

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
МИРЭА – РОССИЙСКИЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ

С.С. ЗАКОЖУРНИКОВ

**АВТОМАТИЗИРОВАННЫЕ СИСТЕМЫ УПРАВЛЕНИЯ
ТЕМПЕРАТУРОЙ И ОСВЕЩЕНИЕМ**

УЧЕБНОЕ ПОСОБИЕ

Москва – 2023

УДК 681.51

ББК 32.966

319

Закожурников С.С. Автоматизированные системы управления температурой и освещением [Электронный ресурс]: учебное пособие / С.С. Закожурников. — М.: МИРЭА – Российский технологический университет, 2023. — 1 электрон. опт. диск (CD-ROM)

1. Учебное пособие содержит теоретический материал и лабораторный практикум. В издании представлены материалы для изучения программирования управляемых технических систем, интернета вещей, методов и алгоритмов интеллектуализации решения прикладных задач при построении АСУ, методов построения интеллектуальных систем управления технологическими процессами и производствами, которые позволяют сформировать у студентов представление о процессах управления в технологических системах, а также получить теоретические и практические навыки, необходимые при выполнении инженерных задач.

Учебное пособие в первую очередь адресовано студентам, обучающимся в ИПТИП по направлению 09.03.02 и 09.04.02 «Информационные системы и технологии». В то же время пособие может представлять интерес для широкого круга читателей, интересующихся программированием, автоматизацией и решением задач управления.

Автор: Закожурников Сергей Сергеевич

Рецензенты:

Гаряев Андрей Борисович, д.т.н., профессор, заведующий кафедрой, кафедра ТМПУ, ФГБОУ ВО «НИУ «МЭИ»

Архангельский Александр Игоревич, к.п.н., доцент, доцент, кафедра Математики, Московский политехнический университет

Минимальные системные требования:

Наличие операционной системы Windows, поддерживаемой производителем.

Наличие свободного места в оперативной памяти не менее 128 Мб.

Наличие свободного места в памяти хранения (на жестком диске) не менее 15 Мб.

Наличие интерфейса ввода информации.

Дополнительные программные средства: программа для чтения pdf-файлов (Adobe Reader).

Подписано к использованию по решению Редакционно-издательского совета

МИРЭА – Российского технологического университета от _____ 2022 г.

Объем ____ Мб

Тираж 10

© Закожурников С.С., 2023

© МИРЭА – Российский технологический университет, 2023

ОГЛАВЛЕНИЕ

Введение.....	5
Глава 1. Программирование в NodeRed.....	5
1.1. Интерфейс пользователя в NodeRed.....	5
1.1.1. Редактор NodeRed.....	5
1.1.2. Рабочая область.....	7
1.1.3. Настройка рабочей области.....	8
1.2. Потоки.....	9
1.2.1. Добавление потока.....	9
1.2.2. Редактирование свойств потока.....	9
1.2.3. Удаление потока.....	10
1.3. Узлы.....	10
1.3.1. Добавление узлов (нодов).....	10
1.3.2. Меню быстрого добавления нод.....	11
1.3.3 Редактирование настроек нод	12
1.3.4 Конфигурационные ноды.....	13
1.3.5 Соединение нод.....	15
1.3.6 Разбиение проводков.....	15
1.3.7 Перемещение проводков.....	16
1.3.8 Удаление проводков.....	17
1.4. Подпотоки	17
1.4.1. Создание пустого подпотока.....	17
1.4.2. Конвертирование нод в подпоток.....	17
1.4.3. Редактирование подпотока.....	18
1.4.4 . Вход и выходы.....	18
1.4.5. Свойства подпотока.....	19
1.4.6. Выбор нод.....	19
1.4.7 Импорт и экспорт потоков.....	20
1.4.8 Поиск.....	23
1.4.9 Палитра.....	24
1.4.10 Меню «Manage palette».....	25
1.4.11 Управление нодами.....	26
1.5. Функции боковой панели.....	27
1.6. Лабораторные работы.....	32
1.6.1 Лабораторная работа «Автоматизированная система управления температурой».....	32

1.6.2. Лабораторная работа «Автоматизированная система управления освещением».....	35
1.6.3. Лабораторная работа «Автоматизированная система управления освещением в помещении».....	40
Глава 2. Программирование в OpenHAB.....	45
2.1 Интерфейс OpenHAB.....	45
2.2 Синтаксис OpenHAB.....	49
2.2.1 Триггеры правил.....	51
2.2.2 Сценарии.....	55
2.2.3 Работа с состояниями элементов: преобразования.....	59
2.3 Проектирование интерфейсов дашбордов с помощью HABPanel.....	72
2.3.1 Концепции.....	73
2.3.2 О сохранении данных.....	74
2.3.3 Основные элементы и функции интерфейса.....	76
2.4 Стандартные виджеты.....	85
2.5 Лабораторная работа «Автоматизация уличного освещения в программе OpenHAB».....	93
Список литературы.....	104

ВВЕДЕНИЕ

В пособии изложены основные принципы работы программ NodeRed и OpenHAB. NodeRed и OpenHAB – это инструменты программирования, предназначенные для объединения аппаратных устройств, прикладных программных интерфейсов и онлайн-сервисов новыми и интересными способами [1, 2].

NodeRed предоставляет редактор на основе браузера, который позволяет легко объединять потоки, используя широкий спектр узлов в палитре, которые можно развернуть в среде выполнения одним щелчком мыши.

OpenHAB — это платформа домашней автоматизации с открытым исходным кодом, независимая от технологий, которая работает как центр умного дома.

При помощи этих программ мы можем автоматизировать освещение или температуру в помещениях различного назначения, тем самым сделав окружающее пространство комфортнее и уютнее. Также с помощью этих программ можно информировать пользователя о неисправности оборудования, срабатывании сигнализации и т.д.

Знание этих программ позволит обучающимся воплотить в жизнь практически любое желание в области практической автоматизации.

Автор надеется, что данное пособие окажет существенную помощь студентам бакалавриата и магистратуры, изучающих дисциплины по автоматизации, управлению и программированию.

Часть I. ПРОГРАММИРОВАНИЕ В NodeRed

1.1. ИНТЕРФЕЙС ПОЛЬЗОВАТЕЛЯ В NodeRed

1.1.1. Редактор NodeRed

Редактор Node-RED включает в себя четыре основные части (рис. 1-2):

- Верхняя часть под названием «шапка» выделена жёлтым цветом. Она включает в себя несколько кнопок: «Deploy», главное меню и опционально меню пользователя.
- Красным цветом на рисунке выделен элемент под названием «палитра». Палитра включает в себя ноды (узлы). Они применяются в редакторе.
- Синим цветом выделена главная рабочая область. Она необходима для формирования потоков.

- Зелёным цветом выделена боковая панель.

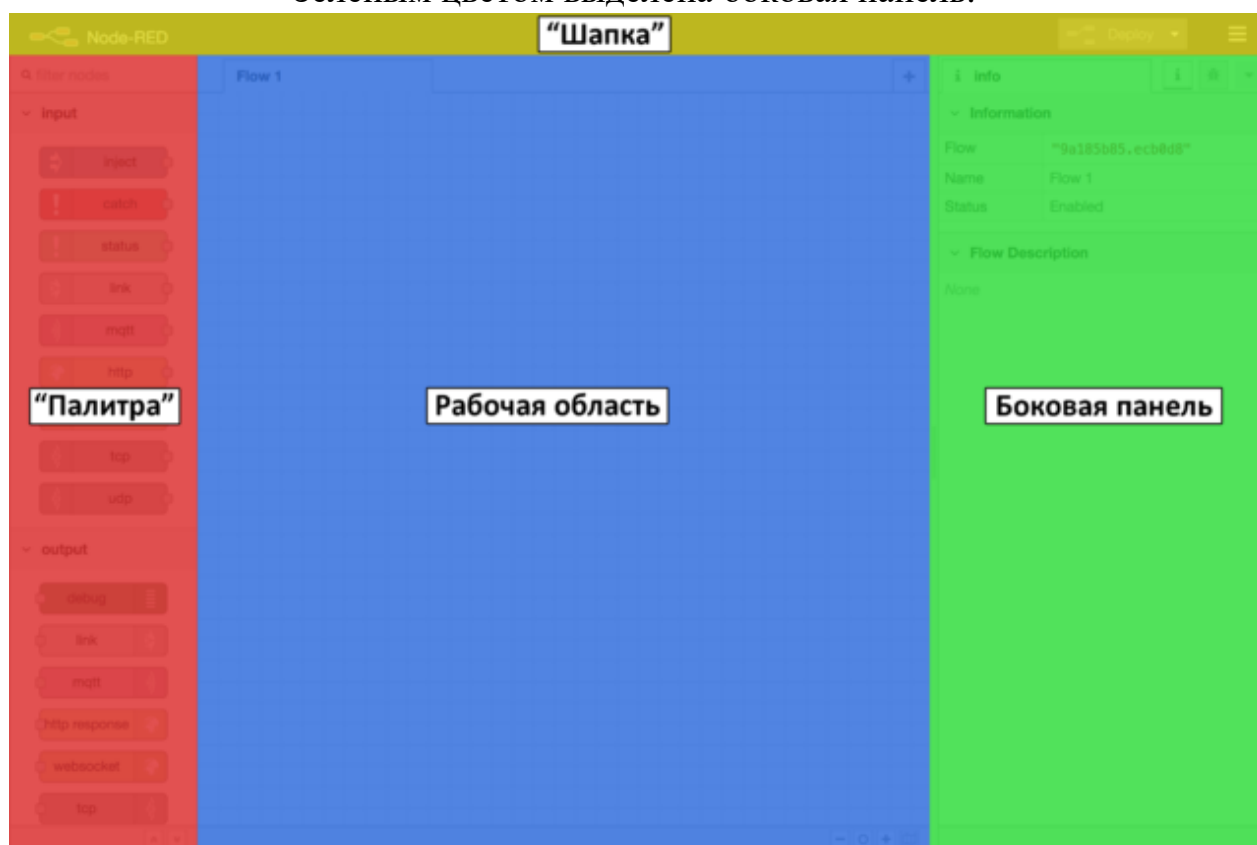


Рисунок 1. Редактор Node-RED

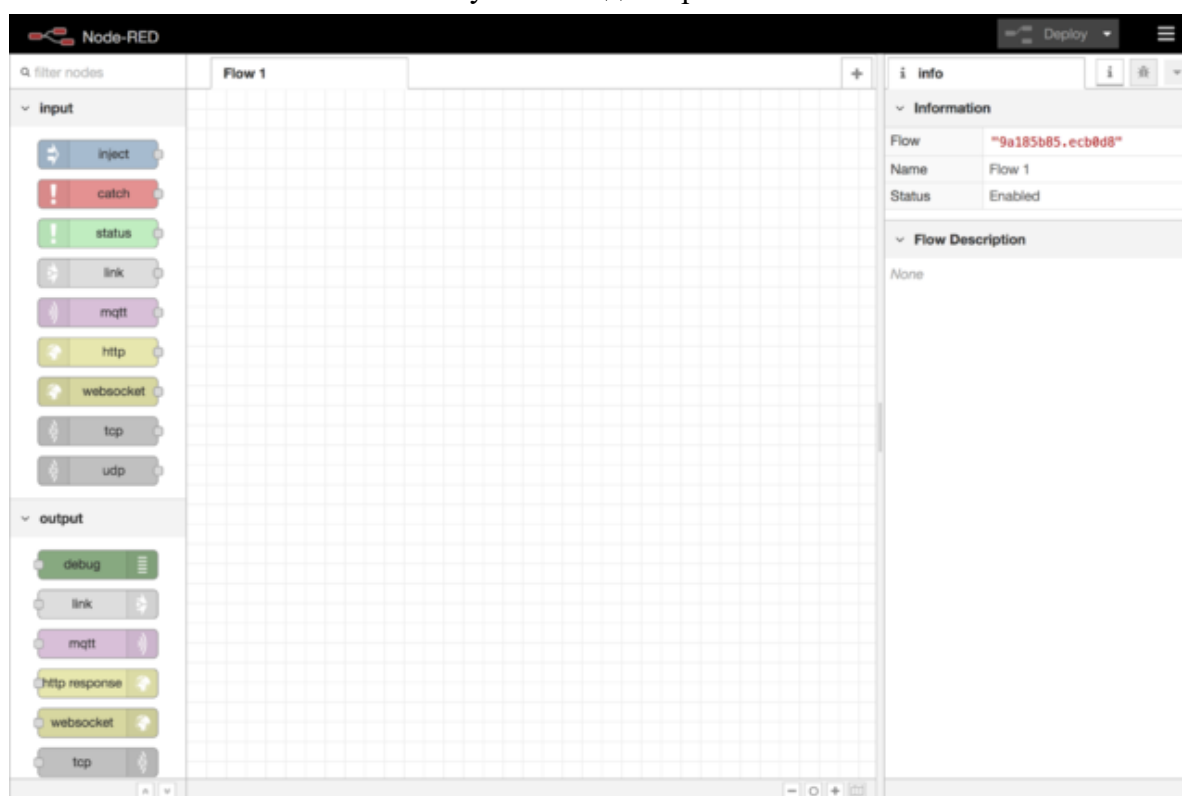


Рисунок 2. Окно редактора

1.1.2. Рабочая область

Рабочая область применяется для формирования потоков. Для решения задач необходимо добавить ноды из «палитры» и соединить их нужным образом.

При открытии потока или подпотока в рабочей области открывается новая вкладка [3].

Инструменты для просмотра рабочей области

В нижней части рабочей области располагается зум. С помощью него пользователь может увеличивать объекты (в операционной системе Windows: **Ctrl** + **=**, в Mac: **⌘** + **=**), уменьшать объекты (в операционной системе Windows: **Ctrl** + **-**, в Mac: **⌘** + **-**), переходить к базовому масштабу (в операционной системе Windows: **Ctrl** + **0**, в Mac: **⌘** + **0**).

Также для удобства просмотра можно активировать навигатор по рабочей области (рис. 3).

