Swordfish API – Developer Documentation

Table of Contents

[1) Requirements: 2](#_Toc516756942)

[2) Overview: 2](#_Toc516756943)

[2.1 Static Resources: 2](#_Toc516756944)

[2.2 Dynamic Resources: 3](#_Toc516756945)

[2.3 Static vs Dynamic Resources: 3](#_Toc516756946)

[2.4 Mockups and templates: 3](#_Toc516756947)

[2.5 Resource\_manager.py file: 3](#_Toc516756948)

[2.6 Utils.py file: 5](#_Toc516756949)

[3) Implementing Dynamic structure 5](#_Toc516756950)

[3.1 Writing Template Files: 6](#_Toc516756951)

[3.2 Writing API Files: 7](#_Toc516756952)

[4) Redfish Emulator :- 9](#_Toc516756953)

[5) Swordfish Functionality: 12](#_Toc516756954)

[5.1 StorageServicesAPI file 13](#_Toc516756955)

# Requirements:

The following are the tested configurations for Linux and Windows environments.

**Python version:** 2.7, 3.1, 3.6. Preferred Python version is 3.6 or higher.

**Ubuntu:** 16.4v

**Windows:** 7,8 and 10

# Overview:

The Swordfish API Emulator can emulate a Swordfish-based system that responds to create, read, update, and delete RESTful API operations to allow developers to model new Swordfish functionality, test clients, demonstrate Swordfish, and do other similar functions. The Swordfish API Emulator extends the DMTF Redfish Interface Emulator by adding code to an installation of the Redfish Interface Emulator code. Review “**Redfish Interface Emulator**” Documentation for more information.

The Redfish Interface Emulator contains Static and Dynamic resources. Static resources are read-only instances created from copies of redfish mockups in ./static directory. Dynamic Resources are modifiable instances whose behavior is defined by creating python files. The python files are an API file that defines the behavior for RESTful operations, and a template file that is used to create initial versions of objects.

The Swordfish Interface Emulator resources are dynamic. It contains the template and API files for Singleton (a single instance of an object) and collection resources. Once the instance files are created for Swordfish, these objects are stored as files in the ./Resource Directory. CRUD operations can be done on these dynamic Resources using REST operations.

## 2.1 Static Resources:

Static resources are in the api\_emulator/redfish/static directory. The Redfish Emulator documentation has instructions for static mockups that can be found in Redfish Interface Emulator docs folder or the Readme file.

## 2.2 Dynamic Resources:

Dynamic resources can have following properties:

* It can create resources and sub-resources using template structure via POST operation.
* Can change the properties of a specific resource.
* GET, POST, PUT & DELETE operations are used.

## 2.3 Static vs Dynamic Resources:

Static resources are populated by placing appropriate JSON mockup folders into the ./static directory. Static resources are read-only objects that cannot be altered (e.g., via PATCH / PUT). If the desired emulator interaction use cases include any modifications to the systems, dynamic resources are required.

Dynamic Resources in emulator can be populated and manipulated through APIs using CRUD operations. Any desired manipulations or desired system behavior may be instrumented in the underlying python API and template files.

## 2.4 Mockups and templates:

Mockups are just an example view or static view of possible configuration. Mockups will simply provide the overview of all swordfish models.

Template is a base structure for creating new element in a collection / Sub Collection. We can create members of collection using template structure. Template structure is designed based on latest SNIA Swordfish latest Schema V1.0.5

Template file divides in to two types of sections.one is “template declaration” and other is “function”

## 2.5 Resource\_manager.py file:

* The redfish root structure is defined in Resourcemanager.py file. So now developer has to make some suggested changes to run swordfish emulator.
* Redfish emulator already contains static and dynamic resources in resourcemanager.py file. It already contains static resources from ./redfish/static directory which is similar/copy to mockup data.
* ResourceManager class can load static resources and dynamic resources.
* Swordfish developerhas to attach API for dynamic resources.
* All the API files are stored in ./api\_emulator/swordfish directory.

