Общество с ограниченной ответственностью

"ИНСАЙТ-менеджмент"

УТВЕРЖДАЮ:

Генеральный директор

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ И.К. Грибанова

«\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_ 2018 г.

О Т Ч Е Т

О НАУЧНО-ИССЛЕДОВАТЕЛЬСКОЙ

И ОПЫТНО-КОНСТРУКТОРСКОЙ РАБОТЕ (НИОКР)

Проектирование и разработка модуля виртуального контроллера.

Проектирование модуля управления задачами сбора данных по событиям

(промежуточный)

Этап № 3

|  |  |  |
| --- | --- | --- |
| Ген. директор  ООО «ИНСАЙТ-Электроникс» |  | С.В. Грибанов |
| Руководитель НИОКР, к.т.н. |  | А.В. Кычкин |
| Исполнитель |  | Р.Р. Гайнанов |

Пермь 2018СПИСОК ИСПОЛНИТЕЛЕЙ

Руководитель работы,

к.т.н. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А.В. Кычкин

подпись, дата

Исполнители темы:

Гайнанов Р.Р.

СОДЕРЖАНИЕ

[Введение 4](#_Toc512958400)

[1. Проектирование и разработка модуля виртуального контроллера 5](#_Toc512958401)

[1.1. Описание схемы модуля виртуального контроллера 5](#_Toc512958402)

[1.2. Описание алгоритма работы с виртуальным устройством 7](#_Toc512958403)

[1.3. Описание перечня функций и команд контроллеру 8](#_Toc512958404)

[1.4. Разработка модуля виртуального контроллера 9](#_Toc512958405)

[1.5. Описание запуска модуля и команд для тестирования 11](#_Toc512958406)

[2. Проектирование модуля управления задачами сбора данных по событиям 13](#_Toc512958407)

[Заключение 15](#_Toc512958408)

[Список литературы 16](#_Toc512958409)

[Приложение А Реализация модуля виртуального контроллера 17](#_Toc512958410)

# Введение

Любая программно-аппаратная система должна качественно выполнять требуемые функции. Для этого необходимо полноценное и всестороннее тестирование. Зачастую для сокращения сроков тестирования и увеличения скорости работы внутри приложения применятся различные виртуальные модули и компоненты. Они способны имитировать пользователя или системы с которой производится соединение. В рамках предметной области IoT будет эффективно протестировать подсистему обмена данными подключаемых устройств и системы. Модуль виртуального контроллера позволит провести предварительное тестирование без использования физического контроллера.

# Проектирование и разработка модуля виртуального контроллера

## Описание схемы модуля виртуального контроллера

Модуль виртуального контроллера состоит из следующих блоков и с разрабатываемой системой взаимодействует через виртуальную локальную сеть (см. рис. 1).



Рисунок 1 – Схема работы виртуального контроллера

Контроллер использует многопоточную подсистему для отправки и приема информационных сообщений в разных потоках без существенного влияния подпроцессов друг на друга. Внутри потоков создаются виртуальные устройства, каждое из которых может генерировать данных в соответствии с принятыми конфигурациями от системы. Система формирует сообщение следующей структуры (см. рис. 2). Данное сообщение позволяет создать и запустить одно виртуальное устройство в рамках программы виртуального контроллера. Каждое устройство может содержать в себе несколько рядов (переменных) генерации данных. Для формирования сообщений от устройств используется планировщик событий, который настраивается на формирование пакета с данными через заданные промежутки времени. Он вызывает функцию по генерации значения переменной по заданной формуле, которая в результате вместе с другой дополнительной информацией и данными для авторизации отправляется системе EMS через виртуальную локальную сеть.

|  |
| --- |
| {  "guid": "aaaaaaaaa-a508-46cc-a428-1787595d63e4",  "pasword": "12345678",  "timeout": 3,  "variables": [{  "name": "temperature",  "type": 1,  "data\_type": 1,  "func": 1  },{  "name": "humidity",  "type": 1,  "data\_type": 1,  "func": 1  },{  ...  }  ]  } |

Рисунок 2 – Структура данных для создания виртуального устройства

Структура сообщения для настройки устройств содержит минимально следующий набор данных:

* guid – идентификатор устройства, который будет использован при подключении (поле «client\_id») и аутентификации в системе в качестве поля «username»;
* pasword – пароль, используемый виртуальным устройством для аутентификации;
* timeout – временной интервал генерации значений и отправки сообщений с ними;
* variables – список виртуальных переменных (каналов источников данных), каждое из которых имеет следующие поля для настройки:
  + name – название переменной;
  + type – тип переменной, может быть трех значений:
    - 0x01 – IN – переменная только для чтения, является источником данных;
    - 0x02 – OUT – переменная только для записи, не генерирует данные, но может быть изменена из вне;
    - 0x03 – INOUT – переменная для чтения и записи, может изменять свое состояние изнутри и из вне системы;
  + data\_type – тип значений переменной, используется системой EMS для проверки правильности принимаемого значения, может иметь значения:
    - 0x01 – NUMBER – числовой тип данных;
    - 0x02 – STRING – строковый тип данных;
    - 0x03 – BOOL – булевый тип данных;
  + func – номер функции, используемый для генерации значений;

## Описание алгоритма работы с виртуальным устройством

Поясним разработанный алгоритм работы с виртуальным устройством на примере диаграммы процесса взаимодействия сервера с модулем (см. рис. 3).



Рисунок 3 – Процесс взаимодействия виртуального устройства и сервера EMS

После подключения модуля к серверу по защищенному протоколу MQTT, используя данные специального пользователя для создания виртуальных устройств – учетная запись «master» с полными правами, модуль «подписывается» (отправляет сообщение) SUBSCRIBE с содержимым интересующих тем, в данном случае это темы «master/new\_device», «master/stop\_device», «master/show\_devices». Таким образом, сервер осуществляя рассылку (путем сообщений PUBLISH через MQTT-брокера) может взаимодействовать с модулем виртуального контроллера. Например, отправив сообщение по каналу «master/new\_device» с описанной выше структурой, котроллер создаст новое устройство, включит генерацию данных для переменных данного устройства и устройство начнет передачу сообщений через заданные промежутки времени. Эта передача будет происходить до тех пор, пока контроллером на будет получено сообщение от сервера по каналу «master/stop\_device» с заданным GUID устройства.

## Описание перечня функций и команд контроллеру

Используя протокол MQTT можно осуществлять обмен информационными сообщениями с виртуальным контроллером. Но для этого необходимо знать перечень реализованных функций в контроллере. За счет иерархии сообщений, представленных в таблице 1, сервер сможет получать данные сразу со всех устройств, используя для этого специальные подстановочные знаки (wildcard): одноуровневые и многоуровневые. Для использования одноуровневых знаков применяется символ «**+**», а для многоуровневых – символ «**#**». А используя списки доступа (ACL) устройства будут иметь доступ только к тем каналам передачи в которых используется их guid.

Таблица 1. Иерархия тем и типов сообщений у устройства

|  |  |  |
| --- | --- | --- |
| Название темы | Тип сообщения | Пояснение |
| /attributes/# | Publish, Subscribe | Отправить/получить информацию и настройки всех устройств |
| /attributes/<guid>/name | Publish, Subscribe | Отправить/получить имя устройства c заданным <guid> |
| /attributes/<guid>/server | Publish, Subscribe | Отправить/получить имя сервера устройства c заданным <guid> |
| /attributes/<guid>/variables/# | Publish, Subscribe | Отправить/получить параметры входов/выходов устройства c заданным <guid> |
| /variables/# | Publish, Subscribe | Отправить/получить значения всех переменных всех устройств |
| /variables/<guid>/# | Publish, Subscribe | Отправить/получить значения всех переменных устройства c заданным <guid> |
| /variables/<guid>/var1 | Publish, Subscribe | Отправить/получить значения переменной с именем var1 устройства c заданным <guid> |
| /variables/<guid>/var2 | Publish, Subscribe | Отправить/получить значения переменной с именем var1 устройства c заданным <guid> |
| /variables/<guid>/var3 | Publish, Subscribe | Отправить/получить значения переменной с именем var1 устройства c заданным <guid> |

## Разработка модуля виртуального контроллера

Описанный ранее алгоритм был реализован на языке программирования Python с использованием open-source библиотеки paho.mqtt [1] для взаимодействия с MQTT сервером.

