Python - Vert.x TCP Eventbus Client

Make the environment

You can download the library for python here. Then unzip it. Within that folder there is a folder called “Eventbus”. It’s our python module. There is another folder “simple example” which includes the example that we are going to discuss.

Let’s get the “client” folder of the “simple example”. Add Eventbus module into it.
It’s our client.
Then take the “server” folder of the “simple example”. It’s our server.

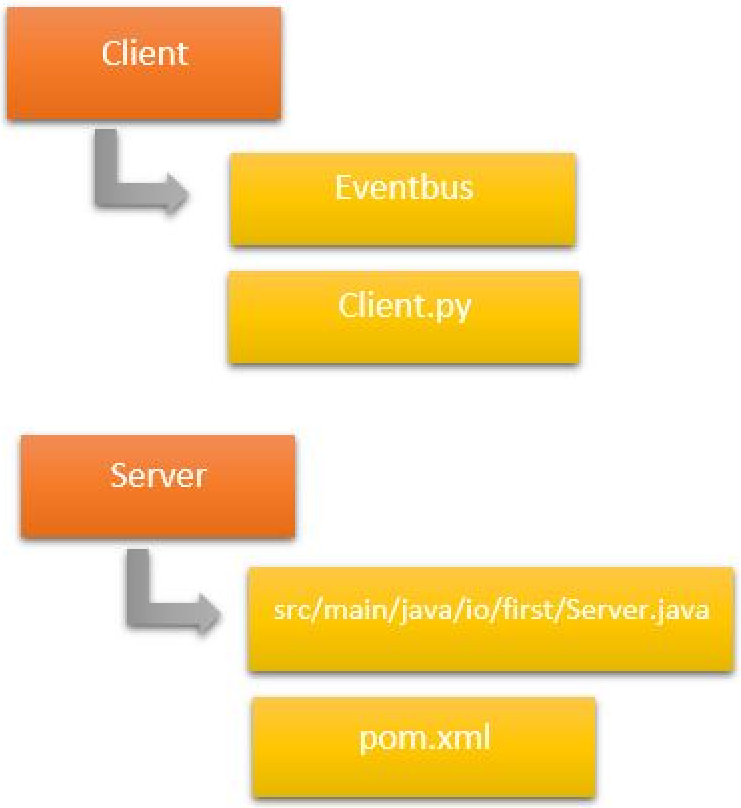


Figure 1- File structure

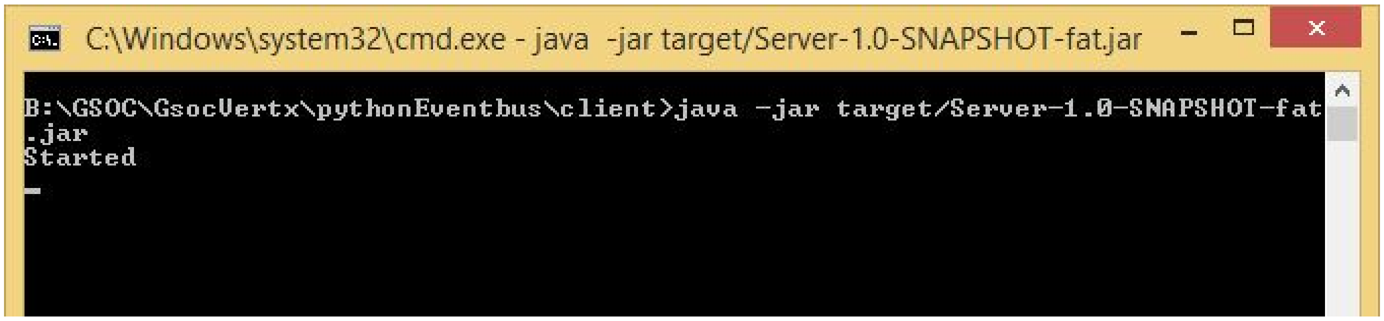
I assume you have installed maven and vert.x 3.2.1. If not,
Download Vertx from here

This provides a quick guide to install vert.x here

Run Server

Then open terminal in “server” folder.