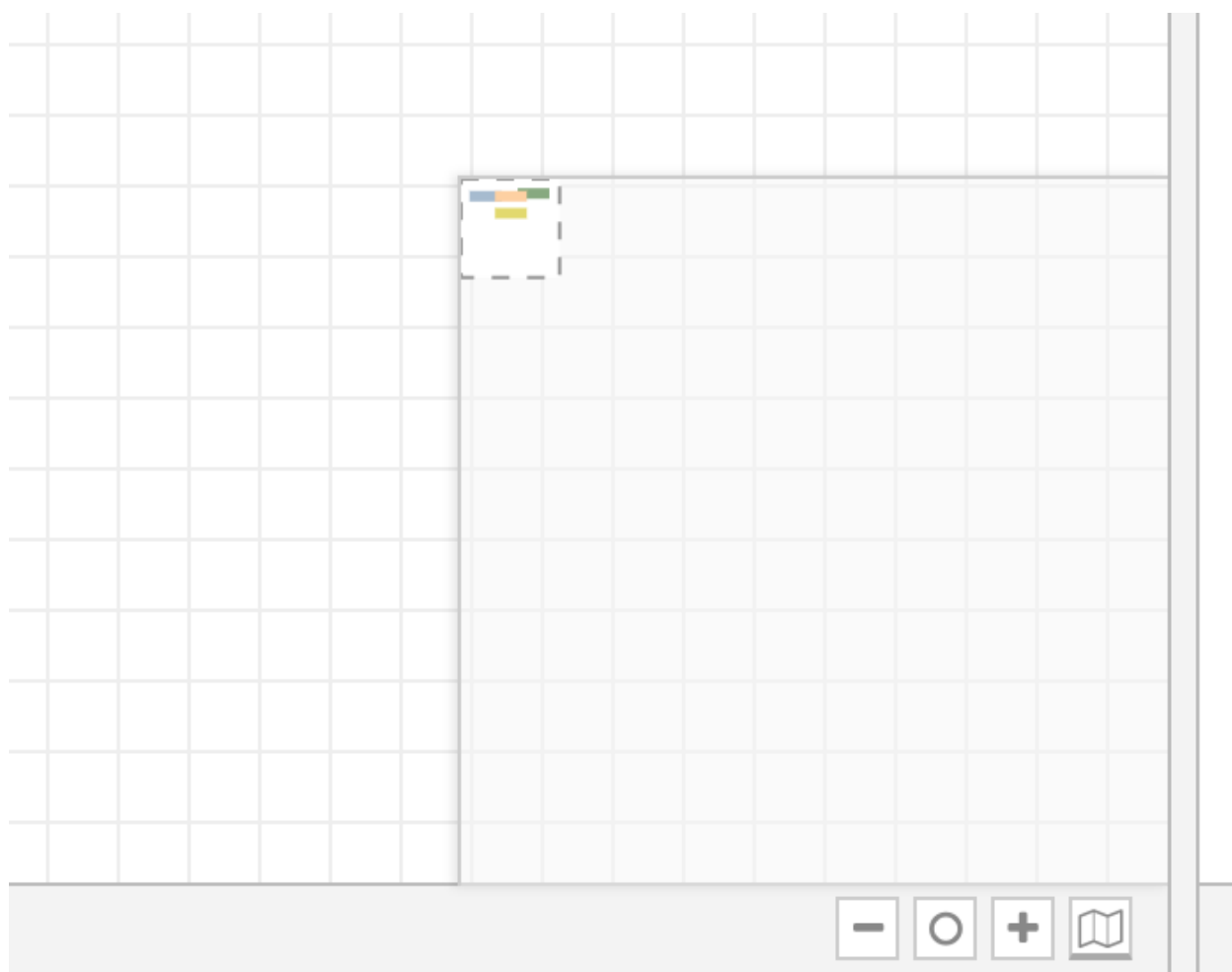


Рисунок 3. Нижняя правая часть рабочей области

В окне навигатора пунктирной линией обозначено место, которое студент использует прямо сейчас. Существует возможность оперативно изменить обозначенное место и перейти к другой части рабочей области. Если возникает неточность, связанная с перемещением ноды за рабочую область, то найти такую ошибку можно также с помощью навигатора.

1.1.3. Настройка рабочей области

Меню «View» позволяет изменить рабочую область (в операционной системе Windows: Ctrl +, в Mac: ⌘ +). Необходимо нажать на кнопку с тремя полосками справа вверху, а затем на «Settings» > «View» (рис. 4).

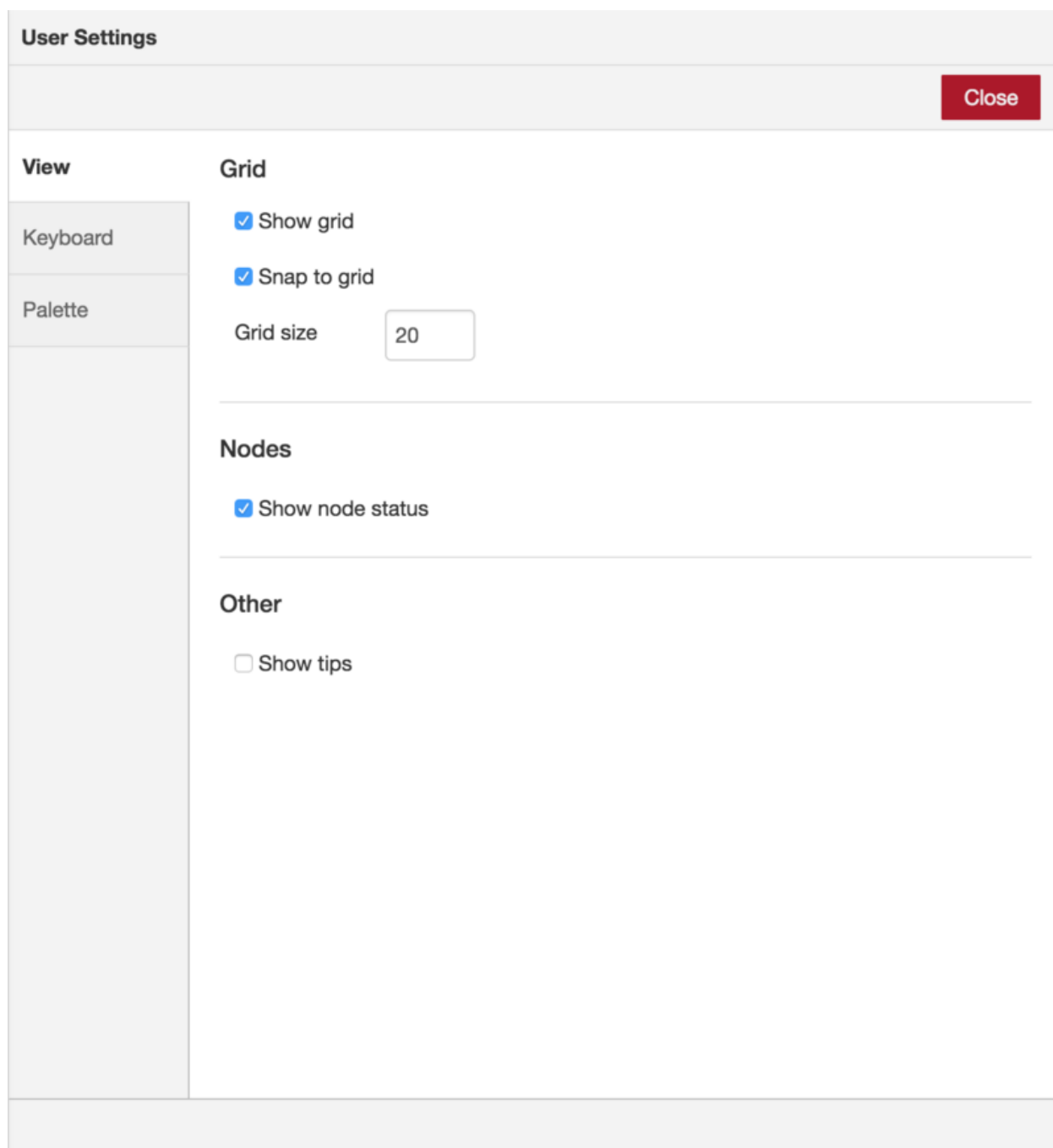


Рисунок 4. Меню «View»

1.2. ПОТОКИ

1.2.1. Добавление потока

Для формирования нового потока необходимо нажать на кнопку с плюсом в правой верхней части рабочей области. Вторым способом формирования – сделать клик по кнопке с 3 полосками справа сверху. Далее нажать на «Flows» > «Add» (рис. 5).



Рисунок 5. Вкладка потока 1

1.2.2. Редактирование свойств потока

Для изменения свойств потока «Edit flow» необходимо нажать на вкладку в верхней части рабочей области. Вторым способом изменения – сделать клик по кнопке с 3 полосками справа сверху. Далее нажать на «Flows» > «Rename» (рис. 6).

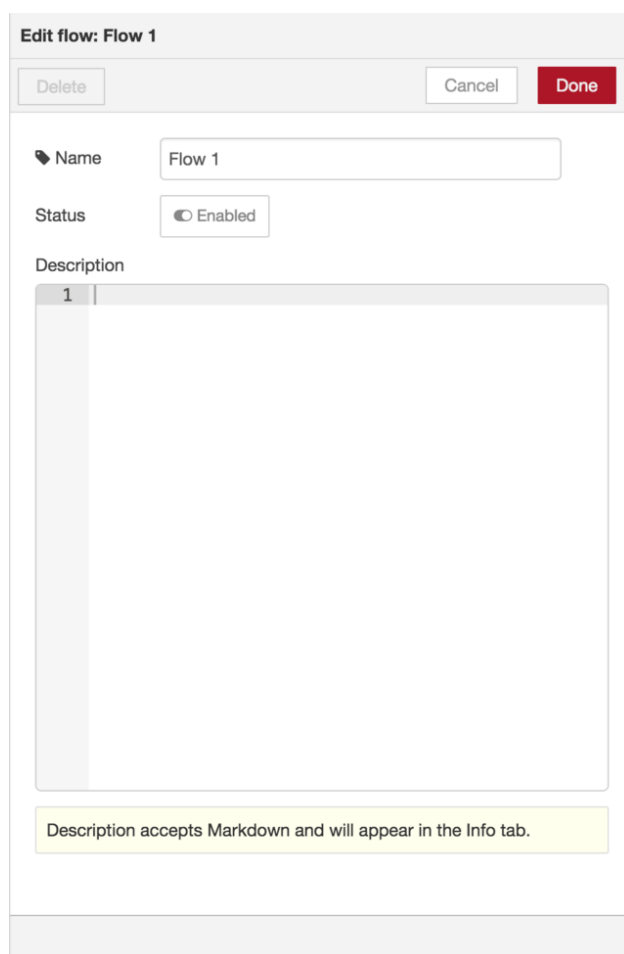


Рисунок 6. Редактор свойств потока

Свойства потока можно изменить в меню. Вкладка «info» необходима для просмотра описания потока. Язык разметки Markdown применяют для определения свойств потока. Свойство «Status» необходимо для включения или выключения потока.

1.2.3. Удаление потока

Существует несколько способов удаления потока. Первый способ – использование кнопки «Delete» в меню «Edit flow». Второй способ – нажатие на кнопку с тремя полосками справа сверху. Далее кликаете на «Flows» > «Delete».

1.3. УЗЛЫ

1.3.1. Добавление узлов (нодов)

Существует несколько способов создания нод (узлов) в рабочей области:

- Перетягивание из «палитры»
- Использование меню быстрого добавления нодов
- Добавление из меню «Library» или «Clipboard»

Узлы связаны друг с другом при помощи «проводков», подсоединённых к портам. Узел имеет один входной порт и несколько выходных портов. Если навести курсор мыши на порт, то можно увидеть его описание. Узел «switch» показывает правило, соответствующее выходному порту. Меню редактирования узла используется для описания свойств портов.

Статусное сообщение и иконка являются частью описания некоторых узлов. Они нужны для того, чтобы показать текущее состояние узла. MQTT-узлы показывают, подключены они или нет (рис. 7).

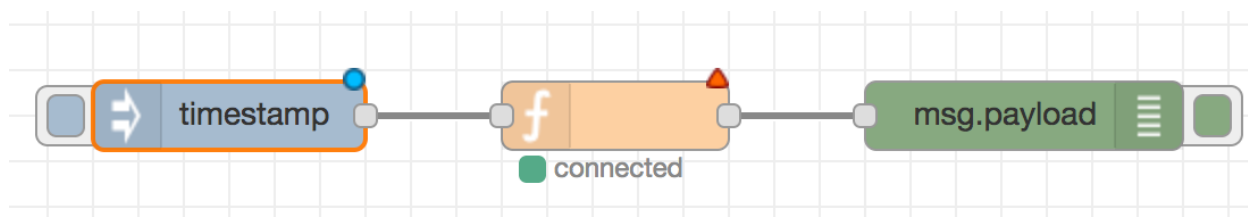


Рисунок 7. Элементы ноды

Синий кружок в верхней части узла показывает несохраненные изменения в ноде. Красный треугольник в верхней части означает наличие ошибки в настройках узла. Слева или справа от некоторых узлов есть кнопка, которая позволяет взаимодействовать с узлом прямо в редакторе. Стандартные ноды с кнопками – это «inject» и «debug».

1.3.2. Меню быстрого добавления нод

Узел можно оперативно добавить в рабочую область с помощью меню быстрого добавления.

Для того, чтобы зайти в меню, необходимо нажать **Ctrl** или **⌘** (если у вас Mac), а затем нажать левой кнопкой мыши в поле рабочей области (рис. 8).

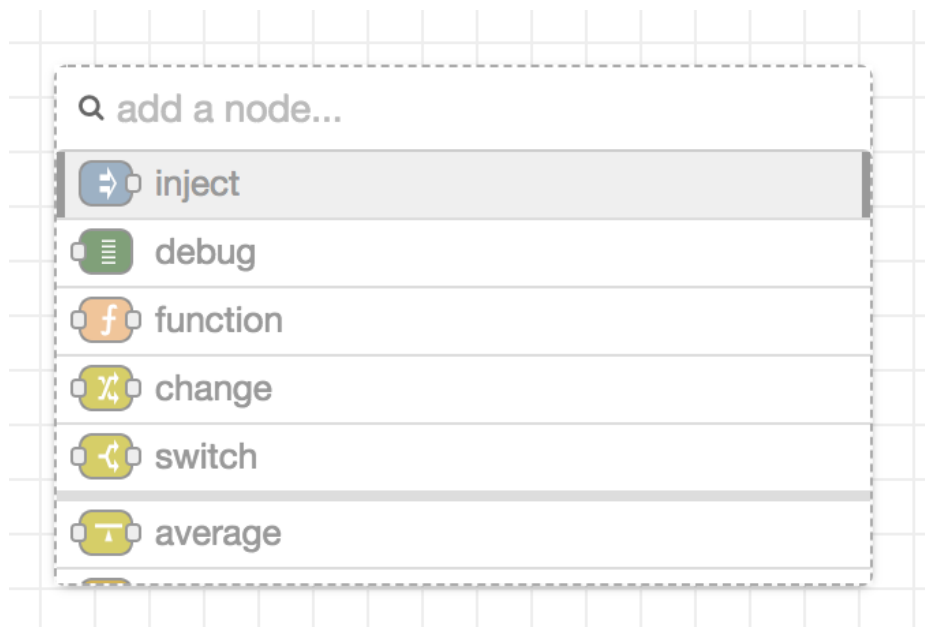


Рисунок 8. Меню быстрого добавления нод

Список всех нод, имеющихя в редакторе, находится в рассматриваемом меню. Список состоит из пяти главных нод, и перечня новых узлов, после представлен полный алфавитный список всех оставшихся нод.