Для создания команд от сервера используется следующий класс (см. рис. 4). Он использует возможности класса «Thread» для обеспечения многопоточности приема и передачи сообщений.

|  |
| --- |
| **class** **ClientThread**(Thread):  **def** \_\_init\_\_(self, name):  """Инициализация потока"""  Thread**.**\_\_init\_\_(self)  self**.**name **=** name  self**.**client **=** Client()  self**.**\_stop\_event **=** threading**.**Event()    **def** **stop**(self):  self**.**\_stop\_event**.**set()    **def** **stopped**(self):  **return** self**.**\_stop\_event**.**is\_set()    **def** **run**(self):  """Запуск потока"""  **try**:  self**.**client**.**subscribe(default\_topics)  **except** **KeyboardInterrupt**:  **print**("exiting") |

Рисунок 4 – Класс модуля виртуального контроллера, используемый для приема сообщений

Данный класс создает отдельный поток и запускает в нем функцию subscribe созданного объекта: self.client.subscribe(default\_topics). Данная функция имеет следующую реализацию (рис. 5).

|  |
| --- |
| **def** **subscribe**(self, topics**=**None):  **if** topics:  subscribe**.**callback(self**.**on\_message\_print, topics,  hostname**=**'ems.insyte.ru', client\_id**=**master\_username,  auth**=**{'username': master\_user, 'password': master\_pass}, tls**=**None) |

Рисунок 5 – Реализация функции приема сообщений от сервера (оформление подписки)

В результате виртуальный контроллер устанавливает связь с сервером по заданным каналам (темам). И обработка получаемых сообщений будет производиться в функции self.on\_message\_print. На рис. 6 показана реализация данной функции. В ней можно найти обработку сообщения для создания новых устройств.

|  |
| --- |
| **def** **on\_message\_print**(self, client, userdata, message):  **print**("TOPIC {}"**.**format(message**.**topic))  **print**("MSG {}"**.**format(message**.**payload**.**decode('utf8')))  **if** message:  **if** message**.**topic **==** 'master/new\_device':  data **=** json**.**loads(message**.**payload**.**decode('utf8'))  **print**(data, type(data))  **try**:  **if** data['guid'] **not** **in** self**.**devices**.**keys():  dev **=** Device(data['guid'], data['password'])  dev**.**timeout **=** data['timeout']  *# dev.publish\_variables = self.publish\_variables*  **if** data['variables']:  vars\_ **=** data['variables']  **for** v **in** vars\_:  dev**.**add\_variable(Variable(v['name'],  v['type'], v['data\_type'], v['func']))  dev\_thread **=** DeviceThread(guid**=**data['guid'],  dev**=**dev,timeout**=**data['timeout'])  self**.**devices[data['guid']] **=** dev\_thread  dev\_thread**.**start()  self**.**show\_devices()  **else**:  **print**('Device already exists ', data['guid'])  **except** **Exception** **as** e:  **print**('Handling run-time error:', e)  **elif** message**.**topic **==** 'master/stop\_device':  data **=** message**.**payload**.**decode('utf8')  **if** data **and** data **in** self**.**devices**.**keys():  t **=** self**.**devices**.**get(data, None)  **if** t:  **print**("stop: ", type(t), t**.**getName())  t**.**stop()  self**.**devices**.**pop(data)  **else**:  **print**('Didn\'t find guid {}'**.**format(data))  **elif** message**.**topic **==** 'master/show\_devices':  self**.**show\_devices() |

Рисунок 6 – Реализация функции получения и обработки сообщений

Другие файлы программ представлены в приложении А.

## Описание запуска модуля и команд для тестирования

На уровне операционной системы сообщения от сервера можно генерировать с помощью соответствующих утилит, а именно mosquitto\_pub [2] и mosquitto\_sub [3]. Пример отправки сообщения для создания нового устройства показан на рис. 7.

|  |
| --- |
| mosquitto\_pub -h localhost -t 'master/new\_device' -u ems -m '{"guid":"bbbbbbbbb-a508-46cc-a428-1787595d646","password":"12345671","timeout":3,"variables":[{"name":"var1","type":1,"data\_type":1,"func":1}]}' |

Рисунок 7 – Команда для создания нового виртуального устройства

Модуль виртуального контроллера распознает такое сообщение, создаст виртуальное устройство с заданными параметрами, и запустит отправку сообщений по каналу «variables/bbbbbbbbb-a508-46cc-a428-1787595d646».

