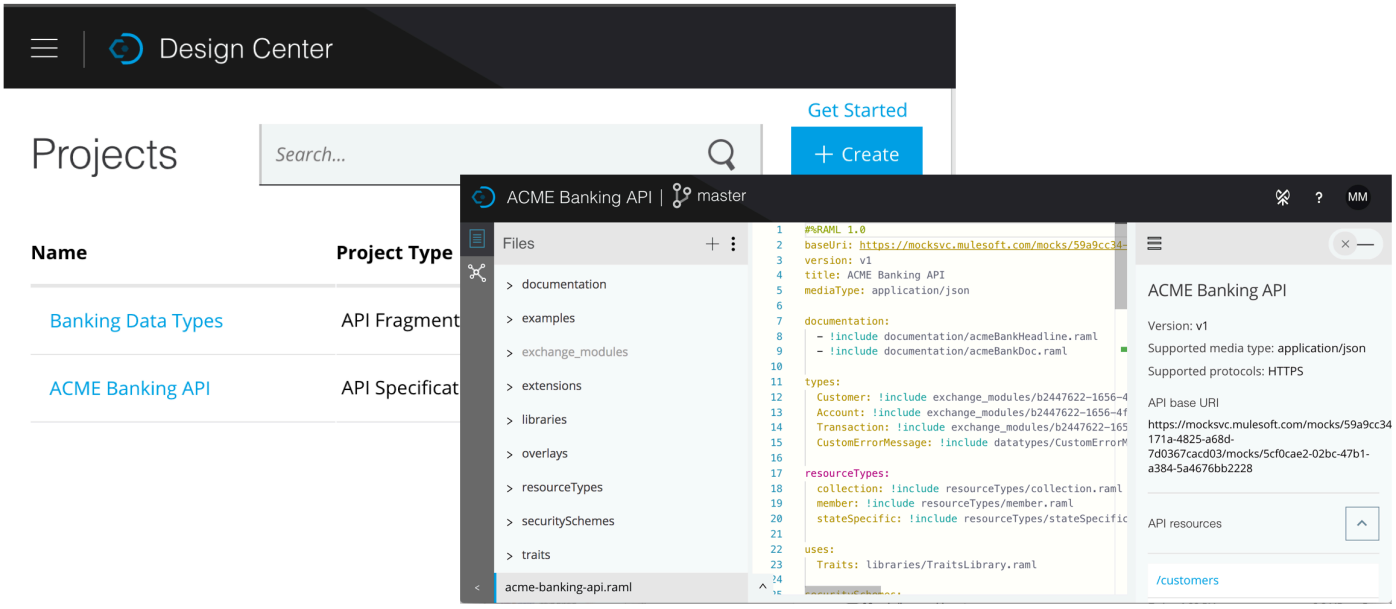
Anypoint Platform Development: API Design

Student Manual

July 30, 2017

Table of Contents

Introducing the Course



Objectives:

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

MuleSoft

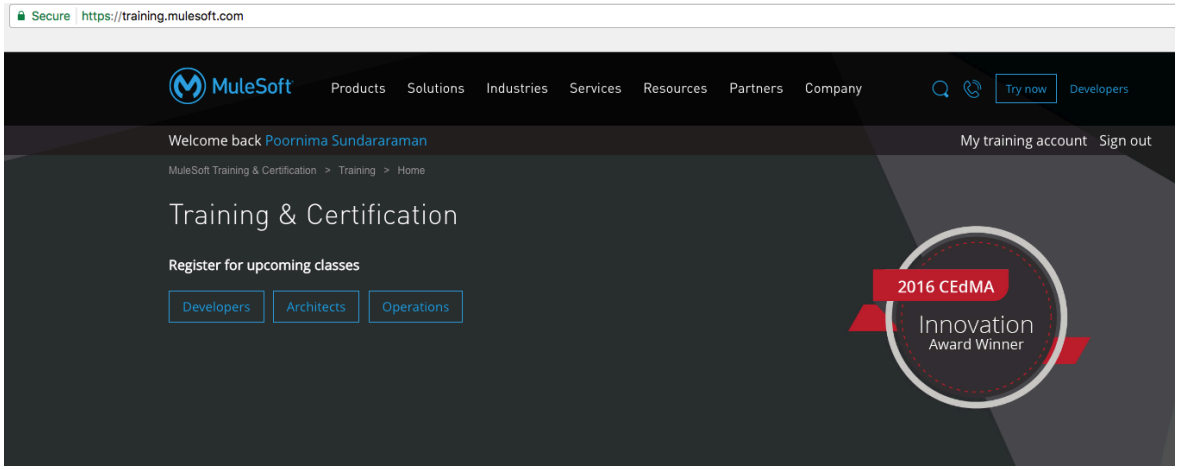
Walkthrough: Set up your computer for class

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

- Download the course files from the MuleSoft Training Learning Management System.

Download student files

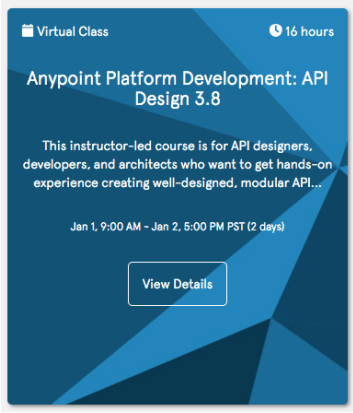
1. In a web browser, navigate to http://training.mulesoft.com.
2. Click the My training account link.



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

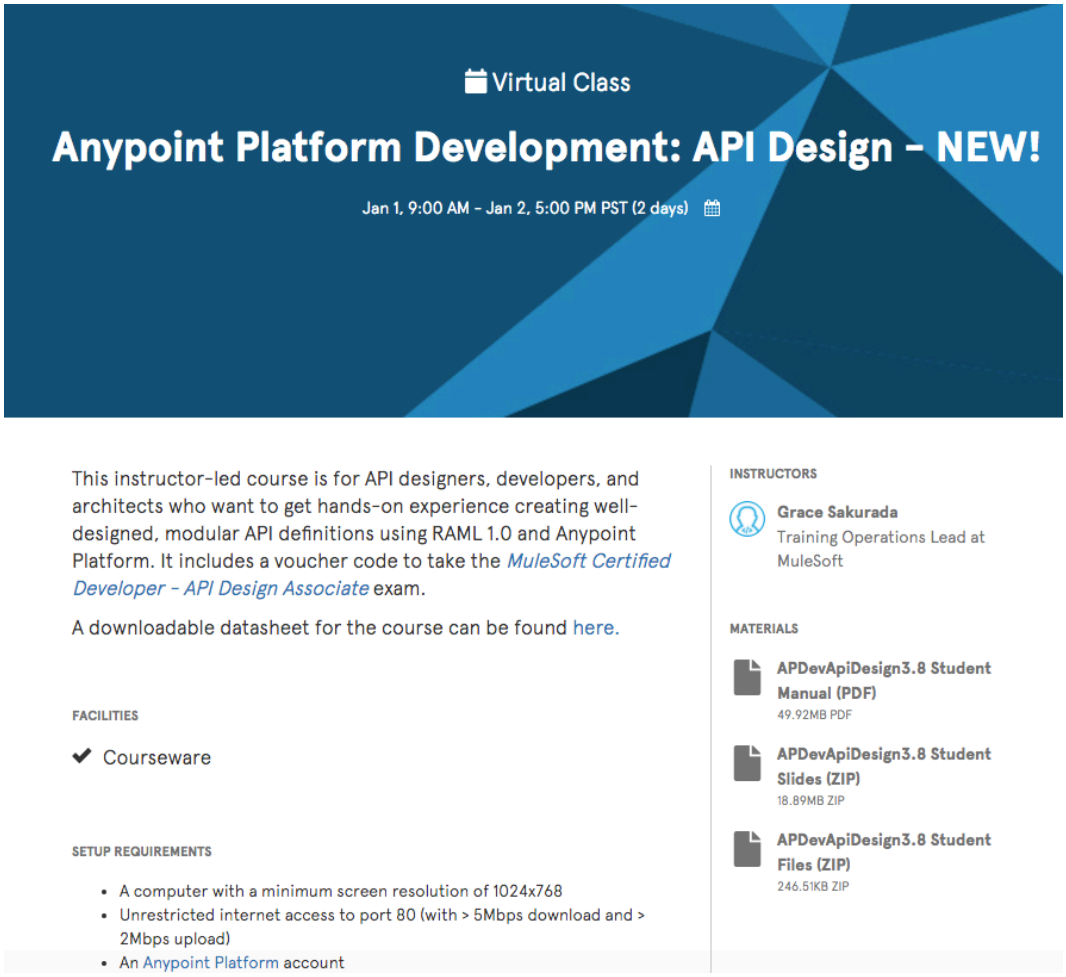
   Note: If you have never logged in before and do not have a password, click the Forgot your password link, follow the instructions to obtain a password, and then log in.

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



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

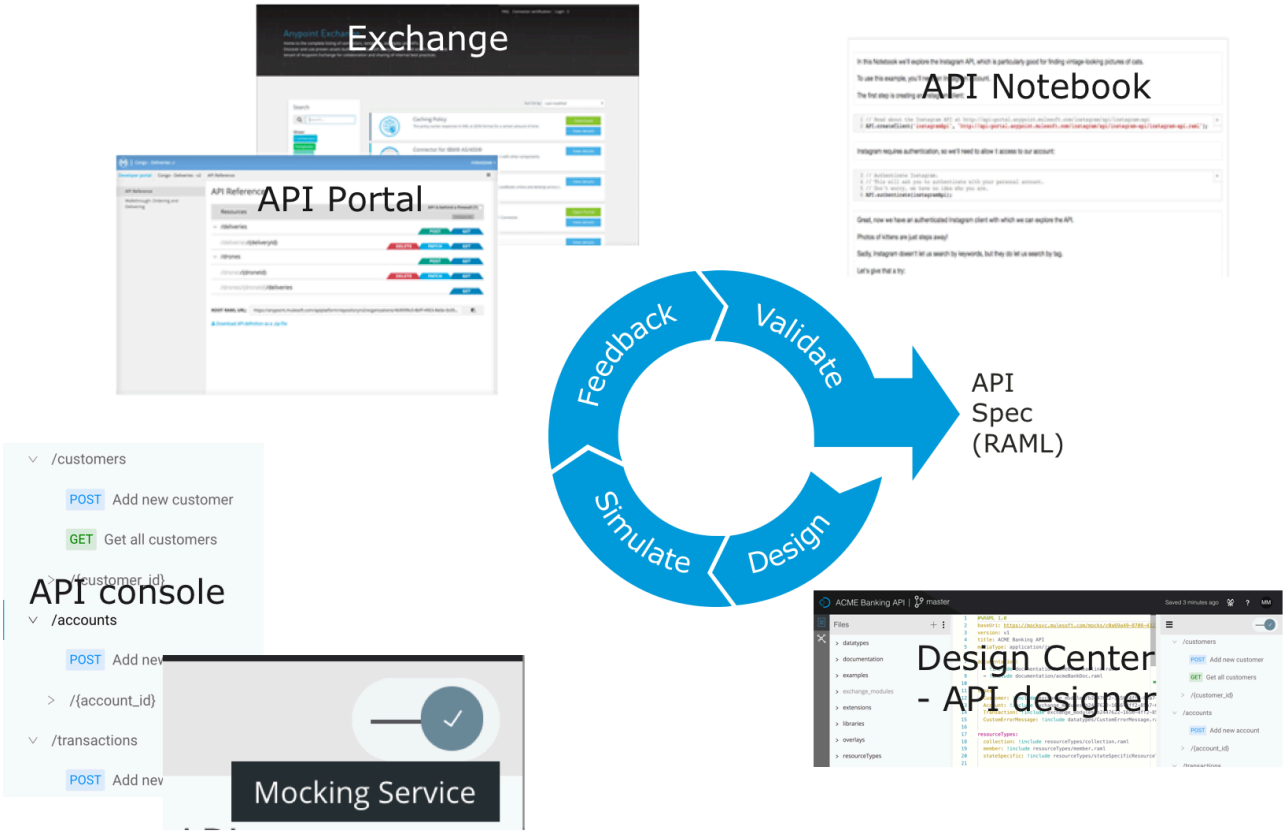
5. Click the event's View Details button.
6. Locate the list of course materials on the right-side of the page.



7. Click the student files link to download the files.
8. Click the student manual link to download the manual.
9. Click the student slides link to download the slides.
10. On your computer, locate the student files ZIP and expand it.
11. 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

```
POST /SendUserMessage HTTP/1.1
Host: api.example.com
Content-type: application/json

{"userId": 501, "message": "Hello!"}
```

REST

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

/customers:
  get:
  post:
  /{customer_id}:
    get:
    patch:
```

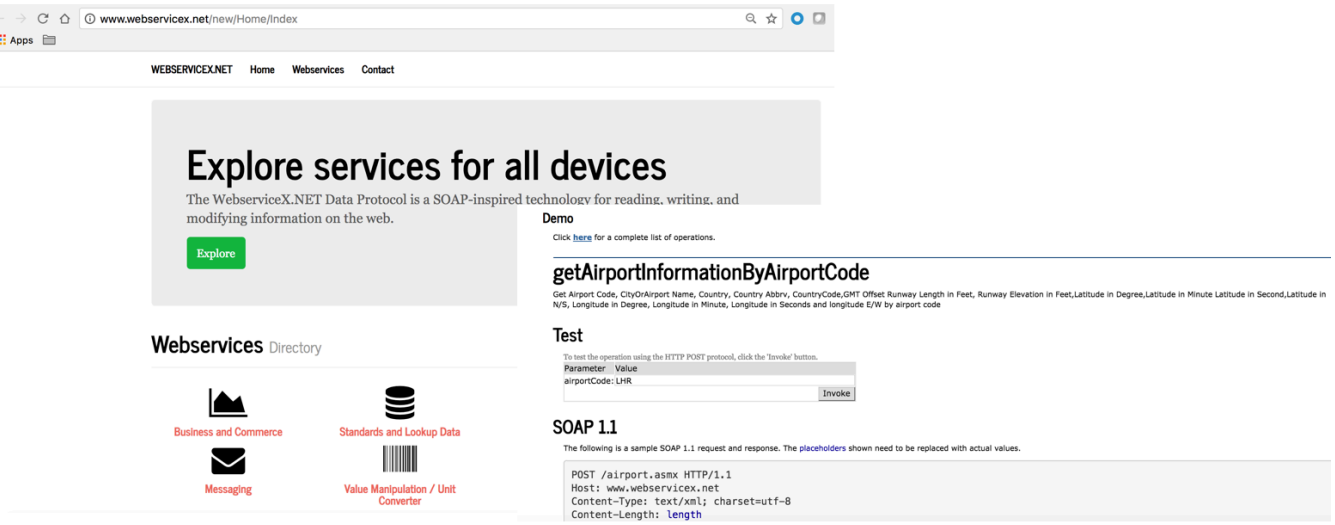
Objectives:

- 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

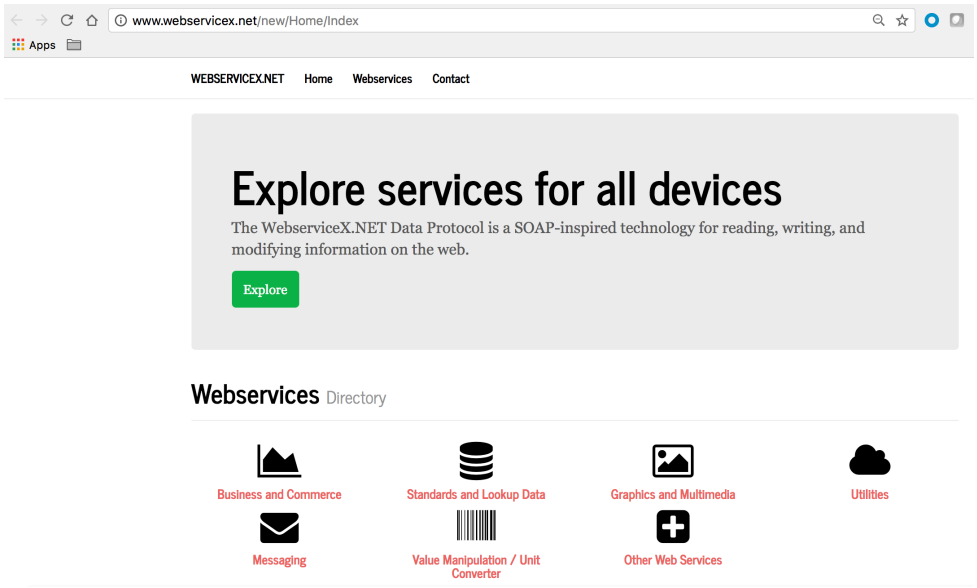
In this walkthrough, you explore and understand a SOAP API. You will:

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

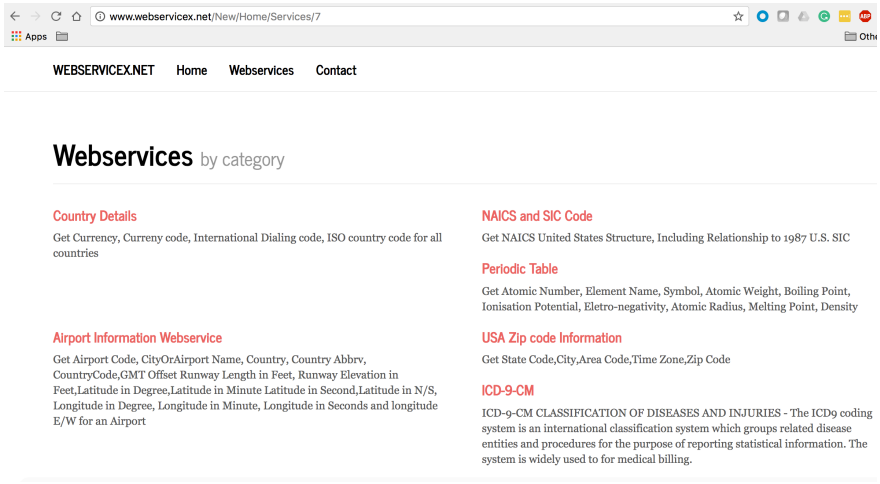


Examine the Airline Information SOAP API

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.

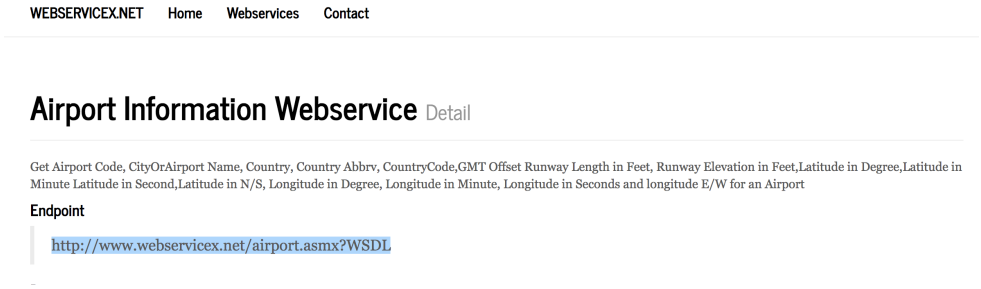


5. In the list of Webservices by category, click the Airport Information Webservice.

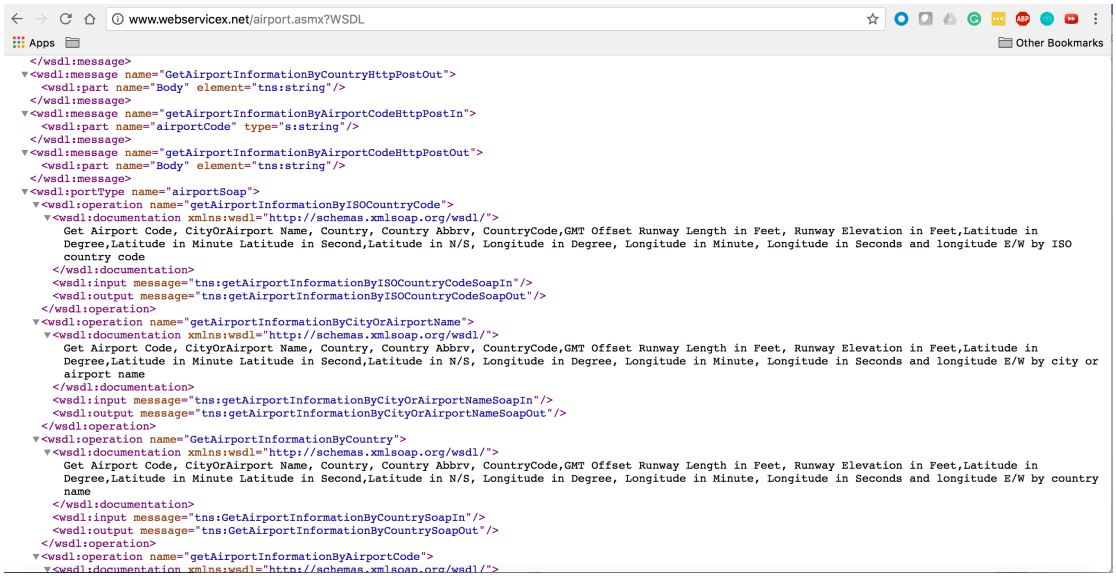


View the Airport Information Webservice endpoint WSDL

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

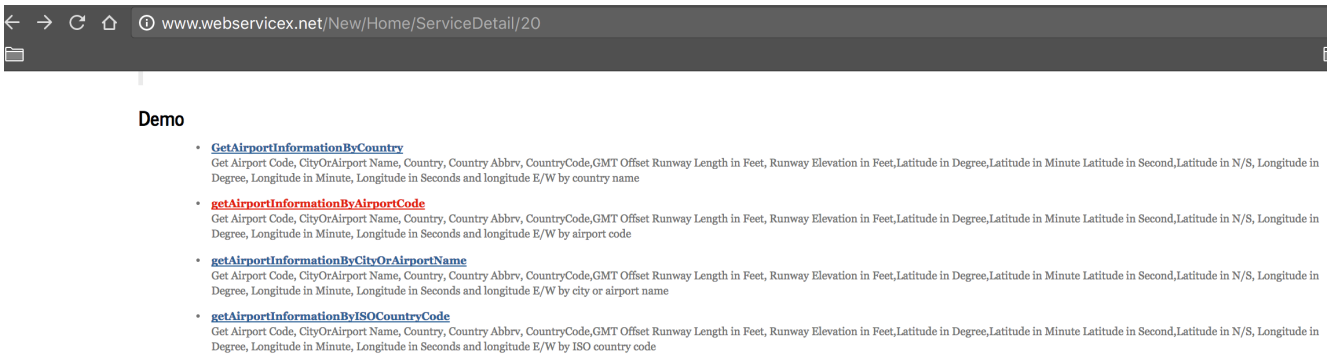


7. In a new tab in the web browser, navigate to the Endpoint URL.
8. Locate the operations and HTTP methods supported by the webservice.

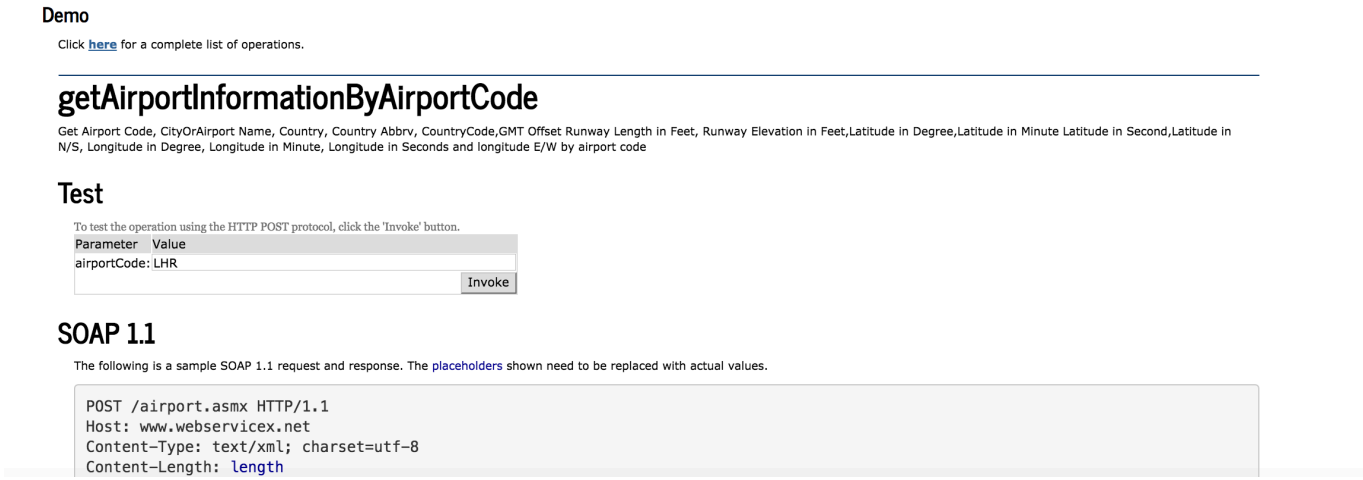


Make a call to retrieve data from an Airport Information Webservice operation

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



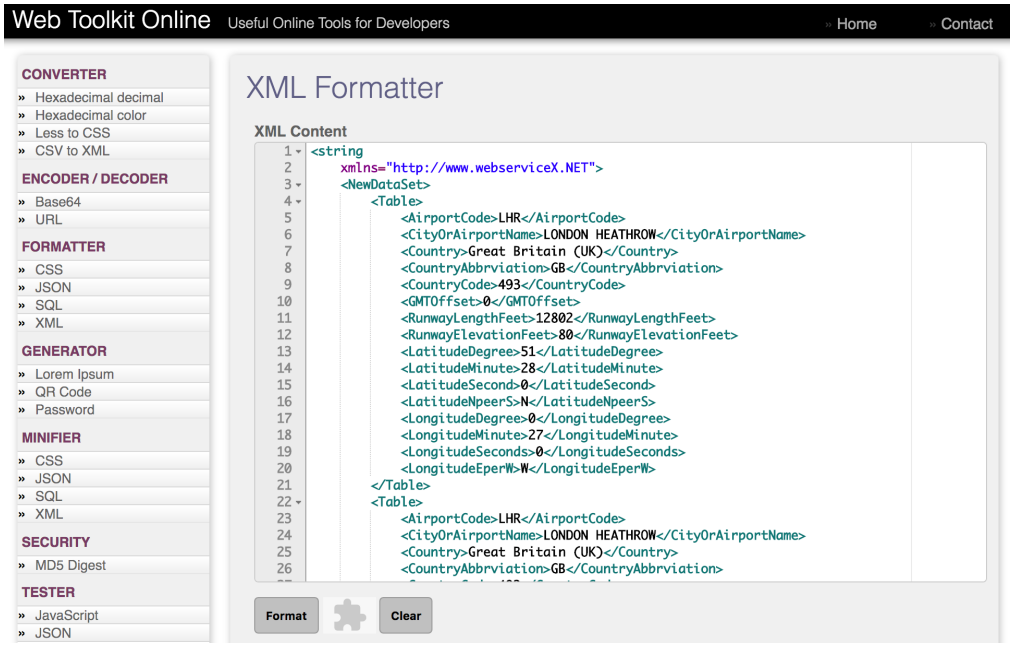
11. In the Test parameter dialog box, type the airportCode parameter value as LHR.
12. Click Invoke.



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

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

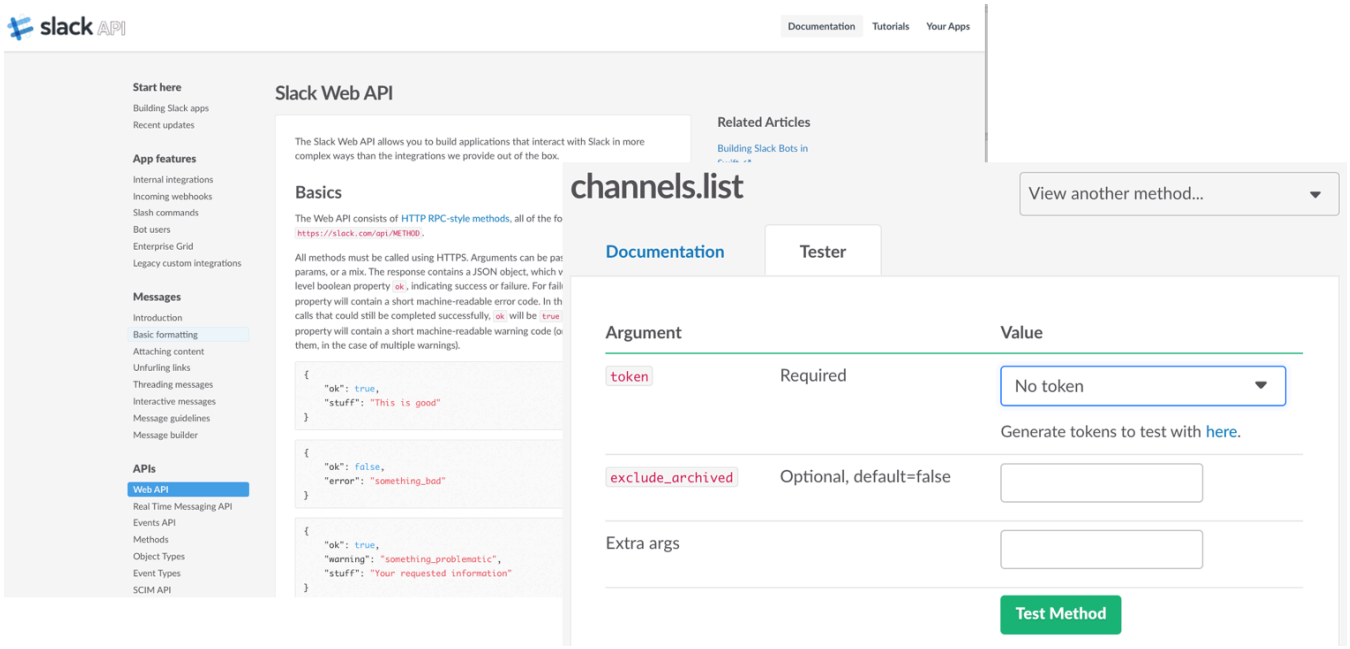
15. Return to the course snippets.txt file and copy the URL for the Web Toolkit Online http://www.webtoolkitonline.com/xml-formatter.html.
16. In a new tab in the web browser, navigate to that URL.
17. Paste the content in the XML Content area.
18. Click Format.



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

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

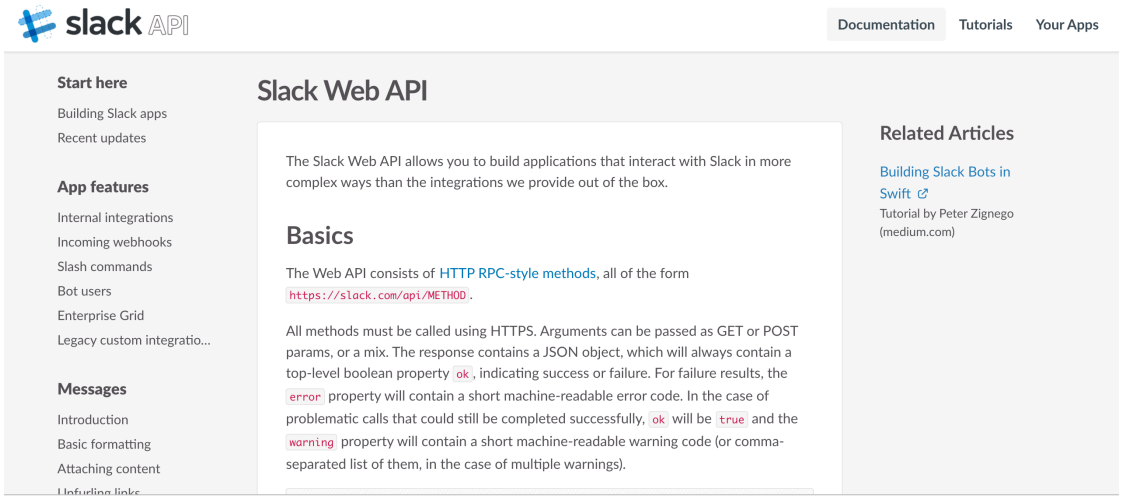
- Examine the Slack RPC API.
- Make a call to an RPC endpoint.



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.

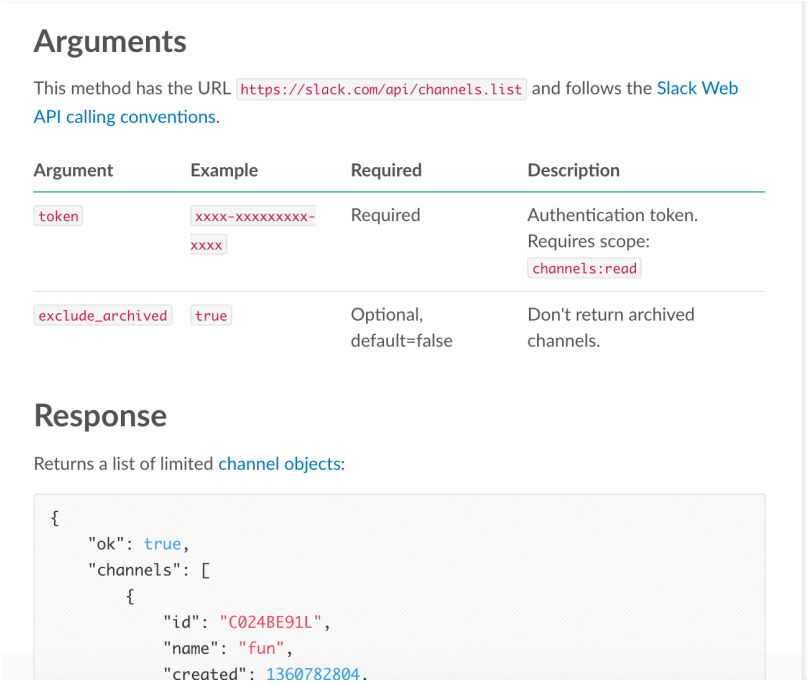


2. Click the HTTP RPC style methods link in the middle of the page.

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
`token`	`xxxx-xxxxxxxxx-xxxx`	Required	Authentication token. Requires scope: `channels:read`
`exclude_archived`	`true`	Optional, default=false	Don't return archived channels.

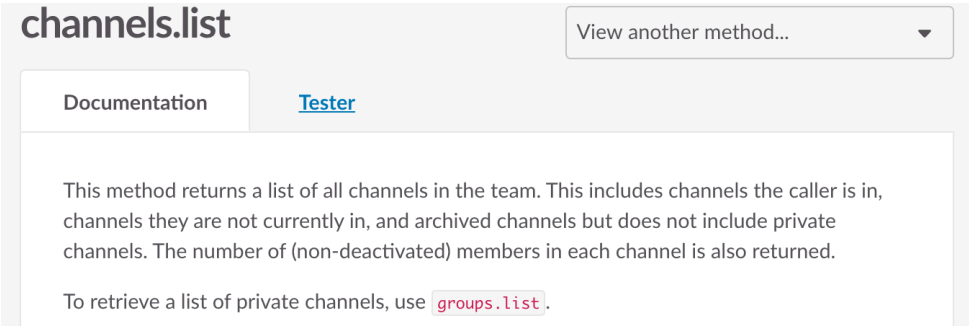
Response

Returns a list of limited channel objects:

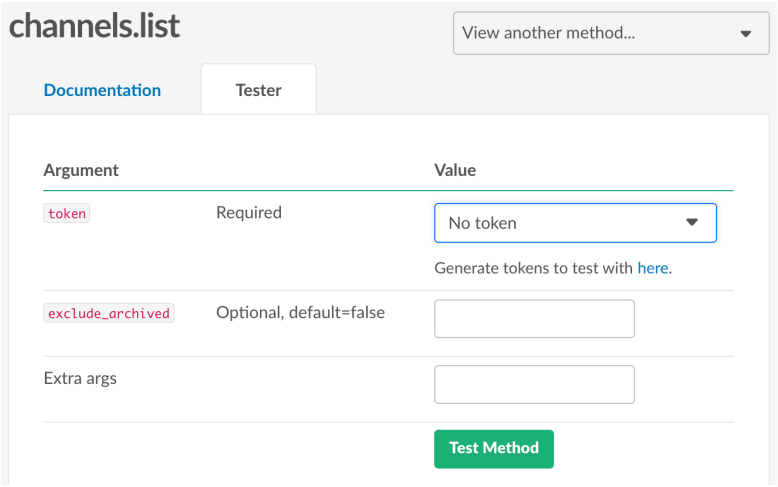
```
{
    "ok": true,
    "channels": [
        {
            "id": "C024BE91L",
            "name": "fun",
            "created": 1360782804,
```

Make a call to an endpoint

6. Scroll back up and click the Tester tab.



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



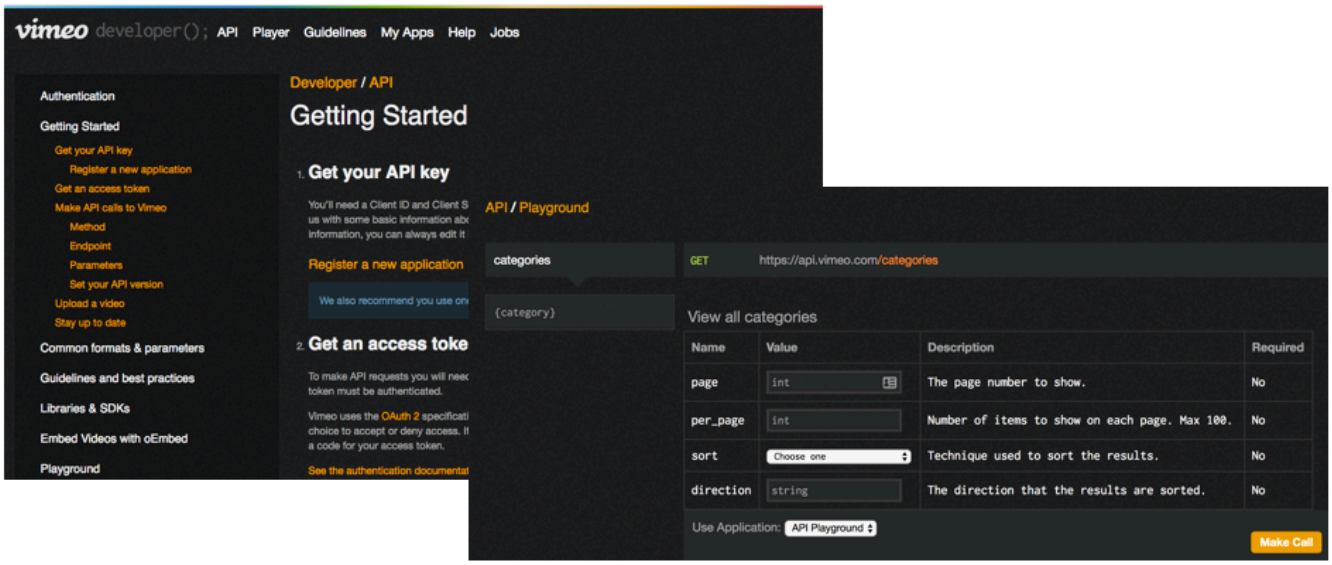
8. Verify that you see the response as not authorized or invalid authorization.

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:

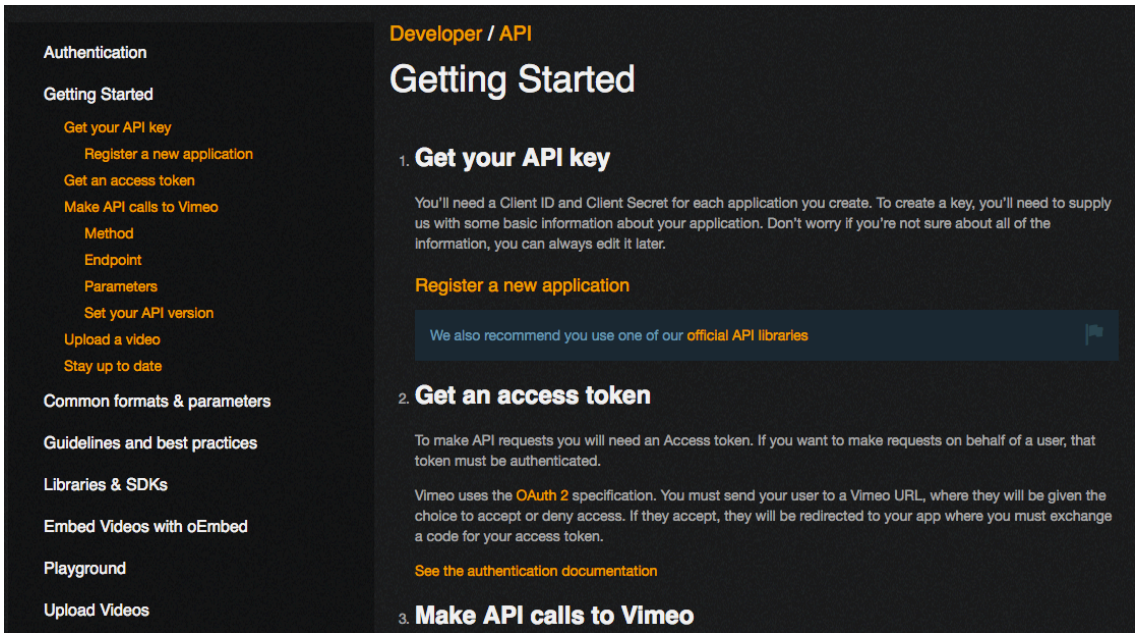
- Examine the Vimeo REST API.
- Make a call to a REST API endpoint to retrieve information.



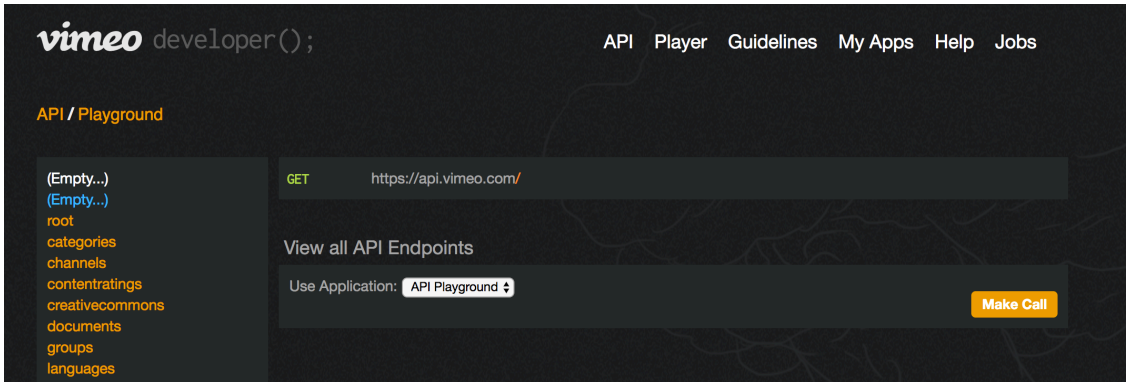
Examine the Vimeo REST API

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

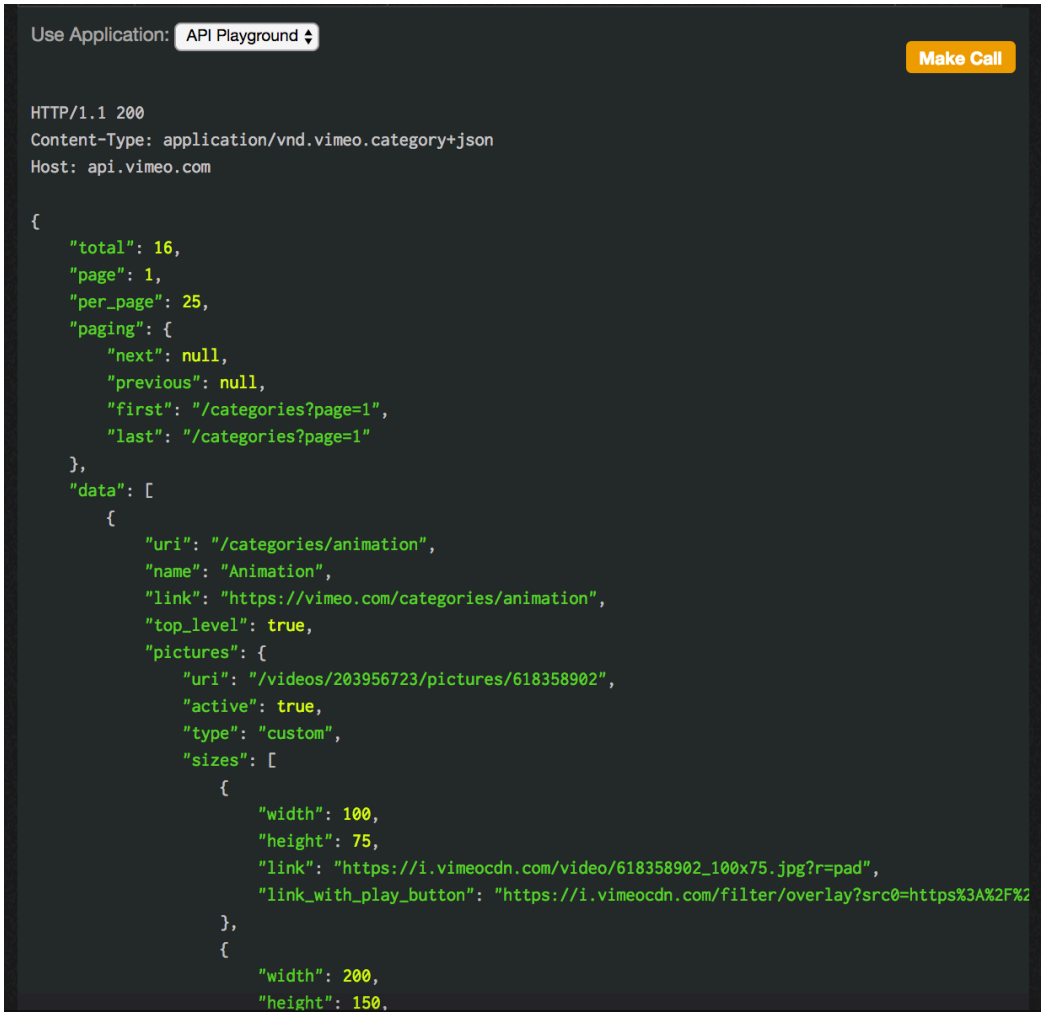
   Note: This URL is also located in the course snippets.txt file.



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



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.



Module 2: Translating Functional Requirements for APIs