В верхней части меню есть поисковая строка.

1.3.3. Редактирование настроек нод

Для редактирования свойств узлов необходимо нажать по узлу или выберите узел, а затем нажать на **↵ Enter** на клавиатуре. Если выделить несколько узлов, то после клика на **↵ Enter** развернётся меню редактирования для первого из узлов (рис. 9).

The image displays three panels of the 'Edit function node' dialog, each with a 'Delete' button, a 'Cancel' button, and a 'Done' button. The top panel shows the 'Properties' tab with fields for 'Name' and 'Function'. The 'Function' field contains a code editor with the following code:

```
1  
2 msg.payload = "hello world";  
3 return msg;
```

The bottom panel shows the 'Description' tab with a 'Description' field containing the text '1 функция "Hello world!"'. The right panel shows the 'Appearance' tab with fields for 'Label', 'Icon', 'Port labels', 'Inputs', and 'Outputs'. The 'Label' field is empty, the 'Icon' field shows a small 'f' icon, the 'Port labels' field is empty, the 'Inputs' field shows '1. none', and the 'Outputs' field shows '1. none'.

Рисунок 9. Меню редактирования ноды

Разделы «Properties» («Свойства»), «Description» («Описание») и «Appearance» («Внешний вид») являются самыми важными в меню узлов. В части «Properties» заключены свойства, принадлежащие выбранному узлу (например, для узла «function», в этой части будет текстовый редактор для введения кода), в «Description» – текстовый редактор описания узла, а в «Appearance» задается иконка узла, названия для портов (рис. 10).

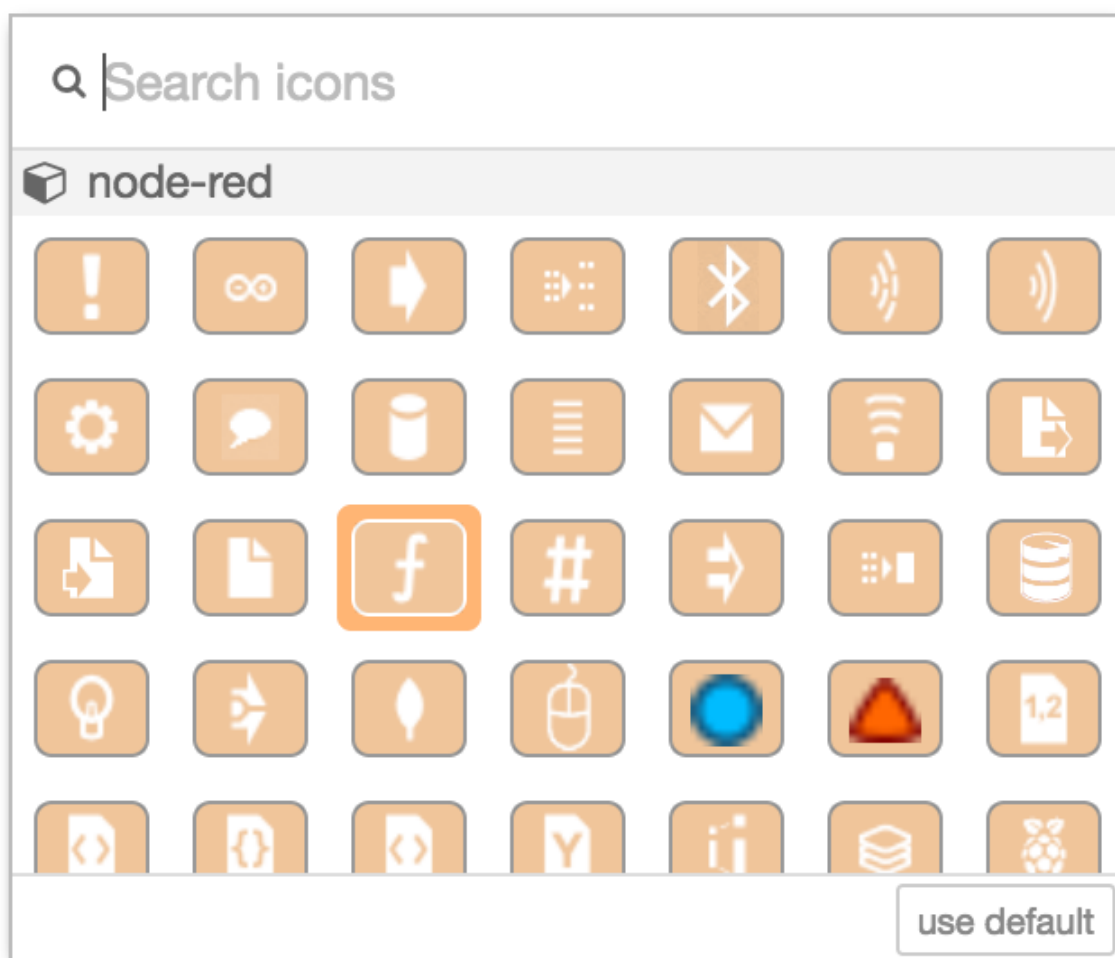


Рисунок 10. Меню для выбора иконки

1.3.4. Конфигурационные ноды

Конфигурационные узлы являются специальным типом узлов. Их особенность – использование многократных настроек, а также возможность использовать несколько обычных узлов потока.

Например, ноды «mqtt in» и «mqtt out» используют конфигурационную ноду «mqtt broker», чтобы воспользоваться одним и тем же соединением с MQTT-брокером.

Меню редактирования ноды предназначено для добавления конфигурационных узлов. В нем существует специальная часть, где разрешено либо выбрать конфигурационную ноду нужного типа, либо добавить новый экземпляр (рис. 11).



Рисунок 11. Добавление конфигурационной ноды

Необходимо нажать на кнопку с иконкой карандаша справа от поля «Server» для использования этого меню (рис. 12).

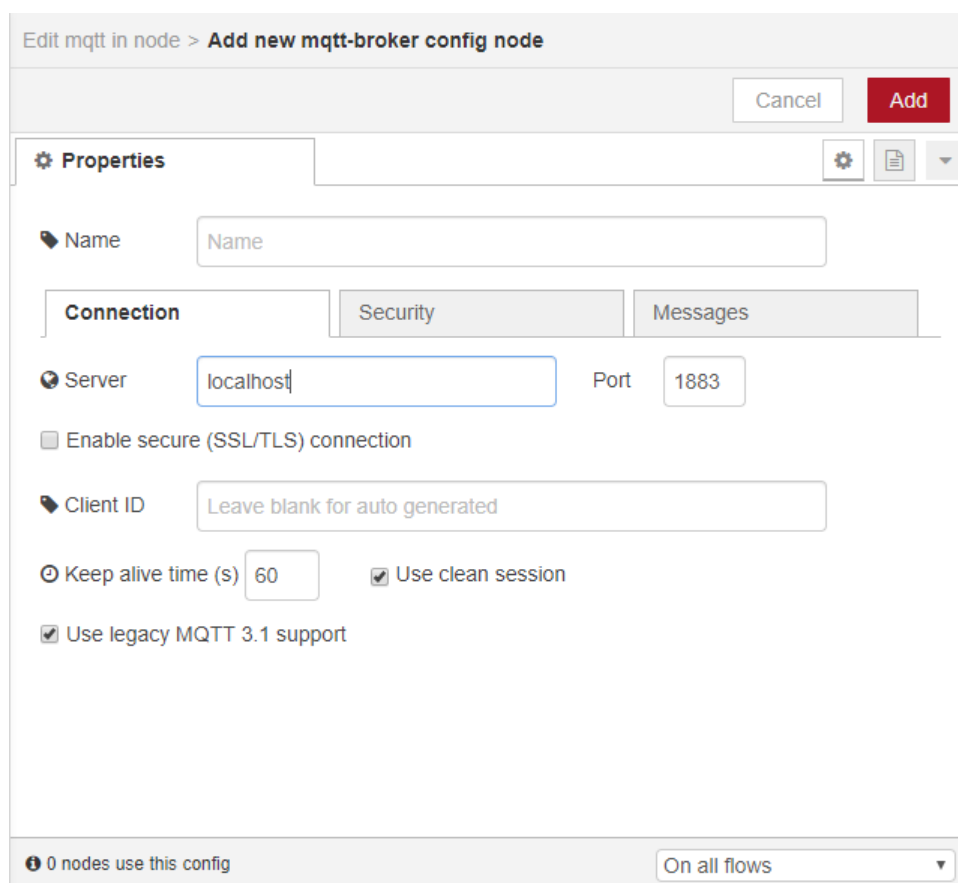


Рисунок 12. Меню добавления конфигурационной ноды

В меню добавления конфигурационного узла нет части «Appearance». Но присутствуют две другие важнейшие части – «Properties» и «Description».

В левой нижней части этого меню показывается, сколько нод используют эту конфигурационную ноду, а в правой нижней части находится меню для выбора того, какими потоками будет использоваться эта конфигурационная нода. По умолчанию она будет использоваться всеми потоками, но через это меню можно задать использование этой ноды только одним потоком.

Часть боковой панели под названием «Config» предназначена для управления узлами конфигурации. Для того, чтобы ею воспользоваться необходимо нажать на кнопку с перевернутым треугольником справа вверху. Далее необходимо нажать «Configuration nodes».

1.3.5. Соединение нод

Для соединения двух узлов необходимо нажать левой кнопкой мыши на порте одного узла, далее потянуть этот «проводок» к порту другого узла, а затем отпустить левую кнопку мыши (рис. 13).

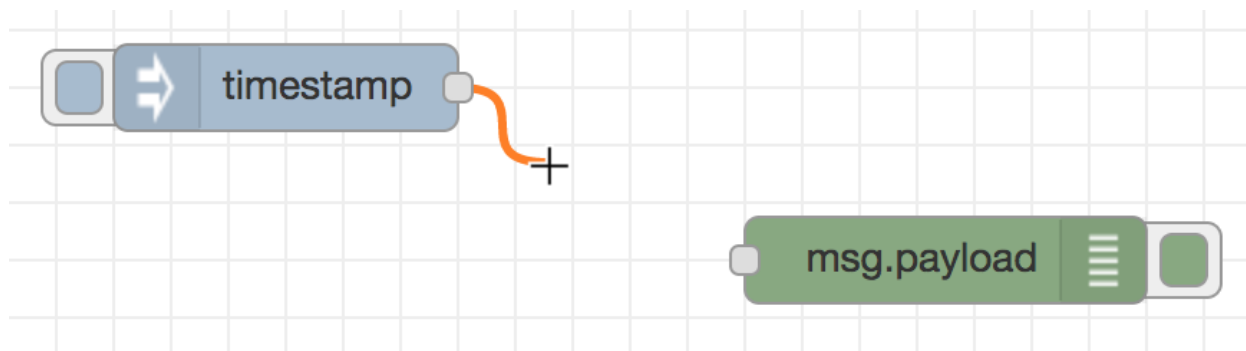


Рисунок 13. Подключение нод друг к другу

Пользователь может нажать клавишу **Ctrl** в Windows или **⌘** в Mac, нажать левой кнопкой мыши по порту одной ноды (и отпустить), а затем по порту другой ноды, а затем отпустить **Ctrl** (Windows) или **⌘** (Mac).

Данный метод можно рассмотреть совместно с работой в меню быстрого добавления узлов (для использования которого тоже требуется клавиша **Ctrl** если у вас Windows или **⌘** если у вас Mac). Это позволяет быстро добавлять в поток нужные ноды, попутно объединяя их с уже созданными нодами потока.

1.3.6. Разбиение проводков

Для встраивания нового узла между двумя нодами вам необходимо перетащить его на провод, который их соединяет. После этого провод выделится штриховой линией. Если отпустить ноду в этом месте, она автоматически

встроится в поток, став промежуточным звеном между двумя прежними нодами (рис. 14).

