ПРОТЕЙ СИСТЕМЫ

PromUC

Руководство пользователя

Выпуск 1.0

www.protei-sm.ru

дек. 08, 2022

Оглавление

1 Работа с системой **1**

1.1 Назначение визуализации . . . 1
1.2 Демонстрационный проект . . . 1
1.3 Просмотр визуализации . . . 1
1.4 Хранение данных конфигурации проекта . . . 8
1.5 Настройка визуализации . . . 9
1.6 Настройка визуализации в интерфейсе системы контроля версий . . . 15
1.7 Источники данных . . . 18
1.8 Создание панелей (элементов) управления . . . 18
1.9 Управление объектами . . . 22
1.10 Просмотр журнала событий . . . 22
1.11 Реакция на аварийные сообщения системы . . . 22

2 Архитектура визуализации **23**

2.1 Используемые Web технологии . . . 23
2.2 Библиотека элементов в формате SVG . . . 23
2.3 Подготовка SVG элементов . . . 24
2.4 Примеры создания простых элементов . . . 24
2.4.1 Создание бинарного элемента . . . 24
2.4.2 Создание текстового элемента . . . 27
2.4.3 Создание элемента с мультисостояниями . . . 29

3 Формирование и ведение данных **47**

3.1 Правила подготовки данных . . . 47
3.1.1 Классификация объектов . . . 47
3.1.2 Основные Классы . . . 48
3.1.3 Классификация topic mqtt . . . 49
3.2 Порядок и средства заполнения базы данных . . . 49
3.2.1 Системные Классы и Объекты . . . 50
3.2.2 Пользовательские Классы и Объекты . . . 50
3.2.3 Системная модель данных . . . 51
3.2.4 Основные решения Framework . . . 53

3.3 Процедуры изменения и контроля базы данных . . . 53
3.3.1 API . . . 53
3.3.2 Фильтры . . . 54
3.3.3 Удаление объектов . . . 55
3.4 Порядок и средства восстановления базы данных . . . 55

4 Конфигурирование правил и сценариев 56
4.1 Введение . . . 56
4.1.1 Сценарии использования . . . 56
4.1.2 Запуск Rule Engine . . . 57
4.1.3 Пример использования - добавление узла проверки температуры . . . 57
4.1.4 Типы сообщений внутри Rule Engine . . . 59
4.2 Узлы входящих сообщений . . . 61
4.2.1 Узел HTTP Input . . . 61
4.2.2 Узел Mercury . . . 61
4.2.3 Узел ModbusTCP . . . 63
4.2.4 Узел BACnet . . . 66
4.3 Узлы фильтрации и маршрутизации . . . 66
4.3.1 Узел проверки типа и направления сообщения . . . 66
4.3.2 Узел проверки наличия полей сообщения . . . 67
4.3.3 Узел проверки типа сообщения . . . 67
4.3.4 Узел выбора типа сообщений . . . 68
4.3.5 Узел проверки типа отправителя сообщений . . . 68
4.3.6 Узел выбора сообщений по типу отправителя . . . 69
4.3.7 Узел скрипт - фильтр . . . 69
4.4 Узлы насыщения . . . 70
4.4.1 Узел вычисления дельты . . . 70
4.4.2 Узел атрибутов получателя . . . 73
4.4.3 Узел атрибутов устройства . . . 74
4.4.4 Узел атрибутов отправителя . . . 75
4.4.5 Узел полей отправителя . . . 76
4.4.6 Узел атрибутов связей . . . 77
4.5 Узлы трансформации . . . 78
4.5.1 Узел изменения отправителя . . . 78
4.5.2 Узел скрипт . . . 79
4.5.3 Узел подготовки сообщения электронной почты . . . 80
4.6 Узлы действий . . . 82
4.6.1 Узел Тревога (Alarm) . . . 82
4.6.2 Узел снятия тревоги . . . 85
4.6.3 Узел задержки (Delay) . . . 88
4.6.4 Узел генератор (Generator) . . . 88
4.6.5 Узел логирования (Log Node) . . . 90
4.6.6 Узел ответа на вызов RPC . . . 90
4.6.7 Узел запроса вызова RPC . . . 91
4.6.8 Узел сохранения атрибутов . . . 92
4.6.9 Узел сохранения временных рядов . . . 93
4.6.10 Узел «Сохранить в базу данных» . . . 95
4.6.11 Узел «Назначить узлу клиента» . . . 96

4.6.12 Узел «Снять назначение узла от клиента» . . . 97
4.6.13 Создать узел связи . . . 98
4.6.14 Удалить узел связи . . . 99
4.6.15 Узел событий GPS-геозоны . . . 100
4.6.16 Узел «Отправить в облако» . . . 103
4.6.17 Узел «Отправить в периферию» . . . 104
4.7 Узлы публикации . . . 105
4.7.1 Узел Kafka . . . 105
4.7.2 Узел MQTT . . . 107
4.7.3 Узел REST API . . . 108
4.7.4 Узел отправки Email . . . 109
4.7.5 Узел отправки SMS . . . 111

Глава 1

Работа с системой

1.1 Назначение визуализации

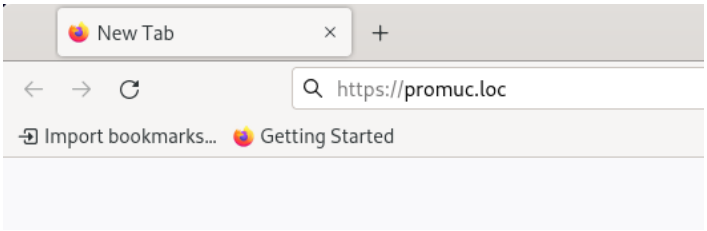
Визуализация позволяет отображать текущее состояние элементов с динамическим его изменением во времени, управлять установками и системами, отправлять команды непосредственно с графических планов. Данный подход упрощает и ускоряет работу оператора, повышает удобство работы, улучшает восприятие текущей обстановки и уменьшает время реакции на события.

1.2 Демонстрационный проект

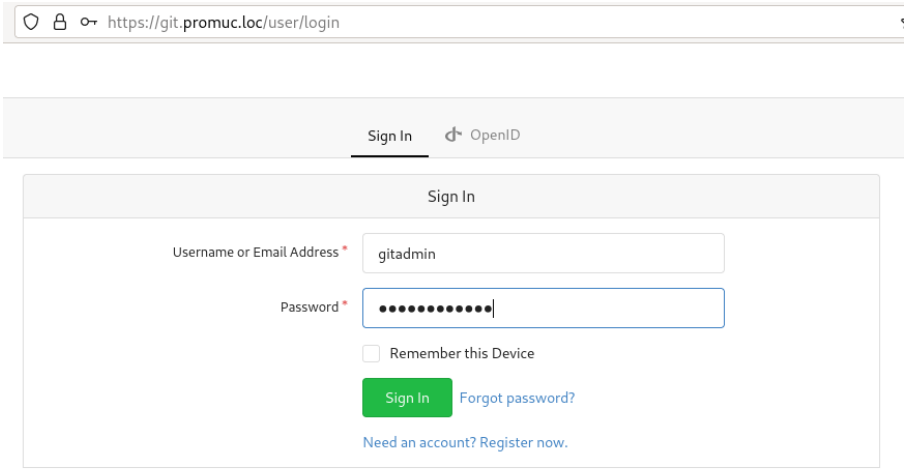
После установки системы ее конфигурация содержит демо-проект, который позволяет посмотреть возможности системы. Данный проект также можно использовать за основу при изучении системы, а также создании своего собственного проекта.

1.3 Просмотр визуализации

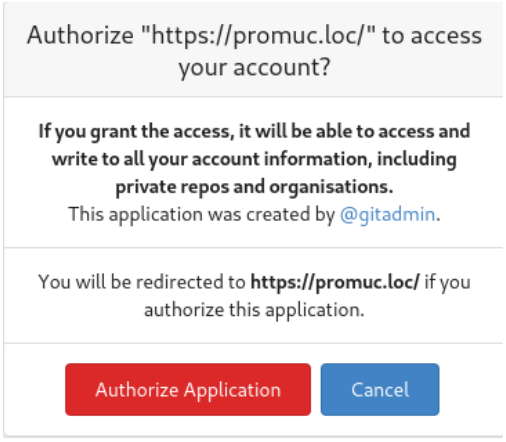
Для запуска интерфейса пользователя необходимо в адресной строке web браузера ввести адрес `https://имядомена`, например, `https://promuc.loc`:



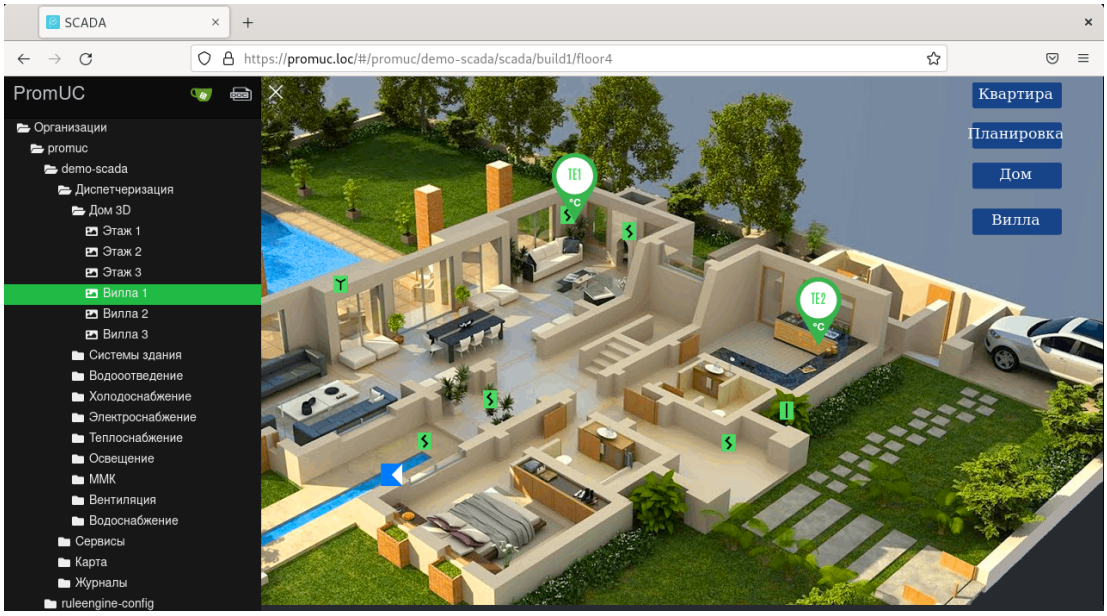
Далее необходимо ввести учётные данные, которые были сгенерированы во время установки системы:



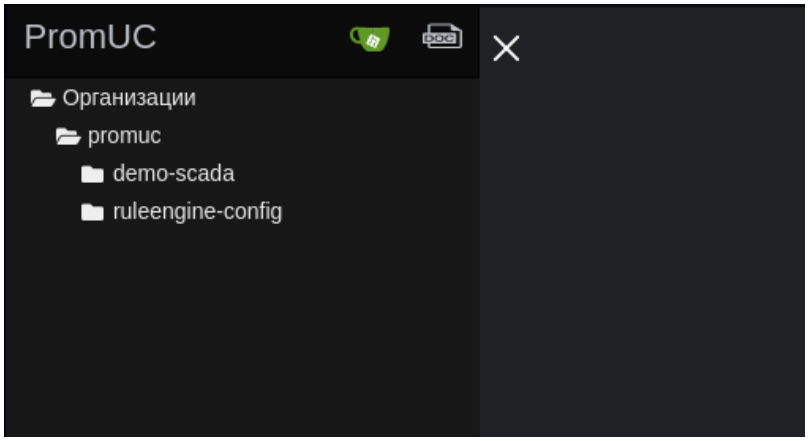
и подтвердить авторизацию приложения (только при первом запуске):



Интерфейс пользователя откроется в окне web браузера:



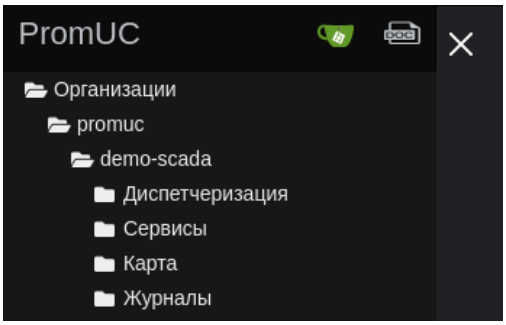
Интерфейс пользователя содержит сворачиваемую панель обзора, в которой в виде иерархического дерева представлена структура проекта:



Представляемая в панели информация делится на уровни иерархии:

- организации
- проекты организации
- данные проекта (могут содержать вложенные папки)

Демо-проект `demo-scada` относится к организации `promuc` и содержит следующие разделы:



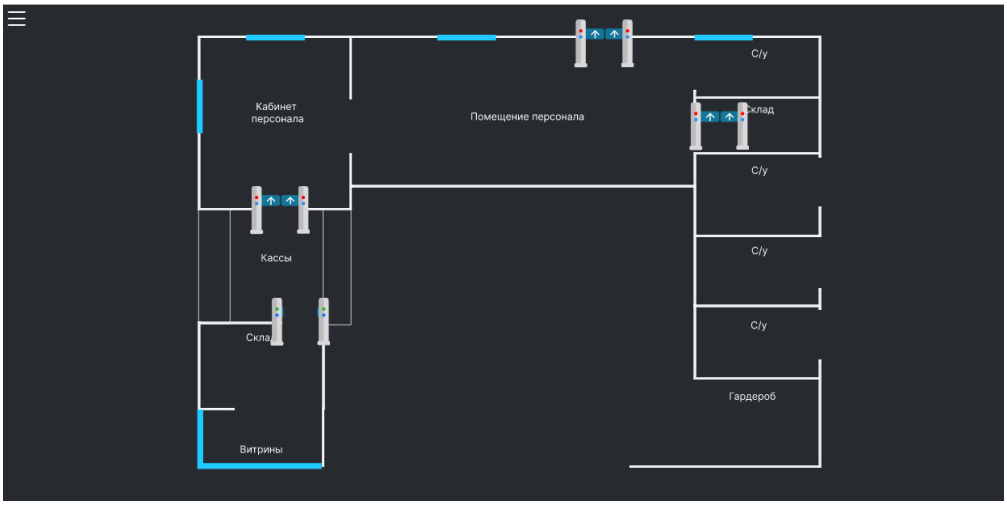
- Диспетчеризация, содержит различные системы, сгруппированные по типу
- Сервисы, содержит ссылки на дополнительные ресурсы
- Карта, содержит данные ГИС в виде слоев и маркеров
- Журналы, содержат настраиваемые отчеты по проекту

Диспетчеризация

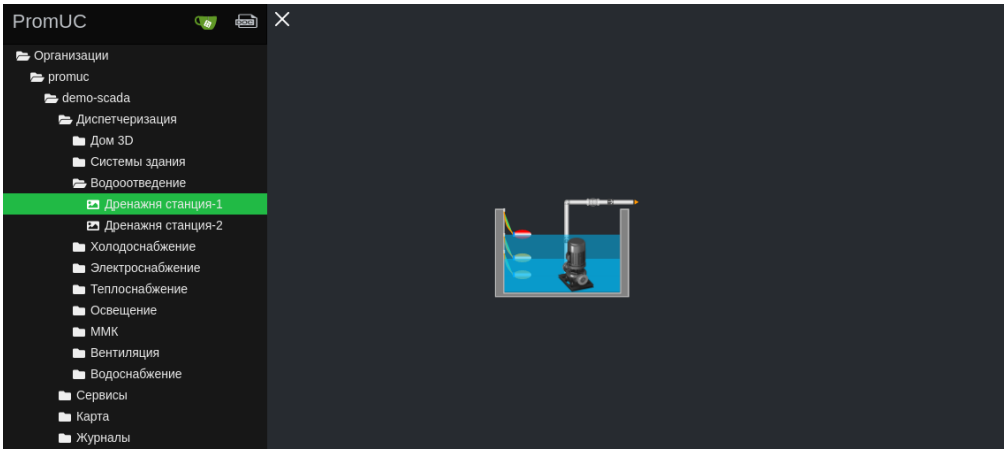
Содержит следующие разделы:

- Дом 3D, отображает размещение и состояние эмулированных устройств на 3-х мерном плане помещений и зданий (вилл) с маркерами и ссылками перехода на другие визуализации:

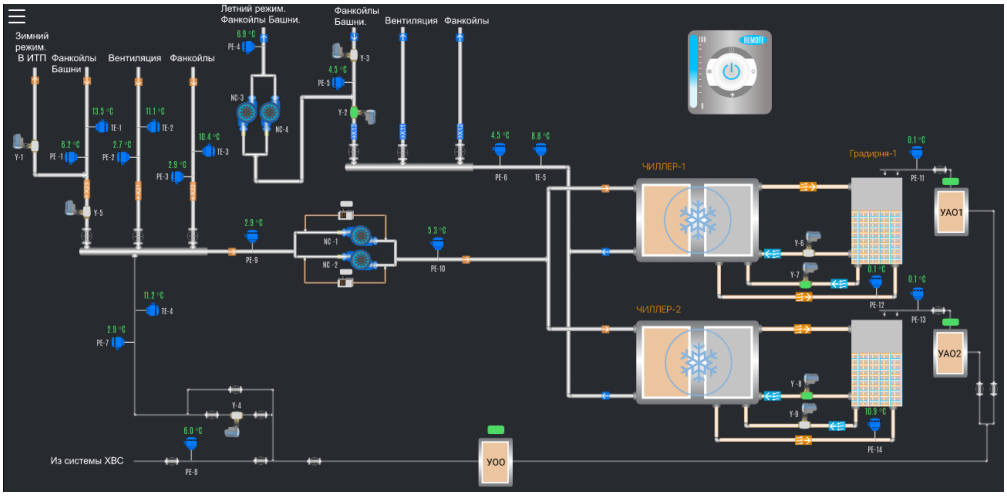
- Системы здания на примере системы контроля и управления доступом (панель свёрнута):



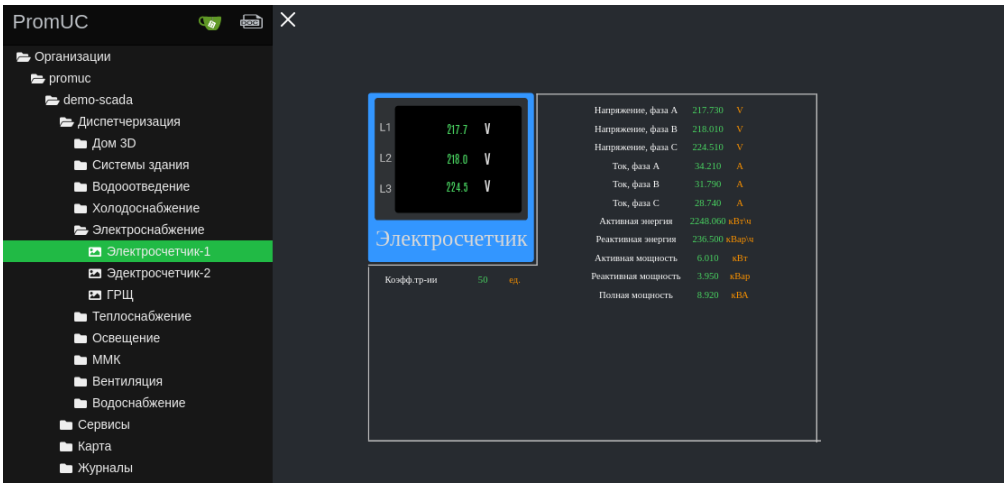
- Водоотведение:



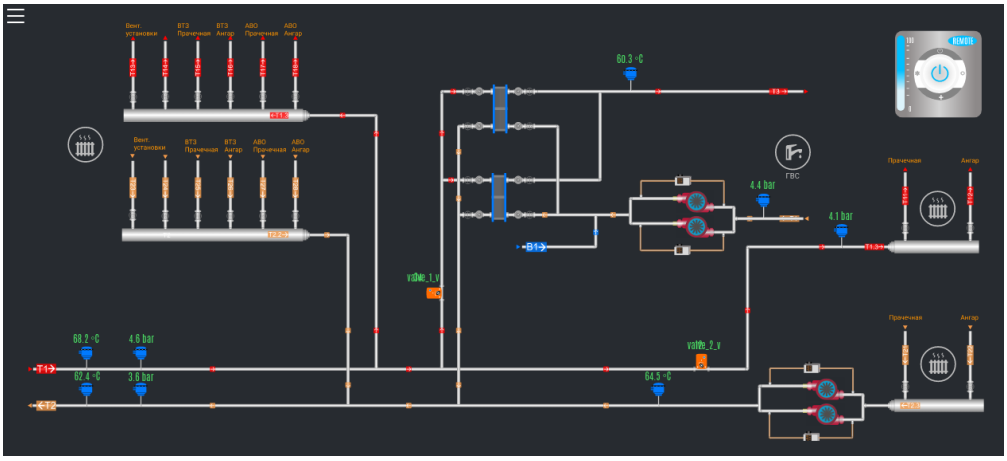
- Холодоснабжение (панель свёрнута):



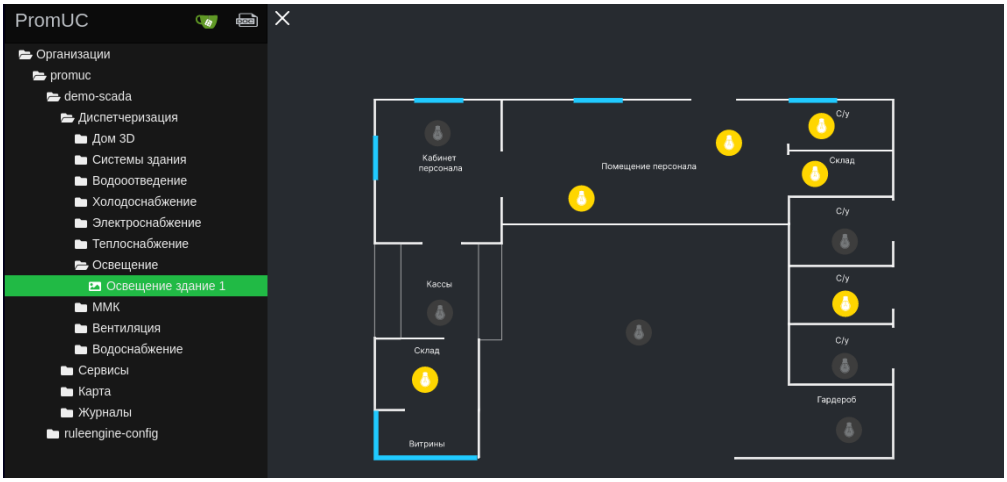
- Электроснабжение:



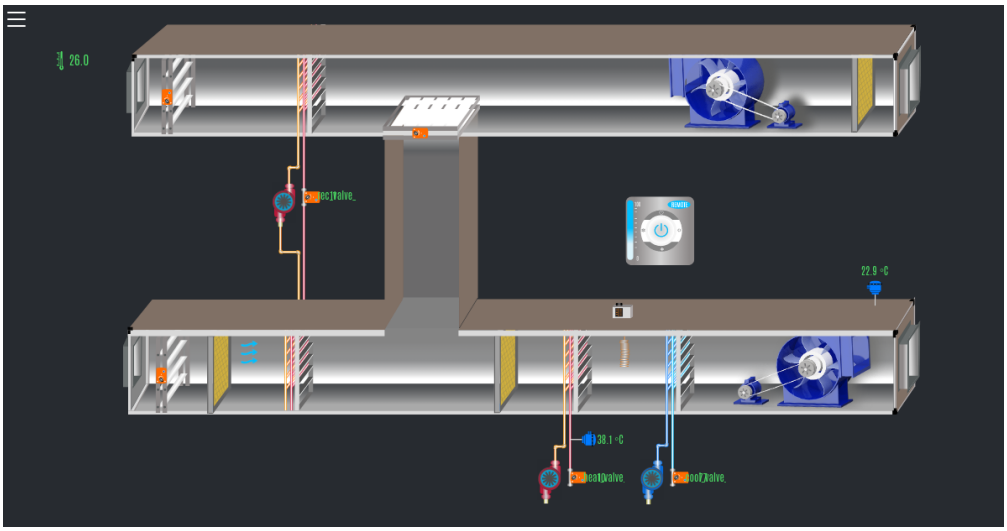
- Теплоснабжение (панель свёрнута):



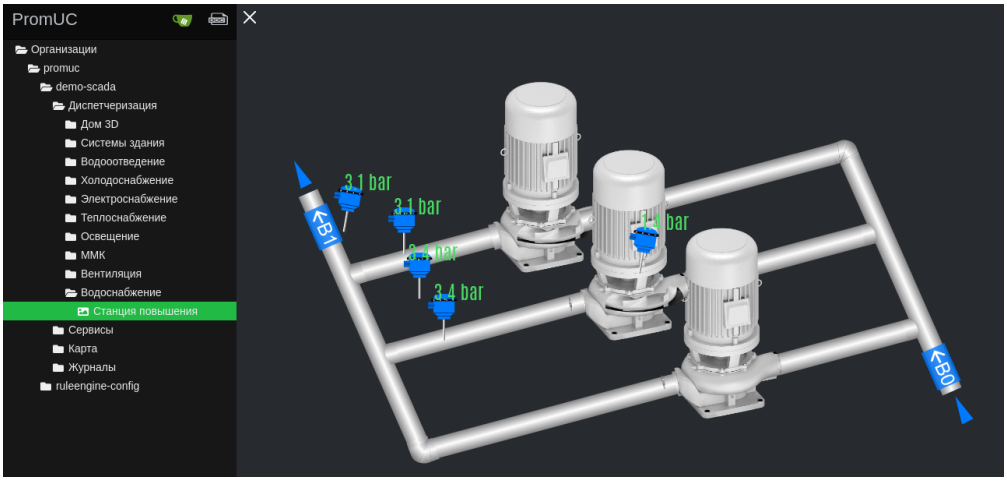
- Освещение:



- Вентиляция (панель свёрнута):



- Водоснабжение:

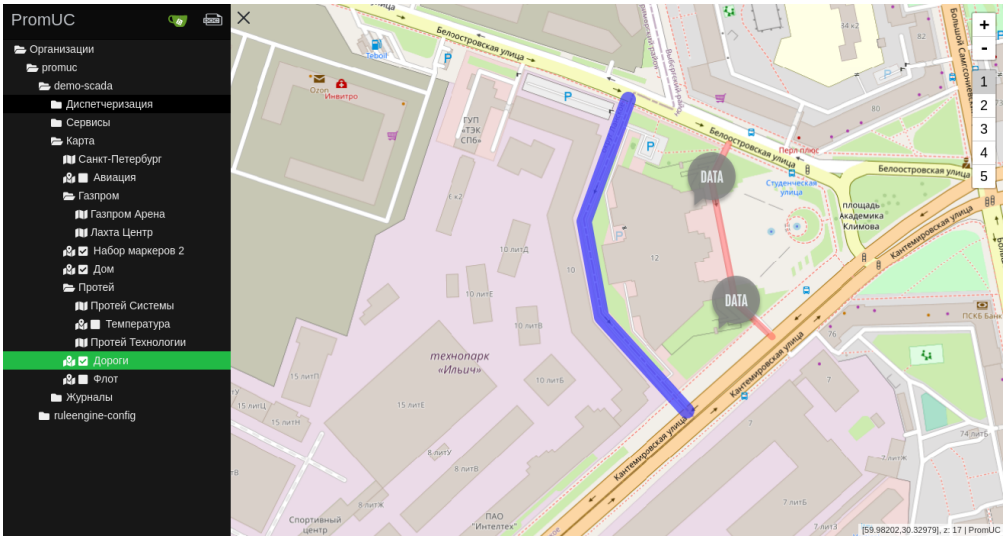


Сервисы

- Ссылка на расширенную документацию PromUC в браузере
- Ссылка на сайт

Карта

Является ГИС. Может использовать как локальный, так и удалённый сервис. В данном демо-проекте для работы требуется подключение к сети Интернет.



- Отображение местности (OSM)
- Отображение маркеров с настройкой их видимости
- Отображение движущихся объектов с всплывающей информацией о них (на примере воздушного и морского транспорта)
- Отображение и работа с маршрутоами
- Привязка к местоположению
- Различные слои с настраиваемой видимостью в зависимости от масштаба
- Выбор отображаемых в данный момент времени слоёв
- Отображение поэтажных планов с ссылками перехода на этажи

Журналы

Позволяют получать отчёты по любым данным в системе в связанном виде.

1.4 Хранение данных конфигурации проекта

Данные конфигурации проекта могут храниться как в файловой системе в виде файлов, а также дополнительно могут находиться под управлением системы контроля версий.

Можно получить доступ к файлам открыв их редактором или, при наличии системы контроля версий, в интерфейсе системы контроля версий.

1.5 Настройка визуализации

Для создания собственной структуры системы диспетчеризации требуется выполнить следующие действия:

1. Используя терминал Linux, перейти в папку `/opt/services/scada/data/source`:

```
cd /opt/services/scada/data/source
```

   В этой директории находятся файлы, которые используются для создания и редактирования мнемо-схем (отображения данных). Внутри этой директории находятся 3 папки: `"svg", "visio", "scada"`:

   - В папке `"svg"` находится библиотека графических элементов, для создания визуализаций, в формате `svg`.

     Структура папки имеет следующий вид:

     svg - Корневая папка svg библиотеки

     elements - Папка с графическими элементами

     ventilation- Элементы для конструирования вент-установок

     electricity - Элементы для конструирования схем электроснабжения

     sensors - Различные датчики и измерения

     pumps - Насосы

     primitive - Примитивы для создания собственных простых элементов

     valves - Регулирующие клапаны

     water - Специфические элементы для конструирования систем водоснабжения

     alarms - Элементы отображения аварий

     canalization - Элементы для конструирования систем водоотведения

     controls - Элементы управления

     labels - Надписи (текстовые элементы)

     pipe - Трубы и различные трубные элементы

     icons - Иконки для панелей управления

     Можно добавлять папки в директорию `svg/elements`, редактировать существующие элементы библиотеки, создавать свои.

     Можно создавать любую структуру папок внутри катологa `"svg"`.

- В папке "visio" находятся шаблоны визуализаций.

  Необходимо создать структуру, в зависимости от нужд предприятия. Например:

  visio - Корневой каталог шаблонов визуализаций (НАЗВАНИЕ "visio" МЕНЯТЬ НЕЛЬЗЯ!)

  electricity - в этой папке шаблоны для схем электроснабжения

  heating - в этой папке шаблоны теплоснабжения

  ventilation - в это папке шаблоны вент-установок

  и т.п.

  Можно создавать любую структуру папок внутри катологa "visio"

  Шаблон - готовая визуализация, созданая в редакторе визуализаций, без связанных сигналов (источника данных), для многократного использования. Отпадает необходимость создавать десятки подобных визуализаций.

  При создании шаблона в редакторе, автоматически создается .yaml-файл, который, при необходимости, можно редактировать:

```
---
# блок кода №1
self: Visio:lightning/house_1
description: Освещение. Здание №1
elements:

# блок кода №2
- self: schemes
  description: Планировка
  iconUrl: Svg:schemes/house_1
  crd:
    - 403
    - 46
  replace: []
  order: 0
  scale:
    - 1
    - 1
  rotate: 0
  name: schemes

# блок кода №3
- self: lamp1
  description: Точка освещения 1
  iconUrl: Svg:elements/electriciy.lamp
  crd:
    - 1382
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    - 375
  replace:
    STATE: lamp-1-state
  order: 6
  scale:
    - 1
    - 1
  rotate: 0
  name: lamp1

# блок кода №4
- self: lamp2
  description: Точка освещения 2
  iconUrl: Svg:elements/electriciy.lamp
  crd:
    - 1383
    - 479
  replace:
    STATE: lamp-2-state
  order: 7
  scale:
    - 1
    - 1
  rotate: 0
  name: lamp3
```

Блок кода №1

self: Название визуализации, так же является ее URL

description: Описание (название) визуализации

elements: Содержит в себе все элементы визуализации

Блок кода №2

self: Название элемента (латиницей), системное.

description: Название элемента (латиницей, кириллицей) для пользователя. Отображается в интерфейсе пользователя.

iconUrl: url элемента библиотеки

crd: координаты элемента на экране (x, y)

replace: [В данном элементе не используется]

order: порядковый номер элемента. Так же является графическим слоем SVG

scale: масштабирование элемента (x,y)

rotate: угол на который повернут элемент, относительного исходного отображения

name: Название элемента (латиницей), системное.

