Predix

Modeling Assets

Student Lab Guide

September 2015

GE Digital

Predix

Getting Started

This guide provides step-by-step instructions for lab exercises. Each lab corresponds to a topic covered in class and provides students with hands-on experience developing applications on the Predix platform.

Course Prerequisites:

- Have a Cloud Foundry account
- Have access to GE's GitHub repository
- Install the most recent DevBox version

Start the DevBox in Oracle VirtualBox

- Login with:

  Username: **predix**

  Password: **predix**

Note: All lab exercises will be performed in your Devbox.

Lab 1 : Using the Asset Service

Learning Objectives

By the end of this lab, students will be able to:

- Create an Asset service instance in Cloud Foundry
- Bind an application to an Asset service instance

Lab Exercises

Directions

Complete the exercises that follow.

Exercise 1: Creating an Asset Service Instance

Overview

To use a Predix Service, you must first create an instance of the service in Cloud Foundry.

Steps

1. Log into Cloud Foundry (CF).

- Open a Terminal and run the command:
  `cf login -a https://api.grc-apps.svc.ice.ge.com/`
  This logs you into GE's API endpoint for CF
- You are prompted to enter the following information:
  - Email: enter your email address and password

```
[predix@localhost ~]$ cf login -a https://api.grc-apps.svc.ice.ge.com
API endpoint: https://api.grc-apps.svc.ice.ge.com

Email> asha.yadav@ge.com

Password>
Authenticating...
OK
```

2. Display all services in the Predix marketplace.

- In the Terminal run the command:

```
cf marketplace
```

```
SFO1502434206A:scripts 502434206$ cf marketplace
Getting services from marketplace in org predix-adoption / space training1 as asha.yadav@ge.com...
OK

service                    plans             description
bizops                     beta-plan         Predix BizOps Service by Nurego Inc.
logstash14-stc-logging-5   beta-plan         Data pipeline to process logsl and event data.  Elasticsearch, Logstash, Ki
p-rabbitmq                 standard          RabbitMQ is a robust and scalable high-performance multi-protocol messaging
p-riakcs2                  developer         An S3-compatible object store based on Riak CS
stc-analytics-catalog      beta-plan         STC Analytics Catalog
stc-analytics-runtime      beta-plan         STC Analytics Runtime
stc-asset                  beta-plan         Service to model and manage industrial assets
stc-monitoring             beta-plan         Manage and monitor apps. By New Relic
stc-postgresql-2           free              PostgreSQL 9.3 service for application development and testing
stc-redis-6                shared-vm         Redis service to provide a key-value store
stc-time-series            beta-plan         Predix Time-Series Service
time-series-dev            Development Plan  Time-Series Service

TIP:  Use 'cf marketplace -s SERVICE' to view descriptions of individual plans of a given service.
```

Analysis: The Assert service is available in the Service catalog. To use the Asset service, you will first need to create an instance of the service in your space. This is done using the **cf create-service** command.

3. Create an Asset service instance.

- In the Terminal run the **`cf create-service --help`** command to display a list of required arguments:

```
USAGE:
   cf create-service SERVICE PLAN SERVICE_INSTANCE
```

where:

- <SERVICE> will be **stc-asset**
- <PLAN> will be **beta-plan**
- <SERVICE_INSTANCE> will be the instance name you choose, such as **AssetService_YourName**

- In the Terminal run the command:
  `cf create-service stc-asset beta-plan AssetService_YourName`
  (Replace *YourName* with your first and last name)

```
[predix@localhost ~]$ cf create-service stc-asset beta-plan AssetService_testuser
Creating service AssetService_testuser in org predix-adoption / space training1 as
OK
```