На сервере можно будет получить и прочитать это сообщение с помощью следующей команды (см. рис. 8).

|  |
| --- |
| mosquitto\_sub -h localhost -t "variables/#" -u ems  variables/bbbbbbbbb-a508-46cc-a428-1787595d646 {"variables": [{"value": -0.4694715617194224, "name": "var1"}], "time": 1522922968} |

Рисунок 8 – Команда для получения сообщений от устройств

Также мы можем с помощью соответствующих команд остановить передачу и просмотреть подключенные устройства (см. рис. 9).

|  |  |  |
| --- | --- | --- |
| mosquitto\_pub  -h localhost  -t "master/show\_devices"  -m '' | mosquitto\_sub  -h localhost  -t "master/devices" master/devices {"count": 1, "devices": ["bbbbbbbbb-a508-46cc-a428-1787595d646"]} | mosquitto\_pub  -h localhost  -t "master/stop\_device"  -m "bbbbbbbbb-a508-46cc-a428-1787595d646" |
| (а) | (б) | (в) |

Рисунок 9 – Различные команды для работы с виртуальным контроллером:   
а – отобразить подключенные виртуальные устройства, б – просмотр результата предыдущей команды, в – отключить устройство с заданным guid

Таким образом, все вышеуказанные команды показывают нормальное функционирование спроектированного и разработанного модуля виртуального контроллера.

# Проектирование модуля управления задачами сбора данных по событиям

Применение методов для сбора данных только по определенным событиям позволит избежать избыточных массивов данных и связанных с ними недостатков, а именно чрезмерной загрузки канала связи, высоких требований к объему долгосрочной памяти (flash-памяти), выполнение избыточных операций по опросу датчиков центральным процессором контроллера.

Такие события, при которых необходимо выполнять сбор данных, принято называть триггерами. Нами были выделены следующие триггеры:

1. Триггер времени (напр., наступление определенного времени или срабатывание таймера).
2. Пороговые триггеры входов контроллера (напр., наступление определенного критического уровня входа).
3. Сигнальные триггеры (напр., получение определенного цифрового сигнала или последовательности).
4. Внешние триггеры (напр., получение команды от внешней системы)
5. Системные триггеры (различные внутрисистемные события, например, появления доступа к сети Интернет, подключение дополнительного модуля и пр.).

Функция сбора данных контроллером связана с функцией передачи данных на сервер, и они могут быть включены или отключены триггерами из описанных выше групп. Также в зависимости от значения триггера можно менять частоту опроса датчиков и отправки данных. На каждый канал собираемых данных можно назначить свои триггеры срабатывания, обеспечив тем самым большую гибкость процесса конфигурирования устройств.

На рис. 10 представлен типовой алгоритм обработки событий триггера.



Рисунок 10 – Схема алгоритма работы модуля сбора данных

# Заключение

В данной работе был спроектирован и разработан модуль виртуального контроллера, показаны основные команды при работе с ним. Благодаря этому модулю можно выполнять разработку серверной части EMS-системы не имея в расположение реального аппаратного контроллера. Он предоставляет широкие возможности для функционального тестирования системы. Модуль может выступать в роли генератора данных по заданным законам распределения величин и обеспечить наполнение базы данных первоначальной информацией для работы с расчетными модулями системы.

Также были определены и классифицированы события, которые могут являться триггерами для включения сбора данных и управления задачами модуля сбора данных.

# Список литературы

1. MQTT software //Eclipse Paho. URL: http://www.eclipse.org/paho/clients/python/ (дата обращения: 10.02.2018).
2. mosquitto\_pub man page. URL: https://mosquitto.org/man/mosquitto\_pub-1.html‎ (дата обращения: 10.02.2018).
3. mosquitto\_sub man page‎. URL: https://mosquitto.org/man/mosquitto\_sub-1.html (дата обращения: 10.02.2018).