Блок кода №3 и №4

self: Название элемента (латиницей), системное.

description: Название элемента (латиницей, кириллицей) для пользователя. Отображается в интерфейсе пользователя.

iconUrl: url элемента библиотеки

crd: координаты элемента на экране (x, y)

replace: Ссылка для связанной переменной

order: порядковый номер элемента. Также является графическим слоем SVG

scale: масштабирование элемента (x,y)

rotate: угол, на который повернут элемент относительно исходного отображения

name: Название элемента (латиницей), системное.

- В папке "scada" находятся визуализации (шаблоны с подключенным источником данных).

  Необходимо создать структуру, в зависимости от нужд предприятия. Например:

  scada - Корневой каталог шаблонов визуализаций с источником данных (НАЗВАНИЕ "scada" МЕНЯТЬ НЕЛЬЗЯ!)

  electricity - в этой папке визуализации схем электроснабжения

  heating - в этой папке визуализации теплоснабжения

  ventilation - в это папке визуализации вент-установок

  и т.п.

  Можно создавать любую структуру папок внутри катологa "scada"

  Чтобы создать визуализацию в «scada» необходимо создать .yaml-файл.

  Задача:

  создать визуализацию вент-установки П-1, которая будет храниться в каталоге визуализаций scada/ventilation и привязать к ней источник данных

  Решение:

  переходим в папку /opt/services/scada/data/source/scada/ventilation и создаем файл ahu-p1.yaml

```
---
description: Вент-установка П-1
reference: Visio:ventilation/ahu_i
data: "mqtt:ws://user@user:mq.4iot.pro:15675/ws#data//ahu"
...
```

«**description:**» - Название визуализации, отображаемое в интерфейсе пользователя PromUC (в данном случае это будет Вентиляция/Вент-установка П-1).

«**reference:**» - Указывает какой шаблон визуализации будет использован (в данном случае "visio/ventilation/ahu_i.yaml")

«**data:**» - источник данных (в данном случае это данные получаемые по протоколу mqtt). Источники данных могут быть локальными и внешними.

- В папке "pult" находятся созданные шаблоны панелей управления (шаблон без источников данных).

  Необходимо создать файлы формата .yaml, добавить требуемый набор элементов управления. Например:

```
---
description: Управление ИТП
elements:

  # Элемент №1
- name: Уставка температуры ГВС
  type: analog
  data: GVS_TEMP_SET
  min: 40
  max: 65
  step: 1

  # Элемент №2
- name: Уставка температуры отопления
  type: analog
  icon: Svg:icons/sq2
  data: OT_TEMP_SET
  min: 50
  max: 80
  step: 1

  # Элемент №3
- name: Режим работы насоса Н-1
  type: binary
  icon_on: Svg:icons/on
  icon_off: Svg:icons/off
```

(continues on next page)

(продолжение с предыдущей страницы)

```
  data: PUMP_1

  # Элемент №4
- name: Режим работы насоса Н-2
  type: multistate
  data: SEASON_MODE
  states:
     1: Зима
     2: Лето
     3: Авто

  ...
```

«description» - Название панели управления.

«elements» - набор элементов управления в панели.

Элемент №1:

«name:» - Название элемента управления. Отображается в панели как описание функционального назначения элемента.

«type:» - Тип элемента управления. Поддерживаемые типы: `analog` - числовое управление, `binary` - дискретное управление, `multistate` - выбор из списка значений.

«data:» - Глобальная переменная - для связывания с источником данных. Источник данных привязывается в файле `scada/{any}.yaml`

«min:» (только для **«type»** - analog) - Минимальное используемое значение

«max:» (только для **«type»** - analog) - Максимальное используемое значение

«step:» (только для **«type»** - analog) - Шаг изменения значения

Элемент №3:

«name:» - Название элемента управления. Отображается в панели как описание функционального назначения элемента.

«type:» - Тип элемента управления. Поддерживаемые типы: `analog` - числовое управление, `binary` - дискретное управление, `multistate` - выбор из списка значений.

«data:» - Название для связанной переменной. Источник данных привязывается в файле `scada/{any}.yaml`

«icon_on:» (только для **«type»** - binary) - Иконка для отображения состояния логической **«1»**

«icon_off:» (только для **«type»** - binary) - Иконка для отображения состояния логической **«0»**

Элемент №4:

«name:» - Название элемента управления. Отображается в панели, как описание функционального назначения элемента.

«type:» - Тип элемента управления. Поддерживаемые типы: `analog` - числовое управление, `binary` - дискретное управление, `multistate` - выбор из списка значений.

«data:» - Название для связанной переменной. Источник данных привязывается в файле `scada/{any}.yaml`

«state:» (только для **«type»** - multistate) - Список значений для выбора.

«1:» - Описание варианта «1», отображается в выпадающем списке панели управления

«2:» - Описание варианта «2», отображается в выпадающем списке панели управления

«3:» - Описание варианта «3», отображается в выпадающем списке панели управления

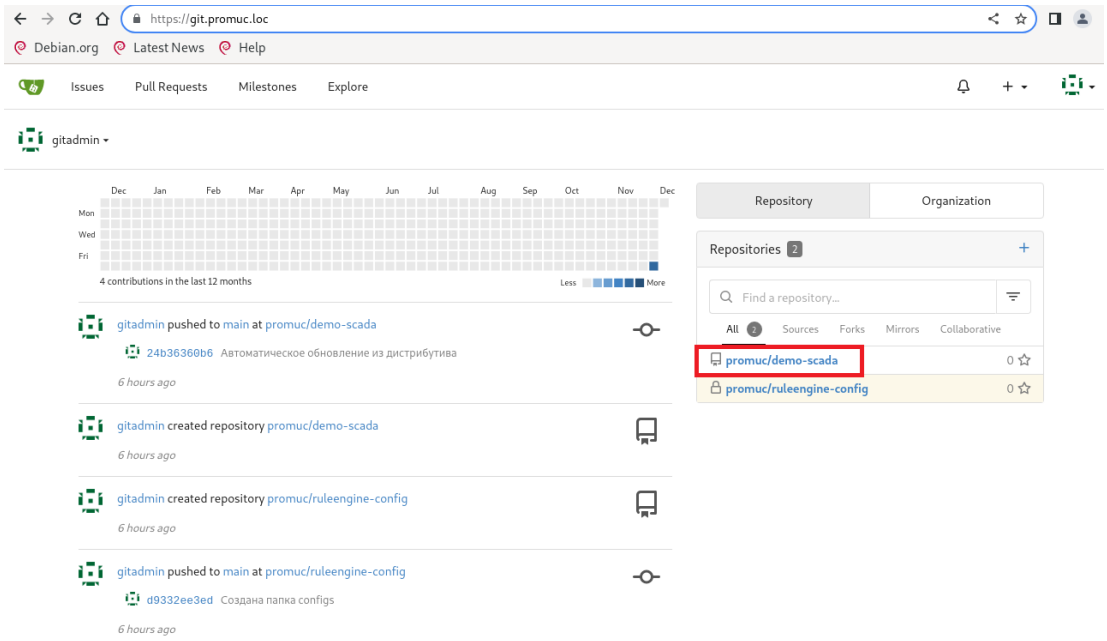
После начальной настройки структуры можно приступить к использованию ПО

1.6 Настройка визуализации в интерфейсе системы контроля версий

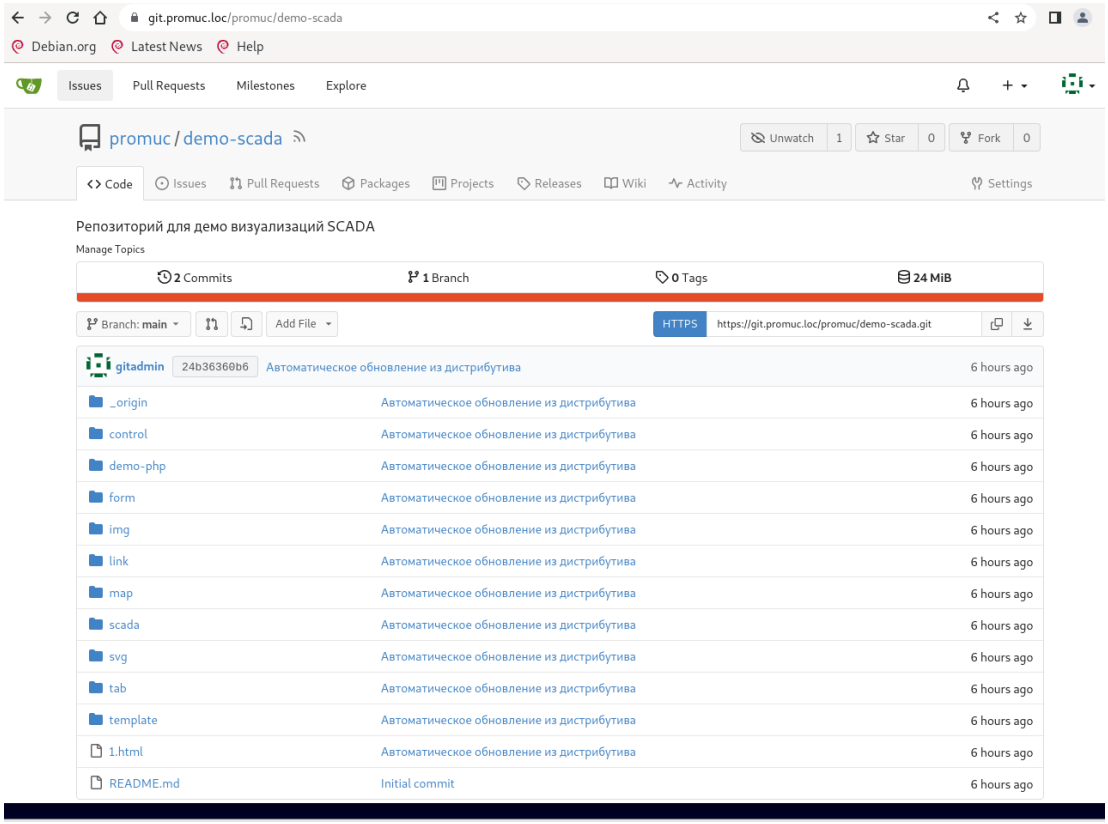
Если используется система контроля версий, то действия описанные выше можно выполнить в web браузере перейдя по ссылке `https://git.promuc.loc/` под учётными данными, которые были сгенерированы при установке системы.

Преимуществом системы контроля версий является отслеживание и фиксация всех изменений с возможностью просмотра истории изменений и отката в случае ошибок.

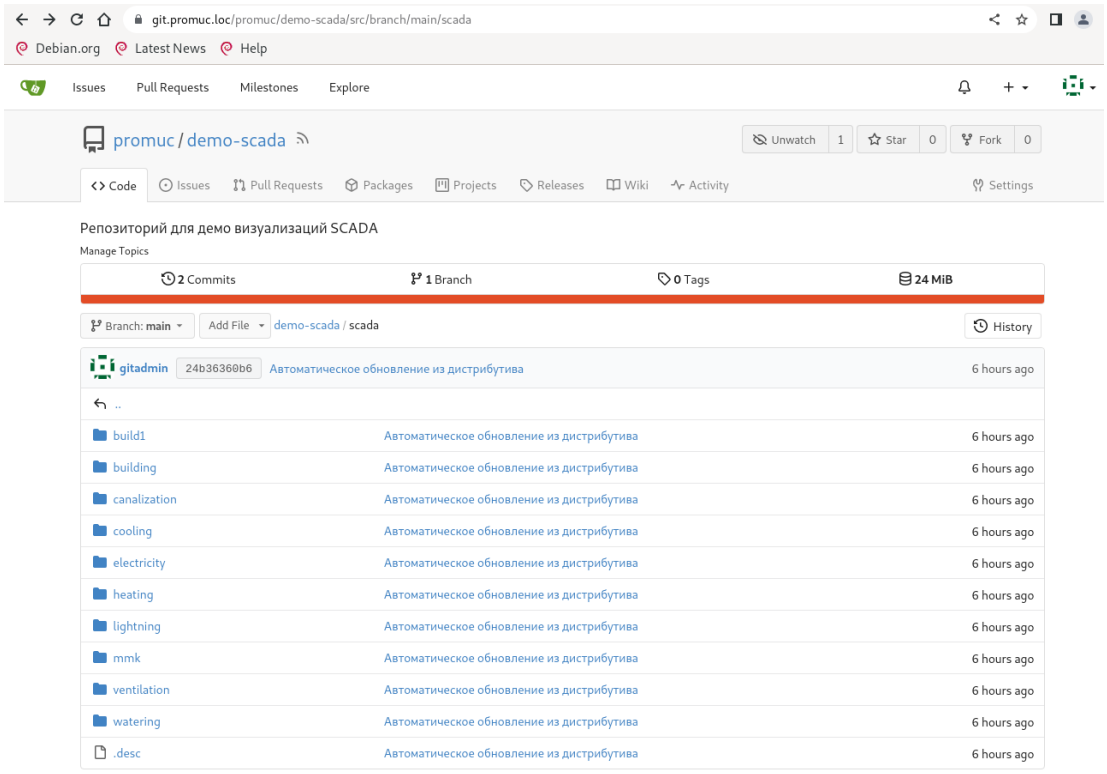
Нужно выбрать существующий или создать новый репозиторий. Для просмотра имеющейся конфигурации демо-проекта необходимо зайти в репозиторий promuc/demo-scada:



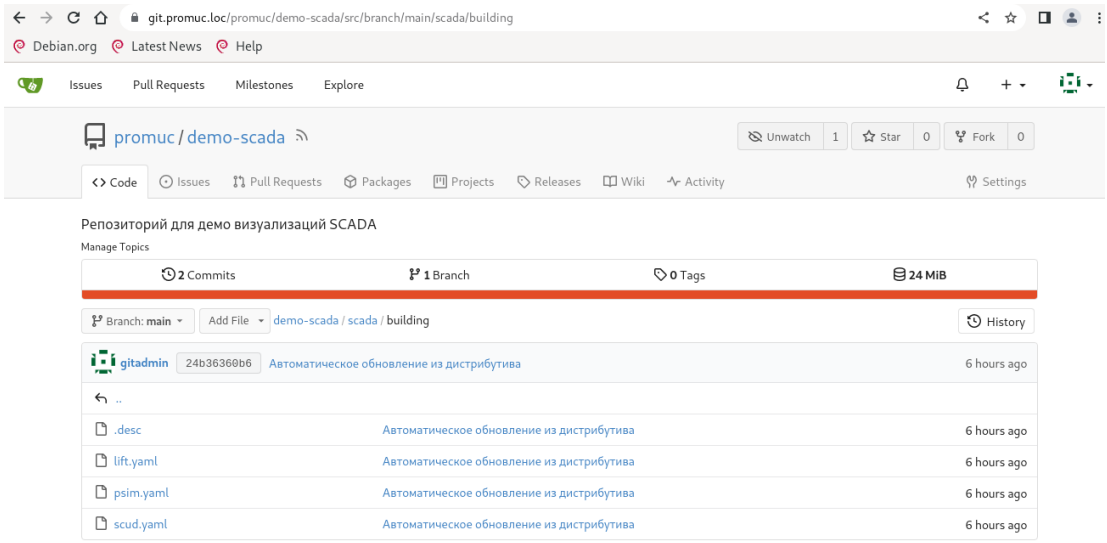
В составе репозитория все вложенные папки и файлы будут иметь ту же структуру, что и в файловой системе:



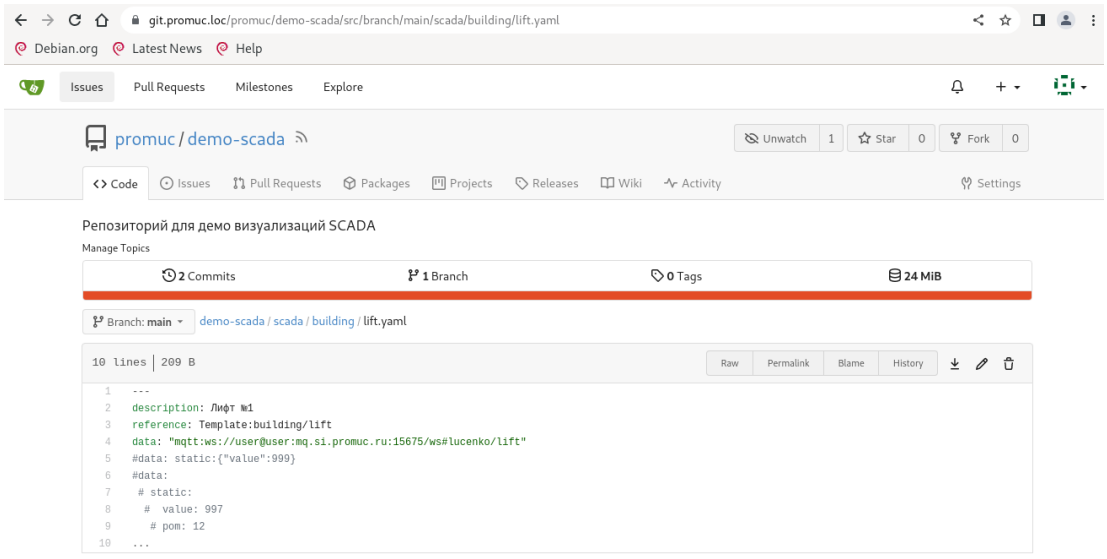
В папке scada:



В папке систем здания:



Открыв файл конфигурации, например, lift.yaml, можно его изменить, включив режим редактирования, и затем сохранить изменения:



Затем необходимо обновить страницу в web браузере, если была открыта редактируемая визуализация или перейти на неё чтобы проверить сделанные изменения.

1.7 Источники данных

В качестве поставщиков данных для отображения, могут использоваться различные источники данных:

- локальная база данных, например:

  `"sql:db://{username}@{pass_user}:{host}/{port}sql#{database_name}/{table_name}`

- mqtt брокер, например:

  `"mqtt:ws://user@user:mq.4iot.pro:15675/ws#data/lightning-2"`

- шлюз сбора данных (BACnet, Modbus, SunApi, Profinet и т.п.) см. модуль «PromUC Gateway»
- GET и POST запросы для различных Api

После создания новых экранов визуализаций можно приступить к их использованию.

1.8 Создание панелей (элементов) управления

Приводится пример создания панели управления для визуализации `scada/ventilation/ahu_p1.yaml`

1. Создание шаблона панели управления «**pult/ahu.yaml**»

   Перейти в папку `source/pult` и создать файл **ahu.yaml**

```yaml
---
description: Управление ПВУ
elements:

# Элемент №1
- name: Уставка температуры притока
  type: analog
  data: TEMP_SET
  min: 32
  max: 14
  step: 1

# Элемент №2
- name: Управление приточным вентилятором
  type: binary
  icon_on: Svg:icons/on
  icon_off: Svg:icons/off
  data: SF_CONTROL

# Элемент №3
- name: Управление вытяжным вентилятором
  type: binary
  icon_on: Svg:icons/on
  icon_off: Svg:icons/off
  data: RF_CONTROL

# Элемент №4
- name: Режим работы системы
  type: multistate
  data: MODE
  states:
     1: Включить
     2: Отключить
     3: Работа по расписанию

# Элемент №4
- name: Выбор сезона
  type: multistate
  data: SEASON
  states:
     1: Зима
     2: Лето
     3: Авто
...
```

2. Перейти в папку `scada/ventilation`. Открыть файл `"ahu_p1.yaml"`:

```
---
description: Вент-установка П-1
reference: Visio:ventilation/ahu_i
data: "mqtt:ws://user@user:mq.4iot.pro:15675/ws#data//ahu"
...
```

3. Внести следующие изменения:

```
---
description: Вент-установка П-1
reference: Visio:ventilation/ahu_i
data1: "mqtt:ws://user@user:mq.4iot.pro:15675/ws#data/ahu"
controls:
    AHU_CONTROL: Pult:ahu
        translate:
            TEMP_SET: data1.sf_t_setpoint
            SF_CONTROL: data1.start_sf
            RF_CONTROL: data1.start_rf
            MODE: data1.unit_mode
            SEASON: data1.season_mode
...
```

«controls:» - Набор панелей управления, используемых в визуализации

«AHU_CONTROL:» - Название панели управления внутри визуализации, используется в редакторе визуализаций

Pult:ahu - Ссылка на шаблон панели управления

«translate:» - В этом поле связывается глобальная переменная элемента панели управления (**«data»**) с переменной из источника данных.

TEMP_SET: - **data** Элемента №1 pult/ahu.yaml

data1.sf_t_setpoint - переменная из источника данных *«mqtt:ws://user@user:mq.4iot.pro:15675/ws#data/ahu»*

SF_CONTROL: - **data** Элемента №2 pult/ahu.yaml

data1.start_sf - переменная из источника данных *«mqtt:ws://user@user:mq.4iot.pro:15675/ws#data/ahu»*

RF_CONTROL: - **data** Элемента №3 pult/ahu.yaml

data1.start_rf - переменная из источника данных *«mqtt:ws://user@user:mq.4iot.pro:15675/ws#data/ahu»*

MODE: - **data** Элемента №4 pult/ahu.yaml

data1.unit_mode - переменная из источника данных *«mqtt:ws://user@user:mq.4iot.pro:15675/ws#data/ahu»*

SEASON: - **data** Элемента №5 pult/ahu.yaml

data1.season_mode - переменная из источника данных *«mqtt:ws://user@user:mq.4iot.pro:15675/ws#data/ahu»*

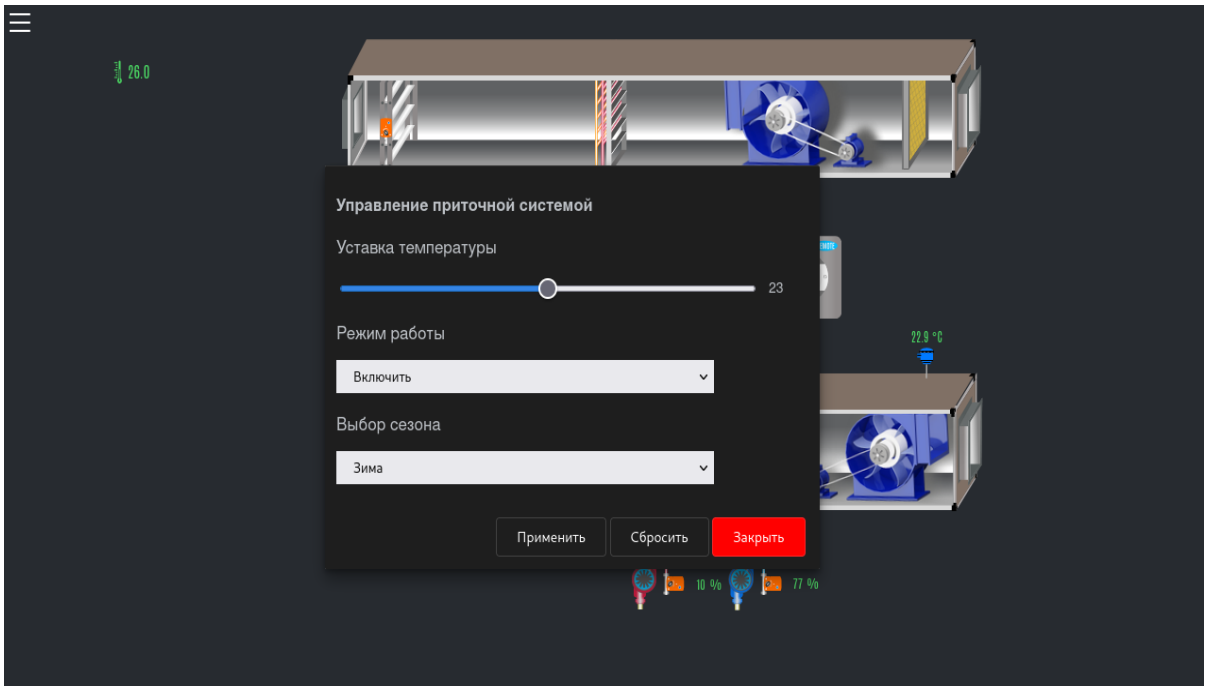
Панель управления создана.

4. Настройка отображения панели управления **«AHU_CONTROL»** внутри визуализации `scada/ventilation/ahu_p1.yaml`

Добавить элемент пульта управления (в примере используются элемент библиотеки `svg/elements/control/remote.svg`) в визуализацию `scada/ventilation/ahu_p1.yaml`

В поле **«reference»** вписать название, созданной в визуализации `scada/ventilation/ahu_p1`, панели управления **AHU_CONTROL**.

Панель управления полностью настроена и теперь отображается при клике на элемент, который с ней связан.



В панели имеются встроенные кнопки:

Записать - Отправить выбранные значения на запись

Отмена - Отмена действия

Закрыть - Выйти из панели управления

1.9 Управление объектами

1. Открыть визуализацию объекта, в который требуется отправить команду управления
2. Кликнуть по иконке панели управления, панель управления откроется
3. Задать требуемые параметры
4. Нажать кнопку «Записать»
5. При необходимости, повторить действия
6. Выйти из панели управления

После управления объектами можно перейти к любой другой из описаных операций.

1.10 Просмотр журнала событий

Демо-проект на данный момент не содержит настроенных шаблонов для журнала событий.

1.11 Реакция на аварийные сообщения системы

В случае возникновения аварийной ситуации, на экране отобразится всплывающее сообщение с описанием возникшей проблемы. Информация о возникшей аварии, с указанием времени возникновения, будет записана в журнал аварий. При клике на данном сообщении будет предложено квитировать аварию (информация о имени пользователя, квитирующего аварию, и время квитирования будет также занесено в журнал) и перейти на соответствующий графический экран для более подробного изучения ситуации. Используя полученную информацию, оператор должен оценить ситуацию и принять меры по устранению аварии: найти причины ее возникновения и устранить их, вернув системы в штатный режим эксплуатации, при необходимости задействовать оперативный персонал.

Глава 2

Архитектура визуализации

2.1 Используемые Web технологии

Данная система полностью построена на WEB технологии используя в своей основе файлы SVG и CSS.

SVG (Scalable Vector Graphics — масштабируемая векторная графика) — язык разметки масштабируемой векторной графики, входящий в подмножество расширяемого языка разметки XML, предназначен для описания двумерной векторной и смешанной векторно/растровой графики в формате XML. Поддерживает как неподвижную, так и анимированную интерактивную графику — или, в иных терминах, декларативную и скриптовую.

CSS (Cascading Style Sheets «каскадные таблицы стилей») — формальный язык описания внешнего вида документа (веб-страницы), написанного с использованием языка разметки (чаще всего HTML или XHTML). Также может применяться к любым XML-документам, например, к SVG.

2.2 Библиотека элементов в формате SVG

Для создания визуализаций необходимо наполнить библиотеку подготовленными (как подготовить файлы смотри в разделе «Подготовка SVG») элементами в формате SVG. Сами элементы могут быть созданы в любой программе векторной графики, но мы рекомендуем для создания и редактирования элементов SVG использовать инструмент Penpot (входит в платформу 4iot) или облачное приложение Figma.

2.3 Подготовка SVG элементов

Для редактирования файла SVG его можно открыть любым текстовым редактором.

Для целей визуализации данных, предусмотрены следующие типы анимаций:

1. Изменение цвета в зависимости от состояния сигнала, связанного с элементом («normal»- зеленый цвет, «fault»- серый цвет, «overridden» – желтый цвет, «in-alarm» - красный цвет) – применимо ко всем типам сигналов;
2. Показать-скрыть элемент (либо его часть) в зависимости от значения, связанного сигнала(«True»- показать, «False»-скрыть или наоборот) – применимо для бинарных сигналов;
3. Вращение элемента (либо его части) в зависимости от значения, связанного сигнала(«True»- вращается, «False»- стоит) – применимо для бинарных сигналов;
4. Отображение значений (например значение температуры) в зависимости от значения, связанного сигнала – применимо для аналоговых сигналов;
5. Изменение размера элемента по осям X или Y в зависимости от значения, связанного сигнала – применимо для аналоговых сигналов;
6. Элемент с мультисостояниями, которые отображаются в зависимости от значения, связанного сигнала – применимо для multistate сигналов;
7. Мигание элемента (либо его части) например «авария» - применимо для бинарных сигналов;

В одном svg-элементе может быть использовано несколько различных анимаций, каждая из которых связана со своим сигналом. Все анимации построены на изменении свойств элементов с использованием CSS классов и стилей.

