panel.

### 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.

```
13 body:
14
15 503:
16 body:
17 type: <<customErrorDataType>>
18
```

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.

9. Press enter.

10. In the shelf, click headers.

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

12. In the shelf click example.

13. Type the value of the example node as:

```
/<<resourcePathName>>/1234/status
14 202:
15 description: The request has been accepted for processing.
16 headers:
17 Location:
18 example: /<<resourcePathName>>/1234/status
```

## Add a new resource to track the state of post requests that are under processing

14. Return to acme-banking-api.raml.

15. Go to the empty line before the /accounts main resource and press enter.

16. Place the cursor in the previous empty line.

17. Press tab to align the cursor with the /accounts nested resource beginning and type:

```
/status:
61 /accounts:
62 get:
63 is:
64 - Traits.hasGetResponse:
65 customErrorDataType: CustomErrorMessage
66 /status:
```

## Add necessary methods and HTTP response statuses to give meaningful information back to the API consumers

18. In the file browser, click the + icon next to the resourceTypes folder and create a new Reaource Type file with the name stateSpecificResourceType.raml.

19. In stateSpecificResourceType.raml, add a new line below the first line.

20. In the shelf, click get.
21. In the shelf, click responses.
22. In the shelf, click 200.
23. In the shelf, click description.
24. 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

```

25. Add a new line and in the shelf, click body.
26. In the shelf, click application/json.
27. In the shelf, click example, and type the value as:

```

{ "status": "Working"}
4
5 200:
6 description: The API is still processin
7 body:
8 application/json:
9 example: { "status": "Working" }

```

28. Press enter to add a new line.
29. Copy the lines containing the details for the 200 response paste it in the new empty line.
30. Press enter and paste the lines again.
31. In the first set of pasted lines, replace 200 with 303 and fix the indentation.

```

8 example: { "status": "Working" }
9
10 303:
11 description: The API is still processing the request
12 body:
13 application/json:
14 example: { "status": "Working" }

```

32. Edit the description node value to:

The processing has finished successfully. Use the URI provided in the Location header for more details.

33. Press enter after the description to add a new line.

34. In the shelf, click headers.
35. In the shelf, click Location.
36. In the shelf, click example, type the value as:

```
/<<resourcePathName>>/1234
10 description: The processing has finished successfully.
11 headers:
12 Location:
13 example: /<<resourcePathName>>/1234
14 body:
```

*Note: Always add a space between the node and the value*

37. In the response JSON example, change the status value from Working to Success.
38. In the second set of pasted lines, change the status code from 200 to 503 and correct the indentation.
39. In the description node below the 503 status code line, change the value of the node to:

The processing has failed

```
17 503:
18 description: The processing has failed
19 body:
20 application/json:
21 example: { "status": "Working" }
```

40. In the response JSON example, change the status value from Working to Failed.
41. Add a new line at the end and place the cursor at the beginning of the line.
42. In the shelf, click delete.
43. In the shelf, click responses.
44. In the shelf, click 200.
45. In the shelf, click description.
46. Type the value of the description node as:

The processing has been canceled

```
22 delete:
23 responses:
24 200:
25 description: The processing has been canceled
```

47. Press enter and copy the lines for the HTTP response body and paste it in the HTTP 200 response body.
48. In the response JSON example, change the status value from Failed to Canceled.

## Include the state specific resource type in the root RAML definition

49. In the file browser, select acme-banking-api.raml.
50. Go to the resourceType include statements at the root and add a new line below the member resourceType.
51. Indent the line and type:

```
stateSpecific: !include resourceTypes/stateSpecificResourceType.raml
17 resourceTypes:
18 collection: !include resourceTypes/collection.raml
19 member: !include resourceTypes/member.raml
20 stateSpecific: !include resourceTypes/stateSpecificResourceType.ra
21
22 uses:
```

52. Add a new line below the line that contains /status nested method and press enter.
53. Press tab and type:

```
type: stateSpecific
67 /status:
68 type: stateSpecific
69
```

54. Copy the lines that define the status resource.
55. Add the /status resource as a nested resource for /accounts and /transactions main resource.
56. In API Console, click the top left menu icon.
57. Click the GET method link under the /customers/{customer\_id}/status resource.