```
ACME Banking API User Functionality

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

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

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

Categories:

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

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

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

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

TRANSACTIONS – Bank accounts co
to perform actions on the trans

i.      Create a new transactio
ii.     Get transactions for a
iii.    Get transaction informa
```

```
Translating categories and actions into resources and methods –

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

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

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

Objectives:

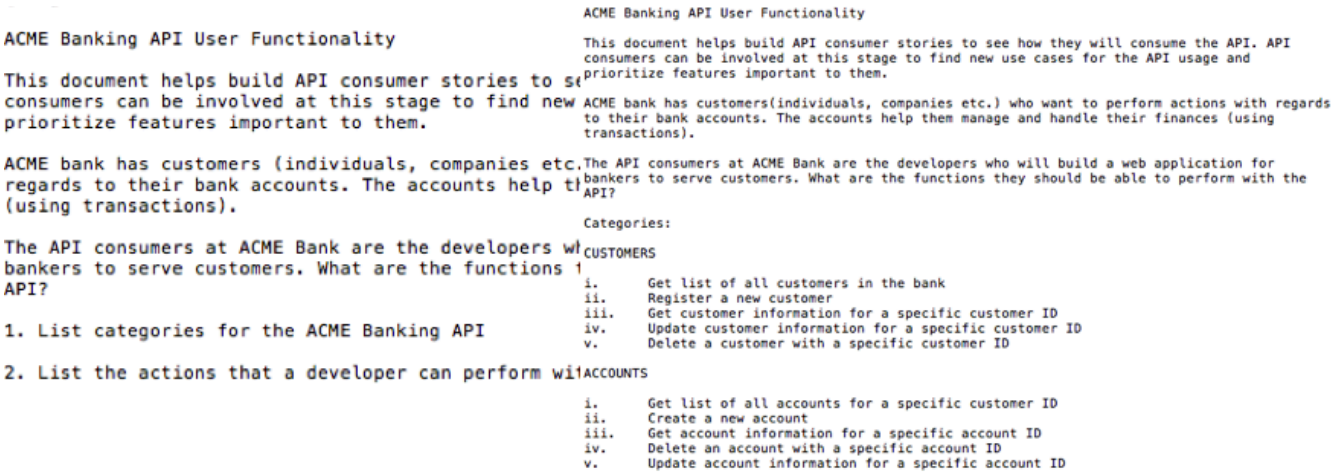
- Identify the different categories and actions for REST APIs.
- Convert categories to resources.
- Select HTTP methods to support the actions on categories.

MuleSoft

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 s
consumers can be involved at this stage to find new
prioritize features important to them.

ACME bank has customers (individuals, companies etc.
regards to their bank accounts. The accounts help t
(using transactions).

The API consumers at ACME Bank are the developers w
bankers to serve customers. What are the functions
API?

1. List categories for the ACME Banking API

2. List the actions that a developer can perform wi
```

```
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

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

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 _______________
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 _______________
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 _______________
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 _______________
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 _______________
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 _______________
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.

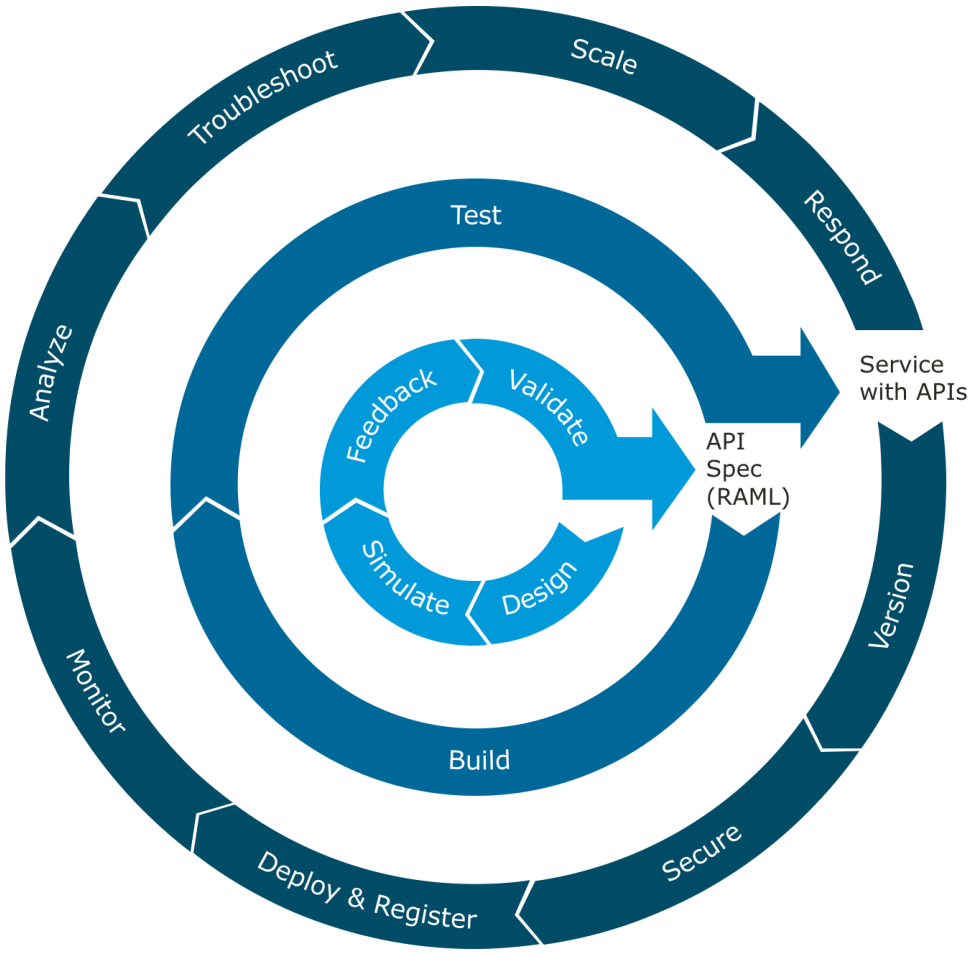
```
CUSTOMERS: Resource /customers
i.  Get list of all customers in the bank                    Resource /customers                          Method GET
ii. Register a new customer                                  Resource /customers                          Method POST
iii.Get customer information for 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 POS
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 DEL
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 POS
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



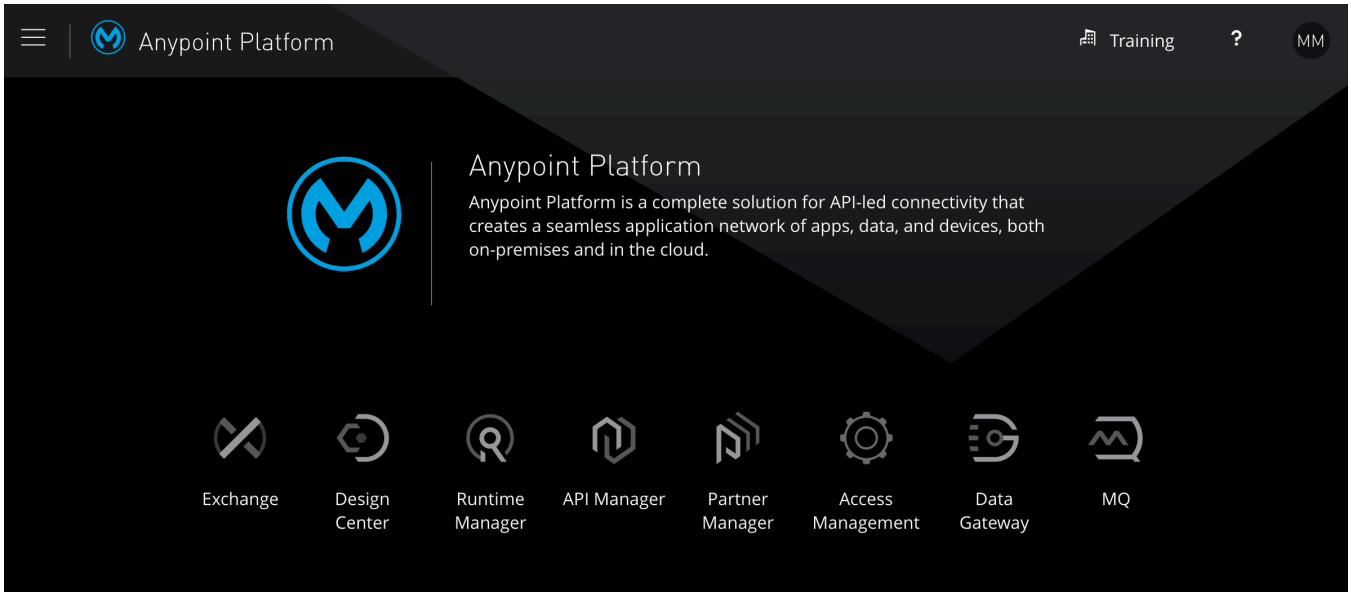
Objectives:

- 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:

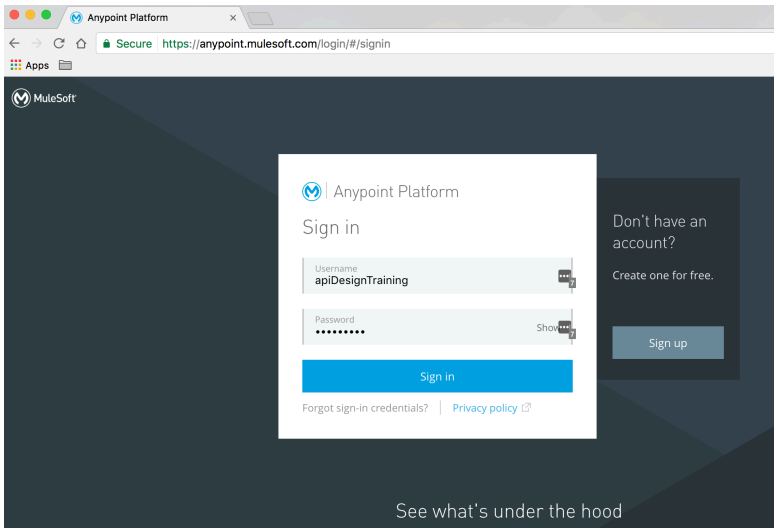
- Log in to Anypoint Platform.
- Explore the API entitlements in Anypoint Platform.



Log in to Anypoint Platform

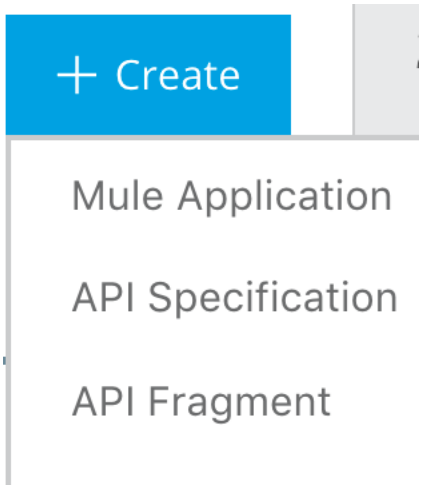
1. In a web browser, navigate to https://anypoint.mulesoft.com.
2. Log in to Anypoint Platform.

   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

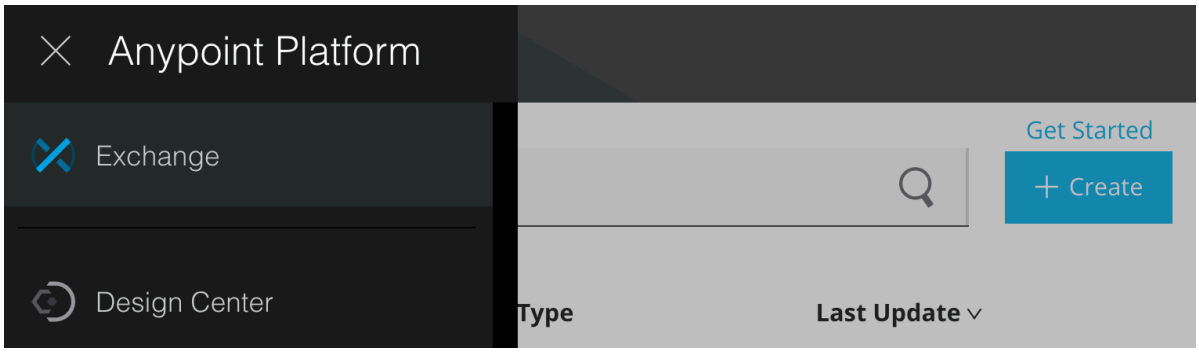
3. In the Anypoint Platform homepage, click the top left menu icon.
4. Click Design Center.
5. In the Design Center homepage, click Create in the top right corner.
6. Verify that you see the options to create API Specification, API Fragment and Mule application.



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

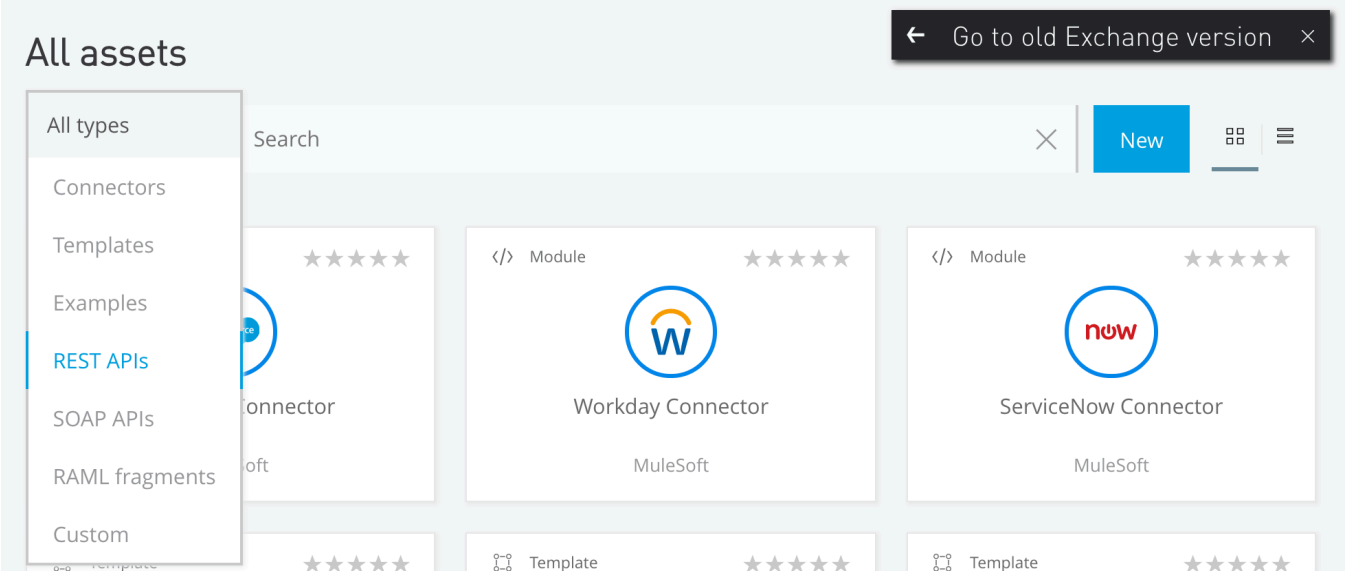
Explore Anypoint Exchange

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

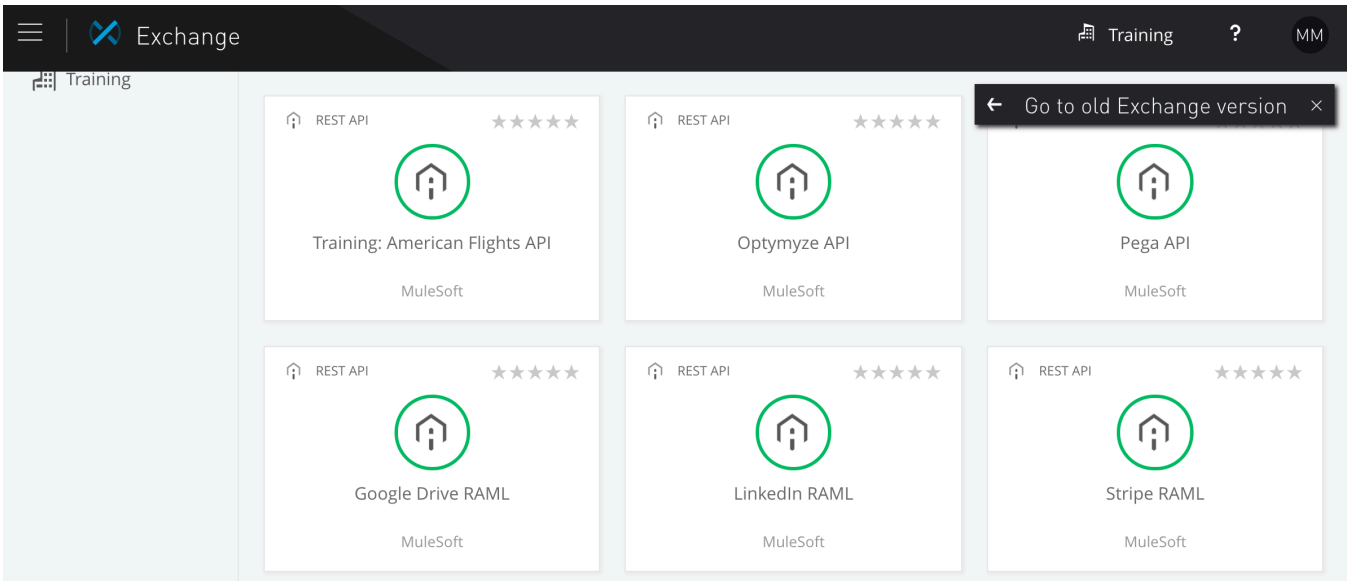


8. In the Exchange webpage, click the down arrow next to All types to the left of the search bar.

9. Click REST APIs.



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

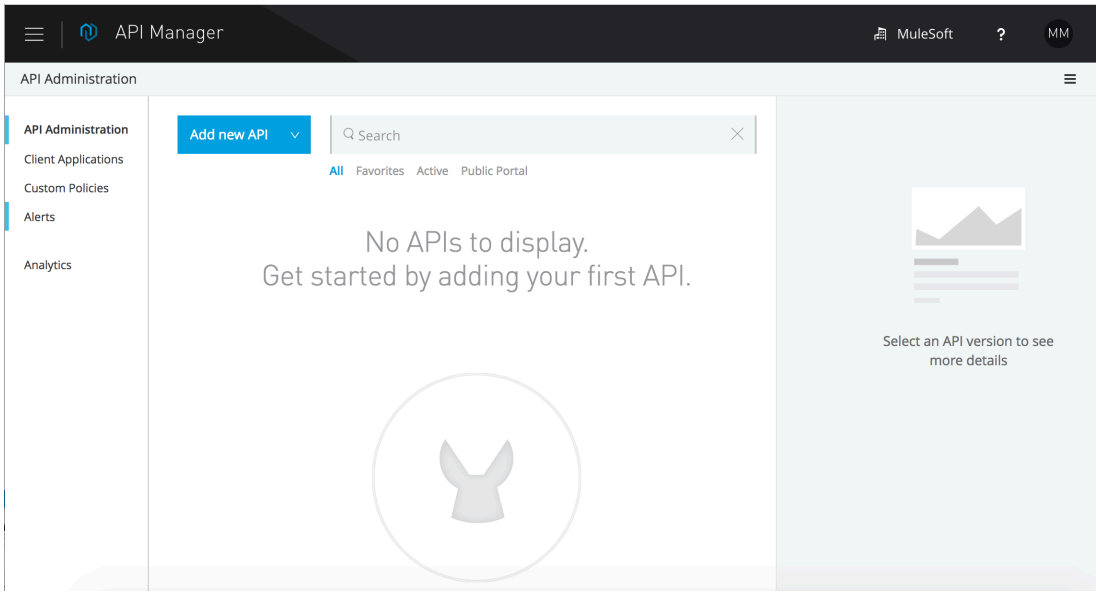


Explore API Manager in Anypoint Platform

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

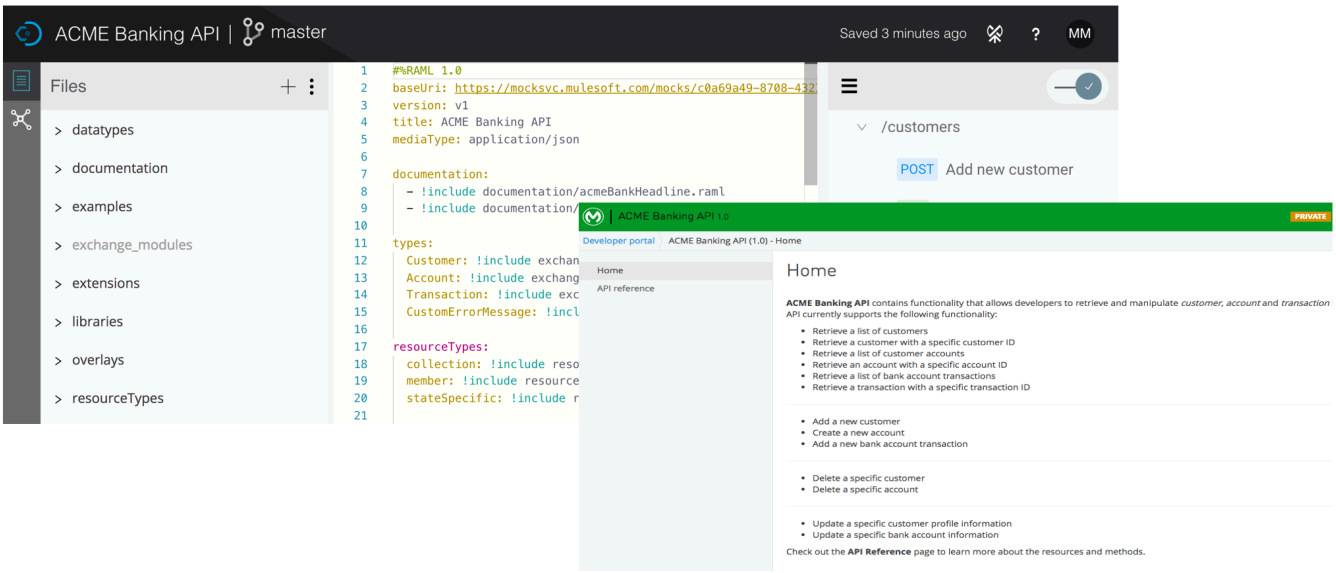
In the web page for API Manager, verify that you can see the following functionality:

- 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.

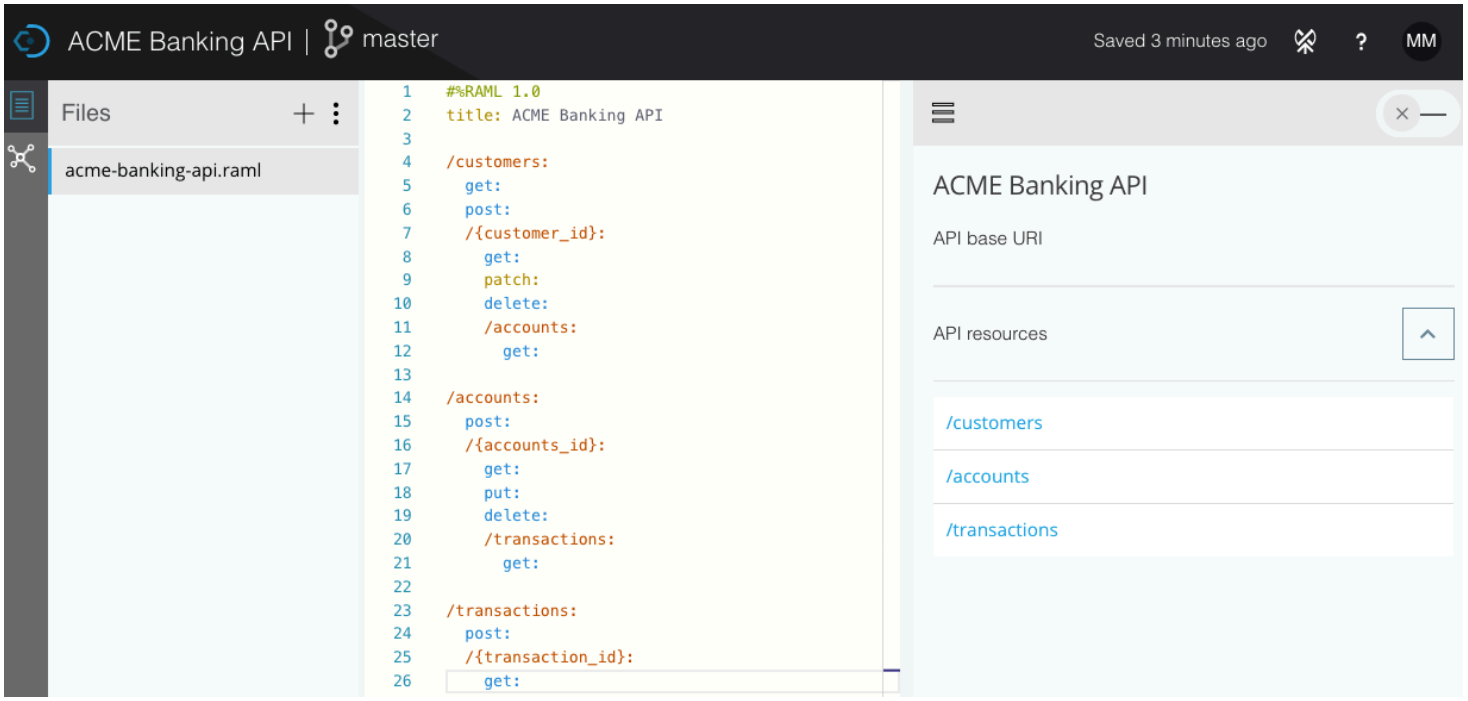
PART 2: Defining APIs with the RESTful API Modeling Language (RAML)



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



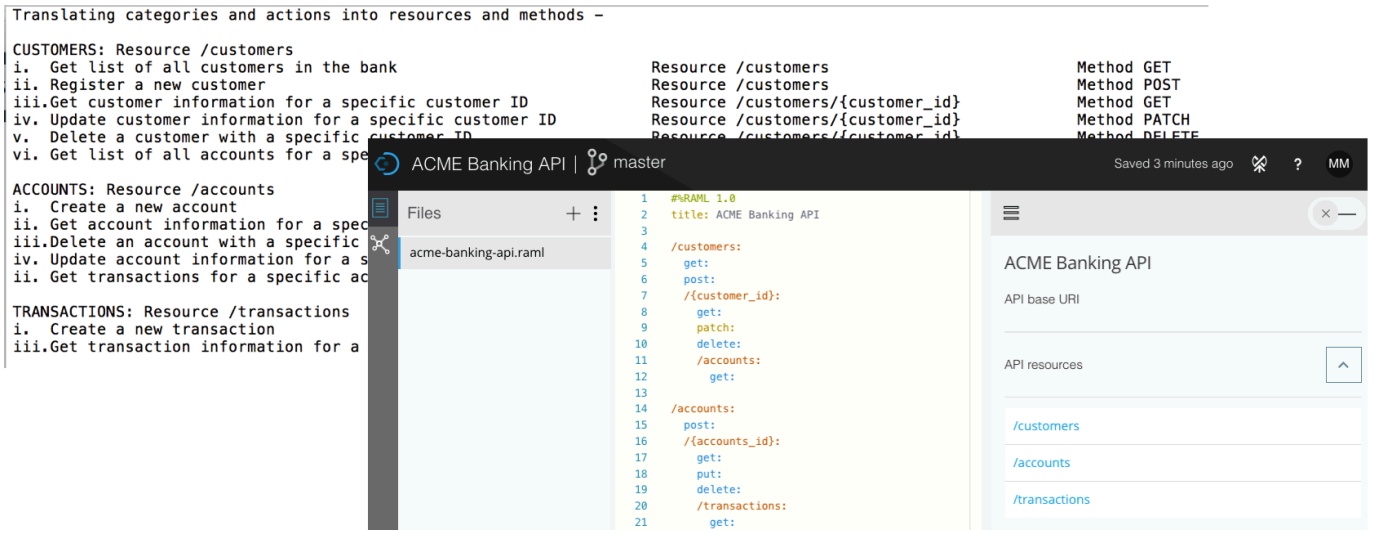
Objectives:

- 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

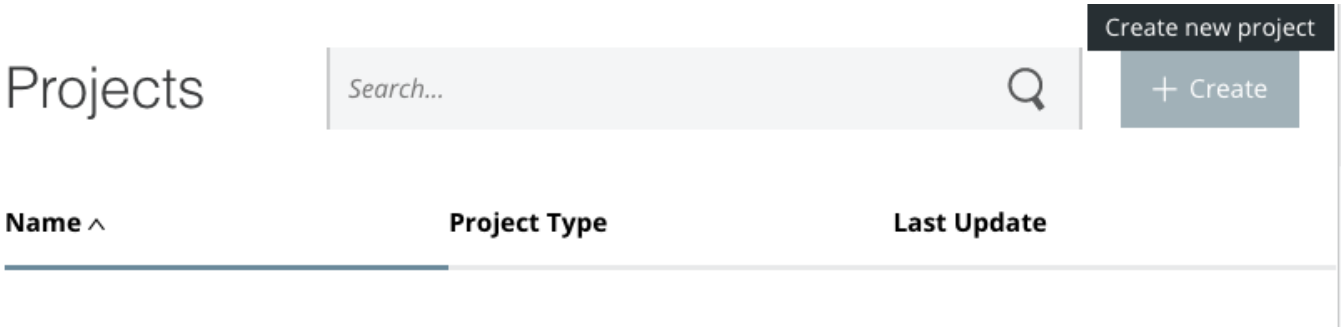
In this walkthrough, you create an API definition for the ACME Banking use case. You will:

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

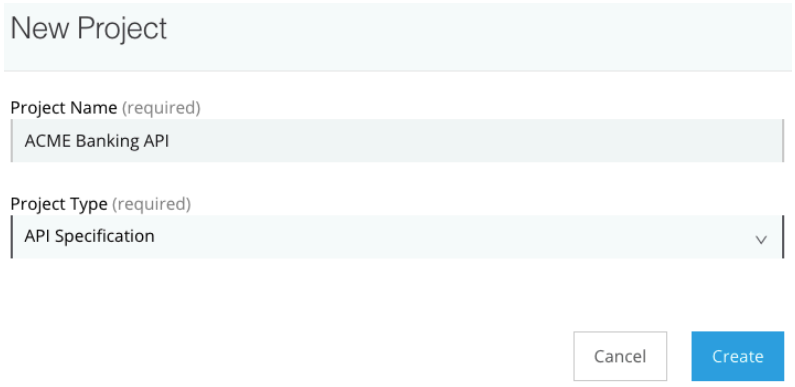


Create a new API in Anypoint Platform Design Center

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



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.

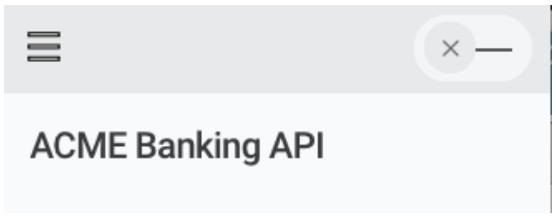
Define the resources

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.

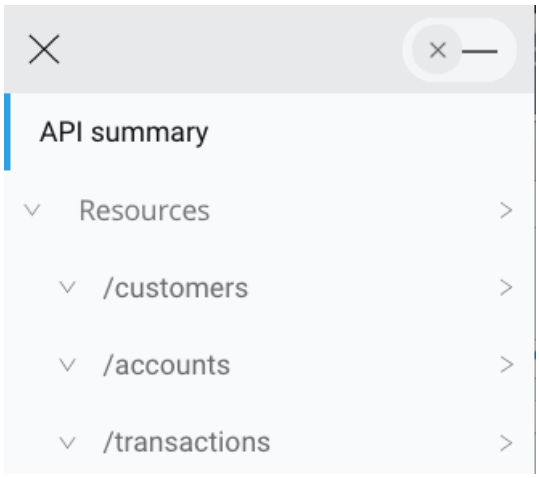
```
/customers:
```

10. Press enter twice and add the /accounts resource.
11. Similarly, add the /transactions resource.

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.

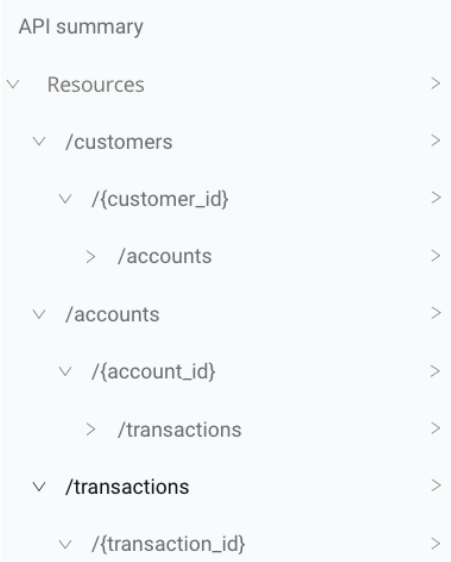
```
7
8   /accounts:
9     /{account_id}:
10      /transactions:
11
```

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
```

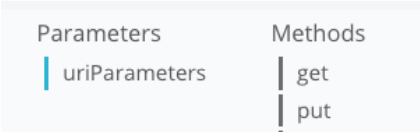
26. Review the resources that have get methods.

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.