2.4 Примеры создания простых элементов

2.4.1 Создание бинарного элемента

Создание бинарного элемента, который в зависимости от состояния связанного сигнала показывает надпись ON или OFF:

Исходный код

```
<svg width="29" height="35" viewBox="0 0 29 35" fill="none" xmlns="http://www.w3.
↪org/2000/svg">
<path id="OFF" d="M1.48757 11.9766C1.48757 9.63281 1.90944 8.02344 2.75319 7.
↪14844C3.41725 6.46094 4.3821 6.11719 5.64772 6.11719C7.49147 6.11719 8.67116 6.
↪875 9.18678 8.39062C9.49147 9.28906 9.64382 10.4844 9.64382 11.9766V21.0703C9.
```

(continues on next page)

(продолжение с предыдущей страницы)

```
↪64382 23.2188 9.34303 24.7852 8.74147 25.7695C8.13991 26.7461 7.08522 27.2344 5.
↪57741 27.2344C4.07741 27.2344 3.01882 26.7422 2.40163 25.7578C1.79225 24.7734 1.
↪48757 23.2109 1.48757 21.0703V11.9766ZM7.21803 11.8359C7.21803 10.5156 7.11257 9.
↪58203 6.90163 9.03516C6.69069 8.48047 6.26491 8.20312 5.62428 8.20312C4.98366 8.
↪20312 4.53835 8.48438 4.28835 9.04688C4.03835 9.60938 3.91335 10.5352 3.91335 11.
↪8242V21.0938C3.91335 22.6641 4.02272 23.7344 4.24147 24.3047C4.46022 24.8672 4.
↪90163 25.1484 5.56569 25.1484C6.22975 25.1484 6.67116 24.8633 6.88991 24.293C7.
↪10866 23.7148 7.21803 22.6484 7.21803 21.0938V11.8359ZM12.7727 6.375H18.9837V8.
↪40234H15.1868V14.918H18.7493V16.9688H15.1868V27H12.7727V6.375ZM21.5149 6.375H27.
↪7258V8.40234H23.929V14.918H27.4915V16.9688H23.929V27H21.5149V6.375Z" fill="green
↪"/>
<path id="ON" d="M4.28835 11.9766C4.28835 9.63281 4.71022 8.02344 5.55397 7.
↪14844C6.21803 6.46094 7.18288 6.11719 8.4485 6.11719C10.2923 6.11719 11.4719 6.
↪875 11.9876 8.39062C12.2923 9.28906 12.4446 10.4844 12.4446 11.9766V21.0703C12.
↪4446 23.2188 12.1438 24.7852 11.5423 25.7695C10.9407 26.7461 9.886 27.2344 8.
↪37819 27.2344C6.87819 27.2344 5.8196 26.7422 5.20241 25.7578C4.59303 24.7734 4.
↪28835 23.2109 4.28835 21.0703V11.9766ZM10.0188 11.8359C10.0188 10.5156 9.91335 9.
↪58203 9.70241 9.03516C9.49147 8.48047 9.06569 8.20312 8.42507 8.20312C7.78444 8.
↪20312 7.33913 8.48438 7.08913 9.04688C6.83913 9.60938 6.71413 10.5352 6.71413 11.
↪8242V21.0938C6.71413 22.6641 6.8235 23.7344 7.04225 24.3047C7.261 24.8672 7.
↪70241 25.1484 8.36647 25.1484C9.03053 25.1484 9.47194 24.8633 9.69069 24.293C9.
↪90944 23.7148 10.0188 22.6484 10.0188 21.0938V11.8359ZM21.9485 6.375H24.
↪093V27H22.3118L17.6829 12.8438L17.8235 18.0469V27H15.5852V6.375H17.4133L22.1946␣
↪20.4141L21.9485 14.4492V6.375Z" fill="green"/>
</svg>
```

Адаптированый код

```
<svg width="29" height="35" viewBox="0 0 29 35" fill="none" xmlns="http://www.w3.
↪org/2000/svg">
<g class="fault inactive" reference="DATA">
<g  class="animation-hide">
<!-Надпись OFF -->
<path id="OFF" d="M1.48757 11.9766C1.48757 9.63281 1.90944 8.02344 2.75319 7.
↪14844C3.41725 6.46094 4.3821 6.11719 5.64772 6.11719C7.49147 6.11719 8.67116 6.
↪875 9.18678 8.39062C9.49147 9.28906 9.64382 10.4844 9.64382 11.9766V21.0703C9.
↪64382 23.2188 9.34303 24.7852 8.74147 25.7695C8.13991 26.7461 7.08522 27.2344 5.
↪57741 27.2344C4.07741 27.2344 3.01882 26.7422 2.40163 25.7578C1.79225 24.7734 1.
↪48757 23.2109 1.48757 21.0703V11.9766ZM7.21803 11.8359C7.21803 10.5156 7.11257 9.
↪58203 6.90163 9.03516C6.69069 8.48047 6.26491 8.20312 5.62428 8.20312C4.98366 8.
↪20312 4.53835 8.48438 4.28835 9.04688C4.03835 9.60938 3.91335 10.5352 3.91335 11.
↪8242V21.0938C3.91335 22.6641 4.02272 23.7344 4.24147 24.3047C4.46022 24.8672 4.
↪90163 25.1484 5.56569 25.1484C6.22975 25.1484 6.67116 24.8633 6.88991 24.293C7.
↪10866 23.7148 7.21803 22.6484 7.21803 21.0938V11.8359ZM12.7727 6.375H18.9837V8.
↪40234H15.1868V14.918H18.7493V16.9688H15.1868V27H12.7727V6.375ZM21.5149 6.375H27.
↪7258V8.40234H23.929V14.918H27.4915V16.9688H23.929V27H21.5149V6.375Z"/>
```

(continues on next page)

(продолжение с предыдущей страницы)

```
</g>
<g  class="animation-display">
<!-Надпись ON -->
<path id="ON" d="M4.28835 11.9766C4.28835 9.63281 4.71022 8.02344 5.55397 7.
↪14844C6.21803 6.46094 7.18288 6.11719 8.4485 6.11719C10.2923 6.11719 11.4719 6.
↪875 11.9876 8.39062C12.2923 9.28906 12.4446 10.4844 12.4446 11.9766V21.0703C12.
↪4446 23.2188 12.1438 24.7852 11.5423 25.7695C10.9407 26.7461 9.886 27.2344 8.
↪37819 27.2344C6.87819 27.2344 5.8196 26.7422 5.20241 25.7578C4.59303 24.7734 4.
↪28835 23.2109 4.28835 21.0703V11.9766ZM10.0188 11.8359C10.0188 10.5156 9.91335 9.
↪58203 9.70241 9.03516C9.49147 8.48047 9.06569 8.20312 8.42507 8.20312C7.78444 8.
↪20312 7.33913 8.48438 7.08913 9.04688C6.83913 9.60938 6.71413 10.5352 6.71413 11.
↪8242V21.0938C6.71413 22.6641 6.8235 23.7344 7.04225 24.3047C7.261 24.8672 7.
↪70241 25.1484 8.36647 25.1484C9.03053 25.1484 9.47194 24.8633 9.69069 24.293C9.
↪90944 23.7148 10.0188 22.6484 10.0188 21.0938V11.8359ZM21.9485 6.375H24.
↪093V27H22.3118L17.6829 12.8438L17.8235 18.0469V27H15.5852V6.375H17.4133L22.1946␣
↪20.4141L21.9485 14.4492V6.375Z"/>
</g>
</g>
</svg>
```

Блок CSS кода

```
/* Стили флагов состояний */

   /* зеленый цвет */
   .normal {
   fill: #4CD964;
   color:#4CD964;
   }
   /* серый цвет */
   .fault {
   fill: #E2E7E9;
   color: #E2E7E9;
   }
   /* красный цвет */
   .in-alarm {
   fill: #FF0000;
   color: #FF0000;
   }
   /* желтый цвет */
   .overridden,
   .out-of-service {
   fill: #FFCC00;
   color: #FFCC00;
   }
/* Скрыть-Показать элемент */
```

(continues on next page)

(продолжение с предыдущей страницы)

```
/* Скрыть */
.active .animation-hide,
.inactive .animation-display,
{
display: none;
}

/* Отобразить */
.inactive .animation-hide,
.active .animation-display,
{
display: block;
}
```

- Описание изменений, сделанных в исходном коде svg-файла:

  1. Из исходного кода удаляем свойство `fill="green"`, делая элемент бесцветным. Цвет будет подставляться в зависимости от флага состояния (`status-flag`) связанного сигнала.
  2. Заключаем оба состояния (OFF и ON) в группу с помощью тэга <g> и задаем этой группе `class="fault inactive"` – это класс для элемента по-умолчанию. `"fault"` = флаг состояния и `inactive` = значение связанного сигнала (система изменит эти два класса в зависимости от реального состояния связанного сигнала).
  3. С помощью тэга <g> задаем классы (`class`) каждой из двух групп (<g>): `class` для OFF ‘‘`animation-hide`’’ – в случае если значение, связанного сигнала = `"true"`, эта часть svg-файла будет скрыта (невидима), а если значение, связанного сигнала= inactive, эта часть svg-файла будет отображена (видна); class для ON ‘‘`animation-display`’’ – в случае если значение, связанного сигнала = `"false"`, эта часть svg-файла будет скрыта(невидима), а если `presentValue` связанного сигнала= `active`, эта часть svg-файла будет отображена(видна);
  4. `reference="DATA"` – это ссылка для связывания с сигналом. Сервер распознает тэг `reference` и изменяет значения в `class="fault inactive"` на соответствующие связанному сигналу состояния (`fault` будет заменен на реальный `status-flag`, `inactive` на реальное значение).

2.4.2 Создание текстового элемента

Создание текстового элемента, который отображает текущее аналоговое (числовое) значение связанного элемента:

Исходный код

```
<svg width="80" height="40" viewBox="0 0 80 24" fill="none" xmlns="http://www.w3.
↪org/2000/svg">
<text x="50%" y="12" text-anchor="middle" fill="green">...</text>
</svg>
```

Адаптированый код

```
 <svg width="80" height="40" viewBox="0 0 80 24" fill="none" xmlns="http://www.w3.
↪org/2000/svg">
<style>
      #SENSOR
      {
        font-family: "Antonio-Regular";
    font-style: normal;
    font-weight: normal;
    font-size: 18px;
    text-align: center;
      }
  </style>
 <g id="SEN" class="fault" format="%.1f" reference="DATA">
 <text x="50%" y="12" text-anchor="middle">...</text>
 </g>
 </svg>
```

Блок CSS кода

```
/* Стили флагов состояний */

   /* зеленый цвет */
   .normal {
   fill: #4CD964;
   color:#4CD964;
   }
   /* серый цвет */
   .fault {
   fill: #E2E7E9;
   color: #E2E7E9;
   }
   /* красный цвет */
   .in-alarm {
   fill: #FF0000;
   color: #FF0000;
   }
   /* желтый цвет */
   .overridden,
   .out-of-service {
   fill: #FFCC00;
```

(continues on next page)

(продолжение с предыдущей страницы)

```
color: #FFCC00;
}
```

- Описание изменений, сделанных в исходном коде svg-файла:

1. Из исходного кода удаляем свойство `fill="green"`, делая текст бесцветным. Цвет будет подставляться в зависимости от флага состояния(`status-flag`) связанного сигнала.
2. Заключаем `<text>` в группу с помощью тэга `<g>` и задаем этой группе `id="SENSOR"`, `class="fault"`, `format="%.1f"`, `reference="DATA"`:
   - `id="SENSOR"` – требуется для задания стиля текста, который будет отображать значение;
   - `class="fault"` - класс для элемента по-умолчанию (система изменит этот класс в зависимости от реального состояния, связанного сигнала);
   - `format="%.1f"` - задается формат числа для отображения (в данном случае это чмсло с одним знаком после запятой);
   - `reference="DATA"` – это ссылка для связывания с сигналом. Система распознает тэг `reference` и изменяет содержание тэга `<text>` на значение связанного сигнала.
3. Спомощью тэга `<style>` зададим стиль для отображаемого текста.

2.4.3 Создание элемента с мультисостояниями

Создание элемента с мультисостояниями, который в зависимости от значения связанного сигнала, отображает соответствующее значению состояние:

Исходный код

```
<svg width="50" height="50" viewBox="0 0 50 50" fill="none" xmlns="http://www.w3.
↪org/2000/svg">
<path d="M25 47C37.1503 47 47 37.1503 47 25C47 12.8497 37.1503 3 25 3C12.8497 3 3␣
↪12.8497 3 25C3 37.1503 12.8497 47 25 47Z" fill="#2F3234"/>
<path d="M37.1531 29.4909C38.6929 28.723 40.2326 27.9232 41.7024 27.1554C41.9123␣
```

(continues on next page)

(продолжение с предыдущей страницы)

```
↪27.0594 42.1573 27.1234 42.2973 27.3154L42.8922 28.3071C43.0322 28.4671 42.9622␣
↪28.723 42.7522 28.819L39.3228 30.6426L42.2973 32.2102C42.5072 32.3062 42.5772 32.
↪5622 42.4373 32.7541L41.8074 33.7779C41.7024 33.9378 41.4224 34.0018 41.2125 33.
↪9059L38.2379 32.3382V36.0174C38.2379 36.2413 38.028 36.4013 37.818 36.4013H36.
↪5582C36.3132 36.4013 36.1383 36.2413 36.1383 36.0174C36.1383 34.4497 36.1033 32.
↪8181 36.1033 31.1865L32.3939 29.2669C32.3939 36.7852 33.7237 36.0814 26.6198 32.
↪3382V36.2413C28.1246 37.0411 29.6293 37.9049 31.0641 38.6728C31.2741 38.7687 31.
↪3441 38.9927 31.2391 39.1846L30.6092 40.1764C30.5042 40.3684 30.2242 40.4324 30.
↪0143 40.3364L26.6198 38.5128V41.6161C26.6198 41.84 26.4099 42 26.1999 42H24.
↪9051C24.6601 42 24.4852 41.84 24.4852 41.6161V38.4808L20.9507 40.3364C20.7408 40.
↪4324 20.4958 40.3684 20.3558 40.2084L19.7609 39.2166C19.621 39.0247 19.6909 38.
↪7687 19.9009 38.6728C21.3707 37.9049 22.9454 37.0091 24.4852 36.2093V32.3062C17.
↪2763 36.1133 18.5711 36.8812 18.5711 29.2989L14.8967 31.2505C14.8967 32.8181 14.
↪8617 34.4497 14.8617 35.9854C14.8617 36.2093 14.6868 36.4013 14.4418 36.4013H13.
↪217C12.972 36.4013 12.7621 36.2093 12.7621 35.9854C12.7621 34.8016 12.7621 33.
↪5859 12.7621 32.3702L9.85754 33.9059C9.64757 34.0018 9.40261 33.9378 9.26263 33.
↪7779L8.63273 32.7541C8.52775 32.5622 8.59774 32.3062 8.77271 32.2102L11.7122 30.
↪6746L8.2128 28.819C8.00284 28.723 7.93285 28.4671 8.07282 28.2751L8.70272 27.
↪3154C8.80771 27.1234 9.05267 27.0594 9.26263 27.1554C10.7674 27.9552 12.3421 28.
↪723 13.8819 29.5229L17.5563 27.5713C10.3824 23.7962 10.3824 25.2038 17.5563 21.
↪4287L13.8469 19.4452C12.3071 20.245 10.8024 21.0448 9.29763 21.8446C9.08766 21.
↪9406 8.8427 21.8766 8.70272 21.6846L8.10782 20.6929C7.96784 20.5329 8.03783 20.
↪277 8.2478 20.181C9.40261 19.5731 10.5574 18.9333 11.7122 18.3254L8.77271 16.
↪7898C8.59774 16.6938 8.52775 16.4378 8.63273 16.2459L9.26263 15.2221C9.40261 15.
↪0622 9.64757 14.9982 9.85754 15.0942L12.7621 16.6298V12.9826C12.7621 12.7587 12.
↪972 12.5987 13.182 12.5987H14.4418C14.6868 12.5987 14.8617 12.7587 14.8617 12.
↪9826C14.8617 14.5503 14.8967 16.1499 14.8967 17.7495L18.6061 19.7011C18.6061 12.
↪1508 17.2763 12.8867 24.4852 16.6618V12.7587C22.9454 11.9589 21.4407 11.1271 19.
↪9359 10.3272C19.7259 10.2313 19.6559 10.0073 19.7609 9.81536L20.3908 8.82358C20.
↪4958 8.63163 20.7758 8.56764 20.9857 8.66362C22.1056 9.27148 23.2954 9.87934 24.
↪4852 10.5192V7.38391C24.4852 7.15996 24.6601 7 24.9051 7H26.1999C26.4099 7 26.
↪6198 7.15996 26.6198 7.38391V10.4552C27.7746 9.84735 28.8945 9.27148 30.0493 8.
↪66362C30.2592 8.56764 30.5042 8.63163 30.6442 8.79159L31.2391 9.78336C31.379 9.
↪97532 31.3091 10.2313 31.0991 10.3272C29.6293 11.0951 28.1246 11.9269 26.6198 12.
↪7267V16.5978C33.7937 12.8227 32.4289 12.1508 32.4289 19.7331L36.0683 17.8135C36.
↪0683 16.1819 36.1383 14.5503 36.1383 13.0146C36.1383 12.7907 36.3132 12.5987 36.
↪5582 12.5987H37.783C38.028 12.5987 38.2379 12.7907 38.2379 13.0146C38.2379 14.
↪2303 38.2379 15.4461 38.2379 16.6618L41.2125 15.0942C41.4224 14.9982 41.7024 15.
↪0622 41.8074 15.2221L42.4373 16.2459C42.5772 16.4378 42.5072 16.6938 42.2973 16.
↪7898L39.3228 18.3574C40.4776 18.9653 41.6324 19.5731 42.7872 20.181C42.9972 20.
↪277 43.0672 20.5329 42.9272 20.7249L42.2973 21.6846C42.1923 21.8766 41.9473 21.
↪9406 41.7374 21.8446C40.2326 21.0448 38.7279 20.277 37.1881 19.4771L33.5137 21.
↪4287C40.6176 25.1718 40.6176 23.7642 33.4437 27.5393L37.1531 29.4909ZM19.6909 26.
↪4516L23.4003 24.5L19.6909 22.5484L15.9815 24.5L19.6909 26.4516ZM26.6198 22.
↪8044L30.2942 20.8528V16.9177L26.6198 18.8693V22.8044ZM31.379 22.5484L27.6697 24.
```

(continues on next page)

(продолжение с предыдущей страницы)

```
↪5L31.3091 26.4196L35.0185 24.468L31.379 22.5484ZM24.4852 26.1956L20.7058 28.
↪1792V32.0503L24.4852 30.0667V26.1956ZM20.7408 20.8208L24.4852 22.8044V18.9333L20.
↪7408 16.9497V20.8208ZM30.2592 28.1152L26.6198 26.1956V30.0667L30.2592 32.0183V28.
↪1152Z" fill="#4CD964"/>
<path d="M12.6865 17.5312H14.875L17.4326 33H15.7188L15.2793 30.0469H12.4756L12.
↪0713 33H10.3398L12.6865 17.5312ZM12.6777 28.5703H15.0508L13.8291 20.3965H13.
↪8027L12.6777 28.5703ZM23.3389 17.5312H25.1494V29.0273C25.1494 30.4746 24.9209 31.
↪5293 24.4639 32.1914C24.0127 32.8535 23.2275 33.1846 22.1084 33.1846C20.9893 33.
↪1846 20.1982 32.8535 19.7354 32.1914C19.2725 31.5293 19.041 30.4746 19.041 29.
↪0273V17.5312H20.8604V28.9834C20.8604 30.0146 20.9014 30.6328 20.9834 30.8379C21.
↪0713 31.0371 21.1621 31.1953 21.2559 31.3125C21.4199 31.5352 21.7041 31.6465 22.
↪1084 31.6465C22.5127 31.6465 22.8115 31.5234 23.0049 31.2773C23.1982 31.0312 23.
↪3008 30.665 23.3125 30.1787C23.3301 29.6924 23.3389 29.2969 23.3389 28.9922V17.
↪5312ZM26.582 17.5312H31.6885V19.043H30.0977V33H28.2871V19.043H26.582V17.5312ZM33.
↪0332 21.7324C33.0332 19.9746 33.3496 18.7676 33.9824 18.1113C34.4805 17.5957 35.
↪2041 17.3379 36.1533 17.3379C37.5361 17.3379 38.4209 17.9062 38.8076 19.043C39.
↪0361 19.7168 39.1504 20.6133 39.1504 21.7324V28.5527C39.1504 30.1641 38.9248 31.
↪3389 38.4736 32.0771C38.0225 32.8096 37.2314 33.1758 36.1006 33.1758C34.9756 33.
↪1758 34.1816 32.8066 33.7188 32.0684C33.2617 31.3301 33.0332 30.1582 33.0332 28.
↪5527V21.7324ZM37.3311 21.627C37.3311 20.6367 37.252 19.9365 37.0938 19.5264C36.
↪9355 19.1104 36.6162 18.9023 36.1357 18.9023C35.6553 18.9023 35.3213 19.1133 35.
↪1338 19.5352C34.9463 19.957 34.8525 20.6514 34.8525 21.6182V28.5703C34.8525 29.
↪748 34.9346 30.5508 35.0986 30.9785C35.2627 31.4004 35.5938 31.6113 36.0918 31.
↪6113C36.5898 31.6113 36.9209 31.3975 37.085 30.9697C37.249 30.5361 37.3311 29.
↪7363 37.3311 28.5703V21.627Z" fill="#4CD964"/>
<path d="M24.601 32.8739C28.5959 32.8739 31.8485 29.618 31.8485 25.6191C31.8485 21.
↪6201 28.5959 18.3643 24.601 18.3643C20.606 18.3643 17.3535 21.6201 17.3535 25.
↪6191C17.3535 29.618 20.606 32.8739 24.601 32.8739Z" fill="#4CD964"/><path d="M33.
↪0556 26.4024L38.9596 25.5177L33.2323 24.6684L42 20.9525L32.3131 22.0142L37.0151␣
↪18.2983L31.6414 20.4216L37.3687 12.8129L29.5202 18.5814L31.7475 12.9899L28.1414␣
↪17.5197L29.3081 8.07078L25.3838 16.9889L24.5 11.0789L23.6515 16.8119L19.9394␣
↪8L21 17.6967L17.2879 12.9899L19.4091 18.3691L11.8081 12.636L17.5707 20.4924L11.
↪9848 18.2629L16.5101 21.8726L7.07071 20.7048L15.9798 24.633L10.0758 25.5177L15.
↪803 26.367L7 30.0829L16.6869 29.0212L11.9848 32.7371L17.3586 30.6138L11.6313 38.
↪2224L19.4798 32.454L17.2525 38.0455L20.8586 33.5157L19.7273 42.9646L23.6515 34.
↪0465L24.5354 39.9565L25.3838 34.2235L29.096 43L28.0354 33.3033L31.7475 38.
↪0101L29.6263 32.6309L37.2273 38.364L31.4646 30.5076L37.0505 32.7371L32.5253 29.
↪1274L41.9646 30.2952L33.0556 26.4024ZM24.5 33.728C19.9747 33.728 16.298 30.0475␣
↪16.298 25.5177C16.298 20.9879 19.9747 17.3074 24.5 17.3074C29.0253 17.3074 32.
↪702 20.9879 32.702 25.5177C32.702 30.0475 29.0253 33.728 24.5 33.728Z" fill="
↪#4CD964"/>
</svg>
```

Адаптированый код

```
<svg width="50" height="50" viewBox="0 0 50 50" fill="none" xmlns="http://www.w3.
↪org/2000/svg">
<g class="fault multi-state-0 " reference="MODE">
<path d="M25 47C37.1503 47 47 37.1503 47 25C47 12.8497 37.1503 3 25 3C12.8497 3 3␣
↪12.8497 3 25C3 37.1503 12.8497 47 25 47Z" fill="#2F3234"/>
<!--cool-->
<g class="array-0">
<path d="M37.1531 29.4909C38.6929 28.723 40.2326 27.9232 41.7024 27.1554C41.9123␣
↪27.0594 42.1573 27.1234 42.2973 27.3154L42.8922 28.3071C43.0322 28.4671 42.9622␣
↪28.723 42.7522 28.819L39.3228 30.6426L42.2973 32.2102C42.5072 32.3062 42.5772 32.
↪5622 42.4373 32.7541L41.8074 33.7779C41.7024 33.9378 41.4224 34.0018 41.2125 33.
↪9059L38.2379 32.3382V36.0174C38.2379 36.2413 38.028 36.4013 37.818 36.4013H36.
↪5582C36.3132 36.4013 36.1383 36.2413 36.1383 36.0174C36.1383 34.4497 36.1033 32.
↪8181 36.1033 31.1865L32.3939 29.2669C32.3939 36.7852 33.7237 36.0814 26.6198 32.
↪3382V36.2413C28.1246 37.0411 29.6293 37.9049 31.0641 38.6728C31.2741 38.7687 31.
↪3441 38.9927 31.2391 39.1846L30.6092 40.1764C30.5042 40.3684 30.2242 40.4324 30.
↪0143 40.3364L26.6198 38.5128V41.6161C26.6198 41.84 26.4099 42 26.1999 42H24.
↪9051C24.6601 42 24.4852 41.84 24.4852 41.6161V38.4808L20.9507 40.3364C20.7408 40.
↪4324 20.4958 40.3684 20.3558 40.2084L19.7609 39.2166C19.621 39.0247 19.6909 38.
↪7687 19.9009 38.6728C21.3707 37.9049 22.9454 37.0091 24.4852 36.2093V32.3062C17.
↪2763 36.1133 18.5711 36.8812 18.5711 29.2989L14.8967 31.2505C14.8967 32.8181 14.
↪8617 34.4497 14.8617 35.9854C14.8617 36.2093 14.6868 36.4013 14.4418 36.4013H13.
↪217C12.972 36.4013 12.7621 36.2093 12.7621 35.9854C12.7621 34.8016 12.7621 33.
↪5859 12.7621 32.3702L9.85754 33.9059C9.64757 34.0018 9.40261 33.9378 9.26263 33.
↪7779L8.63273 32.7541C8.52775 32.5622 8.59774 32.3062 8.77271 32.2102L11.7122 30.
↪6746L8.2128 28.819C8.00284 28.723 7.93285 28.4671 8.07282 28.2751L8.70272 27.
↪3154C8.80771 27.1234 9.05267 27.0594 9.26263 27.1554C10.7674 27.9552 12.3421 28.
↪723 13.8819 29.5229L17.5563 27.5713C10.3824 23.7962 10.3824 25.2038 17.5563 21.
↪4287L13.8469 19.4452C12.3071 20.245 10.8024 21.0448 9.29763 21.8446C9.08766 21.
↪9406 8.8427 21.8766 8.70272 21.6846L8.10782 20.6929C7.96784 20.5329 8.03783 20.
↪277 8.2478 20.181C9.40261 19.5731 10.5574 18.9333 11.7122 18.3254L8.77271 16.
↪7898C8.59774 16.6938 8.52775 16.4378 8.63273 16.2459L9.26263 15.2221C9.40261 15.
↪0622 9.64757 14.9982 9.85754 15.0942L12.7621 16.6298V12.9826C12.7621 12.7587 12.
↪972 12.5987 13.182 12.5987H14.4418C14.6868 12.5987 14.8617 12.7587 14.8617 12.
↪9826C14.8617 14.5503 14.8967 16.1499 14.8967 17.7495L18.6061 19.7011C18.6061 12.
↪1508 17.2763 12.8867 24.4852 16.6618V12.7587C22.9454 11.9589 21.4407 11.1271 19.
↪9359 10.3272C19.7259 10.2313 19.6559 10.0073 19.7609 9.81536L20.3908 8.82358C20.
↪4958 8.63163 20.7758 8.56764 20.9857 8.66362C22.1056 9.27148 23.2954 9.87934 24.
↪4852 10.5192V7.38391C24.4852 7.15996 24.6601 7 24.9051 7H26.1999C26.4099 7 26.
↪6198 7.15996 26.6198 7.38391V10.4552C27.7746 9.84735 28.8945 9.27148 30.0493 8.
↪66362C30.2592 8.56764 30.5042 8.63163 30.6442 8.79159L31.2391 9.78336C31.379 9.
↪97532 31.3091 10.2313 31.0991 10.3272C29.6293 11.0951 28.1246 11.9269 26.6198 12.
↪7267V16.5978C33.7937 12.8227 32.4289 12.1508 32.4289 19.7331L36.0683 17.8135C36.
↪0683 16.1819 36.1383 14.5503 36.1383 13.0146C36.1383 12.7907 36.3132 12.5987 36.
↪5582 12.5987H37.783C38.028 12.5987 38.2379 12.7907 38.2379 13.0146C38.2379 14.
↪2303 38.2379 15.4461 38.2379 16.6618L41.2125 15.0942C41.4224 14.9982 41.7024 15.
```

(continues on next page)

(продолжение с предыдущей страницы)

```
↪0622 41.8074 15.2221L42.4373 16.2459C42.5772 16.4378 42.5072 16.6938 42.2973 16.
↪7898L39.3228 18.3574C40.4776 18.9653 41.6324 19.5731 42.7872 20.181C42.9972 20.
↪277 43.0672 20.5329 42.9272 20.7249L42.2973 21.6846C42.1923 21.8766 41.9473 21.
↪9406 41.7374 21.8446C40.2326 21.0448 38.7279 20.277 37.1881 19.4771L33.5137 21.
↪4287C40.6176 25.1718 40.6176 23.7642 33.4437 27.5393L37.1531 29.4909ZM19.6909 26.
↪4516L23.4003 24.5L19.6909 22.5484L15.9815 24.5L19.6909 26.4516ZM26.6198 22.
↪8044L30.2942 20.8528V16.9177L26.6198 18.8693V22.8044ZM31.379 22.5484L27.6697 24.
↪5L31.3091 26.4196L35.0185 24.468L31.379 22.5484ZM24.4852 26.1956L20.7058 28.
↪1792V32.0503L24.4852 30.0667V26.1956ZM20.7408 20.8208L24.4852 22.8044V18.9333L20.
↪7408 16.9497V20.8208ZM30.2592 28.1152L26.6198 26.1956V30.0667L30.2592 32.0183V28.
↪1152Z" fill="#4CD964"/>
</g>
<!--auto-->
<g class="array-2">
<path d="M12.6865 17.5312H14.875L17.4326 33H15.7188L15.2793 30.0469H12.4756L12.
↪0713 33H10.3398L12.6865 17.5312ZM12.6777 28.5703H15.0508L13.8291 20.3965H13.
↪8027L12.6777 28.5703ZM23.3389 17.5312H25.1494V29.0273C25.1494 30.4746 24.9209 31.
↪5293 24.4639 32.1914C24.0127 32.8535 23.2275 33.1846 22.1084 33.1846C20.9893 33.
↪1846 20.1982 32.8535 19.7354 32.1914C19.2725 31.5293 19.041 30.4746 19.041 29.
↪0273V17.5312H20.8604V28.9834C20.8604 30.0146 20.9014 30.6328 20.9834 30.8379C21.
↪0713 31.0371 21.1621 31.1953 21.2559 31.3125C21.4199 31.5352 21.7041 31.6465 22.
↪1084 31.6465C22.5127 31.6465 22.8115 31.5234 23.0049 31.2773C23.1982 31.0312 23.
↪3008 30.665 23.3125 30.1787C23.3301 29.6924 23.3389 29.2969 23.3389 28.9922V17.
↪5312ZM26.582 17.5312H31.6885V19.043H30.0977V33H28.2871V19.043H26.582V17.5312ZM33.
↪0332 21.7324C33.0332 19.9746 33.3496 18.7676 33.9824 18.1113C34.4805 17.5957 35.
↪2041 17.3379 36.1533 17.3379C37.5361 17.3379 38.4209 17.9062 38.8076 19.043C39.
↪0361 19.7168 39.1504 20.6133 39.1504 21.7324V28.5527C39.1504 30.1641 38.9248 31.
↪3389 38.4736 32.0771C38.0225 32.8096 37.2314 33.1758 36.1006 33.1758C34.9756 33.
↪1758 34.1816 32.8066 33.7188 32.0684C33.2617 31.3301 33.0332 30.1582 33.0332 28.
↪5527V21.7324ZM37.3311 21.627C37.3311 20.6367 37.252 19.9365 37.0938 19.5264C36.
↪9355 19.1104 36.6162 18.9023 36.1357 18.9023C35.6553 18.9023 35.3213 19.1133 35.
↪1338 19.5352C34.9463 19.957 34.8525 20.6514 34.8525 21.6182V28.5703C34.8525 29.
↪748 34.9346 30.5508 35.0986 30.9785C35.2627 31.4004 35.5938 31.6113 36.0918 31.
↪6113C36.5898 31.6113 36.9209 31.3975 37.085 30.9697C37.249 30.5361 37.3311 29.
↪7363 37.3311 28.5703V21.627Z" fill="#4CD964"/>
</g>
<!--heat-->
<g class="array-1">
<path d="M24.601 32.8739C28.5959 32.8739 31.8485 29.618 31.8485 25.6191C31.8485 21.
↪6201 28.5959 18.3643 24.601 18.3643C20.606 18.3643 17.3535 21.6201 17.3535 25.
↪6191C17.3535 29.618 20.606 32.8739 24.601 32.8739Z" fill="#4CD964"/>
<path d="M33.0556 26.4024L38.9596 25.5177L33.2323 24.6684L42 20.9525L32.3131 22.
↪0142L37.0151 18.2983L31.6414 20.4216L37.3687 12.8129L29.5202 18.5814L31.7475 12.
↪9899L28.1414 17.5197L29.3081 8.07078L25.3838 16.9889L24.5 11.0789L23.6515 16.
↪8119L19.9394 8L21 17.6967L17.2879 12.9899L19.4091 18.3691L11.8081 12.636L17.5707
```

(continues on next page)

(продолжение с предыдущей страницы)

```
↪20.4924L11.9848 18.2629L16.5101 21.8726L7.07071 20.7048L15.9798 24.633L10.0758␣
↪25.5177L15.803 26.367L7 30.0829L16.6869 29.0212L11.9848 32.7371L17.3586 30.
↪6138L11.6313 38.2224L19.4798 32.454L17.2525 38.0455L20.8586 33.5157L19.7273 42.
↪9646L23.6515 34.0465L24.5354 39.9565L25.3838 34.2235L29.096 43L28.0354 33.
↪3033L31.7475 38.0101L29.6263 32.6309L37.2273 38.364L31.4646 30.5076L37.0505 32.
↪7371L32.5253 29.1274L41.9646 30.2952L33.0556 26.4024ZM24.5 33.728C19.9747 33.728␣
↪16.298 30.0475 16.298 25.5177C16.298 20.9879 19.9747 17.3074 24.5 17.3074C29.
↪0253 17.3074 32.702 20.9879 32.702 25.5177C32.702 30.0475 29.0253 33.728 24.5 33.
↪728Z" fill="#4CD964"/>
</g>
</g>
</svg>
```

Блок CSS кода

```
/* Стили флагов состояний */

   /* зеленый цвет */
   .normal {
   fill: #4CD964;
   color:#4CD964;
   }
   /* серый цвет */
   .fault {
   fill: #E2E7E9;
   color: #E2E7E9;
   }
   /* красный цвет */
   .in-alarm {
   fill: #FF0000;
   color: #FF0000;
   }
   /* желтый цвет */
   .overridden,
   .out-of-service {
   fill: #FFCC00;
   color: #FFCC00;
   }
/* Скрыть-Отобразить элемент
   /* Скрыть элемент */
   .array-1,
   .array-2,
   .array-3,
   .array-4,
   .array-5,
   .array-6,
```

(continues on next page)

(продолжение с предыдущей страницы)

```
.array-7,
.array-8,
.array-9 {
display: none;
}

/* Отобразить элемент */
.multi-state-1 .array-1,
.multi-state-2 .array-2,
.multi-state-3 .array-3,
.multi-state-4 .array-4,
.multi-state-5 .array-5,
.multi-state-6 .array-6,
.multi-state-7 .array-7,
.multi-state-8 .array-8,
.multi-state-9 .array-9 {
display: block;
}
```

- Описание изменений, сделанных в исходном коде svg-файла:

1. Из исходного кода удаляем свойство `fill=''green''`, делая изображения бесцветными. Цвет будет подставляться в зависимости от флага состояния (`status-flag`) связанного сигнала.
2. Заключаем весь элемент в группу с помощью тэга <g> и задаем этой группе <g `class="fault multi-state-0" reference="MODE">` – это класс для элемента по-умолчанию. `fault` – флаг состояния, связанного сигнала (система изменит этот класс в зависимости от реального состояния, связанного сигнала), `multi-state-0` – значение мультистейт по-умолчанию (0) (система изменит этот класс в зависимости от реального состояния, связанного сигнала).
3. С помощью тэга <g> задаем классы (`class`) каждой из трех групп (<g>): `class для «cool»` ‘‘`array-0`’’ – в случае если значение, связанного сигнала= 0, эта часть svg-файла будет отображена (видима), при иных значениях станет скрыта (невидима); class для «heat» “array-1” – в случае если значение, связанного сигнала= 1, эта часть svg-файла будет отображена (видима), при иных значениях станет скрыта (невидима); class для «auto» “array-2” – в случае если значение, связанного сигнала= 2, эта часть svg-файла будет отображена (видима), при иных значениях станет скрыта (невидима);
4. `reference="DATA"` – это ссылка для связывания с сигналом. Сервер распознает тэг `reference` и изменяет значения в `class="fault multi-state-0"` на соответствующие связанному сигналу состояния (`fault` будет заменен на реальный `status-flag`, `multi-state-0` на реальный `presentValue`).