# Приложение А Реализация модуля виртуального контроллера

Листинг 1. Главный класс для приема сообщений от сервера и создания виртуальных устройств

**from** threading **import** Thread

**import** threading

**import** paho.mqtt.client **as** mqtt

**import** paho.mqtt.publish **as** publish

**import** json

**from** variable **import** Variable, Type, DataType, Function

**from** device **import** Device

**import** paho.mqtt.subscribe **as** subscribe

**from** apscheduler.schedulers.background **import** BackgroundScheduler, BlockingScheduler

default\_hostname **=** 'ems.insyte.ru'

default\_port **=** 1883

master\_username **=** 'master'

master\_password **=** 'PaSSWoRD'

default\_topics **=** ["sample", "master/new\_device", "master/stop\_device", "master/show\_devices"]

**class** **DeviceThread**(Thread):

**def** \_\_init\_\_(self, guid, dev**=**Device(), timeout**=**1):

"""Инициализация потока"""

Thread**.**\_\_init\_\_(self)

self**.**name **=** guid

self**.**device **=** dev

self**.**time **=** timeout

self**.**scheduler **=** BlockingScheduler()

self**.**scheduler**.**add\_job(dev**.**publish, 'interval', seconds**=**timeout)

self**.**device**.**enable\_sending()

self**.**\_stop\_event **=** threading**.**Event()

**def** **stop**(self):

self**.**scheduler**.**shutdown()

self**.**\_stop\_event**.**set()

**def** **stopped**(self):

**return** self**.**\_stop\_event**.**is\_set()

**def** **run**(self):

"""Запуск потока"""

**print**("run thread DeviceThread:", self**.**name)

self**.**scheduler**.**start()

**class** **Client**:

**def** \_\_init\_\_(self):

self**.**devices **=** dict()

self**.**hostname **=** default\_hostname

self**.**port **=** default\_port

self**.**client **=** mqtt**.**Client(client\_id**=**master\_username, protocol**=**mqtt**.**MQTTv311, clean\_session**=**False)

self**.**client**.**user\_data\_set(master\_username)

self**.**client**.**username\_pw\_set(master\_username, master\_password)

**def** **subscribe**(self, topics**=**None):

**if** topics:

**print**('SUBSCRIBE TOPICS', topics)

subscribe**.**callback(self**.**on\_message\_print, topics,

hostname**=**'ems.insyte.ru', client\_id**=**master\_username **+** "SUBS",

auth**=**{'username': master\_username, 'password': master\_password}, tls**=**None)

**def** **on\_message\_print**(self, client, userdata, message):

**print**("TOPIC {}"**.**format(message**.**topic))

**print**("MSG {}"**.**format(message**.**payload**.**decode('utf8')))

**if** message:

**if** message**.**topic **==** 'master/new\_device':

data **=** json**.**loads(message**.**payload**.**decode('utf8'))

**print**(data, type(data))

**try**:

**if** data['guid'] **not** **in** self**.**devices**.**keys():

dev **=** Device(data['guid'], data['password'])

dev**.**timeout **=** data['timeout']

*# dev.publish\_variables = self.publish\_variables*

**if** data['variables']:

vars\_ **=** data['variables']

**for** v **in** vars\_:

dev**.**add\_variable(Variable(v['name'], v['type'], v['data\_type'], v['func']))

dev\_thread **=** DeviceThread(guid**=**data['guid'],dev**=**dev,timeout**=**data['timeout'])

self**.**devices[data['guid']] **=** dev\_thread

dev\_thread**.**start()

self**.**show\_devices()

**else**:

**print**('Device already exists ', data['guid'])

**except** **Exception** **as** e:

**print**('Handling run-time error:', e)

**elif** message**.**topic **==** 'sample':

**print**("THREADS COUNT {}"**.**format(threading**.**active\_count()))

**for** t **in** threading**.**enumerate():

**if** isinstance(t, DeviceThread) **or** isinstance(t, ClientThread):

**print**(t**.**getName(), t**.**stopped())

**elif** message**.**topic **==** 'master/stop\_device':

data **=** message**.**payload**.**decode('utf8')

**if** data **and** data **in** self**.**devices**.**keys():

t **=** self**.**devices**.**get(data, None)

**if** t:

**print**("stop: ", type(t), t**.**getName())

t**.**stop()

self**.**devices**.**pop(data)

**else**:

**print**('Didn\'t find guid {}'**.**format(data))

**elif** message**.**topic **==** 'master/show\_devices':

self**.**show\_devices()

**def** **on\_log**(self, client, userdata, level, buf):

str\_ **=** "LOG: {} [{}]: {}"**.**format(userdata, level, buf)

**print**(str\_)

*# self.log(str)*

**pass**

**def** **show\_devices**(self):

msg **=** dict()

msg["count"] **=** len(self**.**devices)

msg['devices'] **=** []

