Sample_Connector to Giotto

Introduction:

This is documentation for the connectors of Giotto. Giotto is based on Building Depot, with additional features, and the sample_connectors provided here will work for Giotto. Giotto is essentially an Extensible and Distributed Architecture for Sensor Data Storage, Access and Sharing. Connect_bd.py connects the sensors and devices to read, write and update the sensor data in the Giotto.Sample Connectors for various IOT devices like Wemo, Lifx, Netatmo, Sense mother and Nest to Giotto are present at https://github.com/IoT-Expedition/Connectors.

Working:

The connect_bd.py accepts json object from the Device connector programs which provides the Device data to be updated to Giotto. All the major functions are member of the class BD_connect except get_json(). The functions:

- get_json():
  - Accepts the Json object (Format of Json object given below) from the sensor program which verifies the ClientID, ClientKey and Device uuid and calls the create_sensor() or rest_post() based on the information that device is present or not.
- rest_post():
  - Makes a REST API call to update the sensor and all its sensor point's time series data.
- create_sensor():
  - Creates new sensors and its respective sensor points if not present.

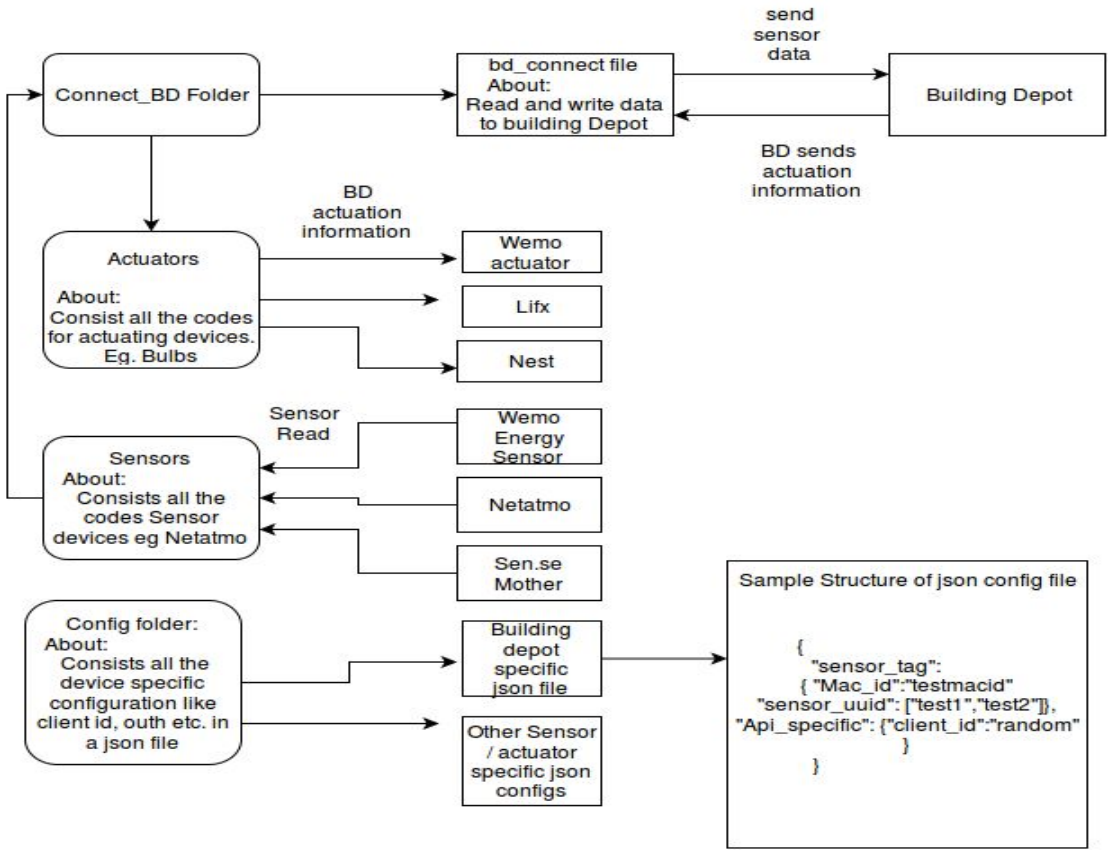

Fig 1:Folder stucture of the sample BD connector

Guidelines to use the Device - connector:

- The configuration of the Giotto should be specified in bd_setting.json and the config information related to the device must be stored in the “<devicename>”.json which should present under config folder.
- If the device is both a Sensor and Actuator then the Connector Program should contain two classes one for actuation and the other for sensing.
- The Device Connector actuator class obtains the actuation command from the connector as required by the device.
- The Device connector sensor class collects the sensor information from the sensor and send it as a Json object to the connect_bd.py. The Sensor Connector should send a Json object to the connect_bd.py in the format:
    - Syntax:
        - { "sensor_data": { <all sensor data> } }
        - Eg:

          Data= {“sensor_data”: {
          “status”=on,
          “energy”=0.5w
          },
          }
    - “sensor_data” should contain all the sensor read information from the sensor.