Fig A: Imports all the api files

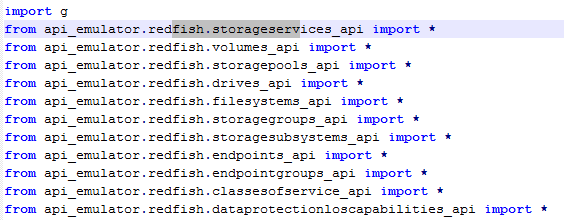
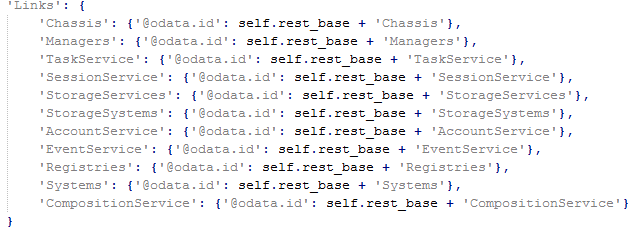


Fig B: Attaching API for dynamic resources



Fig C: Adding Collections to

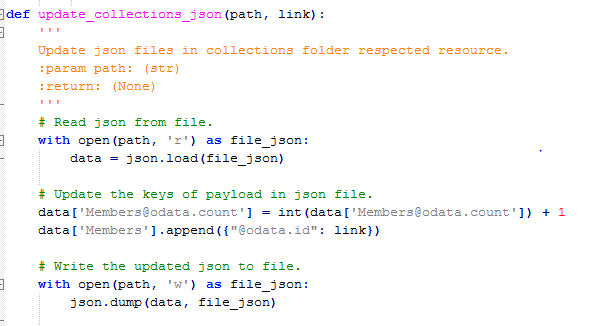


## 2.6 Utils.py file:

The Redfish Interface Emulator file utils.py file contains a small set of support functions. The Swordfish emulator adds new functions to this file:

A new function “Update\_collections\_json()” is added, which will update the number of collections if any new item is added or deleted to collection.

For Example: Volumes have 4 child items in members. If developer/user wants to add new item to volume collection, automatically members count will be increased by one.



# Implementing Dynamic structure

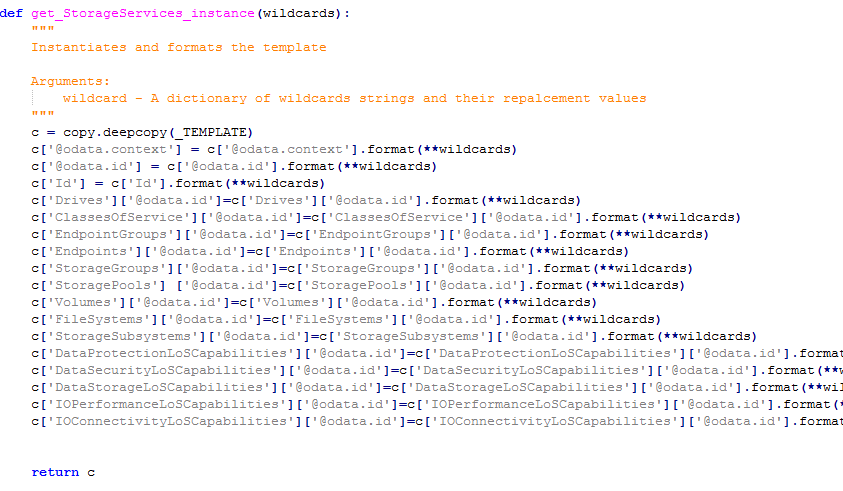
* The static resources are in /api\_emulator/redfish/static/.. Folder, these are just a simple copy.
* The dynamic resource is in /Resource/...Folder. We can specify any path to declare the dynamic resources.
* Developer has to create a template file and api files for dynamic structure.
* The file in template directory and api directory are to be paired appropriately.

## 3.1 Writing Template Files:

* Template file directory is /api\_emulator/swordfish/template
* Template declaration is as similar to json file (shown in below figure). All the properties for a particular resource are to be added by referring the updated schemas released by SNIA-Swordfish.
* Template declaration also contains substitution fields
* Each template contains similar fields {rb}&{id} which are declared in below function are replaced by root\_base(/redfish/v1) and ID .

