Build REST APIs using Swagger and IBM Integration Bus – IIB v10

Installing Swagger locally

1. Install NodeJS
    a. Choose the appropriate installer from https://nodejs.org/download/

```
2. git clone https://github.com/swagger-api/swagger-editor.git
3. cd swagger-editor
4. npm start
```

Note for 3. You will find the Swagger-editor in the below location in Windows

C:\Users\{UserName}\Documents\GitHub\swagger-editor

Once Swagger Editor starts locally it will open the Editor in the below location

http://localhost:8080/#/

The browser based swagger editor is available in the internet at

http://editor.swagger.io/#/

Swagger API Specification can be found here

https://github.com/swagger-api/swagger-spec/blob/master/versions/2.0.md

Create a Simple Rest API Definition in Swagger

Below is a simple REST API named Payment API. The Payment API will enable customers to view all scheduled payments for a Customer and post payments to different accounts setup for bill pay.

The goal here is to just show how to define a REST API using Swagger as use it for development in IIB v10. I will cover how to model REST APIs using RAML in a different post.

Below is the Payment API Swagger definition. Swagger definitions are in YAML format.

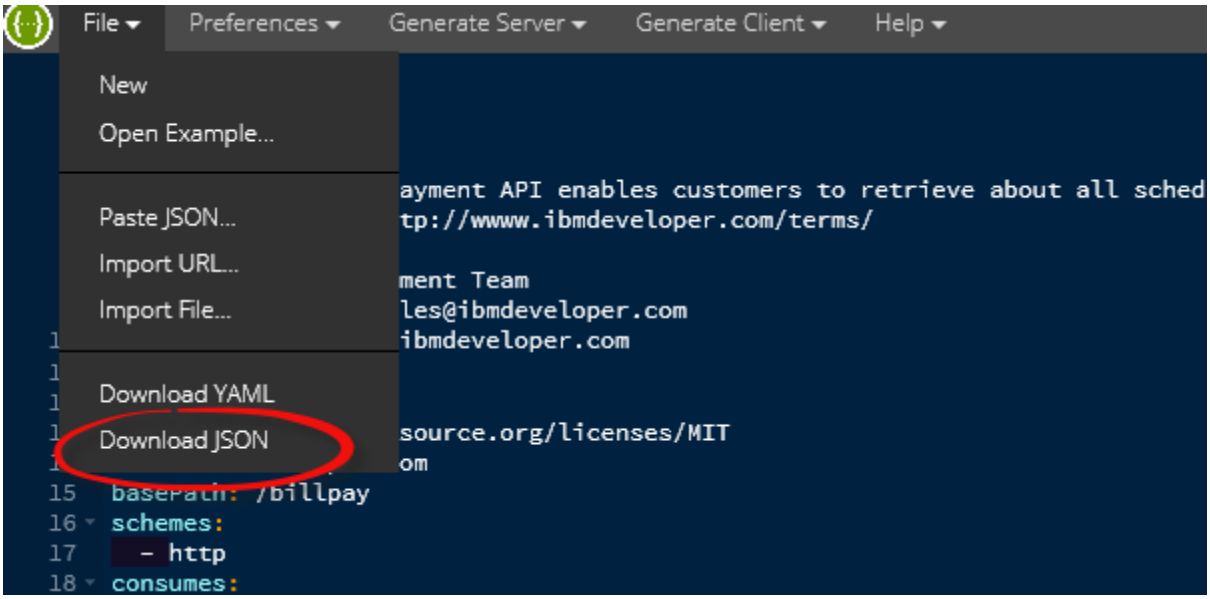
```
swagger: '2.0'
info:
  version: '1.0.0'
  title: Payment API
```

```
  description: The Payment API enables customers to retrieve about all scheduled payments & post payments
  termsOfService: http://wwww.ibmdeveloper.com/terms/
  contact:
    name: API Management Team
    email: juliansmiles@ibmdeveloper.com
    url: http://www.ibmdeveloper.com
  license:
    name: MIT
    url: http://opensource.org/licenses/MIT
host: ibmdeveloper.com
basePath: /billpay
schemes:
  - http
consumes:
  - application/json
produces:
  - application/json
paths:
  /payments/{customerId}:
    get:
      description: Returns all Payments that are scheduled for a customer
      operationId: findPayments
      produces:
        - application/json
        - application/xml
        - text/xml
        - text/html
      parameters:
        - name: customerId
          in: path
          type: string
          description: ID of the Customer
          required: true
      responses:
        '200':
          description: scheduled payments response
          schema:
            type: array
            items:
              $ref: '#/definitions/payments'
        default:
          description: unexpected error
          schema:
            $ref: '#/definitions/errorModel'
    post:
      description: Schedule/Post a new payment
      operationId: postPayment
      produces:
        - application/json
      parameters:
        - name: customerId
          in: path
          type: string
          description: ID of the Customer
          required: true
```

```
        - in: formData
          name: paymentAmount
          description: Payment Amount to be posted
          required: true
          type: number
          format: float
        - in: formData
          name: paymentDate
          description: Payment Date
          required: true
          type: string
        - in: formData
          name: paymentAccount
          description: Account where payment is to be posted
          required: true
          type: string
      responses:
        '200':
          description: payment response
          schema:
            $ref: '#/definitions/paymentId'
        default:
          description: unexpected error
          schema:
            $ref: '#/definitions/errorModel'

definitions:
    payments:
      required:
        - paymentId
        - paymentAccount
        - paymentDate
      properties:
        paymentId:
          $ref: '#/definitions/paymentId'
        paymentAccount:
          type: string
        paymentDate:
          type: string
        tag:
          type: string
    paymentId:
          type: integer
          format: int64
    errorModel:
      required:
        - code
        - message
      properties:
        code:
          type: integer
          format: int32
        message:
          type: string
```