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

/customers:

  /{customer_id}:
    /accounts:

/accounts:
  /{account_id}:
    /transactions:

/transactions:
  /{transaction_id}:
```



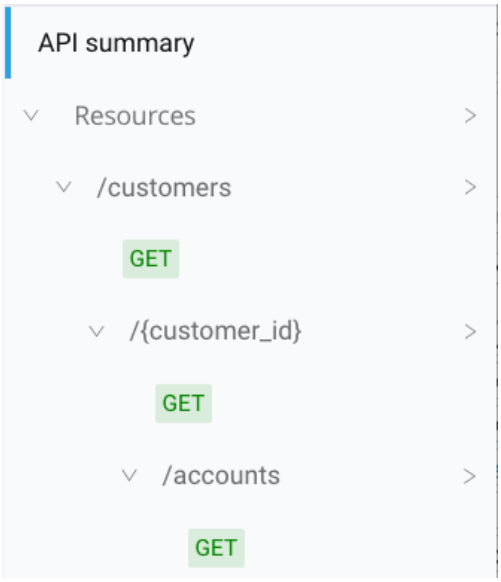
Note: Alternatively, you could press the control and spacebar keys, and click get from the drop-down box.

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.

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

/customers:
  get:
  /{customer_id}:
    get:
    /accounts:
      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 /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
```

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
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
```

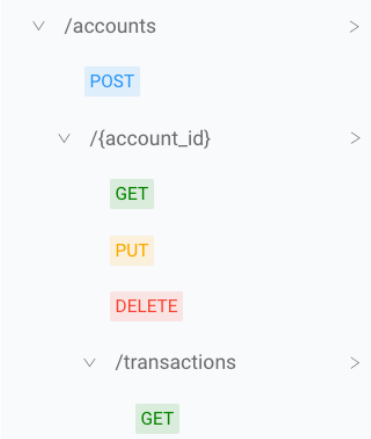
45. Review the methods for the /accounts resource and nested resources.

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

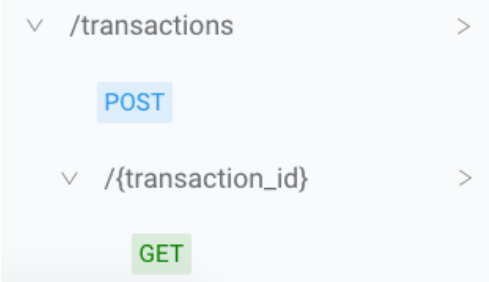
51. Review the methods for the /transactions resource and nested resources.

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

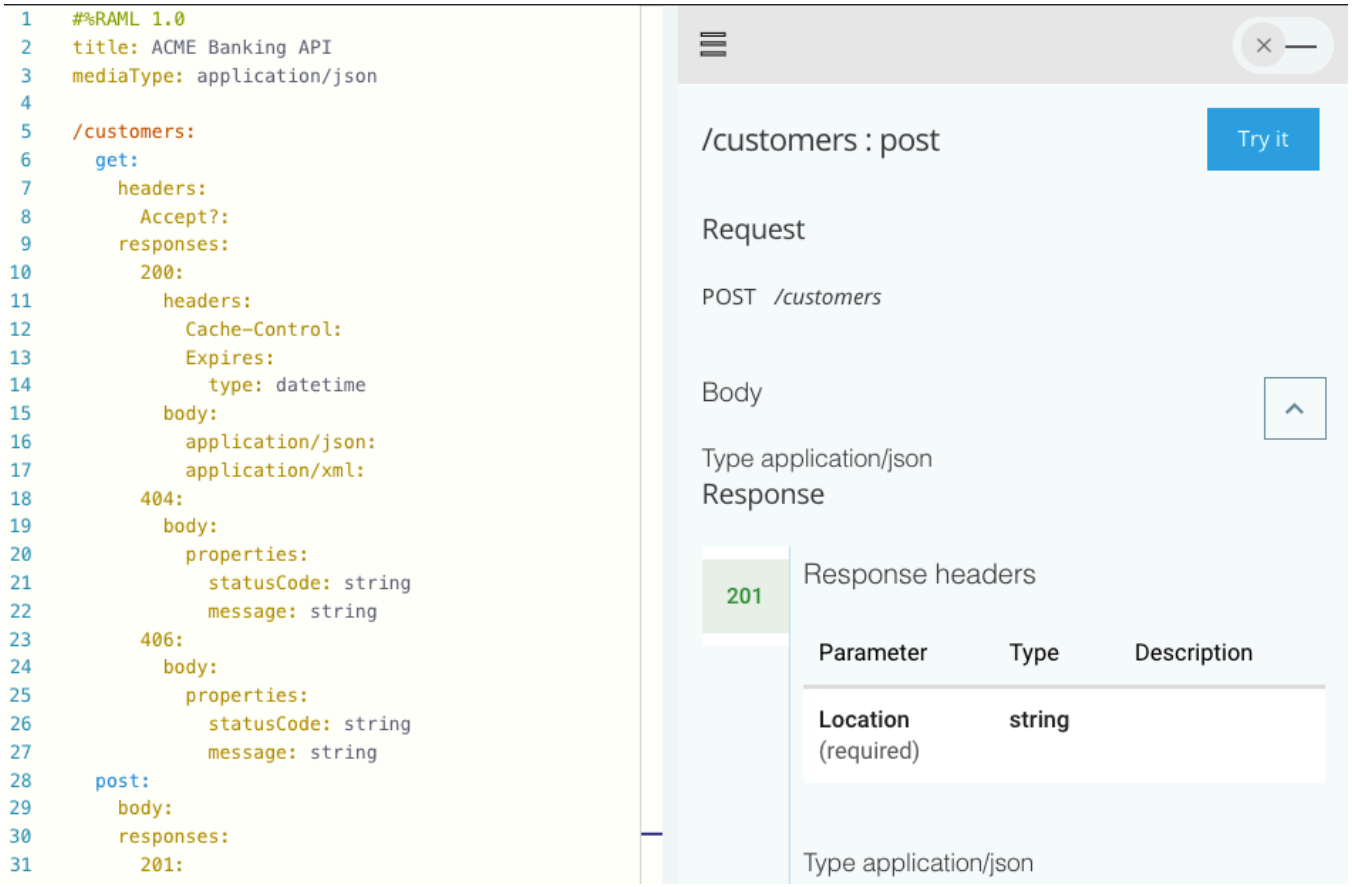
52. Return to API designer.
53. Add a post method to the /transactions resource.

```
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



```raml
#%RAML 1.0
title: ACME Banking API
mediaType: application/json

/customers:
  get:
    headers:
      Accept?:
    responses:
      200:
        headers:
          Cache-Control:
          Expires:
            type: datetime
        body:
          application/json:
          application/xml:
      404:
        body:
          properties:
            statusCode: string
            message: string
      406:
        body:
          properties:
            statusCode: string
            message: string
  post:
    body:
    responses:
      201:
```

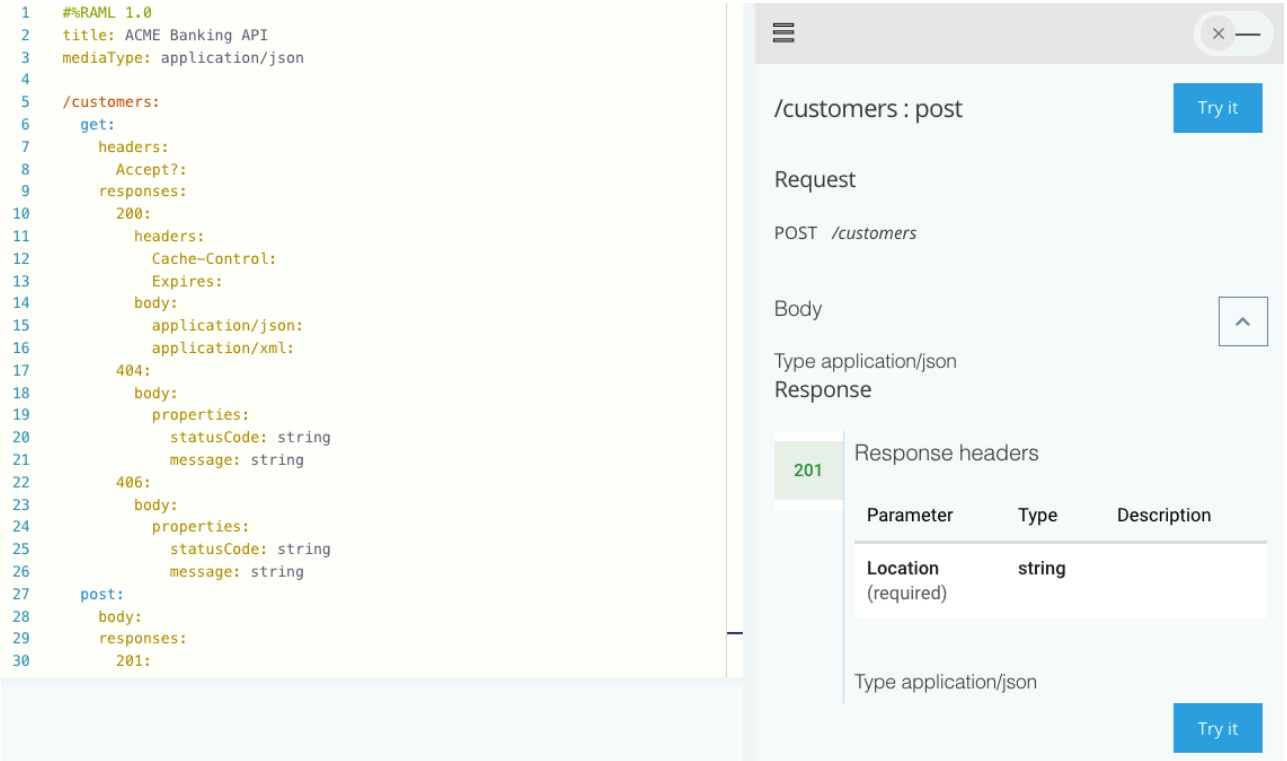

Objectives:

- 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:

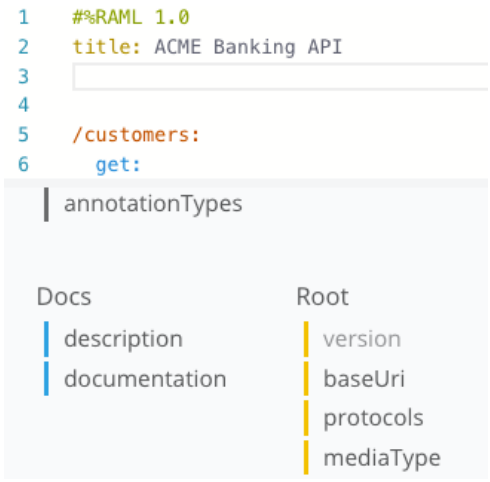
- Define media type for the API resource methods.
- Add a HTTP 200 response body to all GET and DELETE methods indicating success of the HTTP request.
- Add a HTTP 201 response body to all POST methods.
- Add a response body with both HTTP 200 and 201 status codes to the PUT method.
- Add a HTTP 204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



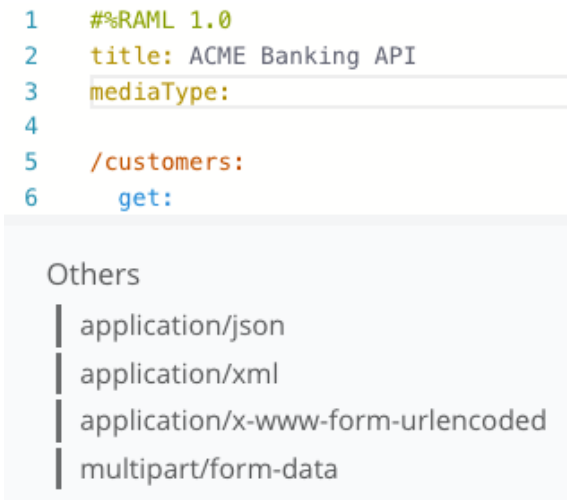
Add a mediaType node to indicate the request and response body type

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.



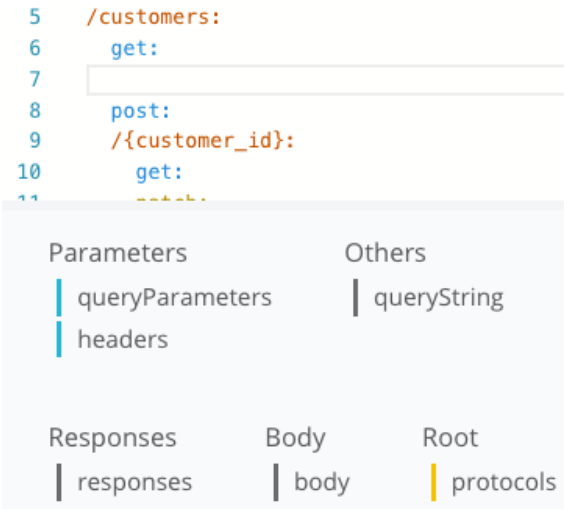
4. Add a space after the colon in the mediaType node.
5. From the shelf, click application/json.



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

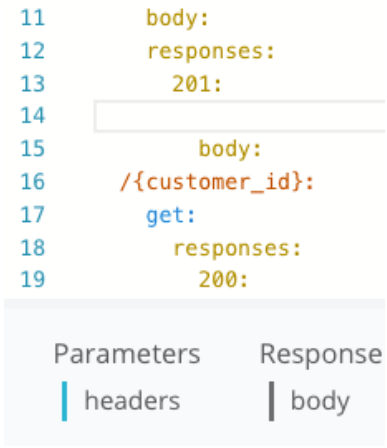
14. In the API designer, add a new line below the /customers post method.
15. Press the tab key to indent.

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.



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.

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.

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

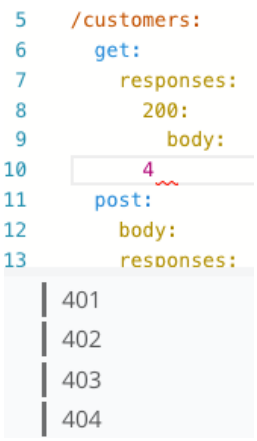
Walkthrough 5-2: Add responses bodies to return custom error information for client-side errors

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.

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. In the shelf, click 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.

10. Press enter and type:

    ```
    message: string
    ```

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:

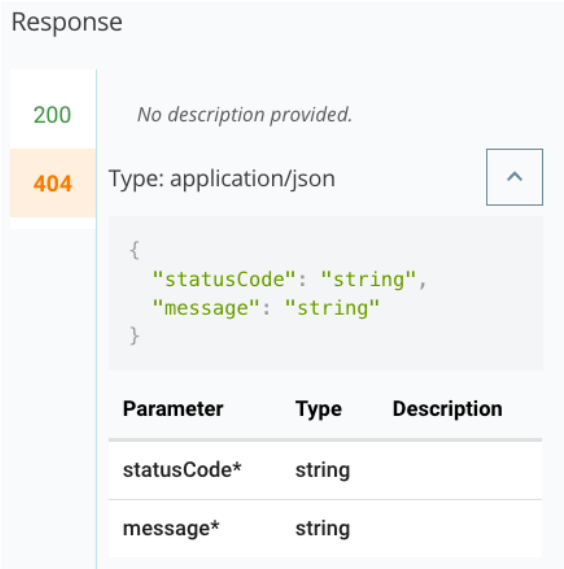
    - /customers/{customer_id}
    - /accounts/{account_id}

```
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.



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

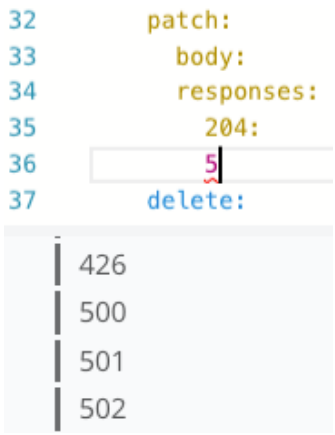
Walkthrough 5-3: Add response bodies to return custom error information for server-side errors

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.

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.



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.

8. Press enter and type:

   ```
   message: string
   ```

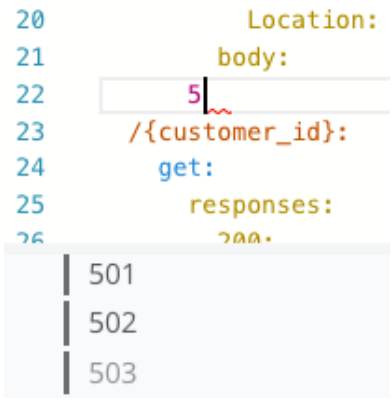
Add the HTTP 501 response body to the PUT method

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
```

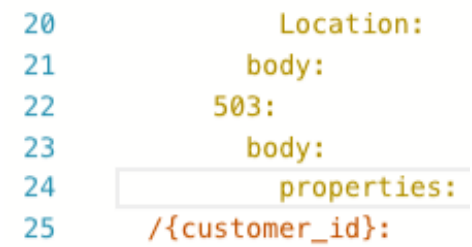
Add the 503 Service Unavailable HTTP status code response to a POST method

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.



14. In the shelf, click body.
15. In the new line, type:

```
properties:
```



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
```

Add the HTTP 503 response body to all POST methods

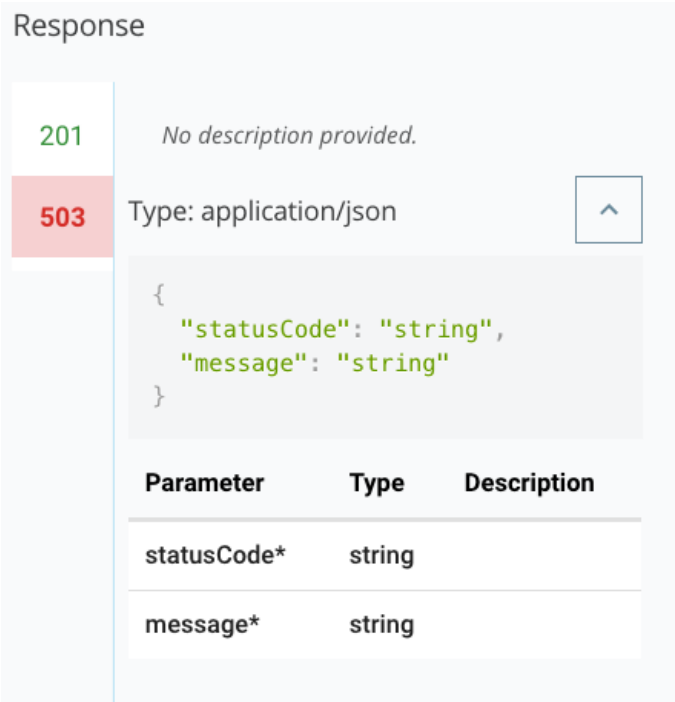
19. Copy the HTTP 503 response body lines.
20. Add the HTTP 503 response body to the post methods of the following resources:

    - /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.



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.

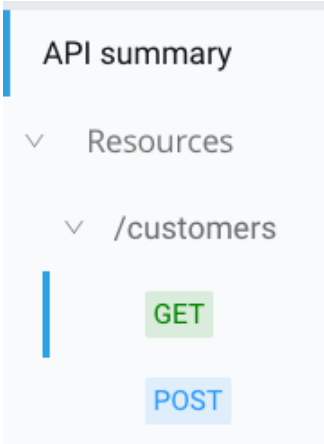
Add XML response media type to the HTTP 200 responses body for /customers GET method

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.



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.

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.

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.

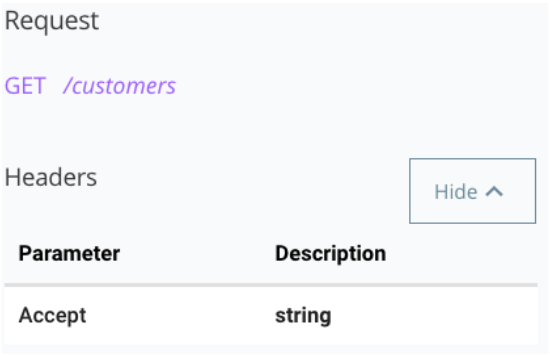
```
5   /customers:
6     get:
7       headers:
8         Accept?:
9       responses:
10        200:
```

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.



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:

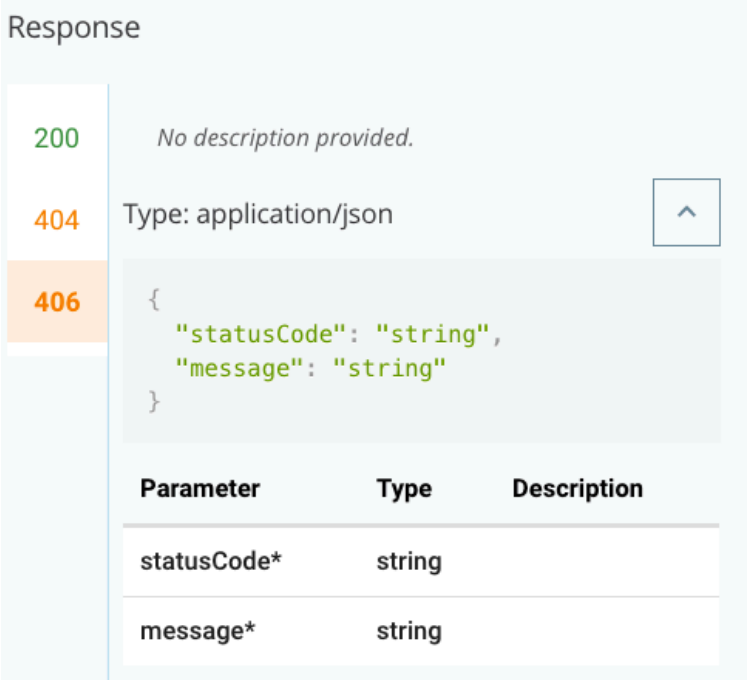
```
properties:
```

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.

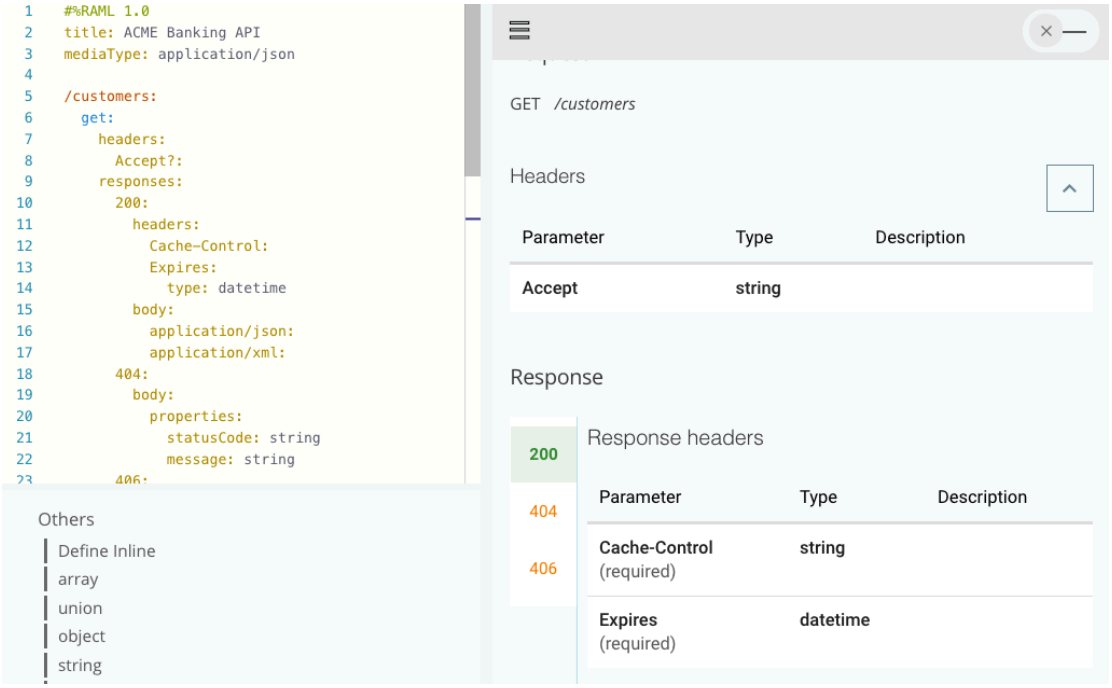


31. Go back to the list of resources by clicking on the top-left menu icon.

Walkthrough 5-5: Add caching information to HTTP responses

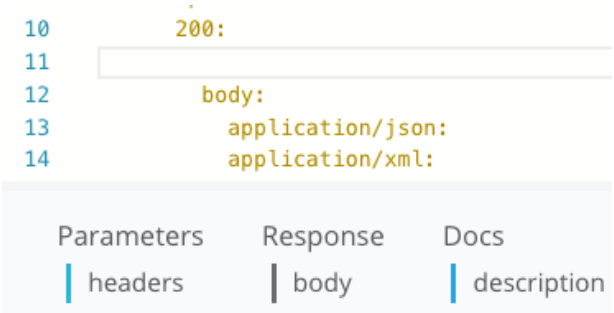
In this walkthrough, you indicate that a resource is cacheable by using cache-control headers. You will:

- 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.



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.



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
- displayName
- example

Others
- facets
- type

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

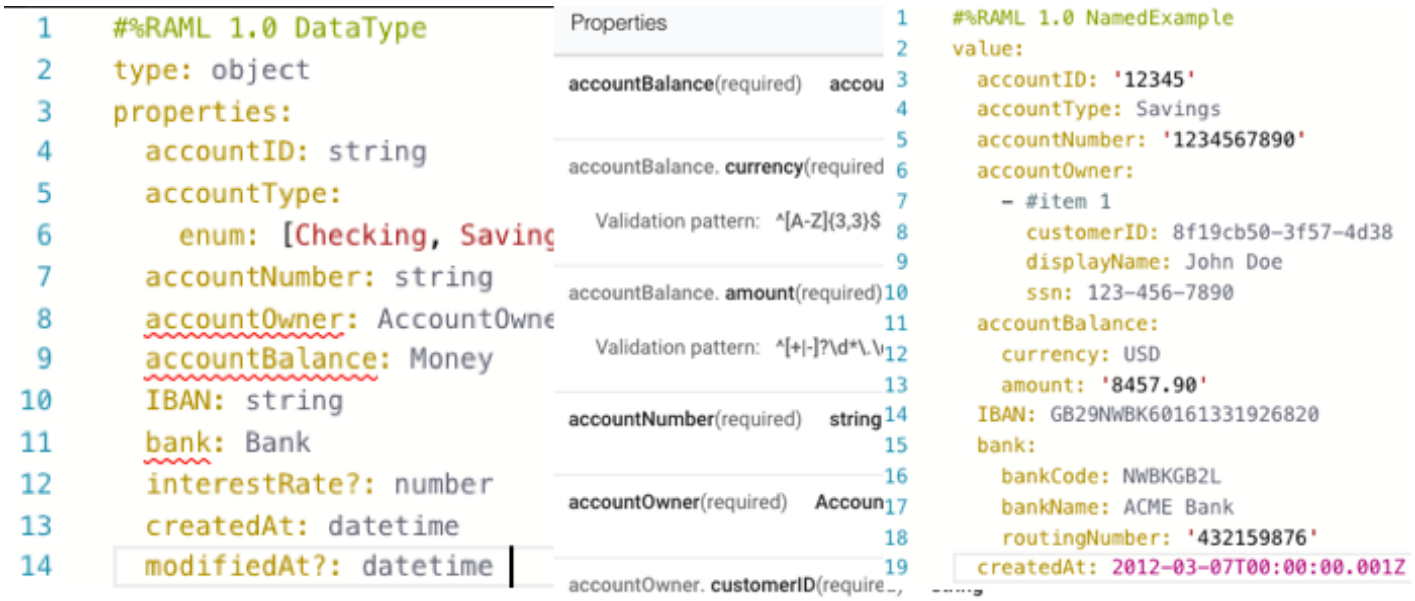
```
11            headers:
12              Cache-Control:
13              Expires:
14                type: datetime
15            body:
```

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



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:

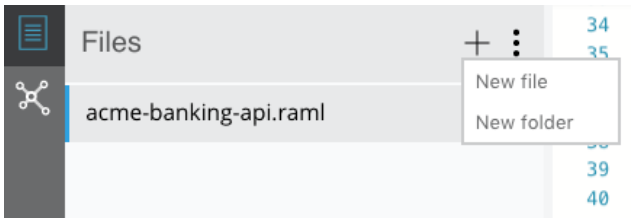
- Create datatype fragment files for the identified datatypes.
- Define the datatypes with required and optional attributes.
- Include the datatype fragments in the main RAML API definition.

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

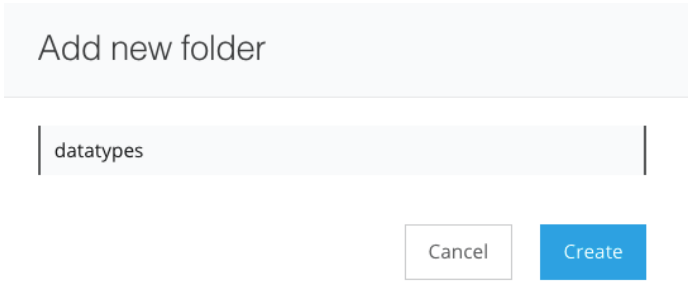
```
#%RAML 1.0
title: ACME Banking API
mediaType: application/json

types:
  Customer: !include datatypes/Customer.raml
  Address: !include datatypes/Address.raml
  Account: !include datatypes/Account.raml
  AccountOwner: !include datatypes/AccountOwner.raml
  Bank: !include datatypes/Bank.raml
  Money: !include datatypes/Money.raml
  Transaction: !include datatypes/Transaction.raml
  CustomErrorMessage: !include datatypes/CustomErrorMessage.raml
```

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.



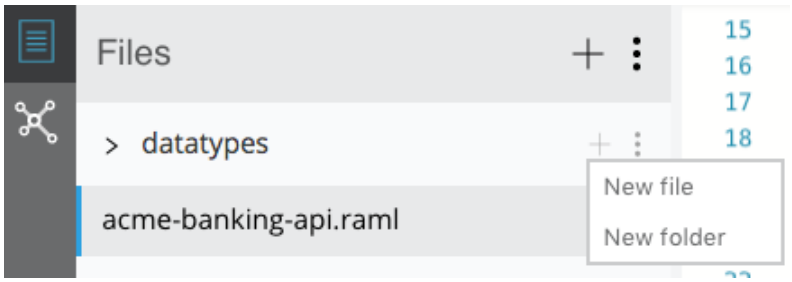
4. In the Add new folder dialog box, set the name to datatypes.



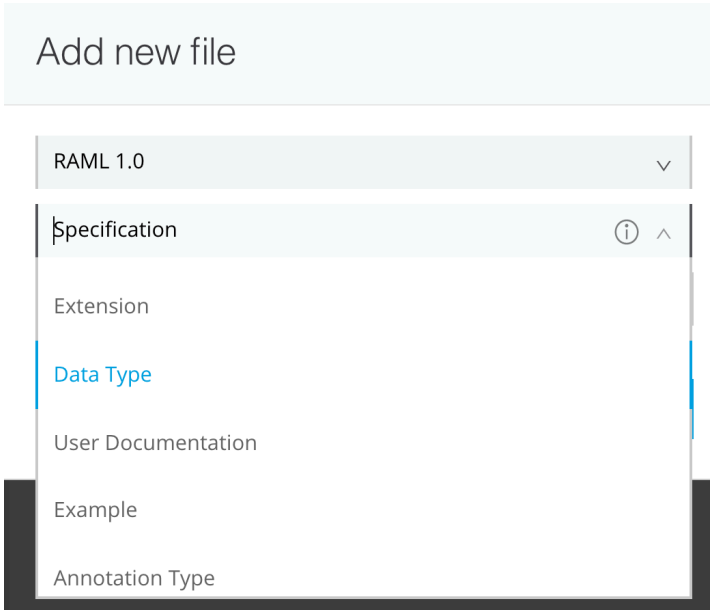
5. Click Create.
6. In the file browser, click acme-banking-api.raml.
7. Go to the /customers resource get method HTTP 404 response body and copy the lines that define the properties of the custom error message.

```
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.

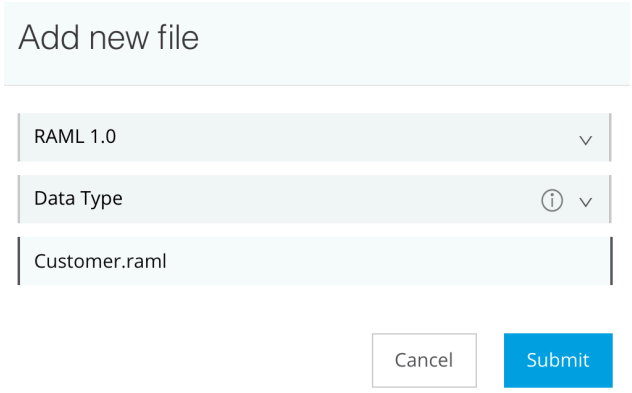
```
#%RAML 1.0 DataType
type: object
properties:
            statusCode: string
            message: string
```

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

```
#%RAML 1.0 DataType
type: object
properties:
  statusCode: string
  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.



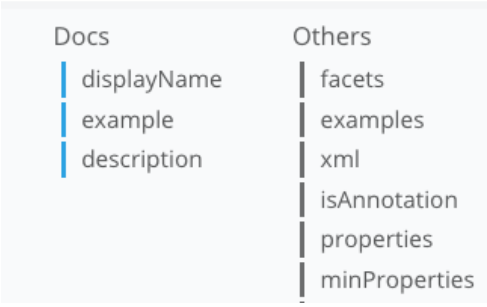
21. Click Create.
22. In the RAML editor, go to the end of the first line and press enter to add a new line.

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
```



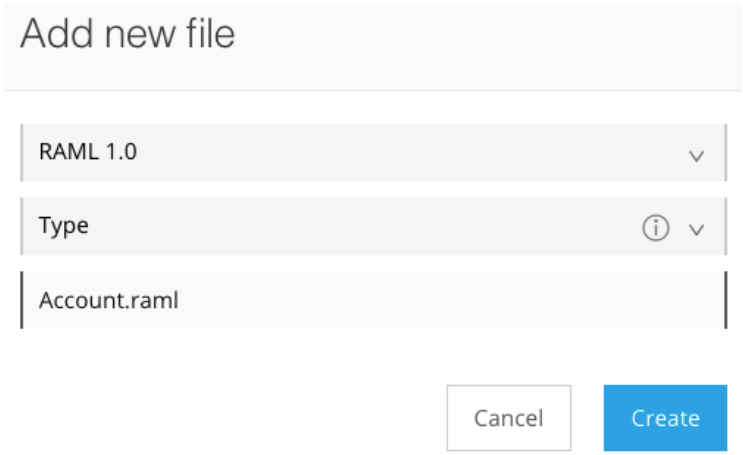
25. Return to the WT 6-1 Datatypes and attributes file in your computer.
26. Return to API designer.
27. Add the attributes for the Customer one below the other; for each, specify the attribute name, followed by a colon, and then the attribute type.
28. Add a ? after the prefix and suffix attribute names 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.

```
#%RAML 1.0 DataType
type: object
properties:
  accountID: string
  accountType:
    enum: [Checking, Savings, Overdraft Savings, Credit Card]
  accountNumber: string
  accountOwner: AccountOwner[]
  accountBalance: Money
  IBAN: string
  bank: Bank
  interestRate?: number
  createdAt: datetime
  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.

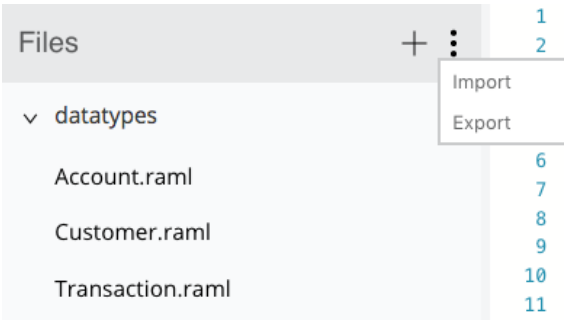
32. Use the enum datatype to specify the transactionType has possible values of atm, deposit, cashWithdrawal, sepa, and online.