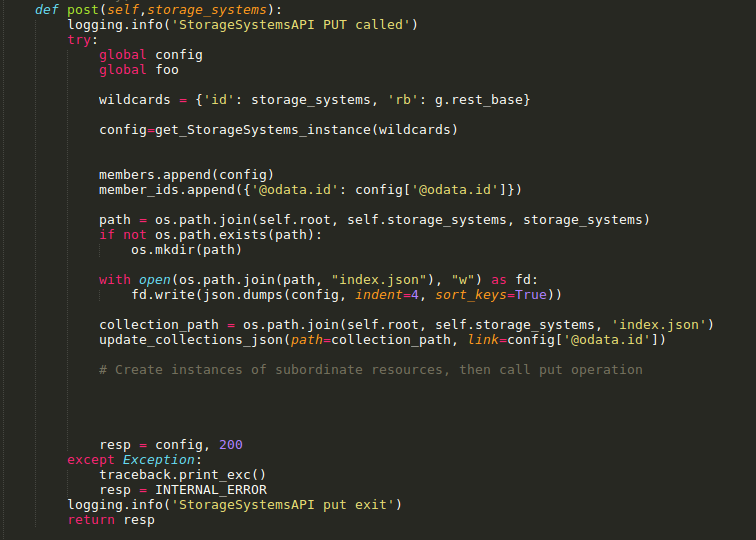


* Below figure contains function, which is used to call the instance of resource.
* We are using wildcards for replacing the values.
* Using deep-copy is to make copy of template declaration.
* {rb} –{rest-base} is default set to {redfish/v1}

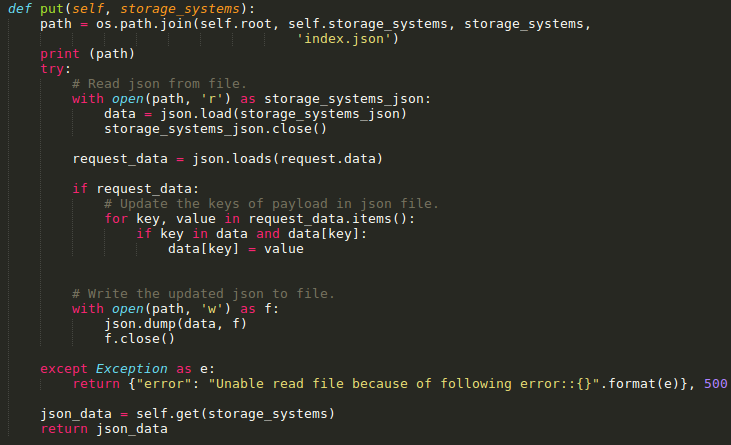


## 3.2 Writing API Files:

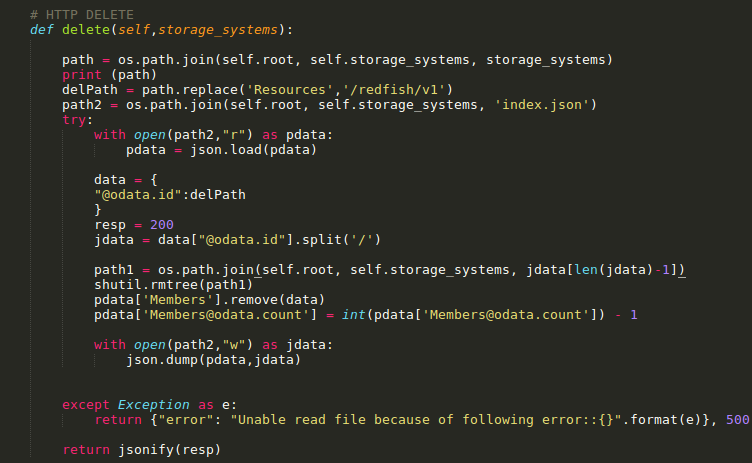
* We have to import the above template function to add a new element in to collection.
* The API file commonly used to retrieve data form remote systems. Using API, we can make request to web server and retrieve data that is needed. In emulator APIcode will interacting with all collections and member resources.
* For all Collection-API’s , GET and POST methods are defined, post command will create a sub resource API to the interface.
* For all API’S there are API classes and collection classes.
* The http functions GET, POST, PUT and DELETE are declared based on requirements.
* The Post function can create resources ,update members and members.id
* Post function can also attach API’s of sub resources and creates instance of sub-resources.