IIB v10 expects the definition to be in JSON format. The Swagger editor provides the option to save the file in JSON format.



The Payment API has 2 methods, GET and POST

Payment API

The Payment API enables customers to retrieve about all scheduled payments & post payments

Version 1.0.0

Contact information

API Management Team
juliansmiles@ibmdeveloper.com
http://www.ibmdeveloper.com

Terms of service

http://wwww.ibmdeveloper.com/terms/

License

MIT

The GET method will retrieve all the payments that are scheduled for a Customer. The Customer Id is the key that identifies a specific customer.

GET /payments/{customerId}

Description

Returns all Payments that are scheduled for a customer

Parameters

Name	Located in	Description	Required	Schema
customerId	path	ID of the Customer	Yes	string

Responses

Code	Description	Schema
200	scheduled payments response	[payments {}]
default	unexpected error	errorModel { code: integer * message: string * }

Try this operation

The POST method enables a customer to post/schedule payments to a specific bill pay account.

POST /payments/{customerId}

Description

Schedule/Post a new payment

Parameters

Name	Located in	Description	Required	Schema
customerId	path	ID of the Customer	Yes	string
paymentAmount	formData	Payment Amount to be posted	Yes	number (float)
paymentDate	formData	Payment Date	Yes	string
paymentAccount	formData	Account where payment is to be posted	Yes	string

Responses

Code	Description	Schema
200	payment response	paymentId integer (int64)
default	unexpected error	errorModel { code: integer * message: string * }

Try this operation

Below are the message models / response message formats.

The errorModel defines a generic error message format for any errors.

The response to the POST is a paymentId.

The response to the GET is the payments structure, which returns the paymentAccount, paymentDate & paymentId

