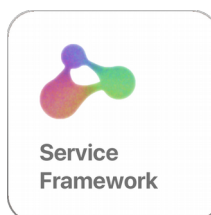


NOOXY Service Framework 1.0

Started by Yves Chen, 10, Mar, 2018



Document Overview

1. Orientation
2. Architecture
3. serverside module
4. clientside module
5. Service, ServiceSocket and ServiceAPI
6. Activities and ActivitySocket(Client socket)
7. NSP(NOOXY Service Protocol)
8. Preinstalled Service
9. Setting file

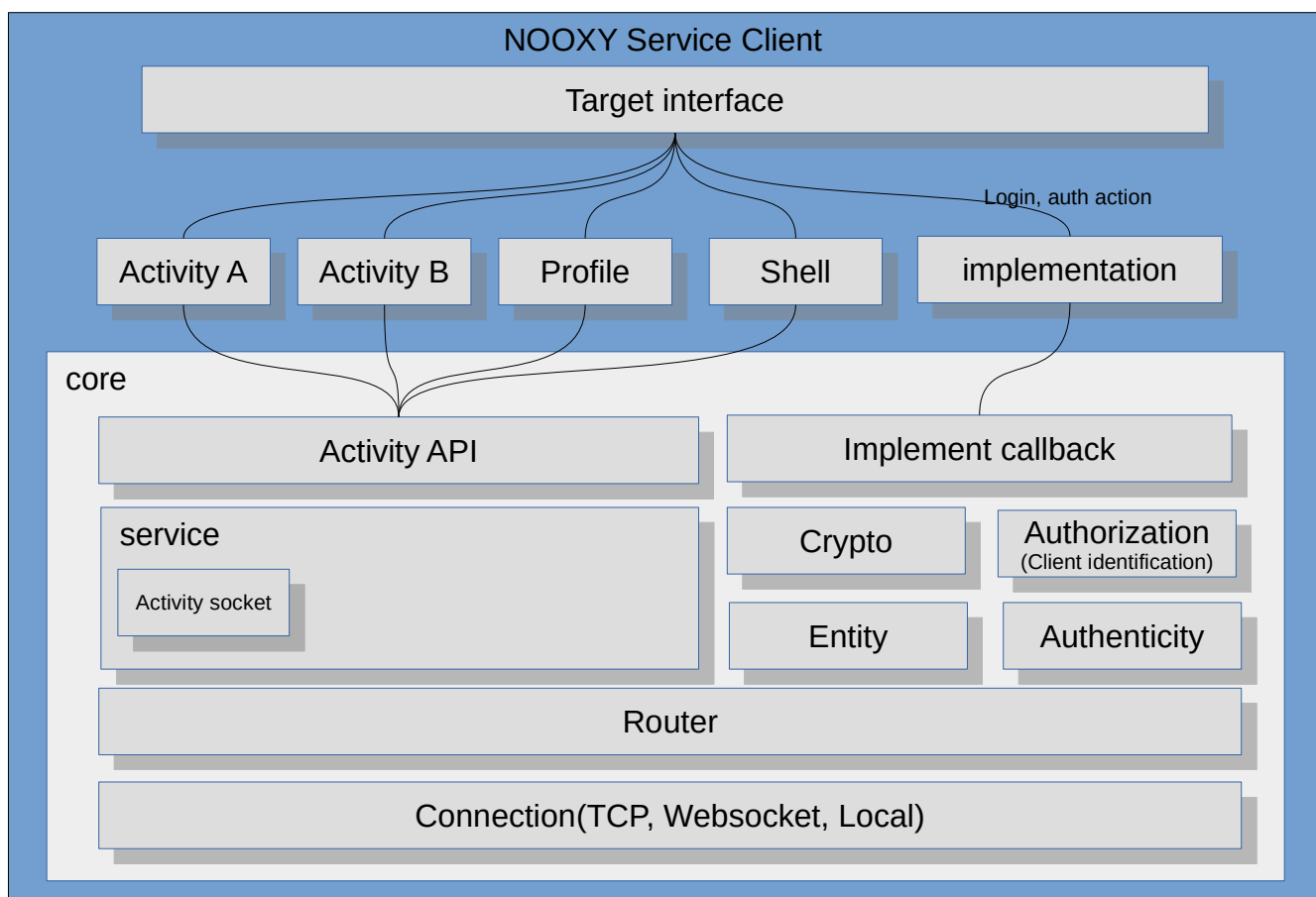
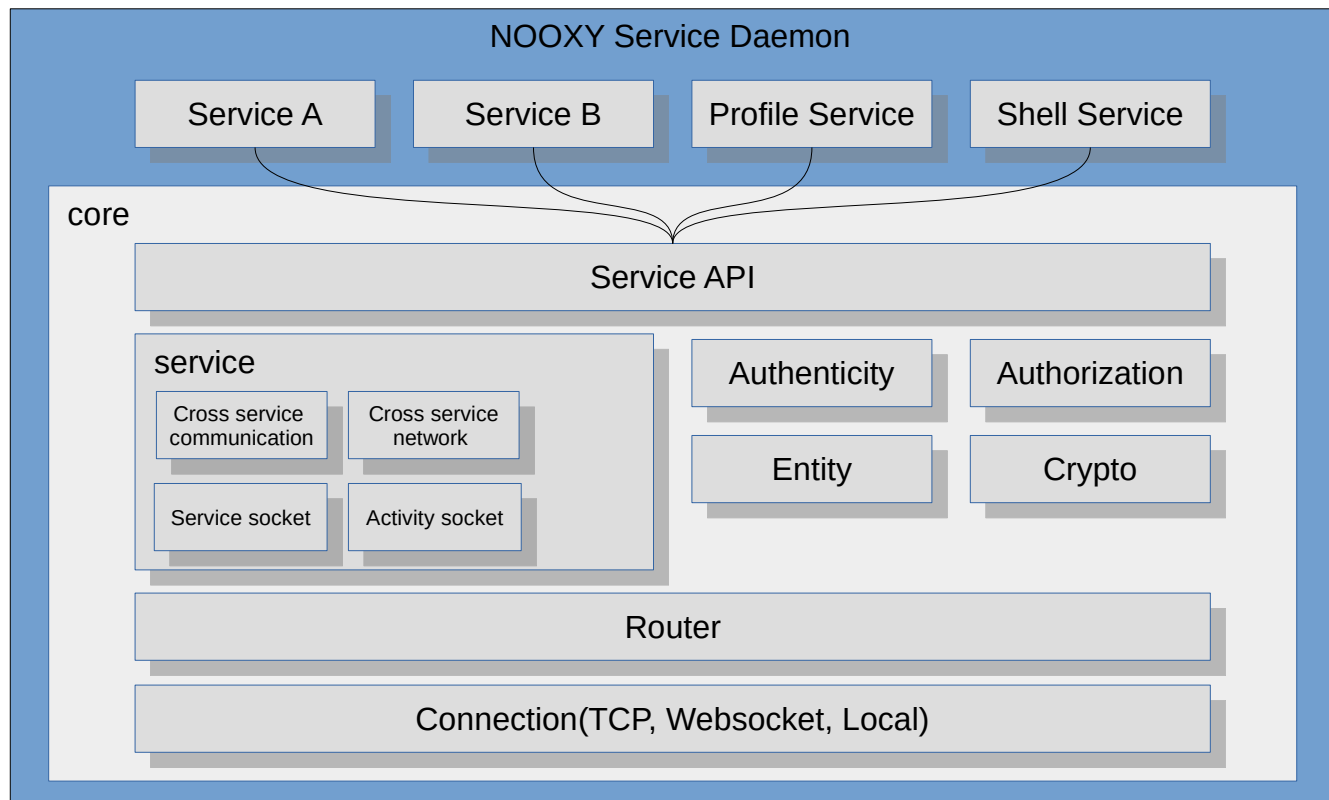
Orientation