* PUT function is used to edit specific properties in a collection.



* Delete Function will try to delete a item from members list . after deleting resource , it will update members and members count in collection.



# Redfish Emulator :-

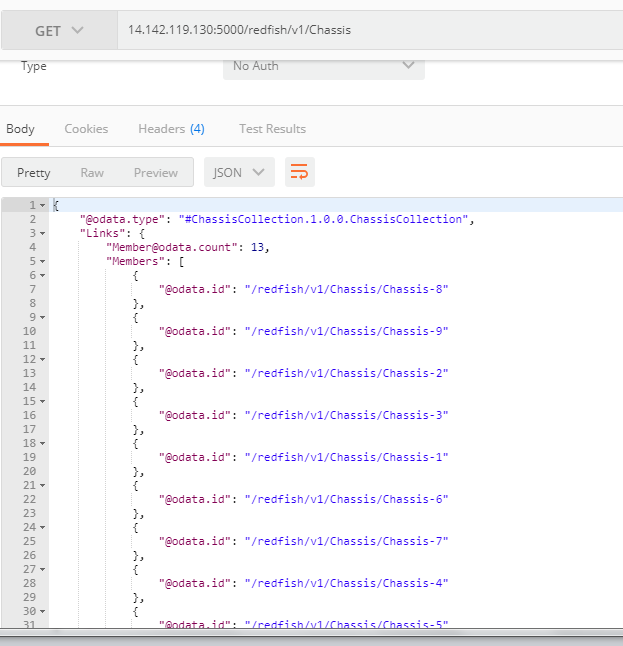
In Redfish Emulator dynamic files are stored in memory. Redfish emulator loads all the static and dynamic files. Emulator used to update and delete an item from collection by passing “id”. After making all changes if redfish used to restart its emulator, the data in memory will be lost and it comes with a new version of redfish.

Redfish automatically creates Template files and API files using “codegen”

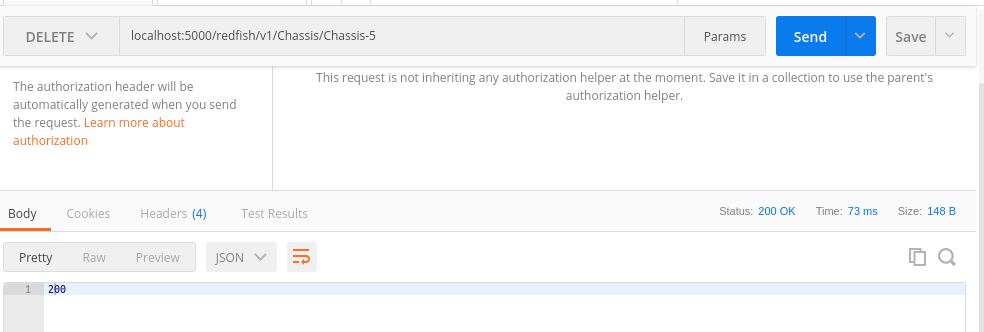
Whenever we start emulator, Redfish used to automate template and api files

Example:

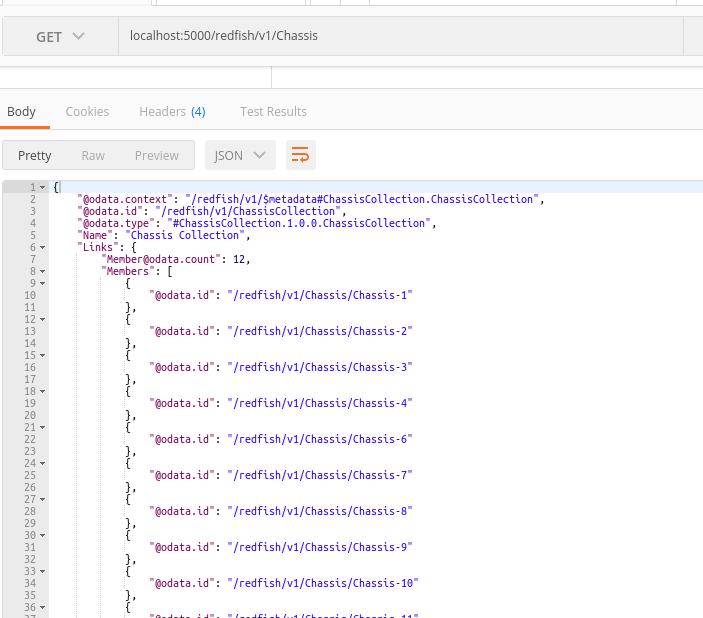
Below figure shows the main collection of chassis, which contains “13” items in a collection. We can go through each member of collection individually. We can update the collection and delete a collection.



The below figure shows deleting “Chassis-5” from collection



From below figure members and member count are updated. Chassis-5 is not in members list.



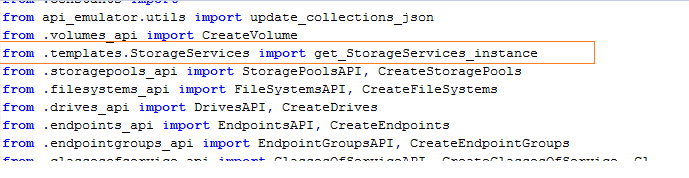
But if we try to restart the emulator, It automatically creates chassis member “chassis-5 “

As it is storing in memory it won’t save the previous updated data. It stores in cache and it automatically re-collects everything form latest changes.

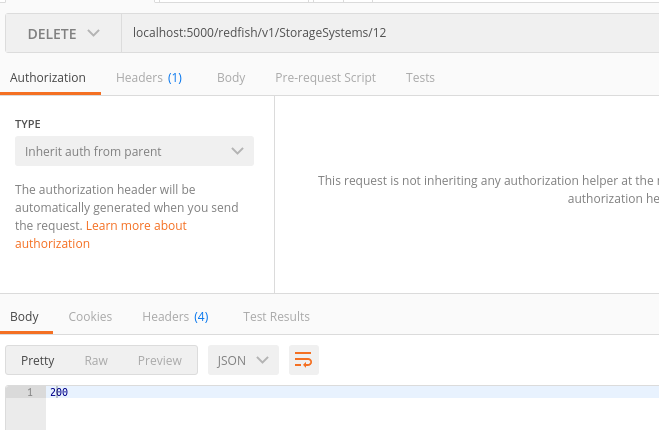
# Swordfish Functionality:

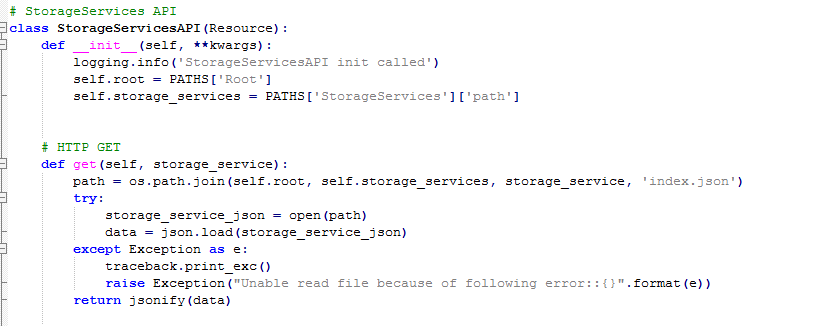
Swordfish Emulator works as an extension of the Redfish emulator. The main functionality of swordfish emulator is to present dynamic resources which are structured in template files defined in the “/template” directory. With the help of these template files we can add new items to collections. Below are the steps to describe how functionality works in the Swordfish emulator.

