**Autogen Tool**: For resource implementation of all RF/SF resources

* Library requirement:
* xmltodict
* psutil
* urllib
* requests
* To generate \*\_api.py file for all the resources, run the below command:

python script\_api.py [schema\_path]

[schema\_path] should be the path of the XML schema directory

* To generate a \*.py template file for all the resources, run the following command:

python script\_template.py [xml\_schema\_path] [json\_schema\_path]

[xml\_schema\_path] should be the path of the XML schema directory.

[json\_schema\_path] should be the path of the JSON schema directory.

* To generate resource implementation for a specific resource, use the following programs:
* generate\_api.py
  + This program will create API implementation code for the resource in the 'APIs' folder.
  + Input to this program would be the XML schema URL for the resource.
* generate\_service\_api.py
  + This program will create API implementation code for the service and/or subservice in the 'Service APIs' folder.
  + Input to this program would be XML schema URL for the service and/or subservice.
* To generate a template file for each individual resource, run the following program:
* generate\_template.py
  + This program will generate template code for the resource in the 'Templates' folder.
  + Input to this program would be JSON and XML schema URLs for the resource.
* This tool also generates add\_resource/add\_service\_resource and add\_import files.
* add\_resource/add\_service\_resource contains "g.api.add\_resource" statements for all implemented resource.
* add\_import contains import statements for generated classes

You can copy the contents from these files to the emulator's resource\_manager.py program.

* Notes:
* Autogen would do resource implementation of all the resources for those having "*Redfish.Uris*", with <resource\*>\_api.py file for each of the URI paths.
* For implementation for resource instance path will get added in the ‘Service APIs’ folder.
* Every \*\_api.py file contains “Program name” and “Resource implementation for” which is a URI of the resource.

**Emulator changes:**

* Library requirements:
* flask
* flask\_restful
* pyjwt – to encode/decode web token used for authentication
* emulator-config.json file new parameters:
* "AUTHENTICATION"
  + Used for authentication service
  + It should be either “Enable” or “Disable”.
  + If enabled, the user must provide either basic authentication or session authentication. Also, the user should run HTTPS requests while querying any resource.
  + If disabled, then run the emulator without any authentication.
* "CERTIFICATE"
  + To run HTTPS service, the emulator requires an HTTPS certificate file and private key file.
  + Follow the steps to create the certificate file and private key file given in “Self signed certificate and key generation guide.docx”
  + Give both file names as array values for the “CERTIFICATE” parameter, first the certificate file name and then the private key file name.
* Authentication implementation:
* Basic Authentication
  + Provide username and password as login credentials.
  + Username value should be “Administrator” or “User”
  + Password value should be “Password” for both usernames.
* Session Authentication
  + Create a session instance with POST to /SessionService/Session/
  + Provide username and password in the post body. Both would have values similar to Basic Authentication
  + After a successful post to the session, note the “X-Auth-Token” and “Location” header values from the response header.
  + Add “X-Auth-Token” to subsequent request headers.
  + To delete the session run, DELETE on the “Location” header value.
* g.py file changes: To encode/decode the authentication token, a secret key is required which is initialized in g.py. Hence to run the emulator with a session, make sure to have an updated g.py file.
* Serviceroot index.json updated with AccountService path details.