NOOXY Service Framework

Started by Yves Chen, 10, Mar, 2018

Document Overview

Orientation

NOOXY

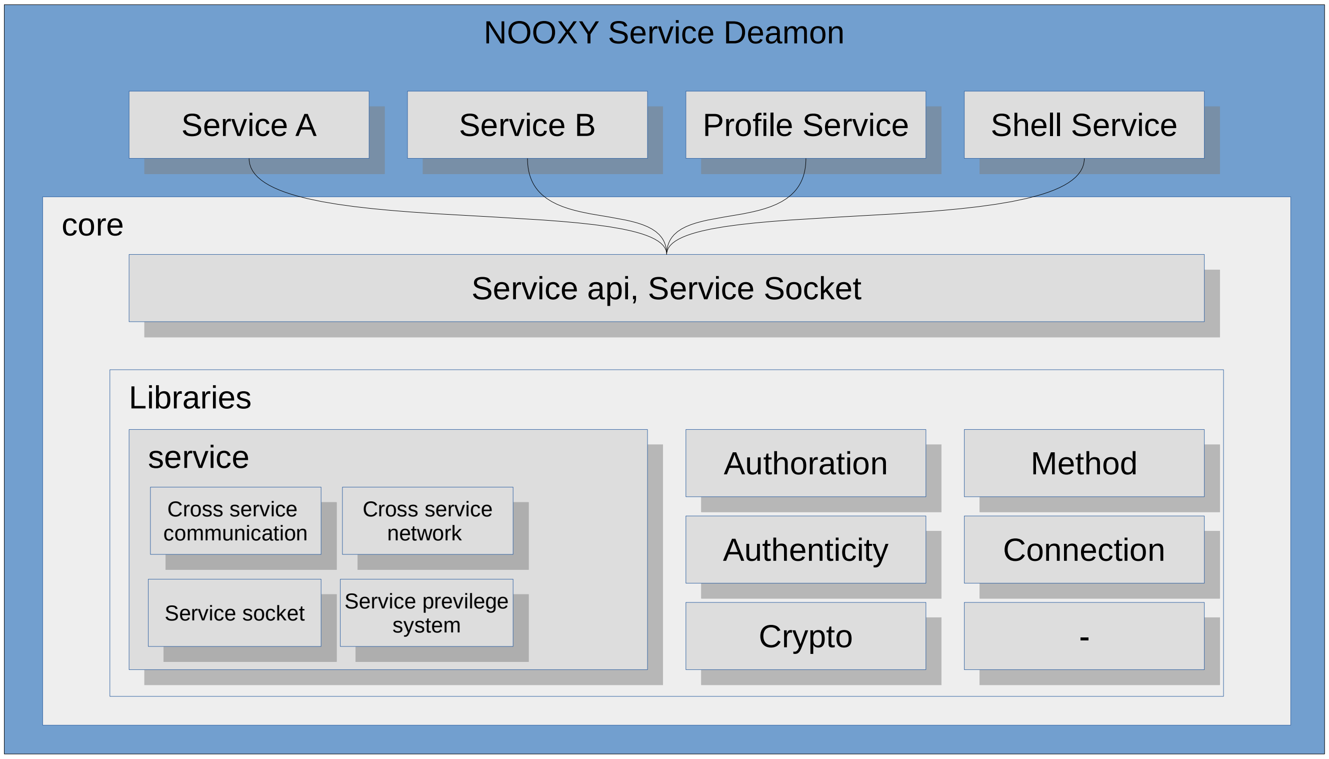
NOOXY Service Framework Orientation

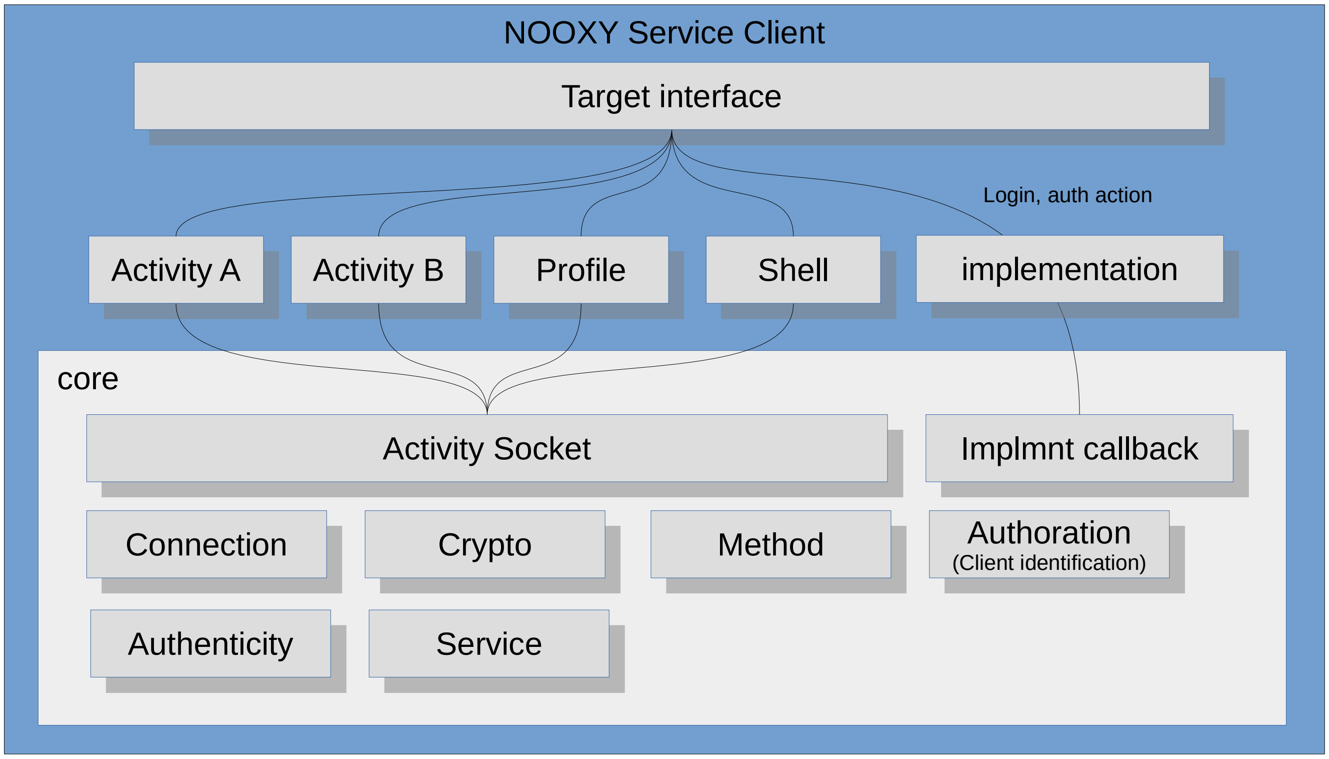
1. User Orientation
2. Server client structure
3. Authoriation system
4. Modurable(base on service)
5. lightweight
6. “Everything based on service” sturcture