4. Verify the Asset service instance created in your training space.

- To list all services in your training space, run the command:
  `cf services`

```
SFO1502434206A:scripts 502434206$ cf services
Getting services in org predix-adoption / space training1 as asha.yadav@ge.com...
OK

name                      service                plan         bound apps
AssetService_YourName     stc-asset              beta-plan
AssetService_testuser     stc-asset              beta-plan
asset-service             stc-asset              beta-plan
billpostgres              stc-postgresql-2       free         predix-alarmservice, predix-alarmservice5-kunalpa
k-analytics               stc-analytics-runtime  beta-plan
myAssetService            stc-asset              beta-plan
mypostgres-YahangWu       rdpg                   shared
mypostgres-kunalpatil     rdpg                   shared       predix-alarmservice5-kunalpatil
mypostgres_MonikaJain     rdpg                   shared       predix-alarmservice-MonikaJain, predix-alarmservi
mypostgres_YahangWu       rdpg                   shared       predix-alarmservice-YahangWu
mypostgres_zehua          rdpg                   shared
mypostgres_zzh            rdpg                   shared       predix-alarmservice-zzh
mypostgreskunal           rdpg                   shared       predix-alarmservice-kunalpatil, predix-alarmservi
mypostgressusankim        rdpg                   shared       predix-alarmservicesusankim
mypostgreszehuazheng      rdpg                   shared
newRelic                  stc-monitoring         beta-plan
postgresAsha              stc-postgresql-2       free         predix-alarmservice-testingforKunal, predix-alarm
postgresql93              stc-postgresql-2       free         predix-alarmservice5-kunalpatil
predixAsset               stc-asset              beta-plan    dev-rmd-refapp-qian, training-rmd-refapp-21242978
predixTimeseries          stc-time-series        beta-plan
training_pgsql193         rdpg                   shared       predix-alarmservice-testingforKunal, predix-alarm
viewPersistenceService    user-provided                       dev-rmd-refapp-qian, training-rmd-refapp-21242978
```

Your instance of the Asset service should now be available in the training space.

Exercise 2: Binding to Your Asset Service Instance

Overview

In this exercise you will bind a data-seed application to your Asset service instance. The data-seed application is one of the reference applications available as part of the Starter Pack in the Predix catalog. This application requires the Asset Service to import asset data into the Asset model.

Steps

1. Find the data-seed application in your Cloud Foundry space.

- In the Terminal run the command:

```
cf a
```

```
[predix@localhost ~]$ cf a
Getting apps in org predix-adoption / space training1 as vicki.soll@ge.com...
OK

name                        requested state   instances   memory   disk   urls
predix-dataseed-training    started           1/1         1G       1G     predix-dataseed-training.grc-a
```

2. Bind the data-seed service to your Assert service instance.

- In the Terminal run the command:
  `cf bind-service predix-dataseed-training AssetService_YourName`
  (Replace YourName with your first and last name)

```
cf bind-service predix-dataseed-training AssetService_testuser
Binding service AssetService_testuser to app predix-dataseed-training in org predi
n / space trainingl as vicki.soll@ge.com...
OK
```

3. Display environment variables for your application.

- In the Terminal run the command:
  `cf env predix-dataseed-training`

```
[predix@localhost ~]$ cf env predix-dataseed-training
Getting env variables for app predix-dataseed-training in org predix-adoption
OK

System-Provided:
{
 "VCAP_SERVICES": {
  "stc-asset": [
   {
    "credentials": {
     "uri": "http://predix-asset-stc.grc-apps.svc.ice.ge.com"
    },
    "label": "stc-asset",
    "name": "predixAsset",
    "plan": "beta-plan",
    "tags": []
```

4. Verify the data-seed application is bound to your asset service instance.

- In the Terminal run the command: **`cf s`**
  - Your service instance should now be listed