* For Example, If we want to create StorageServices as a dynamic resource using template and API files.
* StorageServices is a member of collection. We have to create template and API files with similar name.
  + Template file path is /api\_emulator/swordfish/template/StorageServices.py
  + API file path /api\_emulator/swordfish/storageservices\_api.py
* **Template** file contains two sections as shown in figure.
  + Template declaration
  + Function ("get\_StorageServices\_instance")



* Storageservices\_api.py file contains two different classes: the API class and the collection class.
* StorageServices collectionAPI file
* By using url (/redfish/v1/StorageServices) we can “GET” members and collection data
* We can delete an item from collection using “Delete”.



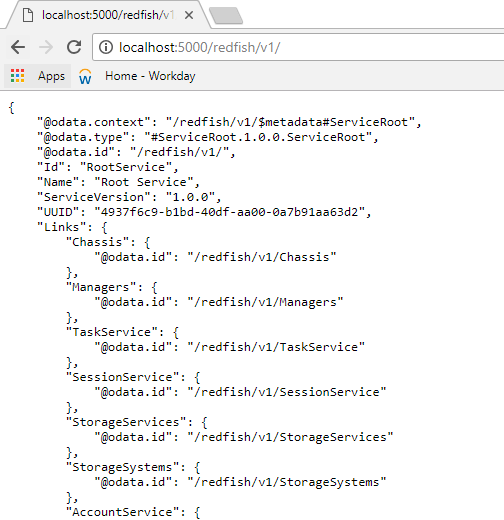


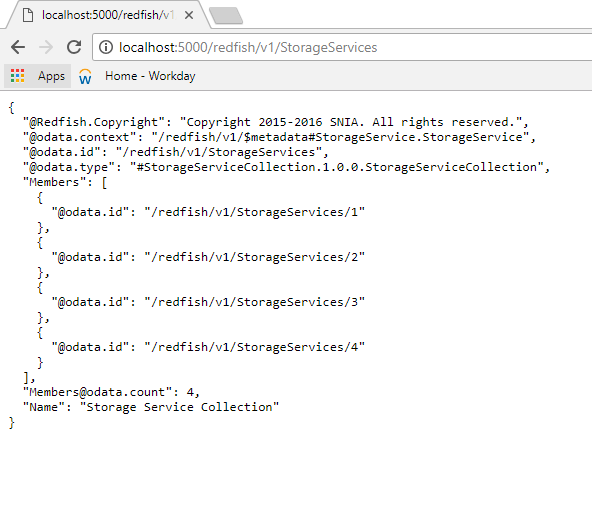
## 5.1 StorageServicesAPI file

* + Add new elements of a collection using “post”. By calling the template file we can create a base structure for new element.

Example:

Go through StorageServices resource by following steps.



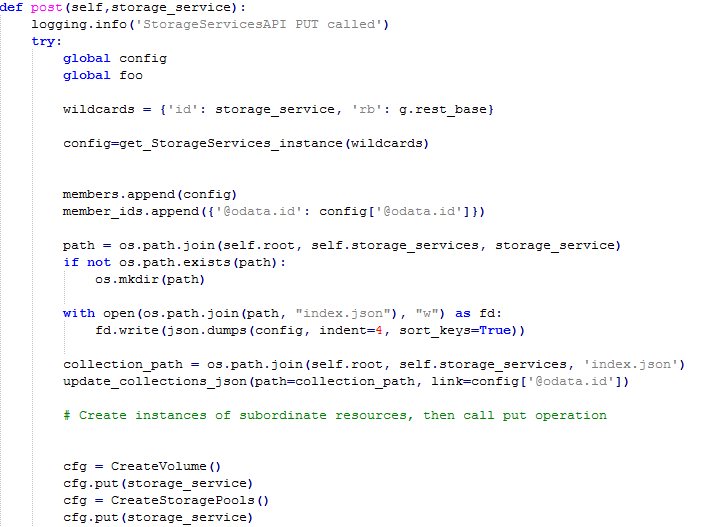


Now we are in **/redfish/v1/storageservices**, here the above figure shows that there are 4 members created in storageservices. If we want to create more members we have to use “post” function.