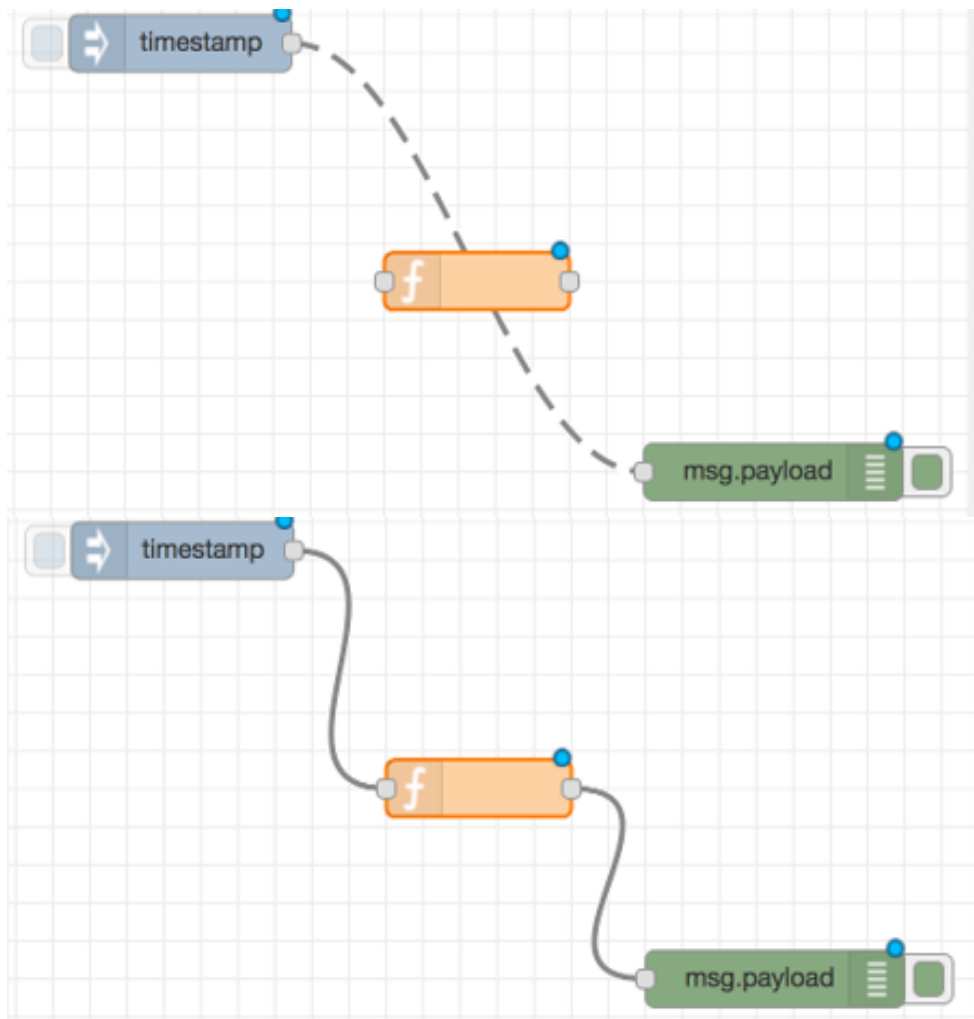


Рисунок 14. Размещение ноды между двумя другими нодами

1.3.7. Перемещение проводков

Для переподключения провода от одного узла к другому необходимо нажать **↑ Shift** на клавиатуре, зажать левую кнопку мыши на порте, от которого нужно отключиться, протяните проводок к новому порту и отпустите левую кнопку мыши. Нельзя отпускать левую кнопку мыши во время перетягивания проводка.

Для переподключения всех проводов, подключённых к порту, необходимо не выбирать ни один из них, нажать **↑ Shift**, а затем зажать левой кнопкой мыши на этом порту.

1.3.8. Удаление проводков

Для удаления проводка необходимо его выделить (кликнув по нему левой кнопкой мыши), а затем нажать на **Delete** на клавиатуре.

1.4. ПОДПОТОКИ

Подпоток – это совокупность узлов, объединенных в одну ноду.

Подпоток применяют для уменьшения визуальной сложности потока [4]. Подпоток образуется в палитре существующих узлов. Индивидуальные экземпляры подпотока могут быть добавлены в рабочую область аналогично всем другим нодам.

Примечание: Подпоток не может содержать экземпляр самого себя – ни напрямую, ни косвенно.

1.4.1. Создание пустого подпотока

Для открытия нового подпотока необходимо нажать на кнопку с тремя полосками справа вверху, а затем кликнуть по «Subflows» > «Create subflow». Это создаст пустой подпоток и откроет его в рабочей области.

1.4.2. Конвертирование нод в подпоток

Существует возможность преобразовать несколько узлов в подпоток, нажав на кнопку с тремя полосками справа вверху, далее кликнув на «Subflows» > «Selection to Subflow». Выбранные ноды будут перемещены в новый подпоток и заменены на ноду-экземпляр подпотока (рис. 15).

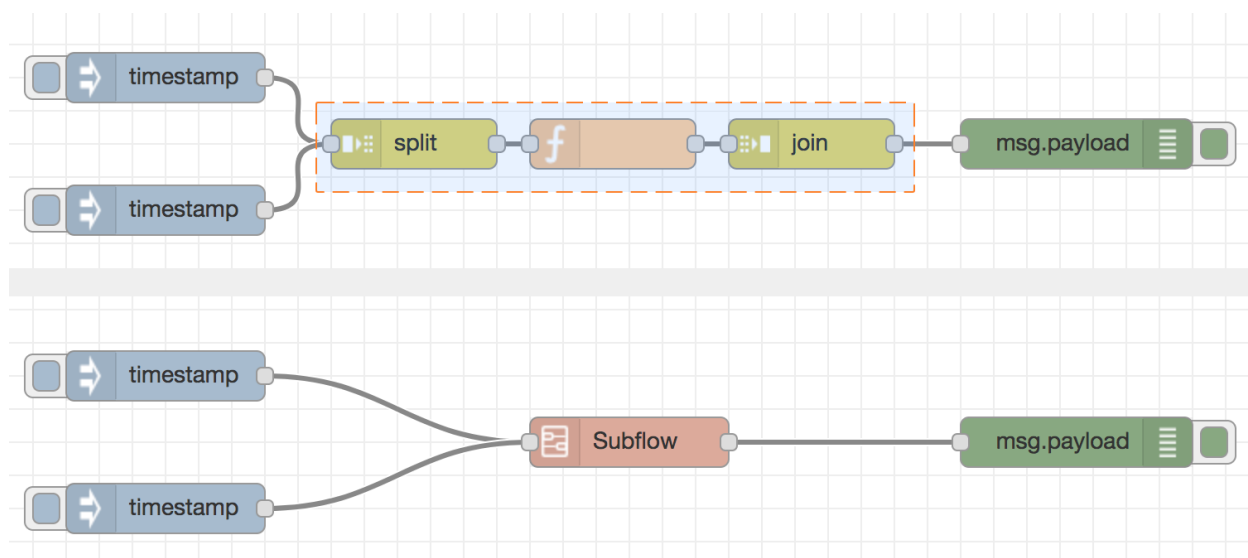


Рисунок 15. Создание подпотока прямо в рабочей области

Узел нового подпотока должен иметь одну входную точку, следовательно, это является важным условием создания нового подпотока (рис. 16).

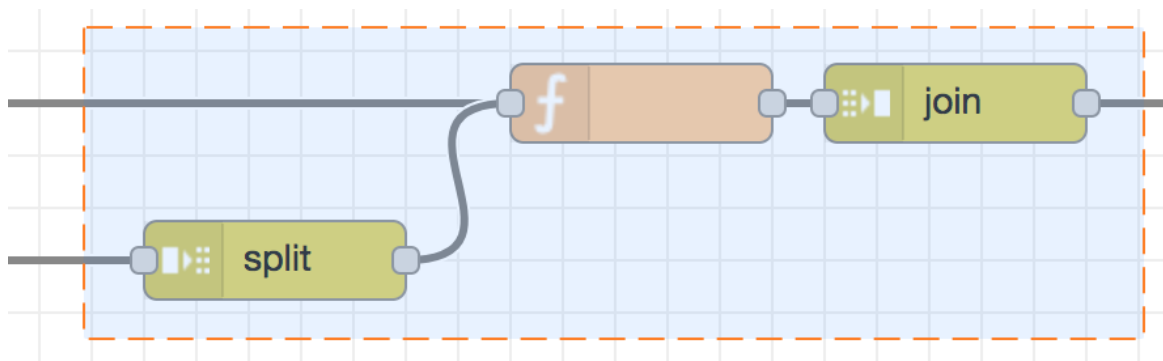


Рисунок 16. Из этих нод подпоток создать не получится

1.4.3. Редактирование подпотока

Для редактирования подпотока можно воспользоваться несколькими способами: во-первых, можно нажать на узел подпотока в «палитре» два раза, во-вторых, кликнуть на кнопку «Edit subflow template» в меню редактирования узла подпотока.

В результате в рабочей области откроется новая вкладка для этого подпотока. В отличие от вкладок для обычных потоков, у вкладок для подпотоков справа есть крестик, с помощью которого их можно закрыть (спрятать).

1.4.4. Вход и выходы

Входы и выходы подпотока изображены в виде квадратных узлов, окаймлённых штриховой линией. У них есть в наличии порт для подключения к потоку. В верхней панели вкладки подпотока есть инструменты для добавления и удаления входных/выходных нод. Как и у стандартных нод, у ноды подпотока может быть только один вход и сколько угодно выходов (рис. 17).

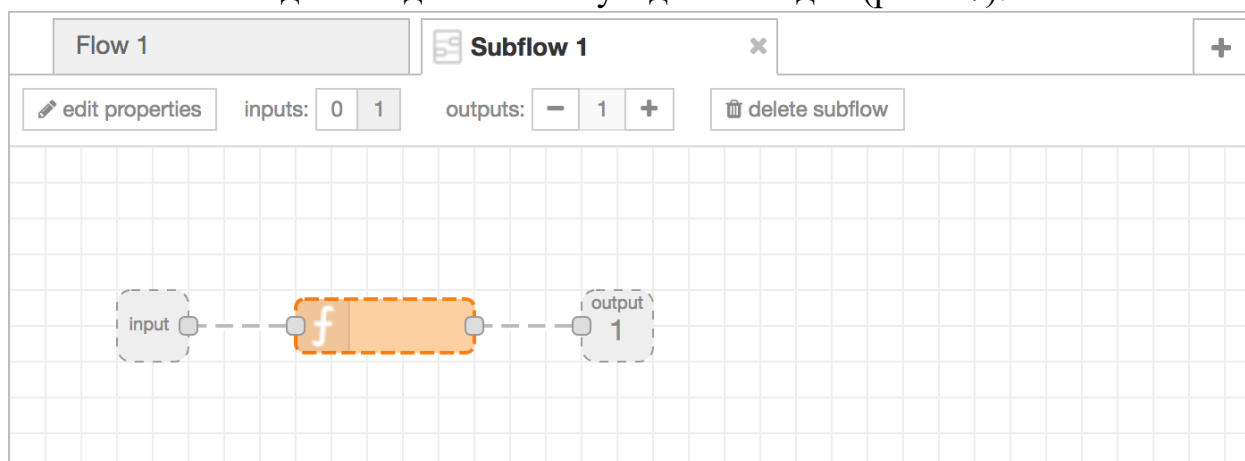


Рисунок 17. Редактирование подпотока

1.4.5. Свойства подпотока

Изменять свойства подпотока, можно нажав на кнопку «**Edit properties**». В данном разделе вы можете изменить название подпотока, а также его описание, задать категорию «палитры» подпотока. Существует возможность задать одну из уже имеющихся категорий или создать новую (рис. 18).

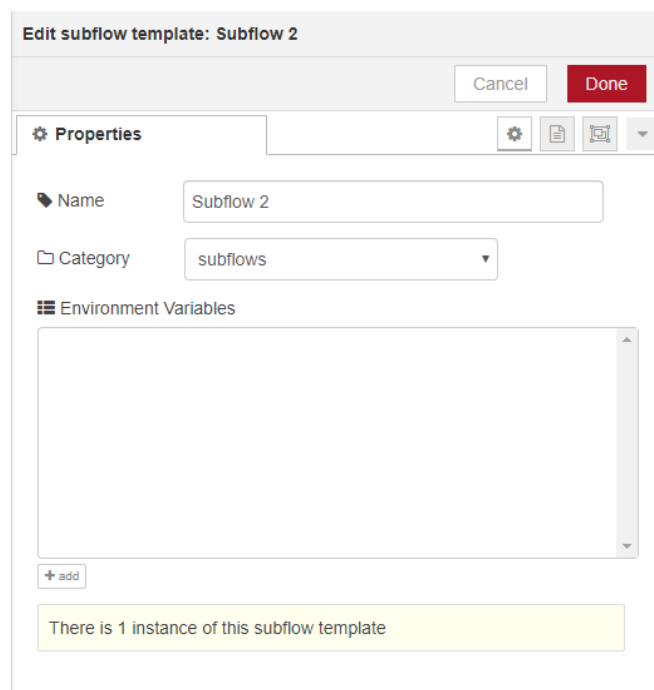


Рисунок 18. Меню редактирования свойств подпотока

В меню находится кнопка «Delete subflow», с помощью которой можно удалить подпоток и все его экземпляры.

1.4.6. Выбор нод

Для выбора узла необходимо нажать по нему левой кнопкой мыши. Данное действие приведёт к отмене выделения элементов, выбранных до этого. Свойства узла, его описание и другая вспомогательная информация будут доступны во вкладке «Info».

При нажатии по узлу и одновременном зажатии **Ctrl**, если вы пользуетесь ОС Windows или **⌘** если у вас Mac, вы можете добавить узел к ранее выбранным нодам (или отменить выбор узла).

Если нажать по ноду и зажать **⇧ Shift**, то вы сможете выделить сам узел и все ноды, которые к нему подключены.

Чтобы выбрать проводок, по нему тоже нужно кликнуть. В отличие от нод, одновременно можно выбрать только один проводок.

Прямоугольное выделение

Выделение нескольких нод с помощью прямоугольного выделения

Прямоугольное выделение – это функция, позволяющая выделить сразу несколько узлов. Чтобы ею воспользоваться необходимо нажать и не отпускать левую кнопку мыши, далее потянуть мышь в сторону. Образованный таким образом прямоугольник должен охватить нужные узлы, а проводки с помощью этого инструмента выделить нельзя (рис. 19).

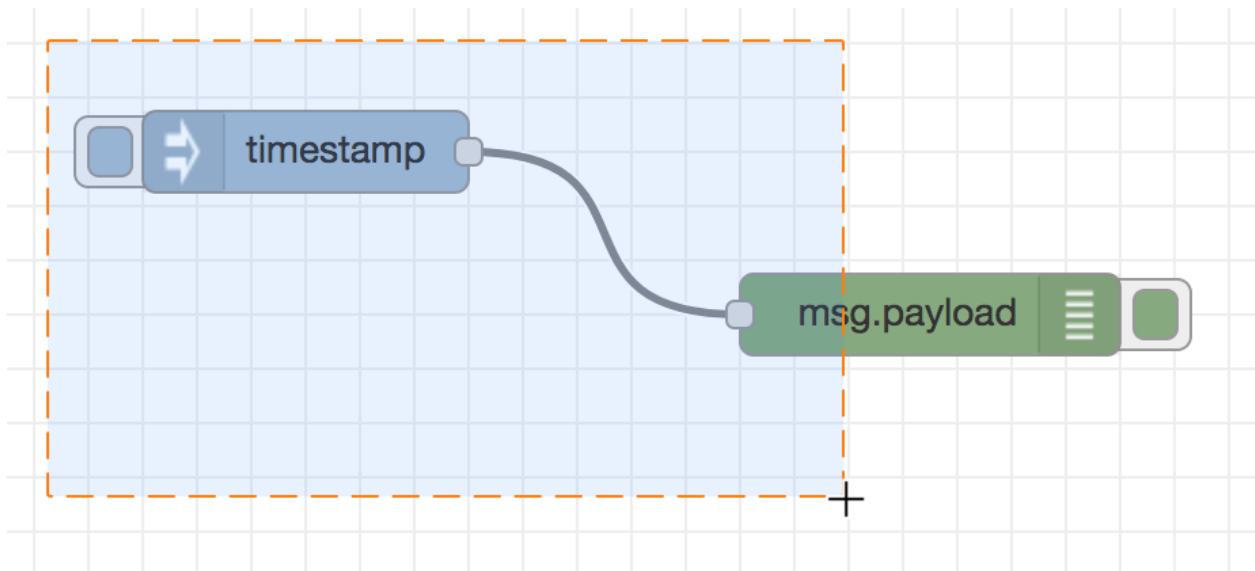


Рисунок 19. Прямоугольное выделение

Выбор всех узлов

Чтобы выбрать все ноды в текущем потоке, кликните по рабочей области с нужным потоком, а затем нажмите горячие клавиши **Ctrl + A** / **⌘ + A**.