```
#%RAML 1.0 DataType
type: object
properties:
  transactionID: string
  fromAccount: Account
  toAccount: Account
  transactionType:
    enum: [atm, deposit, cashWithdrawal, sepa, online]
  transactionName?: string
  transactionAmount: Money
  newAccountBalance: Money
  postedAt: datetime
  completedAt?: datetime
```

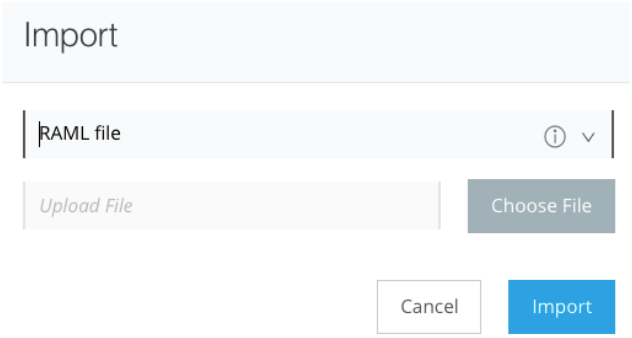
Note: Ignore the red errors in some of the user-defined datatypes reference. Once the files are referenced in the main RAML file, the errors won't affect the API design specification.

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.



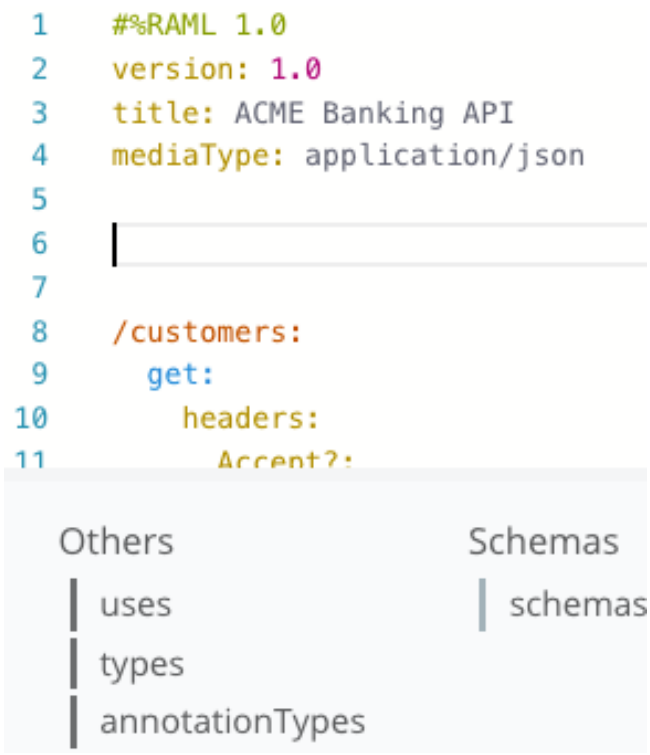
34. Click Import.
35. In the Import file dialog box, click the Choose File button.



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.



44. In the new line type,

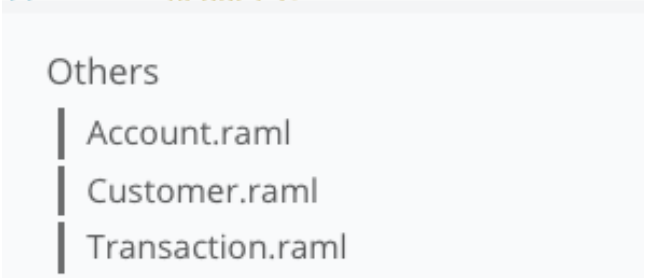
    Customer: !i

    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.

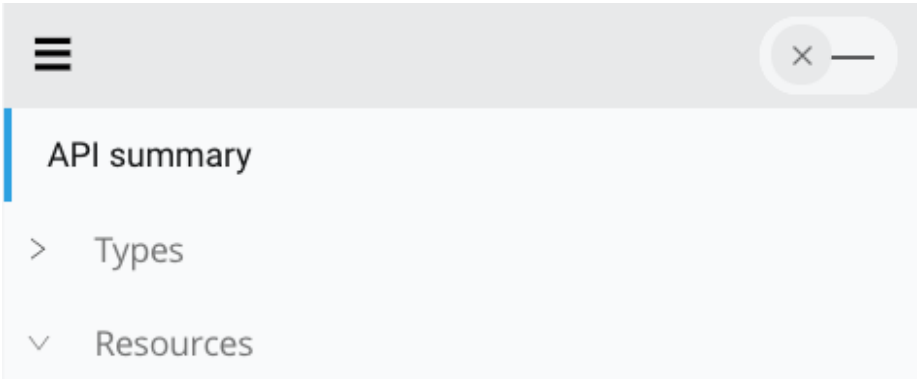
```
6   types:
7     Customer: !include datatypes/
8
9   /customers:
10    get:
11      headers:
```



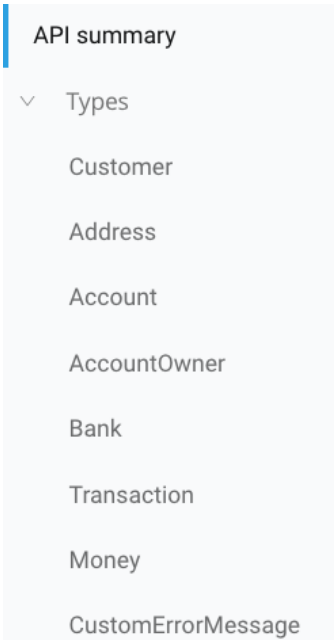
48. Similarly, include rest of the datatypes in the acme-banking-api.raml file.

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.



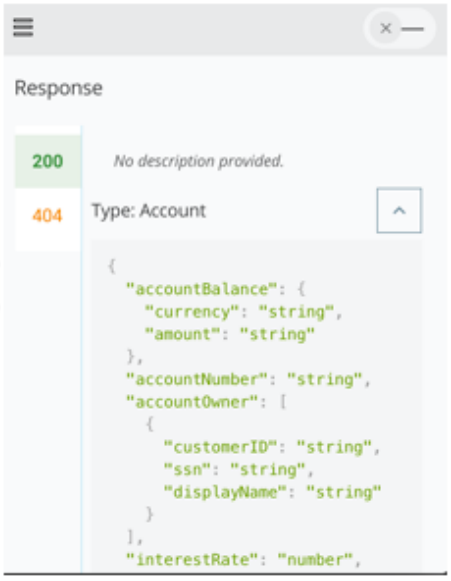
52. Click the Customer datatype to view its properties.

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
```



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        404:
```

Docs
- displayName
- example
- description

Others
- facets
- type
- 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        404:
```

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:
82          get:
83            responses:
84              200:
85                body:
86                  type: Account[]
87              404:
```

```
139       /transactions:
140         get:
141           responses:
142             200:
143               body:
144                 type: Transaction[]
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:
96        body:
97          type: Account
98        responses:
99          201:
100           headers:
```

```
152   /transactions:
153     post:
154       body:
155         type: Transaction
156       responses:
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}:
110       get:
111         responses:
112           200:
113             body:
114               type: Account
```

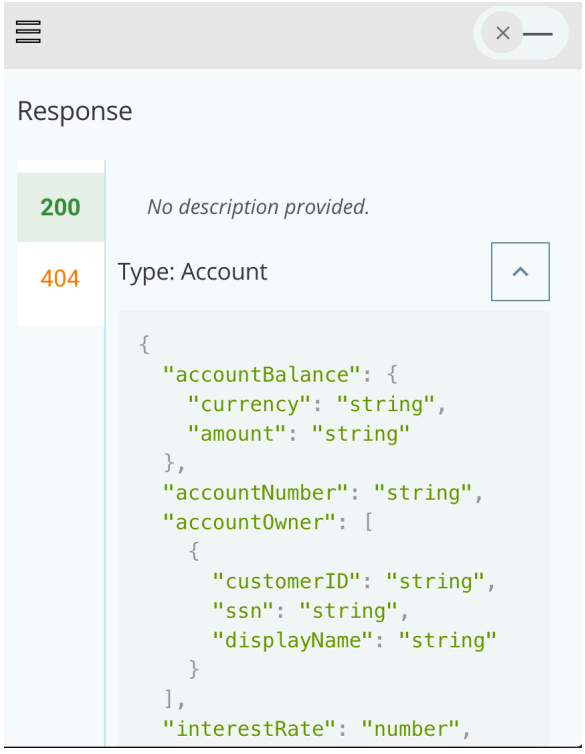
```
168     /{transaction_id}:
169       get:
170         responses:
171           200:
172             body:
173               type: Transaction
```

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.



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:
```

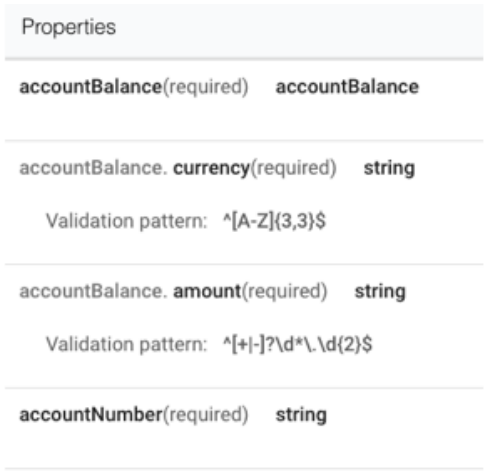
21. Modify all the other error response bodies in the RAML definition to be of this datatype.

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.

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



Specify attribute patterns and format values for necessary datatype attributes

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

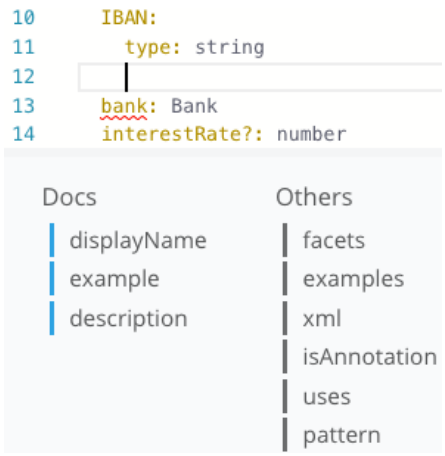
```
  IBAN:

  bank: Bank
  interestRate?: number
  createdAt: datetime
  modifiedAt?: datetime
```

5. 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.



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.

```
#%RAML 1.0 DataType
type: object
properties:
  currency:
    type: string

  amount:
    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.

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

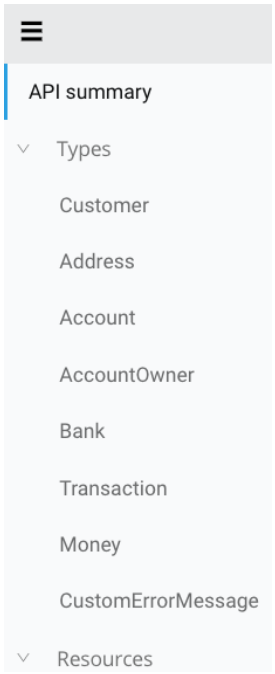
18. Add a new line below the amount type node line.
19. In the shelf, click pattern.
20. Return to the course snippets.txt file and copy the line that contains the pattern value for amount and paste it in the RAML editor.

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

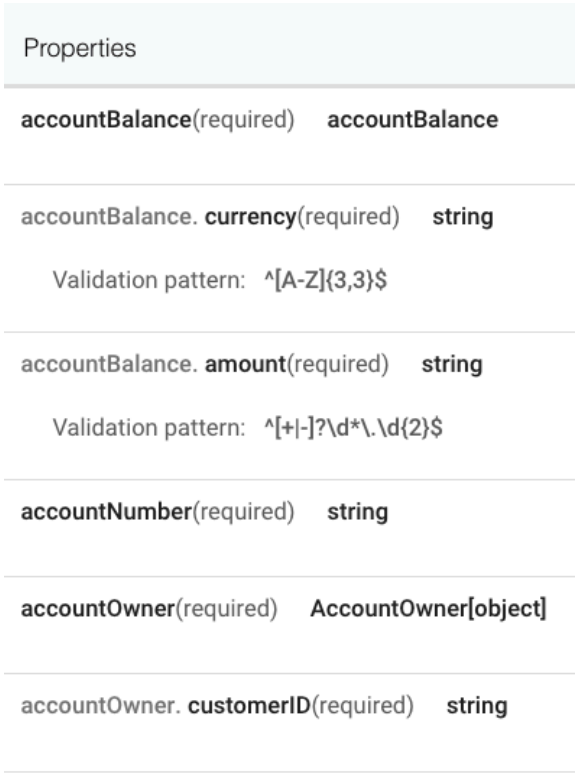
View the API documentation

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.



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.



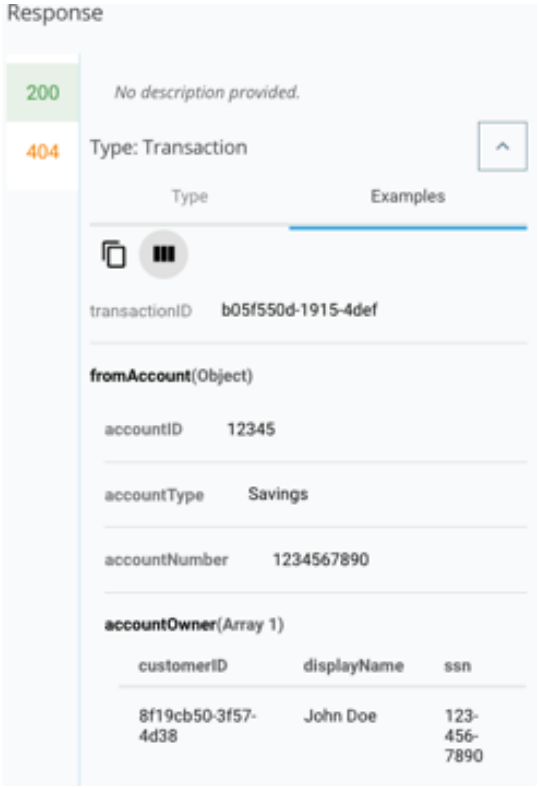
26. Click the top left menu icon in the API Console to view the list of datatypes and resources.

Walkthrough 6-5: Define example fragments for datatypes

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

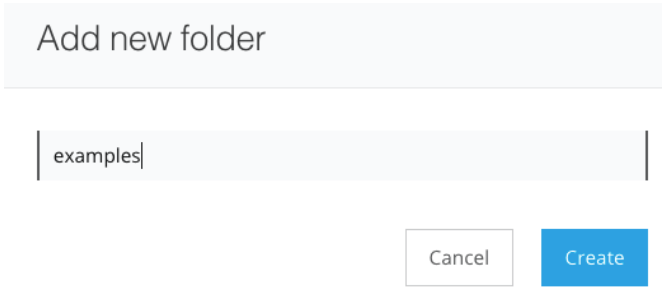
- Create example fragments for the datatypes.
- Include example fragments in response bodies.

```
#%RAML 1.0 NamedExample
value:
  transactionID: b05f550d-1915-4def
  fromAccount:
    accountID: '12345'
    accountType: Savings
    accountNumber: '1234567890'
    accountOwner:
      -
        customerID: 8f19cb50-3f57-4d38
        displayName: John Doe
        ssn: 123-456-7890
    accountBalance:
      currency: USD
      amount: '8457.90'
    IBAN: GB29NWBK60161331926820
    bank:
      bankCode: NWBKGB2L
      bankName: ACME Bank
      routingNumber: '432159876'
    createdAt: 2012-03-07T00:00:00.001Z
  toAccount:
    accountID: '56437'
    accountType: Credit Card
    accountNumber: '4321987650'
    accountOwner:
```

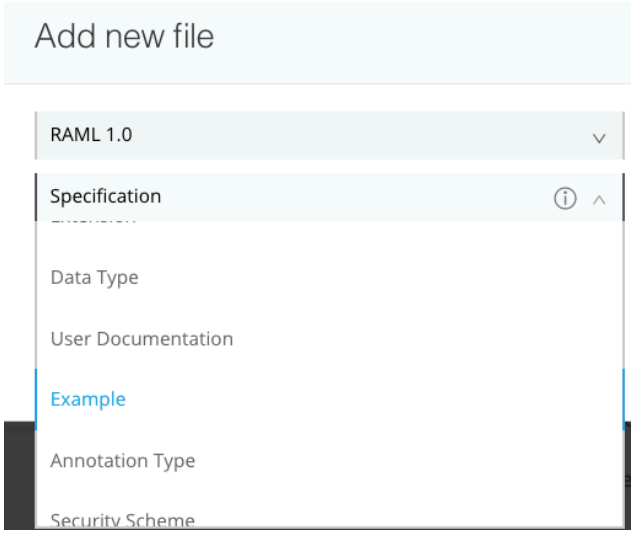


Create example fragment for the Customer datatype

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.



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.

```
#%RAML 1.0 NamedExample
value:
  customerID: 8f19cb50-3f57-4d38
  firstName: John
  lastName: Doe
  displayName: John Doe
  address:
    addressLine1: 1234 Lane
    addressLine2: Apt.#620
    city: San Francisco
    state: California
    zipCode: '94108'
    country: United States
  phone: 415-000-0000
  email: johndoe@example.com
  ssn: 123-456-7890
  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.

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

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.

```
#%RAML 1.0 NamedExample
value:
  transactionID: b05f550d-1915-4def
  fromAccount:
    accountID: '12345'
    accountType: Savings
    accountNumber: '1234567890'
    accountOwner:
      -
        customerID: 8f19cb50-3f57-4d38
        displayName: John Doe
        ssn: 123-456-7890
    accountBalance:
      currency: USD
      amount: '8457.90'
    IBAN: GB29NWBK60161331926820
    bank:
      bankCode: NWBKGB2L
      bankName: ACME Bank
      routingNumber: '432159876'
    createdAt: 2012-03-07T00:00:00.001Z
  toAccount:
    accountID: '56437'
    accountType: Credit Card
    accountNumber: '4321987650'
    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:
```

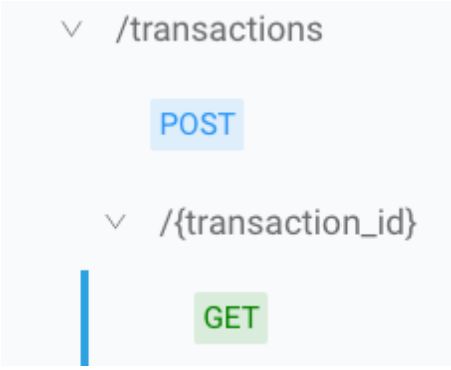
18. Similarly, add the TransactionExample.raml example in the /{transaction_id} resource get method.

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

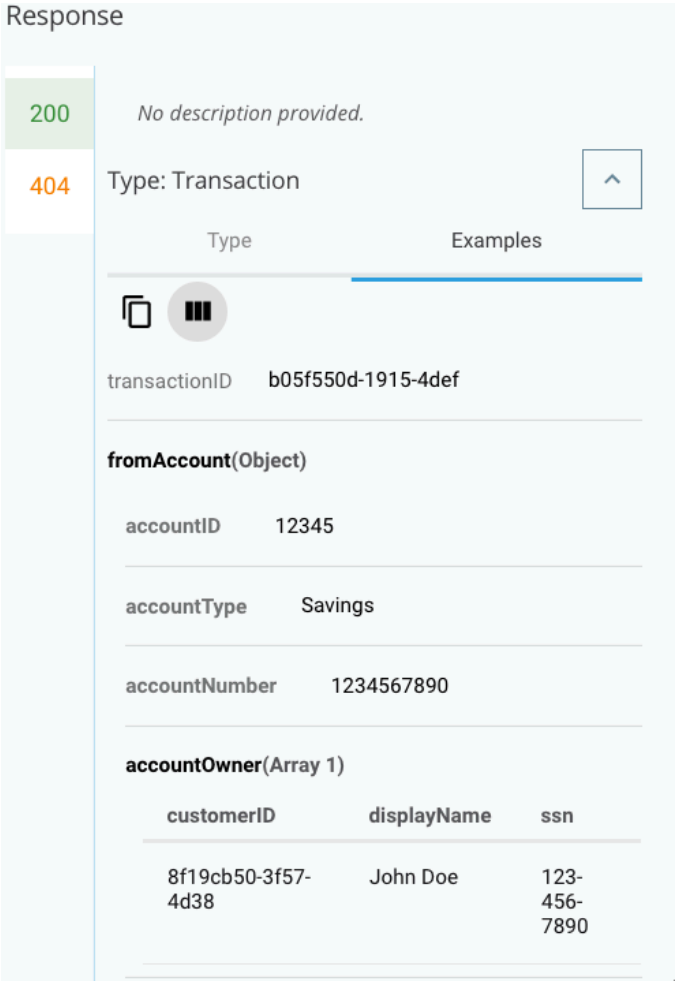
View the API Console

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.

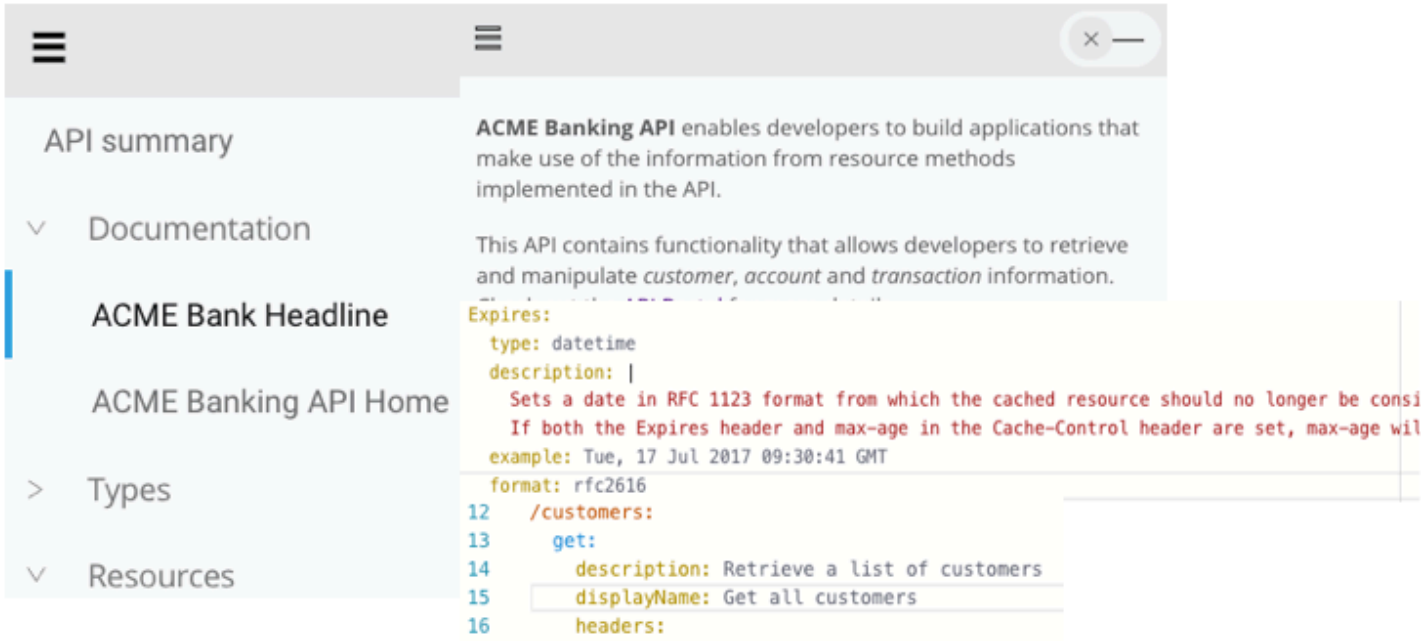


21. In the Response section, click the Examples tab and verify that you see the TransactionExample.



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

Module 7: Documenting and Testing APIs



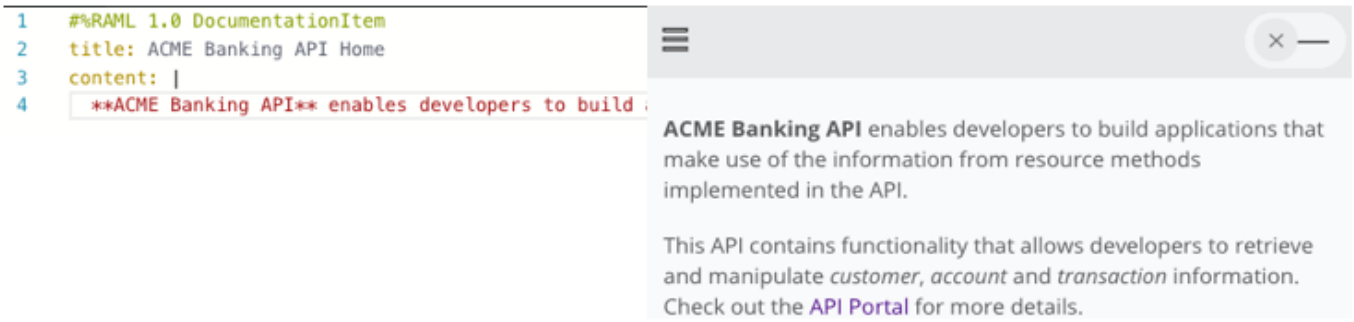
Objectives:

- 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:

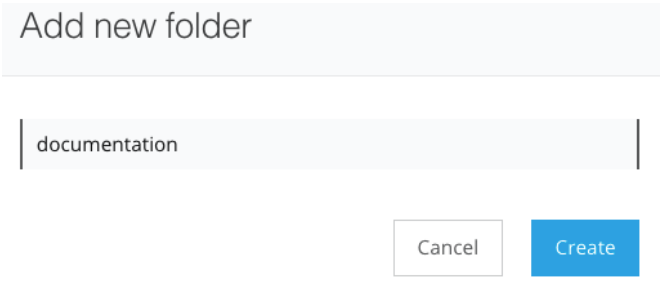
- Create a documentation fragment.
- Link the documentation fragment to the main RAML definition file.
- Link multiple documentation fragments to the documentation node in the RAML API definition.



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 -
```

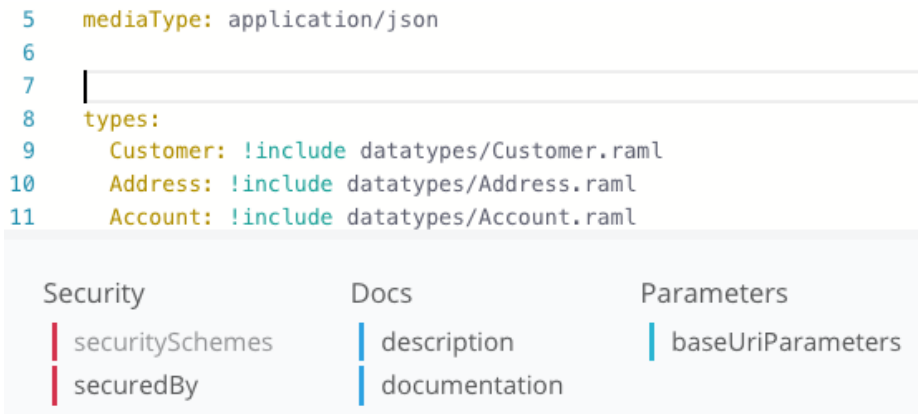
18. Return to API designer and add two new lines after the first snippet documentation.

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

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

  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.

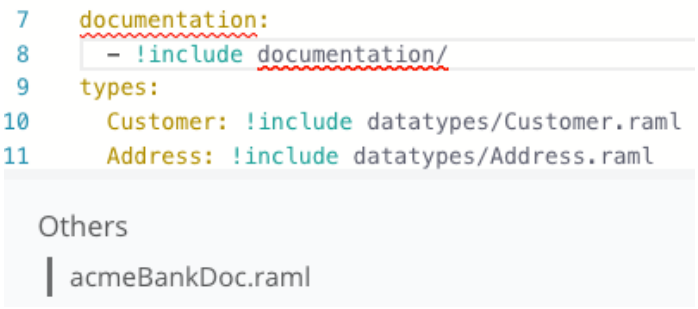


23. In the new line created below the documentation node, type the following:

```
- !include
```

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

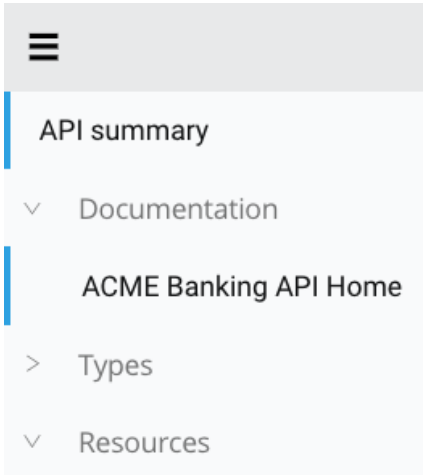
24. In the shelf, click documentation.
23. After the word documentation word, type /.
24. In the shelf, click acmeBankDoc.raml.



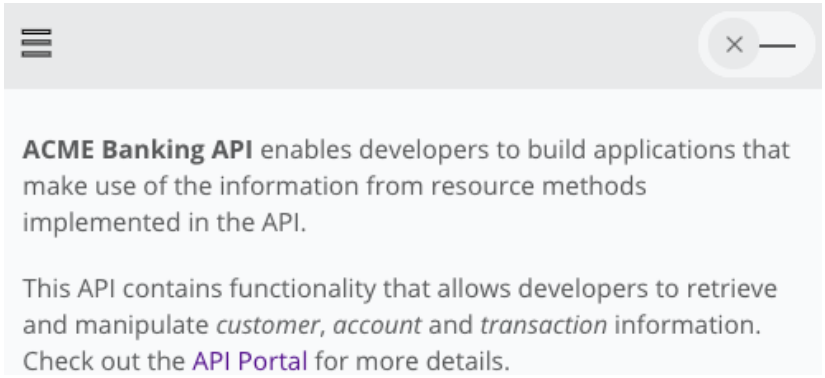
25. Add a new line.

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.



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.
25. Set the file name to acmeBankHeadline.raml.
31. Click Create.
32. 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:

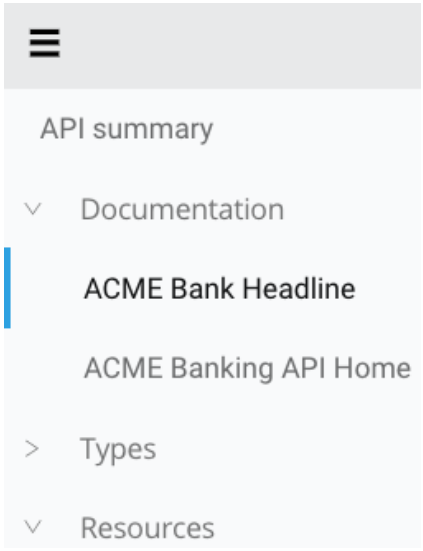
    - !include documentation/acmeBankHeadline.raml

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

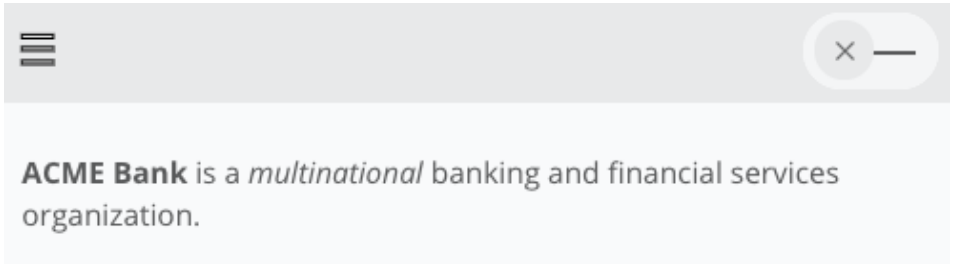
View the API Console

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

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



42. View the documentation content in the API Console.



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.

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

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.

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

Body
body
Root
protocols
Docs
description
displayName

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?:
```

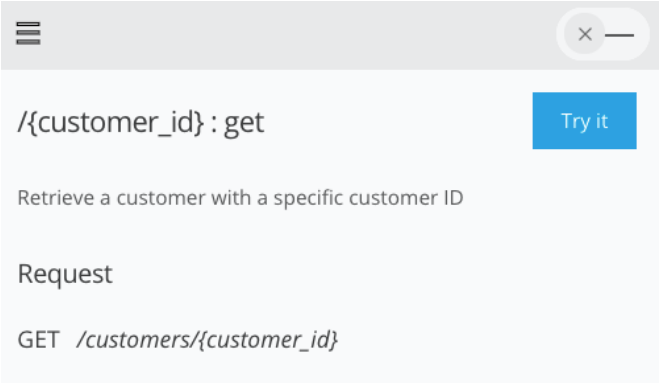
Note: The displayName node helps in better readability of the API functionality in the REST Connect functionality in Flow Designer.

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.



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:
```

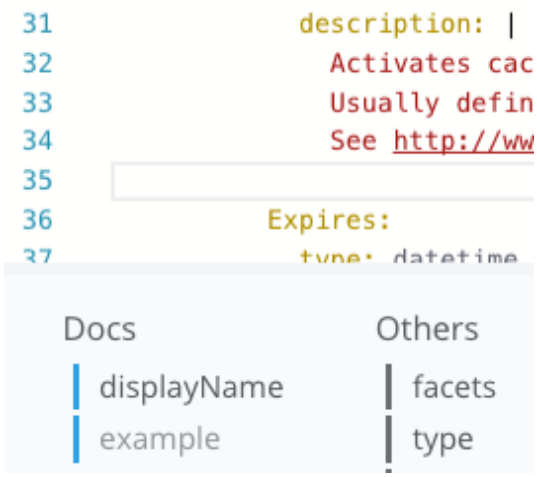
22. Return to API designer, press tab in the new line and paste the text in the new empty line.

```
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 resourc
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.