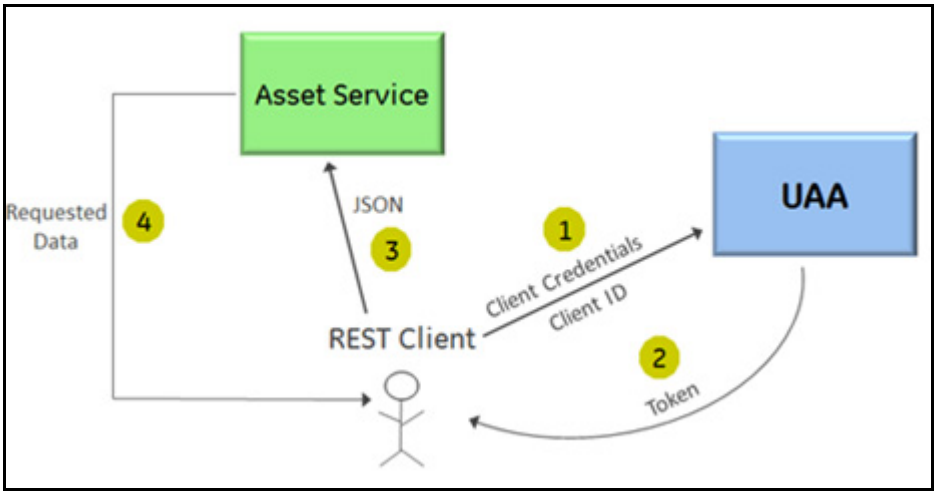
```
[predix@localhost ~]$ cf s
Getting services in org predix-adoption / space training1 as vicki.soll@ge.com...
OK

name                    service     plan        bound apps                  last operation
AssetService_MyTest     stc-asset   beta-plan   predix-dataseed-training    create succeeded
AssetService_testuser   stc-asset   beta-plan   predix-dataseed-training    create succeeded
ashasvc                 stc-asset   beta-plan   predix-dataseed-training    create succeeded
```

Lab 2: Authenticating Against UAA

Learning Objectives

By the end of this lab, students will be able to:

- Retrieve a UAA token from the UAA service



Lab Exercises

- *Fetch a Token from the UAA Service*, page 10

Directions

Complete the exercises that follow.

Exercise 1: Fetch a Token from the UAA Service

Overview

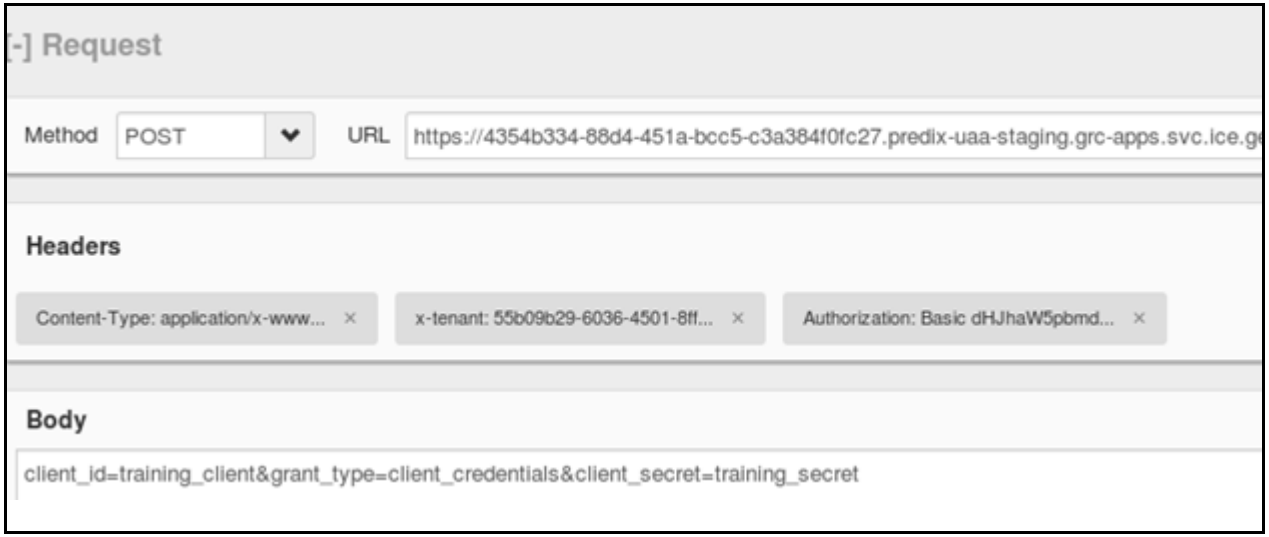
To access and update asset model data, users require a UAA token. Here, you use the REST client to request the token from the UAA Service. The token needs to be added to every data request to the Asset Service.