The subresources are created at the time of member creation by writing logic in “post” Function as shown in the figure below.



The logic of the above output is show in below figure.



We are creating by passing “post” method. We are creating instance of subresources using post operation.

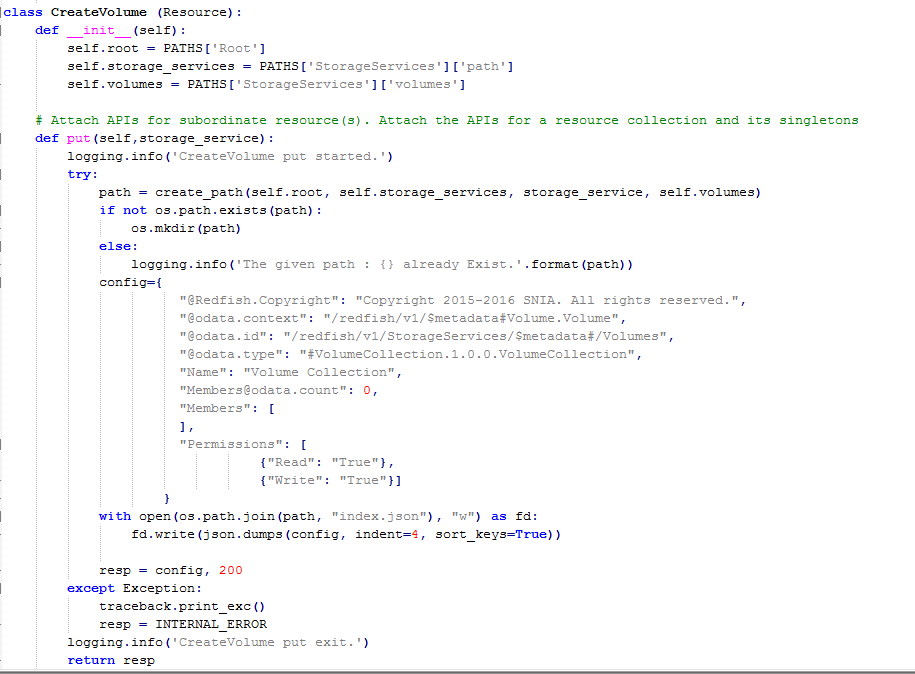
The above process is to create a new resource StorageServices. It stored in root directory.

Whenever we create a new resource , it will automatically creates a folder and subfolder.

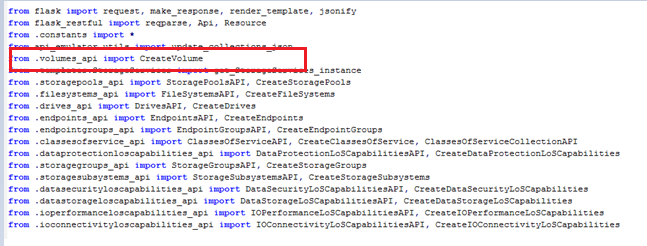
We are creating theses with the help of template files.

Example : we are creating volume as a sub-resource in storageServices.

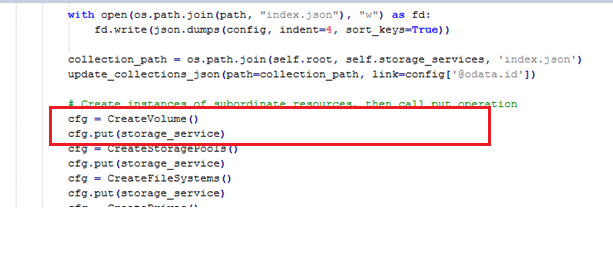
**Step1**:create a class” createvolume ()” in volume\_api.py file



**Step2**:- Importing volumes\_api file to storageservices\_api file



**Step3**:-create instance of volume and then call put operation.



So based on the above code StorageServices, a developer can create resources and sub-resources using template structure.