1) **Build the package**
   >> mvn clean package
2) **run the server**
   >>java -jar target/Server-1.0-SNAPSHOT-fat.jar



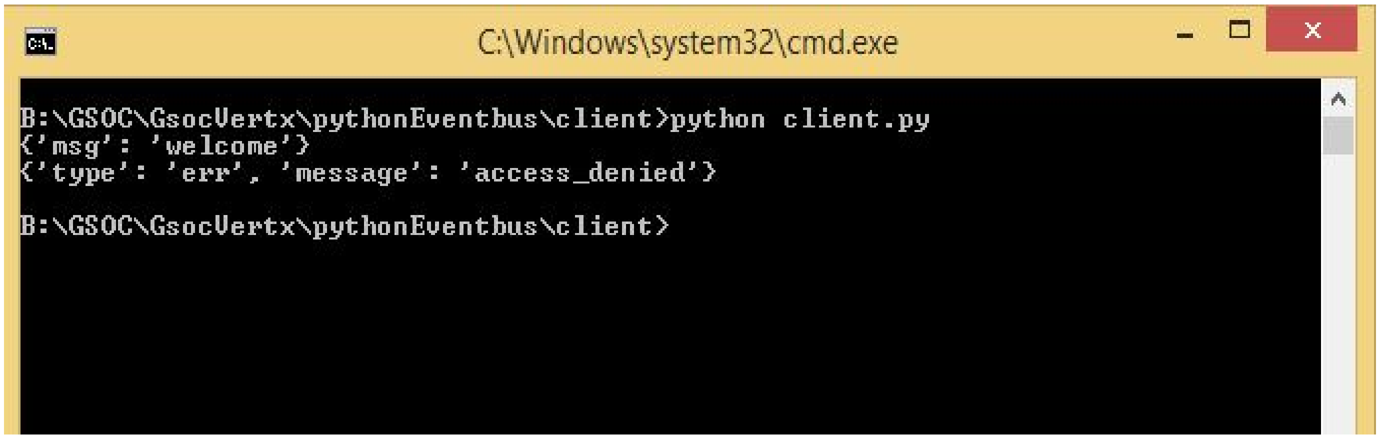
C:\Windows\system32\cmd.exe - java -jar target/Server-1.0-SNAPSHOT-fat.jar

```
B:\GSOC\GsocVertx\pythonEventbus\client>java -jar target/Server-1.0-SNAPSHOT-fat
.jar
Started
_
```

Figure 2- Server started

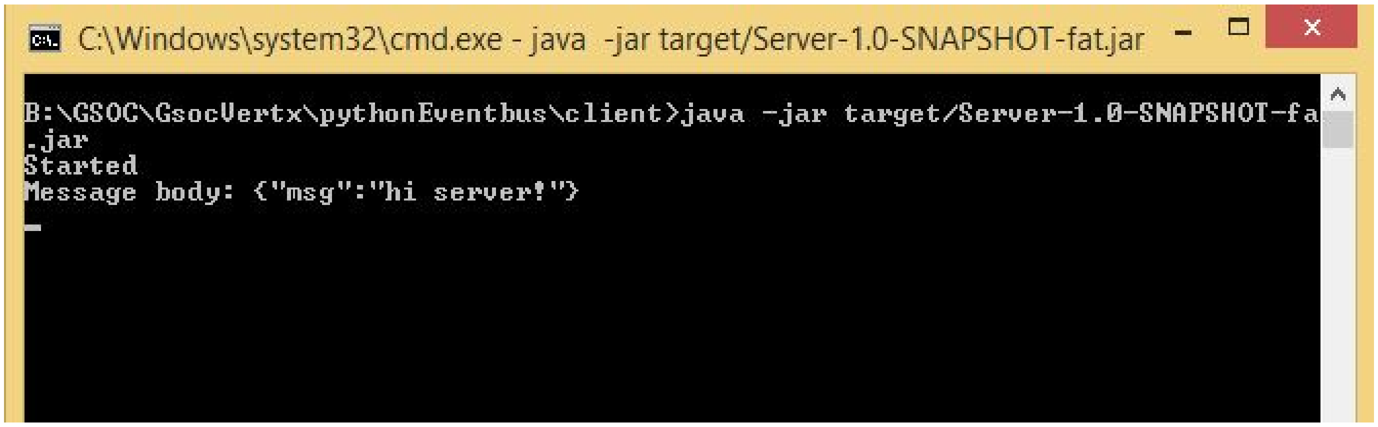
Run Client

Then open terminal in “client” folder.
>>python Client.py



```
B:\GSOC\GsocVertx\pythonEventbus\client>python client.py
{'msg': 'welcome'}
{'type': 'err', 'message': 'access_denied'}

B:\GSOC\GsocVertx\pythonEventbus\client>
```