Steps

1. Launch the REST client.

- In a Firefox browser launch the REST client (click on the red icon shown here)

- You will build a POST request that looks like this: (steps on next page)



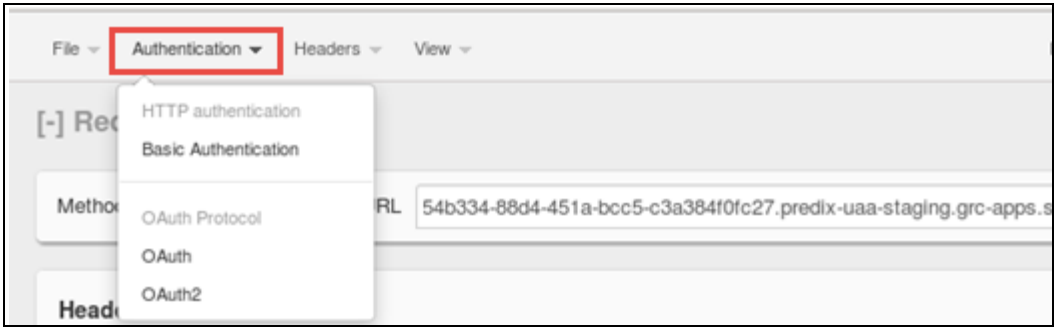
2. Begin configuring the POST request.

- Set the Method and URL
  - Method: **POST** (select from drop-down)
  - URL:

`https://4354b334-88d4-451a-bcc5-c3a384f0fc27.predix-uaa-staging.grc-apps.svc.ice.ge.com/oauth/token`

3. Add username and password credentials as a header.

- Click on the **Authentication** drop-down and select **Basic Authentication**



  - Enter the user name: **`training_client`**
  - Password: **`training_secret`**
  - Click **Okay**; a new authorization header has been added

4. Add a Content-Type header.

- In the Headers drop-down, select **Custom Header**
  - Enter the name: **`Content-Type`**
  - Enter the value: **`application/x-www-form-urlencoded`**
    - Click **Okay**; a second header is added

5. Add an x-tenant header.

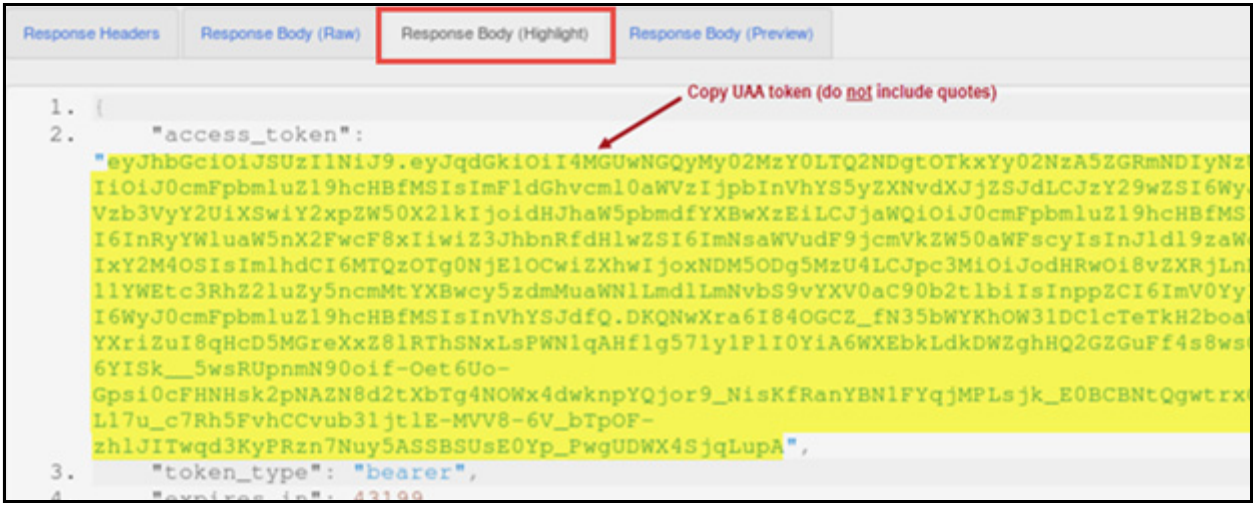
- In the Headers drop-down, select **Custom Header**
- Enter the name: **`x-tenant`**
- Enter the value: **`55b09b29-6036-4501-8ff4-83f6b2807412`**