Пример № 4 – создание элемента управления, на основе «Пример № 3» с мультисостояниями, который в зависимости от значения сигнала управления, отправляет комманду на запись значения:

Адаптированый код Примера № 3

```
<svg width="50" height="50" viewBox="0 0 50 50" fill="none" xmlns="http://www.w3.
↪org/2000/svg">
<g class="fault multi-state-0 " reference="MODE">
<path d="M25 47C37.1503 47 47 37.1503 47 25C47 12.8497 37.1503 3 25 3C12.8497 3 3␣
↪12.8497 3 25C3 37.1503 12.8497 47 25 47Z" fill="#2F3234"/>
<!--cool-->
<g class="array-0">
<path d="M37.1531 29.4909C38.6929 28.723 40.2326 27.9232 41.7024 27.1554C41.9123␣
↪27.0594 42.1573 27.1234 42.2973 27.3154L42.8922 28.3071C43.0322 28.4671 42.9622␣
↪28.723 42.7522 28.819L39.3228 30.6426L42.2973 32.2102C42.5072 32.3062 42.5772 32.
↪5622 42.4373 32.7541L41.8074 33.7779C41.7024 33.9378 41.4224 34.0018 41.2125 33.
↪9059L38.2379 32.3382V36.0174C38.2379 36.2413 38.028 36.4013 37.818 36.4013H36.
↪5582C36.3132 36.4013 36.1383 36.2413 36.1383 36.0174C36.1383 34.4497 36.1033 32.
↪8181 36.1033 31.1865L32.3939 29.2669C32.3939 36.7852 33.7237 36.0814 26.6198 32.
↪3382V36.2413C28.1246 37.0411 29.6293 37.9049 31.0641 38.6728C31.2741 38.7687 31.
↪3441 38.9927 31.2391 39.1846L30.6092 40.1764C30.5042 40.3684 30.2242 40.4324 30.
↪0143 40.3364L26.6198 38.5128V41.6161C26.6198 41.84 26.4099 42 26.1999 42H24.
↪9051C24.6601 42 24.4852 41.84 24.4852 41.6161V38.4808L20.9507 40.3364C20.7408 40.
↪4324 20.4958 40.3684 20.3558 40.2084L19.7609 39.2166C19.621 39.0247 19.6909 38.
↪7687 19.9009 38.6728C21.3707 37.9049 22.9454 37.0091 24.4852 36.2093V32.3062C17.
↪2763 36.1133 18.5711 36.8812 18.5711 29.2989L14.8967 31.2505C14.8967 32.8181 14.
↪8617 34.4497 14.8617 35.9854C14.8617 36.2093 14.6868 36.4013 14.4418 36.4013H13.
↪217C12.972 36.4013 12.7621 36.2093 12.7621 35.9854C12.7621 34.8016 12.7621 33.
↪5859 12.7621 32.3702L9.85754 33.9059C9.64757 34.0018 9.40261 33.9378 9.26263 33.
↪7779L8.63273 32.7541C8.52775 32.5622 8.59774 32.3062 8.77271 32.2102L11.7122 30.
↪6746L8.2128 28.819C8.00284 28.723 7.93285 28.4671 8.07282 28.2751L8.70272 27.
↪3154C8.80771 27.1234 9.05267 27.0594 9.26263 27.1554C10.7674 27.9552 12.3421 28.
↪723 13.8819 29.5229L17.5563 27.5713C10.3824 23.7962 10.3824 25.2038 17.5563 21.
↪4287L13.8469 19.4452C12.3071 20.245 10.8024 21.0448 9.29763 21.8446C9.08766 21.
↪9406 8.8427 21.8766 8.70272 21.6846L8.10782 20.6929C7.96784 20.5329 8.03783 20.
↪277 8.2478 20.181C9.40261 19.5731 10.5574 18.9333 11.7122 18.3254L8.77271 16.
↪7898C8.59774 16.6938 8.52775 16.4378 8.63273 16.2459L9.26263 15.2221C9.40261 15.
↪0622 9.64757 14.9982 9.85754 15.0942L12.7621 16.6298V12.9826C12.7621 12.7587 12.
```

(continues on next page)

(продолжение с предыдущей страницы)

```
↪972 12.5987 13.182 12.5987H14.4418C14.6868 12.5987 14.8617 12.7587 14.8617 12.
↪9826C14.8617 14.5503 14.8967 16.1499 14.8967 17.7495L18.6061 19.7011C18.6061 12.
↪1508 17.2763 12.8867 24.4852 16.6618V12.7587C22.9454 11.9589 21.4407 11.1271 19.
↪9359 10.3272C19.7259 10.2313 19.6559 10.0073 19.7609 9.81536L20.3908 8.82358C20.
↪4958 8.63163 20.7758 8.56764 20.9857 8.66362C22.1056 9.27148 23.2954 9.87934 24.
↪4852 10.5192V7.38391C24.4852 7.15996 24.6601 7 24.9051 7H26.1999C26.4099 7 26.
↪6198 7.15996 26.6198 7.38391V10.4552C27.7746 9.84735 28.8945 9.27148 30.0493 8.
↪66362C30.2592 8.56764 30.5042 8.63163 30.6442 8.79159L31.2391 9.78336C31.379 9.
↪97532 31.3091 10.2313 31.0991 10.3272C29.6293 11.0951 28.1246 11.9269 26.6198 12.
↪7267V16.5978C33.7937 12.8227 32.4289 12.1508 32.4289 19.7331L36.0683 17.8135C36.
↪0683 16.1819 36.1383 14.5503 36.1383 13.0146C36.1383 12.7907 36.3132 12.5987 36.
↪5582 12.5987H37.783C38.028 12.5987 38.2379 12.7907 38.2379 13.0146C38.2379 14.
↪2303 38.2379 15.4461 38.2379 16.6618L41.2125 15.0942C41.4224 14.9982 41.7024 15.
↪0622 41.8074 15.2221L42.4373 16.2459C42.5772 16.4378 42.5072 16.6938 42.2973 16.
↪7898L39.3228 18.3574C40.4776 18.9653 41.6324 19.5731 42.7872 20.181C42.9972 20.
↪277 43.0672 20.5329 42.9272 20.7249L42.2973 21.6846C42.1923 21.8766 41.9473 21.
↪9406 41.7374 21.8446C40.2326 21.0448 38.7279 20.277 37.1881 19.4771L33.5137 21.
↪4287C40.6176 25.1718 40.6176 23.7642 33.4437 27.5393L37.1531 29.4909ZM19.6909 26.
↪4516L23.4003 24.5L19.6909 22.5484L15.9815 24.5L19.6909 26.4516ZM26.6198 22.
↪8044L30.2942 20.8528V16.9177L26.6198 18.8693V22.8044ZM31.379 22.5484L27.6697 24.
↪5L31.3091 26.4196L35.0185 24.468L31.379 22.5484ZM24.4852 26.1956L20.7058 28.
↪1792V32.0503L24.4852 30.0667V26.1956ZM20.7408 20.8208L24.4852 22.8044V18.9333L20.
↪7408 16.9497V20.8208ZM30.2592 28.1152L26.6198 26.1956V30.0667L30.2592 32.0183V28.
↪1152Z" fill="#4CD964"/>
</g>
<!--auto-->
<g class="array-2">
<path d="M12.6865 17.5312H14.875L17.4326 33H15.7188L15.2793 30.0469H12.4756L12.
↪0713 33H10.3398L12.6865 17.5312ZM12.6777 28.5703H15.0508L13.8291 20.3965H13.
↪8027L12.6777 28.5703ZM23.3389 17.5312H25.1494V29.0273C25.1494 30.4746 24.9209 31.
↪5293 24.4639 32.1914C24.0127 32.8535 23.2275 33.1846 22.1084 33.1846C20.9893 33.
↪1846 20.1982 32.8535 19.7354 32.1914C19.2725 31.5293 19.041 30.4746 19.041 29.
↪0273V17.5312H20.8604V28.9834C20.8604 30.0146 20.9014 30.6328 20.9834 30.8379C21.
↪0713 31.0371 21.1621 31.1953 21.2559 31.3125C21.4199 31.5352 21.7041 31.6465 22.
↪1084 31.6465C22.5127 31.6465 22.8115 31.5234 23.0049 31.2773C23.1982 31.0312 23.
↪3008 30.665 23.3125 30.1787C23.3301 29.6924 23.3389 29.2969 23.3389 28.9922V17.
↪5312ZM26.582 17.5312H31.6885V19.043H30.0977V33H28.2871V19.043H26.582V17.5312ZM33.
↪0332 21.7324C33.0332 19.9746 33.3496 18.7676 33.9824 18.1113C34.4805 17.5957 35.
↪2041 17.3379 36.1533 17.3379C37.5361 17.3379 38.4209 17.9062 38.8076 19.043C39.
↪0361 19.7168 39.1504 20.6133 39.1504 21.7324V28.5527C39.1504 30.1641 38.9248 31.
↪3389 38.4736 32.0771C38.0225 32.8096 37.2314 33.1758 36.1006 33.1758C34.9756 33.
↪1758 34.1816 32.8066 33.7188 32.0684C33.2617 31.3301 33.0332 30.1582 33.0332 28.
↪5527V21.7324ZM37.3311 21.627C37.3311 20.6367 37.252 19.9365 37.0938 19.5264C36.
↪9355 19.1104 36.6162 18.9023 36.1357 18.9023C35.6553 18.9023 35.3213 19.1133 35.
↪1338 19.5352C34.9463 19.957 34.8525 20.6514 34.8525 21.6182V28.5703C34.8525 29.
```

(continues on next page)

(продолжение с предыдущей страницы)

```
↪748 34.9346 30.5508 35.0986 30.9785C35.2627 31.4004 35.5938 31.6113 36.0918 31.
↪6113C36.5898 31.6113 36.9209 31.3975 37.085 30.9697C37.249 30.5361 37.3311 29.
↪7363 37.3311 28.5703V21.627Z" fill="#4CD964"/>
</g>
<!--heat-->
<g class="array-1">
<path d="M24.601 32.8739C28.5959 32.8739 31.8485 29.618 31.8485 25.6191C31.8485 21.
↪6201 28.5959 18.3643 24.601 18.3643C20.606 18.3643 17.3535 21.6201 17.3535 25.
↪6191C17.3535 29.618 20.606 32.8739 24.601 32.8739Z" fill="#4CD964"/>
<path d="M33.0556 26.4024L38.9596 25.5177L33.2323 24.6684L42 20.9525L32.3131 22.
↪0142L37.0151 18.2983L31.6414 20.4216L37.3687 12.8129L29.5202 18.5814L31.7475 12.
↪9899L28.1414 17.5197L29.3081 8.07078L25.3838 16.9889L24.5 11.0789L23.6515 16.
↪8119L19.9394 8L21 17.6967L17.2879 12.9899L19.4091 18.3691L11.8081 12.636L17.5707␣
↪20.4924L11.9848 18.2629L16.5101 21.8726L7.07071 20.7048L15.9798 24.633L10.0758␣
↪25.5177L15.803 26.367L7 30.0829L16.6869 29.0212L11.9848 32.7371L17.3586 30.
↪6138L11.6313 38.2224L19.4798 32.454L17.2525 38.0455L20.8586 33.5157L19.7273 42.
↪9646L23.6515 34.0465L24.5354 39.9565L25.3838 34.2235L29.096 43L28.0354 33.
↪3033L31.7475 38.0101L29.6263 32.6309L37.2273 38.364L31.4646 30.5076L37.0505 32.
↪7371L32.5253 29.1274L41.9646 30.2952L33.0556 26.4024ZM24.5 33.728C19.9747 33.728␣
↪16.298 30.0475 16.298 25.5177C16.298 20.9879 19.9747 17.3074 24.5 17.3074C29.
↪0253 17.3074 32.702 20.9879 32.702 25.5177C32.702 30.0475 29.0253 33.728 24.5 33.
↪728Z" fill="#4CD964"/>
</g>
</g>
</svg>
```

Преобразованый код

```
<svg width="50" height="50" viewBox="0 0 50 50" fill="none" xmlns="http://www.w3.
↪org/2000/svg">
<g class="click fault" reference="MODE">
<path d="M25 47C37.1503 47 47 37.1503 47 25C47 12.8497 37.1503 3 25 3C12.8497 3 3␣
↪12.8497 3 25C3 37.1503 12.8497 47 25 47Z" fill="#2F3234"/>
<!--cool-->
<g class="array-0">
<path d="M37.1531 29.4909C38.6929 28.723 40.2326 27.9232 41.7024 27.1554C41.9123␣
↪27.0594 42.1573 27.1234 42.2973 27.3154L42.8922 28.3071C43.0322 28.4671 42.9622␣
↪28.723 42.7522 28.819L39.3228 30.6426L42.2973 32.2102C42.5072 32.3062 42.5772 32.
↪5622 42.4373 32.7541L41.8074 33.7779C41.7024 33.9378 41.4224 34.0018 41.2125 33.
↪9059L38.2379 32.3382V36.0174C38.2379 36.2413 38.028 36.4013 37.818 36.4013H36.
↪5582C36.3132 36.4013 36.1383 36.2413 36.1383 36.0174C36.1383 34.4497 36.1033 32.
↪8181 36.1033 31.1865L32.3939 29.2669C32.3939 36.7852 33.7237 36.0814 26.6198 32.
↪3382V36.2413C28.1246 37.0411 29.6293 37.9049 31.0641 38.6728C31.2741 38.7687 31.
↪3441 38.9927 31.2391 39.1846L30.6092 40.1764C30.5042 40.3684 30.2242 40.4324 30.
↪0143 40.3364L26.6198 38.5128V41.6161C26.6198 41.84 26.4099 42 26.1999 42H24.
↪9051C24.6601 42 24.4852 41.84 24.4852 41.6161V38.4808L20.9507 40.3364C20.7408 40.
```

(continues on next page)

(продолжение с предыдущей страницы)

```
↪4324 20.4958 40.3684 20.3558 40.2084L19.7609 39.2166C19.621 39.0247 19.6909 38.
↪7687 19.9009 38.6728C21.3707 37.9049 22.9454 37.0091 24.4852 36.2093V32.3062C17.
↪2763 36.1133 18.5711 36.8812 18.5711 29.2989L14.8967 31.2505C14.8967 32.8181 14.
↪8617 34.4497 14.8617 35.9854C14.8617 36.2093 14.6868 36.4013 14.4418 36.4013H13.
↪217C12.972 36.4013 12.7621 36.2093 12.7621 35.9854C12.7621 34.8016 12.7621 33.
↪5859 12.7621 32.3702L9.85754 33.9059C9.64757 34.0018 9.40261 33.9378 9.26263 33.
↪7779L8.63273 32.7541C8.52775 32.5622 8.59774 32.3062 8.77271 32.2102L11.7122 30.
↪6746L8.2128 28.819C8.00284 28.723 7.93285 28.4671 8.07282 28.2751L8.70272 27.
↪3154C8.80771 27.1234 9.05267 27.0594 9.26263 27.1554C10.7674 27.9552 12.3421 28.
↪723 13.8819 29.5229L17.5563 27.5713C10.3824 23.7962 10.3824 25.2038 17.5563 21.
↪4287L13.8469 19.4452C12.3071 20.245 10.8024 21.0448 9.29763 21.8446C9.08766 21.
↪9406 8.8427 21.8766 8.70272 21.6846L8.10782 20.6929C7.96784 20.5329 8.03783 20.
↪277 8.2478 20.181C9.40261 19.5731 10.5574 18.9333 11.7122 18.3254L8.77271 16.
↪7898C8.59774 16.6938 8.52775 16.4378 8.63273 16.2459L9.26263 15.2221C9.40261 15.
↪0622 9.64757 14.9982 9.85754 15.0942L12.7621 16.6298V12.9826C12.7621 12.7587 12.
↪972 12.5987 13.182 12.5987H14.4418C14.6868 12.5987 14.8617 12.7587 14.8617 12.
↪9826C14.8617 14.5503 14.8967 16.1499 14.8967 17.7495L18.6061 19.7011C18.6061 12.
↪1508 17.2763 12.8867 24.4852 16.6618V12.7587C22.9454 11.9589 21.4407 11.1271 19.
↪9359 10.3272C19.7259 10.2313 19.6559 10.0073 19.7609 9.81536L20.3908 8.82358C20.
↪4958 8.63163 20.7758 8.56764 20.9857 8.66362C22.1056 9.27148 23.2954 9.87934 24.
↪4852 10.5192V7.38391C24.4852 7.15996 24.6601 7 24.9051 7H26.1999C26.4099 7 26.
↪6198 7.15996 26.6198 7.38391V10.4552C27.7746 9.84735 28.8945 9.27148 30.0493 8.
↪66362C30.2592 8.56764 30.5042 8.63163 30.6442 8.79159L31.2391 9.78336C31.379 9.
↪97532 31.3091 10.2313 31.0991 10.3272C29.6293 11.0951 28.1246 11.9269 26.6198 12.
↪7267V16.5978C33.7937 12.8227 32.4289 12.1508 32.4289 19.7331L36.0683 17.8135C36.
↪0683 16.1819 36.1383 14.5503 36.1383 13.0146C36.1383 12.7907 36.3132 12.5987 36.
↪5582 12.5987H37.783C38.028 12.5987 38.2379 12.7907 38.2379 13.0146C38.2379 14.
↪2303 38.2379 15.4461 38.2379 16.6618L41.2125 15.0942C41.4224 14.9982 41.7024 15.
↪0622 41.8074 15.2221L42.4373 16.2459C42.5772 16.4378 42.5072 16.6938 42.2973 16.
↪7898L39.3228 18.3574C40.4776 18.9653 41.6324 19.5731 42.7872 20.181C42.9972 20.
↪277 43.0672 20.5329 42.9272 20.7249L42.2973 21.6846C42.1923 21.8766 41.9473 21.
↪9406 41.7374 21.8446C40.2326 21.0448 38.7279 20.277 37.1881 19.4771L33.5137 21.
↪4287C40.6176 25.1718 40.6176 23.7642 33.4437 27.5393L37.1531 29.4909ZM19.6909 26.
↪4516L23.4003 24.5L19.6909 22.5484L15.9815 24.5L19.6909 26.4516ZM26.6198 22.
↪8044L30.2942 20.8528V16.9177L26.6198 18.8693V22.8044ZM31.379 22.5484L27.6697 24.
↪5L31.3091 26.4196L35.0185 24.468L31.379 22.5484ZM24.4852 26.1956L20.7058 28.
↪1792V32.0503L24.4852 30.0667V26.1956ZM20.7408 20.8208L24.4852 22.8044V18.9333L20.
↪7408 16.9497V20.8208ZM30.2592 28.1152L26.6198 26.1956V30.0667L30.2592 32.0183V28.
↪1152Z" fill="#4CD964"/>
</g>
<!--auto-->
<g class="array-2">
<path d="M12.6865 17.5312H14.875L17.4326 33H15.7188L15.2793 30.0469H12.4756L12.
↪0713 33H10.3398L12.6865 17.5312ZM12.6777 28.5703H15.0508L13.8291 20.3965H13.
↪8027L12.6777 28.5703ZM23.3389 17.5312H25.1494V29.0273C25.1494 30.4746 24.9209 31.
```

(continues on next page)

(продолжение с предыдущей страницы)

```
↪5293 24.4639 32.1914C24.0127 32.8535 23.2275 33.1846 22.1084 33.1846C20.9893 33.
↪1846 20.1982 32.8535 19.7354 32.1914C19.2725 31.5293 19.041 30.4746 19.041 29.
↪0273V17.5312H20.8604V28.9834C20.8604 30.0146 20.9014 30.6328 20.9834 30.8379C21.
↪0713 31.0371 21.1621 31.1953 21.2559 31.3125C21.4199 31.5352 21.7041 31.6465 22.
↪1084 31.6465C22.5127 31.6465 22.8115 31.5234 23.0049 31.2773C23.1982 31.0312 23.
↪3008 30.665 23.3125 30.1787C23.3301 29.6924 23.3389 29.2969 23.3389 28.9922V17.
↪5312ZM26.582 17.5312H31.6885V19.043H30.0977V33H28.2871V19.043H26.582V17.5312ZM33.
↪0332 21.7324C33.0332 19.9746 33.3496 18.7676 33.9824 18.1113C34.4805 17.5957 35.
↪2041 17.3379 36.1533 17.3379C37.5361 17.3379 38.4209 17.9062 38.8076 19.043C39.
↪0361 19.7168 39.1504 20.6133 39.1504 21.7324V28.5527C39.1504 30.1641 38.9248 31.
↪3389 38.4736 32.0771C38.0225 32.8096 37.2314 33.1758 36.1006 33.1758C34.9756 33.
↪1758 34.1816 32.8066 33.7188 32.0684C33.2617 31.3301 33.0332 30.1582 33.0332 28.
↪5527V21.7324ZM37.3311 21.627C37.3311 20.6367 37.252 19.9365 37.0938 19.5264C36.
↪9355 19.1104 36.6162 18.9023 36.1357 18.9023C35.6553 18.9023 35.3213 19.1133 35.
↪1338 19.5352C34.9463 19.957 34.8525 20.6514 34.8525 21.6182V28.5703C34.8525 29.
↪748 34.9346 30.5508 35.0986 30.9785C35.2627 31.4004 35.5938 31.6113 36.0918 31.
↪6113C36.5898 31.6113 36.9209 31.3975 37.085 30.9697C37.249 30.5361 37.3311 29.
↪7363 37.3311 28.5703V21.627Z" fill="#4CD964"/>
</g>
<!--heat-->
<g class="array-1">
<path d="M24.601 32.8739C28.5959 32.8739 31.8485 29.618 31.8485 25.6191C31.8485 21.
↪6201 28.5959 18.3643 24.601 18.3643C20.606 18.3643 17.3535 21.6201 17.3535 25.
↪6191C17.3535 29.618 20.606 32.8739 24.601 32.8739Z" fill="#4CD964"/>
<path d="M33.0556 26.4024L38.9596 25.5177L33.2323 24.6684L42 20.9525L32.3131 22.
↪0142L37.0151 18.2983L31.6414 20.4216L37.3687 12.8129L29.5202 18.5814L31.7475 12.
↪9899L28.1414 17.5197L29.3081 8.07078L25.3838 16.9889L24.5 11.0789L23.6515 16.
↪8119L19.9394 8L21 17.6967L17.2879 12.9899L19.4091 18.3691L11.8081 12.636L17.5707␣
↪20.4924L11.9848 18.2629L16.5101 21.8726L7.07071 20.7048L15.9798 24.633L10.0758␣
↪25.5177L15.803 26.367L7 30.0829L16.6869 29.0212L11.9848 32.7371L17.3586 30.
↪6138L11.6313 38.2224L19.4798 32.454L17.2525 38.0455L20.8586 33.5157L19.7273 42.
↪9646L23.6515 34.0465L24.5354 39.9565L25.3838 34.2235L29.096 43L28.0354 33.
↪3033L31.7475 38.0101L29.6263 32.6309L37.2273 38.364L31.4646 30.5076L37.0505 32.
↪7371L32.5253 29.1274L41.9646 30.2952L33.0556 26.4024ZM24.5 33.728C19.9747 33.728␣
↪16.298 30.0475 16.298 25.5177C16.298 20.9879 19.9747 17.3074 24.5 17.3074C29.
↪0253 17.3074 32.702 20.9879 32.702 25.5177C32.702 30.0475 29.0253 33.728 24.5 33.
↪728Z" fill="#4CD964"/>
</g>
</g>
</svg>
```

Блок CSS кода

```
/* Стили флагов состояний */

   /* зеленый цвет */
```

(continues on next page)

(продолжение с предыдущей страницы)

```
   .normal {
   fill: #4CD964;
   color:#4CD964;
   }
   /* серый цвет */
   .fault {
   fill: #E2E7E9;
   color: #E2E7E9;
   }
   /* красный цвет */
   .in-alarm {
   fill: #FF0000;
   color: #FF0000;
   }
   /* желтый цвет */
   .overridden,
   .out-of-service {
   fill: #FFCC00;
   color: #FFCC00;
   }
/* Скрыть-Отобразить элемент
   /* Скрыть элемент */
   .array-1,
   .array-2,
   .array-3,
   .array-4,
   .array-5,
   .array-6,
   .array-7,
   .array-8,
   .array-9 {
   display: none;
   }

   /* Отобразить элемент */
   .multi-state-1 .array-1,
   .multi-state-2 .array-2,
   .multi-state-3 .array-3,
   .multi-state-4 .array-4,
   .multi-state-5 .array-5,
   .multi-state-6 .array-6,
   .multi-state-7 .array-7,
   .multi-state-8 .array-8,
   .multi-state-9 .array-9 {
   display: block;
```

(continues on next page)

(продолжение с предыдущей страницы)

```
   }
/* Стили состояния курсора для элементов управления */
   .click:hover {
   cursor: pointer;
   }
```

- Описание изменений, сделанных в адаптированном коде svg-файла:
  - Вместо класса "multi-state-0" назначаем класс "click" и превращаем наш элемент в «кнопку» управления мультистейт сигналом.

  Примечание: таким же образом можно из «аналоговых (цифровых)» и «дискретных (бинарных)» элементов создавать элементы управления. Система сама выберет виджет управления, в зависимости от типа связанного сигнала.

Применяемые CSS классы:

```
/* Стили флагов состояний */

/* задает элементу зеленый цвет */
.normal {
fill: #4CD964;
color:#4CD964;
}
/* задает элементу серый цвет */
.fault {
fill: #E2E7E9;
color: #E2E7E9;
}
/* задает элементу красный цвет */
.in-alarm {
fill: #FF0000;
color: #FF0000;
}
/* задает элементу желтый цвет */
.overridden,
.out-of-service {
fill: #FFCC00;
color: #FFCC00;
}

/* Стили состояния курсора для элементов управления */
.click:hover {
cursor: pointer;
}

/* Маркеры карты */
```

(continues on next page)

(продолжение с предыдущей страницы)

```
#mapicon.sensor {
font-family: Antonio;
font-size: 24px;
text-align: center;
}
#mapicon.sensor.normal {
fill: #00FF00;
color:#00FF00;
}

/* Скрыть элемент */
.active .animation-hide,
.inactive .animation-display,
.array-1,
.array-2,
.array-3,
.array-4,
.array-5,
.array-6,
.array-7,
.array-8,
.array-9 {
display: none;
}

/* Отобразить элемент */
.inactive .animation-hide,
.active .animation-display,
.multi-state-1 .array-1,
.multi-state-2 .array-2,
.multi-state-3 .array-3,
.multi-state-4 .array-4,
.multi-state-5 .array-5,
.multi-state-6 .array-6,
.multi-state-7 .array-7,
.multi-state-8 .array-8,
.multi-state-9 .array-9 {
display: block;
}

/* Неактивные предупреждения для значения "false" */
.inactive .animation-fail {
display: none;
}
```

(continues on next page)

(продолжение с предыдущей страницы)

```
/* Мигания предупреждений для значения "true" */
.active .animation-fail {
display: block;
-webkit-animation: showhide .8s infinite; /* Safari 4+ */
-moz-animation: showhide .8s infinite; /* Fx 5+ */
-o-animation: showhide .8s infinite; /* Opera 12+ */
animation: showhide .8s infinite; /* IE 10+, Fx 29+ */
}
@-webkit-keyframes showhide {
25% { opacity: 0; }
75% { opacity: 1; }
}
@-moz-keyframes showhide {
25% { opacity: 0; }
75% { opacity: 1; }
}
@-o-keyframes showhide {
25% { opacity: 0; }
75% { opacity: 1; }
}
@keyframes showhide {
25% { opacity: 0; }
75% { opacity: 1; }
}

/* Вращения по часовой стрелки. */
.active .animation-spin {
-webkit-animation-name: spin;
-webkit-animation-duration: 3s;
-webkit-animation-iteration-count: infinite;
-webkit-animation-timing-function: linear;
-moz-animation-name: spin;
-moz-animation-duration: 3s;
-moz-animation-iteration-count: infinite;
-moz-animation-timing-function: linear;
-ms-animation-name: spin;
-ms-animation-duration: 3s;
-ms-animation-iteration-count: infinite;
-ms-animation-timing-function: linear;
animation-name: spin;
animation-duration: 3s;
animation-iteration-count: infinite;
animation-timing-function: linear;
}
@-ms-keyframes spin {
```

(continues on next page)

(продолжение с предыдущей страницы)

```
from { -ms-transform: rotate(0deg); }
to { -ms-transform: rotate(359deg); }
}
@-moz-keyframes spin {
from { -moz-transform: rotate(0deg); }
to { -moz-transform: rotate(359deg); }
}
@-webkit-keyframes spin {
from { -webkit-transform: rotate(0deg); }
to { -webkit-transform: rotate(359deg); }
}
@keyframes spin {
from { transform: rotate(0deg); }
to { transform: rotate(359deg); }
}

/* Вращения против часовой стрелки. */
.active .animation-revspin {
-webkit-animation-name: counterspin;
-webkit-animation-duration: 3s;
-webkit-animation-iteration-count: infinite;
-webkit-animation-timing-function: linear;
-moz-animation-name: counterspin;
-moz-animation-duration: 3s;
-moz-animation-iteration-count: infinite;
-moz-animation-timing-function: linear;
-ms-animation-name: counterspin;
-ms-animation-duration: 3s;
-ms-animation-iteration-count: infinite;
-ms-animation-timing-function: linear;
animation-name: counterspin;
animation-duration: 3s;
animation-iteration-count: infinite;
animation-timing-function: linear;
}
@-ms-keyframes counterspin {
from { -ms-transform: rotate(0deg); }
to { -ms-transform: rotate(-359deg); }
}
@-moz-keyframes counterspin {
from { -moz-transform: rotate(0deg); }
to { -moz-transform: rotate(-359deg); }
}
@-webkit-keyframes counterspin {
from { -webkit-transform: rotate(0deg); }
```

(continues on next page)

(продолжение с предыдущей страницы)

```
to { -webkit-transform: rotate(-359deg); }
}
@keyframes counterspin {
from { transform: rotate(0deg); }
to { transform: rotate(-359deg); }
}

/* Центры вращения вентиляторов, крыльчаток и пр. */
.animation-revspin,
.animation-spin {
-webkit-transform-origin: 151px 170px;
-moz-transform-origin: 151px 170px;
-ms-transform-origin: 151px 170px;
transform-origin: 151px 170px;
}

/* Псевдокласс для отображения элемента FAULT   в эл.снабжение */
.active .normal .qf,
.active .in-alarm .qf,
.inactive .normal .qf,
.inactive .in-alarm .qf
{
display: none;
}
.active .fault .qf,
.inactive .fault .qf,
.fault .qf
{
 display: block;
}
```

Глава 3

Формирование и ведение данных

3.1 Правила подготовки данных

В данном разделе описывается формирование модели reference (ссылки) констант и переменных. Классификация констант и переменных позволяет идентифицировать объект и иметь представление об объектах, которым он принадлежит. Данная информация применима к объектам PromUC для сбора, маршрутизации, визуализации и управления.

