# API Documentation

## safetyagent.agent

### Class

**class safetyagent.agent.MonitoringAgent(Agent)**

"""
A base agent class which is to be served by an AgentProcess
An AgentProcess runs a Pyro multiplexed server and serves one Agent object.
"""

**Parameters**

---

The parameters needed is same as the Agent class in the osbrain agent.py
*class* osbrain.agent.Agent (*name=None, host=None, serializer=None, transport=None)*
https://github.com/opensistemas-hub/osbrain/blob/master/osbrain/agent.py

**Attributes**

---

**_log**: dict  for storing the json log file recieved from Fluentd
**_type**: string  the type of the agent itself. Default is 'monitoring_agent'
**rule_manager**: RuleManager  RuleManager determines the rules agent should follow
**server**: Server  the monitor server for listening the logging message of Fluentd

**Methods**

---

#返回_type，default 時是'monitoring_agent'
**type()**
  Return:
    _type: string
#設定監聽最新 message 用的 ip/port
**monitor(host, port)**
  Parameters:

```
    host: string
    port: string or int


#將資料透過 message_channel 送出，可參考 messagechannel 裡定義
send(message_channel, msg)
  Parameters:
    message_channel: string
    msg: string


#設定 MonitoringAgent 需要用到的 Rule 物件
set_rule(*rules):
  Parameters:
    rules: Rule

#從監聽最新 message 用的 ip/port 接收資料後，若 message_channel 不等於 None，則將
Rule 物件判斷後的 verdict 透過指定的 message_channel 傳輸，否則打印到螢幕
trigger_rule(message_channel = None):
  Parameters:
```

**class ControllingAgent(Agent)**

**Parameters**

```
----------
The parameters needed is same as the Agent class in the osbrain agent.py
class osbrain.agent.Agent (name=None, host=None, serializer=None,
transport=None)
https://github.com/opensistemas-hub/osbrain/blob/master/osbrain/agent.py
```

**Attributes**

```
----------
_type: string #the type of the agent itself. Default is 'monitoring_agent'
controller: Controller #控制監控系統的物件詳細參考 controllerset
```

**Methods**

```
----------
#返回_type，default 時是'monitoring_agent'
type()
```

```
  return
    _type: string


#設定 ControllingAgent 使用到的 Controller 物件，用來對監控系統控制
set_controller(controller)
  parameters
    controller: Controller


update_controller(msg)
  parameters
    msg: string 從 MonitoringAgent 物件些收的 message


control()
```

## safetyagent.rule

### Class

**class safetyagent.rule.RuleSelector(*environs)**

```
select the corresponding environment which is able to be updated by the
message
Parameters
----------
environs


Attributes
----------
latest_msg: dict  #最新的 message，從 RuleManager 物件獲得
environ_all: list  #所有 Parameter 中的 Enviroment 物件
selected_environ_list: list #log mark 與最新 message 相同的 Enviroment 物件

Methods
----------
#更新獲得 log mark 與最新 message 相同的 Enviroment 物件
#update the selected environment(not trigger the environment!!) list and
return the selected environment
update(self, msg)
  parameter:
    msg: dict                    #最新的 message，從 RuleManager 物件獲得
```

```
  return:
  selected_environ_list: list  #log mark 與最新 message 相同的 Enviroment 物件
```

**class safetyagent.rule.Environment(log_mark)**
**Parameters**
log_mark

**Attributes**
```
----------
log_mark: dict          #log mark 詳細參考 ruleset
tags: dict              #log_mark['tags'] 詳細參考 ruleset
fields: list            #log_mark['fields']詳細參考 ruleset
environ_fields: dict    #log_mark['fields'] = ['A','B']則其 environ_fields 為
{'A':1,'B':                          2}具有變數值的 dictionay
rules_lists: Rule       #所有註冊被 Environment 物件的 Rule 物件
```

**Methods**
```
----------

#check whether the enviroment can be updated by the message
```
**check(msg)**
```
  parameter:
    msg: dict 最新的 message，從 RuleManager 物件獲得
  return:
    Boolean
#更新 environ_fields 的值
```
**update(msg)**
#添加 Rule 物件，當有最新 message 時，觸發所有加入的 Rule 物件
**add_rule(self, *rules)**
#觸發所有加入的 Rule 物件
**trigger()**

**class safetyagent.rule.Rule(*log_marks)**
**Parameters**
```
----------
log_marks: dict # the tags of the log data needs to be used in the rule
```