Steps on how to write a Simple Device Connector Using the existing APIs:

1. **Obtain the Access Token:**
   Every query to Giotto has to be authenticated by an access token. The client id and secret key required to generate the access token can be obtained after logging into the DataService.Each access token is valid for 24 hours from the time of generation. The Client Id and Secret_key are obtained from the UI at http://www.example.com:81/central/oauth_gen.

   Sample request:
   GET /oauth/access_token/client_id=<client_id>/client_secret=<client_secret>
   Sample response:
   {"access_token": "528d58481bc728a5eb57e73a49ba4539"}

2. **Create a Sensor/Device Point to post time series data:**
   POST /api/sensor
   header = {"Authorization": "bearer " + oauth_token, 'content-type': 'application/json'}

   JSON Parameters:
    - name (*string*) – Name of the sensor
    - identifier (*string*) – An identifier that will be associated with the sensor
    - building (*string*) – Building in which the sensor is located

Returns:
- success (string) – Returns ‘True’ if data is posted successfully otherwise ‘False’
- uuid (string) – Returns the uuid of the sensor on successful creation

Status Codes:
- 200 OK – Success
- 401 Unauthorized – Unauthorized Credentials (See HTTP 401)

3. **Post Time Series Data :**
   Within a single POST request data can be posted to multiple sensor points. The format for each sensor point in the list should be as follows.Base URL: http://www.example.com:82/
   POST /api/sensor/timeseries

   JSON Parameters:

   - **sensor_id** (*string*) – UUID of the sensor to which data has to be posted
   - **samples** (*list*) –
   - A list of the data points that have to be added to the time-series of the sensor point given by sensor_id. Each item in the list has to be of the following format: {“time”: A unix timestamp of a sampling,“value”: A sensor value}

Returns:
- **success** (string) – Returns ‘True’ if data is posted successfully otherwise ‘False’

Status Codes:
- 200 OK – Success
- 401 Unauthorized – Unauthorized Credentials (See HTTP 401)

4. **Read Time Series Data:**
   This retrieves a list of datapoints for the timeseries of the specified Sensorpoint. Base URL: http://www.example.com:82/
   GET /sensor/<sensor-uuid>/timeseries?start_time=<start_timestamp>&end_time=<end_timestamp>&resolution=<resolution_units>
   Parameters:
   - **sensor-uuid** (*string*) – UUID associated with Sensor (compulsory)
   - **start_time** (*integer*) – The starting point of time from which the timeseries data of this sensor point is desired. Has to be a UNIX timestamp. (compulsory)
   - **end_time** (*integer*) – The ending point of time till which the timeseries data of this sensor point is desired. Has to be a UNIX timestamp.(compulsory)
   - **resolution** (*string*) – The resolution of the data required. If not specified will retrieve all the datapoints over the specified interval. Has to be specified in the format time units as an integer + unit identifier e.g. 10s,1m,1h etc. (optional)

   Returns:

- **success** (string) – Returns ‘True’ if data is retrieved successfully otherwise ‘False’
- **data** (struct) – Contains the series
    - **series** (list) – Contains the timeseries data, uuid of the sensor and the column names for the timeseries data
    - **columns** (list) – Contains the names of the columns of the data that is present in the timeseries
    - **name** (string) – uuid of the sensor whose data is being retrieved
    - **values** (list) – Contains the list of timeseries data that has been requested in the order represented by the columns.

Status Codes:

- 200 OK – Success
- 401 Unauthorized – Unauthorized Credentials (See HTTP 401).

5. **Register App:**
   /api/apps - POST,GET

   Request format:
   The following JSON data has to be sent along with the request
   {“email":"synergy@gmail.com"}

   POST request will return the queue id, GET request will return list of applications registered for this user currently

6. **Subscribing to Sensor/Actuator:**
   /api/apps/subscription - POST,DELETE

   Request format:
   The following JSON data has to be sent along with the request
   {“email”:”synergy@gmail.com","app":"<queue-id>", “sensor”:"<sensor-uuid>"}

   POST request will create the binding between the master exchange and the application queue and DELETE request will delete the binding. Both requests will return True upon success and False and error message upon failure.

7. **Sample python code to listen to a queue:**

```
#!/usr/bin/env python
import pika
import sys
connection = pika.BlockingConnection(pika.ConnectionParameters(
        host='localhost'))
channel = connection.channel()
channel.exchange_declare(exchange='master_exchange', type='direct')
print(' [*] Waiting for logs. To exit press CTRL+C')
def callback(ch, method, properties, body):
    print(" [x] %r:%r" % (method.routing_key, body))
channel.basic_consume(callback,
                      queue="<queue-id>",
                      no_ack=True)
```

```
channel.start_consuming()
```