Буфер редактора

Редактор Node-RED поддерживает стандартные операции копирования (**Ctrl + C** / **⌘ + C**), вырезания (**Ctrl + X** / **⌘ + X**) и вставки (**Ctrl + V** / **⌘ + V**).

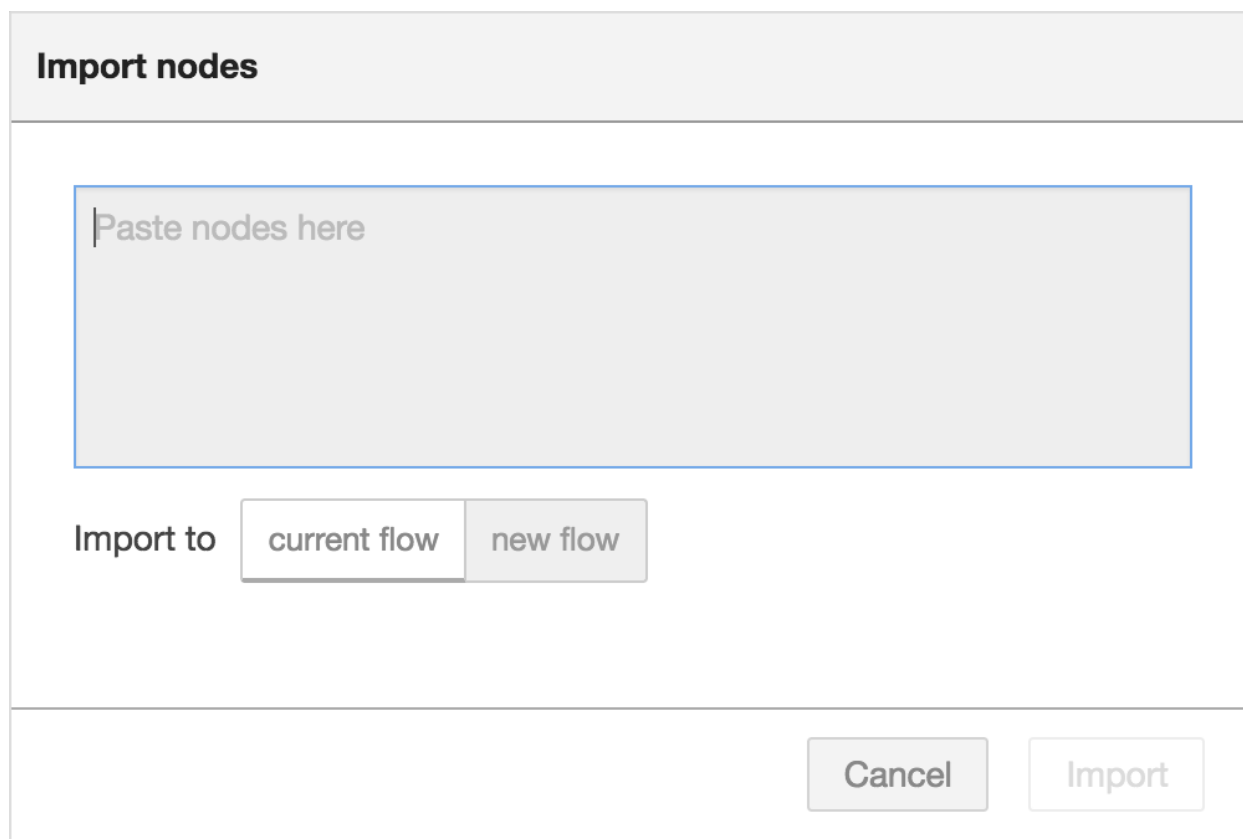
Примечание: Эти операции используют не системный буфер, а буфер редактора.

1.4.7. Импорт и экспорт потоков

Потоки можно импортировать в редактор Node-RED и экспортировать из него при помощи JSON-кода, что заметно упрощает обмен потоками между пользователями.

Импорт потоков

Для импорта потока необходимо воспользоваться меню импорта. Для этого нужно нажать по кнопке с тремя полосками, далее нажать на «Import» > «Clipboard» (также можно нажать **Ctrl + I** / **⌘ + I**). После этого необходимо добавить JSON-код в поле посередине и нажать на кнопку «Import» (рис. 20).



The image shows a dialog box titled "Import nodes". Inside the dialog, there is a large rectangular text area with the placeholder text "Paste nodes here". Below this text area, there is a label "Import to" followed by two buttons: "current flow" and "new flow". At the bottom right of the dialog, there are two buttons: "Cancel" and "Import".

Рисунок 20. Меню импорта нод

Кнопка «Import» будет активна только, если JSON-код будет написан без ошибок.

Здесь также можно задать, куда импортировать поток – в текущий поток или создать новый поток.

Экспорт потоков

Для экспорта потока необходимо воспользоваться меню экспорта. Для этого нужно нажать по кнопке с тремя полосками, далее нажать на «Export» > «Clipboard» (также можно просто нажать **Ctrl + E** / **⌘ + E**). Здесь находится JSON-код, который можно скопировать и поделиться с другими (рис. 21).

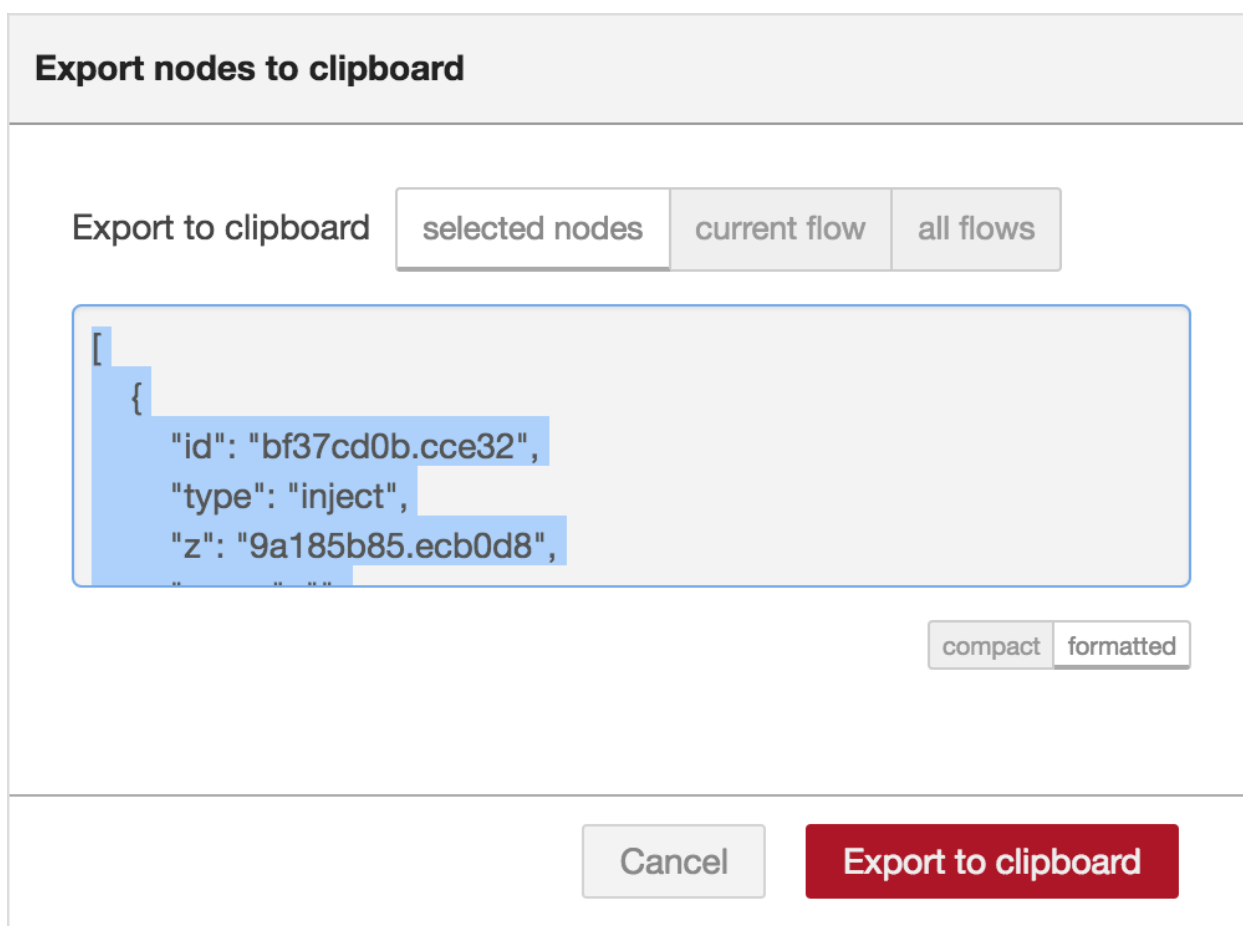


Рисунок 21. Меню экспорта потоков

В данном меню пользователь может задать объекты для экспорта: выделенные узлы, текущий поток или все потоки сразу.

Также пользователь может определить формат экспортируемого JSON-кода – его можно сделать либо компактным, либо упорядоченным. В компактном формате JSON-код будет преобразован в одну строку без пробелов. В упорядоченном формате JSON-код примет удобочитаемый вид – со строчками, имеющими правильные отступы.

1.4.8. Поиск

Для поиска узлов необходимо использовать меню экспорта. Чтобы им воспользоваться, необходимо нажать на горячие клавиши **Ctrl + F** / **⌘ + F** (рис. 22).

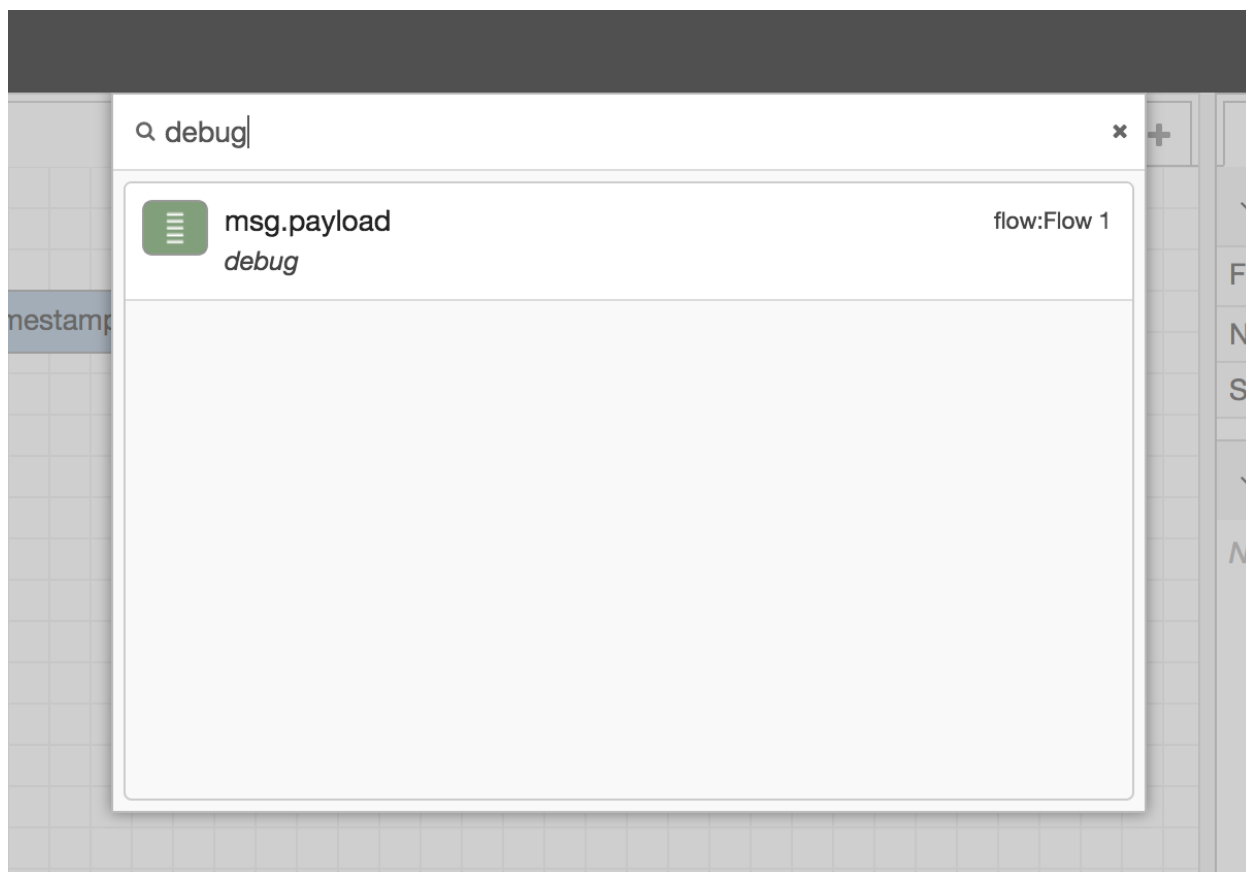


Рисунок 22. Меню поиска

В этом меню показываются все свойства ноды, что позволяет найти ноду по ее ID, типу, названию и т.д.

Если выбрать ноду в выдаче поиска, то она будет отображаться в рабочей области редактора.

1.4.9. Палитра

«Палитра» - это инструмент, включающий в себя все узлы в редакторе. Они сгруппированы в несколько категорий. В самом верху находятся категории «input», «output» и «function», но если у вас есть подпотоки, то самой верхней категорией будет «subflows». Ноды в категории можно развернуть и свернуть – для этого нажмите по ее «шапке» (рис. 23).