NOOXY Service Framework Orientation

1. Entities system(Services, Activities), each entities have it's profile(with contain that showing attached user and user's domain) for deciding should it be trusted.
2. User Orientation, User(in a daemon or a client) can create entities(activities, services). Entities and Users are both owned(registered) by NSd(NOXY Service daemon) of particular domain.
3. Server(we call it "Services") , client(we call it "Activities") structure.
4. Authorization API for Services. Services have responsibilities to protect their contains itself
5. Module idea, "Everything based on service" concept.
6. Lightweight. "Everything based on service" concept.
7. Decentralized. Make it possible to parallelize task in future.
8. NSP(NOXY Service Protocol) is request response style protocol.

Architecture

NOOXY Service Framework Architecture

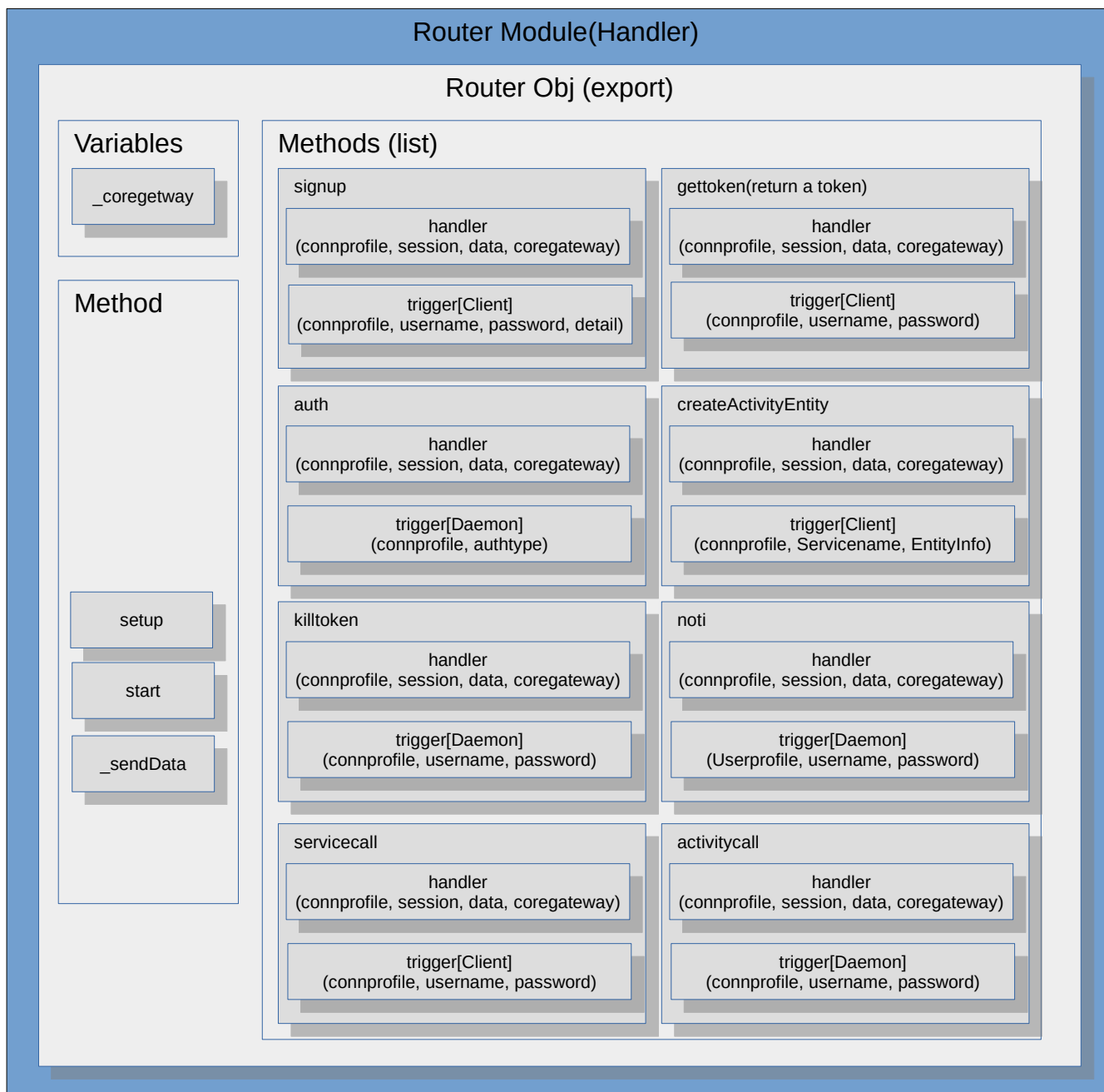


Serverside module

Router Module(Handler)

Objective: A parser and a router. To phrase Json from connection and do local operations. And to switch and trigger between different operations of remotes.