6. Construct the body of the response message.

- In the payload section of the body enter the following:
  `client_id=training_client&grant_type=client_credentials&client_secret=training_secret`
- Click the **Send** button
- A return code **200 OK** indicates the request was successful

7. Copy the token from the response for later use.

- Under the Response section, select the *Response Body (Highlight)* tab
- Copy the UAA token as shown: (only copy text within the quotes, not the quotes themselves)



- You will use this token in the next lab to retrieve assets from the asset mode

Lab 3: Working with Asset Modeling

Learning Objectives

By the end of this lab, students will be able to:

- Invoke Service Asset APIs to retrieve, add and delete asset objects
- Use JSON to define asset objects

Lab Exercises

Directions

Complete the exercises that follow.

Exercise 1: Retrieve Asset Model Data

Overview

In this exercise you retrieve locomotive asset objects using the Asset Service APIs.

Steps

1. Create a **GET** request for all locomotive objects.

- Open a new tab in Firebox and open the REST client
- In the REST client set the Method and URL settings
  - Method: **GET**
  - URL:

  `http://predix-asset-rc.grc-apps.svc.ice.ge.com/locomotive`
- Add a Custom Header
  - Name: **`Content-Type`**
  - Value: **`application/json`**
- Add a second Custom Header
  - Name: **`Authorization`**
  - Value: **`Bearer`** *<paste the Token returned by UAA>*

- Add a third Custom Header
  - Name: **`x-tenant`**
  - Value: **`55b09b29-6036-4501-8ff4-83f6b2807412`**

- **Send** the request; verify a status code of **200 OK** is returned in the *Header Request*

2. Review the returned asset data.

- Select the *Response Body (Preview)* tab to view locomotive assets returned


```
[
    {
        "uri": "/locomotive/1",
        "type": "Diesel-electric",
        "model": "ES44AC",
        "serial_no": "001",
        "emission_tier": "0+",
        "fleet": "/fleet/up-1",
        "manufacturer": "/manufacturer/GE",
        "engine": "/engine/v12-1",
        "hqLatLng":
        {
            "lat": 33.914605,
            "lng": -117.253374
        }
    },
    {
        "uri": "/locomotive/10",
        "type": "Diesel-electric",
        "model": "SD70ACe",
        "serial_no": "0010",
        "emission_tier": "0+",
        "fleet": "/fleet/up-4",
        "manufacturer": "/manufacturer/electro-motive-diesel",
        "engine": "/engine/v16-2-5",
        "hqLatLng":
        {
            "lat": 47.941049,
            "lng": -100.126484
        }
    },
```

3. Copy JSON for a single asset (for use in next exercise).

- In the *Response Body (Preview)* tab, copy the JSON code for the first locomotive asset as shown here

```
[
    {
        "uri": "/locomotive/1",
        "type": "Diesel-electric",
        "model": "ES44AC",
        "serial_no": "001",
        "emission_tier": "0+",
        "fleet": "/fleet/up-1",
        "manufacturer": "/manufacturer/GE",
        "engine": "/engine/v12-1",
        "hqLatLng":
        {
            "lat": 33.914605,
            "lng": -117.253374
        }
    },
```

  - Include the starting bracket "**[**" and the closing brace "**},**" for the asset

Exercise 2: Add an Asset to the Asset Model

Overview

Post a request to add a new locomotive asset to your asset model.