Architecture

NOOXY

NOOXY Service Framework Architecture



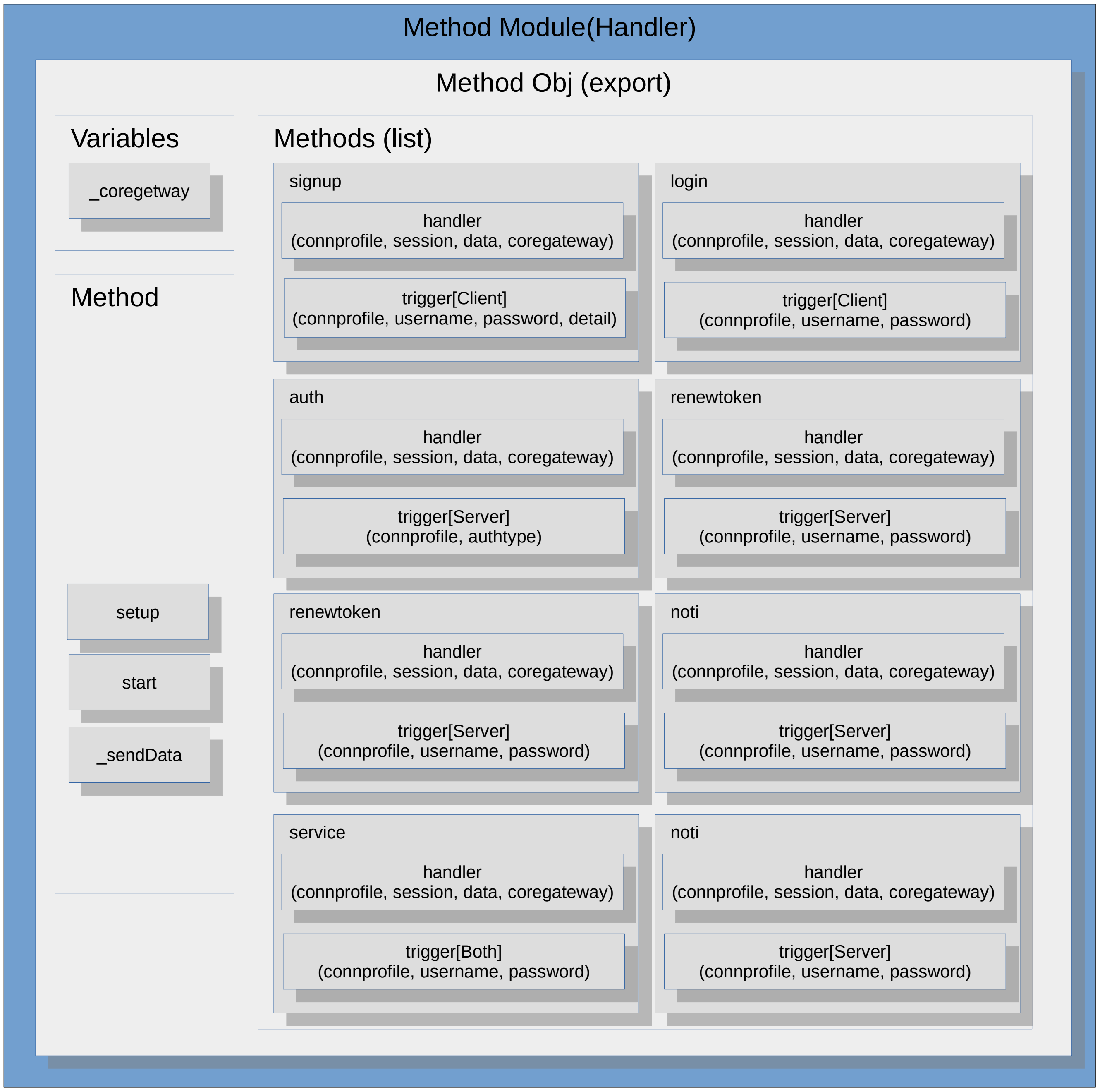


Serverside modure

Method Module(Handler)

Objective: A parser or a router. To pharse json between connetion. And switch, and trigger between different operations.