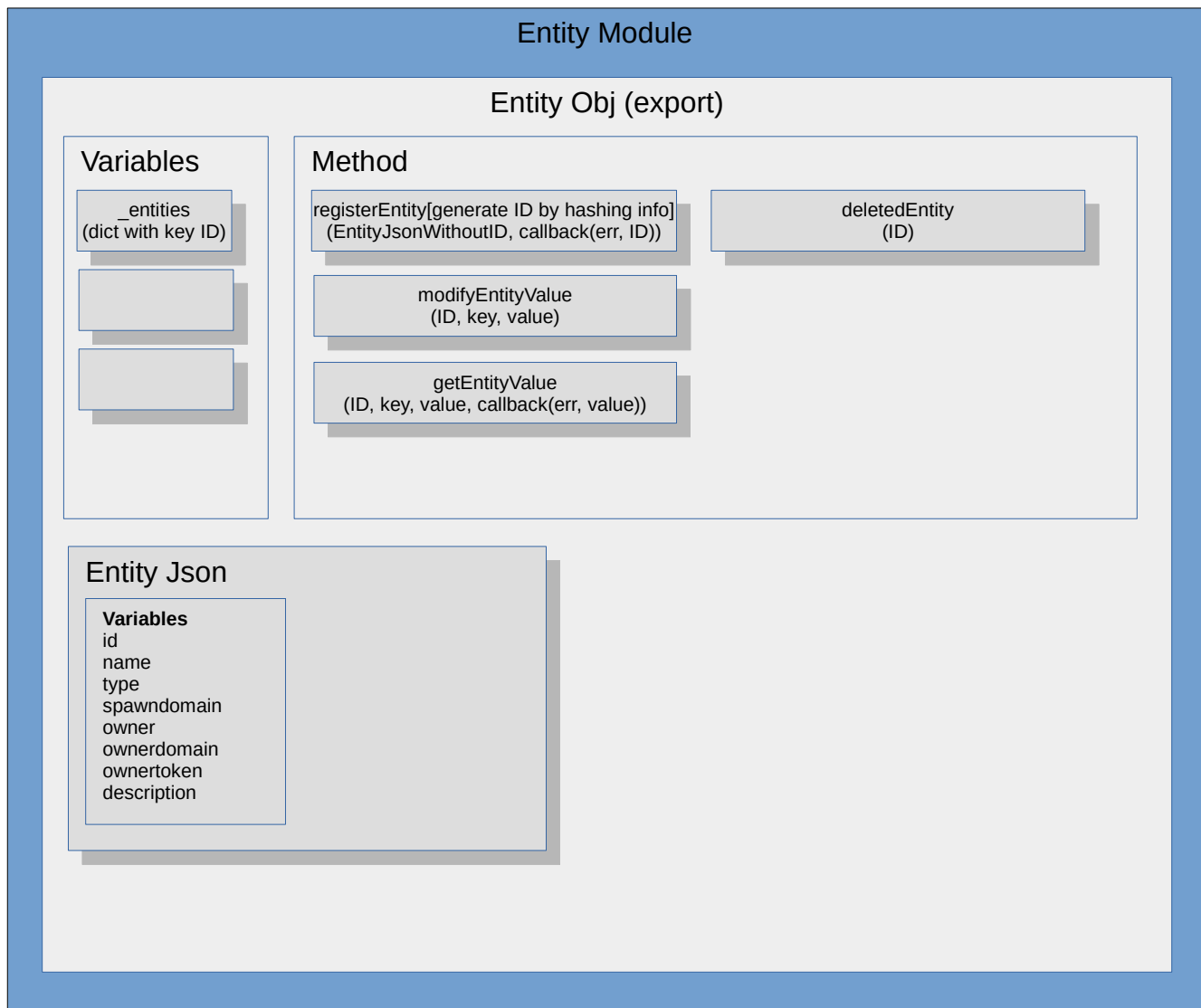
Figure:



Entity Module

Objective: Create identity system for Service , Activity or future stuff. Entities are generated and being realtime. So there is no need for databases.

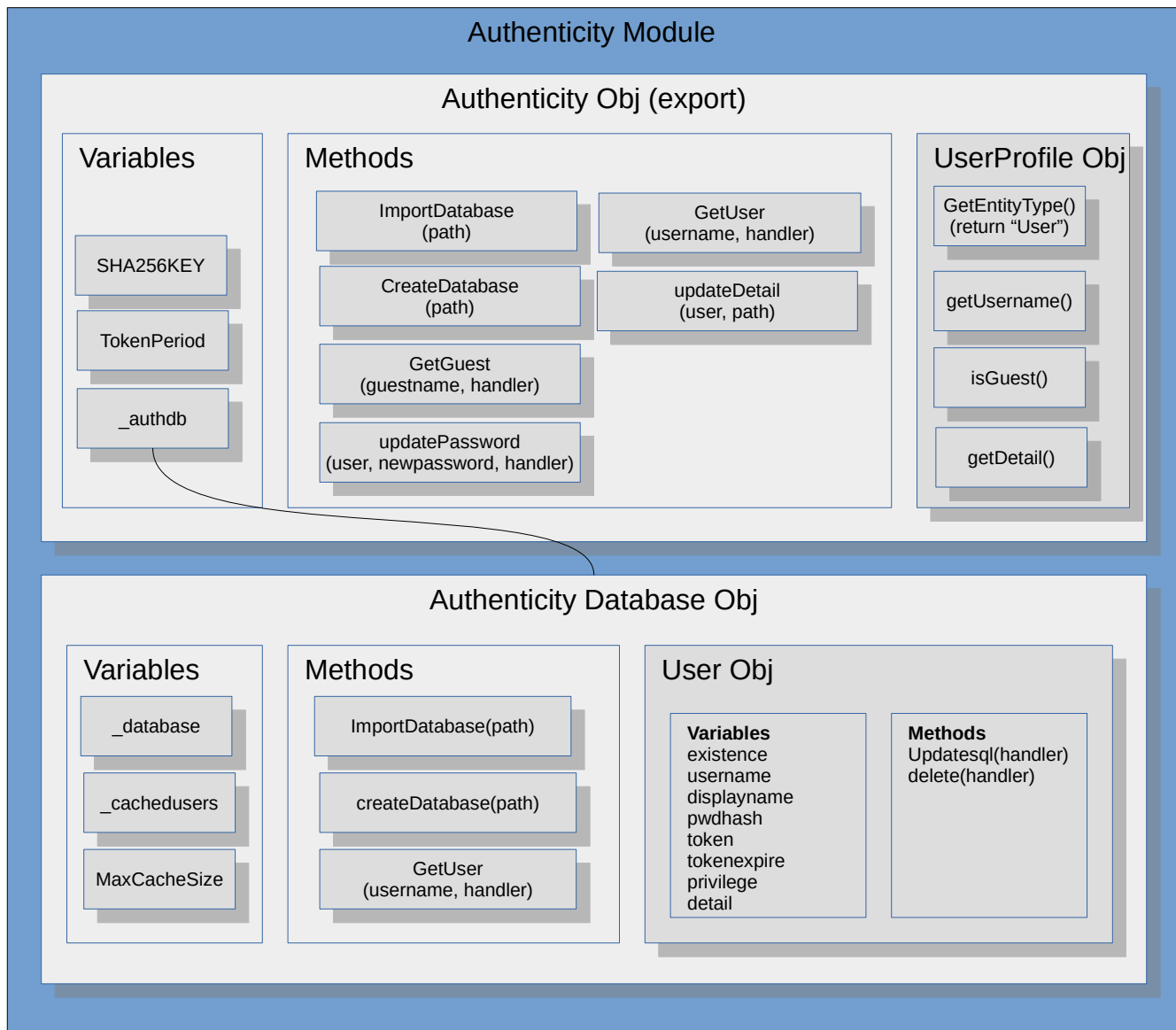
Figure:



Authenticity Module

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

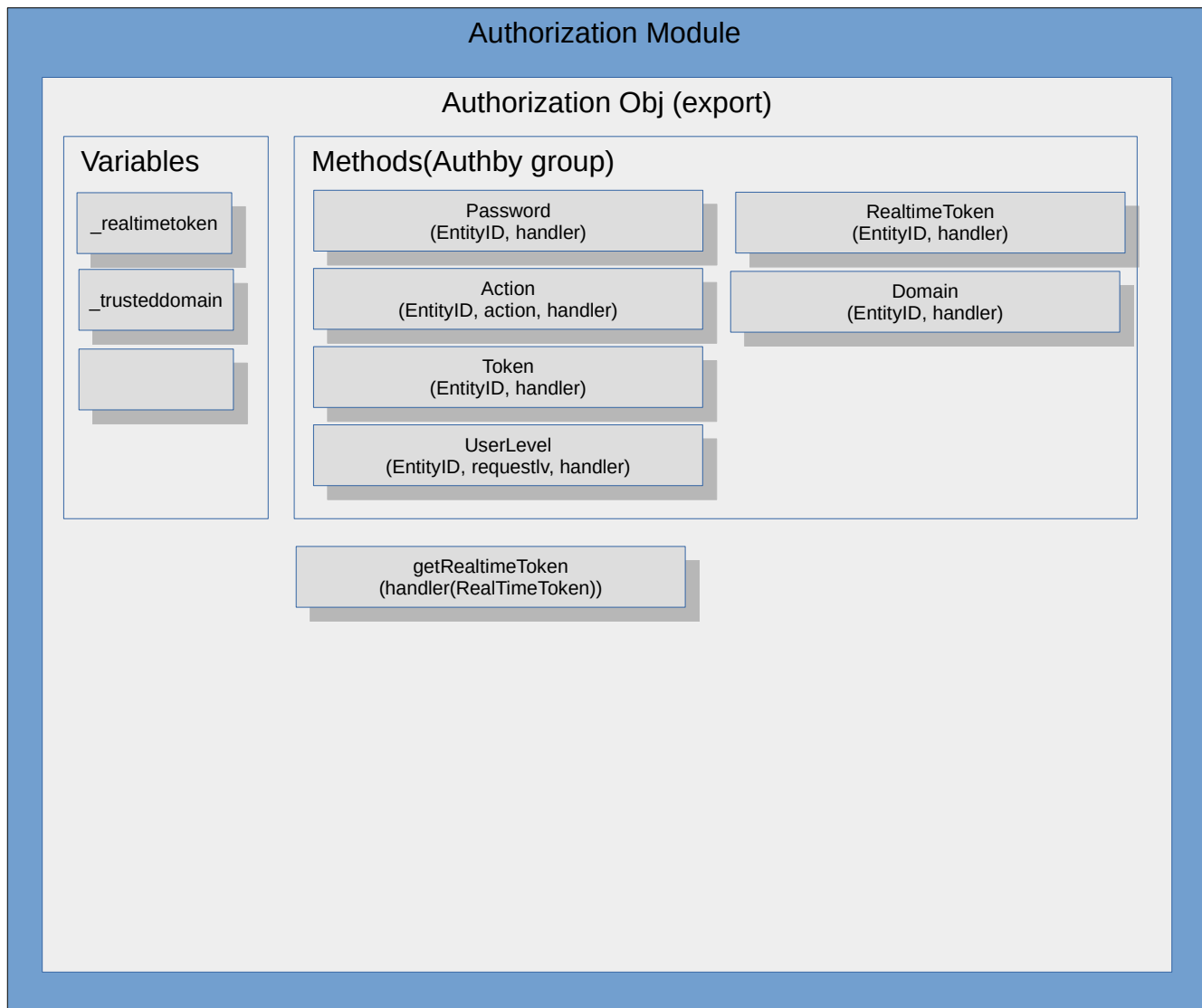
Figure:



Authorization Module

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

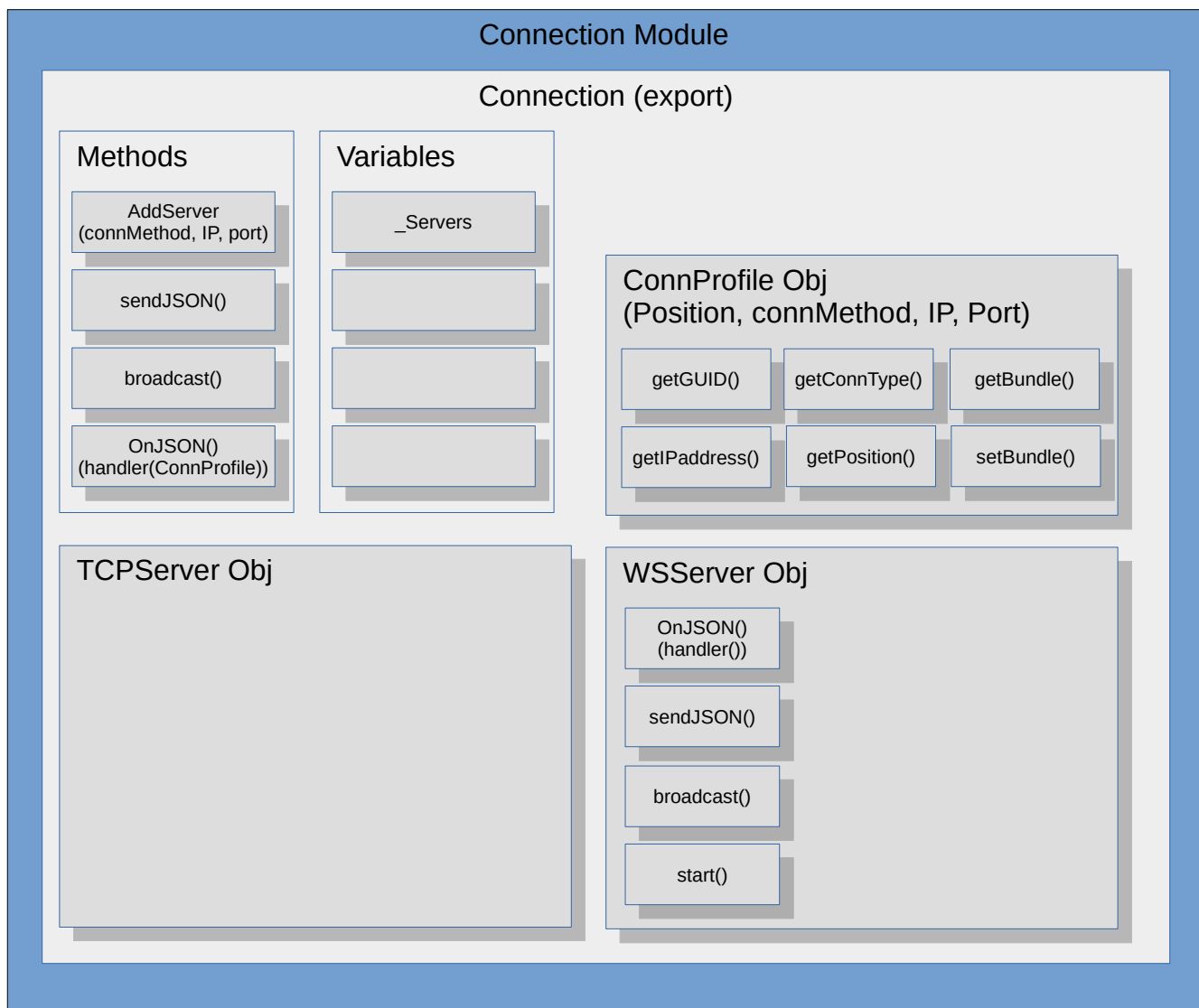
Figure:



Connection Module

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

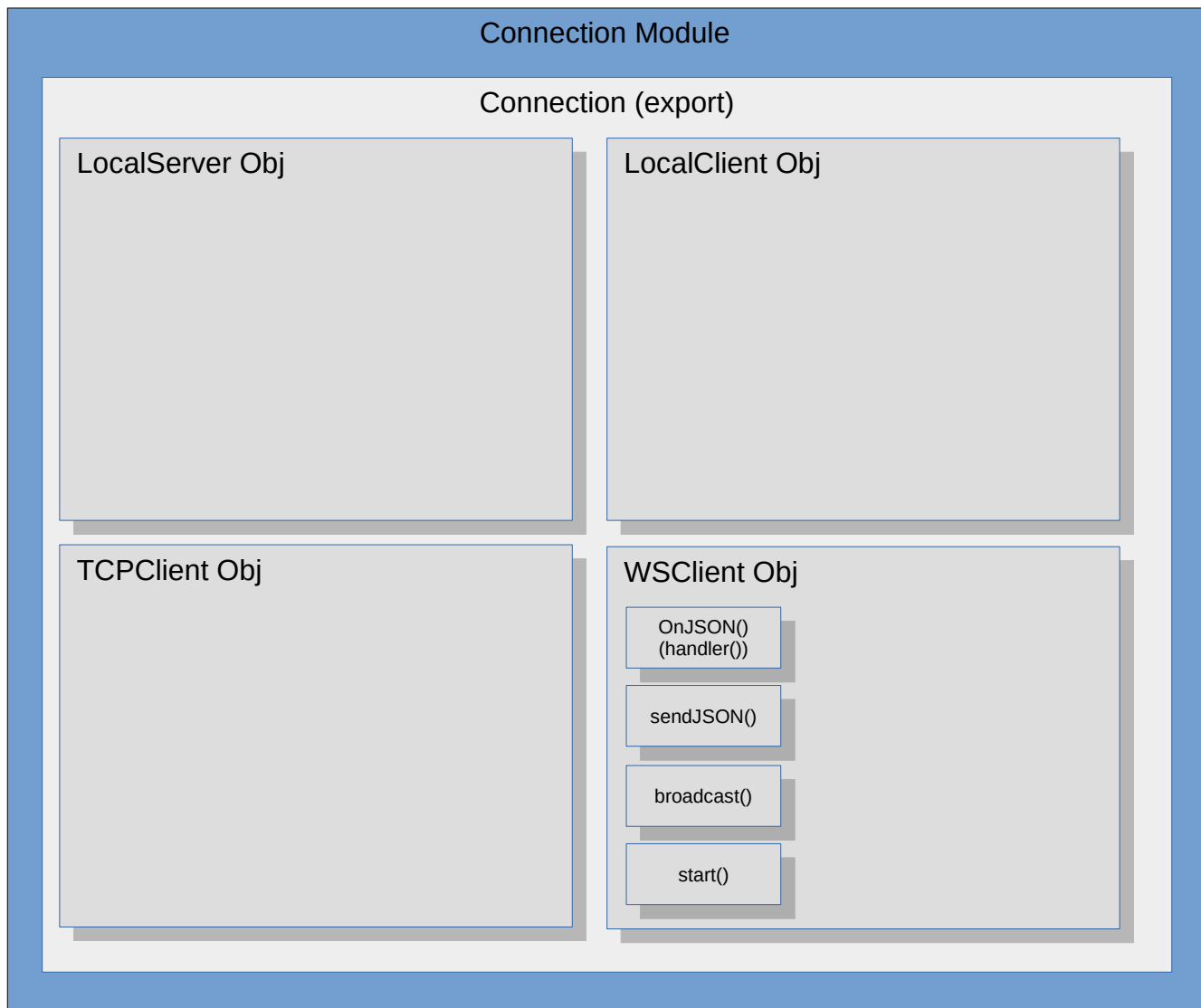
Figure:



Connection Module

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

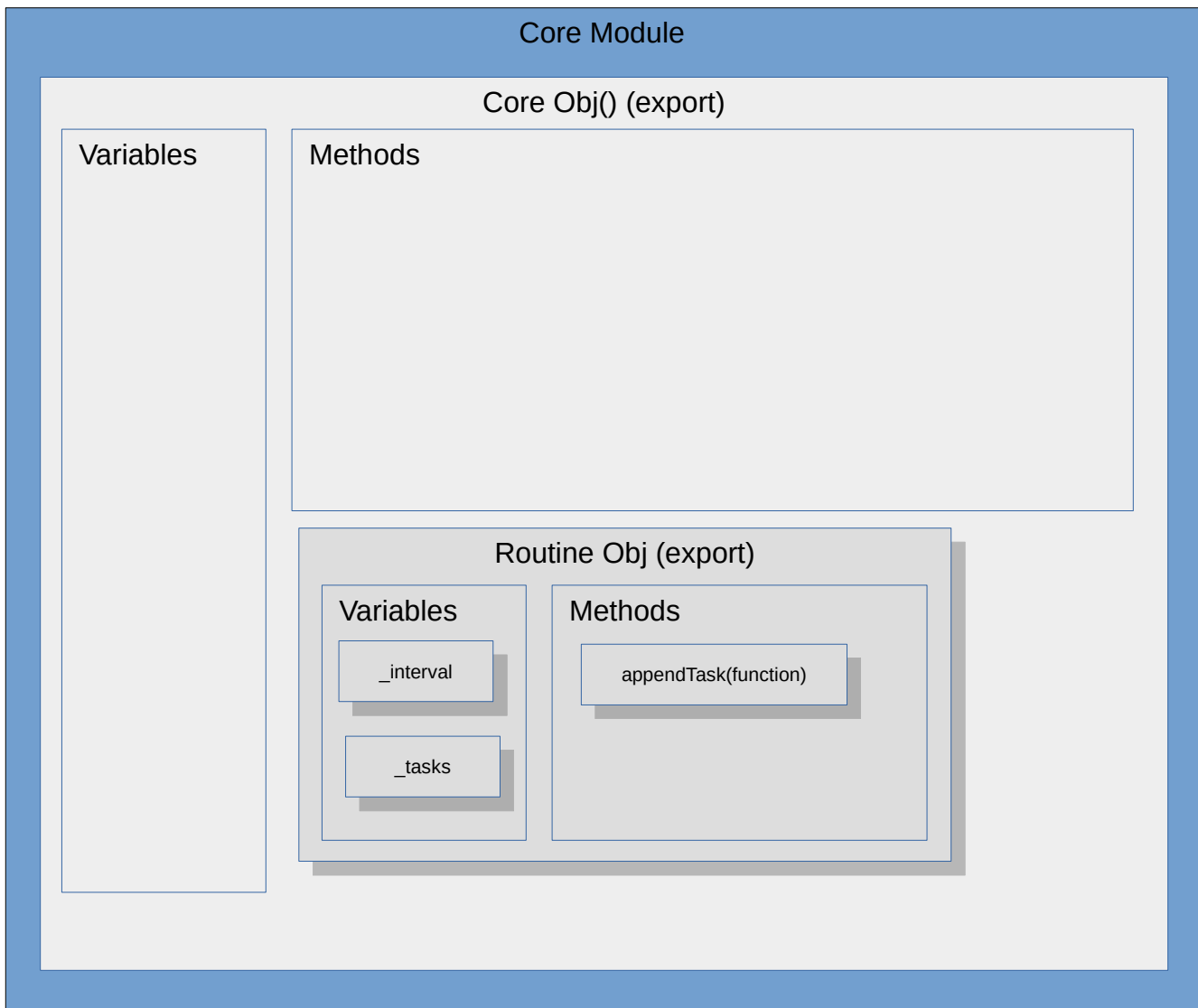
Figure:



Core 1

Objective: provide functions for runtime use, glue

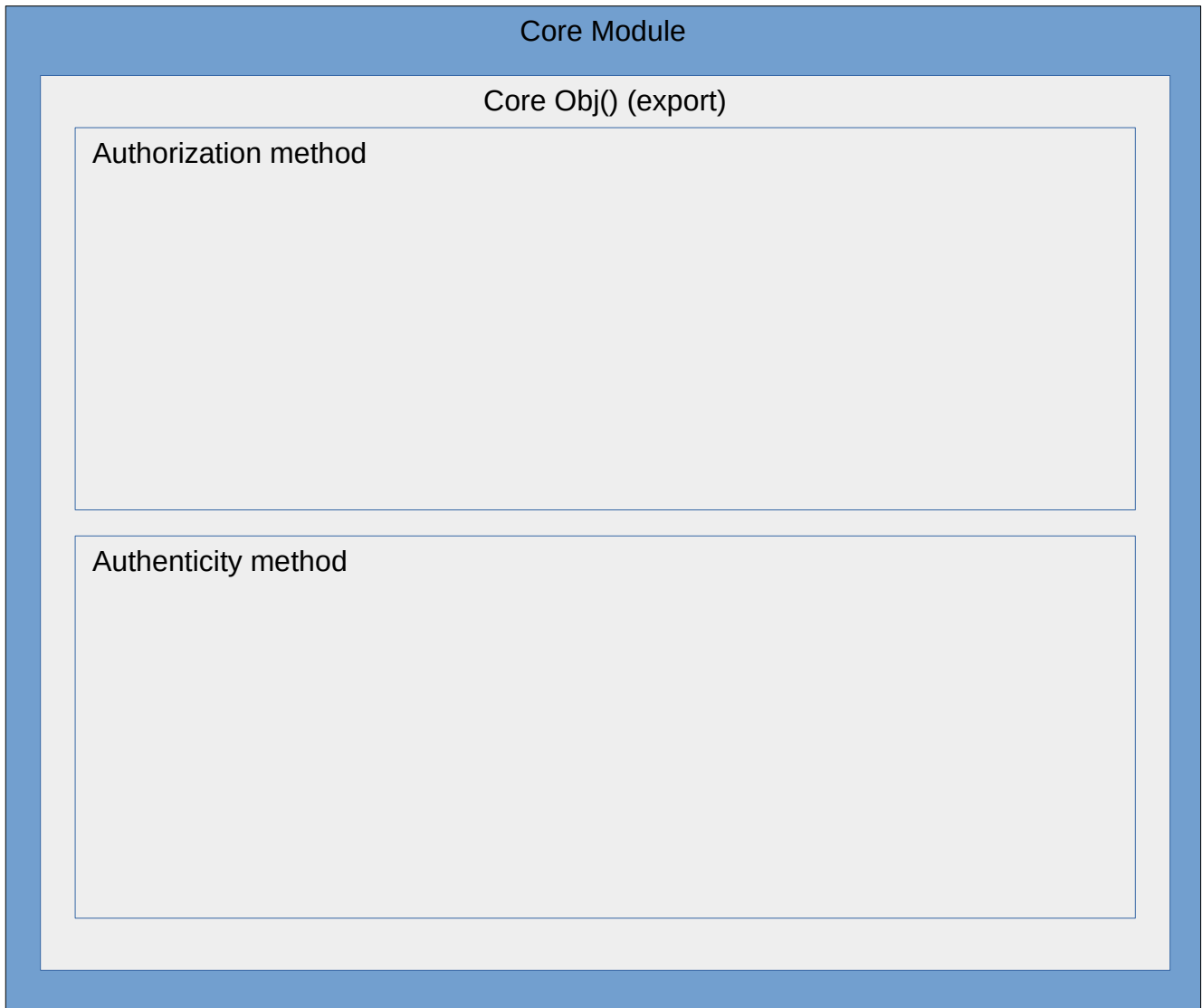
Figure:



Core 2

Objective: provide functions for runtime use, glue

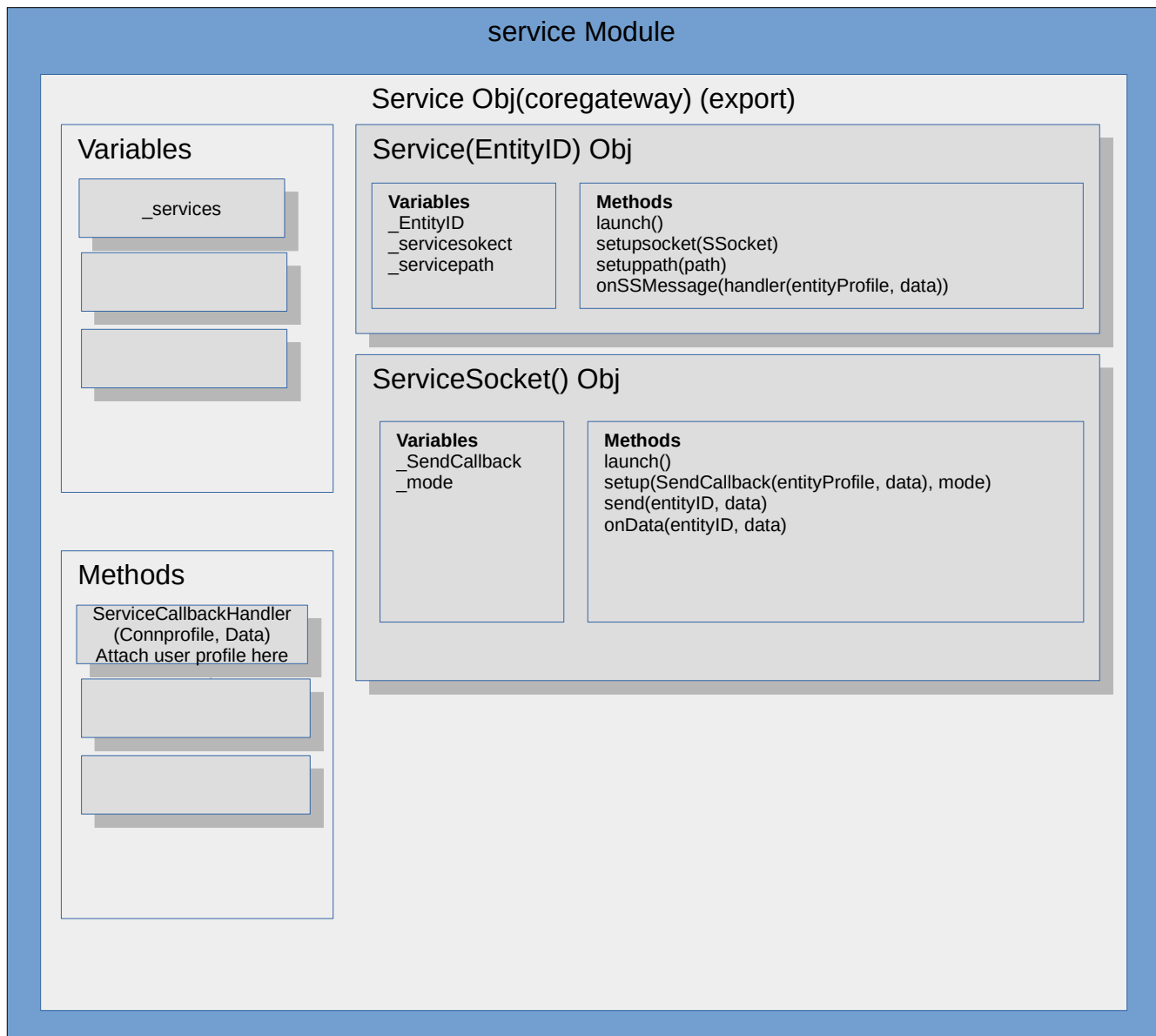
Figure:



Service Module 1

Objective: provide and manage 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:



Clientside module

Service, Servicesocket and API

Explanation 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(N00XY 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(EntityID, Message) {
        // do something
    }

    ss.sendMessage(EntityID, "NSF is cool!");
    // do something with api
}

function end() {

}

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

Beware that code in Service is ran as a superuser,

Service API list

API.Service.killService(Servicename)
API.Service.disableService(Servicename)
API.Service.enableService(Servicename)
API.Service.startService(Servicename)
API.Service.getListofService()
API.Service.getDetailofService(Servicename)
API.Service.ServiceSocket.onMessage(ClientProfile, message) [Callback]
API.Service.ServiceSocket.sendMessage(ClientProfile, message)
API.Service.ServiceSocket.onBytes() [not yet]
API.Service.ServiceSocket.sendBytes() [not yet]
API.Service.ActivitySocket.createSocket(Profile(of an entity), TargetServicename)[return a ActivitySocket]
API.Authorization.Authby.ClientPassword(UserProfile)
API.Authorization.Authby.ClientAction
API.Authorization.Authby.ClientToken
API.Authenticity.killToken(UserProfile)[will logout clients]
API.Daemon.shutdown
API.Daemon.restart
API.Daemon.addRoutine(function)
API.Notification.sentnoti(ClientProfile, message, iconurl)
API.Notification.broadcast(message, iconurl)

NSP(NOOPY Service Protocol)

Basic Concept of NSP

1. NSP is based on text, based on Json data structure.
2. It's communication style is like http. Existing a method, a request and a response.
3. NSP method is designed to be handle by core, not recommend to let service have direct access.
4. Once a NSP package was sent. It contains 3 main parts.
 1. "method" for identify the type of operation
 2. "session" for identify the stage of request or response.
 3. "data" for the actual data that be transferred.
5. There are following standard methods for NSP.
 - 1.
6. In order to focus on data that be transferred We will abridge some terms.
 1. "method" refer to "m"
 2. "session" refer to "s"
 3. "data" refer to "d"
 4. method terms will be explained next page

Methods of NSP

1. Login
2. signup
3. auth
- 4.

JSON Structure of NSP

"GETTOKEN"

username=text, password=text, token=text

Request(client):

```
{
  m: "GT",
  s: "rq",
  d: {u: username, p: password}
}
```

Response(daemon):

```
{
  m: "GT",
  s: "rs",
  d: {t: token}
}
```

"SIGNUP"

username=text, password=text, success=boolean

Request(client):

```
{
  m: "SU",
  s: "rq",
  d: {u: username, p:password}
}
```

Response(daemon):

```
{
  m: "SU",
  s: "rs",
  d: {s: success}
}
```

"Auth"

auth_data_structure=dictionary

Request(daemon):

```
{
  m: "AU",
  s: "rq",
  d: {auth_data_structure}
}
```

Response(client):

```
{
  m: "AU",
  s: "rs",
  d: {auth_data_structure}
}
```

JSON Structure of NSP

"Auth": Verify messages

Messages = text, accept = boolean

```
Request(daemon):
{
  m: "AU",
  s: "rq",
  d: {
    m: "VM",
    d: {m: messages}
  }
}
```

Response(client):

```
{
  m: "AU",
  s: "rs",
  d: {
    m: "VM",
    d: {a: accept}
  }
}
```

"Auth": Verify Password

Username = text, password = text

```
Request(daemon):
{
  m: "AU",
  s: "rq",
  d: {
    m: "PW"
  }
}
```

Response(client):

```
{
  m: "AU",
  s: "rs",
  d: {
    m: "PW",
    d: {u: username, p: password}
  }
}
```

"Auth": Verify Token

Username = text, token = text

```
Request(daemon):
{
  m: "AU",
  s: "rq",
  d: {
    t: "TK"
  }
}
```

Response(client):

```
{
  m: "AU",
  s: "rs",
  d: {
    m: "TK", d: {u: username, p: password}
  }
}
```

JSON Structure of NSP

```
"SERVICECALL"
service_data_structure = dictionary
Request(client):
{
    m: "SC",
    s: "rq",
    d: {service_data_structure}
}
Response(daemon):
{
    m: "SC",
    s: "rs",
    d: {service_data_structure}
}

"SERVICECALL": bind user
Request(client):
{
    m: "SC",
    s: "rq",
    d: {
        m: "BU",
        d: {u: username}
    }
}
Response(daemon):
{
    m: "SC",
    s: "rs",
    d: {}
}

"SERVICECALL": callservicesocket
data = anytype
Request(client):
{
    m: "SC",
    s: "rq",
    d: {
        m: "SS",
        d: data
    }
}
Response(daemon):
{
    m: "SC",
    s: "rs",
    d: {
        m: "SS",
        d: {}
    }
}
```

JSON Structure of NSP

```
"ACTIVITYCALL"
service_data_structure = dictionary
Request(daemon):
{
    m: "AC",
    s: "rq",
    d: {service_data_structure}
}
Response(client):
{
    m: "AC",
    s: "rs",
    d: {service_data_structure}
}
```

```
"ACTIVITYCALL": callactivitysocket
data = anytype
Request(daemon):
{
    m: "AC",
    s: "rq",
    d: {
        m: "AS",
        d: data
    }
}
Response(client):
{
    m: "AC",
    s: "rs",
    d: {
        m: "SS",
        d: {}
    }
}
```

Preinstalled Service

Preinstalled Service list

Shell Service(for superuser remotely manage NSF)

Profile Service (mange user icon, phone, email etc.)

Grouping Service()

Analytic Service(gather User info, recognizing is it IoT or browser etc)