Steps

1. Use the POST method to add a new asset.

- Open a new tab in Firefox and open the REST client
  - Change the Method to **POST**
  - Verify the URL:

  `http://predix-asset-rc.grc-apps.svc.ice.ge.com/locomotive`

2. Add authentication and content-type headers.

- Add a Custom Header
  - Name: **`Content-Type`**
  - Value: **`application/json`**
  - Click **Okay**
- Add a Custom Header
  - Name: **`x-tenant`**
  - Value: **`55b09b29-6036-4501-8ff4-83f6b2807412`**
- Add a Custom Header
  - Name: **`Authorization`**
  - Value: **`Bearer <`***paste the Token returned by UAA*`>`**

  TIP: Copy and paste the UAA token from the browser tab where you created your first POST request

3. Construct the body of the message.

- Paste the contents of the asset you copied into the **Body** of the request
- Make the following changes to the JSON
  - **uri**: replace `locomotive/1` with **`locomotive/your_name`**
  - **serial_no**: replace the existing value with a value of your choice
  - **model**: enter a value of your choice
- Remove the comma after the closing curly brace "**}**"
- Add a closing bracket "]" after the last curly brace "**}**"

Body


```
[
  {
     "uri": "/locomotive/testuser",
     "type": "Diesel-electric",
     "model": "ES44test",
     "serial_no": "099999",
     "emission_tier": "0+",
     "fleet": "/fleet/up-1",
     "manufacturer": "/manufacturer/GE",
     "engine": "/engine/v12-1",
     "hqLatLng":
     {
        "lat": 33.914605,
        "lng": -117.253374
     }
  }
]
```


- **Send** the request
- In the *Response Headers* tab a **204 No Content** status code is returned, indicating the asset was successfully added

4. Submit a GET request to retrieve the new asset from the asset model.

- Select the browser tab that contains the **GET** request for all locomotive assets
- Append the name of your *new* locomotive asset to the end of the URL (in our example, the name is /testuser)

- **Send** the request
- Select the *Response Body (Highlight)* tab to view the returned locomotive asset

- This verifies that you successfully added a new asset to the asset model

Exercise 3: Link Domain Objects

Objective

In this exercise you link your new locomotive asset object to a different manufacturer object.

Steps

1. Link the locomotive asset to a different manufacturer.

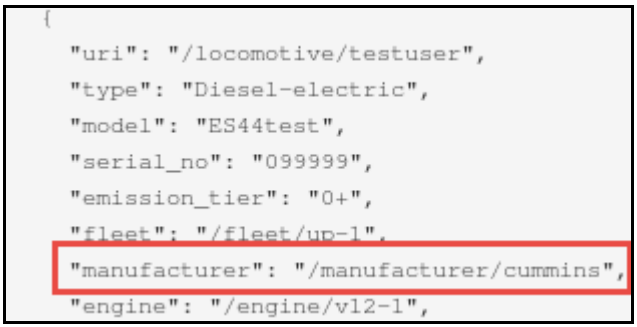
- In the REST client change the Method to: **PUT**
  - URL: verify it points to <u>your</u> locomotive object (our example uses /testuser):

    `http://predix-asset-rc.grc-apps.svc.ice.ge.com/locomotive/testuser`
- In the Body section, change the manufacturer property to point to `cummins`

  `manufacturer:"/manufacturer/cummins"`
- **SEND** the request
- Verify a status code **204 OK** is returned

2. Send a GET request to verify the new link.

- In the REST client change the Method to: **GET**
- **SEND** the request
- In the *Response Body (Raw)* tab, verify the manufacturer property now references `cummins`:



Exercise 4: Delete an Asset from the Asset Model

Overview

In this exercise you use the DELETE API method to remove the locomotive asset (added in the previous exercise) from the asset model.