Models

errorModel

```
errorModel {
  code:    integer *
  message: string *
}
```

paymentId

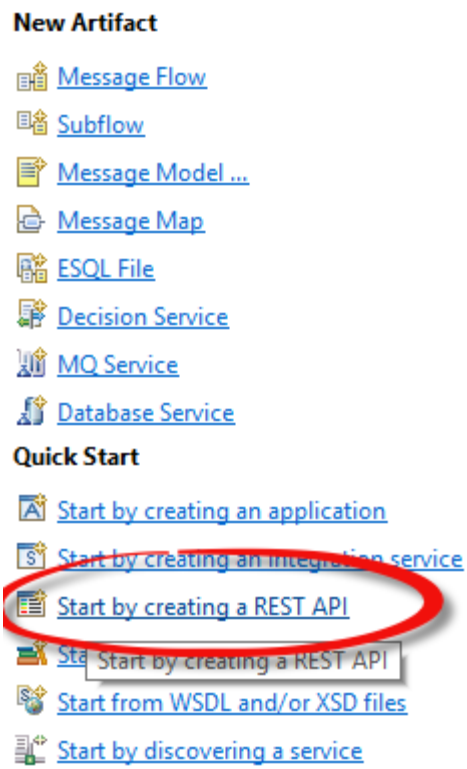
```
paymentId integer (int64)
```

payments

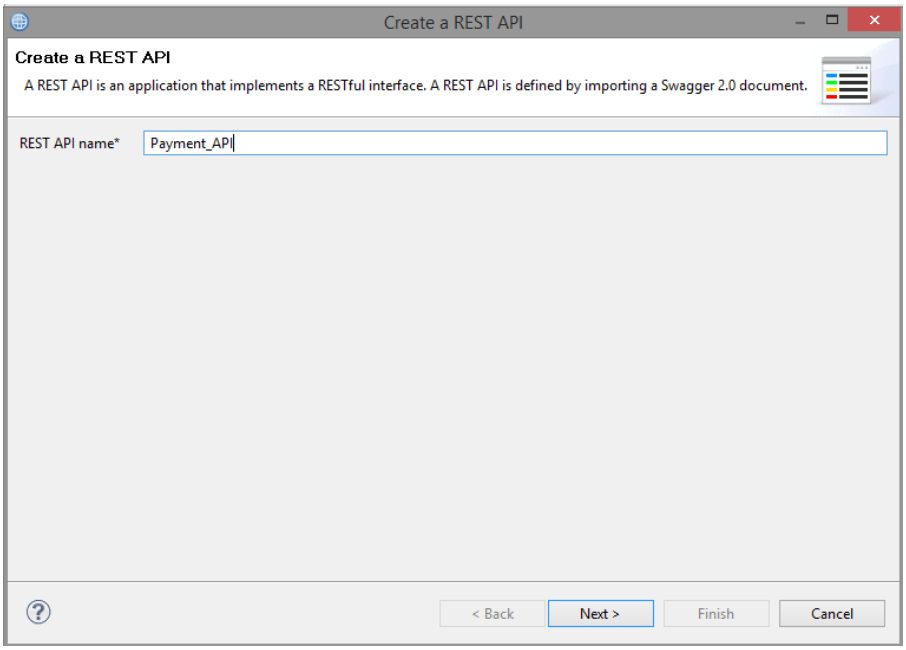
```
payments {
  paymentAccount: string *
  paymentDate:    string *
  paymentId:      paymentId integer * (int64)
  tag:            string
}
```

REST API Implementation using Integration Bus - IIBv10

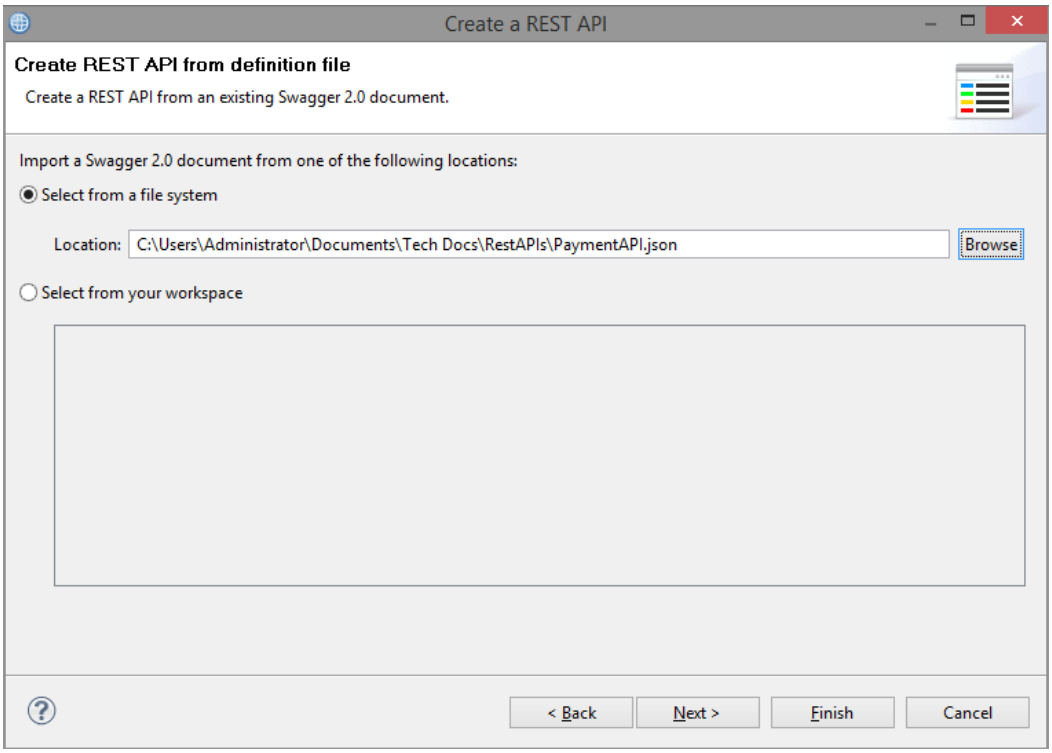
Choose New -> Start by creating a REST API