```
▽ /{customer_id}
 GET Get a/an customer by customer ID
 PATCH Update a/an customer by customer ID
 DELETE Delete a/an customer by customer ID
> /accounts
▽ /status
 GET
 DELETE
```

58. Scroll down to the response and verify that you can see the various status codes for the responses.

## Response

**200** The API is still processing the request

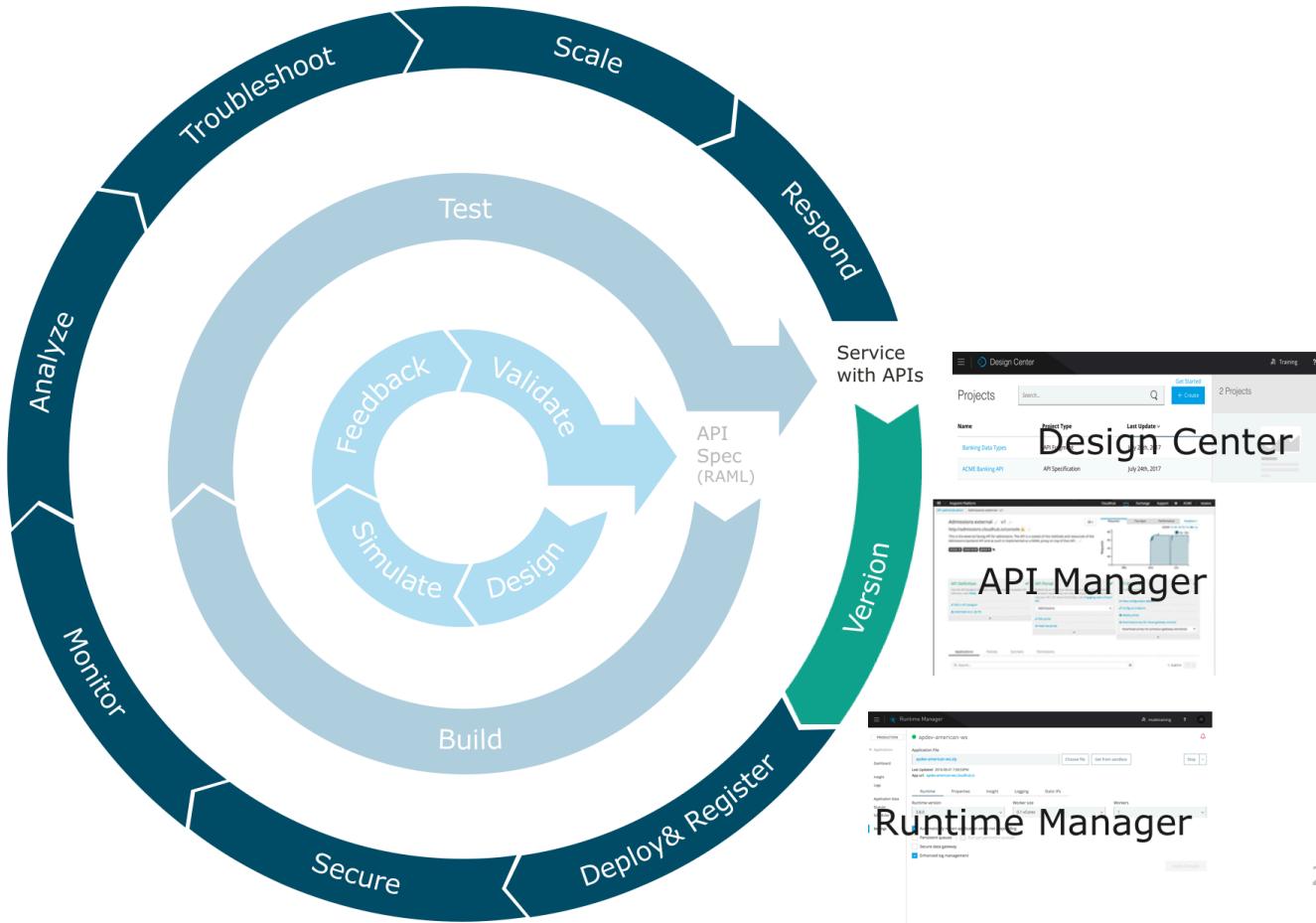
Type application/json

**303**  

**503** {  
 "status": "Working"  
}

Try it

# 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 Design Center.
- Learn how to publish the new version to Exchange and API Manager.
- Deprecate the old version of the API.

Publish API specification to Exchange

Name (required) ACME Banking API Asset version (required) 2.0.0 Current version: 1.0.0

Main file (required) acme-banking-api.raml API version (required) v2 ACME Banking API 1.0

Show advanced > API Status: Unregistered Set the API URL... ACME Banking API helps applications consume information related to bank customers, their accounts and transaction activity in these accounts.

DEPRECATED Export More

v1 create branch v1 from master 83 /transactions: 84 get: 85 ...

### Add a new version of API definition in the API Designer

1. Return to API designer.
2. Click the branch icon on top with the active master branch next to it.

ACME Banking API | master

Files + :

> datatypes

79  
80  
81  
82  
83

3. In the Find or create a branch search text box, type v1 and click create v1 branch from master.

ACME Banking API | master

Files +

> datatypes

> documentation

v1 create branch v1 from master

83 /transactions:  
84 get:  
85 ...

*Note: This way the master branch always points to the most latest version of the API being designed.*

- In the RAML editor, verify that the acme-banking-api.raml file is open.
- Replace the value of the version field with v2.

---