Примечание: Информация об объектах PromUC Framework (хранение данных) находится в разделе «Порядок и средства заполнения базы данных» и не относится к разделу «Правила подготовки данных»

3.1.1 Классификация объектов

Классификатор объектов имеет обязательную, типизированную часть и пользовательскую часть, которая разрабатывается в рамках проекта.

```
/owner/project/class/objects/objects/objects/
```

- owner - Владелец, логин юридического или физического лица
- project - Проект, название проекта. В одной системе может быть несколько проектов
- class - Класс, назначение классификатора в системе

Остальная часть классификатора имеет произвольную форму, разработанную в рамках проекта.

Примечание: Ниже приводятся примеры применения классификатора для сферы IoT, но данный подход может быть применен к любым моделям в сфере IT и даже за ее пределами.

Пример применения классификатора ниже для получения сигнала с датчика температуры:

```
/gazprom/lakhta-center/telemetry/heating/itp-2/hub-27/ai/27074
```

рекомендуется при проектировании конфигураций для визуализации данных использовать относительный принцип, при этом:

подставится текущий {owner}

```
./lakhta-center/telemetry/heating/itp-2/hub-27/ai/27074
```

подставится текущий {owner} и {project}

```
telemetry/heating/itp-2/hub-27/ai/27074
```

3.1.2 Основные Классы

Наименование класса в классификаторе определяет назначение всего классификатора в системе и находится после {owner} и {project}

/owner/project/class/objects/objects/objects/

Основные классы в PromUC:

- scada - вызывает визуализацию конкретной системы
- template - вызывает шаблон визуализации
- svg - svg примитивы
- img - png, gif, jpeg примитивы
- pult - вызывает пульт управления
- telemetry - данные с датчиков и контроллеров
- control - телеуправление

Примечание: Классы определяют алгоритмы (функции или методы) для работы с этими объектами, могут быть дополнены разработчиками PromUC.

3.1.3 Классификация topic mqtt

Для отправки сообщения в топик брокера MQTT используется классификатор класса {telemetry}

Пример топика публикации MQTT:

```
/gazprom/lakhta-center/telemetry/heating/itp-2/hub-27/ai/27074
```

сокращение топика при отправке в типизированной части не допускается.

Допускается подписка на группу топиков. Пример подписки на группу:

```
/gazprom/lakhta-center/telemetry/*
```

При этом для каждого пользователя в RabbitMQ устанавливается точка входа с правами по приведенному выше примеру.

Для публикации сообщения управления через брокер (в случае если такая функция активирована) служит класс {control} !!!!!

Пример топика управления аналоговым сигналом:

```
/gazprom/lakhta-center/telemetry/heating/itp-2/hub-27/ao/27032 !!!!!!
```

3.2 Порядок и средства заполнения базы данных

PromUC Framework — это гибкий интерфейс доступа к данным, позволяющий создавать необходимую модель данных с возможностью модернизации с минимальными затратами в процессе эксплуатации системы.

Основными элементами модели данных в PromUC Framework являеются Классы, Объекты и Свойства.

Определение:

- Класс - модель для создания объектов определённого типа, описывающая их структуру (набор полей и их начальное состояние) и определяющая алгоритмы (функции или методы) для работы с этими объектами.
- Свойство - описывает объекты, экземпляры в классе, создавая его структуру.
- Объект - экземпляр класса.

Классы классифицируют объекты, которые, в свою очередь, могут также являться классами. Класс может быть наследован от другого класса. Объекты - это экземпляры классов. Они могут иметь входящие или исходящие связи с объектами других классов.

Класс является объектом родительского класса.

Каждый класс описан свойствами. Свойства - это так же объекты определенного класса «Поля» унаследованного от класса «Свойства». Объекты класса «Поля» связаны с свойством Тип данных.

Классы и Объекты PromUC Framework делятся на Системные и Пользовательские.

3.2.1 Системные Классы и Объекты

Системные объекты имеют отражение в коде системы PromUC Framework, определяющем алгоритмы (функции или методы) для работы с этими объектами.

Примечание: Расширение системных объектов - это увеличение функциональности системы и производится разработчиком системы PromUC Framework.

Класс "objects" является корнем системы. Он содержит свойства общие для всех объектов, например, «экземпляр». Расширение класса "objects" позволит добавить свойства всем объектам системы.

Например:

```
Добавим пользовательское свойство "Дата" для обозначения даты создания объекта.
Это свойство получат все объекты системы (Классы, Поля, Связи итд)
```

Все объекты классифицированы и каждый класс является экземпляром (объектом) класса «Классы». Класс «Классы» является экземпляром (объектом) самого себя. Класс «Классы» унаследован от Класса «objects» который так же является экземпляром (объектом) класса «Классы». Другими словами класс «objects» и класс «Классы» классифицированы и описаны в классе «Классы» а сам класс «Классы» унаследован от класса «objects».

Класс «Свойства» является абстрактным (не предполагает создания экземпляров) и описывает общие свойства всех своих потомков. От класса «Свойства» унаследованы классы «Поля», «Значения», «Константы». От класса «Поля» унаследованы Классы «Связи», которые описывают связи между объектами. Класс «Действие» описывает какие действия можно производить с объектами, экземпляры (объекты) класса «Действие» могут быть только системные.

Например:

```
Системный объект "delete" описывает удаление объектов
```

3.2.2 Пользовательские Классы и Объекты

Пользовательские классы не имеют отражение в коде системы PromUC Framework, они классифицируются в системном классе «Классы», поэтому система понимает как с ними работать. Пользователь может создать любую модель данных на основе системных классов, имеющую большое количество пользовательских классов и объектов, а так же связей между ними. Пользователь может наследовать пользовательские классы от системных перенося их функционал на свою модель, заполнять базу данными и получать данные в нужном виде и с необходимыми условиями.

Такой подход позволяет создавать гибкие пользовательские интерфейсы, где не требуется написание специализированных форм, функций или методов для каждой модели работы с базой данных, вместо этого пишутся универсальные инструменты работы с PromUC Framework, что в будущем не требует исправления программного кода в случае модернизации системы.

Например, мы создаем пользовательский класс «Сотрудники» в системном классе «Классы», в классе «Поля» добавляем все необходимые свойства описывающие класс «Сотрудники», например «Фамилия», «Имя», «Отчество» некоторые свойства, например «Должность», связываем с объектами Класса «Должности», который описывается и классифицируется своими свойствами. Форма для заполнения данных в клиентском приложении должна быть разработана универсальная, которая, при открытии, запрашивает объекты класса «Поля» и свойства класса «Сотрудники». В ответ приходит вся необходимая информация описывающая свойства объекта «Сотрудники» в классе «Поля» и «Наименование», «Тип данных» и тд, что позволяет вывести пользователю необходимую форму для просмотра и редактирования данных в различных представлениях. В случае модернизации, например мы добавили классу «Сотрудники» новое свойство, а классу «Поля» объект «Департамент» и связали с объектами класса «Департаменты». Нам нет необходимости дорабатывать программу, так как при открытии формы, она получит все свойства, включая новые, и отобразит их. Это позволит вносить и просматривать новые данные.

На самом деле PromUC Framework позволяет клиенту отдать по запросу сразу всю необходимую информацию структурируя ее в JSON (о чем ниже), но в примере мы разбирали последовательность для понимания как это работает.

3.2.3 Системная модель данных

Для каждого класса создается таблица в базе данных. В таблице, принадлежащей классу, хранятся принадлежащие ему объекты. В столбцах таблицы отражены свойства класса, которые, в свою очередь, являются объектами класса «Поля» кроме унаследованных, унаследованные свойства хранятся в родительских классах. Иначе говоря таблица, соответствующая классу, содержит в столбцах специфичные ему свойства.

Идентификаторы всем экземплярам классов (объектам) присваиваются сквозным методом; для системных, начиная с 10 000 000, и для пользовательских, начиная с 20 000 000. Колонки таблиц связаны с объектами класса «Связь» унаследованного от класса «Поля» и именуются тем же сквозным идентификатором.

Таблица 1: Модель БД

Класс	Свойства	Наименование
10 000 000		Объекты
	10 000 000	
	10 000 022	Класс-владелец
	10 000 023	Объекты-помощники
10 000 001		Классы
	10 000 000	
	10 000 024	Обозначение
	10 000 025	Название
	10 000 026	
	10 000 027	Унаследованы
	10 000 059	Количество элементов
10 000 002		Внутренний тип
	10 000 000	

continues on next page

Таблица 1 – продолжение с предыдущей страницы

Класс	Свойства	Наименование
	10 000 028	Обозначение
	10 000 029	Название
	10 000 030	PG тип
	10 000 031	PG код скаляра
	10 000 032	PG код массива
10 000 003		Типы
	10 000 000	
	10 000 033	Обозначение
	10 000 034	Название
	10 000 035	Внутренний тип
10 000 004		Свойства
	10 000 000	
	10 000 036	Класс-владелец
	10 000 037	Обозначение
	10 000 038	Название
	10 000 039	Тип
	10 000 040	Массив
	10 000 041	Измерения
	10 000 042	Значение функции
10 000 005		Поля
	10 000 000	
	10 000 043	Значение по-умолчанию
	10 000 044	Обязательное
10 000 006		Связи
	10 000 000	
	10 000 045	Класс источник
	10 000 046	Связь с владельцем
	10 000 047	Обратная связь
10 000 007		Значения
	10 000 000	
10 000 008		Константы
	10 000 000	
10 000 009		Функции
	10 000 000	
10 000 010		Действия
	10 000 000	
	10 000 048	Обозначение
	10 000 050	Тип
	10 000 051	Массив
	10 000 052	Измерения

3.2.4 Основные решения Framework

- создание интерфейса пользователя с гибкими возможностями для доступа к хранилищу информации;
- получение наглядного представления связанных данных;
- создание с помощью технологии no code хранилища данных необходимой модели с быстрым доступом к нему по API;
- получение данных по API используя различные фильтры, условия, сортировки, ограничения;
- поиск данных;
- создание приложения на основе данных и автоматическое изменение формы и структуры приложения, следующее за изменениями структуры данных.

Основная идея работы с сервисом - своя уникальная гибкая модель данных, позволяющая обеспечить все функциональные требования с возможностью модернизации с минимальными затратами в процессе эксплуатации.

3.3 Процедуры изменения и контроля базы данных

3.3.1 API

"GET","/api/get/[id]", api_get_object, «Получить объект по id»

"GET","/api/delete/[id]", api_delete_object, «Удалить объект по id»

"GET","/api/list/{имя класса}", api_get_list, «Получить объекты, возможно добавить условия»

"GET","/api/delete/{имя класса}", api_delete_list, «Удалить объекты по условию»

"GET","/api/json/[id]/{имя поля-связи}", api_get_json, «Получить объекты и объекты по указанным связям»

"POST","/api/put", api_save_object, «Сохранить объект. Если нет ID - создать объект, owner_id - класс, содержащий объект»

"POST","/api/puts/[class_id]", api_save_objects, «Сохранить набор объектов класса [class_id]»

3.3.2 Фильтры

PromUC Framework обладает возможностью гибкой фильтрации и сортировки данных, позволяя в запросах передавать необходимые условия и делать поиск данных по сложному критерию

Методы фильтрации по условию: `filter` - ключевое слово, которое означает, что дальше будут условия фильтрации `field` (свойство), `value` (значение):

- `filter[field]=value` «Фильтрация по условию»
- `filter[field]=value&filter[field]=value` «Фильтрация по нескольким условиям»

Методы фильтрации по выражению: - `filter` ключевое слово, которое означает, что дальше будут условия фильтрации - `field` (свойство), `value` (значение)

- `filter[field]=[">",value]` «Фильтрация по выражению больше»
- `filter[field]=["<",value]` «Фильтрация по выражению меньше»
- `filter[field]=["!=",value]` «Фильтрация по выражению не равно»
- `filter[field]=["<>",value]` «Фильтрация по выражению не равно»
- `filter[field]=[">=",value]` «Фильтрация по больше или равно»
- `filter[field]=["<=",value]` «Фильтрация по меньше или равно»
- `filter[field]=["~",value]` «Фильтрация по выражению содержит»

Если тип фильтруемого поля ссылется на внутренний тип bool, то фильтр будет выглядеть так:

```
filter[field]=value
```

Если тип фильтруемого поля ссылается на внутренний тип text, то тогда следует экранировать value двойными кавычками ":

```
filter[field]=["~","value"]
filter[field]=["<>","value"]
filter[field]=["!=","value"]
```

Методы сортировки:

- `order` - ключевое слово, которое означает, что дальше будут условия сортировки
- `id, name` - (свойство) для сортировки по нескольким свойствам
- `class_id-` - (свойство) для обратной сортировки

```
order=id                сортировка по ID
order=name,class_id-    сортировка сначала по name, а затем обратная по class_id
```

Метод ограничения вывода:

- `limit` - лимит выводимых объектов по результату поиска.
- `offset` - смещение объектов (пропуск с первого найденного)

```
limit=100      ограничить вывод 100 объектами
offset=1000    получить объекты с смещением на 1000 объектов
```

Можно совмещать различные условия, выражения, сортировки, например:

```
* /api/list/{class}?filter[fieldName1]=fieldValue1&filter[fieldName2]=fieldValue2&
↪offset=1000&limit=100
```

3.3.3 Удаление объектов

Для удаления объектов используйте API описанный в разделе «API». При удалении объекта, у которого есть наследники, его наследники тоже будут удалены. Удаление данных возможно как по условию так и с использованием фильтров.

Пример метода «Удаление по значению»:

```
/api/delete/{mnemo}
```

Пример метода «Удаление используя фильтры»:

```
/api/delete/{class}?filter[id]=20000243
```

3.4 Порядок и средства восстановления базы данных

Модель данных может быть сохранена с учетом версионности в формате YAML и использована на других проектах. Системную и Пользовательскую модель рекомендуется сохранять в отдельных файлах конфигурации.

Глава 4

Конфигурирование правил и сценариев

4.1 Введение

Rule Engine - это простая в использовании платформа для создания рабочих процессов на основе событий. Есть 3 основных компонента:

- **Сообщение** - любое входящее событие. Это могут быть входящие данные с устройств, событие жизненного цикла устройства, событие REST API, запрос RPC и т. д.
- **Узел правила** - функция обработки входящего сообщения. Существует много различных типов узлов, которые могут фильтровать, преобразовывать или выполнять некоторые действия с входящим сообщением.
- **Цепочка правил** - узлы соединяются друг с другом связями, поэтому исходящее сообщение от узла правила отправляется следующим подключенным узлам правил.

4.1.1 Сценарии использования

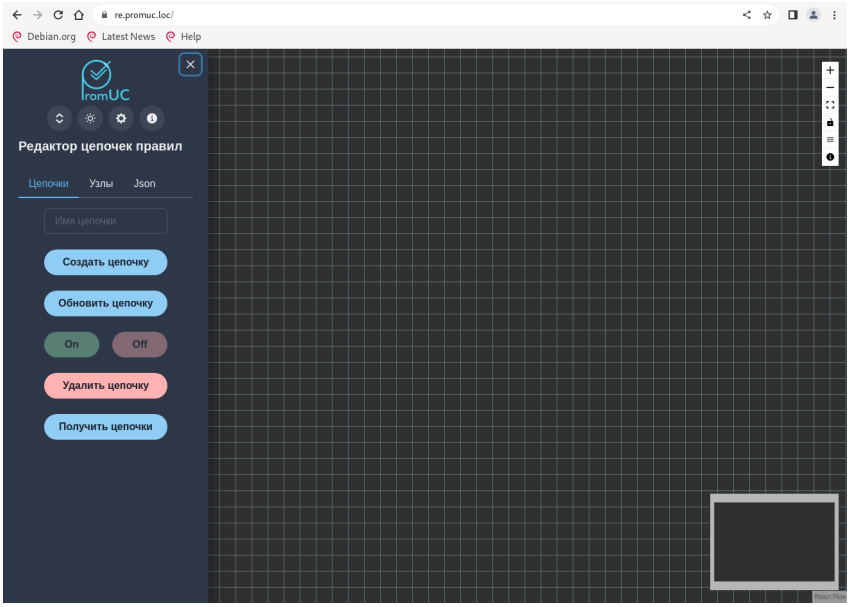
Rule Engine - это настраиваемая платформа для обработки сложных событий. Вот несколько распространенных сценариев использования:

- Проверка и изменение данных для входящей телеметрии или атрибутов перед сохранением в базе данных.
- Агрегация телеметрии. Например, данные с нескольких устройств могут быть агрегированы в соответствующий актив.
- Создание/обновление/очистка сигналов на основе определенных условий.
- Запуск действий на основе событий жизненного цикла устройства. Например, создание оповещения, если устройство находится в режиме онлайн/оффлайн.

- Загрузка дополнительных данных, необходимых для обработки. Например, порогового значения температуры устройства, которое определено в атрибуте владельца устройства.
- Запуск вызовов REST API во внешние системы.
- Отправка электронных писем при возникновении сложного события и использование атрибутов других сущностей в шаблоне электронной почты.
- Обработка событий в соответствии с настройками в аккаунте пользователя.
- Вызовы RPC на основе определенных условий.
- Интеграция с внешними системами, через Kafka, MQTT, REST API и т. д.

4.1.2 Запуск Rule Engine

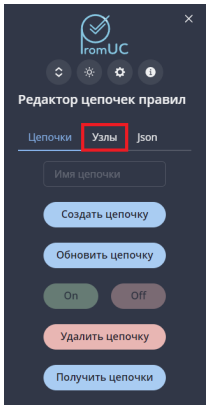
Чтобы открыть интерфейс Rule Engine необходимо перейти в web браузере по ссылке `https://re.имядомена`, например, `https://promuc.loc`:



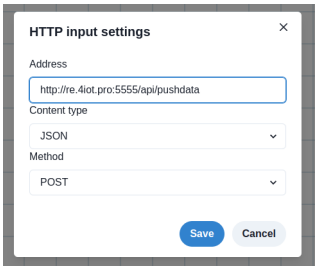
4.1.3 Пример использования - добавление узла проверки температуры

Предположим, что ваше устройство использует датчик DHT22 для сбора и передачи температуры в Rule Engine. Датчик DHT22 может измерять температуру от -40°C до +80°C. В этом примере мы настроим Rule Engine для хранения температуры в диапазоне от -40 до 80 °C и регистрации всех остальных показаний в MQTT брокер.

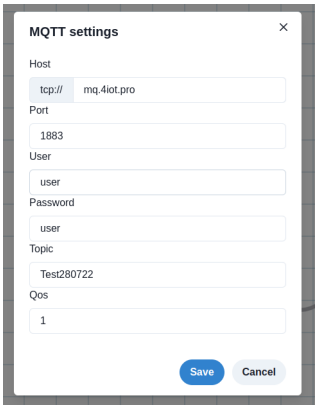
1. Откройте страницу Rule Engine.
2. Выберите `Узлы`:



3. Перетащите узел `HTTP Input` в цепочку. Откроется окно настройки узла. Мы будем использовать следующие настройки узла:

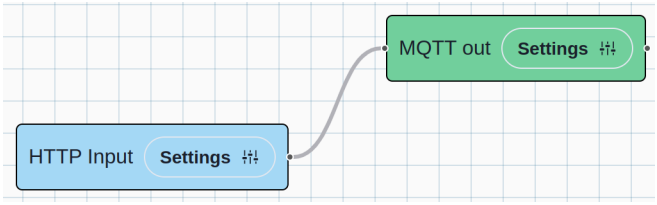


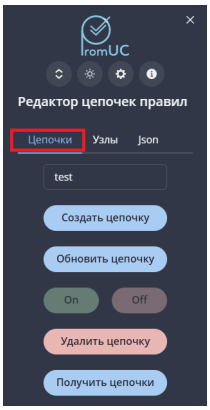
4. Перетащите узел `MQTT` в цепочку. Откроется окно настройки узла. Мы будем использовать следующие настройки узла:



5. Подключите узел `HTTP Input` к узлу `MQTT Out` с помощью связи:

6. Перейдите во вкладку `Цепочки`, введите имя цепочки и нажмите «Создать цепочку»:





7. Для того чтобы проверить работу цепочки можно отправить данные командой curl:

```
{
  curl -k -H "Content-Type: application/json" -X POST -d '{"temp":23}'␣
↪http://re.4iot.pro:5555/api/pushdata
}
```

Результат можно увидеть в MQTT в соответствующем топике.

4.1.4 Типы сообщений внутри Rule Engine

- Запрос устройства на публикацию атрибутов

  Тип: `POST_ATTRIBUTES_REQUEST`

  Имя: `Post attributes`

  Метаданные: `deviceName, deviceType`

  Формат сообщения: `key/value json`

  ```
  {
    "currentState": "IDLE"
  }
  ```

- Запрос устройства на публикацию телеметрии

  Тип: `POST_TELEMETRY_REQUEST`

  Имя: `Post telemetry`

  Метаданные: `deviceName, deviceType`

Формат сообщения: key/value json

```
{
  "temperature": 22.7
}
```

- RPC вызов с устройства на сервер

  Тип: TO_SERVER_RPC_REQUEST

  Имя: RPC Request from Device

  Метаданные: deviceName, deviceType, requestId

  Формат сообщения: json

```
{
  "method": "getTime",
  "params": {
      "param1": "val1"
  }
}
```

- RPC вызов с сервера на устройство

  Тип: RPC_CALL_FROM_SERVER_TO_DEVICE

  Имя: RPC Request to Device

  Метаданные: deviceType, oneway

  Формат сообщения: json

```
{
  "method": "getGpioStatus",
  "params": {
      "param1": "val1"
  }
}
```

- Событие об активности устройства

  Тип: ACTIVITY_EVENT

  Имя: Activity Event)

  Метаданные: deviceType, oneway

  Формат сообщения: json

```
{
  "active": true,
  "lastConnectTime": 1526979083267,
  "lastActivityTime": 1526979083270,
```

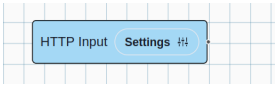
(continues on next page)

(продолжение с предыдущей страницы)

```
  "lastDisconnectTime": 1526978493963,
  "lastInactivityAlarmTime": 1526978512339,
  "inactivityTimeout": 10000
}
```

4.2 Узлы входящих сообщений

4.2.1 Узел HTTP Input

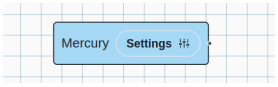


Узел приема данных по REST API.

Пример отправки данных:

```
curl -k -H "Content-Type: application/json" -X POST -d '{"data":10, "some": 1}'␣
↪http://re.4iot.pro:5555/api/pushdata
```

4.2.2 Узел Mercury



Узел опроса счетчиков электрической энергии Меркурий - 234

Конфигурация узла:

- `Addresses` - Адрес счетчика (можно указать несколько адресов через запятую «11,45,44»)
- `Delay, m` - Задержка между опросами счетчиков, мин
- `Boudrate` - Скорость линии RS-485
- `Bytesize` - Количество бит данных
- `Timeout` - Время ожидания ответа от счетчика, сек

Узел опрашивает следующие значения:

- ток
- напряжение
- cos(f) по всем фазам
- углы между фазными напряжениями
- активную и реактивную энергию по всем фазам

- накопленную энергию за текущий день, за предыдущий день и нарастающим итогом

Пример вывода блока:

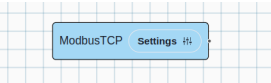

```
{
    "U": {
        "p1": 0.35,
        "p2": 0.35,
        "p3": 226.86
    },
    "I": {
        "p1": 0.00,
        "p2": 0.00,
        "p3": 0.39
    },
    "CosF": {
         "p1": 0.00,
         "p2": 0.00,
         "p3": 0.60,
         "sum": 0.60
    },
    "F": 50.00,
    "A": {
        "p1": 41943.03,
        "p2": 41943.03,
        "p3": 41943.03
    },
    "P": {
        "p1": 0.00,
        "p2": 0.00,
        "p3": 53.45,
        "sum": 53.45
    },
    "S": {
        "p1": 0.00,
        "p2": 0.00,
        "p3": 89.83,
        "sum": 89.83
    },
    "PR": {
        "ap": 120.51
    },
    "PR-day": {
        "ap": 86.00
    },
    "PR-night": {
        "ap": 34.51
```


(continues on next page)

(продолжение с предыдущей страницы)

```
    },
    "PY": {
        "ap": 0.00
    },
    "PT": {
        "ap": 0.04
    }
}
```

4.2.3 Узел ModbusTCP



Узел опроса устройств по протоколу Modbus TCP

Конфигурация:

- `Addresses` - ip адрес устройства
- `Port` - порт
- `Slave ID` - ID устройства
- `Reg addr` - адрес регистра для опроса
- `Num of reg` - количество опрашиваемых регистров
- `Function` - модбас функция опроса
- `Timeout` - время ожидание ответа
- `Settings file` - загрузка настроек из файла

Формат файла с настройками:

```
[
  {
    "Name": "first_device",
    "Ip": "192.168.1.158",
    "Port": 502,
    "Registers": [
      {
        "Function": 1,
        "Name": "BulkTank_Lvl",
        "Reference": 0,
        "Num": 1,
        "Data Type": "BOOL",
        "Scale": 10,
```

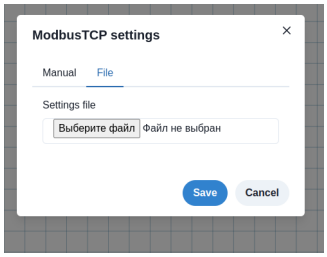
(continues on next page)

(продолжение с предыдущей страницы)

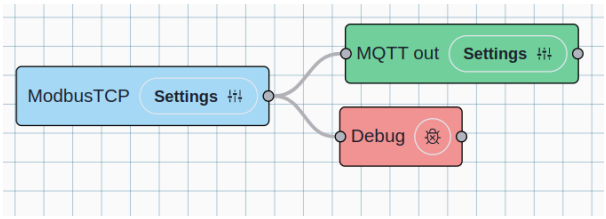
```
        "Frequency": 1000
      },
      {
        "Function": 2,
        "Name": "BulkTank_Lvl",
        "Reference": 1,
        "Num": 1,
        "Data Type": "INT16",
        "Scale": 10,
        "Frequency": 1000
      },
      {
        "Function": 3,
        "Name": "BulkTank_Lvl",
        "Reference": 9999,
        "Num": 1,
        "Data Type": "INT32",
        "Scale": 10,
        "Frequency": 1000
      },
      {
        "Function": 4,
        "Name": "BulkTank_Lvl",
        "Reference": 3,
        "Num": 1,
        "Data Type": "UINT16",
        "Scale": 10,
        "Frequency": 1000
      }
    ]
  }
]
```

Пример создания цепочки из файла:

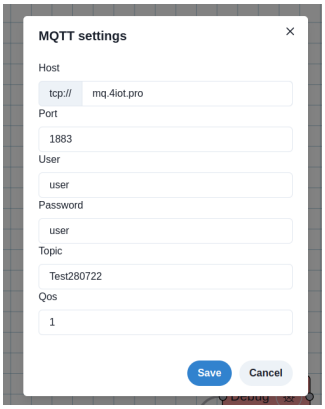
1. Добавим узел `ModbusTCP` - Выберем вкладку File - загрузим файл ввиде, как указано выше, в формате `json`:



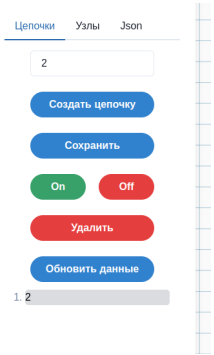
2. Добавим узел Debug и MQTT, соединим узлы связью:



3. Настроим узел MQTT:

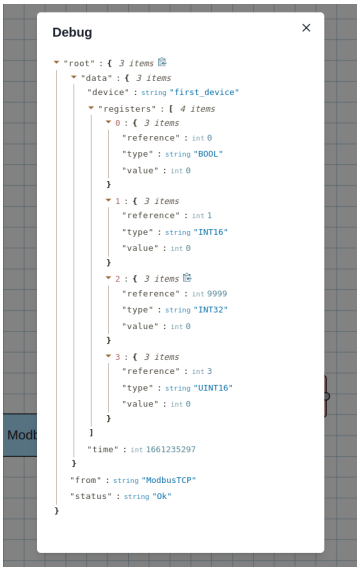


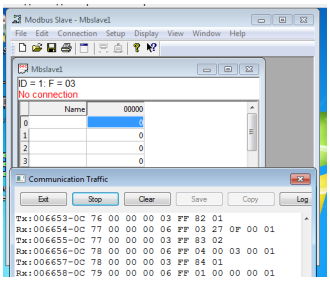
4. Сохраняем цепочку:



5. Смотрим результат внутри узла Debug или на MQTT брокере в соответствующем топике:

6. Для эмуляции данных используется ПО Modbus Slave:





4.2.4 Узел BACnet

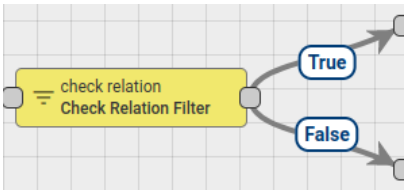
Узел опроса устройств в сети BACnet.

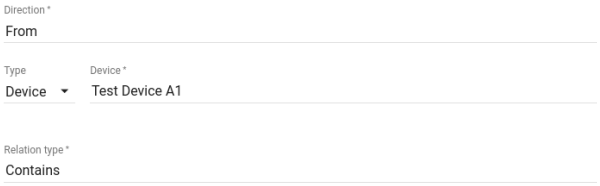
4.3 Узлы фильтрации и маршрутизации

4.3.1 Узел проверки типа и направления сообщения

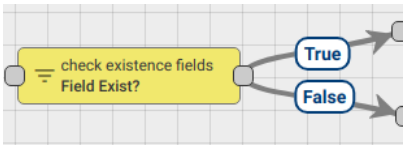
Проверяет принадлежность сообщения к отправителю по типу и направлению

Если отправитель совпадает, то на выходе `True`, иначе `False`.