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:
```

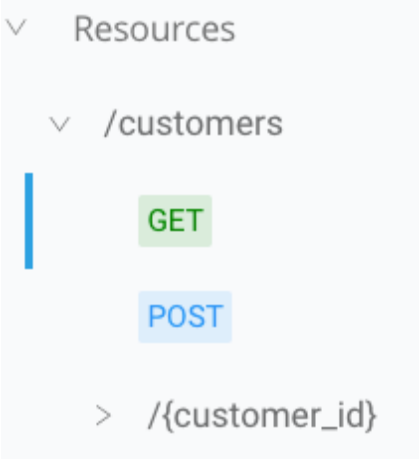
29. Press enter to add a new line and add an example node.

30. Type the following value for the example node:

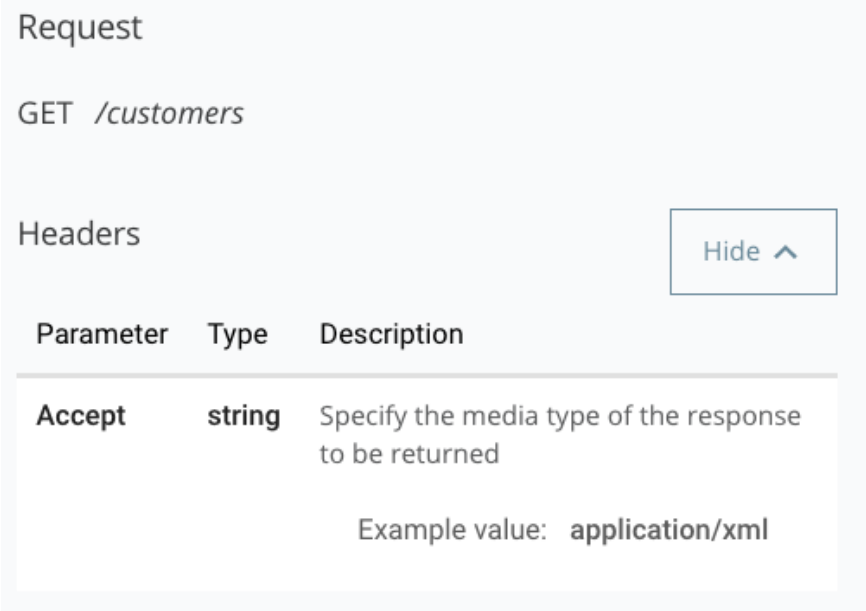
```
/customers/8f19cb50-3f57-4d38
```

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.

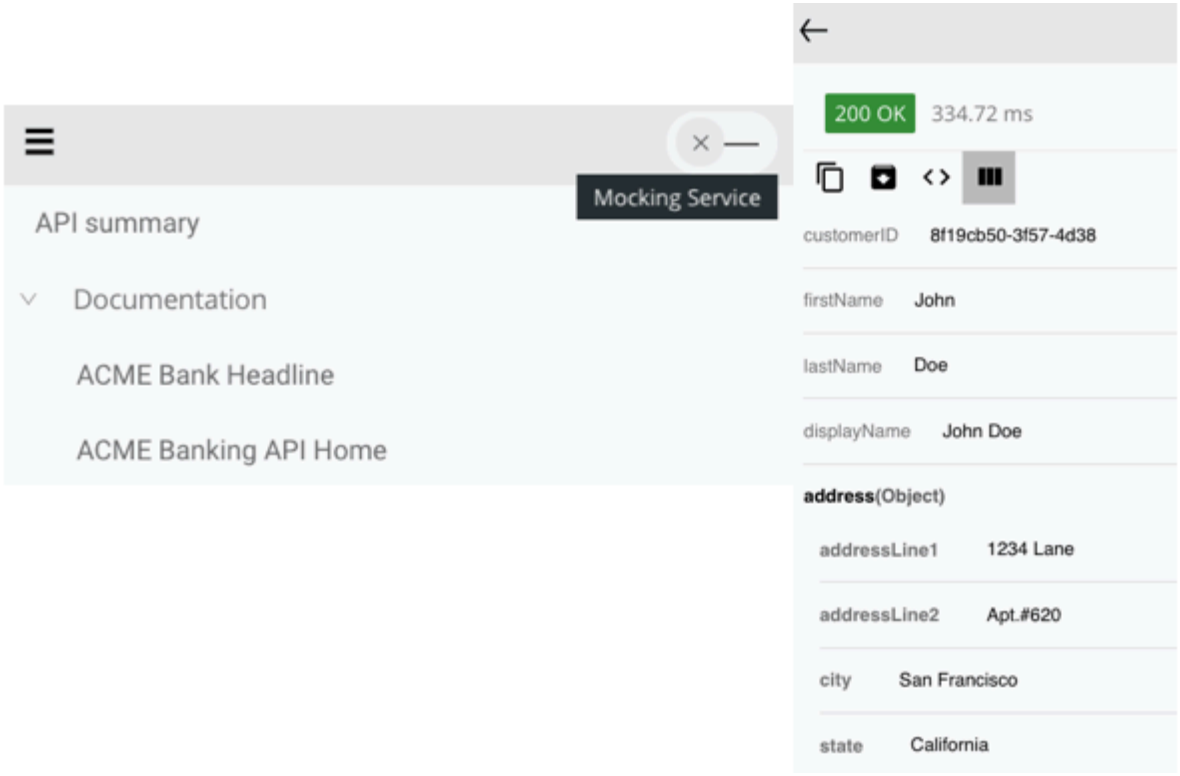


34. Click the top left menu icon to go back to the list of resources

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

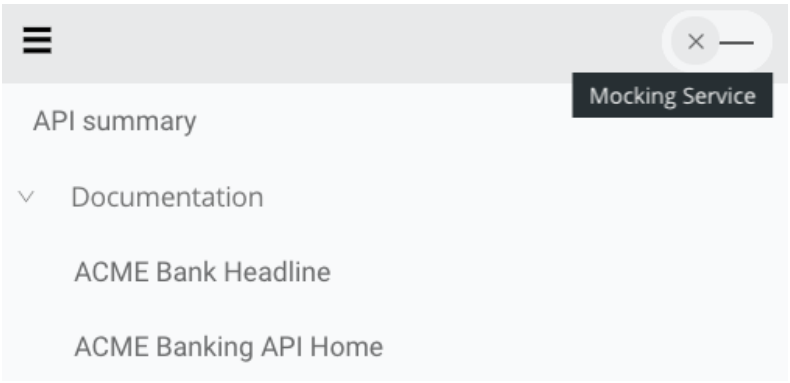
In this walkthrough, you will test your API by enabling mocking service. You will:

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

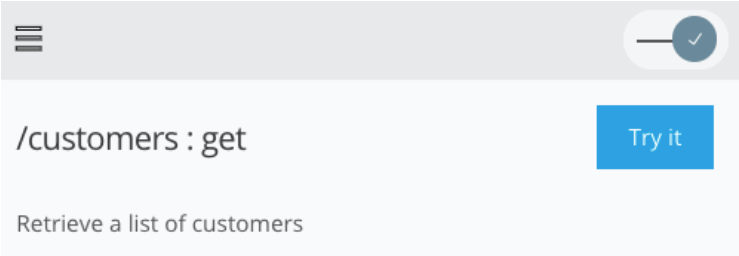


Enable the mocking service in the API Console

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



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

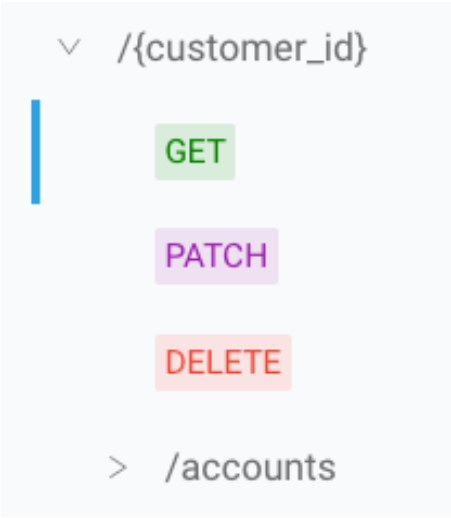


4. Look at acme-banking-api.raml in the RAML editor; you should now see a new baseUri parameter was added above the title node.

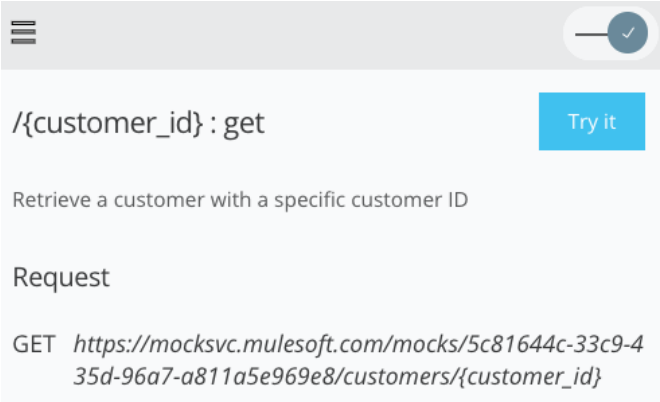
```
1   #%RAML 1.0
2   baseUri: https://mocksvc.mulesoft.com/mocks/5c81644c-33c9-435d-96a7-a811a5e969e8
3   version: 1.0
4   title: ACME Banking API
5   mediaType: application/json
```

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

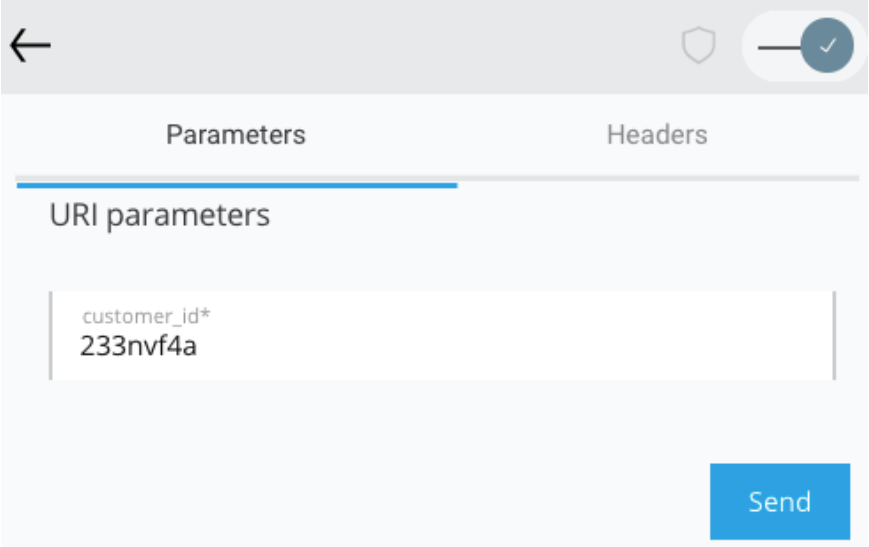
5. In the API Console, click the top left menu icon.
6. Go to the /{customer_id} resource, expand it (to view the methods) and click the GET method link.



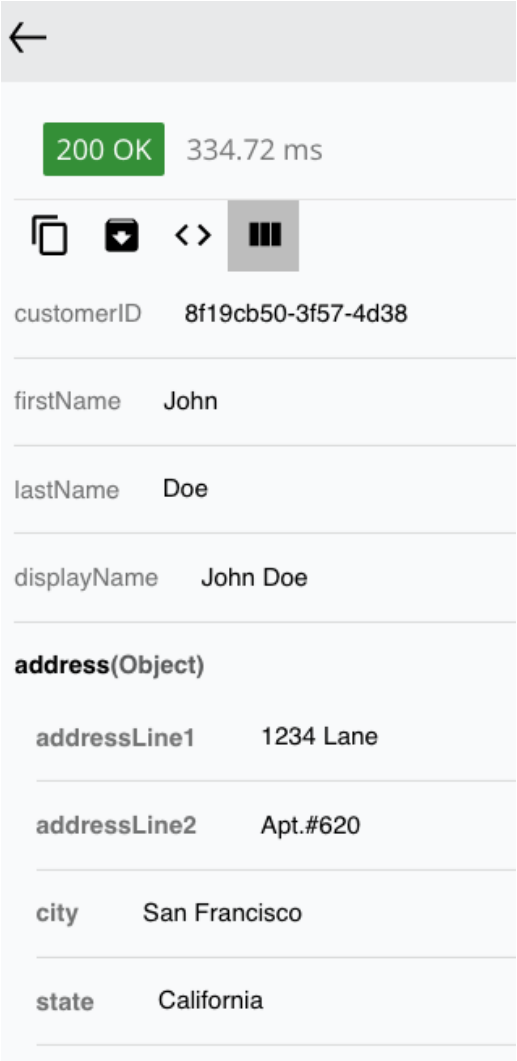
7. Click Try it.



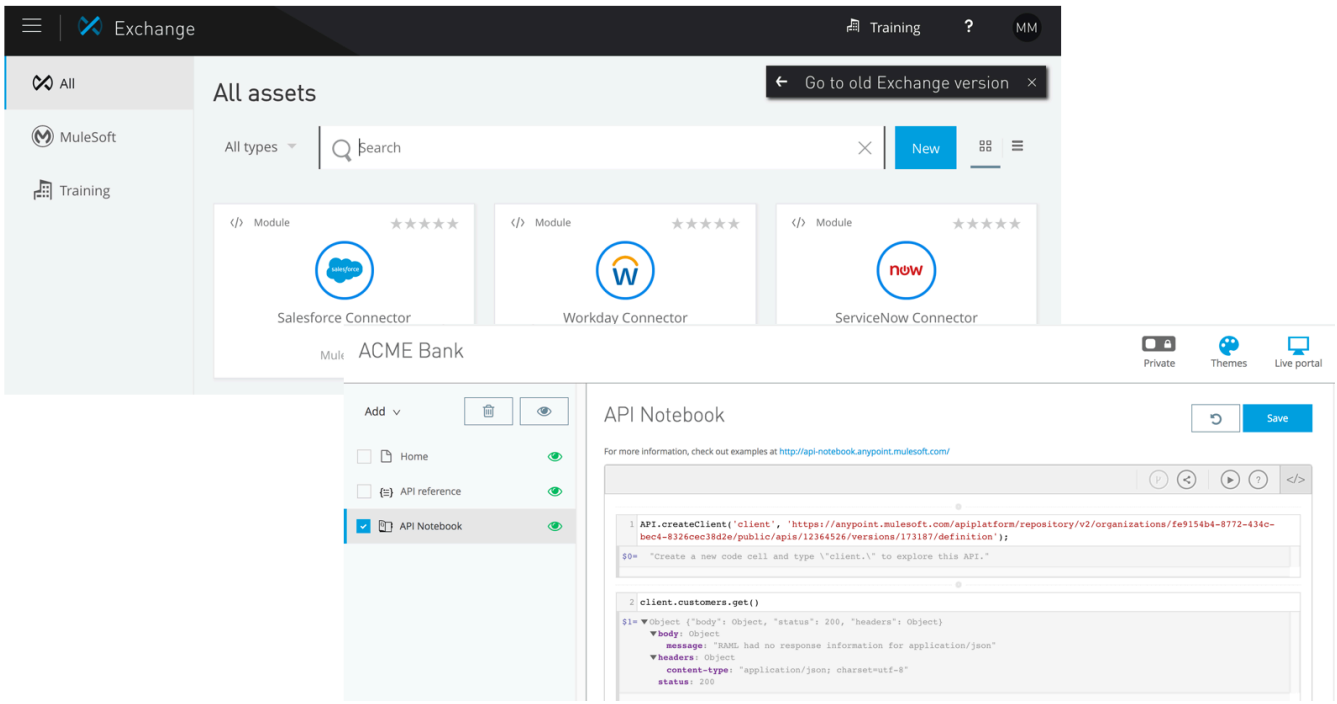
8. In the URI parameters field, enter any value as customer_id and click Send.



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



Module 8: Making APIs Discoverable



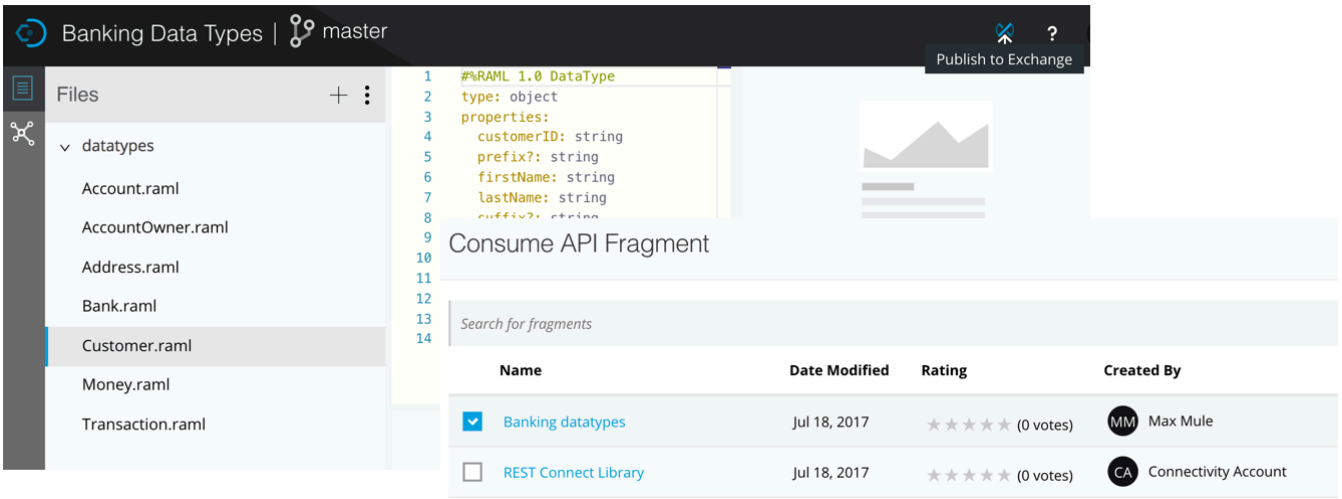
Objectives:

- 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

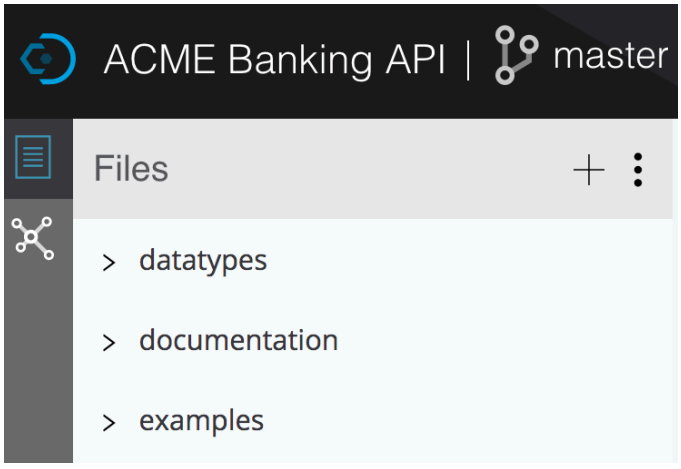
In this walkthrough, you promote reusability of an API fragment by publishing it to the private Anypoint Exchange. You will:

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



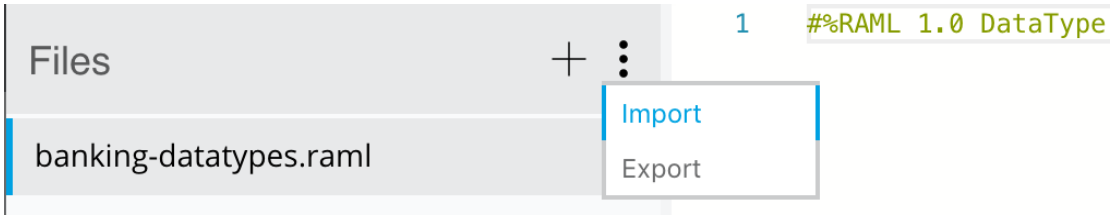
Create an API fragment project in Anypoint Platform Design Center

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.

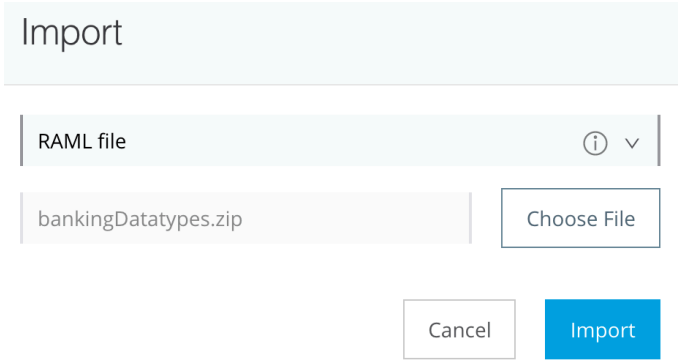


2. In the Design Center homepage, click the Create button next to the search bar.

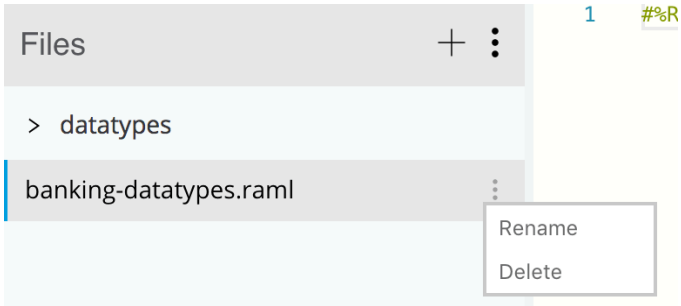
3. In the New Project dialog box, type the Project name as Banking Data Types.
4. In the Project Type field, click the drop-down menu and select API Fragment.
5. In the Fragment Type field, click the drop-down menu and select Data Type.
6. Click Create.
7. In API designer, click the menu icon in the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



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



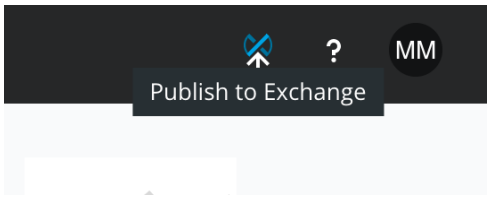
12. In the file browser section, click the menu icon next to banking-datatypes.raml.



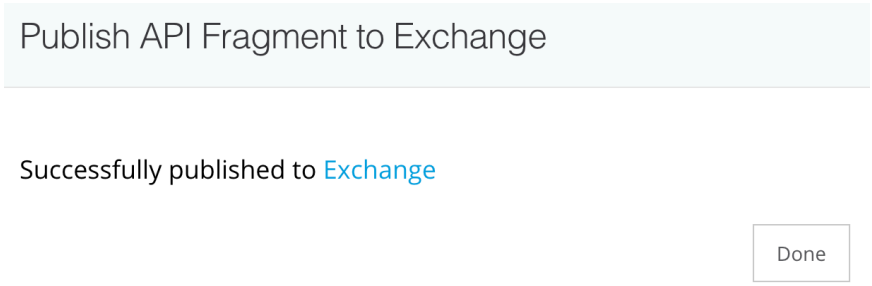
13. Click Delete.

Publish an API Fragment to Anypoint Exchange

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



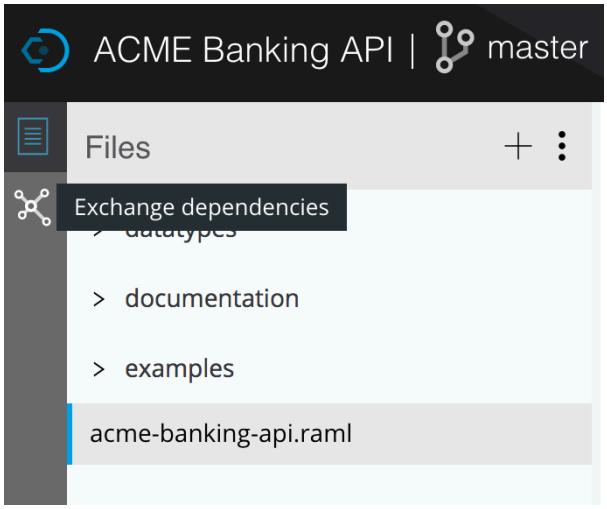
15. In the Publish API Fragment to Exchange dialog box, click Publish.
16. Verify that you see a dialog box with a message that the Banking datatypes project was successfully published to Exchange.



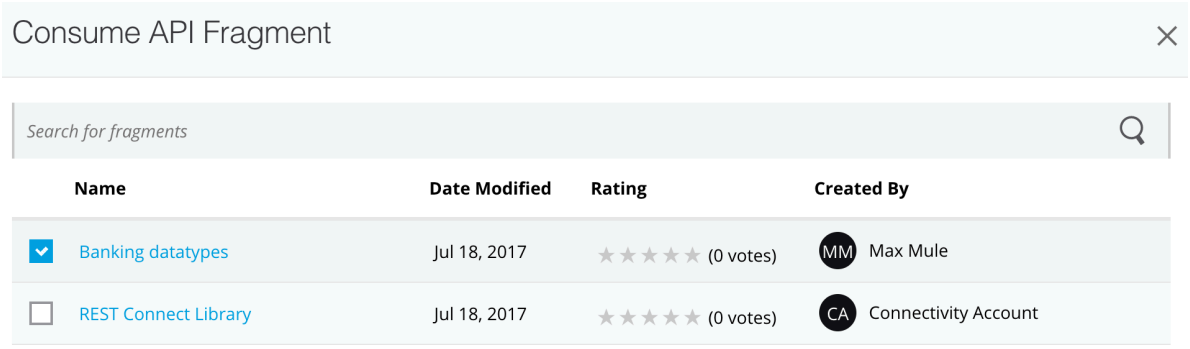
17. Click Done.

Consume an API Fragment from Anypoint Exchange

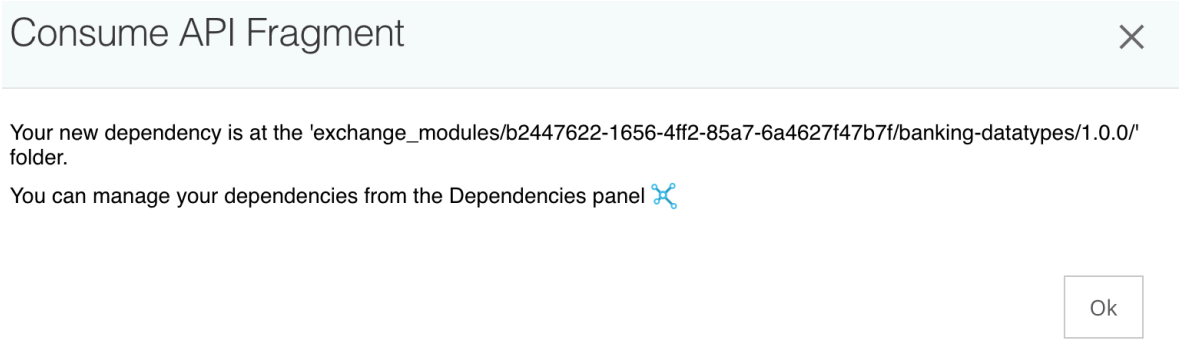
18. In the API designer page, click the Design Center icon in the top left corner next to the Banking datatypes project name.
19. In the Design Center homepage, click the ACME Banking API project.
20. In the file browser section, click the Exchange dependencies icon below the file icon.



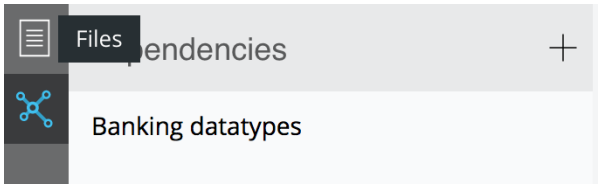
21. Click the + icon in the Dependencies header section.
22. In the Consume API Fragment dialog box, check the box for Banking datatypes API fragment.



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.



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

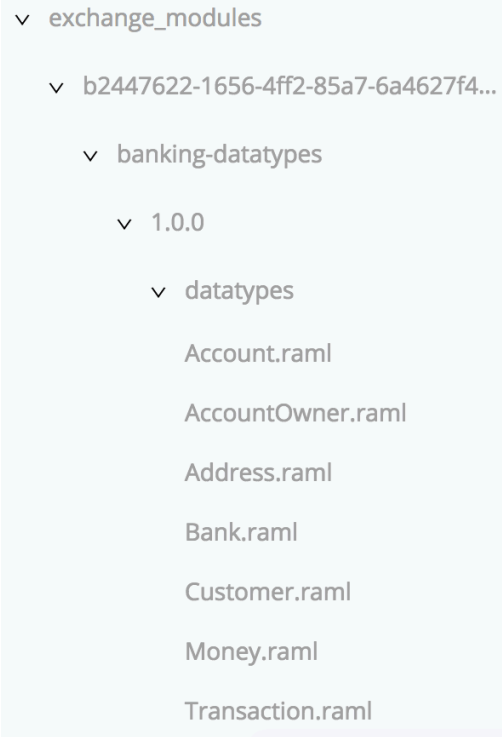


26. In the file browser section, click the acme-banking-api.raml file.
27. In the RAML editor, go to the lines that contain the types definition.
28. Remove the lines that include Money, AccountOwner, Bank, Address datatype fragments.

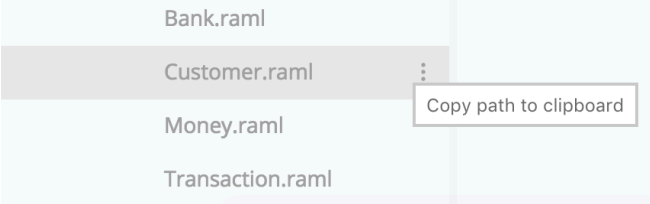
```
11  types:
12    Customer: !include datatypes/Customer.raml
13    Account: !include datatypes/Account.raml
14    Transaction: !include datatypes/Transaction.raml
15    CustomErrorMessage: !include datatypes/CustomErrorMessage.raml
```

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.



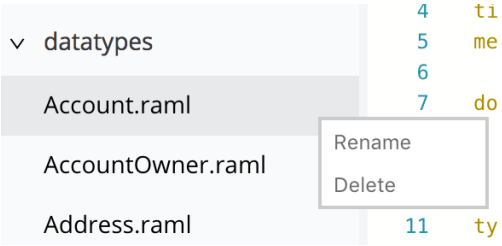
31. Click Copy path to clipboard.
32. In the acme-banking-api.raml file in the editor, replace the include path for the Customer datatype with the path that you copied.

```
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
```

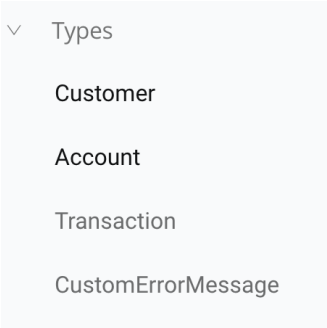
33. Similarly, repeat the steps to change the include path for Account and Transaction datatypes.

```
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.



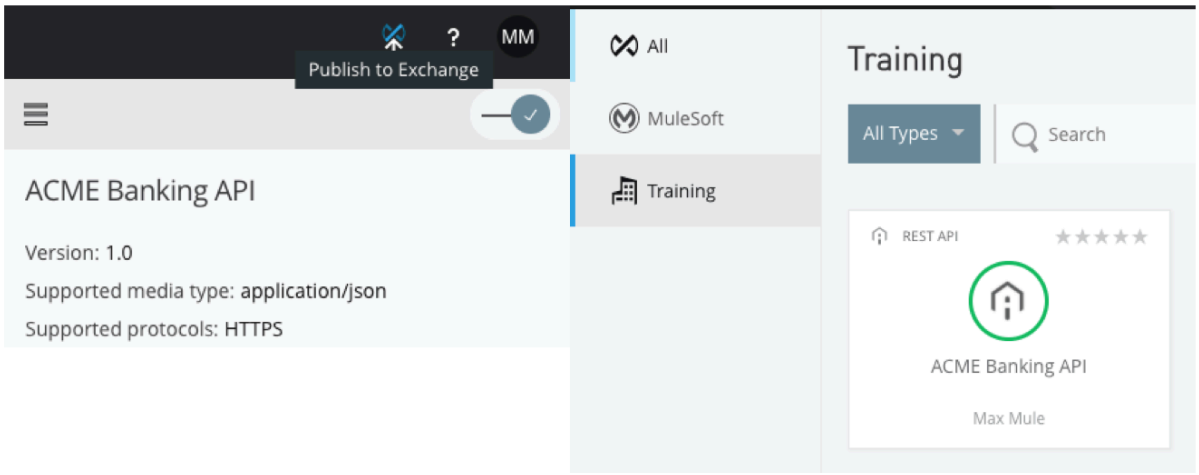
36. Similarly, delete the rest of the datatype files except the CustomErrorMessage.raml file.
37. In the API Console, click the top left menu icon and expand the Types section.
38. Verify that you see the Customer, Account and Transaction datatypes from exchange dependencies listed.



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

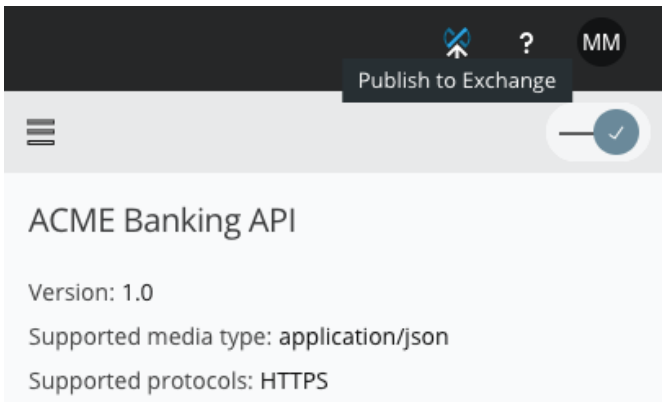
In this walkthrough, you enhance the discoverability of an API by adding it to the private Anypoint Exchange. You will:

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

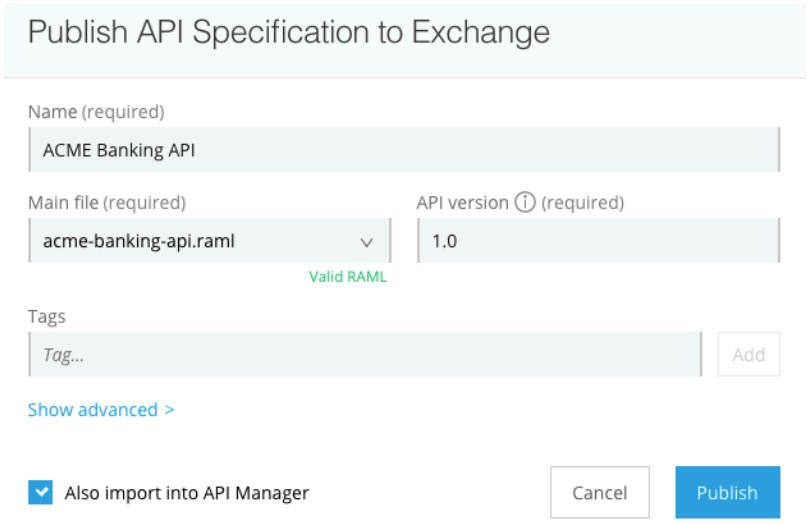


Publish a RAML API Specification to private Exchange

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



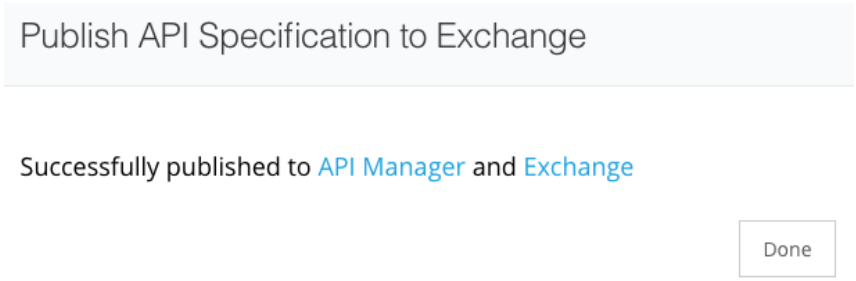
3. In the Publish API Specification to Exchange, check the option Also import into API Manager.



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.

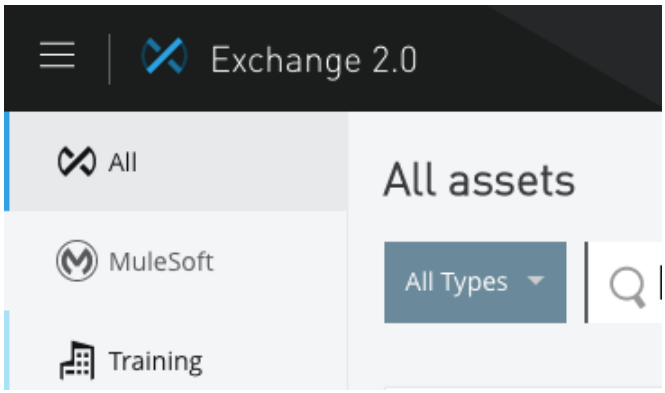


6. Click Done.

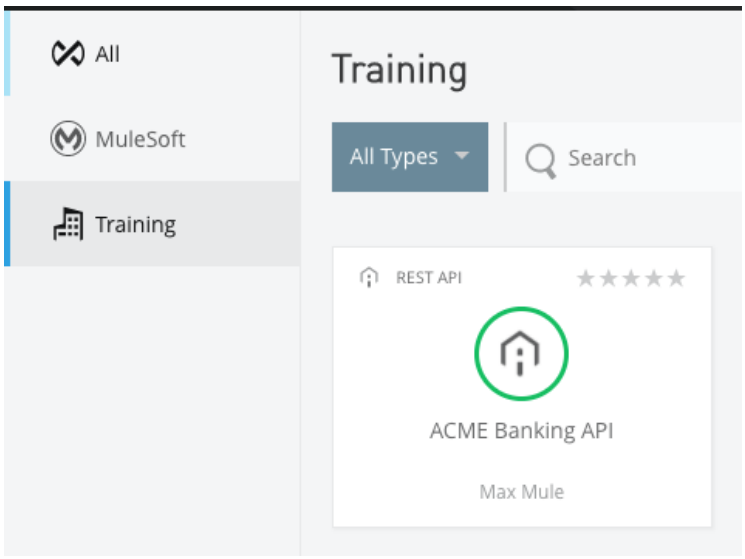
View the asset in Exchange

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.

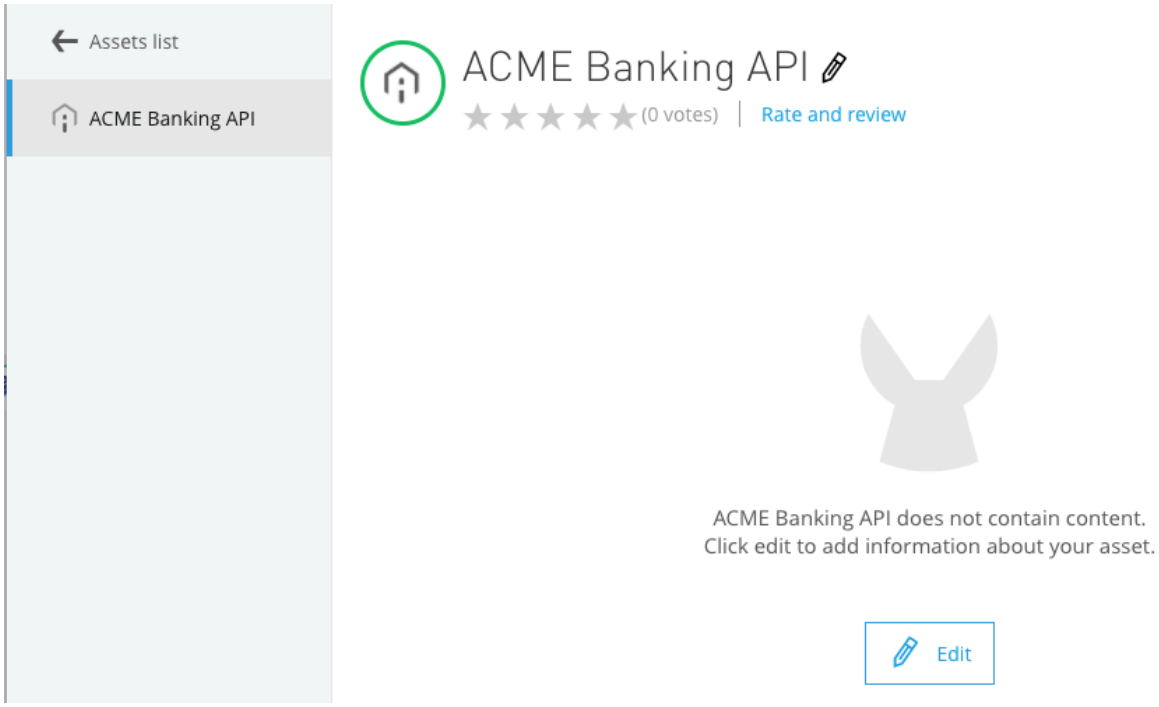


11. In the list of assets in your business group, click ACME banking API.

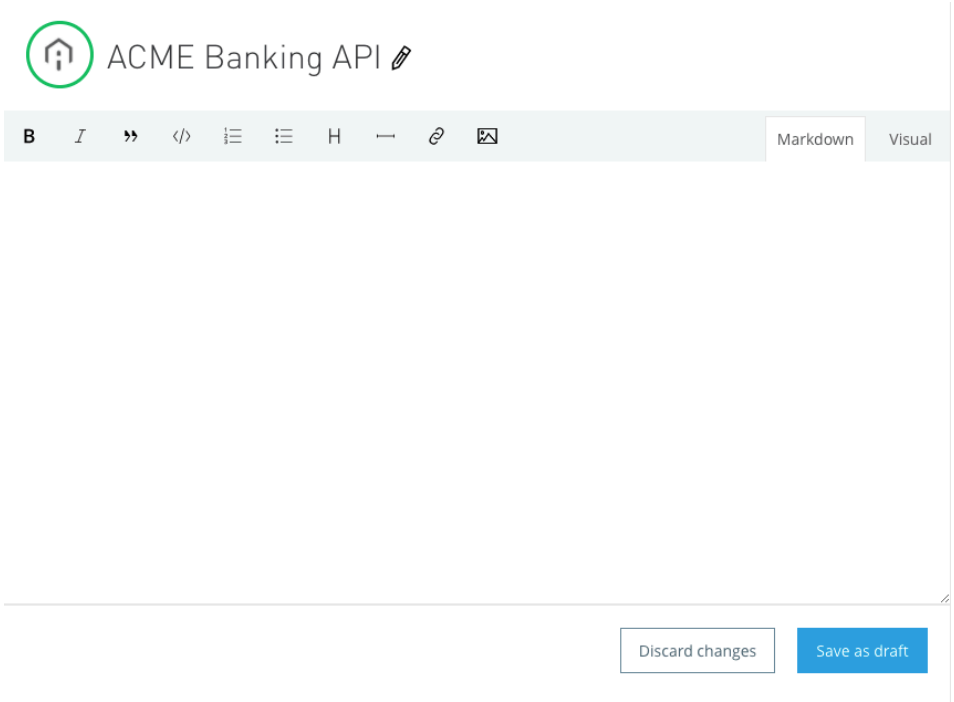


Add documentation to an Exchange asset

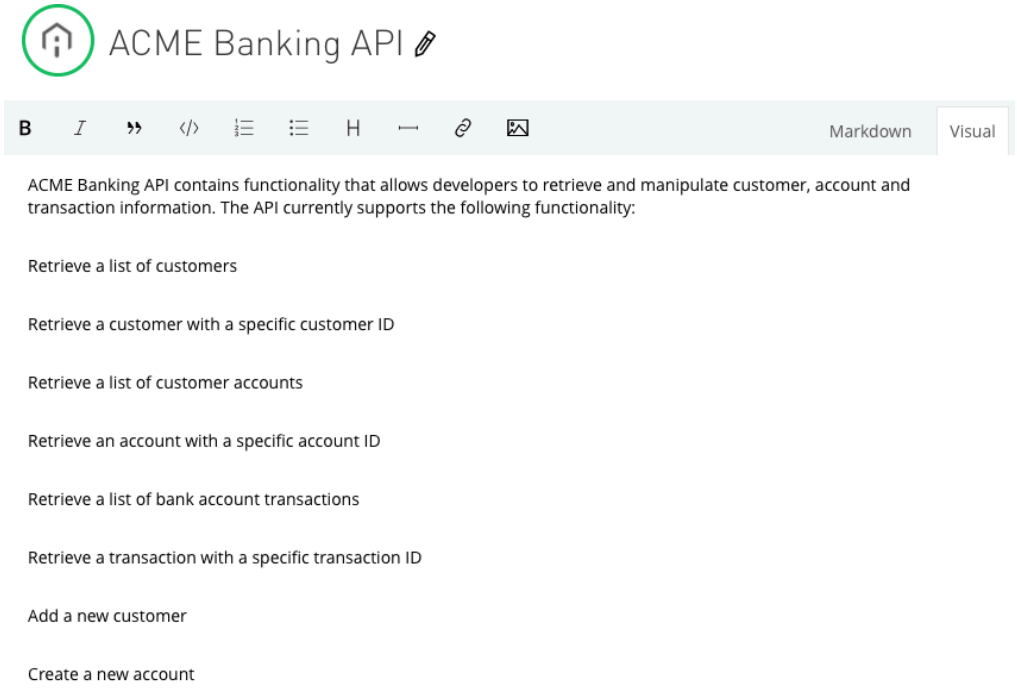
12. In the ACME Banking API asset page, click Edit in the center of the page to add information about the asset.



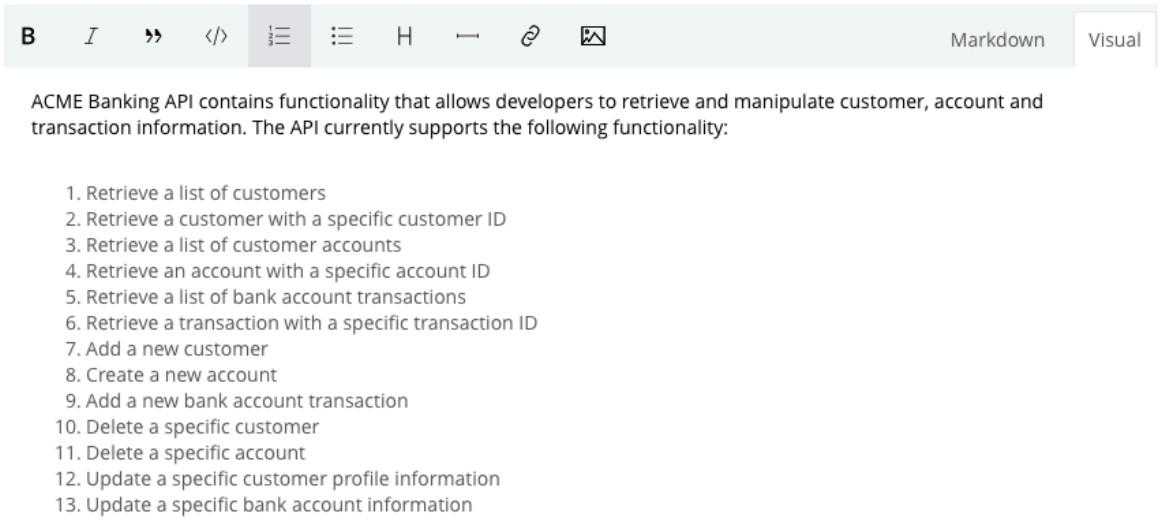
13. In the top right corner of the ACME Banking API content editor, change the tab from Markdown to Visual.



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.

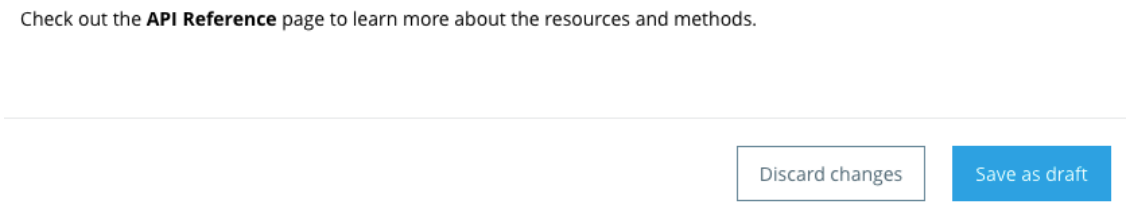


17. Select the lines that list out the API functionality and click the numbered bullets icon on top to add numbering.

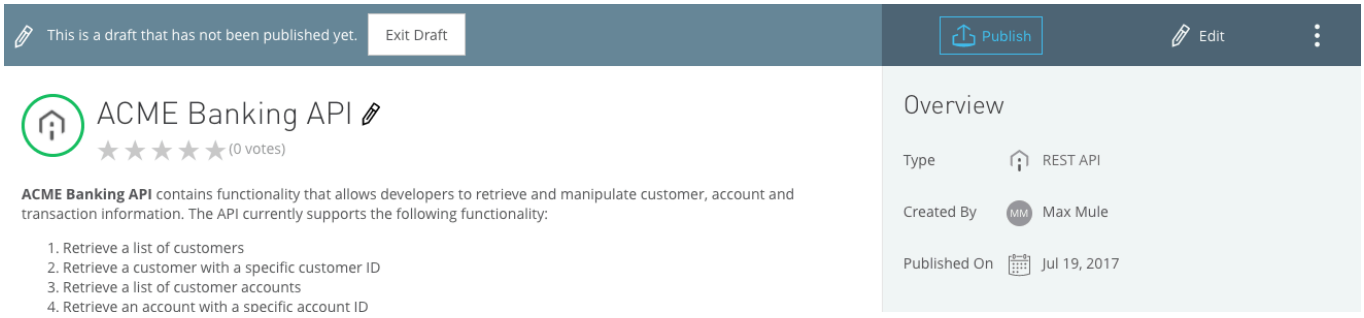


18. Select the ACME Banking API text at the beginning and click the B icon to bold the text.
19. Similarly, make the API Reference text below the numbered list bold.