```

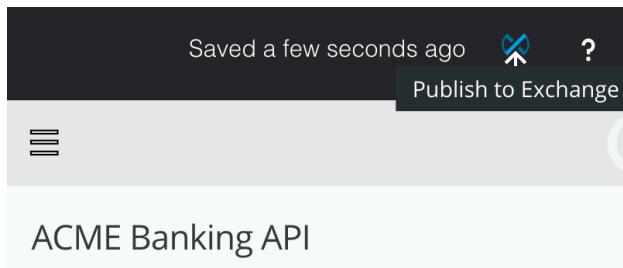
1 #%RAML 1.0
2 baseUri: https://mocksvc.mulesoft.com/mocks
3 version: v2
4 title: ACME Banking API
5 mediaType: application/json

```

*Note: Now you are all set to make changes to the master branch v2 version of the ACME Banking API.*

## Verify that you can publish the new version of the API to Exchange and API Manager

- In the API designer page, click the Publish to Exchange button on the top right corner.



- In the Publish API specification to Exchange dialog box, verify that the asset version and the API version are set to the 2.0 numbers respectively.

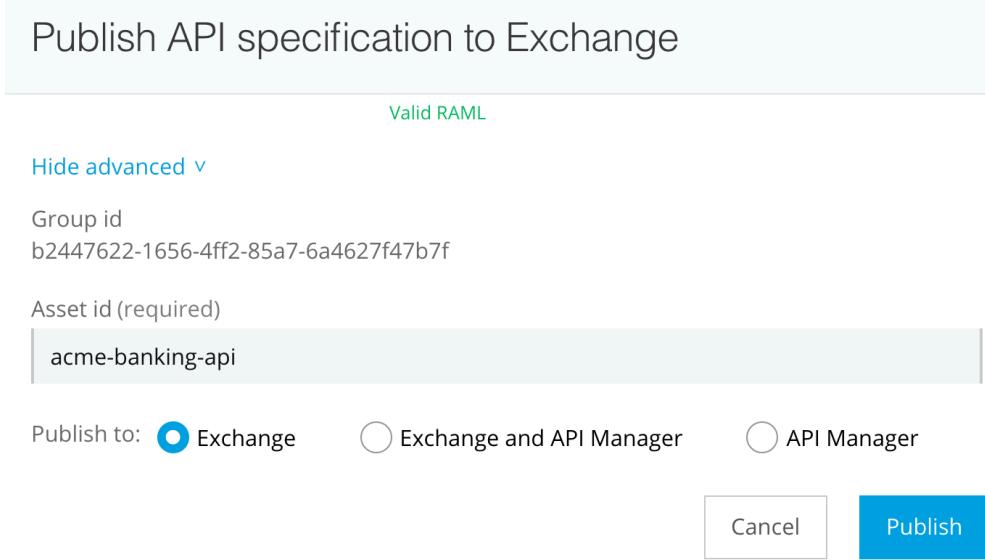
Publish API specification to Exchange

|                                                                              |                            |
|------------------------------------------------------------------------------|----------------------------|
| Name (required)                                                              | Asset version ⓘ (required) |
| ACME Banking API                                                             | 2.0.0                      |
| Current version: 1.0.0                                                       |                            |
| Main file (required)                                                         | API version ⓘ (required)   |
| acme-banking-api.raml                                                        | v2                         |
| Valid RAML                                                                   |                            |
| <a href="#">Show advanced &gt;</a>                                           |                            |
| <input type="button" value="Cancel"/> <input type="button" value="Publish"/> |                            |

*Note: We will not publish v2 into Exchange yet, since v1 is not live and being consumed yet. Only managed and live APIs should be versioned.*