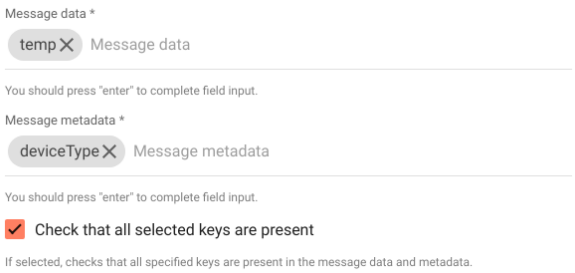




4.3.2 Узел проверки наличия полей сообщения

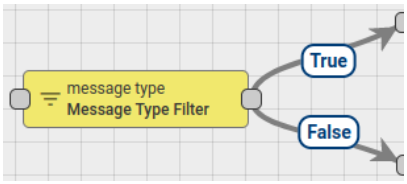


Узел проверяет наличие выбранных ключей из входящих данных сообщения и метаданных.



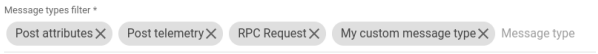
Если все поля присутствуют, то на выходе `True`, иначе `False`.

4.3.3 Узел проверки типа сообщения

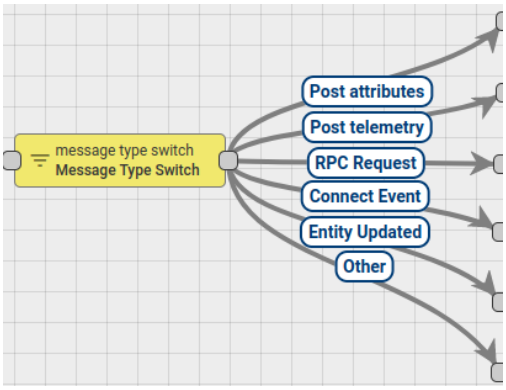


В конфигурации узла администратор определяет набор разрешенных типов сообщений. В системе есть предопределенные типы сообщений, такие как `Post attributes`, `Post telemetry`, `RPC Request`, и т.д. Администратор также может определить любые настраиваемые типы сообщений в конфигурации узла.

Если тип сообщения совпадает, то на выходе `True`, иначе `False`.

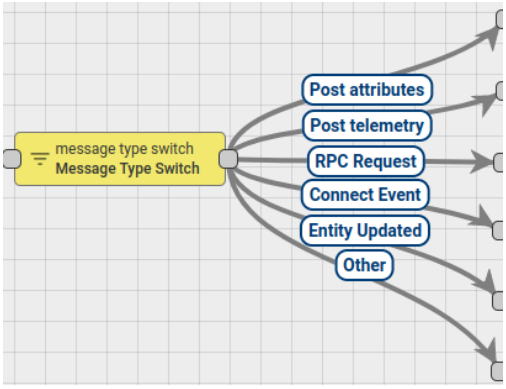


4.3.4 Узел выбора типа сообщений



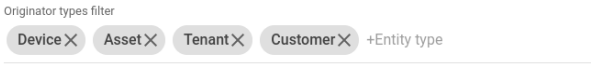
Маршрутизация входящих сообщений по типу сообщения. Сообщение отправляется в соответствующую его типу цепочку. Если тип сообщения не подходит ни к одной цепочке, оно отправляется в цепочку `Other`.

4.3.5 Узел проверки типа отправителя сообщений

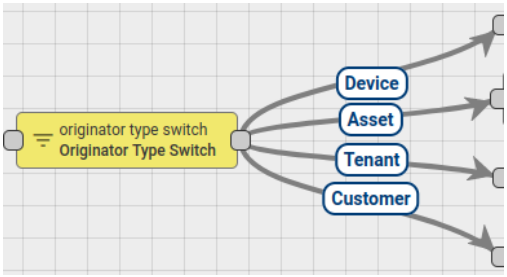


В конфигурации узла администратор определяет разрешенные типы отправителей для входящего сообщения.

Если тип сообщения совпадает, то на выходе `True`, иначе `False`.

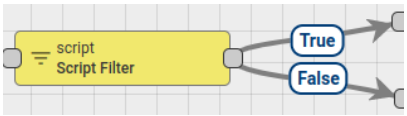


4.3.6 Узел выбора сообщений по типу отправителя



Маршрутизирует входящие сообщения по типу отправителя.

4.3.7 Узел скрипт - фильтр

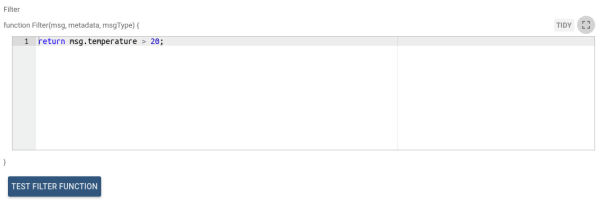


Обрабатывает входящее сообщение через JavaScript.

Функция JavaScript получает 3 входных параметра:

- `msg` — полезная нагрузка сообщения.
- `metadata` - это метаданные сообщения.
- `msgType` - тип сообщения.

Скрипт должен возвращать логическое значение. Если `True` - отправить сообщение через цепочку `True`, иначе используется цепочка `False`.



Доступ к полезной нагрузке сообщения можно получить через переменную `msg`. Например:

```
msg.temperature < 10;
```

Доступ к метаданным сообщения можно получить через переменную `metadata`. Например:

```
metadata.customerName === 'Джон'
```

Доступ к типу сообщения можно получить через переменную `msgType`. Например:

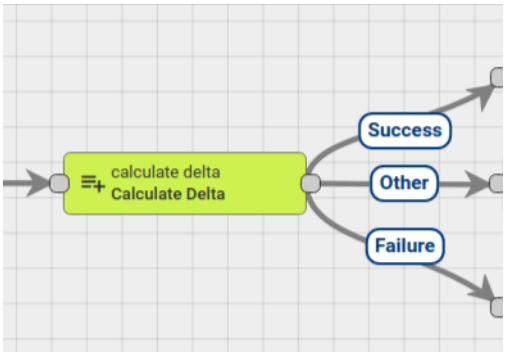
```
msgType === 'POST_TELEMETRY_REQUEST'
```

Пример скрипта:

```
if(msgType === 'POST_TELEMETRY_REQUEST') {
      if(metadata.deviceType === 'vehicle') {
            return msg.humidity > 50;
      } else if(metadata.deviceType === 'controller') {
            return msg.temperature > 20 && msg.humidity > 60;
      }
}
return false;
```

4.4 Узлы насыщения

4.4.1 Узел вычисления дельты



Вычисляет дельту между предыдущим показанием и текущим, и добавляет ее к сообщению. Полезно для интеллектуального учета. Например, когда прибор учета воды сообщает абсолютное значение счетчика один раз в сутки. Чтобы узнать потребление за текущий день, необходимо сравнить значение за предыдущий день со значением за текущий день.

Расчет дельты выполняется для данных отправителя, например, для устройства, ресурса или потребителя.

Конфигурация узла:

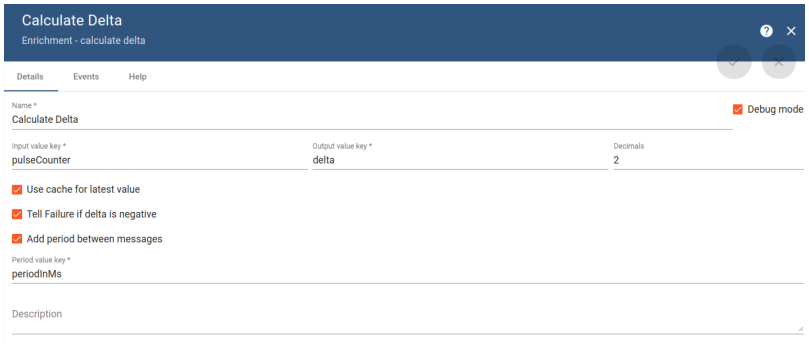
- `Входные данные` - (по-умолчанию `pulseCounter`) поле, которое будет использоваться для выходных данных.
- `Выходные данных` - (по-умолчанию `delta`) поле, в котором будет храниться результат вычисления дельты.
- `Точность` - точность расчета дельты.
- `Использование кэша` - (по-умолчанию `enabled`) включает кэширование последних значений в памяти.

- `Отказ, если дельта отрицательна` - (по-умолчанию `enabled`) отправляет сообщение в цепочку `Отказ`, если значение дельты отрицательное.
- `Задержка между сообщениями` - (по-умолчанию `disabled`) добавляет задержку между текущим и предыдущим сообщением.

Тип выходного сообщения:

- `Успех` - если во входящем сообщении присутствует поле с входными данными;
- `Другое` - если во входящем сообщении отсутствует поле с входными данными;
- `Отказ` - если установлен параметр `Сообщать Отказ, если дельта отрицательна` и расчет дельты возвращает отрицательное значение.

Давайте рассмотрим поведение узла правила на примере. Предположим следующую конфигурацию:



Предположим, что следующие сообщения исходят от одного и того же устройства и поступают на узел правил в той последовательности, в которой они перечислены:

```
{
      "msg": {
            "pulseCounter": 42
      },
      "metadata": {
            "ts": "1616510425000"
      }

}
{
      "msg":
      {
            "pulseCounter": 73
      },
      "metadata": {
            "ts": "1616510485000"
      }
}
{
```

(continues on next page)

(продолжение с предыдущей страницы)

```
      "msg": {
            "temperature": 22
      },
      "metadata": {
            "ts": "1616510486000"
      }
      "msg": {
            "pulseCounter": 42
      },
      "metadata": {
            "ts": "1616510487000"
      }
}
```

Вывод будет следующим:

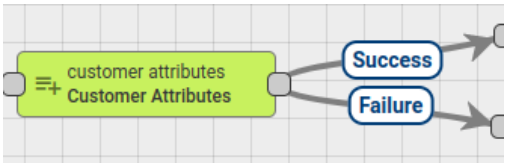
```
{
      "msg": {
            "pulseCounter": 42,
            "delta": 0,
            "periodInMs": 0
      },
      "metadata": {
            "ts": "1616510425000"
      },
      "relation": "Success"
}
{
      "msg": {
            "pulseCounter": 73,
            "delta": 31,
            "periodInMs": 60000
      },
      "metadata": {
            "ts": "1616510485000"
      },
      "relation": "Success"
}
{
      "msg": {
            "temperature": 22
      },
      "metadata": {
            "ts": "1616510486000"
      },
      "relation": "Other"
```

(continues on next page)

(продолжение с предыдущей страницы)

```
}
{
      "msg": {
            "pulseCounter": 42
      },
      "metadata": {
            "ts": "1616510487000"
      },
      "relation": "Failure"
}
```

4.4.2 Узел атрибутов получателя



Сопоставление между полем данных и полем метаданных.

Метаданные исходящего сообщения будут содержать настроенные атрибуты, если они существуют. Чтобы получить доступ к выбранным атрибутам в других узлах, вы можете использовать этот шаблон `metadata.temperature`.

Допускаются следующие типы отправителей сообщений: `Customer, User, Asset, Device`. Если обнаружен неподдерживаемый тип отправителя, выдается ошибка. Если отправитель не указал данные получателя, используется цепочка `Failure`, иначе `Success`. Вы можете использовать `${metadataKey}` для значения из метаданных, `$[messageKey]` для значения из тела сообщения.

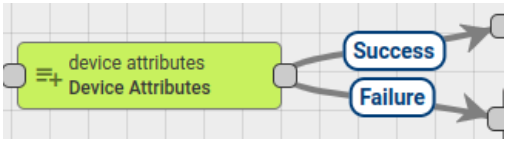
Пример: есть следующие метаданные `{"country": "England"}`. Кроме того, у вас есть поле названия страны, со значением — столица (`{"England": "London"}`).

Цель состоит в том, чтобы получить столицу из атрибута страны из метаданных и добавить результат в метаданные с ключом «город». Для этого вы можете использовать `${country}` в качестве исходного атрибута и `city` в качестве целевого атрибута.

Результатом будет `{"city": "London"}`.

Вы можете увидеть пример из реальной жизни, где используется этот узел, в примере отправки сообщений по электронной почте. TODO ссылку

4.4.3 Узел атрибутов устройства

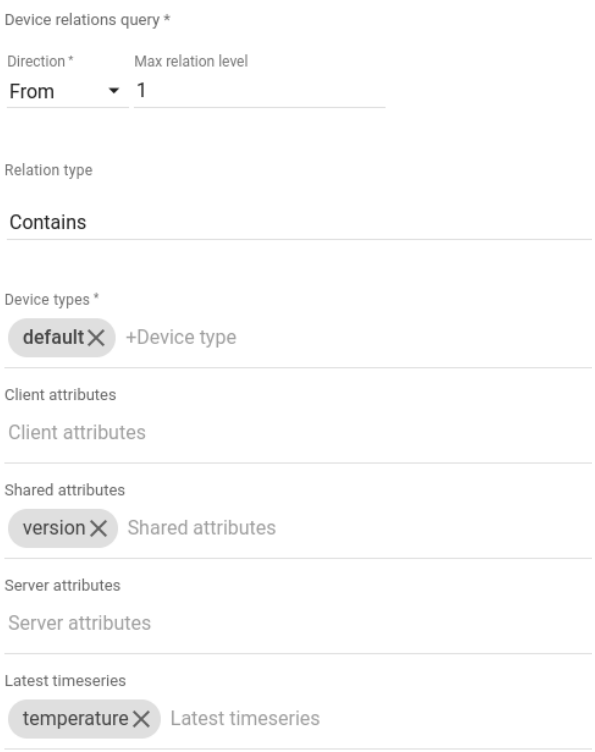


Узел находит устройство, связанное с отправителем, с помощью настроенного запроса и добавляет поля `потребитель`, `область видимости` и `значение последней телеметрии` в метаданные сообщения.

Атрибуты добавляются в метаданные с префиксом области видимости:

- общий атрибут -> `shared_`
- атрибут клиента -> `cs_`
- атрибут сервера -> `ss_`
- телеметрия -> префикс не используется

Например, общий атрибут «`версия`» будет добавлен в метаданные с именем «`общая_версия`». Атрибуты клиента будут использовать префикс `cs_`. Атрибуты сервера используют префикс `ss_`. Последнее значение телеметрии добавляется в метаданные сообщения как есть, без префикса. В конфигурации «Запрос взаимосвязей устройств» администратор может выбрать необходимое направление и уровень глубины взаимосвязи. Также тип отношения можно настроить с помощью необходимого набора типов устройств.



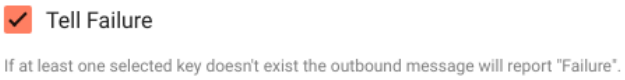
Если было найдено несколько связанных объектов, для расширения атрибутов используется только первый объект, остальные объекты будут отброшены.

Если связь не найдена, то цепочка `Failure`, иначе `Success`.

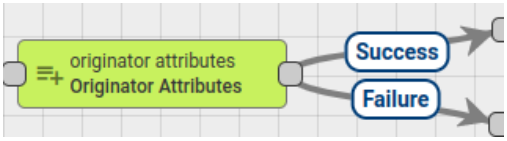
Если атрибут или телеметрия не были найдены, они не добавляются в метаданные сообщения и по-прежнему направляются через цепочку `Success`.

Метаданные исходящего сообщения будут содержать настроенные атрибуты, только если они существуют. Чтобы получить доступ к выбранным атрибутам в других узлах, вы можете использовать этот шаблон `metadata.temperature`.

Узел правил имеет возможность включать/отключать отчеты об ошибках, если хотя бы один выбранный ключ не существует в исходящем сообщении:



4.4.4 Узел атрибутов отправителя



Добавьте атрибуты отправителя сообщения (область клиент\общий\сервер) и значение последней телеметрии в метаданные сообщения.

Атрибуты добавляются в метаданные с префиксом области видимости:

- общий атрибут -> `shared_`
- атрибут клиента -> `cs_`
- атрибут сервера -> `ss_`
- телеметрия -> префикс не используется

Например, общий атрибут `версия` будет добавлен в метаданные с именем `общая_версия`.

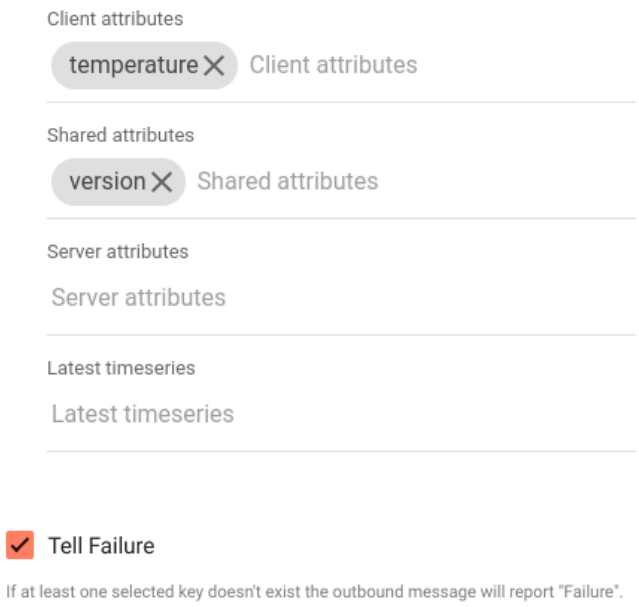
Атрибуты клиента будут использовать префикс `cs_`.

Атрибуты сервера используют префикс `ss_`. Последнее значение телеметрии добавляется в метаданные сообщения как есть, без префикса.

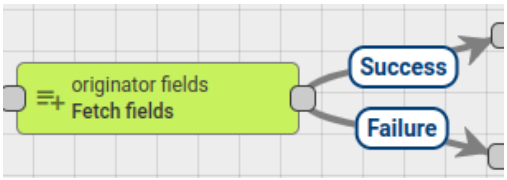
Метаданные исходящего сообщения будут содержать настроенные атрибуты, если они существуют.

Чтобы получить доступ к извлеченным атрибутам в других узлах, вы можете использовать шаблон `metadata.cs_temperature`.

Узел правил имеет возможность включать/отключать отчеты об ошибках, если хотя бы один выбранный ключ не существует в исходящем сообщении:



4.4.5 Узел полей отправителя



Узел извлекает значения полей сущности отправителя сообщения и добавляет их в метаданные сообщения. Администратор может настроить сопоставление между именем поля и именем атрибута метаданных. Если указанное поле не является частью полей объекта отправителя сообщения, оно будет проигнорировано.

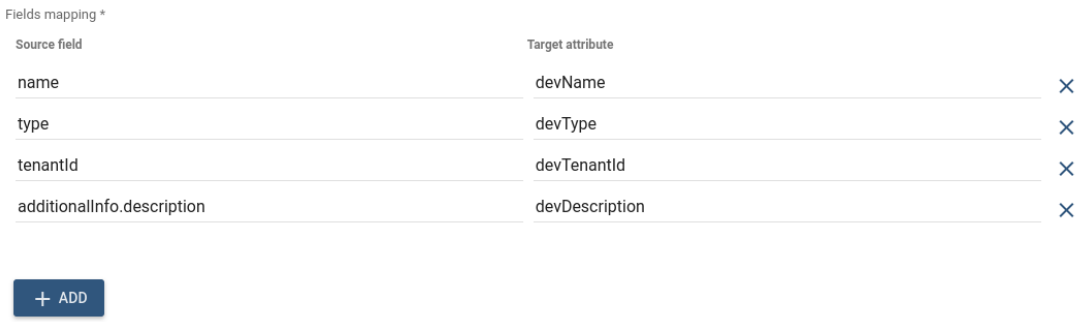
Допускаются следующие типы отправителей сообщений: `Tenant`, `Customer`, `User`, `Asset`, `Device`, `Alarm`, `Rule Chain`.

Если тип отправителя неизвестен, используется цепочка `Отказ`, иначе - цепочка `Успех`.

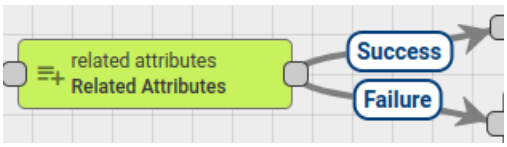
Если значение поля не найдено, оно не добавляется в метаданные сообщения и по-прежнему маршрутизируется через цепочку успеха.

Метаданные исходящего сообщения будут содержать настроенные атрибуты, только если они существуют.

Чтобы получить доступ к извлеченным атрибутам в других узлах, вы можете использовать этот шаблон `metadata.devType`.



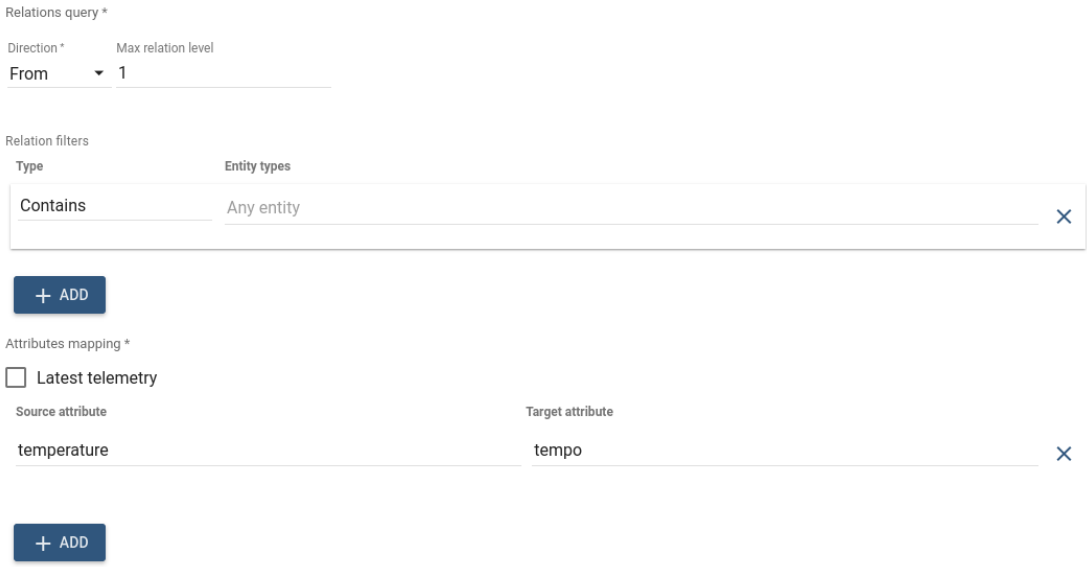
4.4.6 Узел атрибутов связей



Узел находит связанный объект объекта отправителя сообщения с помощью настроенного запроса и добавляет атрибуты или значение последней телеметрии в метаданные сообщения.

Администратор может настроить сопоставление между исходным именем атрибута и именем атрибута метаданных. В конфигурации «Запрос отношений» Администратор может выбрать необходимое Направление и уровень глубины отношений. Также можно настроить набор фильтров отношений с требуемым типом отношения и типами сущностей.

В конфигурации узла есть флажок `Последняя телеметрия`. Если этот флажок установлен, `Node` будет получать последние данные телеметрии для настроенных ключей. В противном случае `Node` будет получать атрибуты области действия сервера.



Если обнаружено несколько связанных объектов, то для добавления атрибутов используется только первый объект, остальные объекты отбрасываются.

Если связанный объект не найден, используется цепочка сбоев, в противном случае — цепочка успехов.

Метаданные исходящего сообщения будут содержать настроенные атрибуты, если они существуют.

Чтобы получить доступ к извлеченным атрибутам в других узлах, вы можете использовать шаблон `metadata.tempo`.

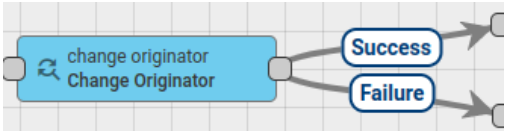
Вы можете использовать `${metadataKey}` для значения из метаданных, `$[messageKey]` для значения из тела сообщения.

Пример этой функции вы можете увидеть в описании к узлу Атрибуты клиента. TODO ссылка

4.5 Узлы трансформации

Узлы преобразования используются для изменения полей входящего сообщения, таких как `отправитель`, `тип сообщения`, `полезная нагрузка` и `метаданные`.

4.5.1 Узел изменения отправителя



Все входящие сообщения имеют поле отправителя, которое идентифицирует объект, отправляющий сообщение. Это может быть `Устройство`, `Ресурс`, `Клиент`, `Арендатор` и т.д.

Этот узел используется в тех случаях, когда отправленное сообщение должно быть обработано как сообщение от другого объекта. Например, Устройство отправляет данные телеметрии, а данные телеметрии следует копировать в Ресурс более высокого уровня или Клиенту. В этом случае администратор должен добавить этот узел перед узлом сохранения временных рядов.

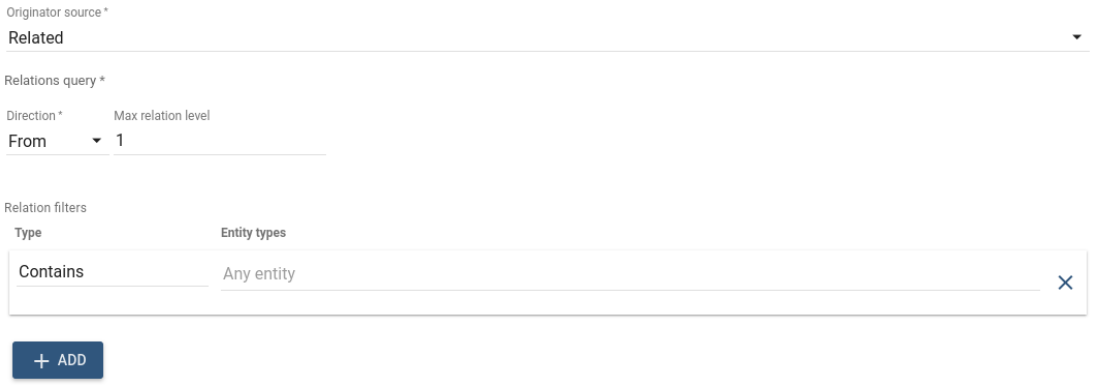
Автор может быть изменен на:

- Клиента
- Арендатора
- Связанный объект, идентифицируемый «проверкой связей»

В конфигурации `Проверка связи` Администратор может выбрать необходимое Направление и уровень глубины отношений. Также можно настроить набор фильтров отношений с требуемым типом отношения и типами сущностей.

Давайте рассмотрим поведение узла правила на примере. Предположим следующую конфигурацию:

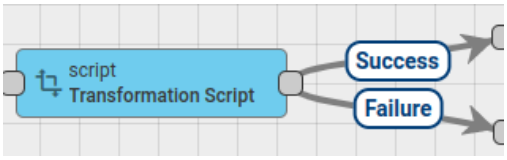
Если обнаружено несколько связанных объектов, только первый объект используется в качестве нового отправителя, другие объекты отбрасываются.



Цепочка отказов используется, если не найдено ни одной связанной сущности/клиента/арендатора, в противном случае - цепочка Success.

Исходящее сообщение будет иметь новый идентификатор отправителя.

4.5.2 Узел скрипт



Изменяет полезную нагрузку сообщения, метаданные или тип сообщения, используя настроенную функцию JavaScript.

Функция JavaScript получает 3 входных параметра:

- msg — полезная нагрузка сообщения.
- метаданные - это метаданные сообщения.
- msgType - тип сообщения.

```
{
      msg: new payload,
      metadata: new metadata,
      msgType: new msgType
}
```

Все поля результирующего объекта являются необязательными и будут взяты из исходного сообщения, если они не указаны.

Исходящее сообщение от этого узла будет новым сообщением, созданным с использованием настроенной функции JavaScript.

Функцию преобразования JavaScript можно проверить с помощью функции Test JavaScript.

Например, Узел получает сообщение с полезной нагрузкой:


```
{
      "temperature": 22.4,
      "humidity": 78
}
```

Оригинальные метаданные:

```
{
      "sensorType" : "temperature"
}
```

Исходный тип сообщения — `POST_TELEMETRY_REQUEST`

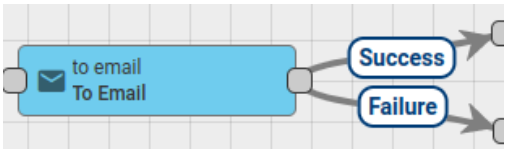
Должны быть выполнены следующие модификации:

- изменить тип сообщения на `CUSTOM_UPDATE`
- добавить дополнительную версию атрибута в полезную нагрузку со значением `v1.1`
- изменить значение атрибута `sensorType` в метаданных на `roomTemp`

Следующая функция преобразования выполнит все необходимые модификации:

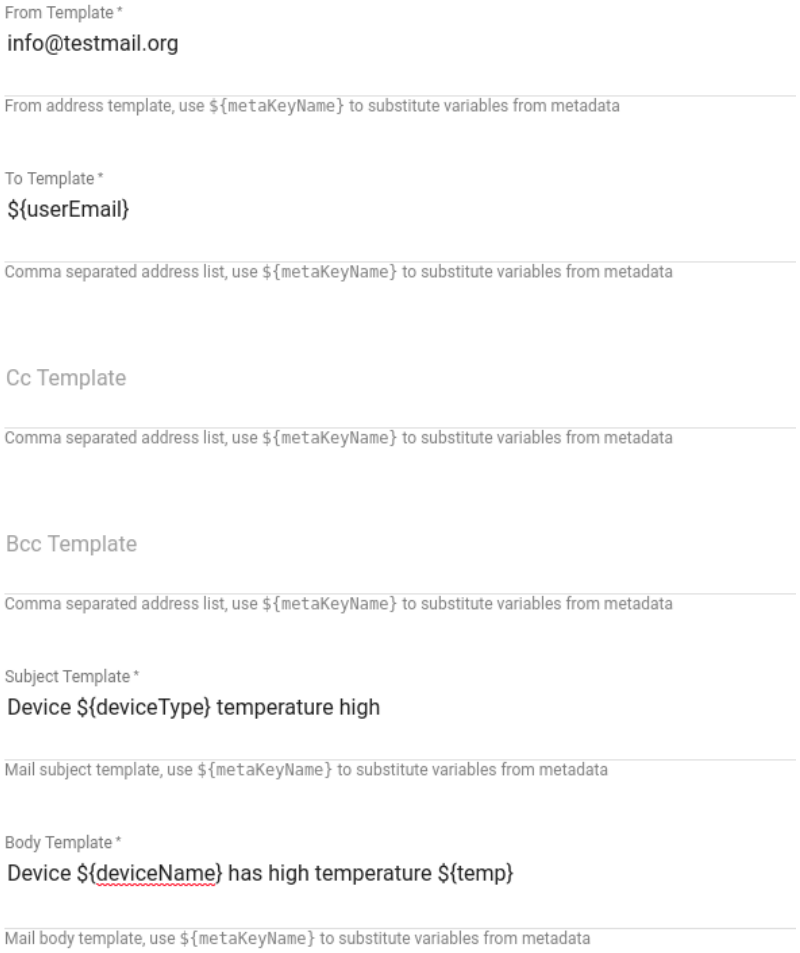
```
var newType = "CUSTOM_UPDATE";
msg.version = "v1.1";
metadata.sensorType = "roomTemp";
return {msg: msg, metadata: metadata, msgType: newType};
```

4.5.3 Узел подготовки сообщения электронной почты



Преобразует сообщение в сообщение электронной почты, заполняя поля электронной почты значениями, полученными из метаданных сообщения.

Установите тип выходного сообщения `SEND_EMAIL`, который может быть принят позже узлом отправки электронной почты. Все поля электронной почты можно настроить для использования значений из метаданных.



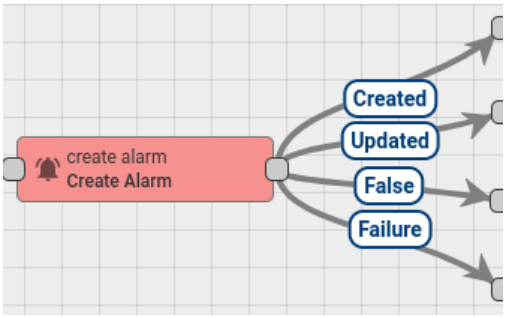
Например, входящее сообщение имеет поле `deviceName` в метаданных, а тело письма должно содержать его значение. В этом случае на значение `deviceName` можно ссылаться как `${deviceName}`, как в следующем примере:

```
Device ${deviceName} has high temperature
```

Кроме того, этот узел может подготавливать вложения электронной почты, если метаданные входящего сообщения содержат поле вложений со ссылкой на файлы, хранящиеся в базе данных.