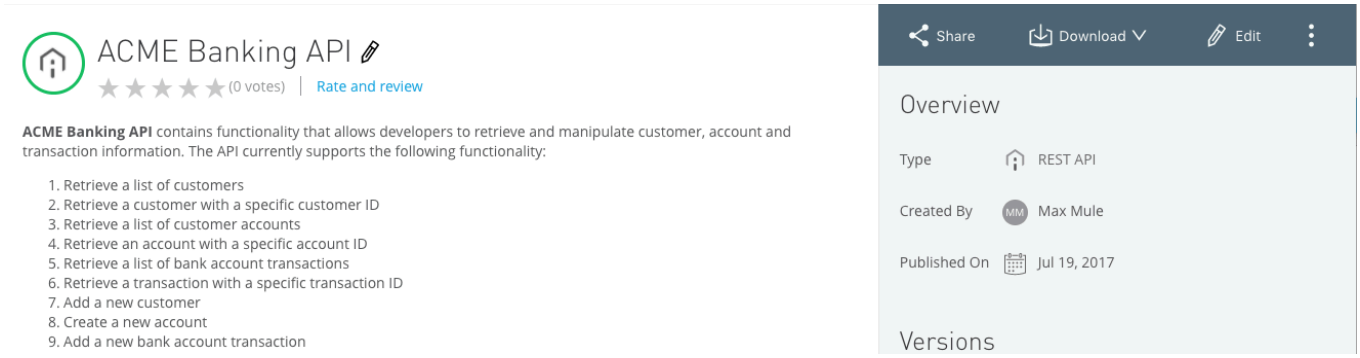
20. In the bottom left corner of the editor, click the Save as draft button.



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



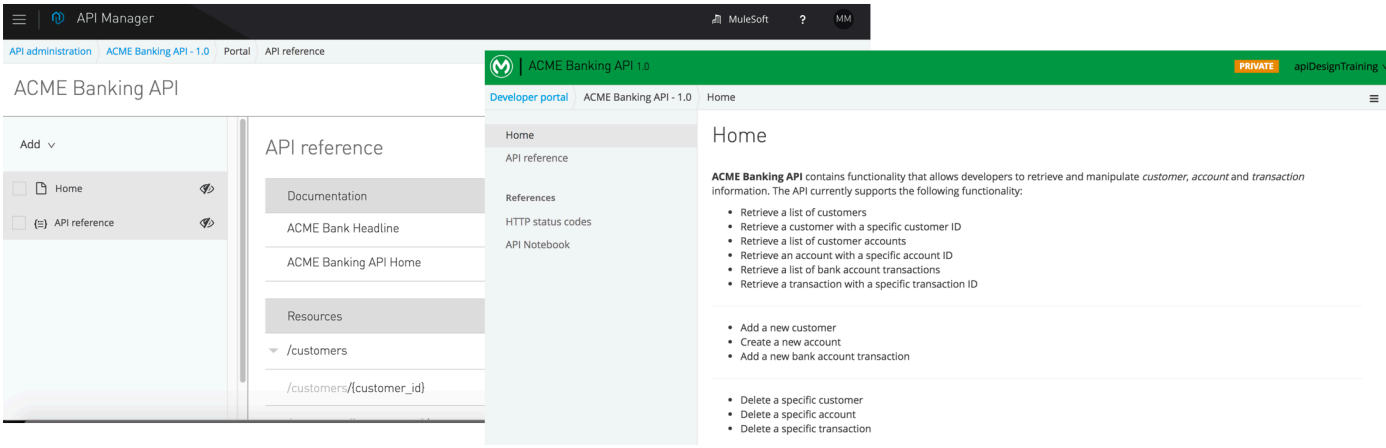
22. Verify that you are redirected to the Asset homepage, with the information about the asset listed in the center panel.



Walkthrough 8-3: Create and customize an API Portal

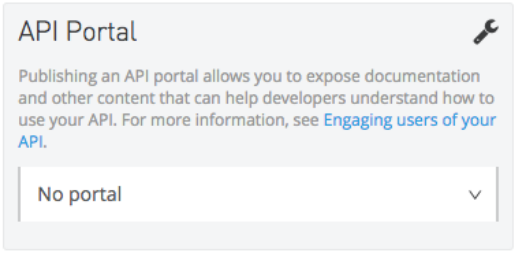
In this walkthrough, you will create an API Portal documenting information about the ACME Banking API. You will:

- 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.

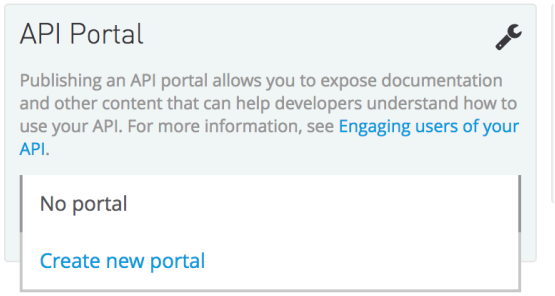


Create an API Portal

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.

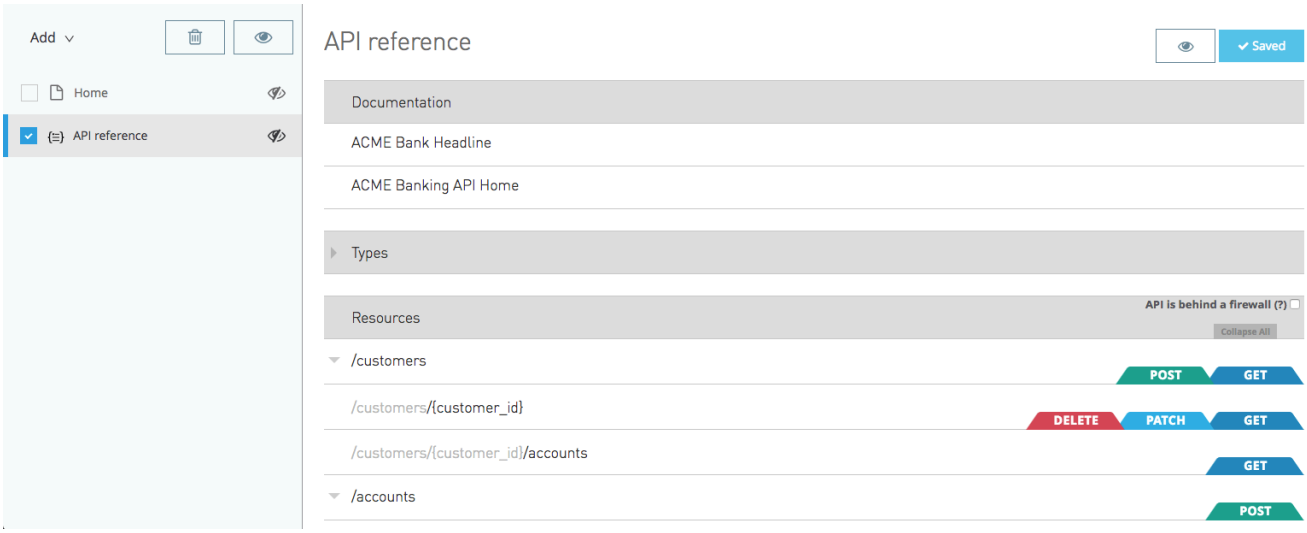


6. In the API Portal Designer, locate the two pages that were automatically created: Home and API reference.
7. In the left-side navigation, click the Home page link.
8. Examine the Home page editor.

   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.

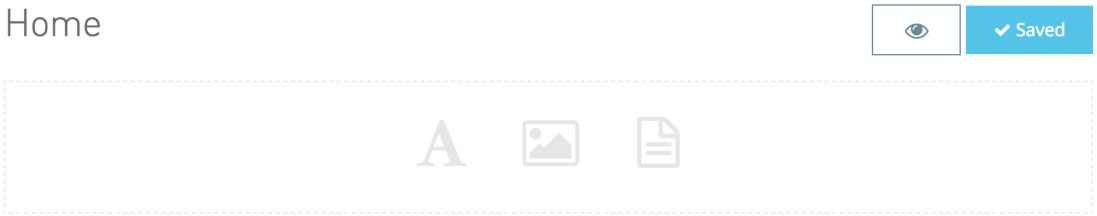
    Note: The API reference page is only populated if the API definition has a base URI specified.



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.

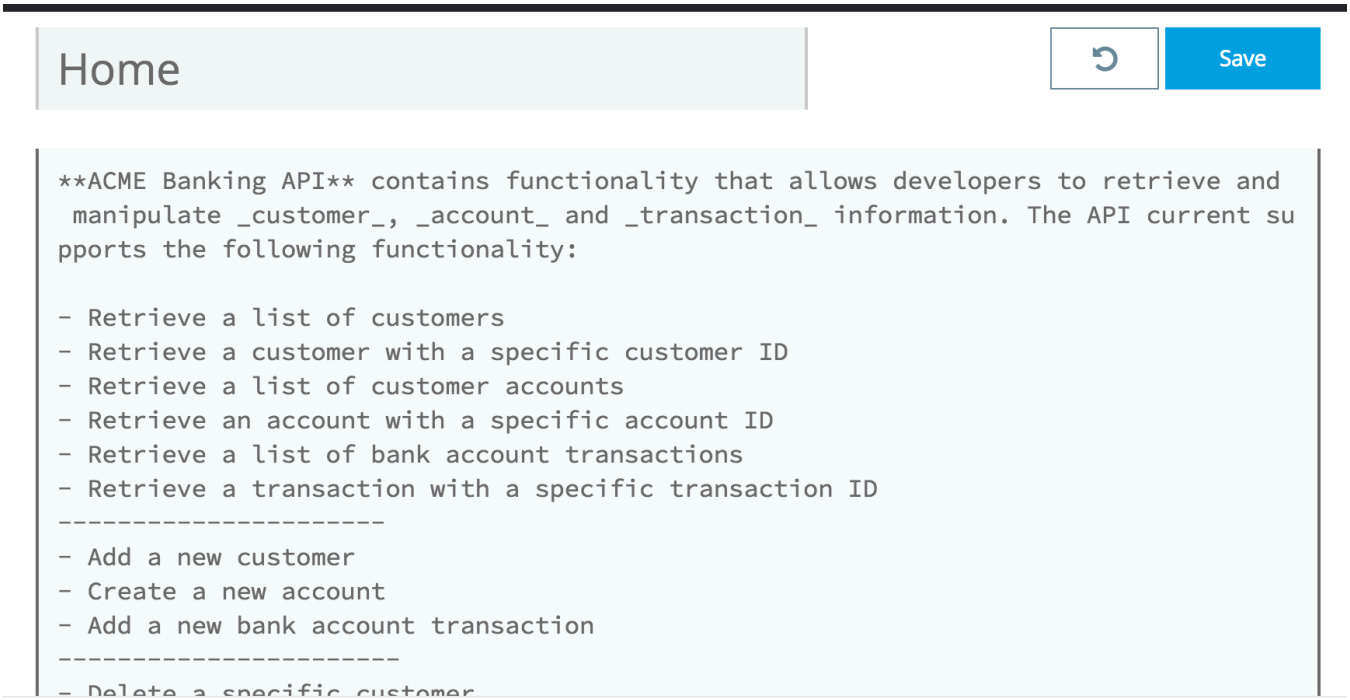


13. Return to the course snippets.txt file.
14. Locate the Module 8 section and copy the ACME Banking API Home content.

```
-------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.

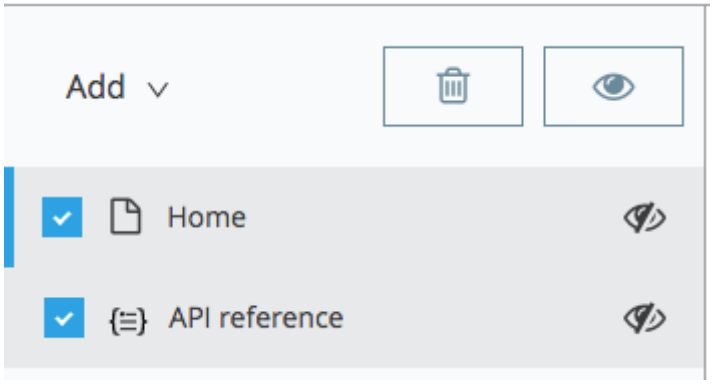
16. Paste the copied lines here.



17. Click the Save button.

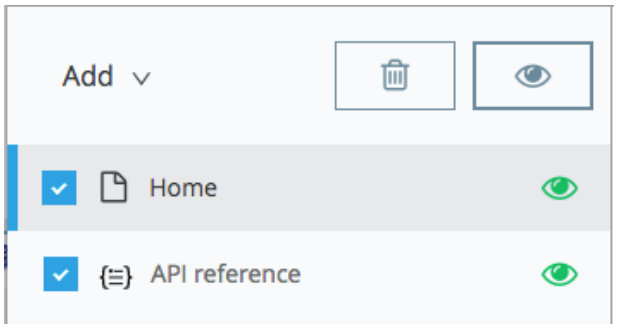
Publish the API Portal

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

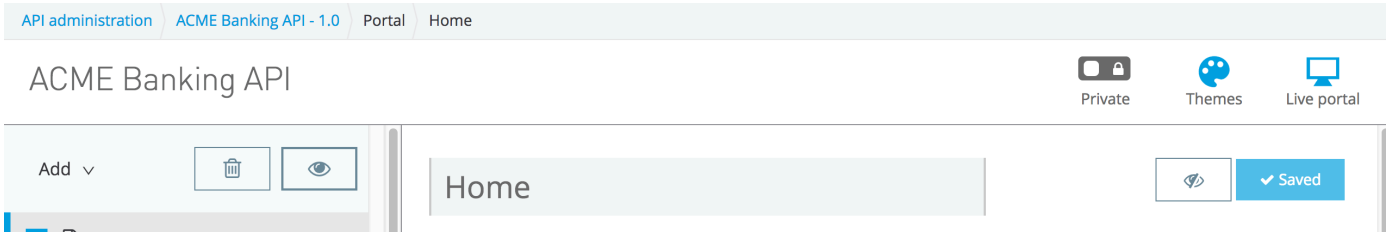


19. Click the eye button to set the selected pages to be visible in the API Portal.

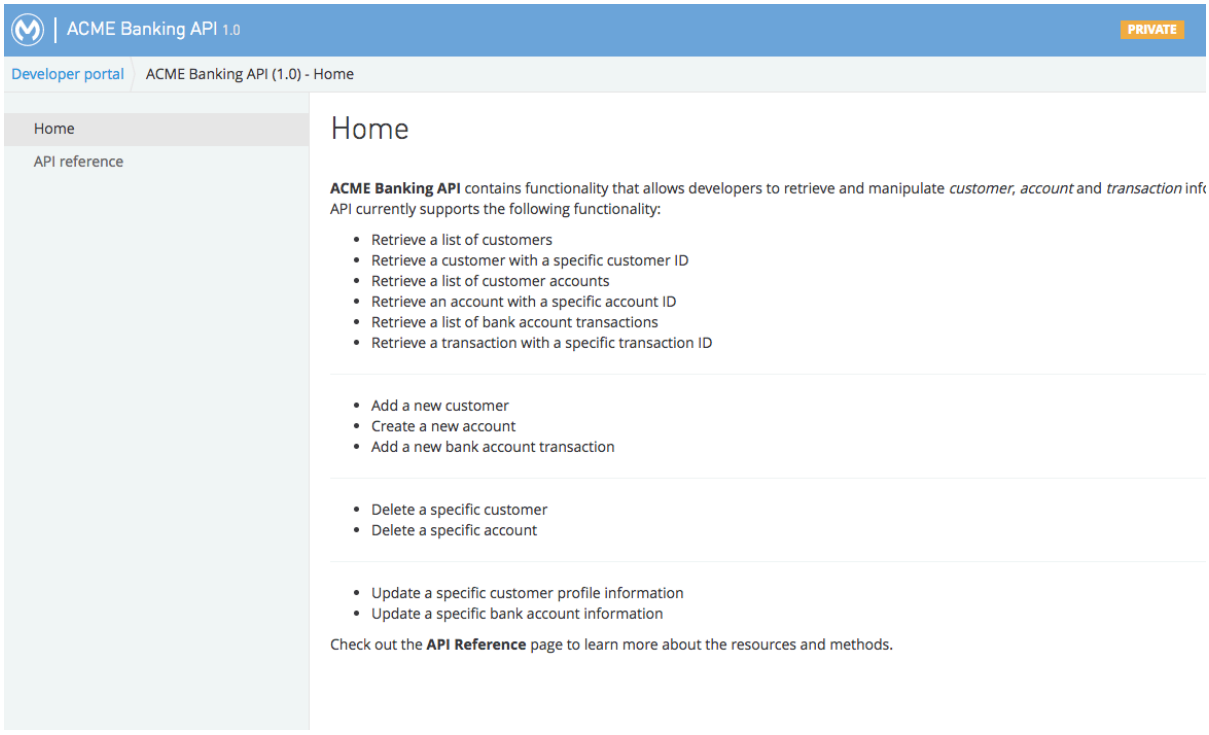
    Note: Anypoint Platform will not publish a page with zero content. Once you click the eye icon, it should turn green for all the selected pages.



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



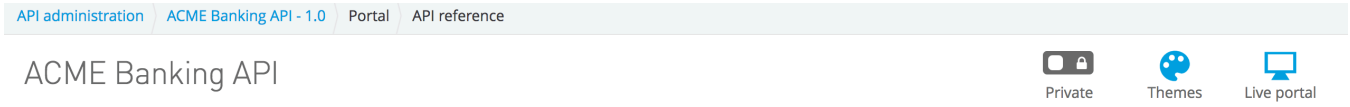
21. Examine the generated API Portal.



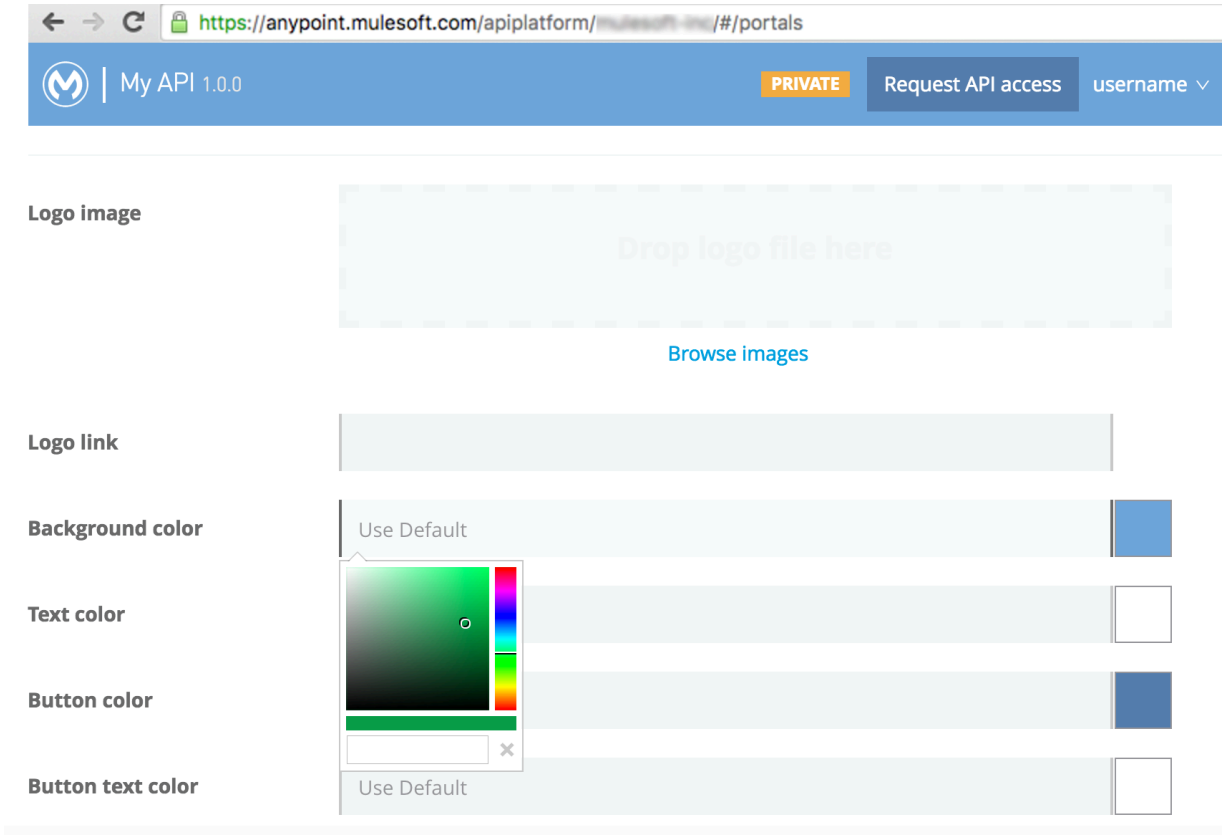
22. Close the tab with containing the Live Portal and return to the tab with the API Portal Designer.

Define a color scheme for the API Portal

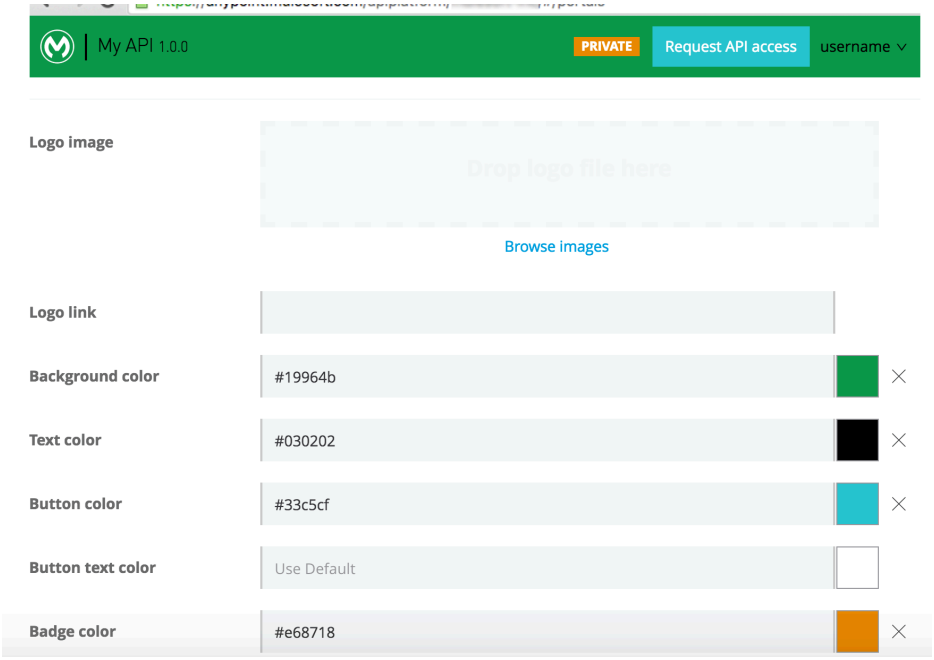
23. In the right corner of the API Portal Designer, click Themes.



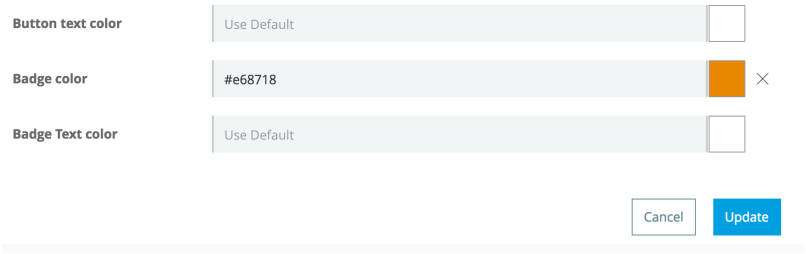
24. In the API portal theme settings dialog box, go to the Background color option and click the text area that mentions Use Default.
25. Select a color from the drop-down graph or type a color's hex value.



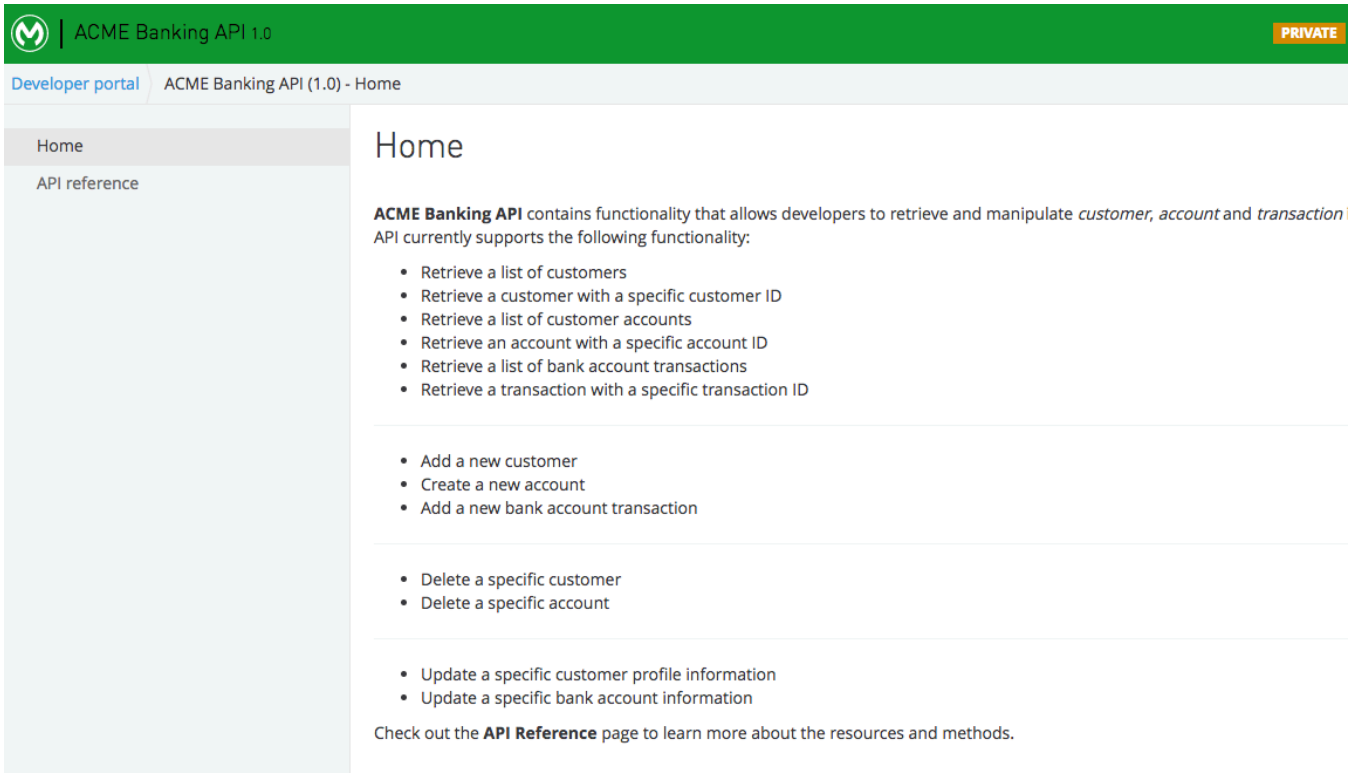
26. Change the text color, button color and button text color using the same approach of clicking over the Use Default text box and picking a desired color.



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.

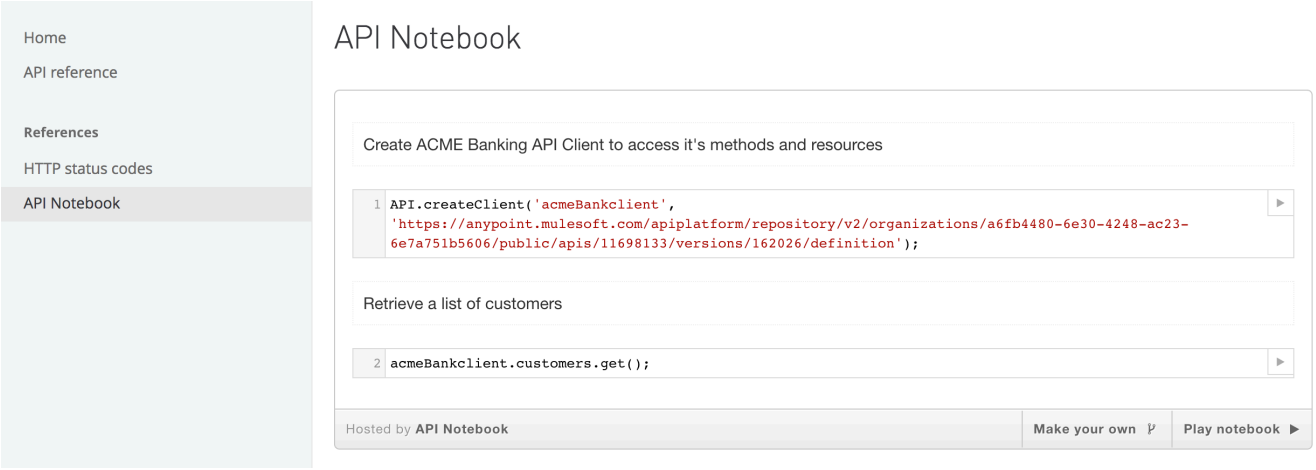


29. Close the Live Portal and return to the API Portal Designer.

Walkthrough 8-4: Create a sample use case with API Notebook inside the API Portal

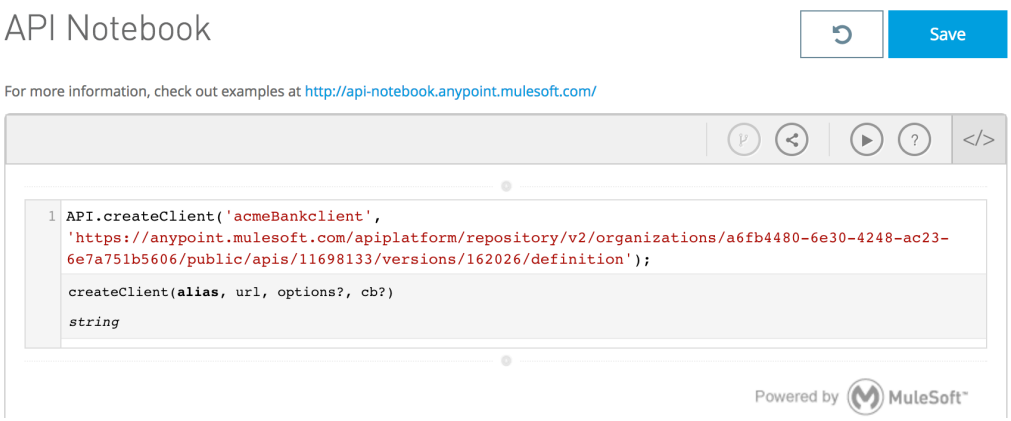
In this walkthrough, you create an API Notebook inside the API Portal. You will:

- 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.



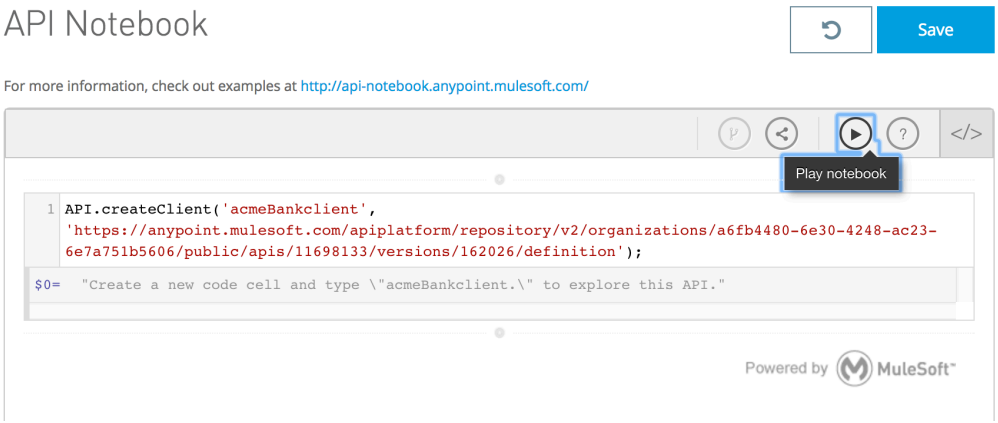
Add an API Notebook in the API Portal Designer

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’.



4. Click the Play notebook button to play the Notebook.
5. Verify you get this message below the code cell:

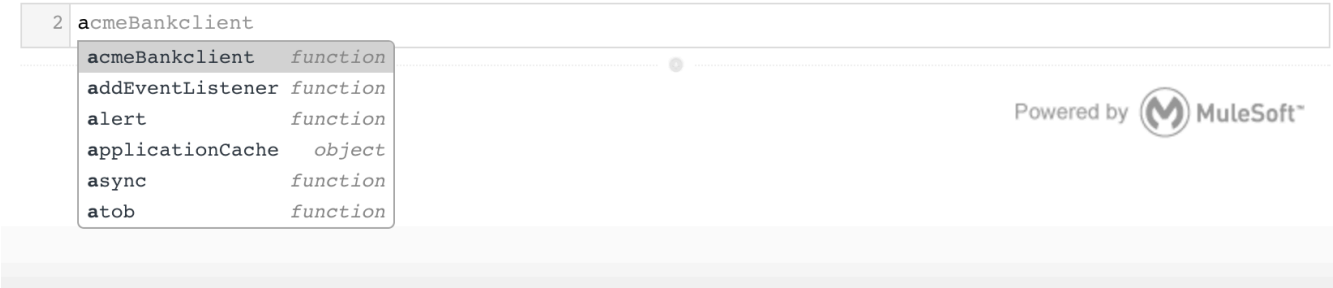
   “Create a new node cell and type \ “acmeBankClient.\” to explore this API.”



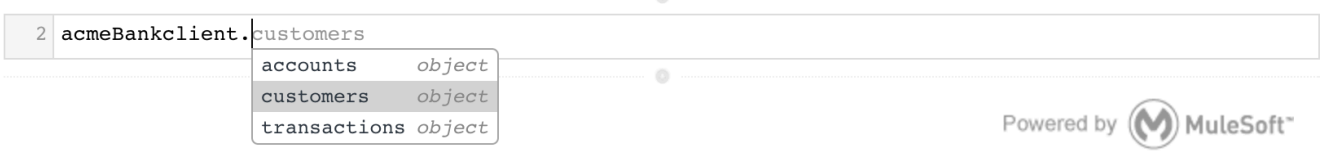
Insert a code cell to explore the ACME Banking API

6. Hover over the small circle that is in the middle underneath the createClient code cell.

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



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



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();
```

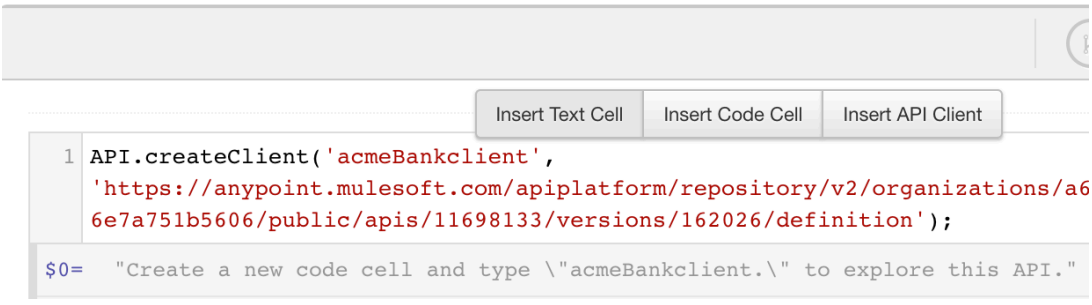
12. Press enter to execute the code cell.
13. Verify that you see the status code 200 returned in the result below the code cell.

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

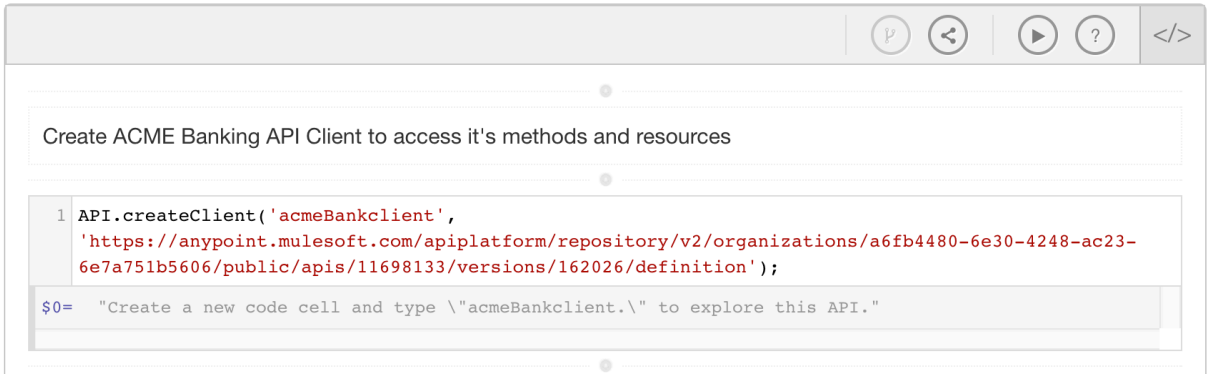
14. Click the menu icon in the right corner of code cell 3 and click Delete from the drop-down menu.

Insert text cells to make the API Notebook readable

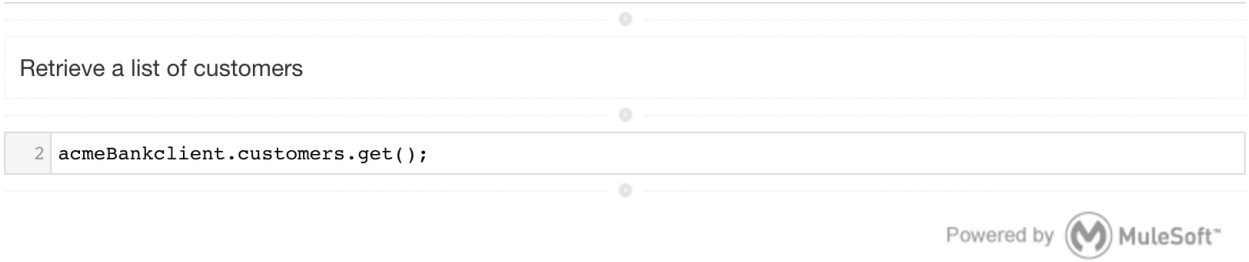
15. Locate the circle in the middle above code cell 1 and hover over it.
16. Click Insert Text cell.



17. Click inside the new text cell and type Create ACME Banking API client to access its methods and resources.



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.