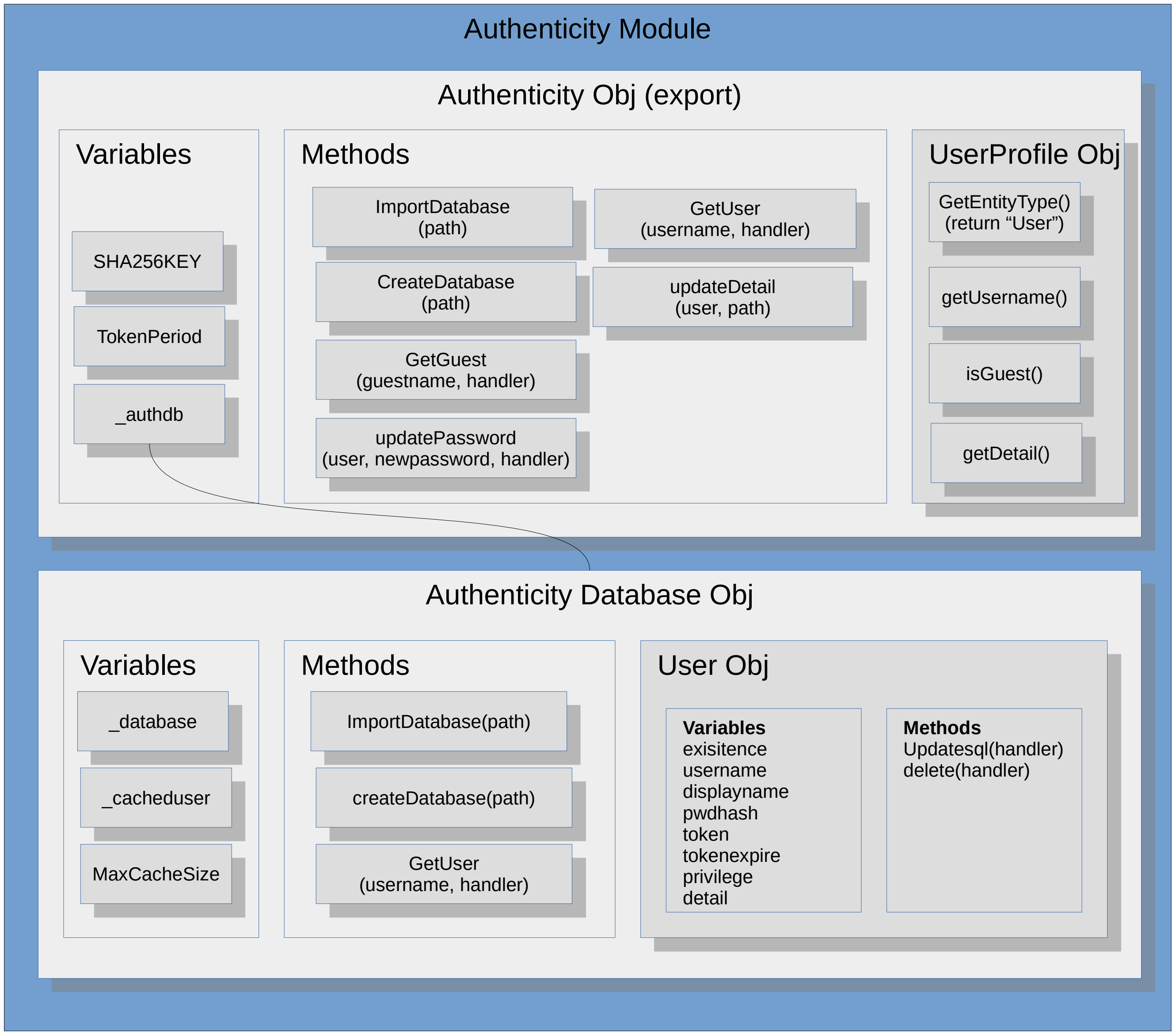
Figure:


Method Module(Handler)
Method Obj (export)
Variables
_coregetway
Method
setup
start
_sendData
Methods (list)
signup
handler
(connprofile, session, data, coregateway)
trigger[Client]
(connprofile, username, password, detail)
login
handler
(connprofile, session, data, coregateway)
trigger[Client]
(connprofile, username, password)
auth
handler
(connprofile, session, data, coregateway)
trigger[Server]
(connprofile, authtype)
renewtoken
handler
(connprofile, session, data, coregateway)
trigger[Server]
(connprofile, username, password)
renewtoken
handler
(connprofile, session, data, coregateway)
trigger[Server]
(connprofile, username, password)
noti
handler
(connprofile, session, data, coregateway)
trigger[Server]
(connprofile, username, password)
service
handler
(connprofile, session, data, coregateway)
trigger[Both]
(connprofile, username, password)
noti
handler
(connprofile, session, data, coregateway)
trigger[Server]
(connprofile, username, password)

Authenticity Module

Objective: To interact with database, Providing Users Obj cahcing, Creating User Obj, User identification.