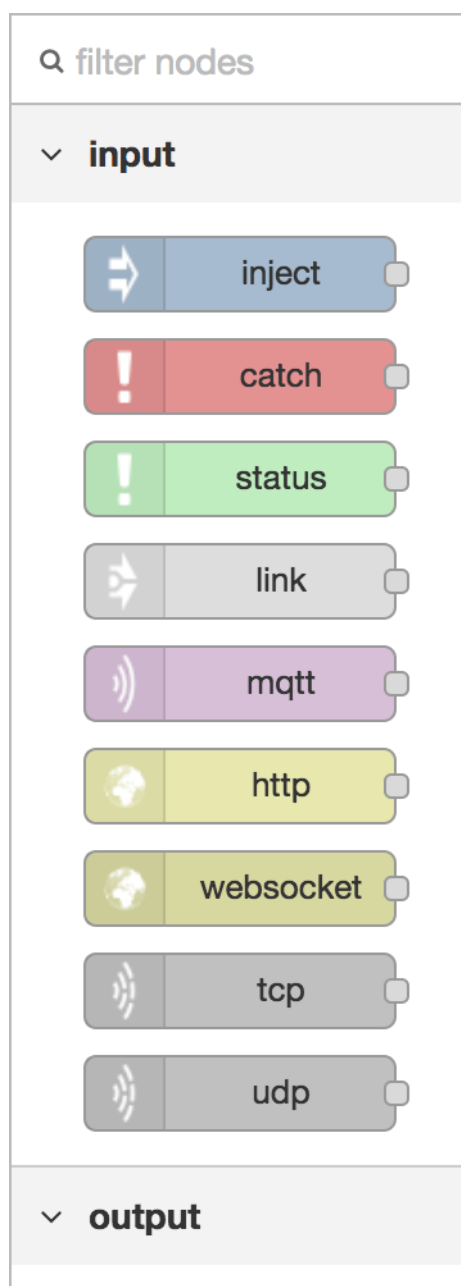


Рисунок 23. Палитра

Чтобы развернуть/свернуть все категории, воспользуйтесь двумя кнопками в самом низу «палитры» – они изображены в виде двойных стрелочек, направленных вниз и вверх. В верхней части палитры есть поле для ввода текста, предназначенное для поиска нужных нод.

1.4.10. Меню «*Manage palette*»

Меню «*Manage palette*» применяют для добавления в «палитру» новых узлов. Для его активации необходимо нажать по кнопке с тремя стрелочками справа сверху, а затем на пункт «*Manage palette*». Или использовать сочетание клавиш **Ctrl + ⬆ Shift + P** / **⌘ + ⬆ Shift + P** (рис. 24).

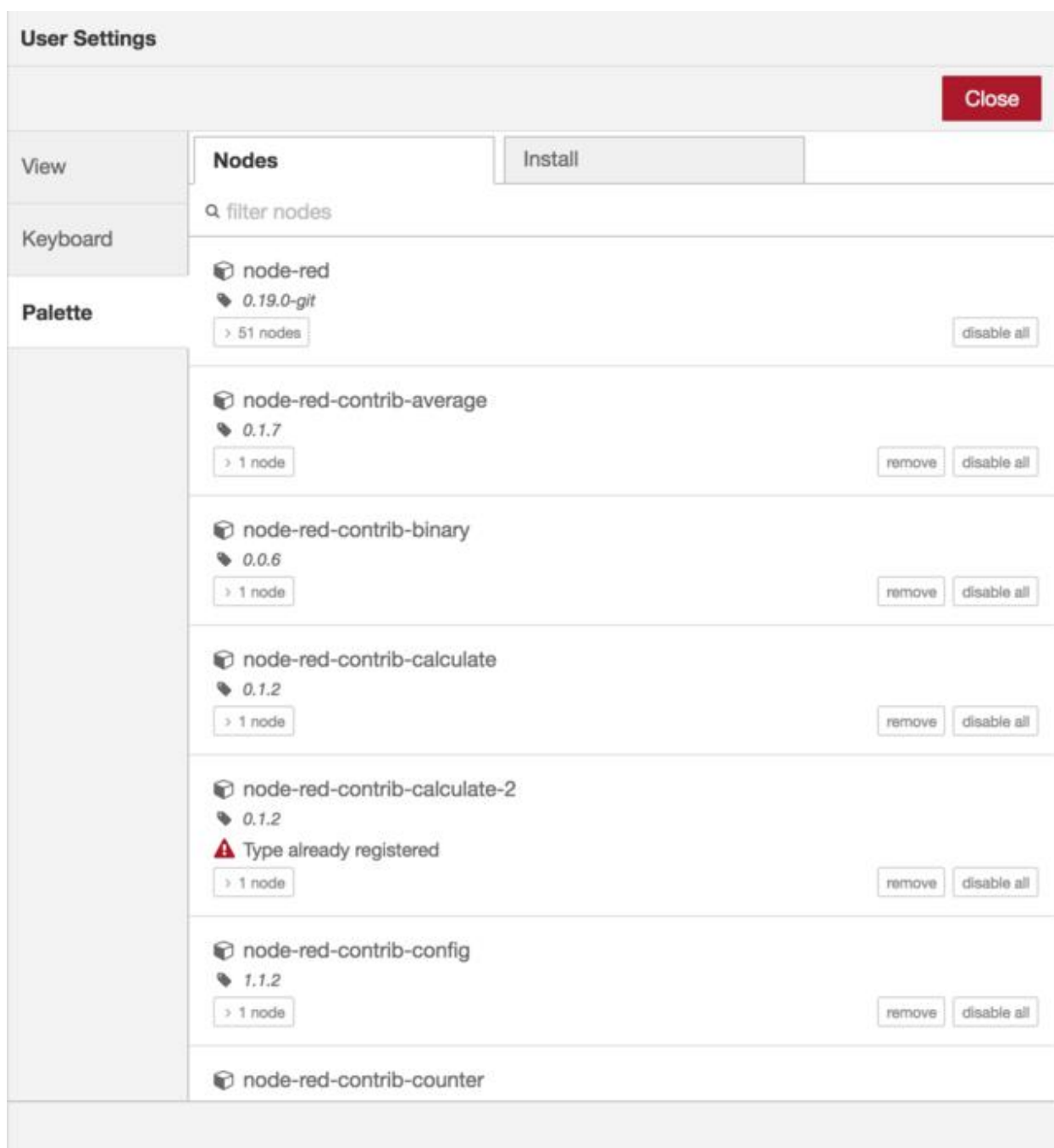


Рисунок 24. Меню «Manage palette» – вкладка «Nodes»

Меню «Manage palette» содержит две вкладки:

- «Nodes» – содержит модули, установленные в редактор
- «Install» – содержит модули, которые потенциально можно установить в редактор

1.4.11. Управление нодами

Полный список узлов, входящих в модуль, можно найти на вкладке «Nodes».

Также в функции этой вкладки входят удаление, обновление и отключение модуля. Используемый в потоке узел нельзя удалить или отключить.

Установка нод

На вкладке «Install» можно искать модули, а затем устанавливать их в редактор (рис. 25).

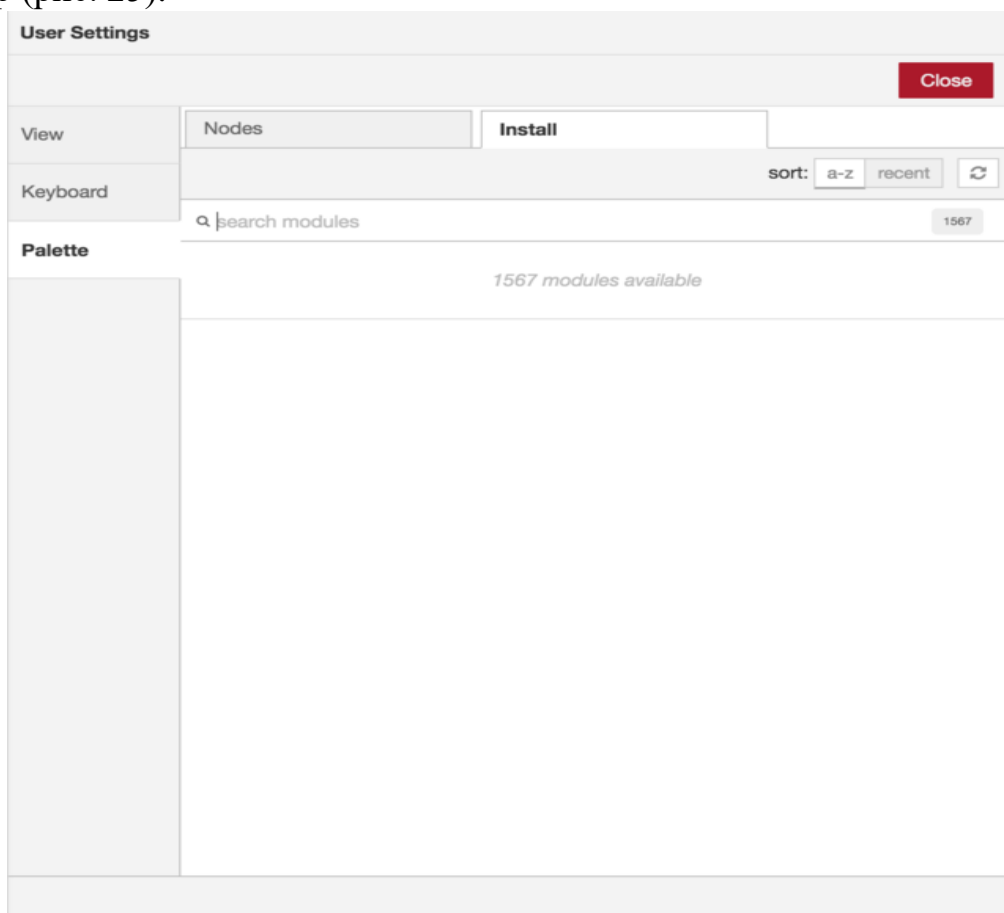


Рисунок 25. Меню «Manage palette» – вкладка «Install»

Для нахождения модуля необходимо написать его название в поисковую строку. На экране появится информация о модуле, включая то, когда он в последний раз был обновлен, а также ссылка на документацию. Чтобы установить модуль, кликните на кнопку «Install».

1.5. ФУНКЦИИ БОКОВОЙ ПАНЕЛИ

Боковая панель содержит несколько полезных инструментов редактора, а именно основную и дополнительную информацию об узле, сообщения, отправляемые узлам, информацию по управлению конфигурационными нодами и содержимое контекстов.

- Вкладка «Info» («Node information»)

- Вкладка «Debug» («Debug messages»)
- Вкладка «Config» («Configuration nodes»)
- Вкладка «Context» («Context data»)

У некоторых нод есть собственные вкладки в боковой панели – например, у нод модуля «node-red-dashboard».

Для открытия вкладки необходимо нажать по ее иконке или открыть выпадающее меню (кнопка с треугольником в правой верхней части боковой панели) (рис. 26).

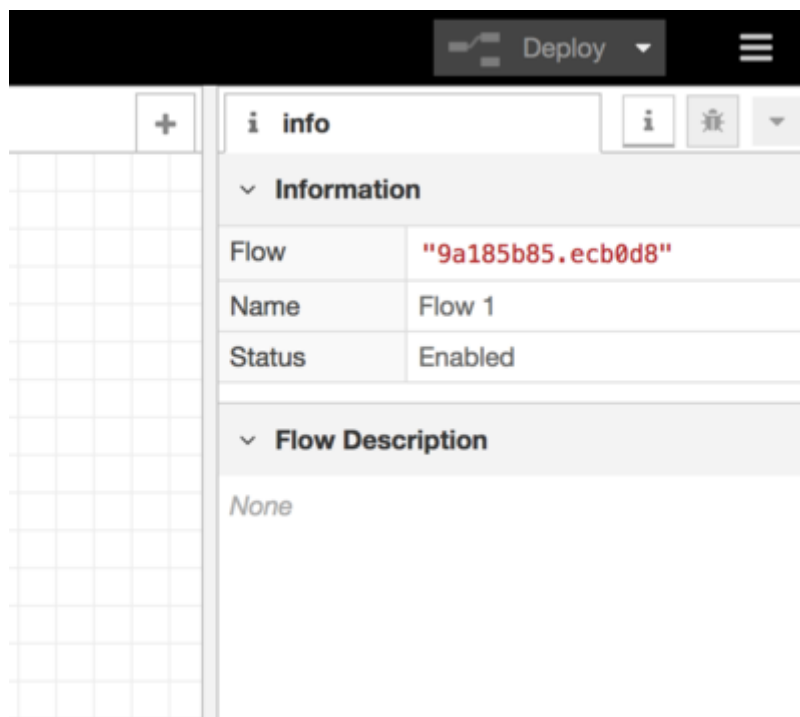


Рисунок 26. Боковая панель

Размер боковой панели регулируется. Можно нажать левой кнопкой мыши по ее левому краю и тянуть влево или вправо. Чтобы скрыть боковую панель нужно потянуть до максимума вправо. Чтобы показать боковую панель, необходимо нажать по кнопке с тремя полосками справа сверху, а затем на «View» > «Show sidebar» или просто нажать горячие клавиши **Ctrl + Пробел** / **⌘ + Пробел**.

Вкладка «Info»

На этой вкладке отображается информация об узле, выбранном в данный момент:

- свойства
- дополнительная информация

Для перемещения на вкладку «Info» необходимо нажать: **Ctrl + g** / **⌘ + g** и после этого **i** (рис. 27).