Figure 3 - Client running

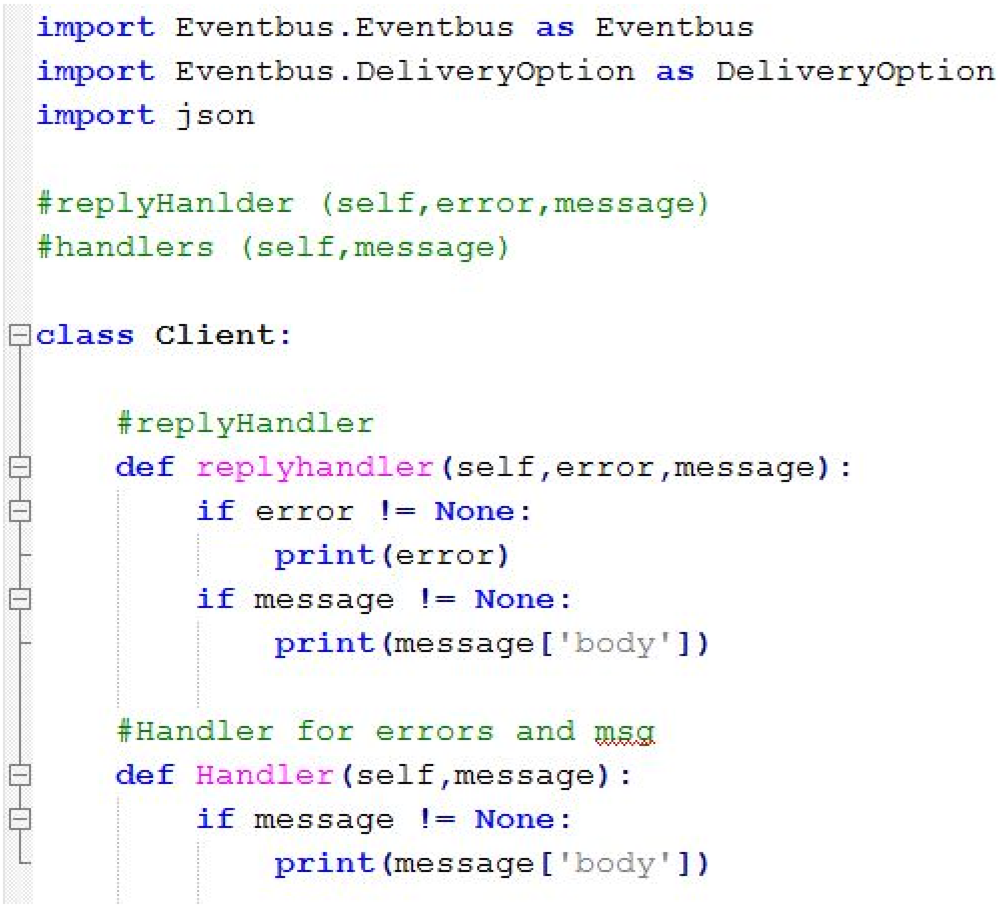

C:\Windows\system32\cmd.exe - java -jar target/Server-1.0-SNAPSHOT-fat.jar

```
B:\GSOC\GsocVertx\pythonEventbus\client>java -jar target/Server-1.0-SNAPSHOT-fa
.jar
Started
Message body: {"msg":"hi server!"}
_
```

Figure 3 - Server running

If you could run examples then you done. Let’s discuss the client and server examples