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"
```

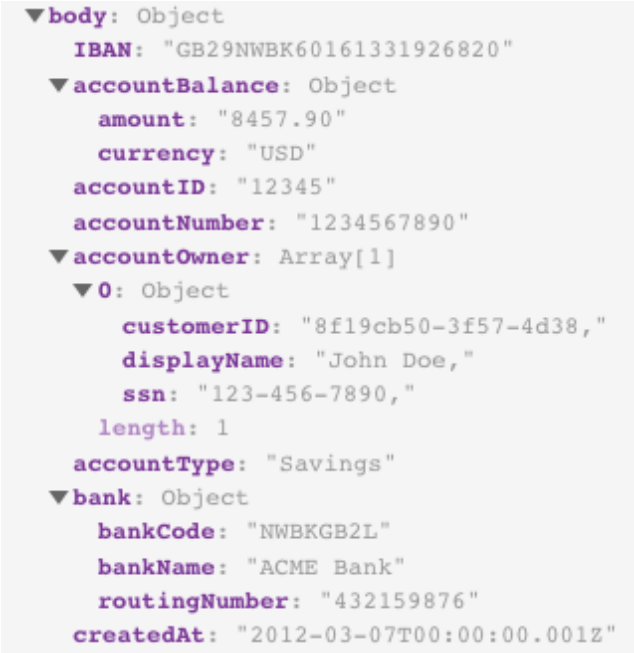
Perform string manipulation on the datatypes

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.

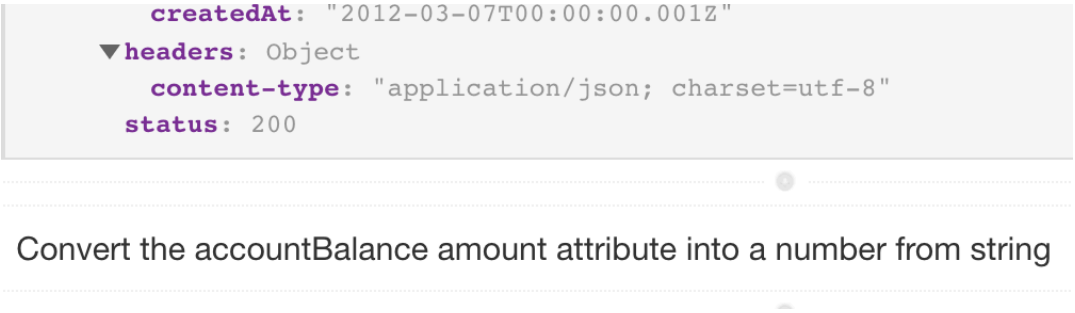
```
6 accountA=acmeBankclient.accounts.account_id("12345").get();
```

36. In the response returned, click the arrow before body to observe the structure of the account information.

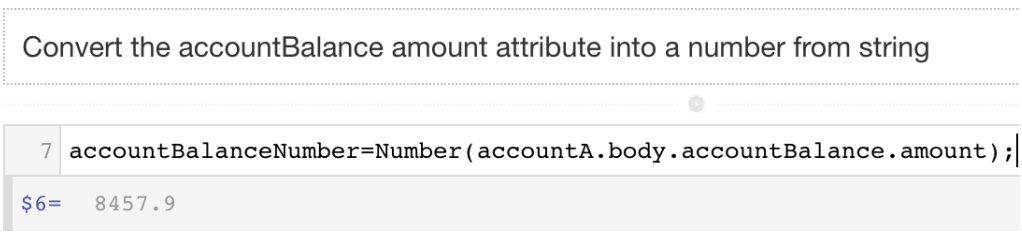
37. Expand the accountBalance, accountOwner, and the bank objects to see the underlying values.



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.



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.



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(" "+
   accountA.body.accountBalance.amount);
16 }
17 concatAccountBalance(accountA);
```

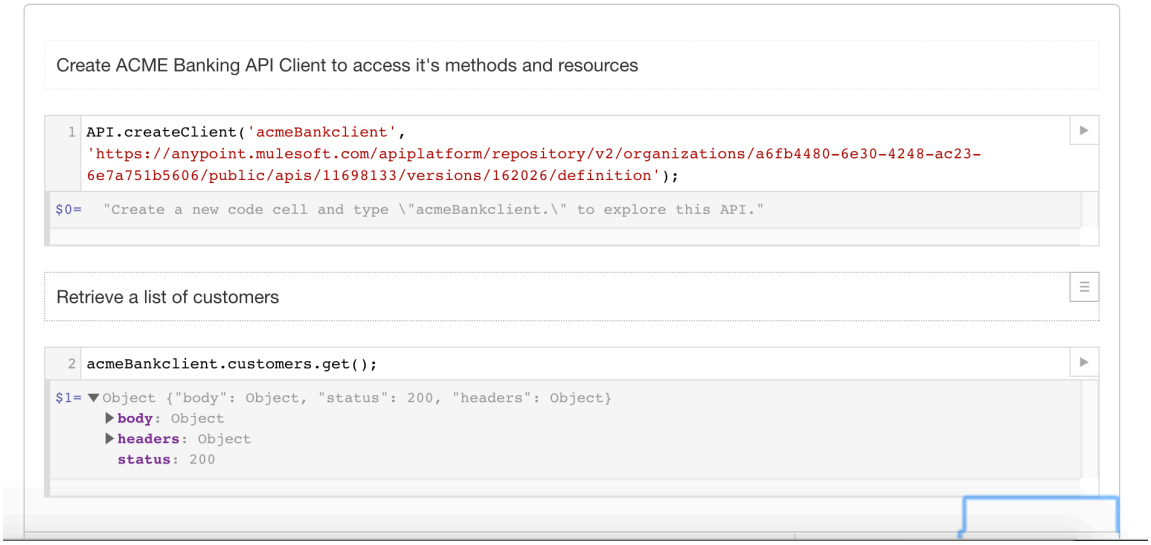
```
$8=   "The accountBalance is USD 8457.90"
```

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



58. Return to the API Portal and close it.

Module 9: Reusing Patterns

```
#%RAML 1.0 ResourceType
post?:
  description: Add a new <<resourcePathName | !singularize>>
  displayName: Add new <<resourcePathName | !singularize>>
  body:
    type: <<reso
  responses:
```

```
  get:
    description: Retrieve a list of customers
    is:
      - cacheable
      - hasAcceptHeader:
          customErrorDataType: CustomErrorMessage
    responses:
      200:
```

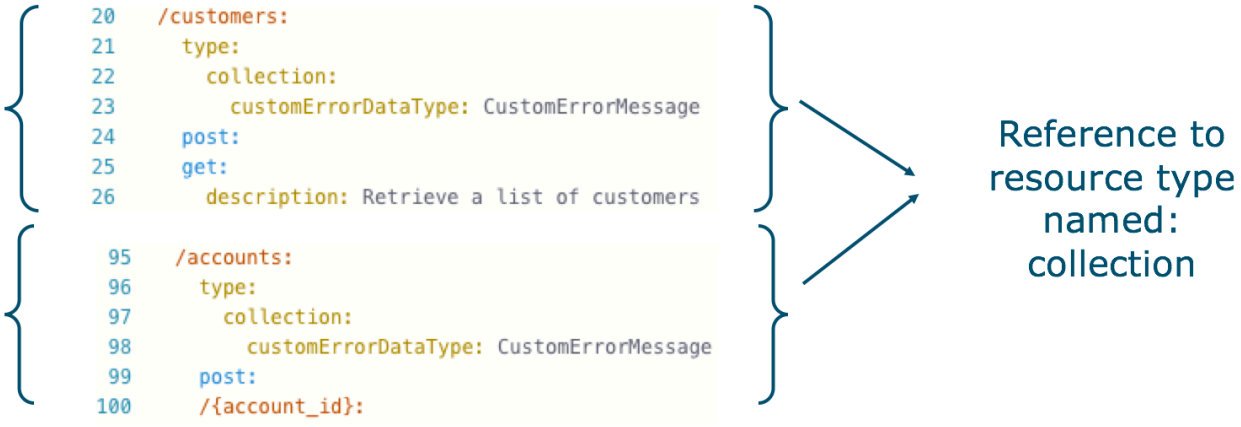
Objectives:

- 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.

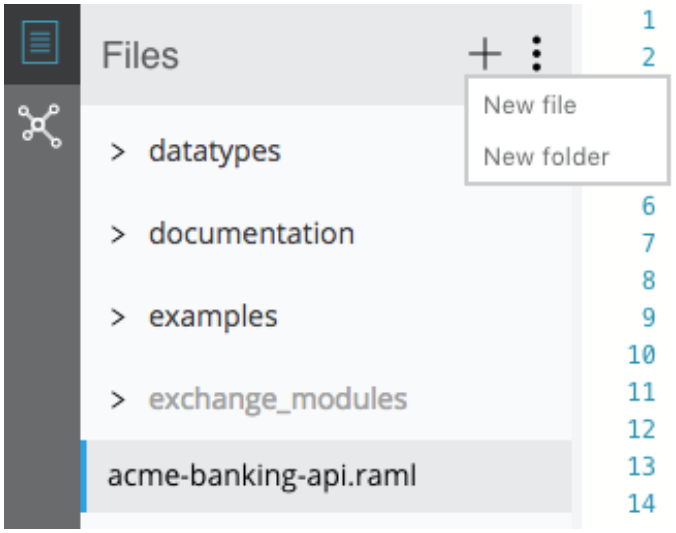


Define a collection resource type fragment

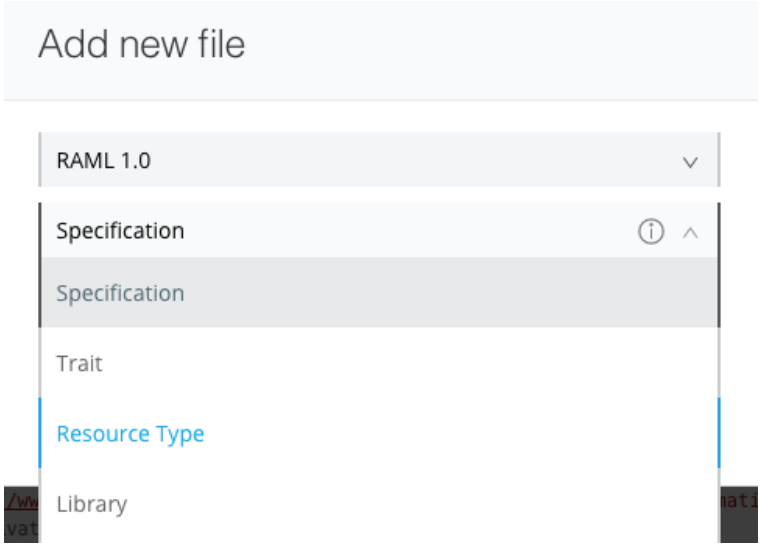
1. In the Anypoint Platform webpage, click the top-left menu icon.
2. Select Design Center.
3. In the Design Center projects page, click ACME Banking API.

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 | !uppercamelcase>>
```

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.

```
#%RAML 1.0 ResourceType
post?:
  description: Add a new <<resourcePathName | !singularize>>
  displayName: Add new <<resourcePathName | !singularize>>
  body:
    type: <<resourcePathName | !singularize | !uppercamelcase>>
  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:
```

```
resourceTypes:

/customers:
  get:
    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.

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

/customers:
  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}:
```

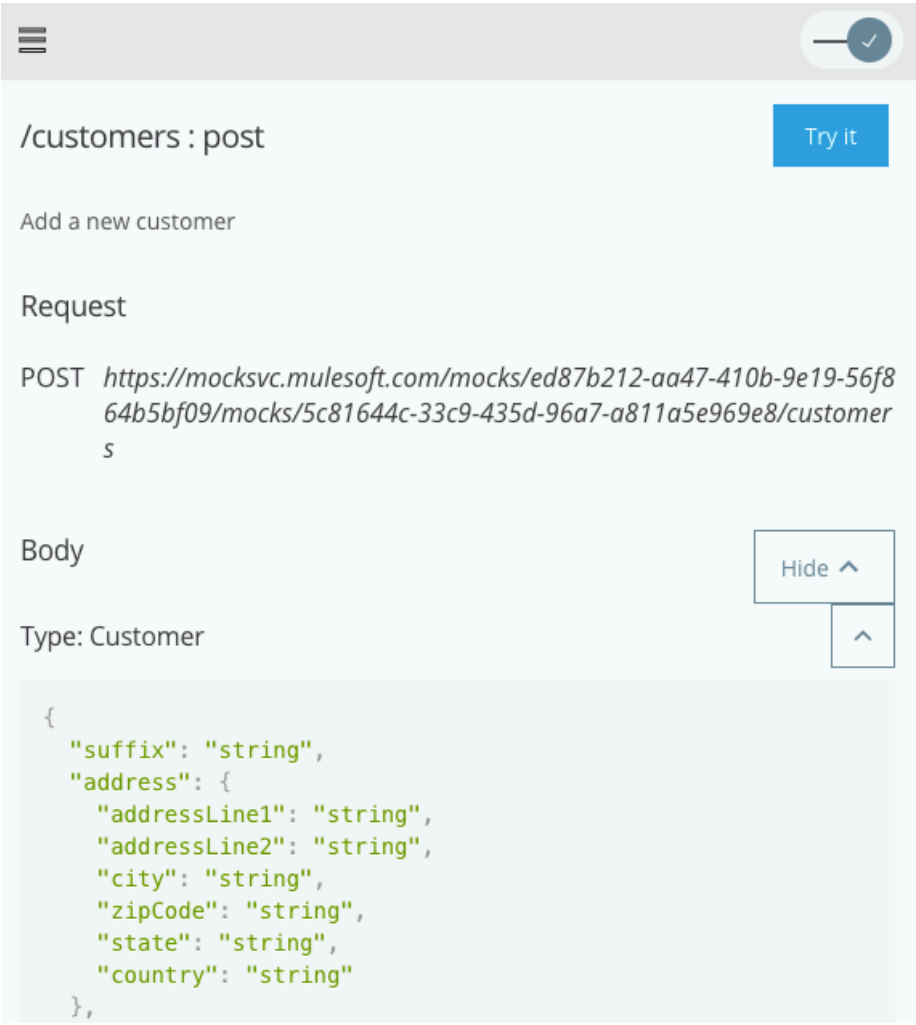
34. Similarly, in the /transactions main resource, delete the post method definition and add a reference to the collection resource type.

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

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.



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?:
```

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 >>
```

```
#%RAML 1.0 ResourceType
get?:
  description: Retrieve a <<resourcePathName | !singularize>> with a specific <<resourcePathName | !singularize>>
  responses:
    200:
      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 | !uppercamelcase>>.
19. In the next line, replace the value of the example node with <<exampleValue>>.

```
responses:
  200:
    body:
      type: <<resourcePathName | !singularize | !uppercamelcase>>
      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 | !uppercamelcase>>
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:
```

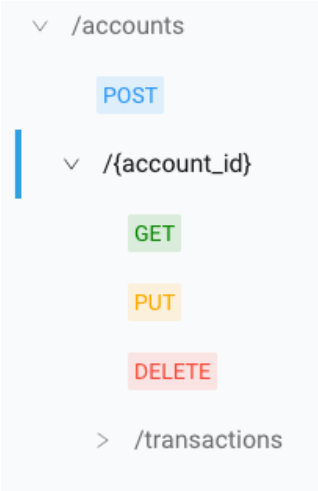
```
82      /{account_id}:
83        type:
84          member:
85            exampleValue: !include examples/AccountExample.raml
86            customErrorDataType: CustomErrorMessage
87        get:
88        put:
89        delete:
```

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           customErrorDataType: CustomErrorMessage
110       get:
```

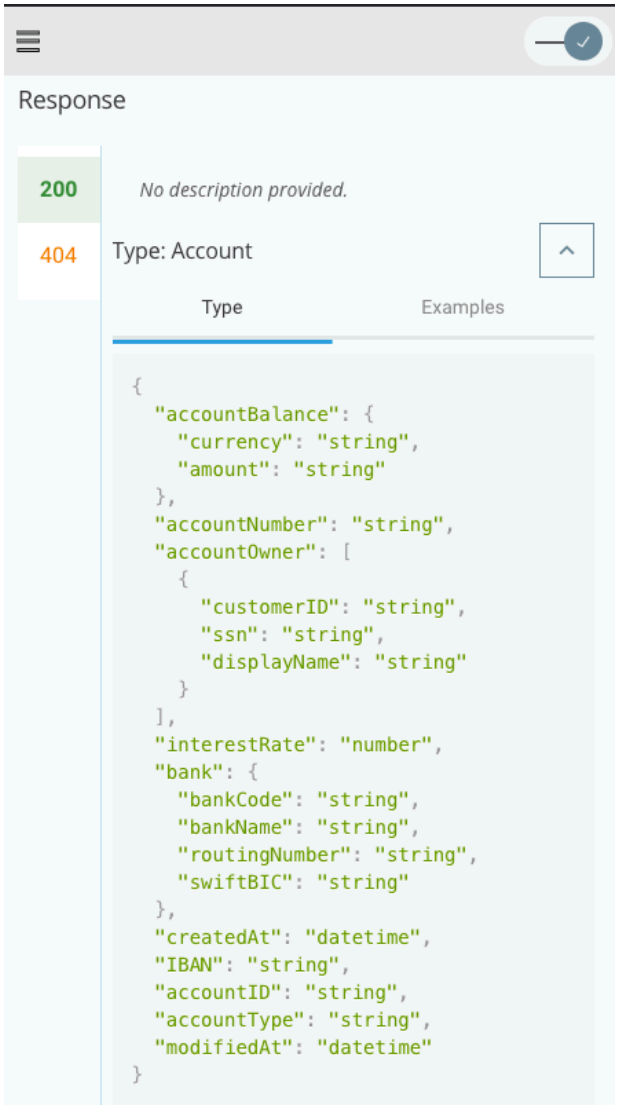
View the API Console

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.

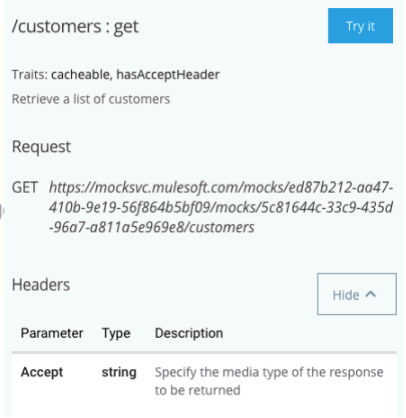


Walkthrough 9-3: Define and use various traits for resources and methods

In this walkthrough, you refactor the API definition by creating reusable traits for resources and methods. You will:

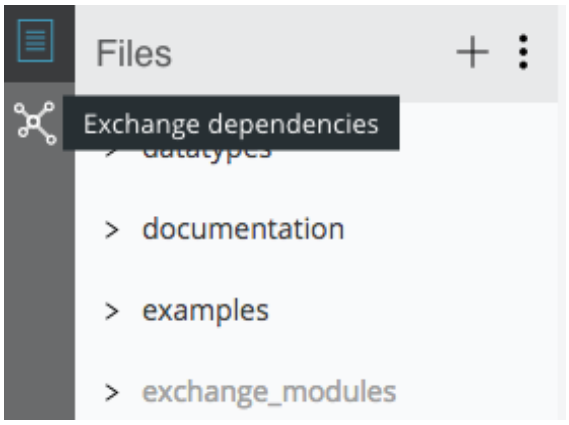
- Consume a cacheable trait asset from Anypoint Exchange.
- Define a flexible content type trait to be applied to the resource method with an Accept header in the request.
- Refactor the resource methods to use these traits.

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



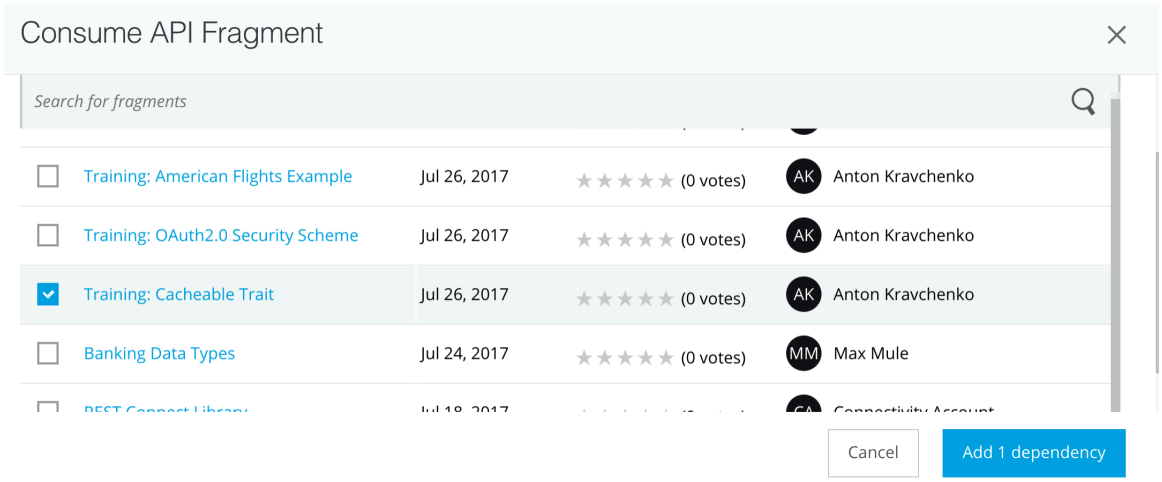
Consume a cacheable trait asset from Anypoint Exchange 2.0

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.

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



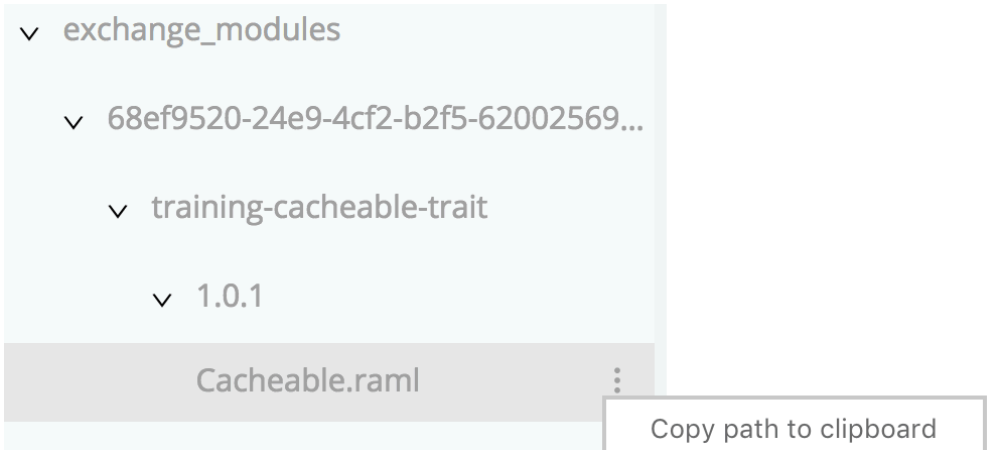
5. Click Add Dependency.

Reference the cacheable trait inside a RAML API Specification

6. In the file browser section, click the files menu icon.
7. In the file browser section, click acme-banking-api.raml ;if the file is not open in the RAML editor.
8. Locate the line that includes the member resource type file and add two new lines below it.
9. Remove indentation in the second new line created.
10. From the shelf, click traits.
11. In the new line, type:

```
cacheable: !include
```

12. In the file browser section, expand the exchange_modules folder to locate the Cacheable.raml trait file.
13. 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.

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

31. In the file browser, click acme-banking-api.raml.
32. In the /customers get method responses, copy the HTTP 406 response lines.

```
        404:
          body:
            type: CustomErrorMessage
        406:
          body:
            type: CustomErrorMessage
    /{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.

```
    406:
      body:
        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:

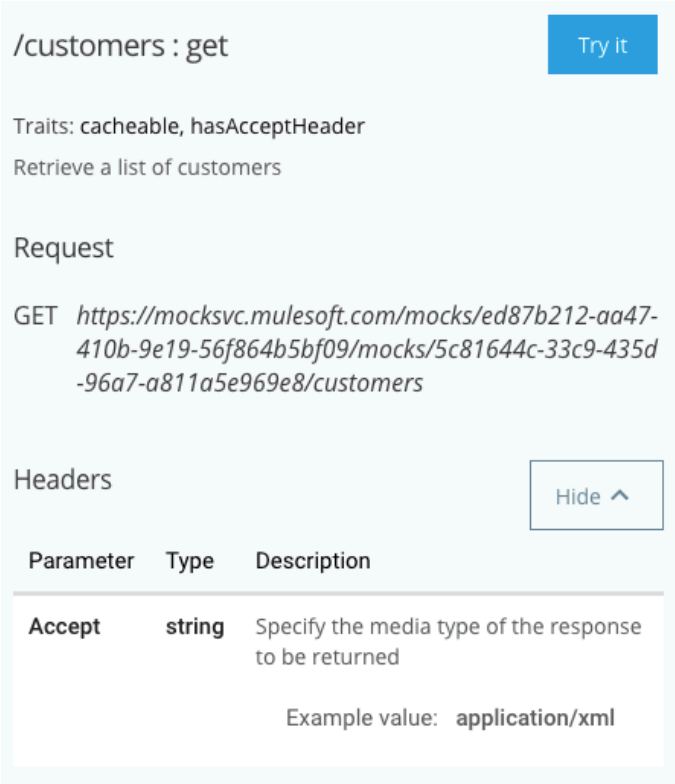
```
is:
 - cacheable
 - hasAcceptHeader:
     customErrorDataType: CustomErrorMessage
```

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

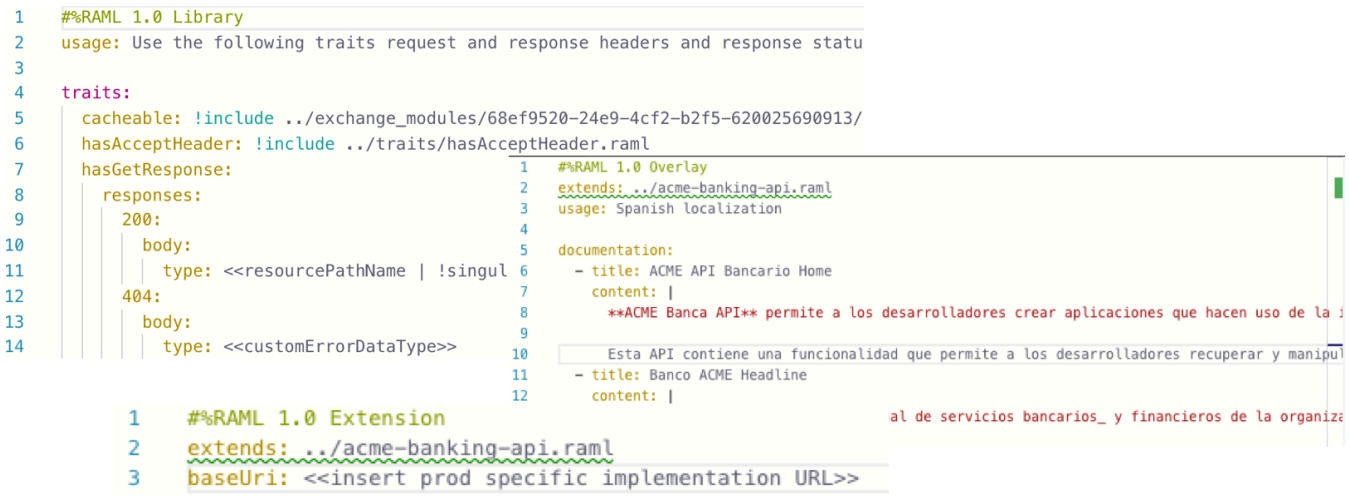
View the API Console

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.



Module 10: Modularizing APIs



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.

```
#%RAML 1.0 Library
usage: Use the following traits request and response headers and response statu

traits:
  cacheable: !include ../exchange_modules/68ef9520-24e9-4cf2-b2f5-620025690913/
  hasAcceptHeader: !include ../traits/hasAcceptHeader.raml
  hasGetResponse:
    responses:
      200:
        body:
          type: <<resourcePathName | !singu
      404:
        body:
          type: <<customErrorDataType>>
```

```
      is:
        - Traits.cacheable
        - Traits.hasAcceptHeader:
            customErrorDataType: CustomErrorMessage
      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
    ```

```
#%RAML 1.0 Library
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 | !uppercamelcase>>[]
    ```

```
6           200:
7             body:
8               type: <<resourcePathName | !singularize | !uppercamelcase>>[]
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.

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
```

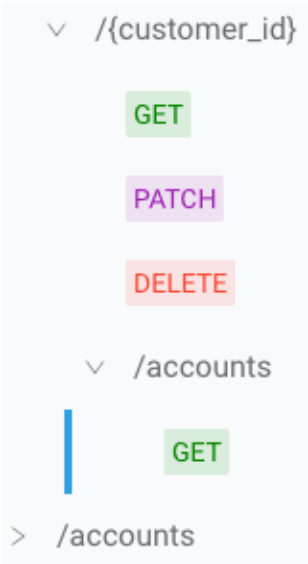
31. Similarly, in the /transactions nested resource get method, delete the lines that define the get method response and apply the hasGetResponse trait.

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

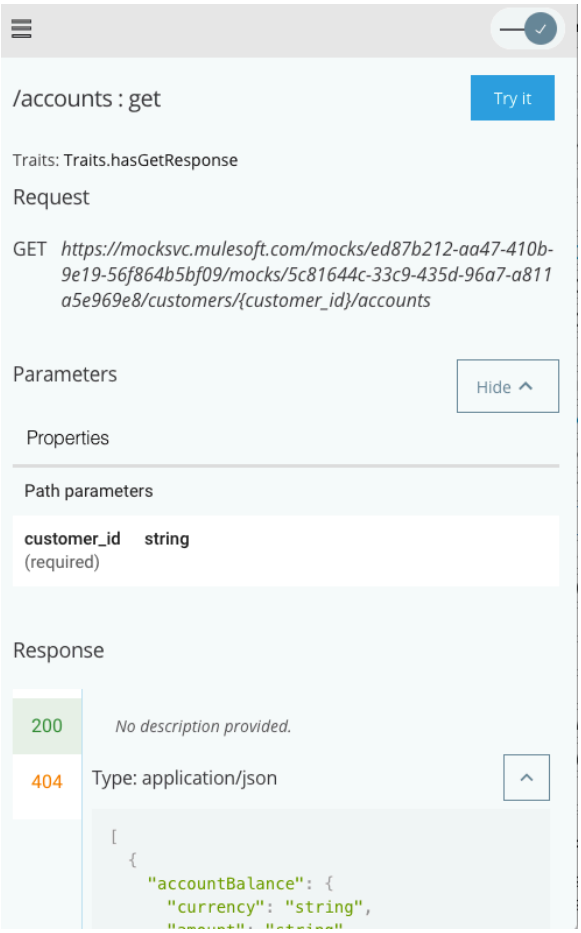
View the API Console

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.



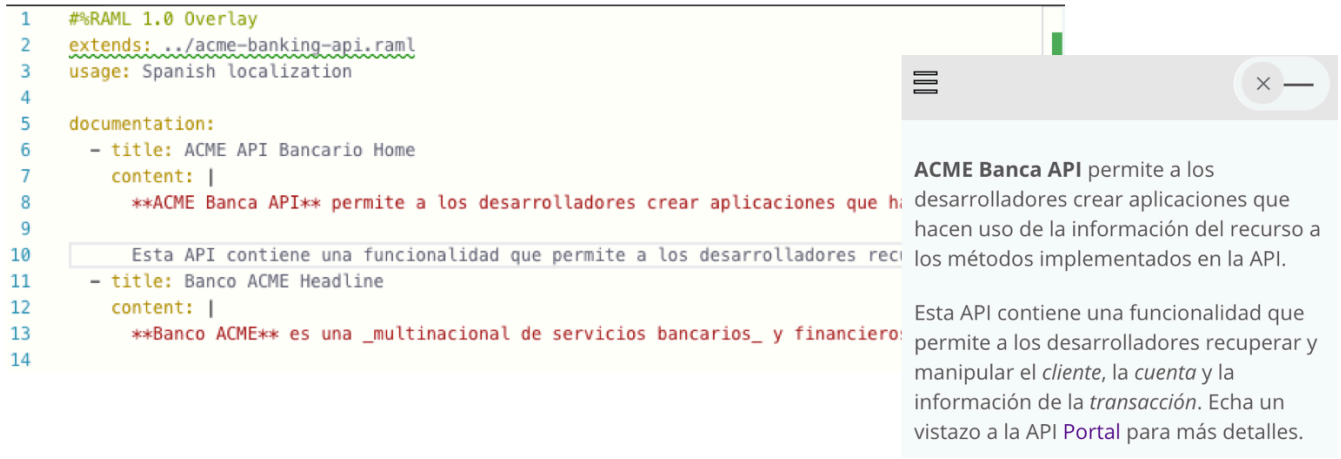
34. Verify that the hasGetResponse trait has been applied to the resource.



Walkthrough 10-2: Internationalize documentation and resource description using an overlay

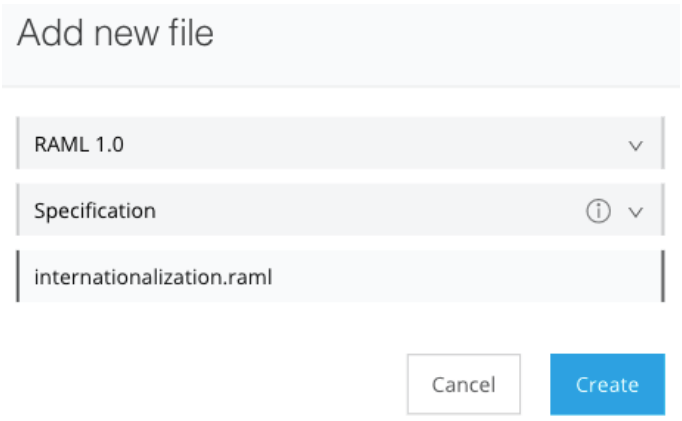
In this walkthrough, you internationalize the ACME Banking API using overlays. You will:

- Create a RAML overlay fragment file.
- Add documentation and resource method description nodes in the overlay file to support Spanish localization.



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.



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.

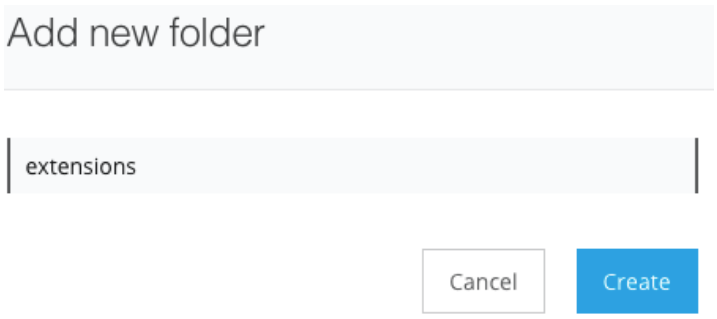
Walkthrough 10-3: Define and use API extensions to promote portability to test in multiple environments

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.

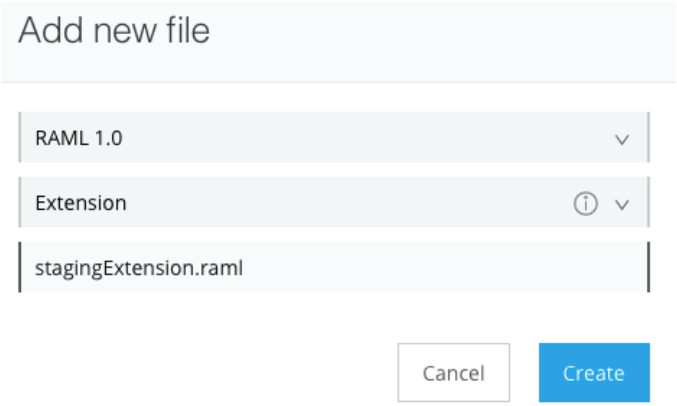
Create API extension files

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.

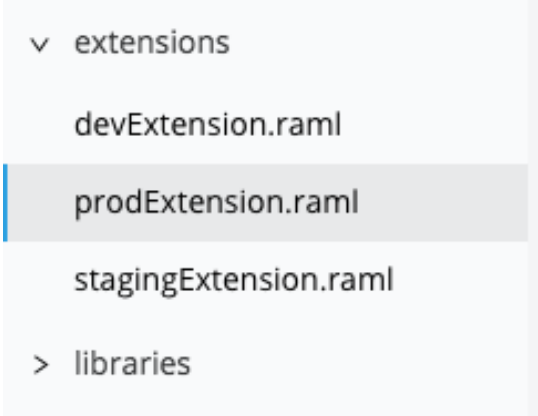


4. Click the + icon next to the extensions folder and select New file.

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

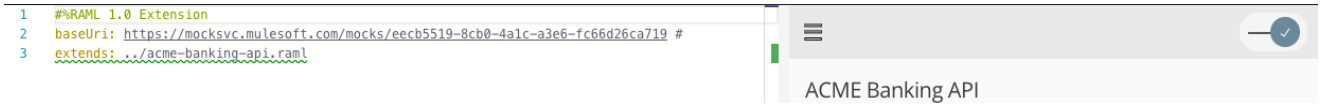
7. In the file browser, locate the extensions folder and select stagingExtension.raml.
8. Go to the line that contains the extends node and type the value as:

```
../acme-banking-api.raml
```

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

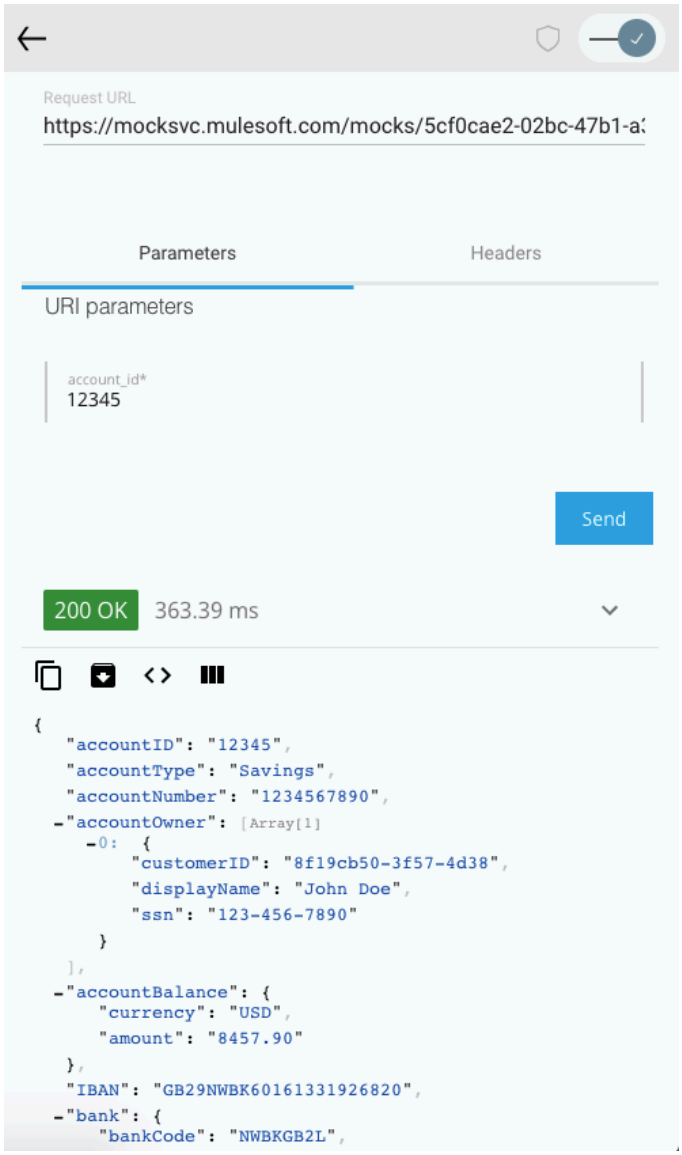
9. In API Console, turn on the mocking service at the top right corner.

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



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.



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:

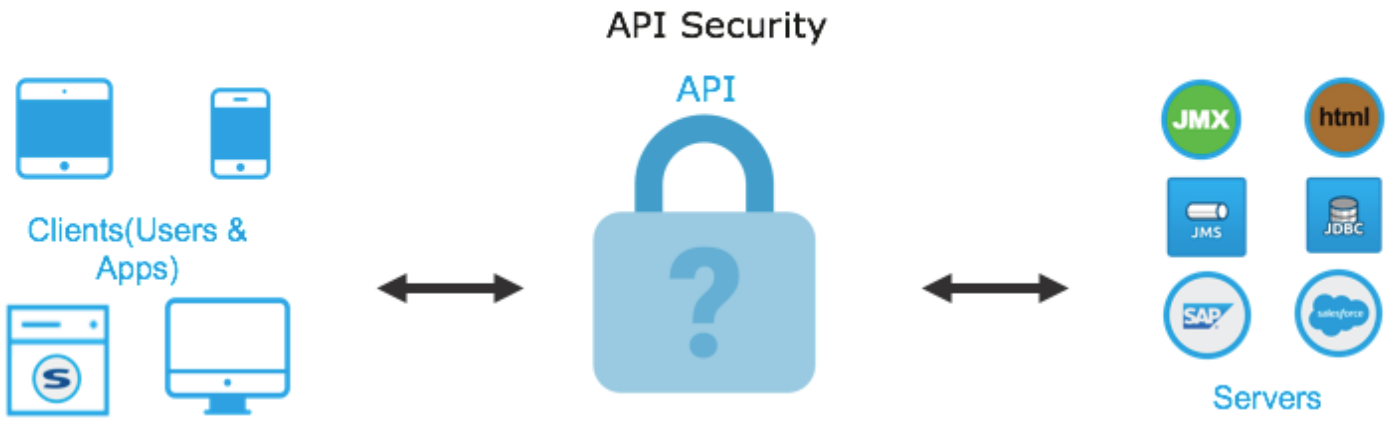
```
baseUri: <<insert dev environment specific implementation URL>>
```

```
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:

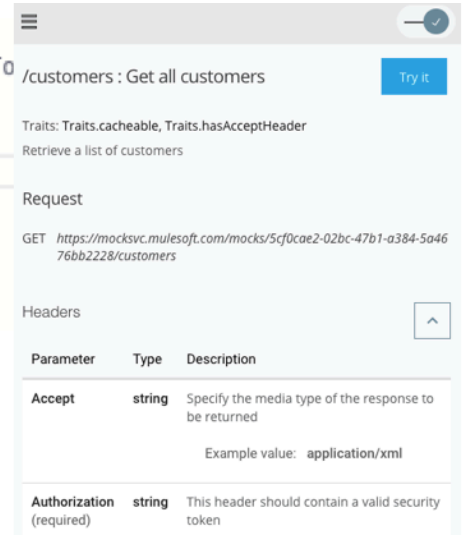
- Define API security requirements.
- Use security schemes to apply resource and method level policies.
- Define custom security scheme for APIs.
- Apply an OAuth2.0 external provider policy to resource methods.