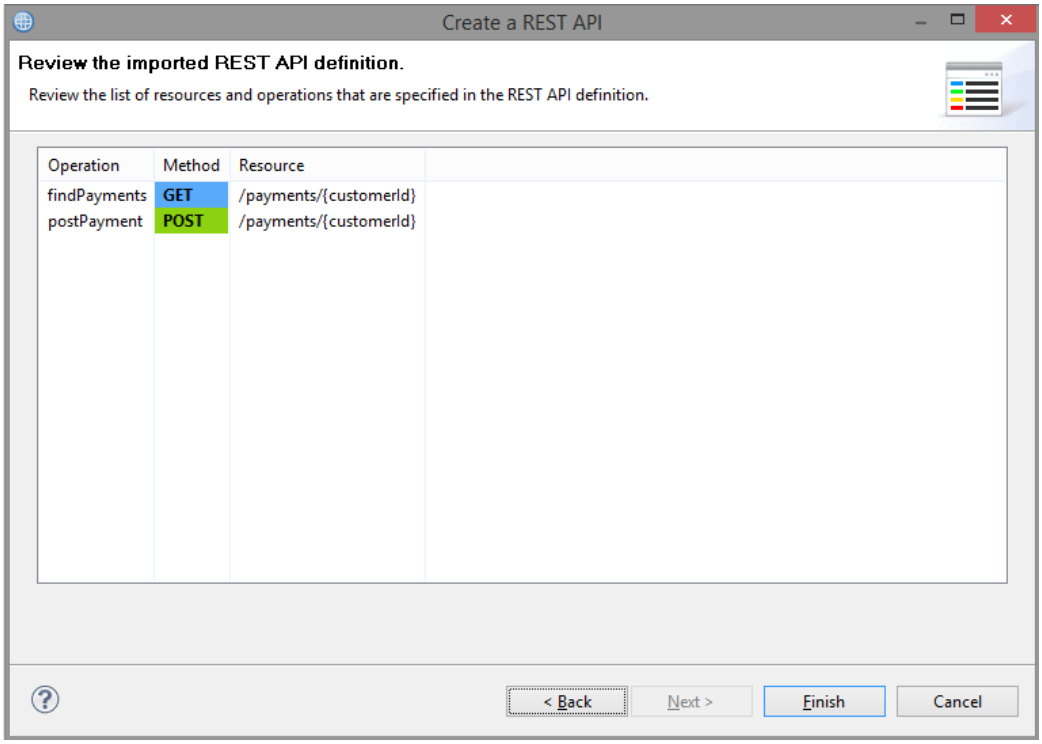
Specify the name of the REST API



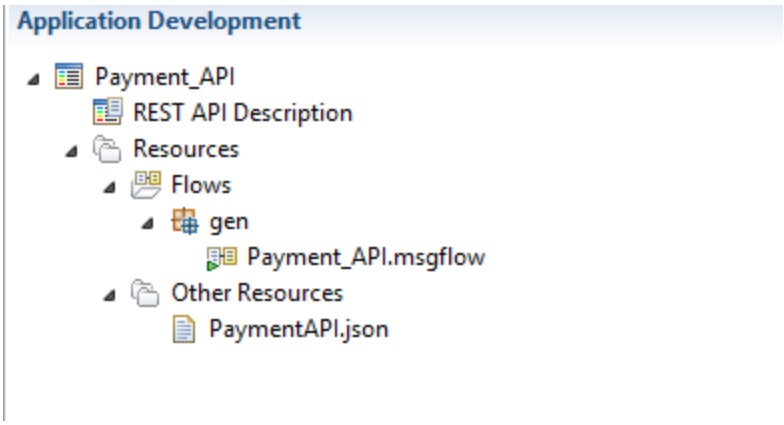
Select the PaymentAPI.json file we downloaded earlier from the Swagger Editor