4.6 Узлы действий

4.6.1 Узел Тревога (Alarm)



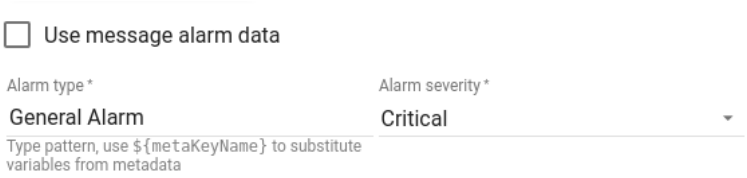
Этот узел пытается загрузить последнюю тревогу с настроенным **типом** тревоги для отправителя сообщения. Если существует **неснятая** тревога, то эта тревога будет обновлена, в противном случае будет создана новая тревога.

Конфигурация узла:

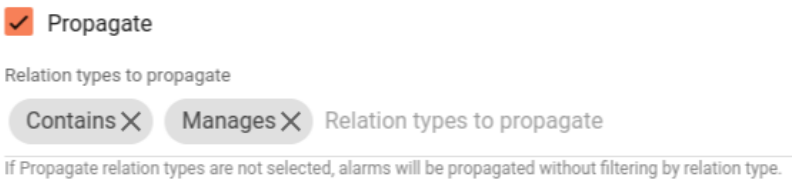
- `Alarm Details Builder` - скрипт
- `Alarm Type` - любая строка, представляющая тип тревоги
- `Alarm Severity` - серьезность тревоги. Может принимать значения `{CRITICAL | MAJOR | MINOR | WARNING | INDETERMINATE}`
- `Propagate` - следует ли распространять тревогу на все связанные родительские объекты.

Узел тревоги имеет возможность:

- прочитать конфигурацию тревоги из сообщения
- получить тип тревоги, используя шаблон с полями из метаданных сообщения:

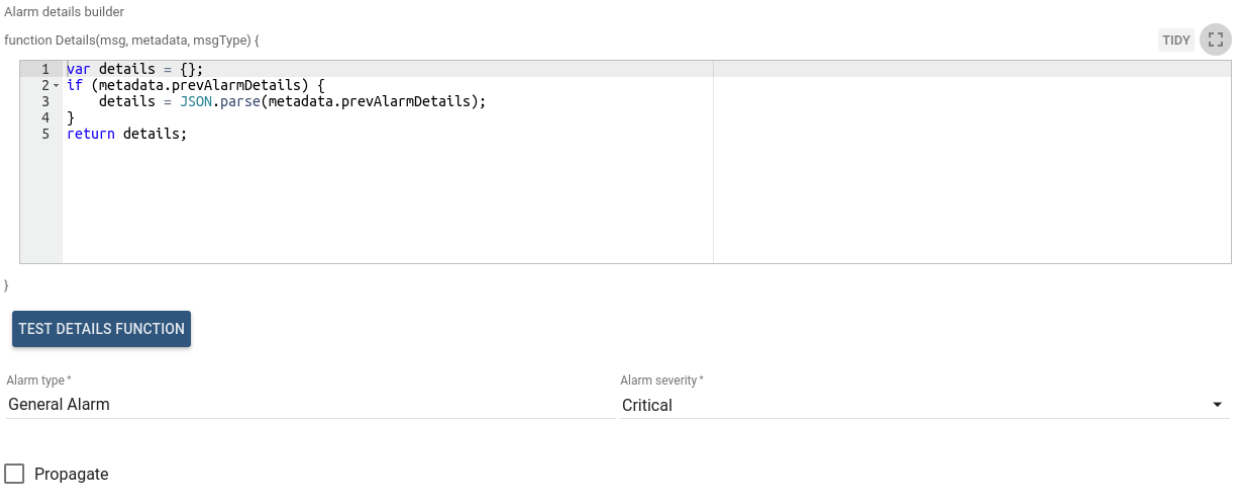


- распространение фильтрации на родительские объекты по типам отношений:



`Alarm Details Builder` скрипт, используемый для создания *JsonNode* сведений об аварийных сигналах. Это полезно для хранения дополнительных параметров внутри *Alarm*. Например, вы можете сохранить пару имя и данные из исходного сообщения или метаданных.

`Alarm Details Builder` script должен возвращать объект *сведений*.



Доступ к данным сообщения можно получить через свойство `msg`. Например, `msg.temperature`.

Доступ к метаданным сообщения можно получить через свойство `metadata`. Например, `metadata.customerName`.

Доступ к типу сообщения можно получить через свойство `msgType`, например `msgType`.

ОПЦИОНАЛЬНО: предыдущие сведения о тревоге можно получить через `metadata.prevAlarmDetails`. Если предыдущего сигнала тревоги не существует, то этого поля не будет.

ОБРАТИТЕ ВНИМАНИЕ: `metadata.prevAlarmDetails` — это необработанное строковое поле, и его необходимо преобразовать в объект, используя следующую конструкцию:

```
var details = {};
if (metadata.prevAlarmDetails) {
    details = JSON.parse(metadata.prevAlarmDetails);
}
```

Функцию сценария `Alarm Details Builder` можно проверить с помощью функции *Test JavaScript*.

Пример функции конструктора деталей

Эта функция берет свойство счетчика из предыдущего сигнала тревоги и увеличивает его. Также помещает атрибут температуры из полезной нагрузки входящего сообщения в сведения о тревоге.

```
var details = {temperature: msg.temperature, count: 1};

if (metadata.prevAlarmDetails) {
    var prevDetails = JSON.parse(metadata.prevAlarmDetails);
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    if(prevDetails.count) {
        details.count = prevDetails.count + 1;
    }
}

return details;
```

Тревога создана/обновлена с такими свойствами:

- `Alarm details` — объект, возвращаемый из сценария *Alarm Details Builder*.
- `Alarm status` - если новая тревога -> ACTIVE_UNACK. Если тревога имеется -> не меняется
- `Severity` — значение из конфигурации узла
- `Propagation` - значение из конфигурации узла
- `Alarm type` - значение из конфигурации узла
- `Alarm start time` - если новый будильник -> текущее системное время. Если тревога есть -> не меняется
- `Alarm end time` - текущее системное время

Исходящее сообщение будет иметь следующую структуру:

- `Message Type` - ALARM
- `Originator` - тот же отправитель из входящего сообщения
- `Payload` - JSON-представление новой тревоги, которая была создана/обновлена
- `Metadata` - все поля из исходных метаданных сообщения

После создания новой Тревоги Исходящее сообщение будет содержать дополнительное свойство внутри Метаданных - *isNewAlarm* со значением *true*. Сообщение будет передано через созданную цепочку.

После обновления существующего сигнала тревоги исходящее сообщение будет содержать дополнительное свойство внутри метаданных — *isExistingAlarm* со значением *true*. Сообщение будет передано через обновленную цепочку.

Вот пример полезной нагрузки исходящего сообщения:

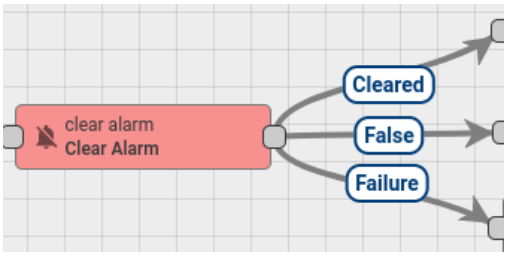
```
{
    "tenantId": {
        "entityType": "TENANT",
        "id": "22cd8888-5dac-11e8-bbab-ad47060c9bbb"
    },
    "type": "High Temperature Alarm",
    "originator": {
        "entityType": "DEVICE",
```

(continues on next page)

(продолжение с предыдущей страницы)


```
        "id": "11cd8777-5dac-11e8-bbab-ad55560c9ccc"
    },
    "severity": "CRITICAL",
    "status": "ACTIVE_UNACK",
    "startTs": 1526985698000,
    "endTs": 1526985698000,
    "ackTs": 0,
    "clearTs": 0,
    "details": {
        "temperature": 70,
        "ts": 1526985696000
    },
    "propagate": true,
    "id": "33cd8999-5dac-11e8-bbab-ad47060c9431",
    "createdTime": 1526985698000,
    "name": "High Temperature Alarm"
}
```


4.6.2 Узел снятия тревоги



Этот узел загружает последнюю тревогу с настроенным типом тревоги для отправителя сообщения и сбрасывает тревогу, если она существует.

Конфигурация узла:

- `Alarm Details Builder` - скрипт
- `Alarm Type` - любая строка, представляющая тип тревоги

Узел правил имеет возможность получить тип тревоги, используя шаблон с полями из метаданных сообщения:

`Alarm Details Builder` скрипт, используемый для обновления сведений о тревоге `JsonNode`. Это полезно для хранения дополнительных параметров внутри `Alarm`. Например, вы можете сохранить пару имя/значение атрибута из полезной нагрузки исходного сообщения или метаданных.

Alarm Details Builder скрипт должен возвращать объект details.

- Доступ к данным сообщения можно получить через свойство msg. Например, msg.temperature
- Доступ к метаданным сообщения можно получить через свойство metadata. Например, metadata.customerName
- Доступ к типу сообщения можно получить через свойство msgType. Например msgType
- Доступ к текущим сведениям о тревоге можно получить через metadata.prevAlarmDetails.

ОБРАТИТЕ ВНИМАНИЕ: metadata.prevAlarmDetails — это необработанное строковое поле, и его необходимо преобразовать в объект, используя следующую конструкцию:

```
var details = {};
if (metadata.prevAlarmDetails) {
    details = JSON.parse(metadata.prevAlarmDetails);
}
```

Функцию сценария Alarm Details Builder можно проверить с помощью функции *Test JavaScript*.

Пример функции конструктора деталей

Эта функция берет свойство счетчика из предыдущего сигнала тревоги и увеличивает его. Также помещает атрибут температуры из полезной нагрузки входящего сообщения в сведения о тревоге.

```
var details = {temperature: msg.temperature, count: 1};

if (metadata.prevAlarmDetails) {
    var prevDetails = JSON.parse(metadata.prevAlarmDetails);
    if(prevDetails.count) {
        details.count = prevDetails.count + 1;
    }
}

return details;
```

Этот узел обновляет текущий сигнал тревоги:

- изменяет статус тревоги `status` на `CLEARED_ACK`, если она уже была подтверждена, иначе на `CLEARED_UNACK`
- устанавливает `clear time` на текущее системное время
- обновляет сведения об аварийных сигналах с помощью нового объекта, возвращенного из сценария `Alarm Details Builder`.

В случае, когда Тревога не существует или это уже Сброшенная Тревога, исходное Сообщение будет передано следующим узлам через цепочку `False`.

В противном случае новое сообщение будет передано по цепочке `Cleared`.

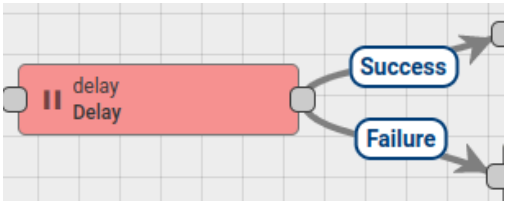
Сообщение на выходе будет иметь следующую структуру:

- `Message Type` - ALARM
- `Originator` - тот же отправитель из входящего сообщения
- `Payload` - JSON-представление тревоги, которая была сброшена
- `Metadata` - все поля из исходных метаданных сообщения. Также будет добавлено дополнительное свойство внутри метаданных - *isClearedAlarm* со значением *true*.

Вот пример полезной нагрузки исходящего сообщения:

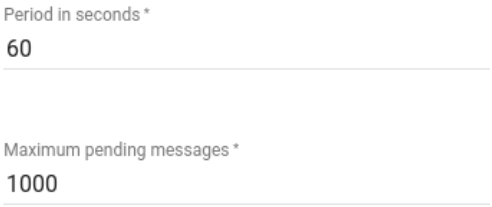
```
{
    "tenantId": {
        "entityType": "TENANT",
        "id": "22cd8888-5dac-11e8-bbab-ad47060c9bbb"
    },
    "type": "High Temperature Alarm",
    "originator": {
        "entityType": "DEVICE",
        "id": "11cd8777-5dac-11e8-bbab-ad55560c9ccc"
    },
    "severity": "CRITICAL",
    "status": "CLEARED_UNACK",
    "startTs": 1526985698000,
    "endTs": 1526985698000,
    "ackTs": 0,
    "clearTs": 1526985712000,
    "details": {
        "temperature": 70,
        "ts": 1526985696000
    },
    "propagate": true,
    "id": "33cd8999-5dac-11e8-bbab-ad47060c9431",
    "createdTime": 1526985698000,
    "name": "High Temperature Alarm"
}
```

4.6.3 Узел задержки (Delay)



Задерживает входящие сообщения на настраиваемый период.

Конфигурация:

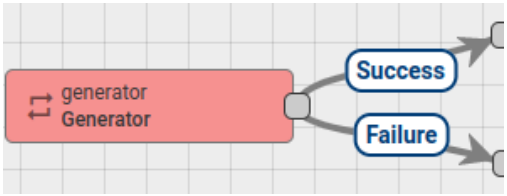


- `Period in seconds` - указывает значение периода, в течение которого входящее сообщение должно быть приостановлено
- `Maximum pending messages` - указывает максимально допустимое количество ожидающих сообщений (очередь приостановленных сообщений)

Когда период задержки для определенного входящего сообщения будет достигнут, оно будет удалено из очереди ожидания и перенаправлено на следующие узлы через цепочку `Success`.

Каждое следующее сообщение будет направляться через цепочку `Failure`, если будет достигнуто максимальное количество ожидающих сообщений.

4.6.4 Узел генератор (Generator)



Генерирует сообщения с настраиваемым периодом. Функция JavaScript используется для генерации сообщений.

Конфигурация узла:

- Частота генерации сообщений в секундах
- Автор сообщения

- Функция JavaScript, которая будет генерировать фактическое сообщение

Функция JavaScript получает 3 входных параметра:

- `prevMsg` - представляет собой предварительно сгенерированную полезную нагрузку сообщения.
- `prevMetadata` - представляет собой ранее сгенерированные метаданные сообщения.
- `prevMsgType` - является ранее сгенерированным типом сообщения.

Скрипт должен вернуть следующую структуру:

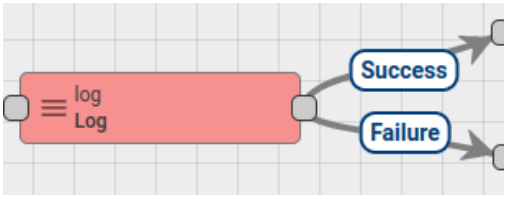
```
{
    msg: new payload,
    metadata: new metadata,
    msgType: new msgType
}
```

Все поля результирующего объекта являются необязательными и будут взяты из ранее сгенерированного сообщения, если они не указаны.

Исходящее сообщение от этого узла будет новым сообщением, созданным с использованием настроенной функции JavaScript.

Этот узел можно использовать для отладки цепочки правил.

4.6.5 Узел логирования (Log Node)



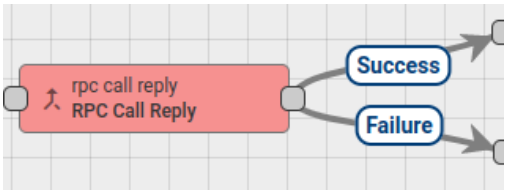
Преобразует входящее сообщение с помощью настроенной функции JavaScript в строку и записывает окончательное значение в файл журнала. Уровень журнала `INFO` используется для ведения журнала. Функция JavaScript получает 3 входных параметра:

- `metadata` - это метаданные сообщения
- `msg` - это полезная нагрузка сообщения
- `msgType` - это тип сообщения

Скрипт должен возвращать строковое значение.

Функцию преобразования JavaScript можно проверить с помощью функции *Test JavaScript*.

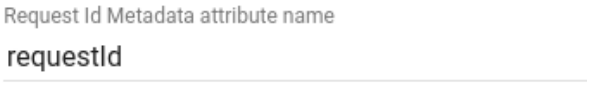
4.6.6 Узел ответа на вызов RPC



Отправляет ответ инициатору вызова RPC. Все входящие запросы RPC проходят через цепочку правил как сообщения. Также все запросы RPC имеют поле идентификатора запроса. Он используется для сопоставления запросов и ответов. Отправитель сообщения должен быть объектом устройства, поскольку ответ RPC инициируется отправителю сообщения.

Конфигурация узла имеет специальное сопоставление поля идентификатора запроса. Если сопоставление не указано, поле метаданных `requestId` используется по умолчанию.

Запрос `RPC` можно получить через разные транспорты:



- MQTT
- HTTP
- CoAP

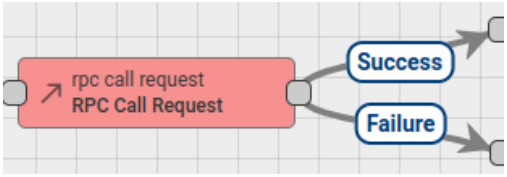
Пример полезной нагрузки сообщения:

```
{
    "method": "setGpio",
    "params": {
        "pin": "23",
        "value": 1
    }
}
```

Сообщение будет направлено через цепочку Failure в следующих случаях:

- Отправитель входящего сообщения не является объектом устройства
- Идентификатор запроса отсутствует в метаданных сообщения
- Полезная нагрузка входящего сообщения пуста

4.6.7 Узел запроса вызова RPC



Отправляет запросы RPC на Устройство и маршрутизирует ответы на следующие узлы Правил. Отправитель сообщения должен быть объектом устройства, поскольку запрос RPC может быть инициирован только для устройства.

В конфигурации узла есть поле Timeout, используемое для указания времени ожидания ответа от устройства.

Полезная нагрузка сообщения должна иметь правильный формат для запроса RPC. Он должен содержать поля метода и параметров, например:

```
{
    "method": "setGpio",
    "params": {
        "pin": "23",
        "value": 1
    }
}
```

Если `Message Payload` содержит поле `requestId`, его значение используется для идентификации RPC-запроса к Устройству. В противном случае будет сгенерирован случайный идентификатор запроса.

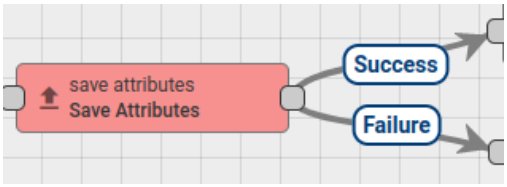
Исходящее сообщение будет иметь того же отправителя и метаданные, что и входящее сообщение. Ответ от устройства будет добавлен в полезную нагрузку сообщения.

Сообщение будет направлено через цепочку `Failure` в следующих случаях:

- Отправитель входящего сообщения не является объектом устройства
- Во входящем сообщении пропущены поля метода или параметров
- Если Node не получит ответ в течение настроенного тайм-аута

В противном случае сообщение будет направлено через цепочку `Success`.

4.6.8 Узел сохранения атрибутов



Сохраняет атрибуты полезной нагрузки входящего сообщения в базу данных и связывает их с сущностью, которая идентифицируется отправителем сообщения. Настроенная область используется для определения области действия атрибутов.

Поддерживаемые типы областей:

- Атрибуты клиента
- Общие атрибуты
- Атрибуты сервера

Ожидает сообщения с типом сообщения POST_ATTRIBUTES_REQUEST. Если тип сообщения не POST_ATTRIBUTES_REQUEST, сообщение будет маршрутизироваться через цепочку Failure.

Когда атрибуты загружаются через существующий API (HTTP / MQTT / CoAP и т.д.), сообщение с правильной полезной нагрузкой и типом будет передано во входной узел корневой цепочки правил.

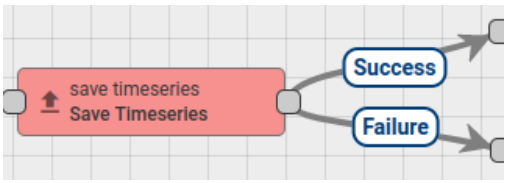
В тех случаях, когда требуется инициировать сохранение атрибутов внутри цепочки правил, цепочка правил должна быть настроена на преобразование полезной нагрузки сообщения в ожидаемый формат и установить тип сообщения POST_ATTRIBUTES_REQUEST. Это можно сделать с помощью Script Transformation Node.

Пример ожидаемой полезной нагрузки сообщения:

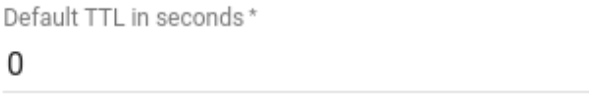
```
{
    "firmware_version": "1.0.1",
    "serial_number": "SN-001"
}
```

После успешного сохранения атрибутов исходное Сообщение будет передано следующим узлам по цепочке Success, в противном случае используется цепочка Failure.

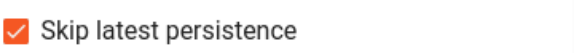
4.6.9 Узел сохранения временных рядов



Сохраняет данные временных рядов из полезной нагрузки входящего сообщения в базу данных и связывает их с сущностью, которая идентифицируется отправителем сообщения. Настроенные секунды TTL используются для истечения срока действия данных временных рядов. Значение 0 означает, что срок действия данных никогда не истечет.



Кроме того, вы можете отключить обновление значений для входящих ключей для последних данных временных рядов (таблица ts_kv_latest), если для флага Пропустить последнее сохранение установлено значение true. Это может быть полезно для высоконагруженных вариантов использования, чтобы уменьшить нагрузку на базу данных.



Обратите внимание, что эту функцию можно включить, если вариант использования не требует расширенной фильтрации на информационных панелях.

Для получения последнего значения сохраненные данные могут быть получены с ограничением 1 и порядком DESC.

Ожидает сообщения с типом сообщения POST_TELEMETRY_REQUEST. Если тип сообщения не POST_TELEMETRY_REQUEST, сообщение будет маршрутизироваться через цепочку отказов.

Когда данные временных рядов публикуются через существующий API (HTTP / MQTT / CoAP и т.д.), сообщение с правильной полезной нагрузкой и типом будет передано во входной узел корневой цепочки правил.

В тех случаях, когда требуется инициировать сохранение данных временных рядов внутри цепочки правил, цепочка правил должна быть настроена на преобразование полезной нагрузки сообщения в ожидаемый формат и установите тип сообщения POST_TELEMETRY_REQUEST. Это можно сделать с помощью Script Transformation Node.

Метаданные сообщения должны содержать поле ts. В этом поле указывается временная метка опубликованных данных телеметрии в миллисекундах.

Кроме того, если метаданные сообщения содержат поле TTL, его значение используется для истечения срока действия данных таймсерии, в противном случае используется TTL из конфигурации узла.

Вы можете включить параметр useServerTs, чтобы использовать метку времени обработки сообщения вместо метки времени из сообщения. Полезно для всех видов последовательной обработки, если вы объединяете сообщения из нескольких источников (устройств, ресурсов и т.д.).

В случае последовательной обработки платформа гарантирует, что сообщения обрабатываются в порядке их поступления в очередь. Однако метка времени сообщений, созданных несколькими устройствами/серверами, может быть несинхронизирована задолго до того, как они будут помещены в очередь. Уровень БД имеет определенные оптимизации для игнорирования обновлений таблиц «атрибутов» и «последних значений», если новая запись имеет метку времени, которая старше, чем предыдущая запись.

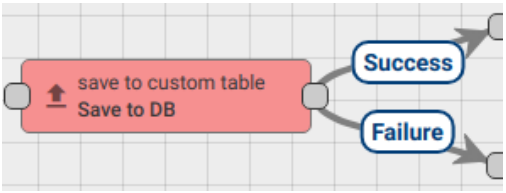
Итак, чтобы убедиться, что все сообщения будут обработаны корректно, следует включить этот параметр для сценариев последовательной обработки сообщений.

Пример ожидаемой полезной нагрузки сообщения:

```
{
    "values": {
        "key1": "value1",
        "key2": "value2"
    }
}
```

После успешного сохранения данных временных рядов исходное сообщение будет передано следующим узлам по цепочке Success, в противном случае используется цепочка Failure.

4.6.10 Узел «Сохранить в базу данных»

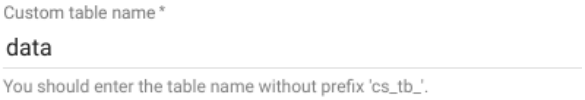


Узел хранит данные из полезной нагрузке входящего сообщения в базу данных `Cassandra` в предопределенной настраиваемой таблице, которая должна иметь префикс `cs_tb_`, чтобы избежать вставки данных в общие таблицы ТВ.

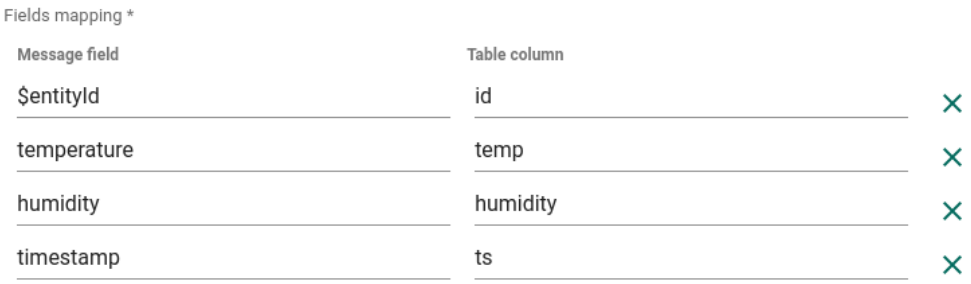
Обратите внимание, что этот узел правила можно использовать только для `Cassandra DB`.

Конфигурация:

Администратор должен задать имя пользовательской таблицы без префикса: `cs_tb_`:



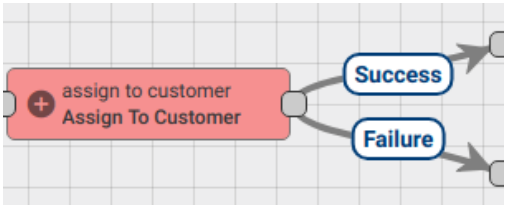
Администратор может настроить сопоставление между именами полей сообщения и именами столбцов таблицы. Если ключом сопоставления является `$entityIdpo`, который идентифицируется отправителем сообщения, то в соответствующее имя столбца (значение сопоставления) будет записан идентификатор отправителя сообщения.



Если указанное поле сообщения не существует в данных сообщения или не является примитивом JSON, исходящее сообщение будет маршрутизироваться через цепочку `Failure`, в противном случае сообщение будет маршрутизироваться через цепочку `Success`.

Примечание: Убедитесь, что вы не используете ключи метаданных в конфигурации — возможны только ключи данных.

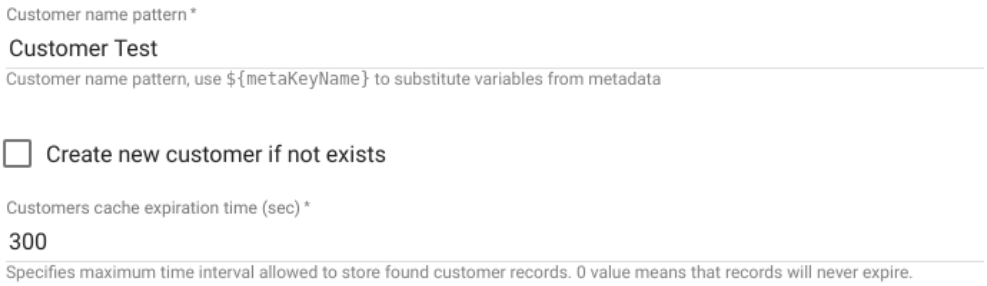
4.6.11 Узел «Назначить узлу клиента»



Назначить объект отправителя сообщения клиенту.

Допускаются следующие типы отправителей сообщений: `Asset`, `Device`, `Entity View`, `Dashboard`.

Находит целевого клиента по шаблону имени клиента, а затем назначает этому клиенту сущность-инициатора. Можно создать нового клиента, если он не существует. Для этого параметр `Создать нового клиента, если он не существует` должен быть установлен в `true`



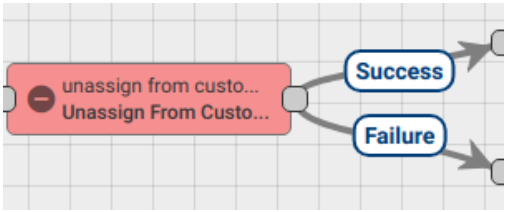
- `Customer name pattern` — можно задать прямое имя клиента или можно использовать шаблон, который будет преобразован в реальное имя клиента с использованием метаданных сообщения.
- `Create new customer if not exists` — если флажок установлен, будет создан новый клиент, если он не существует.
- `Customers cache expiration time` — указывает максимальный интервал времени в секундах, разрешенный для хранения записей о найденных клиентах. Значение 0 означает, что срок действия записей никогда не истечет.

Сообщение будет направлено через цепочку `Failure` в следующих случаях:

- Когда тип сущности отправителя не поддерживается.
- Целевой клиент не существует, и флажок `Create new customer if not exists`, не установлен.

В других случаях сообщение будет маршрутизироваться через цепочку `Success`.

4.6.12 Узел «Снять назначение узла от клиента»

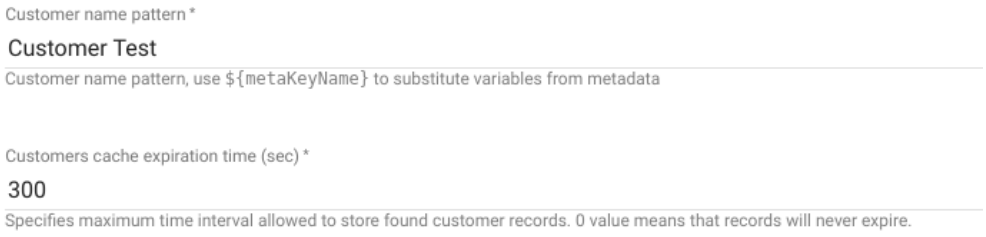


Отменить назначение объекта отправителя сообщения от клиента.

Допускаются следующие типы отправителей сообщений: `Asset`, `Device`, `Entity View`, `Dashboard`.

Находит целевого клиента по шаблону имени клиента, а затем отменяет назначение объекта-инициатора для этого клиента.

Конфигурация:



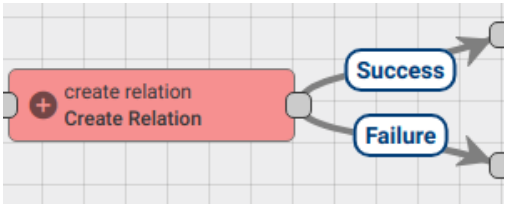
- `Customer name pattern` — можно задать прямое имя клиента или можно использовать шаблон, который будет преобразован в реальное имя клиента с использованием метаданных сообщения.
- `Customers cache expiration time` — указывает максимальный интервал времени в секундах, разрешенный для хранения записей о найденных клиентах. Значение `0` означает, что срок действия записей никогда не истечет.

Сообщение будет направлено через цепочку `Failure` в следующих случаях:

- Когда тип сущности отправителя не поддерживается.
- Целевой клиент не существует.

В других случаях сообщение будет маршрутизироваться через цепочку `Success`.

4.6.13 Создать узел связи



Создайте отношение от выбранной сущности к отправителю сообщения по типу и направлению.

Допускаются следующие типы отправителей сообщений: `Asset`, `Device`, `Entity View`, `Customer`, `Tenant`, `Dashboar`.