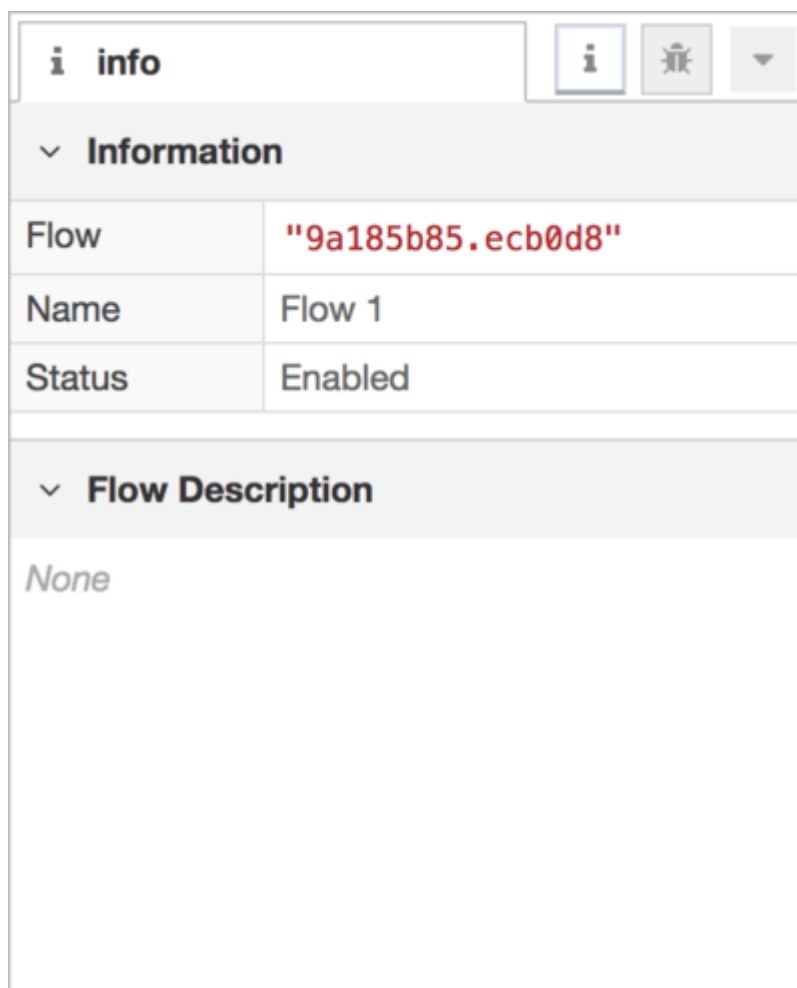


Рисунок 27. Вкладка «Info» на боковой панели

Если никакой ноды не выбрано, в этой вкладке показывается описание текущего потока (его можно поменять в меню редактирования свойств потока). Чтобы открыть его, кликните на кнопку с тремя полосками справа вверху, а затем на «Flows» > «Rename».

Вкладка «Debug»

На этой вкладке показываются сообщения, отправляемые нодам «debug» в потоке, а также некоторые сообщения-логи от редактора (рис. 28).

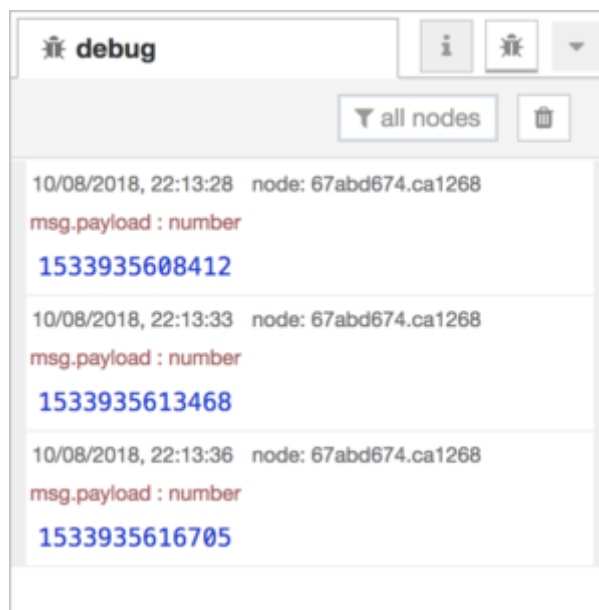


Рисунок 28. Вкладка «Debug» на боковой панели

По умолчанию вкладка «Debug» показывает все отладочные сообщения. Но их можно отфильтровать – для этого кликните на кнопку «all nodes», в результате чего откроется меню фильтрации отладочных сообщений (рис. 29).

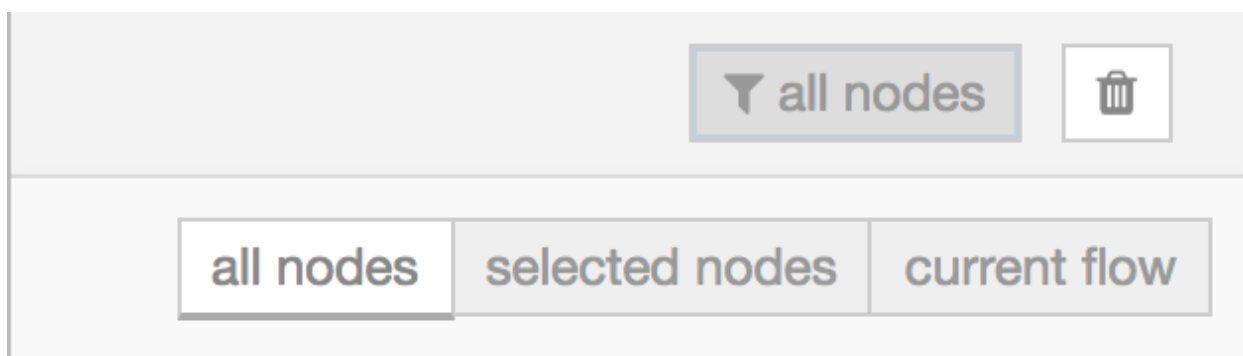


Рисунок 29. Меню фильтрации отладочных сообщений

Данное меню содержит три главные компоненты:

- «all nodes» – отображает все сообщения
- «selected nodes» – выбор узла «debug» из списка всех доступных узлов
- «current flow» – отображает сообщения от узлов, которые находятся в потоке, в данный момент открытом состоянии в рабочей области

Вкладка «Debug» может показать только 100 последних сообщений, включая скрытые. Если в потоке присутствуют ноды «debug», выдающие ненужную информацию, их лучше не отфильтровывать, а отключить, нажав на соответствующую кнопку в рабочей области.

Кроме того, содержимое вкладки «Debug» можно в любой момент очистить, нажав на кнопку с иконкой мусорной корзины.

Для отображения вкладки «Debug» в новом окне браузера необходимо кликнуть по кнопке с иконкой монитора в правой части экрана.

Вкладка «Config»

На этой вкладке отображены все конфигурационные ноды, сгруппированные по области их действия (рис. 30).

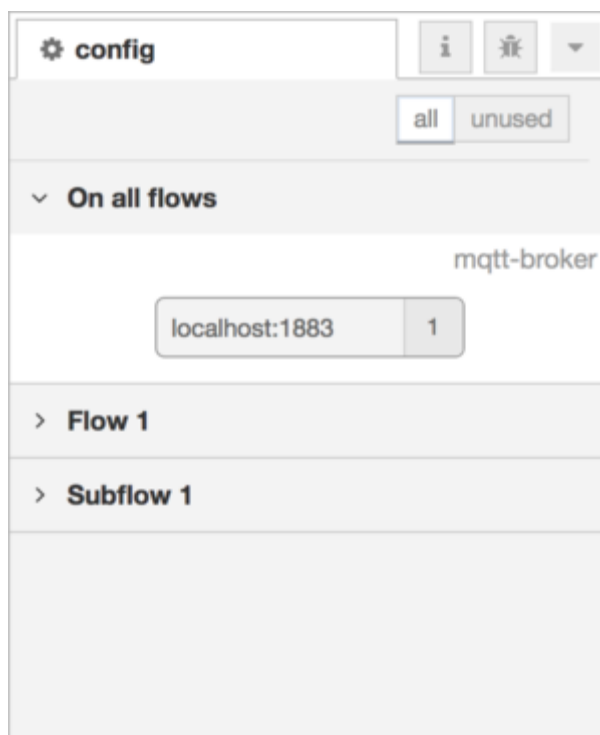


Рисунок 30. Вкладка «Config» на боковой панели

На вкладке отображаются тип и название конфигурационных узлов. Неиспользуемые в данный момент конфигурационные узлы выделяются пунктирной линией. Кнопка «unused» справа вверху позволяет увидеть неиспользуемые ноды. Двойное нажатие по меню конфигурационной ноды позволяет вносить в неё изменения.

Вкладка «Context»

В данной вкладке отображается содержимое разных контекстов (рис. 31).

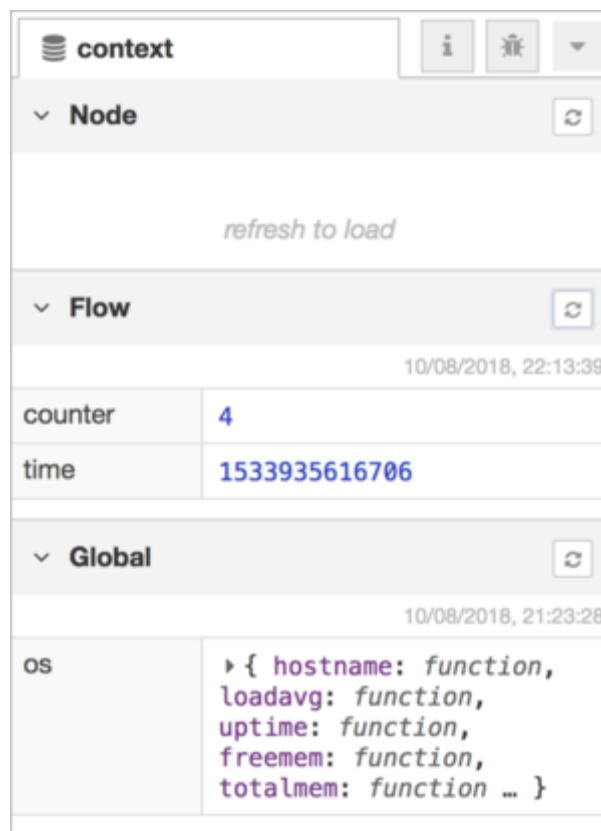


Рисунок 31. Вкладка «Context» на боковой панели

Для перехода на вкладку «Context» с помощью горячих клавиш, необходимо нажать: **Ctrl + g** / **⌘ + g** и после этого кнопку **x**. Вкладка «Context» включает в себя 3 основные части (по одной для каждого контекста). В разделе «Node» показан контекст ноды, с которой пользователь работает прямо сейчас. Для загрузки данных необходимо нажать на кнопку обновления справа вверху. После этого содержимое контекста станет доступным. В части под названием «Flow» отображается контекст текущего потока. Данные этого раздела загружаются автоматически при переключении на другой поток в рабочей области. В разделе «Global» отображается глобальный контекст. Данные здесь обновляются с каждой загрузкой редактора. Для отображения внесенных изменений необходимо нажать на кнопку обновления, соответствующую нужному контексту. Если провести курсором мыши над названием какого-либо контекстного свойства, покажется кнопка обновления, с помощью которой можно обновить конкретно это значение. Если провести курсором мыши над значением какого-либо контекстного свойства, покажется кнопка для копирования этого значения в системный буфер. Значение будет преобразовано в JSON-формат, поэтому вы сможете скопировать не всякое значение.

1.6. ЛАБОРАТОРНЫЕ РАБОТЫ

1.6.1. Лабораторная работа «Автоматизированная система управления температурой»

Для более точного получения информации об объекте управления рекомендуется устанавливать как можно больше датчиков, регистрирующих изменение параметров исследуемого объекта. Если говорить об измерении температуры, то можно разместить несколько датчиков температуры равномерно по площади помещения и считать температуру как среднее арифметическое показаний с этих датчиков [5].

Рассмотрим три датчика температуры. Выберем в ноде `wb-input`, после чего отобразится учёт информации с трёх каналов (рис. 32).

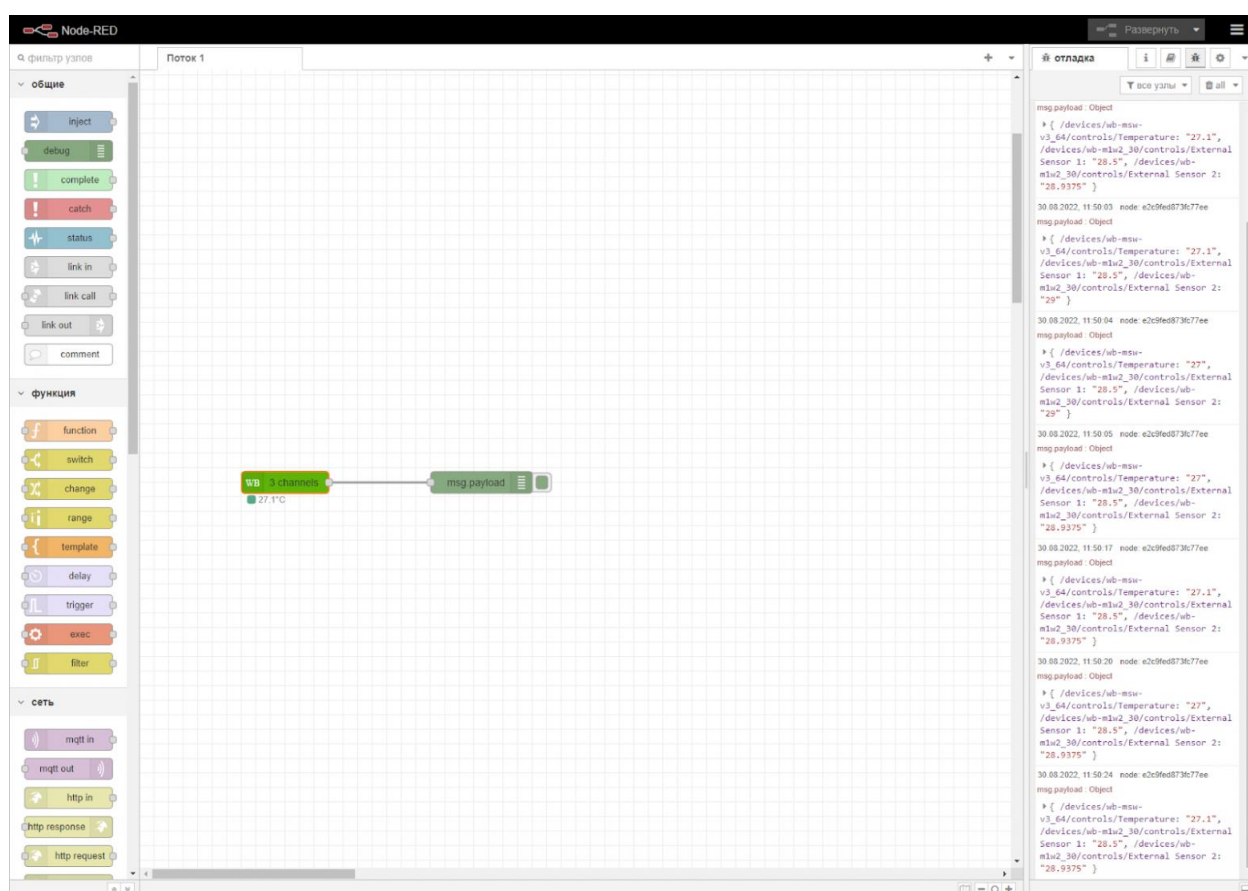


Рисунок 32. Узел `wb-input`

Следовательно, получаем информацию с трех датчиков. Для получения среднего значения добавлен нод `change`, которая меняет значения `msg.payload` и добавлю изменение на `msg.math.avg`, что соответствует среднему значению (рис. 33).