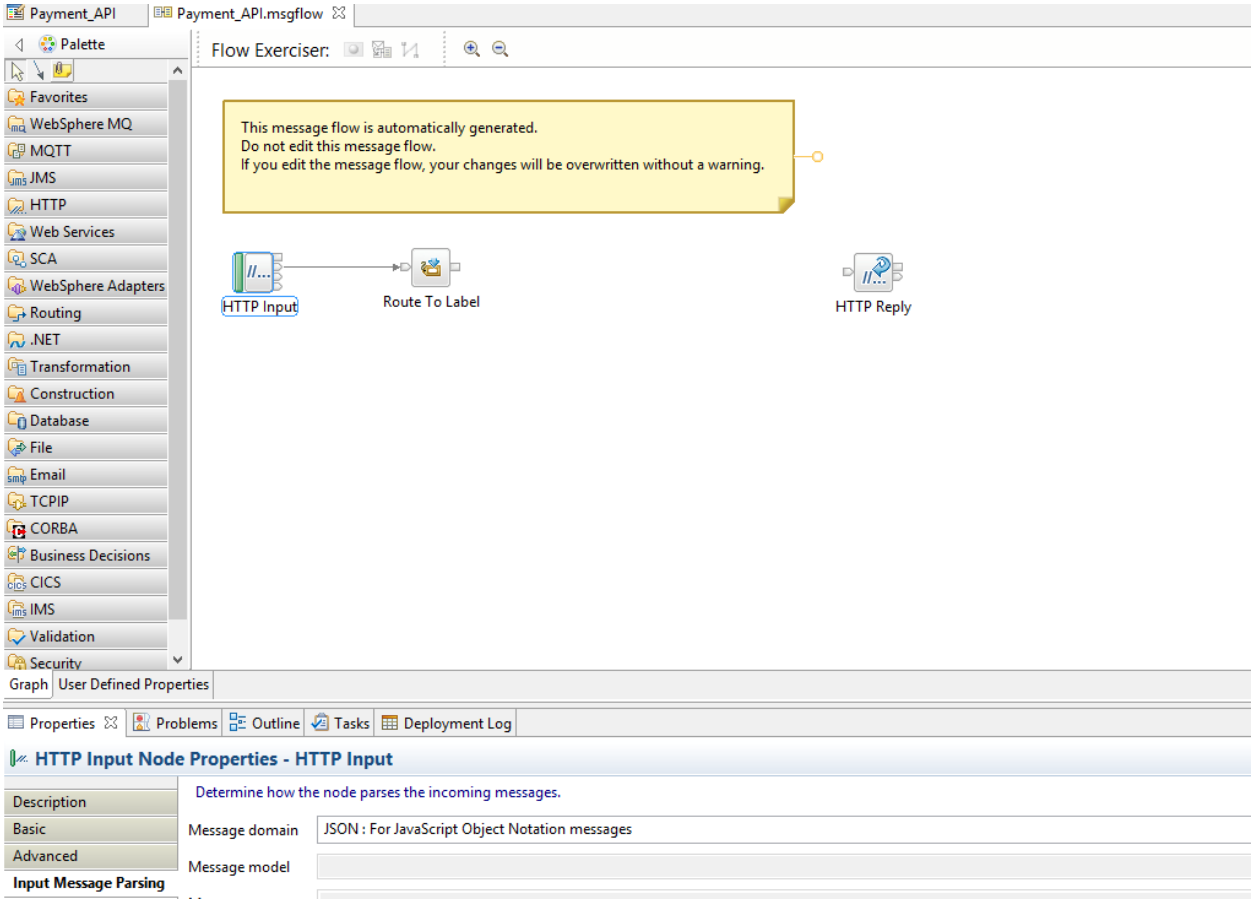
Review the API definition



Once the import is successful, the API description and sample message flows are created like below.



Below is the IIB generated Payment_API.msgflow. The default message domain as you can see below is JSON.



Below is the REST API description inside the toolkit.

Payment_API

REST API base URL: /billpay

You can access the operations in the REST API by pointing your web browser to the following URL, where <hostname> is the host name and <port_number> is the port number:
http://<hostname>:<port_number>/billpay

Operations — Expand all / Collapse all

/payments/{customerId}

GET findPayments — Implement the operation

Path Parameters	Required	Description
customerId	Yes	ID of the Customer

POST postPayment — Implement the operation

Form Parameters	Required	Description
paymentAccount	Yes	Account where payment is to be posted
paymentDate	Yes	Payment Date
paymentAmount	Yes	Payment Amount to be posted
Path Parameters	**Required**	**Description**
customerId	Yes	ID of the Customer

Error Handling

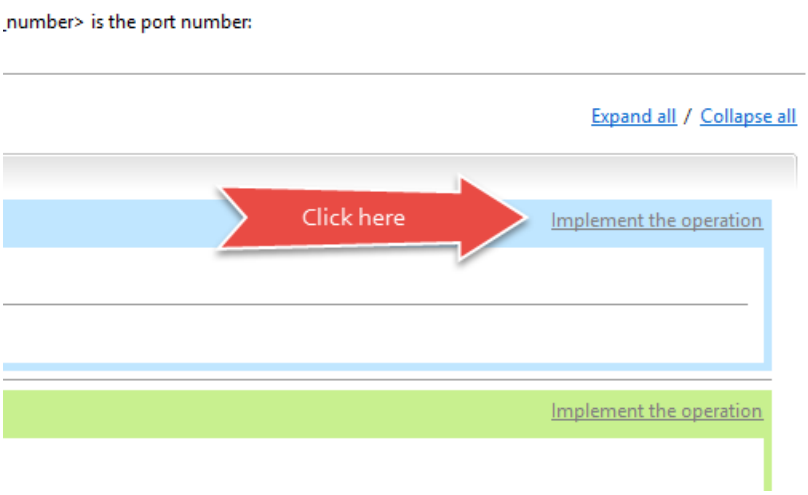
Implement the Catch handler	The subflow to which the message is routed if an exception is not handled in an operation subflow.
Implement the Failure handler	The subflow to which the message is routed if an error occurs.
Implement the Timeout handler	The subflow to which a timeout message is routed if an operation subflow does not respond to the client within the expected time limit.

Security

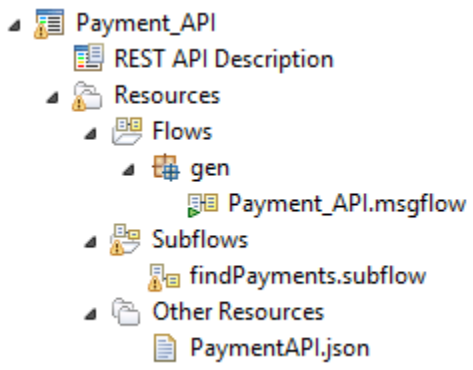
☐ Enable HTTPS

You can enable other security settings in the BAR file editor.

