# Глоссарий

**Атрибуты устройства** (device attributes)– конфигурируемые значения устройства IoT (например, частота опроса, IP Адрес получения данных, режимы работы входов и др.), задаются в режиме инициализации интегратором и могут быть изменены в дальнейшем им же (если не сказано иное).

**Переменные устройства** (device variables) – данные, полученные от подключенных к устройству внешних источников данных (датчиков и иного промышленного оборудования), а также данные и команды отправляемые им.

**Измерения** – значения переменных устройства, полученные от источников данных (датчиков) в определенный момент времени (временная метка и значение)

# Проектирование пользовательских интерфейсов

Один из основных интерфейсов – интерфейс добавления новых устройств. Полностью он представлен в приложении Г. Ниже приведем моменты касательно ввода настроек для подключения устройств к серверу с помощью различных протоколов, рассмотренных ранее.

Для подключения устройства к серверу по протоколу MQTT используется форма ввода, представленная на рис. 26.

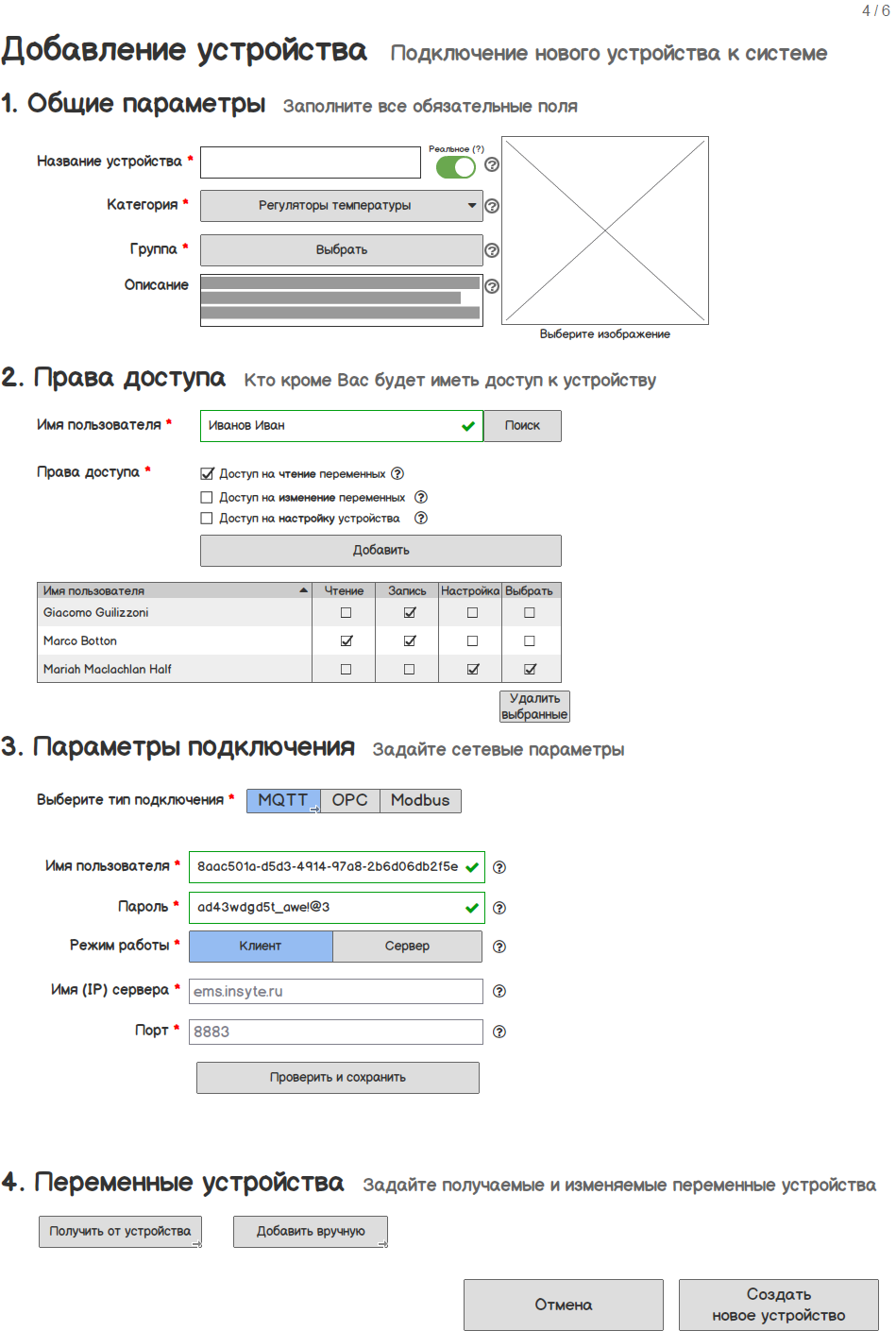


Рисунок 26 – Процесс настройки параметров подключения устройства по MQTT

## Пользователь добавляет в систему свое реальное устройство

1. Пользователь авторизуется в системе.
2. Пользователь нажимает кнопку «Добавить (создать, подключить?) устройство»
3. Система проверяет наличие прав на добавление новых устройств
4. Система отображает окно ввода параметров нового устройства
5. Пользователь заполняет обязательные поля (см. схему БД), категорию устройства
6. Пользователь выбирает протокол обмена (MQTT, Modbus, OPC):

Был выбран MQTT:

* 1. Система отображает форму для настройки подключения
  2. Пользователь заполняет настройки подключения: IP, Порт
  3. Система выдает DevGUID и пароль пользователю

Был выбран OPC:

1. Система отображает форму для настройки подключения
2. Пользователь заполняет настройки подключения к серверу: IP, порт, логин, пароль, сертификаты SSL
   1. Система подключается к серверу и получает список тегов