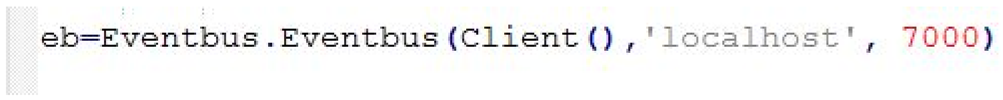
Client class



```python
import Eventbus.Eventbus as Eventbus
import Eventbus.DeliveryOption as DeliveryOption
import json

#replyHanlder (self,error,message)
#handlers (self,message)

class Client:

    #replyHandler
    def replyhandler(self,error,message):
        if error != None:
            print(error)
        if message != None:
            print(message['body'])

    #Handler for errors and msg
    def Handler(self,message):
        if message != None:
            print(message['body'])
```

Main thread

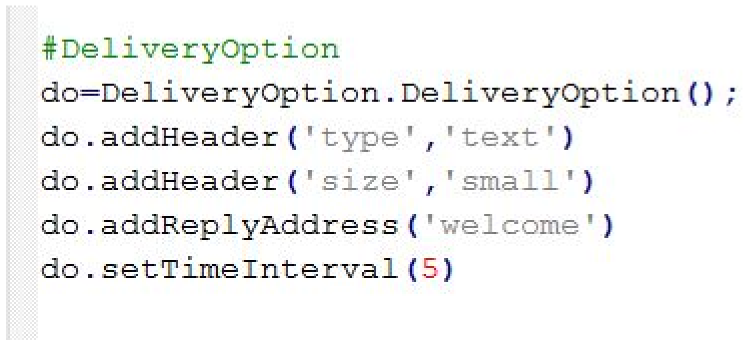
Create Eventbus



```
eb=Eventbus.Eventbus(Client(),'localhost', 7000)
```

Eventbus(instance, host, port)
Later we are going to use methods of Client class. Because of that we should pass it as a parameter. Instance is an essential parameter. Default values of host and port are localhost and 7000.

DeliveryOption



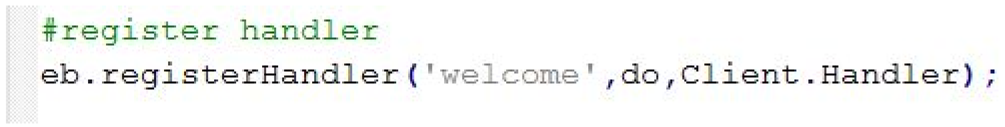
```
#DeliveryOption
do=DeliveryOption.DeliveryOption();
do.addHeader('type','text')
do.addHeader('size','small')
do.addReplyAddress('welcome')
do.setTimeInterval(5)
```

DeliveryOption object is holding few important parameters,

1) Headers
2) Reply address
3) Time interval

Headers
headers can be added to a message using a DeliveryOption object.
Reply address
Reply address of the message can be added to a message using a DeliveryOption object.
Time Interval
After that time if a reply message did not get, reply handler will be called with a timeout error. We will discuss more about it later.

Registering a handler



```
#register handler
eb.registerHandler('welcome',do,Client.Handler);
```

registerHandler(address, DeliveryOption, Handler)
After registration any message that comes to the address will be routed to the Handler.

Handlers
Handlers are the functions that take only one argument. Function name does not matter.

Eg: Handler(message), add(msg)
The message is a dictionary object.

Send

```
#jsonObject -body
body={'msg':'hi server!',}
```

```
#send
eb.send('welcome',body,do)
```

Body can be defined as a dictionary object.

send(address, body, DeliveryOption)
The address and the body are essential parameters.

```
#send
eb.send('ignore',body,do,Client.replyhandler)
```

send(address, body, DeliveryOption,replyHandler)

Reply Handlers

Reply handlers are the functions that take only 2 arguments. Function name does not matter.
Eg: replyHandler(error, message), testReply(err, msg)
The message and error are dictionary objects.

If a replyHandler has been set, the program will wait for a message DeliveryOption.timeInterval period. If a reply message received, program will execute next line and message will be routed to the replyHandler with a null error. If not replyHandler will be called with a “TIMEOUT” error.

Close socket

```
#close after 5 seconds
eb.closeConnection(5)
```

Finally, socket must be closed.
closeConnection(timeInterval)
Socket will be close automatically after the timeinterval. You can wait for other messages in this period. Default timeInterval is 30 sec.

There is another example “TimeKeeper”. Check it for more information.