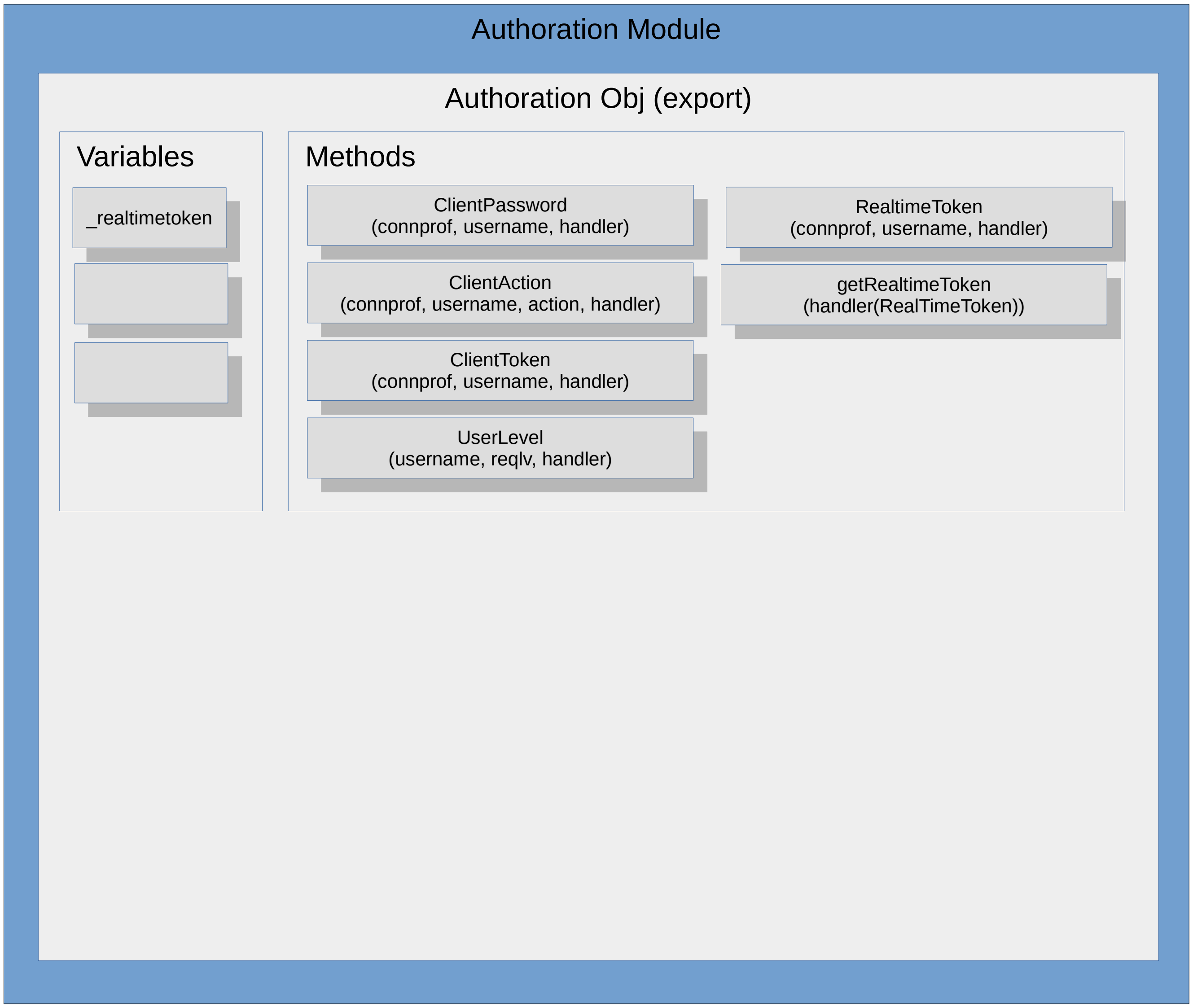
Figure:



Authoration Module

Objective: To provide function to take authorative actions. Confirming the sensitive data or opearation is permitted.