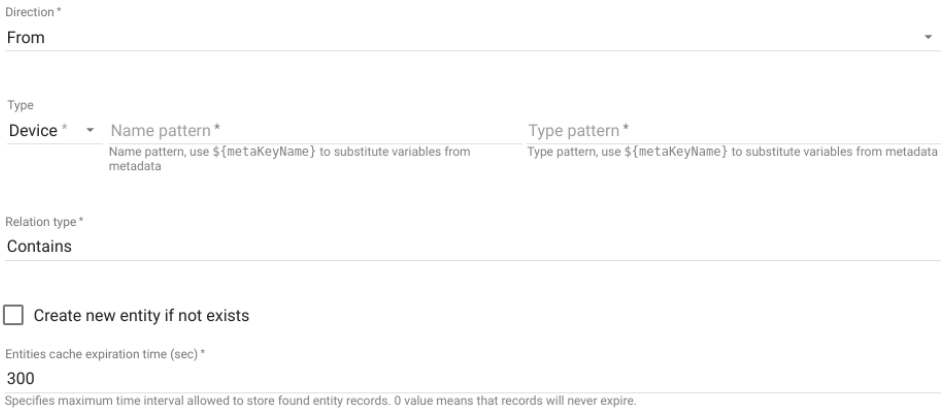
Находит целевую сущность по шаблонам ключей метаданных, а затем создает связь между исходной сущностью и целевой сущностью. Если выбран тип объекта `Asset`, `Device` или `Customer` создаст новый объект, если он не существует, и установлен флажок: `Create new Entity if not exists`.

Если выбран тип объекта `Актив` или `Устройство`, вам необходимо установить два шаблона:

- шаблон имени сущности;
- шаблон типа сущности.

В противном случае следует установить только шаблон имени.

Конфигурация:



- `Direction` - разрешены следующие типы: `From`, `To`.
- `Relation type` - тип направленных подключений к объекту отправителя сообщения. Типы по умолчанию `Contains` и `Manages` можно выбрать из раскрывающегося списка.
- `Name pattern and Type pattern` - можно установить прямое имя/тип объекта или можно использовать шаблон, который будет разрешен к реальному имени/типу объекта с использованием метаданных сообщения.

- `Entities cache expiration time` — указывает максимальный интервал времени в секундах, разрешенный для хранения найденных записей целевых сущностей. Значение 0 означает, что срок действия записей никогда не истечет.

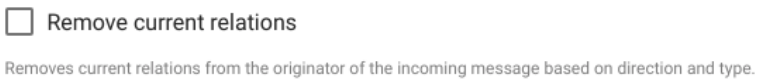
Сообщение будет направлено через цепочку `Failure` в следующих случаях:

- Когда тип сущности отправителя не поддерживается.
- Целевой объект не существует.

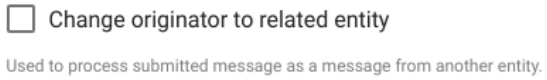
В других случаях сообщение будет маршрутизироваться через цепочку `Success`.

Имеется возможность:

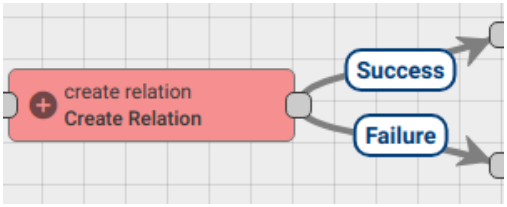
- удалить текущие отношения с отправителем входящего сообщения в зависимости от направления и типа:



- изменить отправителя входящего сообщения на выбранный объект и обрабатывать исходящие сообщения как сообщения от другого объекта:



4.6.14 Удалить узел связи



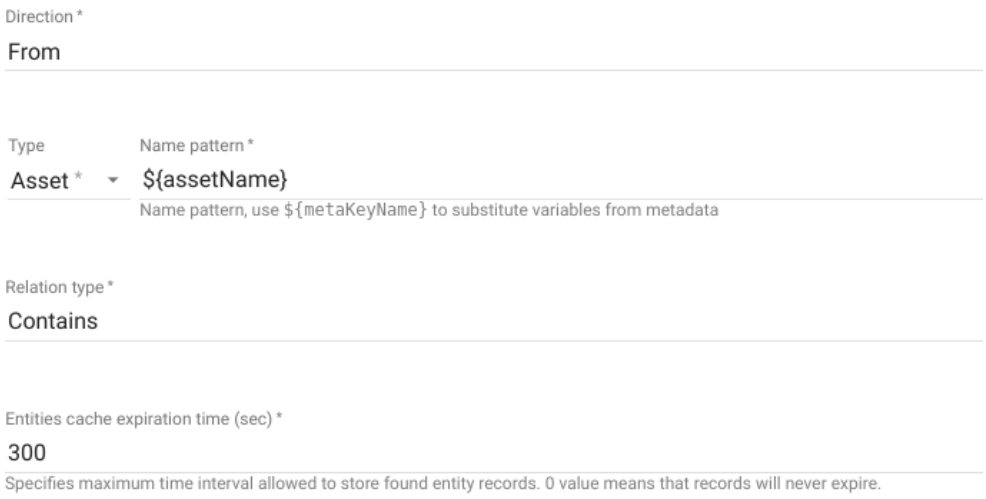
Удалить связь выбранного объекта с отправителем сообщения по типу и направлению.

Допускаются следующие типы отправителей сообщений: `Asset`, `Device`, `Entity View`, `Customer`, `Tenant`, `Dashboard`.

Находит целевую сущность по шаблону имени сущности, а затем удаляет связь между исходной сущностью и этой сущностью.

Конфигурация:

- `Direction` - разрешены следующие типы: `From`, `To`.
- `Relation type` - тип направленных подключений к объекту отправителя сообщения. Типы по умолчанию `Contains` и `Manages` можно выбрать из раскрывающегося списка.
- `Name pattern` - можно задать прямое имя объекта или можно использовать шаблон, который будет преобразован в реальное имя объекта с использованием метаданных сообщения.



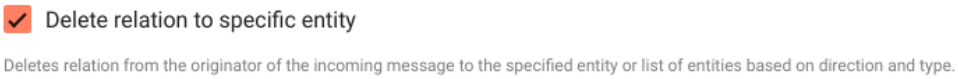
- `Entities cache expiration time` — указывает максимальный интервал времени в секундах, разрешенный для хранения найденных записей целевых сущностей. Значение 0 означает, что срок действия записей никогда не истечет.

Сообщение будет направлено через цепочку отказов в следующих случаях:

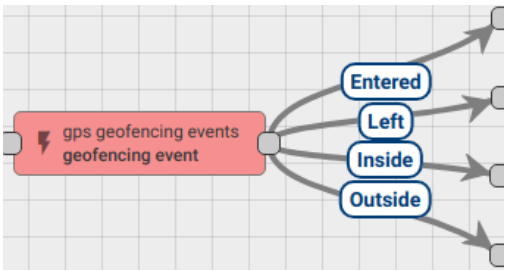
- Когда тип сущности отправителя не поддерживается.
- Целевой объект не существует.

В других случаях сообщение будет маршрутизироваться через цепочку `Success`.

Узел правил имеет возможность удалять отношение отправителя входящего сообщения к указанному объекту или к списку объектов на основе направления и типа, отключив следующий флажок в конфигурации узла правила:

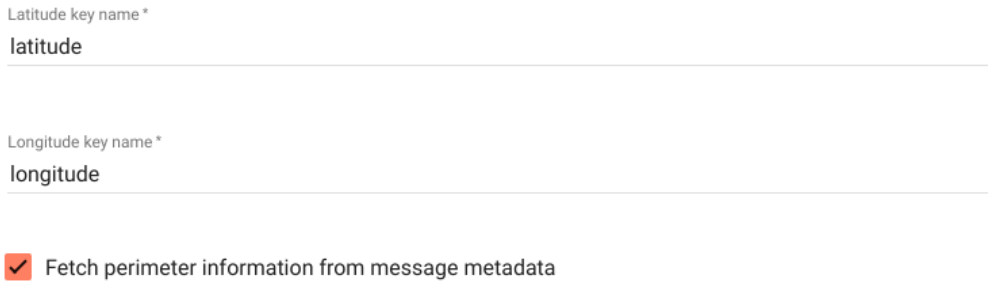


4.6.15 Узел событий GPS-геозоны



Производит входящие сообщения по параметрам на основе GPS. Извлекает широту и долготу из данных или метаданных входящего сообщения и возвращает различные события на основе

параметров конфигурации.



Узел правила по умолчанию извлекает информацию о периметре из метаданных сообщения. Если флажок `Fetch perimeter information from message metadata` не установлен, необходимо настроить дополнительную информацию.

Получение данных из метаданных события:

Существует два варианта определения площади в зависимости от типа периметра:

- `Полигон`

  Метаданные входящего сообщения должны включать ключ с именем периметра и следующую структуру данных:

```
[
    [lat1,lon1],
    [lat2,lon2],
    ... ,
    [latN,lonN]
]
```

- `Окружность`

```
{
  "centerLatitude": "value1",
  "centerLongitude": "value2",
  "range": "value3"
}
```

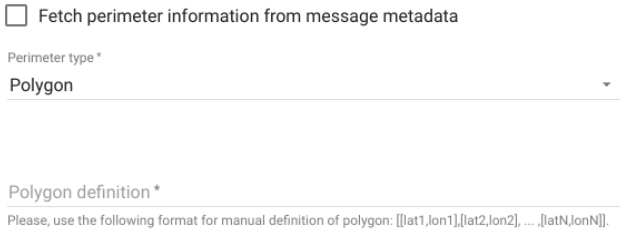
Все значения имеют тип с плавающей точкой двойной точности.

Ключ `rangeUnit` должен содержать одно из значений `METER`, `KILOMETER`, `FOOT`, `MILE`, `NAUTICAL_MILE` (заглавными буквами).

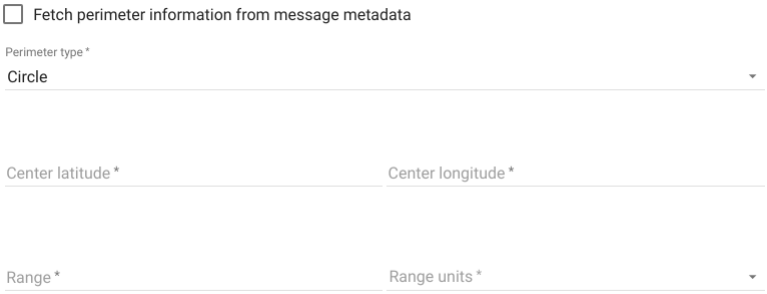
Получение данных из конфигурации узла:

Существует два варианта определения площади в зависимости от типа периметра:

- `Полигон`



- `Окружность`

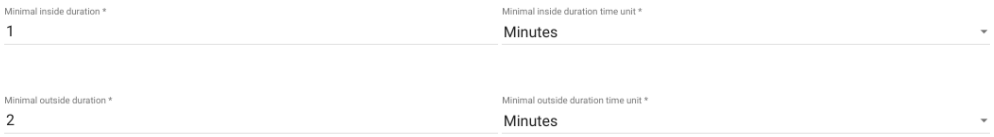


Типы событий:

Существует 4 типа событий, управляемых узлом правила геозоны:

- `Entered` — сообщает, когда широта и долгота из входящего сообщения принадлежат требуемой области периметра в первый раз;
- `Left` — сообщает, когда широта и долгота из входящего сообщения не принадлежат требуемой области периметра в первый раз;
- `Inside` и `Outside` события используются для сообщения о текущем состоянии.

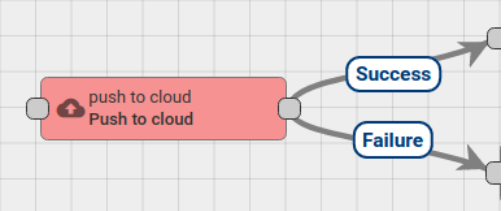
Администратор может настроить пороговое значение продолжительности для сообщения о внутреннем или внешнем событии. Например, если минимальное внутреннее время установлено равным 1 минуте, отправитель сообщения считается находящимся внутри периметра через 60 секунд после входа в зону. Минимальное внешнее время определяет, когда отправитель сообщения также считается находящимся вне периметра.



Цепочка `Failure` будет использоваться, когда:

- входящее сообщение не имеет настроенного ключа широты или долготы в данных или метаданных.
- отсутствует определение периметра;

4.6.16 Узел «Отправить в облако»



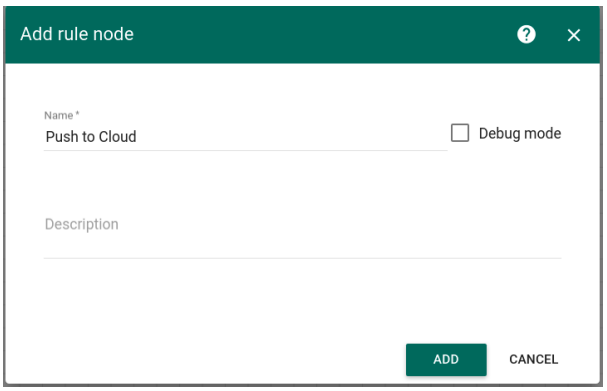
Позволяет отправить сообщения с периферии в облако. Этот узел используется только на периферии для отправки сообщений в облако. Как только сообщение поступит на этот узел, оно будет преобразовано в облачное событие и сохранено в локальной базе данных. Узел не отправляет сообщения напрямую в облако, а сохраняет события в облачной очереди. Поддерживает следующие типы оригинаторов:

- DEVICE
- ASSET
- ENTITY_VIEW
- DASHBOARD
- TENANT
- CUSTOMER
- EDGE

Также узел поддерживает следующие типы сообщений:

- POST_TELEMETRY_REQUEST
- POST_ATTRIBUTES_REQUEST
- ATTRIBUTES_UPDATED
- ATTRIBUTES_DELETED
- ALARM

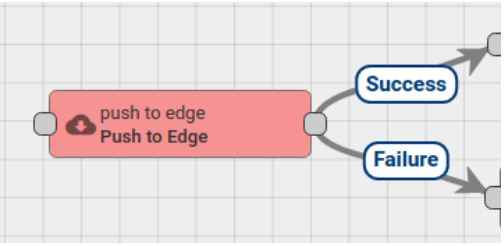
В случае успешного события сохранения в базу данных сообщение будет направлено по маршруту Success.



Сообщение будет направлено через цепочку `Failure` в следующих случаях:

- Узлу не удалось сохранить периферийное событие в базе данных
- Прибыл неподдерживаемый тип отправителя
- Пришло сообщение неподдерживаемого типа

4.6.17 Узел «Отправить в периферию»



Отправляйте сообщения из облака на периферию. Отправитель сообщения должен быть назначен конкретной периферии, или отправителем сообщения является сам объект EDGE. Этот узел используется только в облачных экземплярах для отправки сообщений из облака на периферию. Как только сообщение поступит на этот узел, оно будет преобразовано в периферийное событие и сохранено в базе данных. Узел не отправляет сообщения напрямую в периферию, а сохраняет события в пограничной очереди.

Поддерживает следующие типы оригинаторов:

- `DEVICE`
- `ASSET`
- `ENTITY_VIEW`
- `DASHBOARD`
- `TENANT`
- `CUSTOMER`
- `EDGE`

Также узел поддерживает следующие типы сообщений:

- `POST_TELEMETRY_REQUEST`
- `POST_ATTRIBUTES_REQUEST`
- `ATTRIBUTES_UPDATED`
- `ATTRIBUTES_DELETED`
- `ALARM`

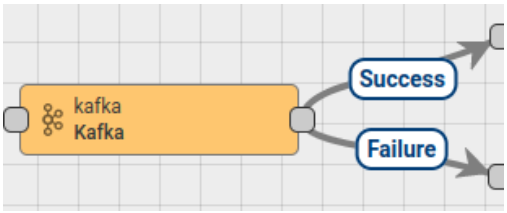
В случае успешного события границы хранилища в базу данных сообщение будет направлено по маршруту `Success`.

Сообщение будет направлено через цепочку `Failure` в следующих случаях:

- Узлу не удалось сохранить периферийное событие в базе данных
- Прибыл неподдерживаемый тип отправителя
- Пришло сообщение неподдерживаемого типа

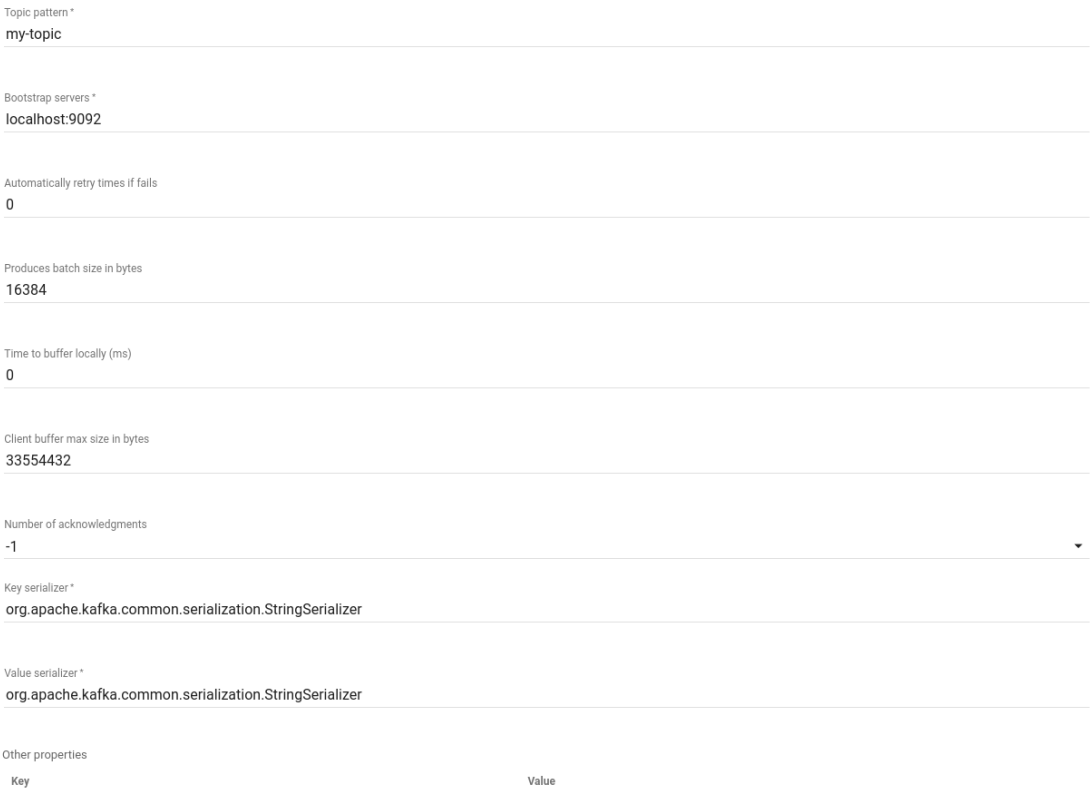
4.7 Узлы публикации

4.7.1 Узел Kafka



Узел Kafka отправляет сообщения брокерам Kafka. Ожидает сообщения любого типа сообщения. Отправит запись через производителя Kafka на сервер Kafka.

Конфигурация:



- `Topic pattern` - может быть статической строкой или шаблоном, который разрешается с помощью свойств метаданных сообщения. Например, `${deviceType}`

- `bootstrap servers` - список брокеров kafka, разделенных запятой.
- `Automatically retry times` - количество попыток повторной отправки сообщения в случае сбоя соединения.
- `Produces batch size` - размер пакета в байтах для группировки сообщений с одним и тем же разделом.
- `Time to buffer locally` - максимальная продолжительность локального окна буферизации в мс.
- `Client buffer max size` - максимальный размер буфера в байтах для отправки сообщений.
- `Number of acknowledgments` - количество подтверждений, которые узел должен получить, прежде чем рассматривать запрос завершенным.
- `Key serializer` - по умолчанию `org.apache.kafka.common.serialization.StringSerializer`
- `Value serializer` - по умолчанию `org.apache.kafka.common.serialization.StringSerializer`
- `Other properties` - любые другие дополнительные свойства могут быть предоставлены для подключения брокера kafka.

`Published body` - узел отправит полную полезную нагрузку сообщения в тему Kafka. При необходимости цепочка правил может быть настроена на использование цепочки узлов преобразования для отправки правильной полезной нагрузки в Kafka.

`Исходящее сообщение` - с этого узла будет содержать свойства смещения ответа, раздела и темы в метаданных сообщения. Полезная нагрузка, тип и отправитель исходного сообщения не будут изменены.

Примечание: если вы хотите использовать `Confluent cloud` в качестве брокера kafka, вам следует добавить следующие свойства:

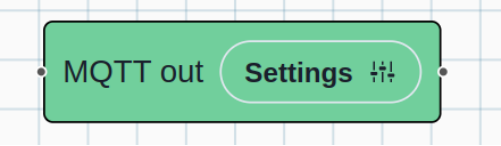
Таблица 1: Таблица параметров

Ключ	Значение
ssl.endpoint.identification.algorithm	https
sasl.mechanism	PLAIN
sasl.jaas.config	PLAIN
org.apache.kafka.common.security.plain.PlainLogin - обязателен, модель: username="CLUSTER_API_KEY", password="CLUSTER_API_SECRET";	SASL_SSL

`CLUSTER_API_KEY` - ваш ключ доступа из настроек кластера.

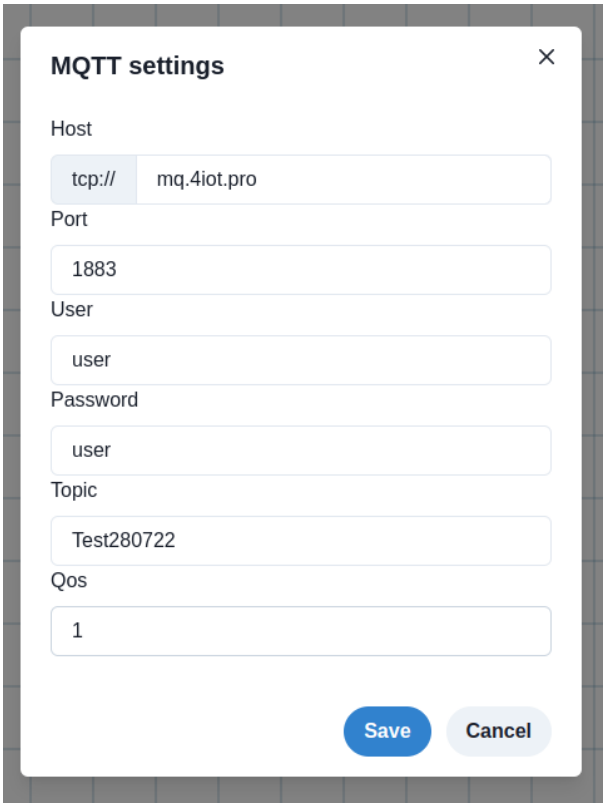
`CLUSTER_API_SECRET` - секрет доступа из настроек кластера.

4.7.2 Узел MQTT



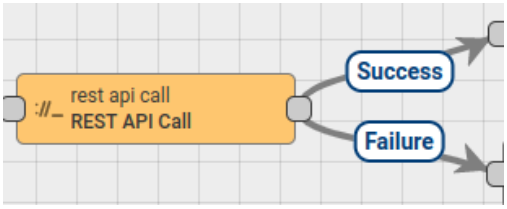
Публикует полезную нагрузку входящего сообщения в топик настроенного MQTT брокера

Конфигурация:



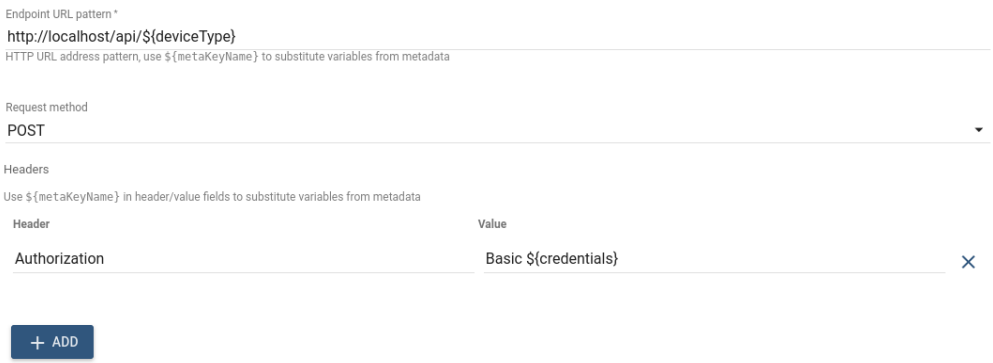
- `Host` - хост MQTT брокера
- `Port` - порт MQTT брокера
- `User` - Логин
- `Pssword` - Пароль
- `Topic` - Имя топика, куда будут отправленны данные
- `Qos` - уровень качества обслуживания

4.7.3 Узел REST API



Вызов REST API к серверу REST

Конфигурация:



- `Endpoint URL pattern` - может быть статической строкой или шаблон, который разрешен с помощью свойств метаданных сообщения. Например `${deviceType}`
- `Request method` - `GET, POST, PUT, DELETE`
- `Headers` - заголовки запроса, заголовок или значение может быть статической строкой или шаблон, который разрешен с помощью свойств метаданных сообщения.

Endpoint URL

URL-адрес может быть статической строкой или шаблоном. Для разрешения шаблонов используются только метаданные сообщений. Таким образом, имена свойств, используемые в шаблонах, должны существовать в метаданных сообщения, иначе необработанный шаблон будет добавлен в URL-адрес.

Например, если полезная нагрузка сообщения содержит свойство `DeviceType` со значением `container`, то этот шаблон:

`<http://localhost/api/${deviceType}/update>`

будет преобразован в

`<http://localhost/api/container/update>`

Headers

Коллекция имени/значения заголовка может быть настроена. Эти заголовки будут добавлены в запрос REST. Шаблон должен использоваться как для имени заголовка, так и для его значения.

Например `${DeviceType}`. Для разрешения шаблонов используются только метаданные сообщений. Таким образом, имена свойств, используемые в шаблоне, должны существовать в метаданных сообщения, иначе необработанный шаблон будет добавлен в заголовок.

Request body

Узел отправит полную полезную нагрузку сообщения в настроенную конечную точку REST. При необходимости Цепочка правил может быть настроена на использование цепочки узлов преобразования для отправки корректной полезной нагрузки.

Исходящее сообщение

Из этого узла будут содержаться заголовки `response status`, `StatusCode`, `statusReason` и `responce` в метаданных сообщения. Полезная нагрузка исходящего сообщения будет такой же, как и тело ответа. Тип исходного сообщения и его составитель не будут изменены.

Чтобы отправить один файл в качестве тела запроса, добавьте поле `вложения` в сообщение `метаданных` с `uuid` файла, хранящегося в базе данных. В этом случае любые данные сообщения будут проигнорированы и будет отправлено только содержимое файла. Чтобы определить тип содержимого запроса, используйте следующий параметр заголовка:

```
Content-Type: application/json; charset=UTF-8
```

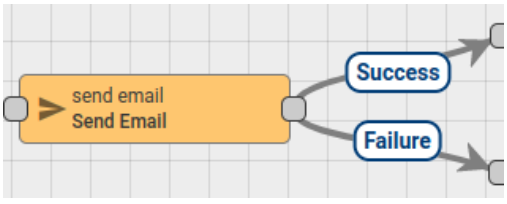
Вот пример сообщения метаданных для отправки одного файла:

```
{
  "attachments": "e18b6950-dfca-11eb-affb-8db134b46d68"
}
```

Примечание: Это часть функции хранения файлов.

В случае успешного запроса исходящее сообщение будет передано следующим узлам по цепочке `Success`, в противном случае используется цепочка `Failure`.

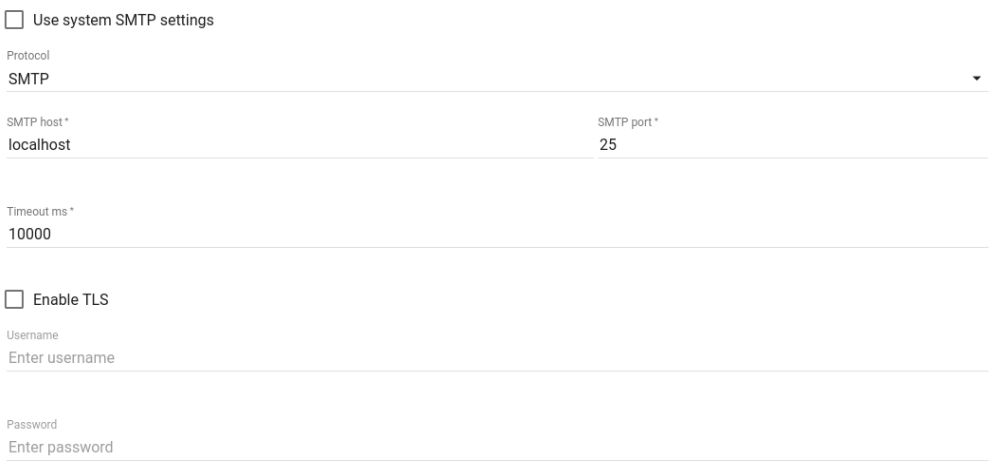
4.7.4 Узел отправки Email



Узел отправляет входящее сообщение с помощью настроенного почтового сервера.

Этот узел работает только с сообщениями, созданными с помощью узла преобразования `To Email`, пожалуйста, соедините этот узел с узлом `To Email`, используя цепочку `Success`.

Конфигурация:



- `Use system SMTP settings` - если включено, то будет использоваться почтовый сервер по умолчанию настроенный на системном уровне
- `Protocol` - Транспортный протокол почтового сервера: `SMTP` или `SMTPS`
- `SMTP host` - Хост почтового сервера
- `SMTP port` - Порт почтового сервера
- `Timeout ms` - тайм-аут чтения в миллисекундах
- `Enable TLS` - если включен, то позволяет использовать команду `STARTTLS` (если поддерживается сервером). Не доступно для ESP.
- `Username` - имя пользователя учетной записи на почтовом хосте, если таковое имеется
- `Password` - пароль для учетной записи на почтовом хосте, если таковой имеется

Этот узел может работать с почтовым сервером по умолчанию, настроенным на системном уровне. Пожалуйста, найдите более подробную информацию о том, как настроить системные параметры SMTP по умолчанию.

Если для этого узла требуется определенный почтовый сервер - отключите флажок `Use system SMTP settings` и настройте почтовый сервер вручную.

Кроме того, этот узел может создавать вложения электронной почты, если входящее сообщение имеет подготовленное поле метаданных `вложений` со ссылкой на файлы, хранящиеся в базе данных.

Поддерживается несколько вложений. Используйте разделенные запятыми ссылки без пробелов (`uuid`) для отправки большого количества файлов. Пример `метаданных` сообщения:

```
{
  "attachments": "e18b6950-dfca-11eb-affb-8db134b46d68,17383b4c-6000-4bb8-be04-
↪b1cb15aa18c5"
}
```

Примечание: Это часть функции хранения файлов.

В случае успешной отправки почты исходное сообщение будет передано следующим узлам по цепочке `Success`, в противном случае используется цепочка `Failure`.

4.7.5 Узел отправки SMS

Узел может построить SMS-сообщение на основе полей метаданных из входящего сообщения и отправить его с помощью доступных SMS-провайдеров.

Конфигурация:

- `Use system SMS provider settings` - если включено, то будет использоваться сервер поставщика SMS по умолчанию, настроенный на системном уровне. Дополнительные сведения см. в разделе Настройки поставщика SMS-сообщений.
- `Phone Numbers To template` - Позволяет настроить несколько телефонных номеров, на которые будет отправляться SMS. При необходимости вы можете ссылаться на поля из метаданных сообщения.
- `SMS message template` - Позволяет настроить тело SMS - сообщения. При необходимости вы можете ссылаться на поля из метаданных сообщения.

Этот узел может работать с поставщиком SMS по умолчанию, настроенным на системном уровне.

Если SMS-сообщение будет успешно отправлено всем получателям, исходное сообщение будет передано следующим узлам по цепочке `Success`, в противном случае используется цепочка `Failure`.