   2. Система отображает список тегов пользователю и предлагает добавить их в систему
   3. Пользователь создает соответствующие переменные на основе тегов

Был выбран Modbus:

1. Система отображает форму для настройки подключения
2. Пользователь заполняет настройки подключения к серверу: IP, порт
3. Пользователь устанавливает разметку областей памяти устройства (создает переменные, которые будут ссылаться на некоторые участки памяти)
4. Система запоминает настройки и на их основе создаст переменные после создания устройства
5. Пользователь нажимает «ОК/Добавить»
6. Система сохраняет конфигурацию
7. Пользователь подключает устройство к системе с выданными учетными данными
8. Система получает настройки от устройства с параметрами переменных
9. Система отправляет статус о том, что устройство было подключено и сконфигурировано или нет
10. Пользователь переходит в настройки устройства и видит список переменных или пустой список
11. Пользователь может изменить параметры переменных или создать новые переменные

**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.

# Дополнительные варианты примеров сообщений

Листинг 3. Пример сообщения, содержащего несколько измерений

[

{

"timestamp": "2017-12-02T10:01:06Z",

"value": 22.4

},

{

"timestamp": "2017-12-02T10:01:16Z",

"value": 28.4

},

{

"timestamp": "2017-12-02T10:01:26Z",

"value": 31.5

}

]

Листинг 4. Пример сообщения, содержащего одновременное измерение нескольких величин

{

"timestamp": "2017-12-02T10:01:06Z",

"variables": [

{"name": "input1",

"value": 22.4},

{"name": "input2",

"value": 28.4},

{"name": "input3",

"value": 31.5}

]

}

Листинг 5. Пример файла, хранящего значения измерений

timestamp;value

"2017-12-02T10:01:06Z";"22.4"

"2017-12-02T10:01:16Z";"25.1"

"2017-12-02T10:01:26Z";"28.3"

Листинг 6. Пример файла, хранящего значения нескольких связанных измерений

timestamp;input1;input2;input3

"2017-12-02T10:01:06Z";"22.4";"23.4";"29"

"2017-12-02T10:01:16Z";"25.1";"20.4";"34"

"2017-12-02T10:01:26Z";"28.3";"26.1";"44"