Choose 'Implement the operation' next to both



This will create a sub-flow for each operation i.e. one for GET and one for POST. Below is the sub-flow for the GET operation. The name of the sub-flow reflects the operationId in the API definition.



Before we complete the entire implementation, I would like to deploy the API to make sure we can invoke the API.

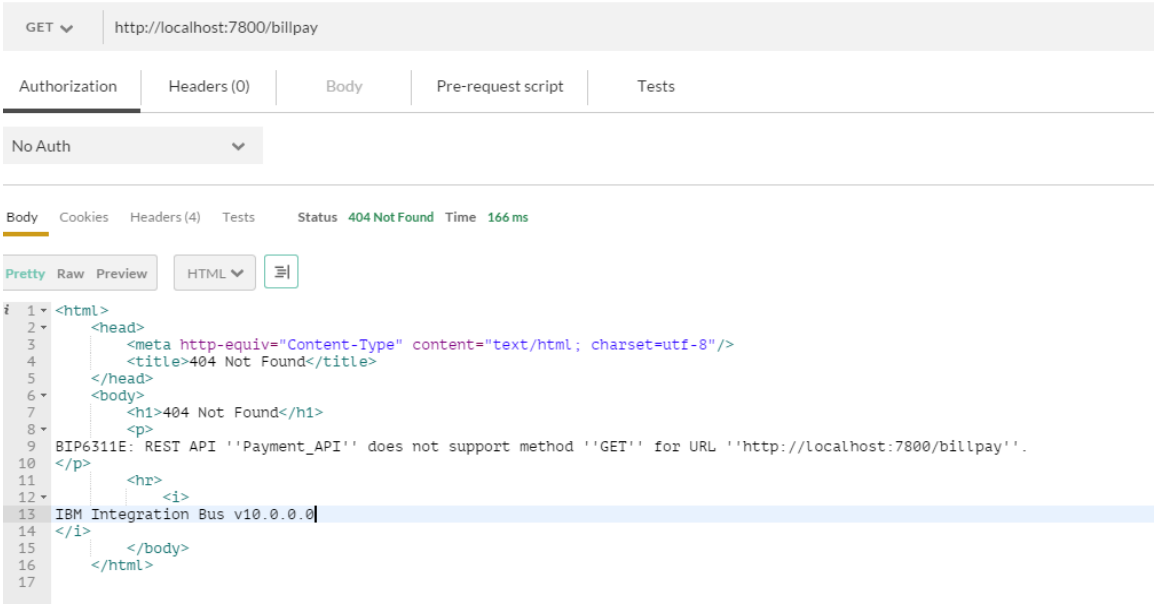
To test the API, I use Postman which is great for REST API testing. There are many other tools available and you can use the tool of your choice. Postman is available in the Chrome App Store.

Right Click on the Payment API and deploy it to an integration server. Once it is deployed, you should be able to see the properties like below.

Property	Value
API	
Base URL for local invocations	http://localhost:7800/billpay
Base URL for remote invocations	http://192.168.28.1:7800/billpay
Local URL for the REST API definitions	http://localhost:7800/billpay/PaymentAPI.json
Remote URL for the REST API definitions	http://192.168.28.1:7800/billpay/PaymentAPI.json