*The new version of an API in Exchange, will have a separate API Portal page for documentation purposes.*

- In the same dialog box, click the Show advanced link.
- Verify that you see the option to publish to API Manager.



*Note: Once the API specification is ready to be implemented, import to API Manager. The API Portal inside API Manager can be shared between API versions. The non-identical items in the shared API Portal will be the version id in the portal link, the API reference page and API Notebooks.*

- Click Cancel to exit out of the dialog box.

## Deprecate the older version of the API

- In API designer, click the top left Design Center icon.
- In the list of projects page in Design Center, click the top left menu icon.
- Click API Manager.
- In API Manager, locate ACME Banking API and click 1.0 version.
- In the API version details page, locate the More button on the top right corner of the page and click it.

ACME Banking API    1.0

API Status:  Unregistered

Set the API URL...

Add a description...

More

- [Export](#)
- [Deprecate](#)
- [Delete](#)
- [Terms & Conditions](#)

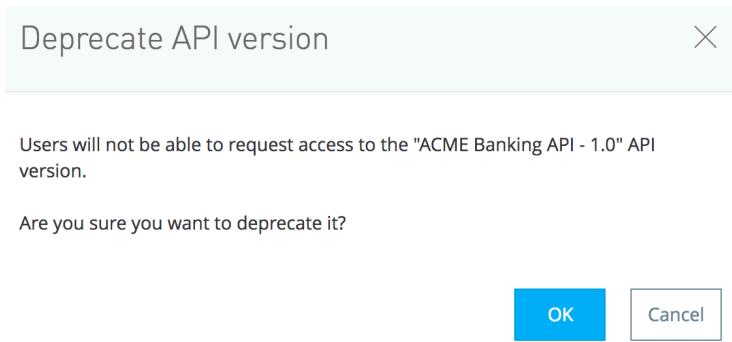
[ADD A TAG](#)

16. Click Deprecate.
17. Verify you see the dialog box that has a warning that users will not be able to access the API version when it is deprecated.



*Note: Deprecation should be performed only when an API is live in production, being used by consumers.*

18. Click OK.



19. Verify that you see a yellow Deprecated tag next to the API name and version.

A screenshot of an API details page for "ACME Banking API". The page shows the API name, version (1.0), and a "DEPRECATED" tag. There are buttons for "Export" and "More". Below the API name, it says "API Status: Unregistered". A text input field "Set the API URL..." is shown. A descriptive text for the API is present, along with a "Edit" icon.

*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.*

*The API Portal also contains the same Deprecated tag.*