**Attribute**

---

**flag:** Boolean # 用來判斷 Rule 物件是否被 Environment 物件更新及返回 verdict_list 中的 verdict 給 RuleManager 類。當更新時為 True，返回時設 False

**log_marks_list**: list #list to store log marks

**condiction_action_lists:** list #存放要被觸發的 function

**environ_fields:** list #存放從 Environment 物件接收的 environ_fields，是為 dictionary

**fields: list** #log mark 裡 field 項的值

**number: int** # 一個 Rule 物件所需使用到的 log mark 個數

**verdict_list: list** #condiction_action_lists 中被觸發的 function 返回值

**Methods**

---

```
# get the log_marks of the rule in the list
get_log_marks()
  return
    log_marks_list


# 接收來自 Environment 物件的 environ_fiels
trigger(environ_fields, fields)
  parameters
    fields: list #log_mark 中的 fields 項 ex:["A","B"]
    environ_fields: dict #由 Environment 物件接收到具有值的 dictionary
ex:{"A":1,"B":2}


#返回 trigger 後產生的判斷值，是為存放 string 的 list
get_verdict()
  return
    verdict_list: list
# 添加需要被觸發的 function
add_condition_action(*cas)
  parameters
    cas:function #其返回值必須是 None 或是 string
```

**class safetyagent.rule.RuleManager( *rules, env_cls = Environment, rule_selector_cls = RuleSelector)**

'''

提供 MonitoringAgent 物件使用的接口。
具有當新的 message 接收後會觸發 Environment 物件方法，以及處理當 Rule 物件產生的 verdict 方法
'''
**Parameters**
---
rules: Rule #任意數量的 Rule 物件
env_cls: Environment class
rule_selector_cls: RuleSelector class

**Attributes**
---

rule_selector_cls: RuleSelector class
env_cls: Environment class
environ_list: list  #存放所有與 Environment 物件的
log_marks_list: list  #存放 Rule 物件中的所有 log_mark 值
rules_list: list  #存放從 parameter 中獲得的任意數量 Rule 物件
selector: RuleSelector #塞選 Rule 用的物件

**Methods**
---
# 接收新的 message 後更新存放在 environ_list 中的 Enviroment 物件
**trigger(msg)**
  parameters
    msg: dict #新進的 message

# 處理來自 Rule 物件中返回的 verdict
**handle_verdict(handler, message_channel = None)**
  parameters
    handler: function  #callback function，當有新的 verdict 產生後觸發
    message_channel: string #需為 safetyagent.messagechannel 中的 channel_local

## safetyagent.messagechannel

### function

**safetyagent.messagechannel.set(monitoring_agent, controlling_agent, remote_agent, channel_local, channel_remote, handler)**

```
# set the Push-Pull communication pattern for the controlling agent and the
monitoring agent
# 使用 ZeroMQ 的 Push-Pull 通訊模式綁定 monioring agent 與 controlling agent 連接，
controlling agent 為 Pull 端，monitoring agent 為 Push 端
Parameters
----------
    monitoring_agent: MonitoringAgent  #本地端的 monitoring agent
    controlling_agent: ControllingAgent #本地端的 controlling agent
    remote_agent: MonitoringAgent #遠端的 monitoring agent
    channel_local: string  #遠端 agent 與本地端 controlling agent 通訊頻道名稱
    channel_remote: string  #本地端 agent 與本地端 controlling agent 通訊頻道名稱
    handler: function  #callback function 決定當 controlling agent 接收資料後的
function
```

## safetyagent.jsonsocketserver

### Class

**class safetyagent.jsonsocketserver.Server(host, port)**

```
A socket server to receive the json type message

Parameters
----------
host : str   #Host address
port : str or int   #Port

Attributes
----------
socket : object     #the return object from the socket.socket()


Methods
----------
# 等待客戶端連線，若已先有客戶連線則中斷原有客戶連線，並重新連線
accept()


# 將 dict 的資料轉成 json tring byte 並送出其長度及資料
```

```
send(data)
    parameter:
      data: dict
```

#先接收 string 的長度後創建相應的記憶體空間，每次接收 1 個 string byte 資料並存放在記憶體空間直到資料傳輸完成

```
recv()
    return
      deserialized: dict
```

#關閉 socket 連線

```
close()
```

### Exception

# client disconncetion error occurs

**exception safetyagent.jsonsocketserver.ClientDisconnectionError(Exception)**

## safetyagent.controllerset

存放 controller 物件的檔案。controller 物件為所有符合具有 display(), update(msg), run()接口的物件。如下所示。

```
class VirtualController:
    def __init__(self):
        self.value = ""
    def display(self):
        print(self.value)
    def update(self, msg):
        self.value = msg
    def run(self):
        while True:
            print('the signal received',self.value)
```

## safetyagent.ruleset

log_mark 是為具有{”tags”:{“host”:  “s”, “type”: “s” , “device”: “s”}, “field” : [“s”,”s”,”s”]}pattern 的字典(dictionary)，其中”s”是可以替換的 string

```
log_mark =  {
        "tags":{"host": "alphabot","type": "distance",
"device":"utrasonic_sensor"},
        "fields": ["front"]
        }
```