**for** k **in** self**.**devices**.**keys():

msg['devices']**.**append(str(k))

publish**.**single("master/devices", payload**=**str("{}"**.**format(json**.**dumps(msg))),

hostname**=**self**.**hostname, port**=**self**.**port, client\_id**=**master\_username,

auth**=**{'username': master\_username, 'password': master\_password}, tls**=**None)

**def** **log**(self, msg):

publish**.**single("master/log", payload**=**str("{}"**.**format(msg)),

hostname**=**self**.**hostname, port**=**self**.**port, client\_id**=**master\_username,

auth**=**{'username': master\_username, 'password': master\_password}, tls**=**None)

**class** **ClientThread**(Thread):

**def** \_\_init\_\_(self, name):

"""Инициализация потока"""

Thread**.**\_\_init\_\_(self)

self**.**name **=** name

self**.**client **=** Client()

self**.**\_stop\_event **=** threading**.**Event()

**def** **stop**(self):

self**.**\_stop\_event**.**set()

**def** **stopped**(self):

**return** self**.**\_stop\_event**.**is\_set()

**def** **run**(self):

"""Запуск потока"""

**try**:

self**.**client**.**subscribe(default\_topics)

**except** **KeyboardInterrupt**:

**print**("exiting")

**if** \_\_name\_\_ **==** '\_\_main\_\_':

thread **=** ClientThread('master')

thread**.**start()

Листинг 2. Класс Device для работы с виртуальными устройствами

**import** json

**from** variable **import** Variable, Type, DataType, Function

**import** sched

**import** time

**import** paho.mqtt.publish **as** publish

default\_timeout **=** 10

default\_hostname **=** 'ems.insyte.ru'

default\_port **=** 1883

master\_username **=** 'master'

master\_password **=** 'PaSSWoRD'

**class** **Device**:

**def** \_\_init\_\_(self, guid**=**None, password**=**None):

self**.**guid **=** guid

self**.**password **=** password

self**.**variable\_names **=** []

self**.**variables **=** []

self**.**timeout **=** default\_timeout

self**.**time **=** None

self**.**sending **=** False

self**.**publish\_variables **=** None

**def** **add\_variable**(self, var**=**Variable()):

self**.**variable\_names**.**append(var**.**name)

self**.**variables**.**append(var)

**def** **serialize**(self):

result **=** {

'time': self**.**time,

'variables': []

}

**for** v **in** self**.**variables:

result['variables']**.**append(v**.**serialize\_to\_dict())

**return** result

**def** **publish**(self):

self**.**time **=** int(time**.**time())

**for** v **in** self**.**variables:

v**.**update()

**try**:

publish**.**single("variables/{}"**.**format(self**.**guid),

payload**=**json**.**dumps(self**.**serialize()),

hostname**=**default\_hostname,

port**=**default\_port,

client\_id**=**self**.**guid,

auth**=**{'username': self**.**guid, 'password': self**.**password},

tls**=**None)

**except** **Exception** **as** err:

**print**('Handling run-time error:', err)

Листинг 3. Класс Variable для генерации данных переменных устройства

**from** enum **import** Enum, IntEnum

**import** json

**import** math

**import** time

**class** **Type**(Enum):

IN **=** 1

INOUT **=** 2

OUT **=** 3

**class** **DataType**(Enum):

NUMBER **=** 1

STRING **=** 2

LIST **=** 3

**class** **Function**(Enum):

SIN **=** 1

COS **=** 2

TAN **=** 3

**class** **Variable**:

**def** \_\_init\_\_(self, name**=**None, type\_**=**1, data\_type**=**1, function\_**=**1):

self**.**name **=** name

self**.**type **=** Type(type\_)

self**.**data\_type **=** DataType(data\_type)

self**.**function **=** Function(function\_)

self**.**value **=** None

self**.**timeout **=** None

**def** **serialize\_to\_dict**(self):

**return** {

'name': str(self**.**name),

'value': self**.**value

}

**def** **update**(self):

time\_ **=** int(time**.**time())

**if** self**.**function **==** Function**.**SIN:

self**.**value **=** math**.**sin(math**.**radians(time\_))

**elif** self**.**function **==** Function**.**COS:

self**.**value **=** math**.**cos(math**.**radians(time\_))

**else**:

self**.**value **=** math**.**tan(math**.**radians(time\_))