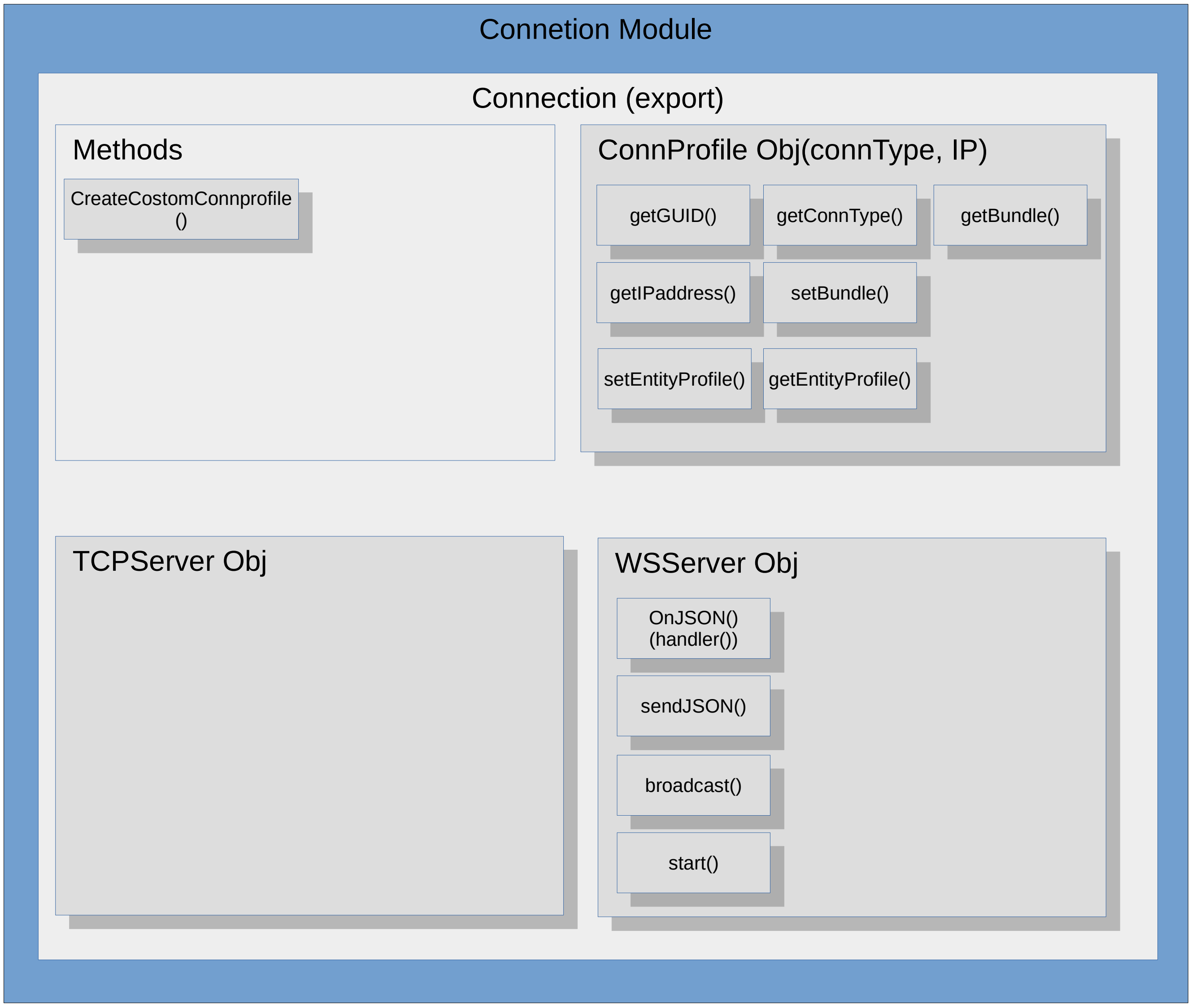
Figure:


Authoration Module
Authoration Obj (export)
Variables
_realtimetoken
Methods
ClientPassword (connprof, username, handler)
ClientAction (connprof, username, action, handler)
ClientToken (connprof, username, handler)
UserLevel (username, reqlv, handler)
RealtimeToken (connprof, username, handler)
getRealtimeToken (handler(RealTimeToken))

Connetion Module

Objective: Create a interface to get communication with remote device.

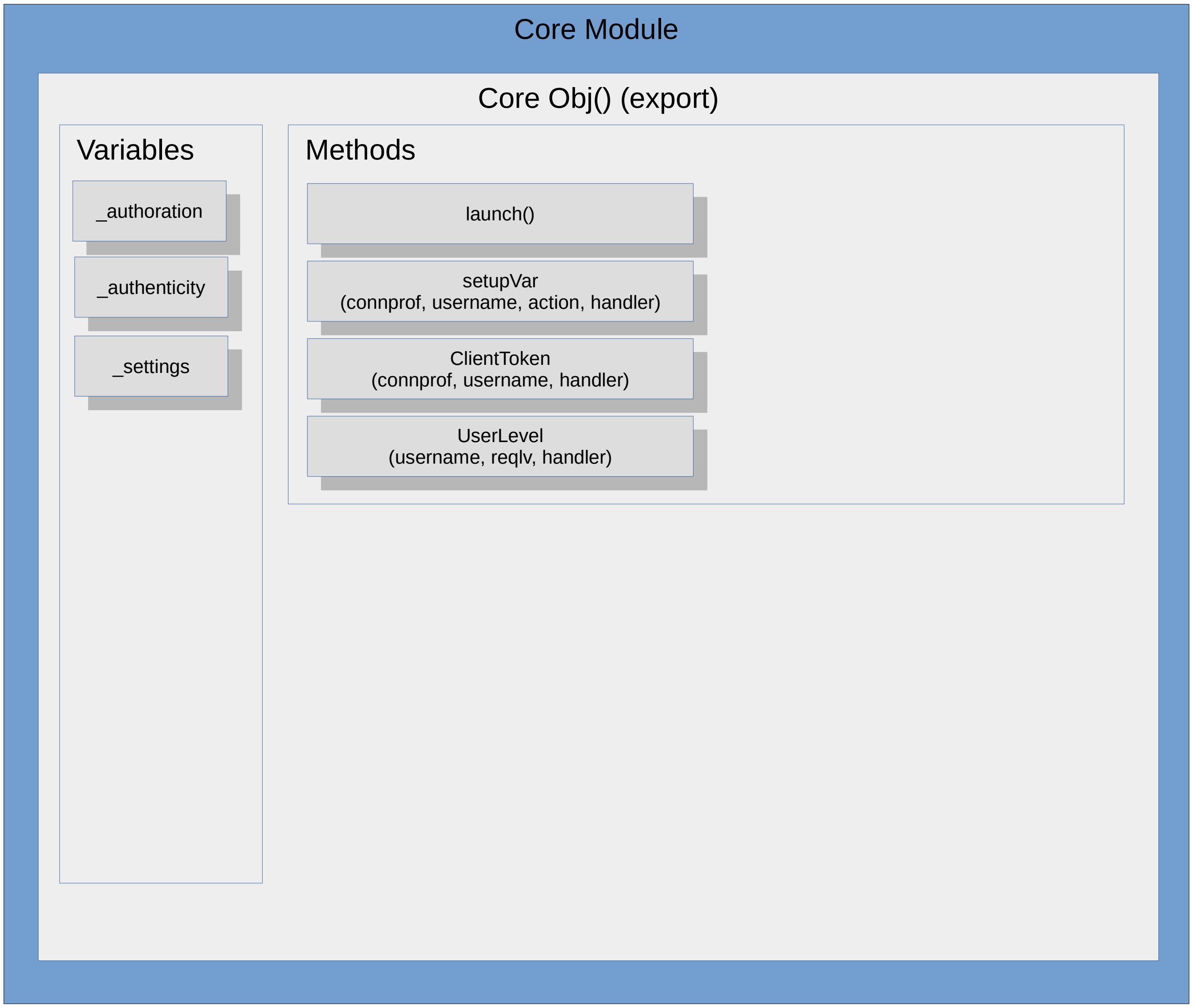
Figure:



Core 1

Objective: provide functions for runtime use, glue

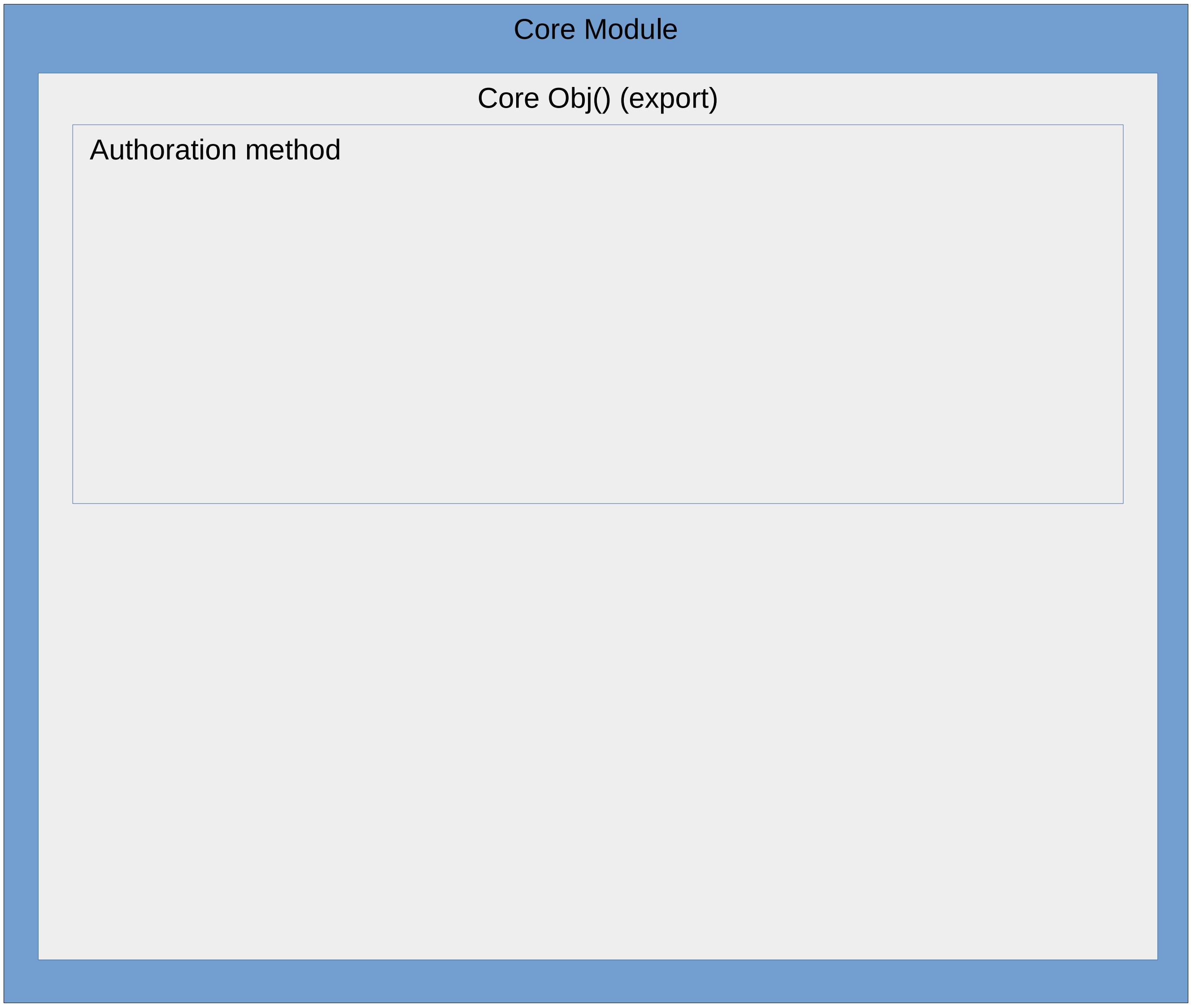
Figure:



Core 2

Objective: provide functions for runtime use, glue

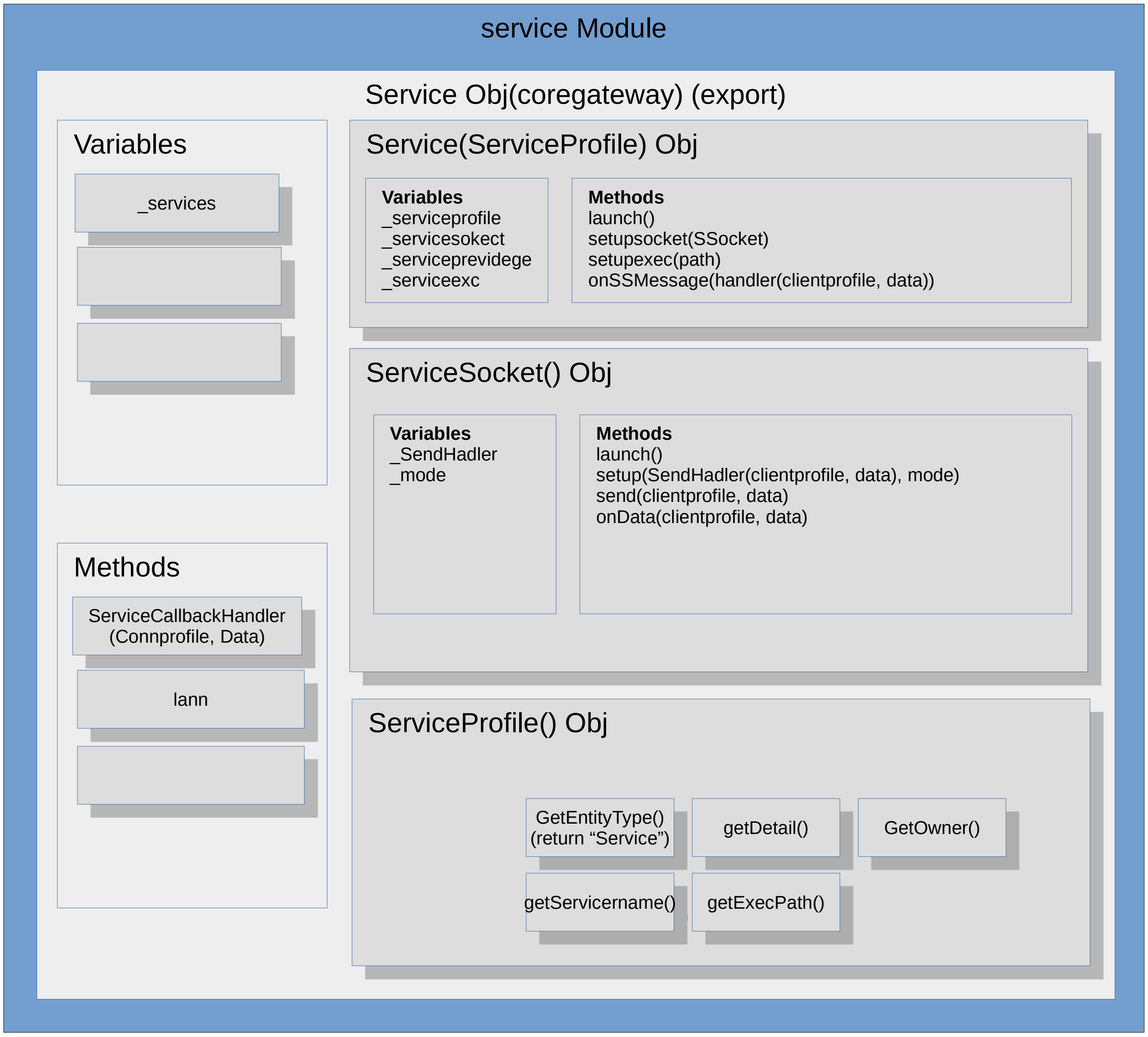
Figure:



Service Module 1

Objective: provide and mange service api, and route the messages on internet

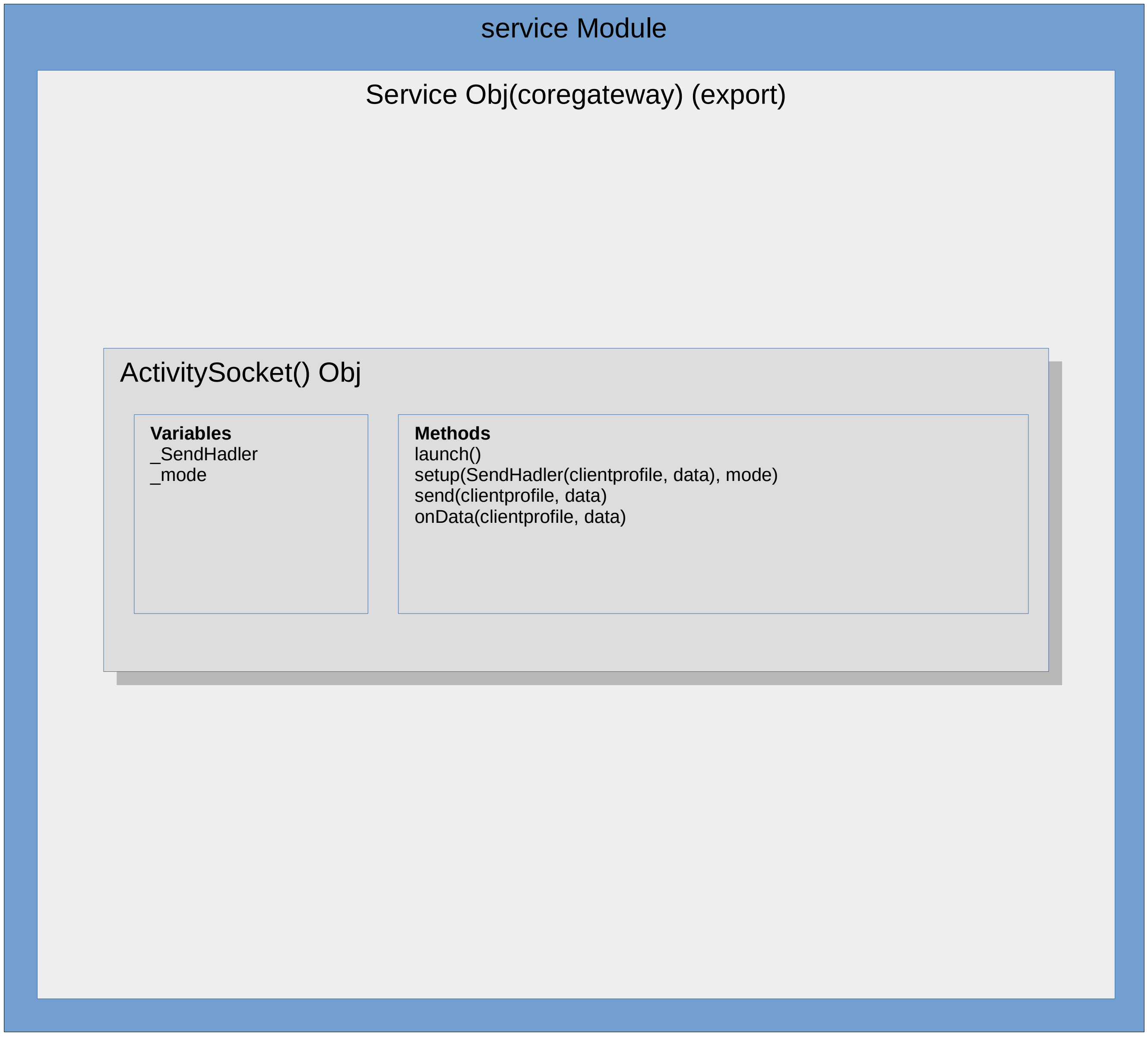
Figure:



Service Module 2

Objective: provide and mange service api, and route the messages on internet

Figure:


service Module
Service Obj(coregateway) (export)
ActivitySocket() Obj
Variables
_SendHadler
_mode
Methods
launch()
setup(SendHadler(clientprofile, data), mode)
send(clientprofile, data)
onData(clientprofile, data)

Clientside module

Service, Servicesocket and API

Explaination of how service work

Once the core of the NSF is started.
The core of NSF will navigate the directories of “services” directory which is under the root of NSF files. And in that directory it will exist a file called “entry.js”.
The figure below can help you understand the concept.

```
------|--(NSd(NOOXY Service deamon))-- …
      |
      |--(services)--|--(services_A)--|--(entry.js)
      |              |                |--(manifest.json)
      |              |
      |              |--(services_B)--|--(entry.js)
      |                               |--(manifest.json)
      |
      |--(service_files)-- …
      |
      |--(launch.js)
```

After the core finish navigating the directories under “services”. It will call the entry.js and call it’s function “start()” and pass API parameter in to start() function.
Below show how the “entry.js” file might be.

In entry.js

```
      function start(api) {
            let ss = api.Service.ServiceSocket
            ss.onMessage = function(ConnProfile, Message) {
                  // do somthing
            }

            ss.sendMessage(ConnProfile, “NSF is cool!”);
            // do something with api
      }

      function end() {

      }

      module.exports = {start: start, end: end}
```

Beware that code in Service is run as a NSFsuperuser,

Service API list

NSF.Service.KillService(Servicename)
NSF.Service.startService(Serivcename)
NSF.Service.getListofService()
NSF.Service.getDetailofService(Servicename)
NSF.Service.disableService(Servicename)
NSF.Service.enableService(Servicename)
NSF.Service.ServiceSocket.onMessage(ClientProfile, message) [Callback]
NSF.Service.ServiceSocket.sendMessage(ClientProfile, message)
NSF.Service.ServiceSocket.onBytes() [not yet]
NSF.Service.ServiceSocket.sendBytes() [not yet]
NSF.Service.ActivitySocket.createSocket(Profile(of an entity), TargetServicename)
NSF.Authoration.Authby.ClientPassword(UserProfile)
NSF.Authoration.Authby.ClientAction
NSF.Authoration.Authby.ClientToken
NSF.Deamon.shutdown
NSF.Deamon.restart
NSF.Deamon.

Preinstalled Service

Preinstalled Service list

Shell Service
Profile Service
Grouping Service