Steps

1. Use the DELETE method to remove an asset from the asset model.

- Update the GET request
  - Change the Method to **DELETE**
  - Use the existing URL that points to your new locomotive asset
- **Send** the request
- In the *Response Headers* tab, a return code of **204 No Content** indicates the Delete operation was successful

2. Submit a GET request to verify the asset was deleted.

- In the REST client change the Method to **GET**
- **Send** the request
- In the *Response Headers* tab, the response code should be **404 Not Found**
  - This indicates that the asset was successfully deleted

Lab 4: Querying Assets Using GEL

Learning Objectives

By the end of this lab, students will be able to:

- Construct filters using Graph Expression Language to query assets

Lab Exercises

Directions

Complete the exercises that follow.

Exercise 1: Construct GEL Queries

Overview

In this exercise you construct different GEL queries to control asset data returned by GET requests.

Steps

1. Add a **fields** clause to display selected fields for locomotive objects.

- In the REST client, verify the Method is: **GET**
- Verify the request URL references the `/locomotive` domain object
- Append the following fields clause to the end of the URL

  `?fields=uri,model,manufacturer`

- **SEND** the request
- Select the *Response Body (Preview)* tab to view the results

1. Add a **filter** to query all locomotives of *type* Diesel-electric.

- In the RESt client, append the following filter clause to the end of the URL

  `?filter=type=Diesel-electric`

- **SEND** the request
- Select the *Response Body (Preview)* tab to view assets returned

2. Query locomotives that are Diesel-electric **and** are a model of SD70ACe.

- Append the following filter clause to the end of the URL

  `?filter=type=Diesel-electric:model=SD70ACe`

  Note: The **:** symbol denotes an AND operation

- **SEND** the request
- Verify the correct subset of locomotive objects are returned

3. Query locomotives that have an engine type of v12-6.

Note: up until this point, you have constructed filters with simple attributes; now you will add a filter with an attribute that references another domain object.

- Append the following filter clause to the end of the URL

  `?filter=engine=/engine/v12-6`

  (the "**`/engine/`**" references a path to the engine domain object)

- **SEND** the request
- A single locomotive object is returned

4. Construct a forward-relate query to retrieve.

- Change the URL to reference the **engine** domain object

  `http://predix-asset-rc.grc-apps.svc.ice.ge.com/engine`
- Append the following filter clause to the end of the URL

  `?filter=type=Diesel-electric>engine`

- **SEND** the request
- All engines that are part of Diesel-electric type locomotives are returned
- Query Logic
  - The query first returns all objects with a type property=Diesel-electric
  - From that result set, find all objects that have an engine relationship to other objects
  - From that new set of objects, return only engine objects

5. Construct a backwards-relate query to retrieve fleet assets for customer CSX.

- Change the URL to reference the **fleet** domain object

  `http://predix-asset-rc.grc-apps.svc.ice.ge.com/fleet`

- Append the following filter clause to the end of the URL

  `?filter=name=CSX<customer`

  ```
  http://predix-asset-rc.grc-apps.svc.ice.ge.com/fleet?filter=name=CSX<customer
  ```

- **SEND** the request
- All fleets owned by customer "CSX" are returned


```
{
  "uri": "/fleet/csx-1",
  "name": "CSX Fleet 1",
  "customer": "/customer/csx"
},
{
  "uri": "/fleet/csx-2",
  "name": "CSX Fleet 2",
  "customer": "/customer/csx"
},
{
  "uri": "/fleet/csx-3",
  "name": "CSX Fleet 3",
  "customer": "/customer/csx"
}
```


- Query logic:
  - The query first returns all objects with a name property of CSX
  - From those objects, traverse *backwards* on the customer relationship (this returns another set of objects)
  - From that new set of objects, return only fleet objects