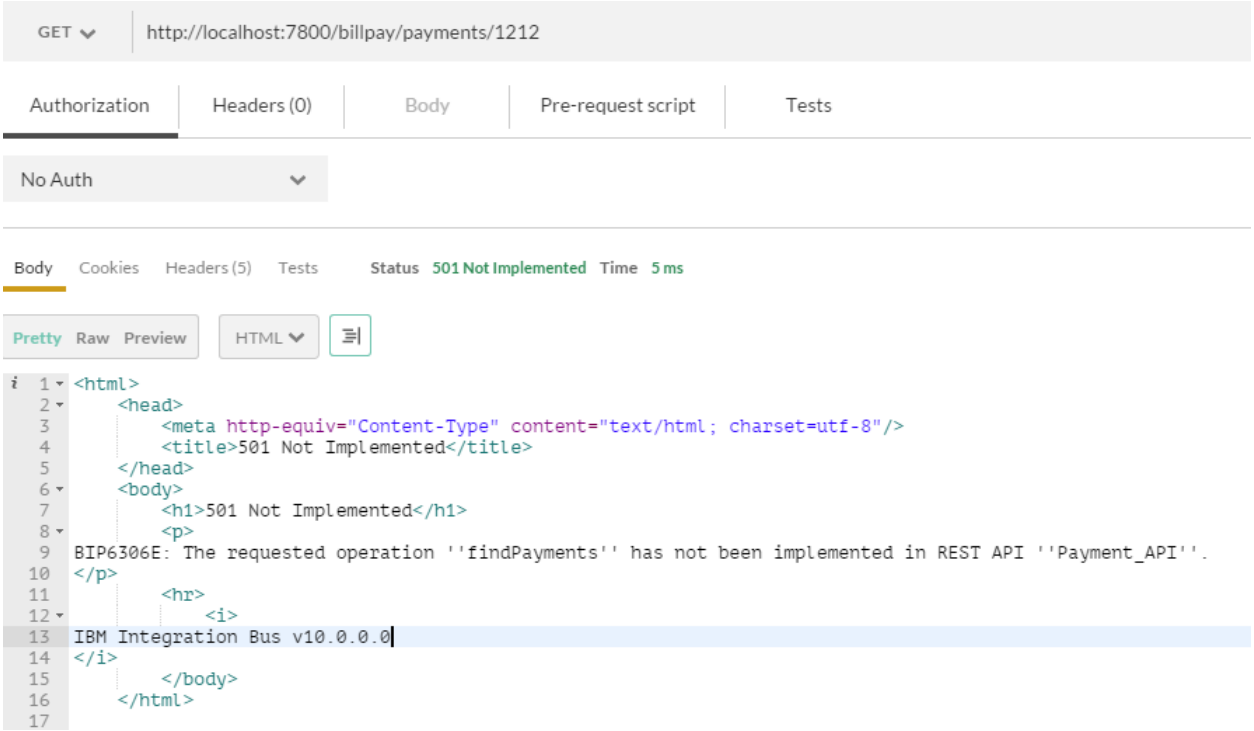
It shows you the URL you need to invoke.

Let's try to invoke the above URL using Postman



This returns a 404 as the API does not support base URL invocation. The right URL should be base URL + /billpay/{customerId}

Let's change the URL to http://localhost:7800/billpay/payments/1212



This returns a 501 as we haven't completed the sub-flow implementations yet.

Now that we can invoke the API, let's move on to complete the message flows.

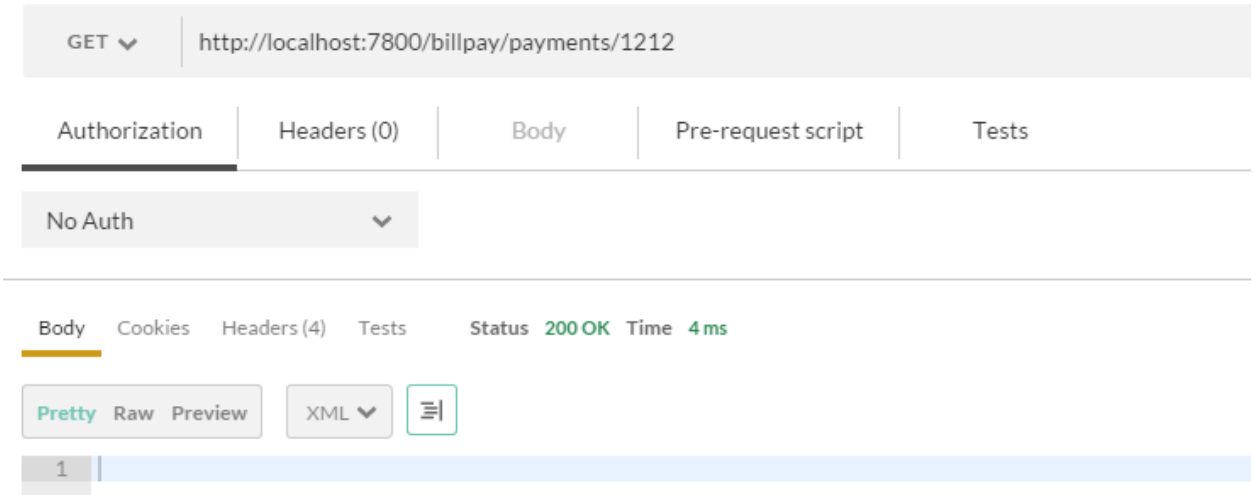
Let's add a Compute Node to both the sub-flows and leave the ESQL code like below, deploy and test

```
CREATE COMPUTE MODULE findPayments_Compute
    CREATE FUNCTION Main() RETURNS BOOLEAN
    BEGIN
        CALL CopyMessageHeaders();
        CALL CopyEntireMessage();
        RETURN TRUE;
    END;

    CREATE PROCEDURE CopyMessageHeaders() BEGIN
        DECLARE I INTEGER 1;
        DECLARE J INTEGER;
        SET J = CARDINALITY(InputRoot.*[]);
        WHILE I < J DO
            SET OutputRoot.*[I] = InputRoot.*[I];
            SET I = I + 1;
        END WHILE;
    END;

    CREATE PROCEDURE CopyEntireMessage() BEGIN
        SET OutputRoot = InputRoot;
    END;
END MODULE;
```

When you try to test GET operation now, you a 200 Status code. But the response is empty as we haven't implemented any code to return as valid JSON response.