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:

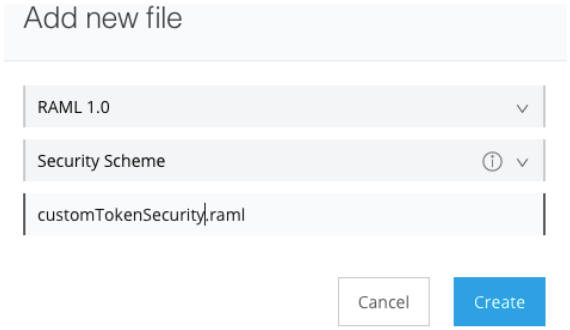
- Create a custom security scheme file.
- Reference the custom security scheme in the main RAML API definition.
- Apply the security scheme to certain resource methods.

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



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.

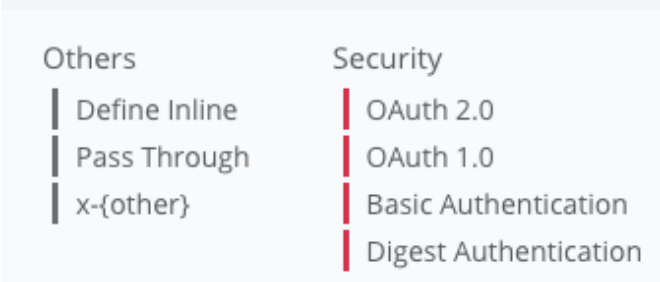


7. Click Create.

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:
```



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
```

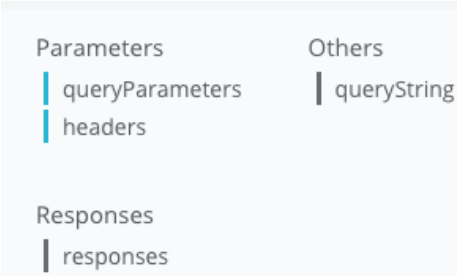
14. Press enter to add a new line.

```
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.

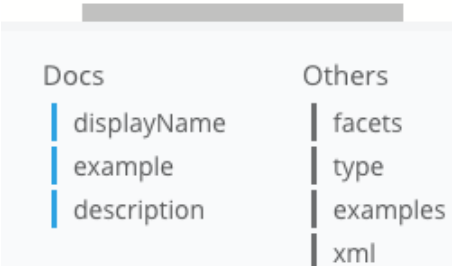
16. In the shelf, click headers.

```
#%RAML 1.0 SecurityScheme
type: x-customToken
description: This security scheme
describedBy:
```



17. In the shelf, click Authorization.
18. In the shelf, click description.

```
describedBy:
  headers:
    Authorization:
```

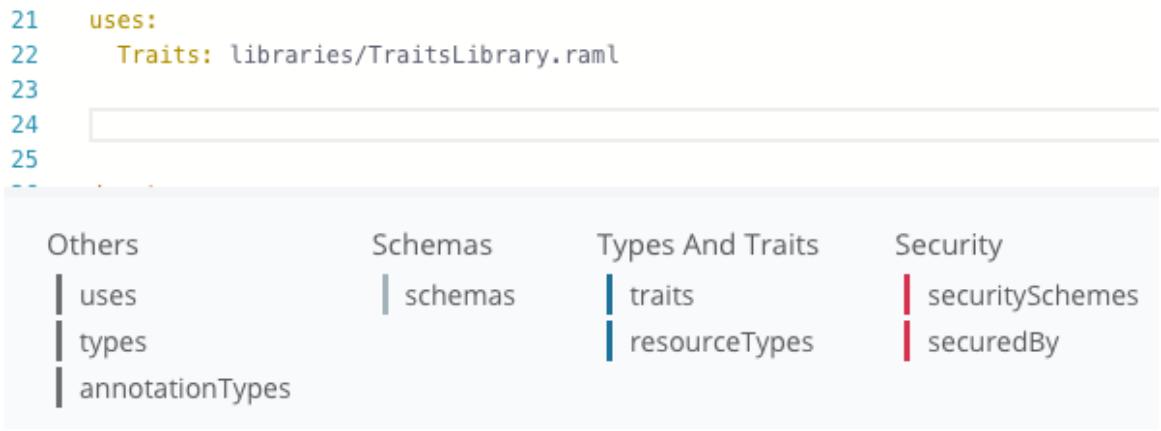


19. Type the value of the description node as: This header should contain a valid security token.
20. Press enter to add a new line.
21. In the shelf, click type.
22. In the shelf, click string.

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.



28. In the new line, type:

```
customTokenSecurity: !include securitySchemes/customTokenSecurity.raml
```

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.

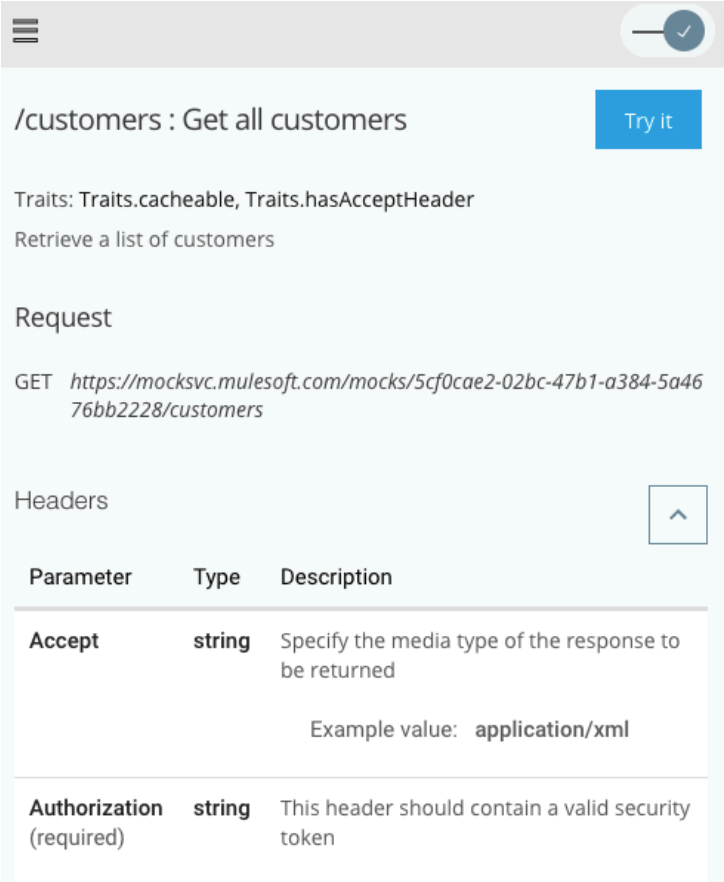
31. In the shelf, click customTokenSecurity.

32. Press enter to add a new line.

```
securitySchemes:
  customTokenSecurity: !include securitySchemes/customTokenSecurity.raml

securedBy: customTokenSecurity

/customers:
  type:
    collection:
      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.



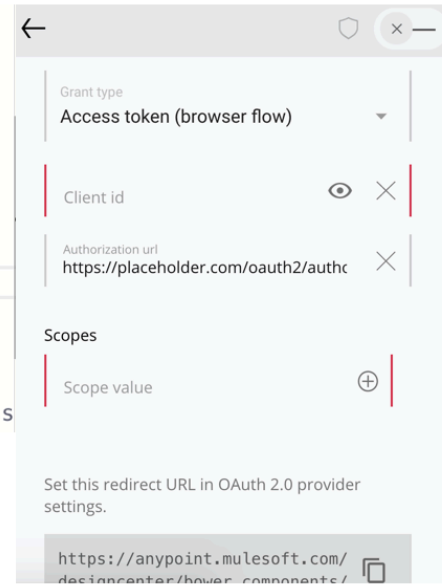
Note: Go to any other resource method and notice the Authorization header added to all the resource methods by adding to the root of the RAML definition. Custom security schemes like the customTokenSecurity are not supported for testing using the Try It option.

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:

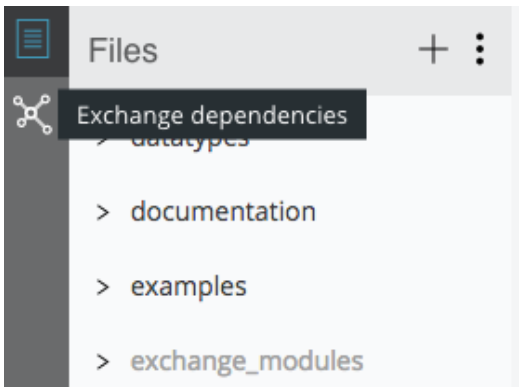
- Consume an OAuth2.0 security scheme fragment file.
- Reference the OAuth2.0 security scheme in the RAML API definition.
- Apply the security scheme in the API resource methods.

```
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
```



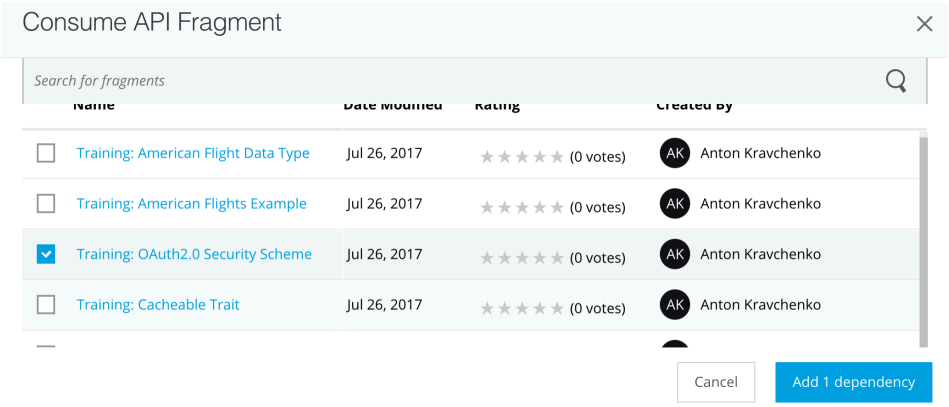
Consume an OAuth2.0 security scheme fragment file

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.

4. In the Consume API Fragment dialog box, locate Training: OAuth2.0 Security Scheme and check the box.



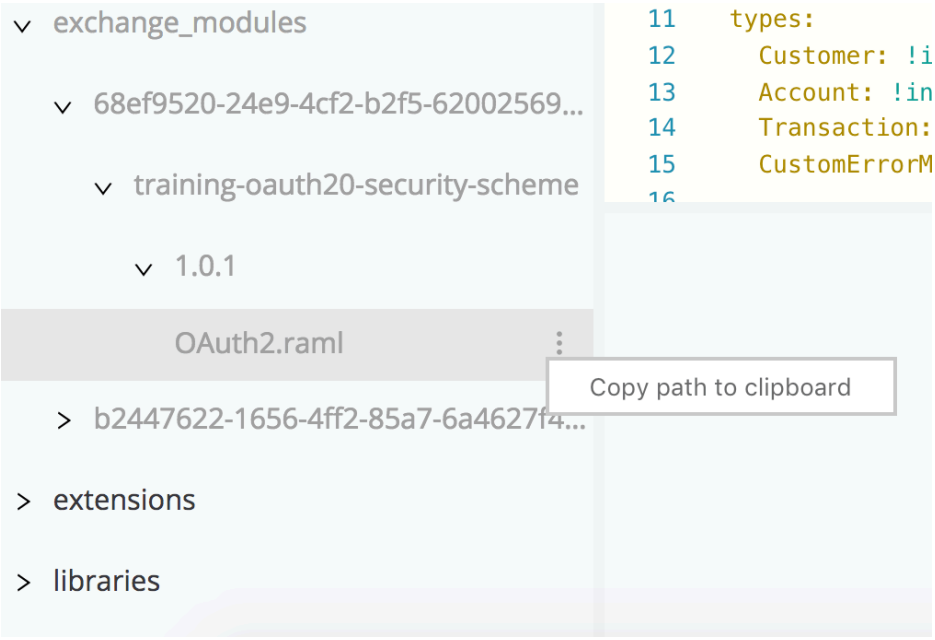
5. Click Add Dependency.

Reference the OAuth 2.0 security scheme inside a RAML API Specification

6. In the file browser section, click acme-banking-api.raml ;if the file is not open in the RAML editor.
7. Locate the customTokenSecurity include statement and add a new line below it.
8. In the new line, type:

```
oauth2_0: !include
```

9. In the file browser section, expand the exchange_modules folder to locate the OAuth2.raml security scheme file.
10. Click the menu icon next to the file name and click Copy path to clipboard.



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

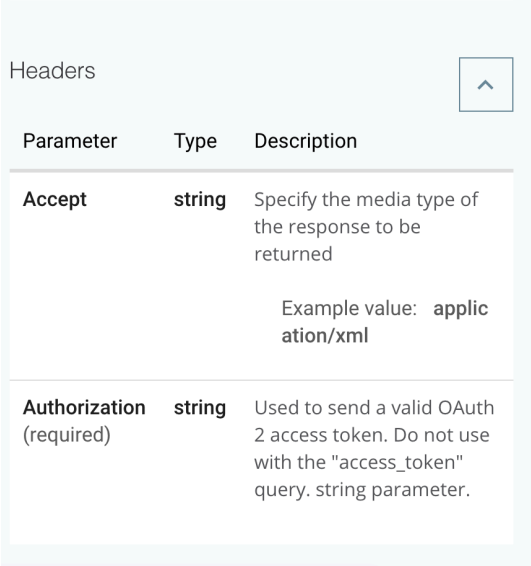
    Note: If you imported the OAuth2.raml file from the studentFiles folder into a traits folder, include the path as securitySchemes/OAuth2.raml.

Secure the resource methods that update bank customer and account information with the OAuth2.0 security scheme

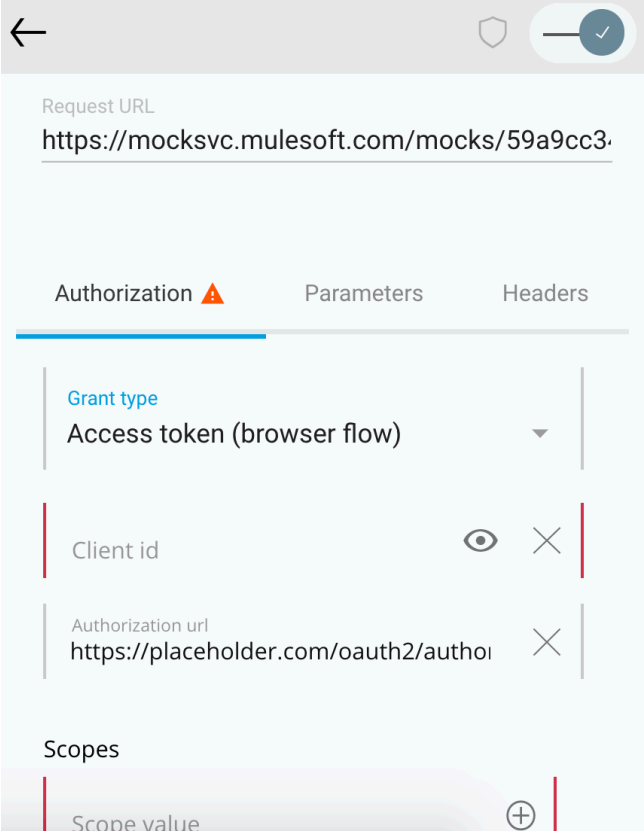
12. In the /{customer_id} nested resource patch method, add a new line below the line that contains the displayName node.
13. In the shelf, click securedBy.

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.

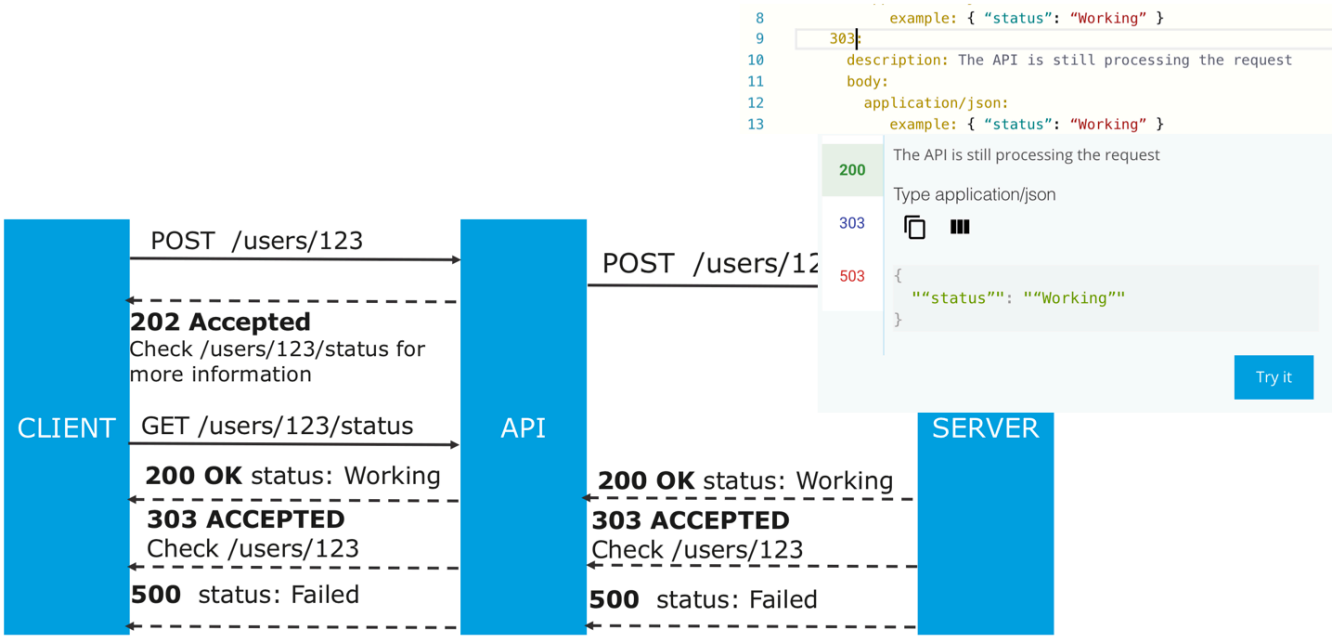
17. Scroll to the Request Headers section and verify that you can see an Authorization header.



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.



Module 12: Enhancing API Responses using Hypermedia



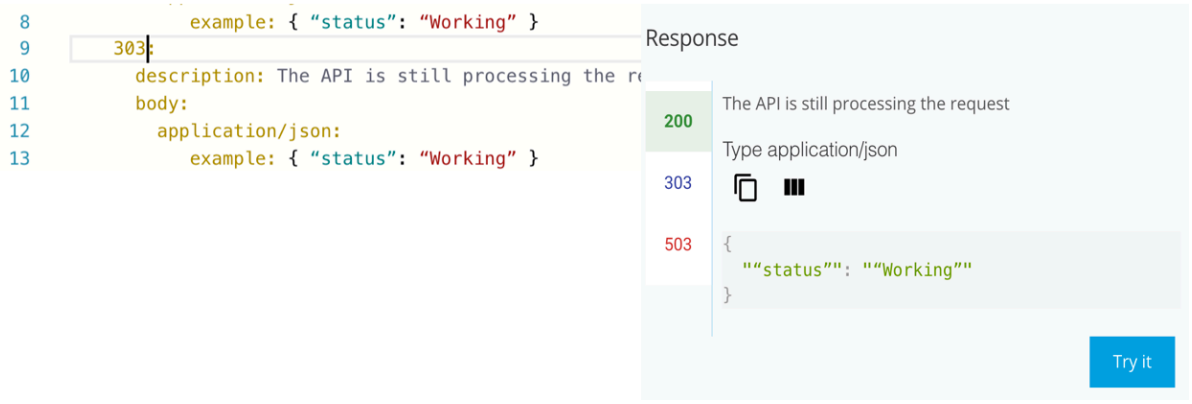
Objectives:

- 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.



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       200:
5         description: The API is still processin
6         body:
7           application/json:
8             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       303:
10        description: The API is still processing the request
11        body:
12          application/json:
13            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
```

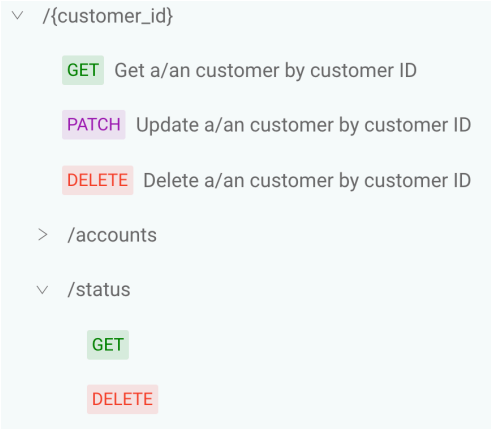
```
17    resourceTypes:
18      collection: !include resourceTypes/collection.raml
19      member: !include resourceTypes/member.raml
20      stateSpecific: !include resourceTypes/stateSpecificResourceType.ra
21
22    uses:
```

52. Add a new line below the line that contains /status nested method and press enter.
53. Press tab and type:

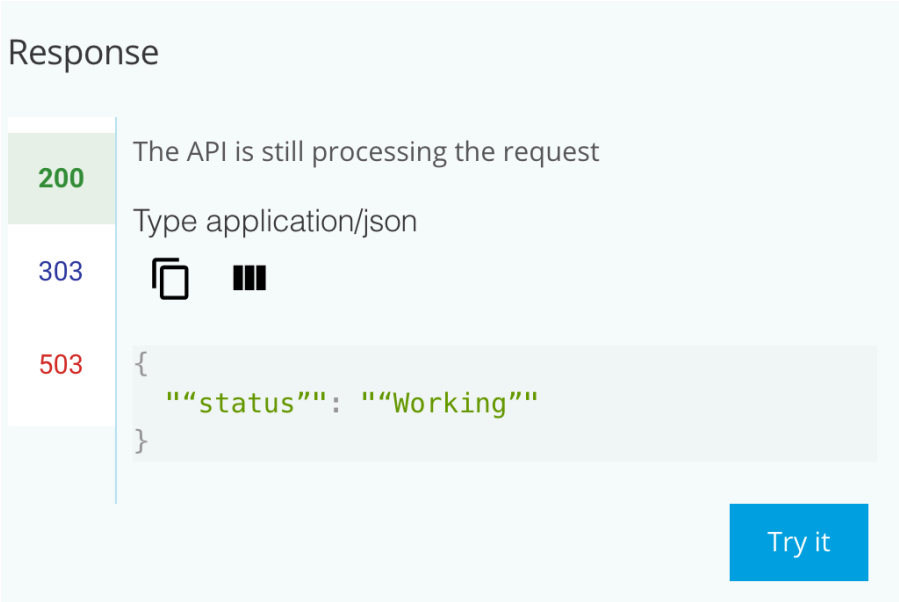
```
type: stateSpecific
```

```
67        /status:
68          type: stateSpecific
69
```

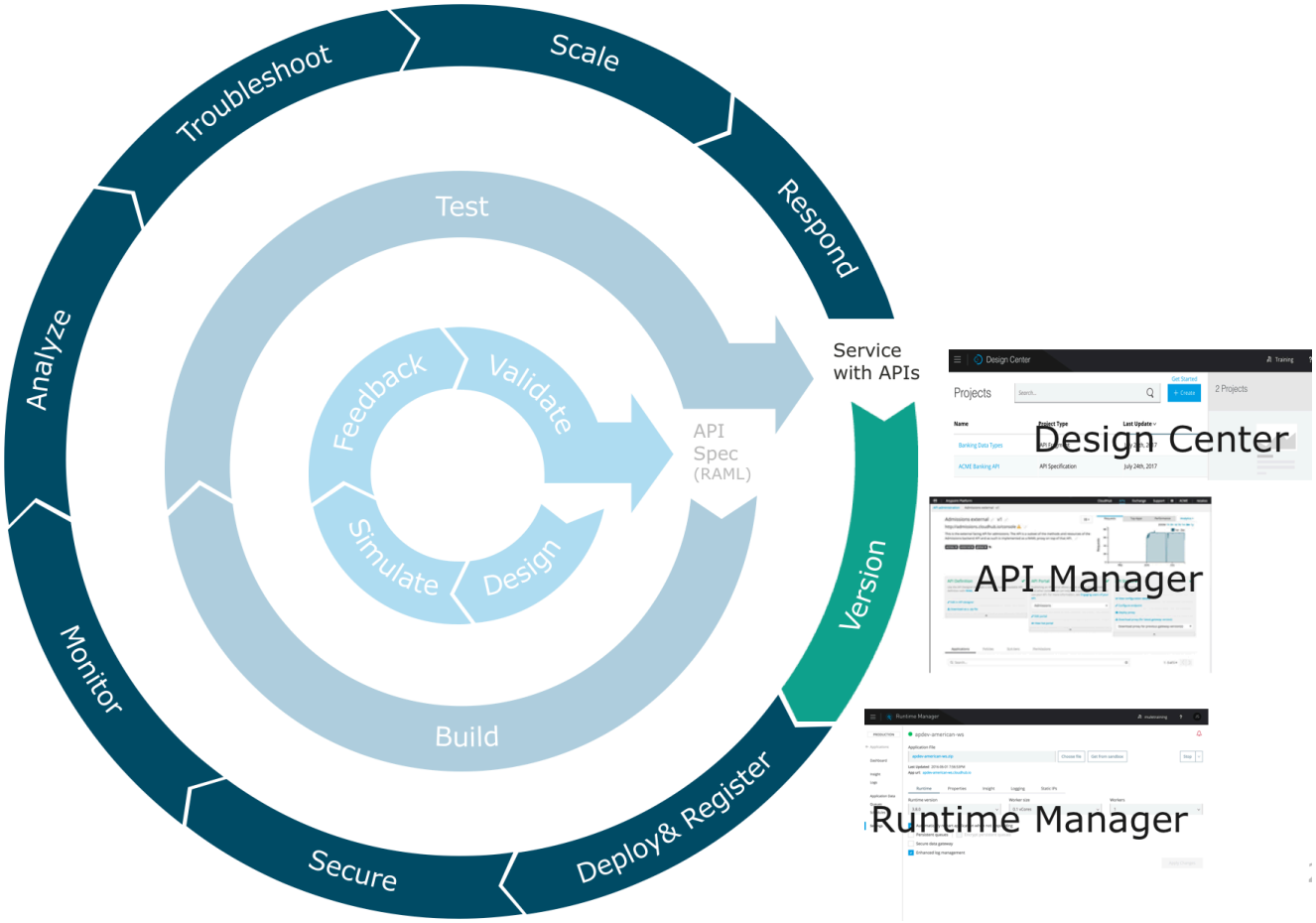
54. Copy the lines that define the status resource.
55. Add the /status resource as a nested resource for /accounts and /transactions main resource.
56. In API Console, click the top left menu icon.
57. Click the GET method link under the /customers/{customer_id}/status resource.



58. Scroll down to the response and verify that you can see the various status codes for the responses.



Module 13: Versioning APIs



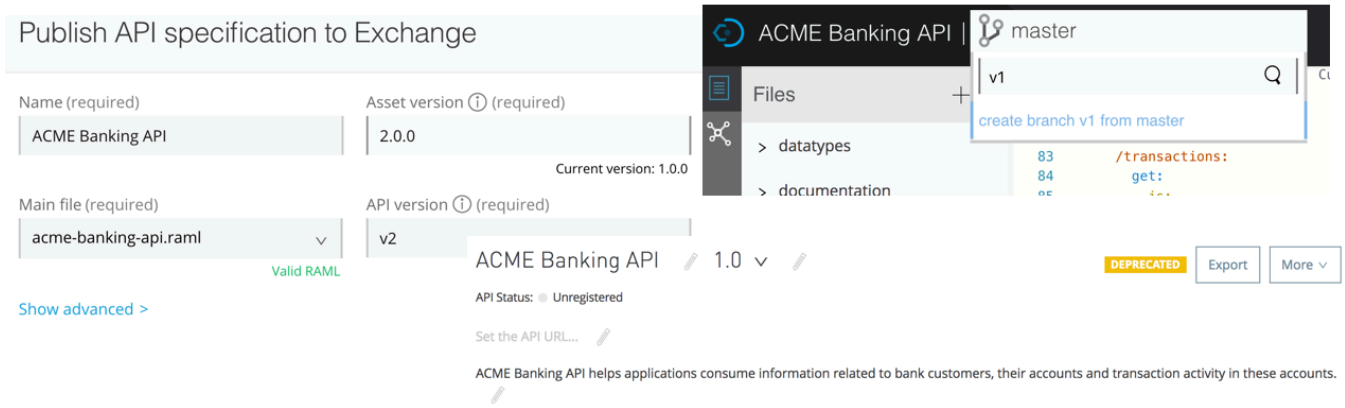
Objectives:

- 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

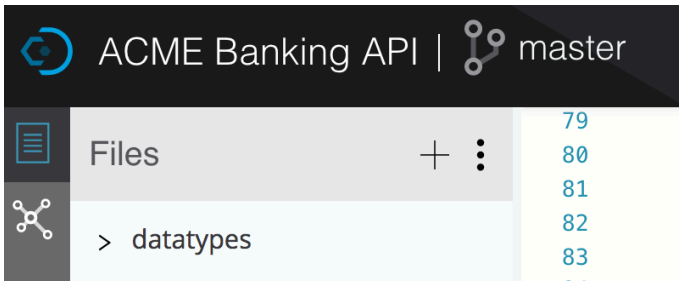
In this walkthrough, you create a new version of the ACME Banking API. You will:

- 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.

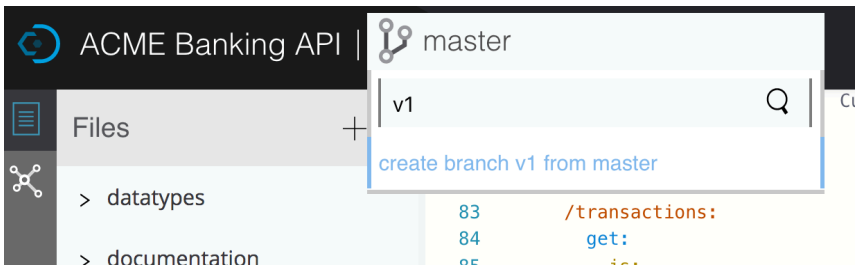


Add a new version of API definition in the API Designer

1. Return to API designer.
2. Click the branch icon on top with the active master branch next to it.



3. In the Find or create a branch search text box, type v1 and click create v1 branch from master.



Note: This way the master branch always points to the most latest version of the API being designed.

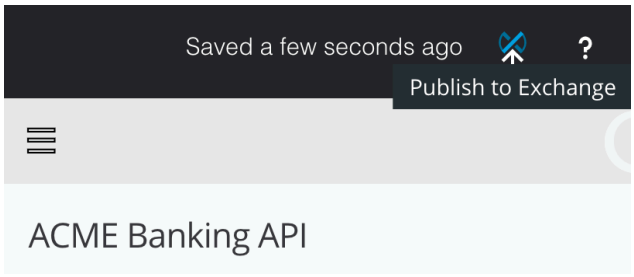
4. In the RAML editor, verify that the acme-banking-api.raml file is open.
5. Replace the value of the version field with v2.

```
#%RAML 1.0
baseUri: https://mocksvc.mulesoft.com/mocks
version: v2
title: ACME Banking API
mediaType: application/json
```

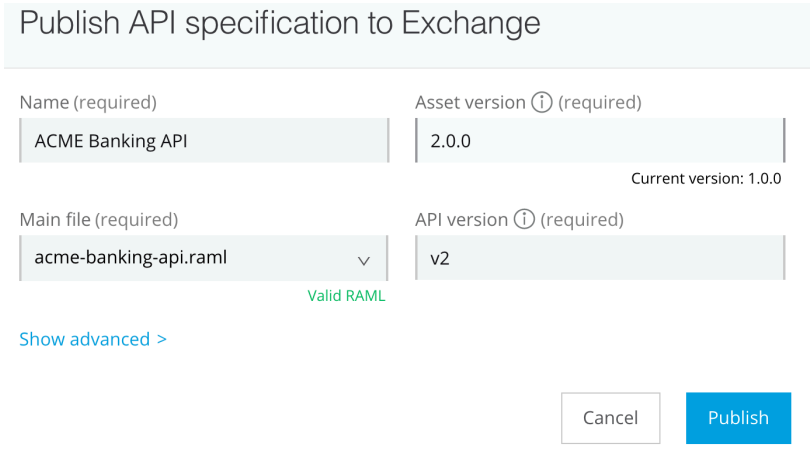
Note: Now you are all set to make changes to the master branch v2 version of the ACME Banking API.

Verify that you can publish the new version of the API to Exchange and API Manager

6. In the API designer page, click the Publish to Exchange button on the top right corner.



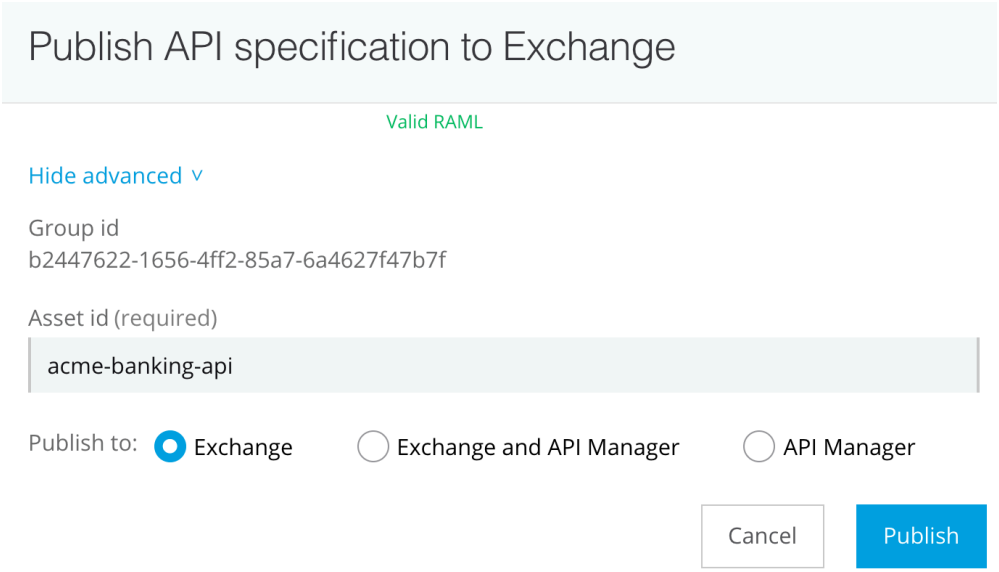
7. In the Publish API specification to Exchange dialog box, verify that the asset version and the API version are set to the 2.0 numbers respectively.



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.

8. In the same dialog box, click the Show advanced link.
9. Verify that you see the option to publish to API Manager.



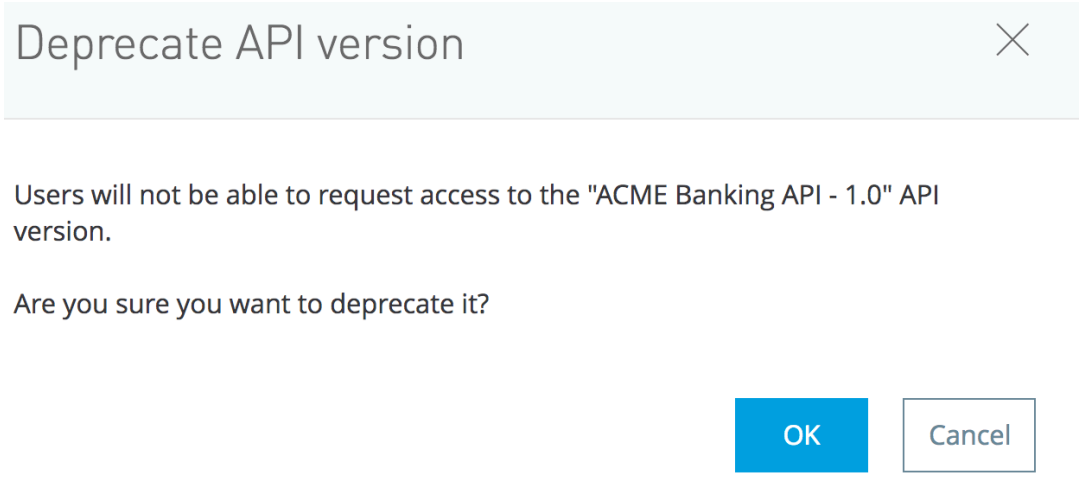
Note: Once the API specification is ready to be implemented, import to API Manager. The API Portal inside API Manager can be shared between API versions. The non-identical items in the shared API Portal will be the version id in the portal link, the API reference page and API Notebooks.

10. Click Cancel to exit out of the dialog box.

Deprecate the older version of the API

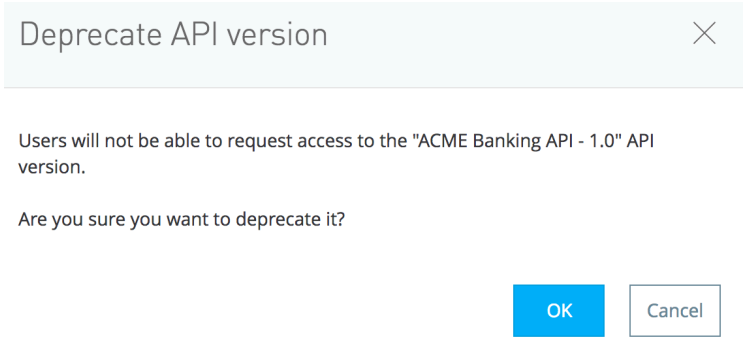
11. In API designer, click the top left Design Center icon.
12. In the list of projects page in Design Center, click the top left menu icon.
13. Click API Manager.
14. In API Manager, locate ACME Banking API and click 1.0 version.
15. In the API version details page, locate the More button on the top right corner of the page and click it.

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.

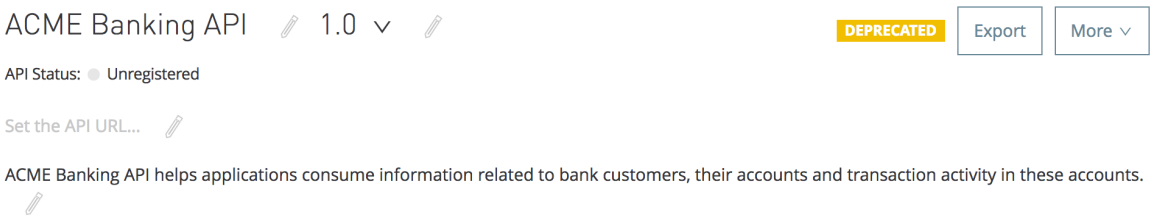


Note: Deprecation should be performed only when an API is live in production, being used by consumers.

18. Click OK.



19. Verify that you see a yellow Deprecated tag next to the API name and version.



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.