Let's make some changes to the above ESQL to return a valid response.

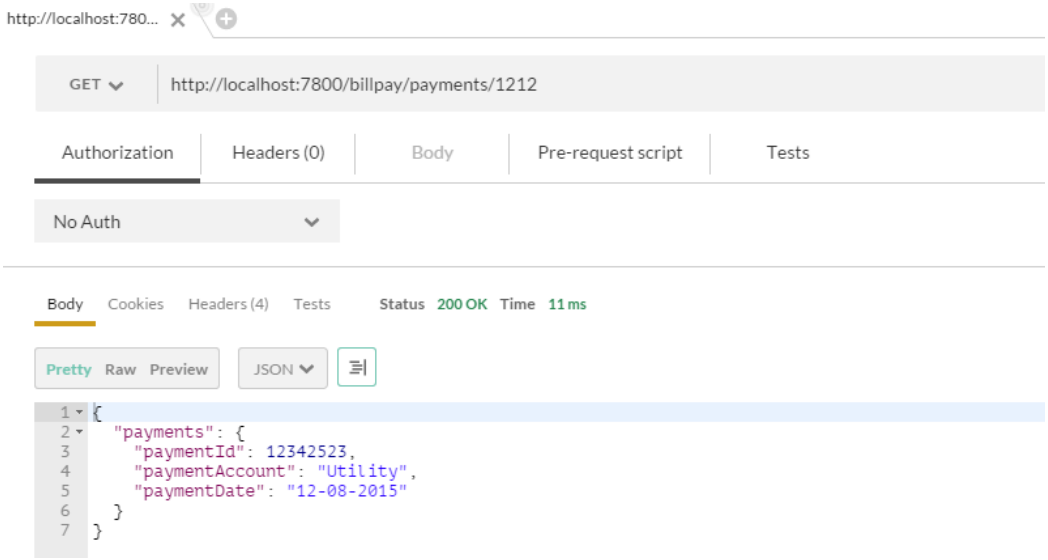
```
CREATE COMPUTE MODULE findPayments_Compute
    CREATE FUNCTION Main() RETURNS BOOLEAN
    BEGIN
        CALL CopyMessageHeaders();
        CALL CopyEntireMessage();
        RETURN TRUE;
    END;

    CREATE PROCEDURE CopyMessageHeaders() BEGIN
        DECLARE I INTEGER 1;
        DECLARE J INTEGER;
        SET J = CARDINALITY(InputRoot.*[]);
        WHILE I < J DO
            SET OutputRoot.*[I] = InputRoot.*[I];
            SET I = I + 1;
        END WHILE;
    END;

    CREATE PROCEDURE CopyEntireMessage() BEGIN
        SET OutputRoot.JSON.Data.payments.paymentId = 12342523;
        SET OutputRoot.JSON.Data.payments.paymentAccount = 'Utility';
        SET OutputRoot.JSON.Data.payments.paymentDate = '12-08-2015';
    END;
END MODULE;
```

I have just hardcoded the values as our goal is to just build a basic API. I haven't shown how to integrate with a backend etc. for a comprehensive implementation as it is beyond the scope of what I am trying to demonstrate here.

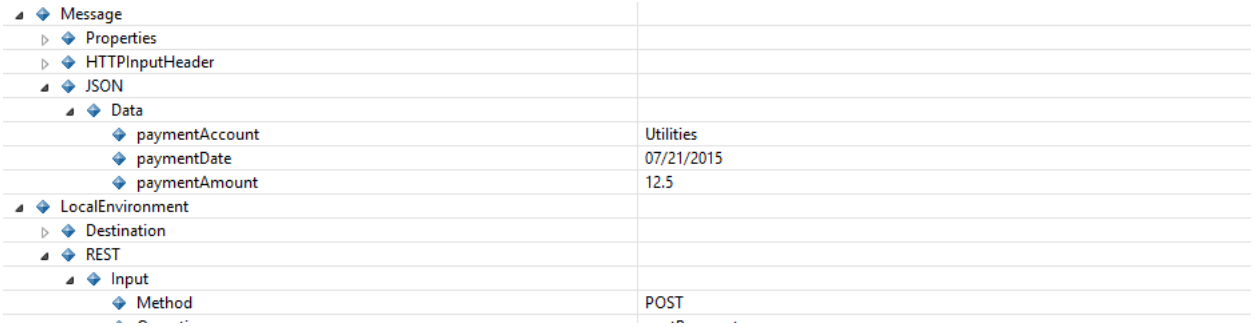
Now let's try to invoke the URL again for GET



Now you get a valid JSON response.

You can try your POST operation similarly

Here is the representation of the JSON message tree in the debugger for the POST



I hope this information was useful to you.