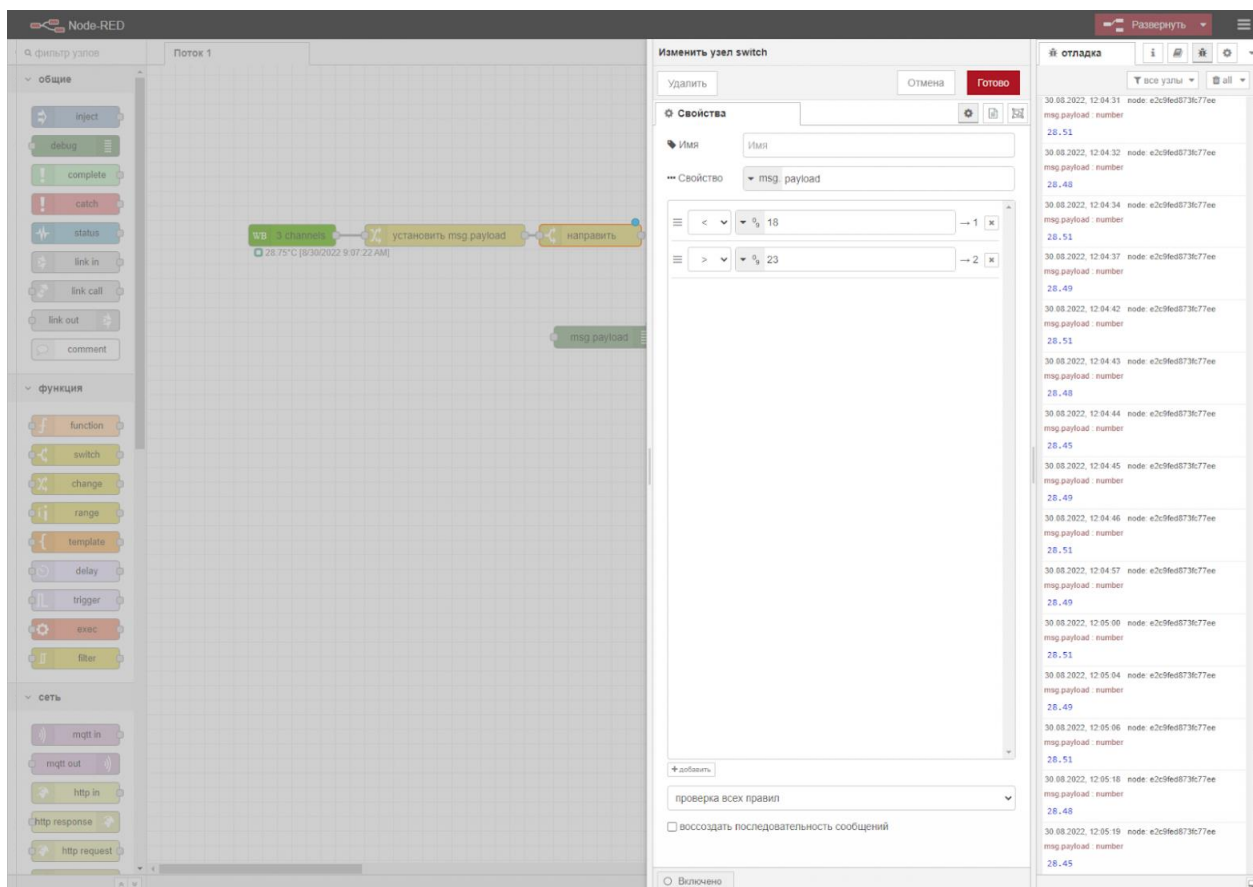


Рисунок 33. Свойства узла change

Далее нужно запрограммировать включение теплого пола, для управления которым используется канал K1 модуля реле WB-MR6C v.2.

Условие работы: Например, теплый пол включается при охлаждении помещения до 18 °С, а выключается – при нагреве до 23 °С. Необходимо выбрать тип `msg.payload` – численное значение, а не строку символов (меняется слева от поля значения).

Добавим в программу ноду ветвления Switch. Верхнее ветвление – включение нагрева, нижнее – выключение. Для включения теплого пола мы добавляем ноду `change` и меняем значение `msg.payload` (численное!) на 1. Для выключения добавляем такую же ноду `change` к нижнему ветвлению со значением 0 (рис. 34).

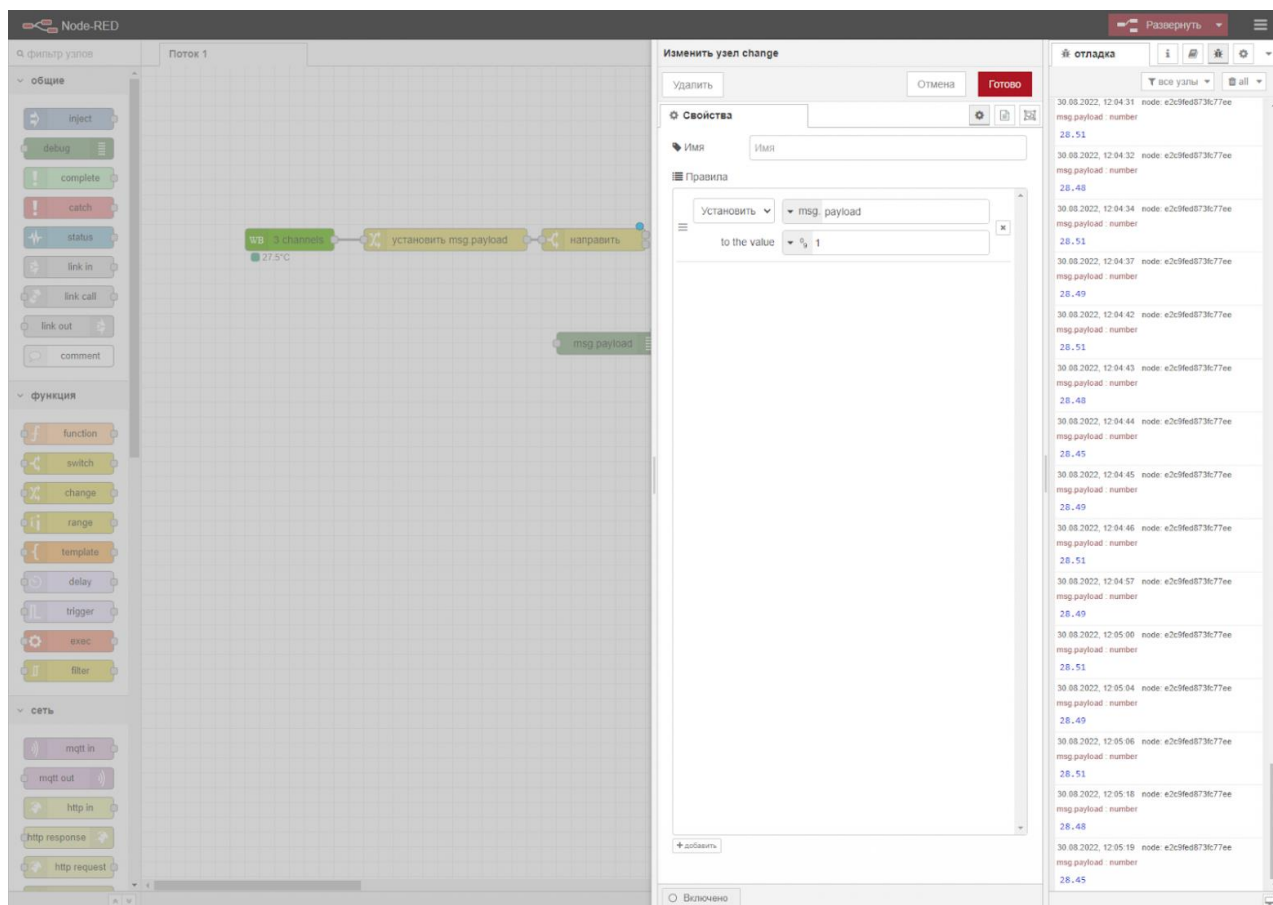


Рисунок 34. Изменение свойств switch

Мы успешно запрограммировали работу теплого пола. Далее рассмотрим работу батареи отопления, которая имеет терморегулятор. Считаем, что батарея может нагревать комнату пропорционально степени открытия клапана (заслонки) на ней. Пусть значение 0 соответствует полному закрытию клапана (заслонки), а 100 соответствует полному открытию. Используем ноду `wb-input`, в свойствах указываем нужный датчик температуры (в данном случае WB-MSW). Затем добавляем ноду ветвления `switch` и указываем три условия. Например, при температуре ниже 20 °C – заслонка открыта на 100 %, при температуре от 20 до 23 °C – на 70 %, а выше 23 °C – на 50 %.

В каждом случае мы воспользуемся нодой `change`, которая отправит на терморегулятор значение 100, 70 или 50. Не забывайте выставлять поле с числом, а не строкой! В нашем случае мы подключили не терморегулятор, а лампу накаливания, которую можно диммировать. Лампа подключена к первому каналу модуля WB-MDM3 (рис. 35).

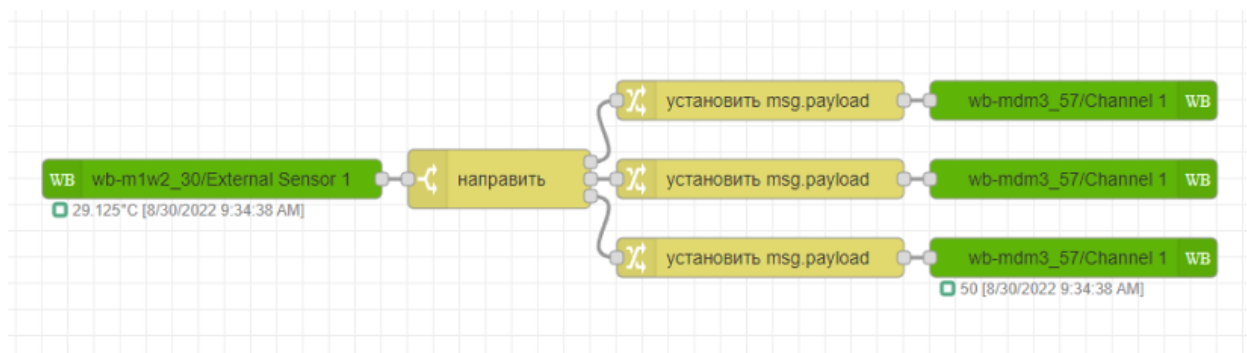


Рисунок 35 – Подключение лампы к первому каналу модуля WB-MDM3

Данный этап можно повторить для всех терморегуляторов в доме. Таким образом была настроена работа теплого пола и батареи.

1.6.2. Лабораторная работа «Автоматизированная система управления освещением»

Реализуем два варианта работы. Первый вариант: освещение дворовой территории. Свет будет включаться автоматически по показаниям датчика освещения или по астрономическому времени восхода и заката. Второй вариант – освещение внутри помещения. Белый свет горит с 7 утра до 23 вечера, а в остальное время будет гореть красное дежурное освещение.

Освещение дворовой территории проще всего настроить по астрономическому времени восхода и заката. Здесь установлена дополнительная ноду Big Timer ([node-red-contrib-bigtimer](#)) через главное меню – управление палитрой – установить (рис. 36).

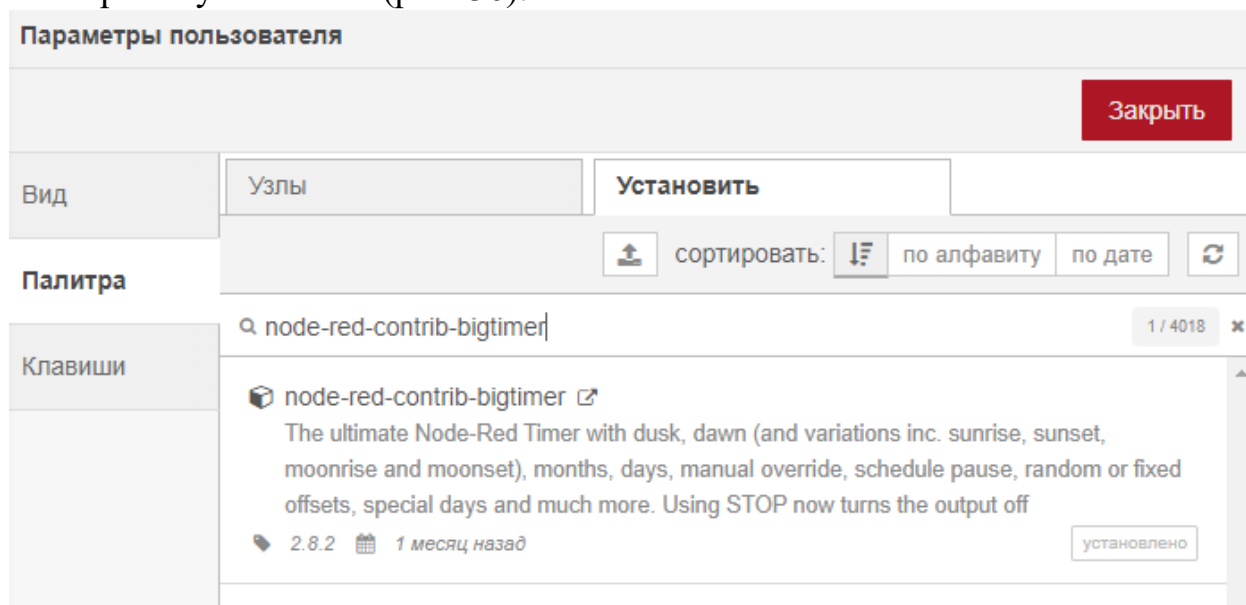


Рисунок 36. Установка узла Big Time

В настройках ноды важно указать широту (Latitude) и долготу (Longitude) места настройки освещения. В верхней строчке On Time выставляется время включения ноды, в нашем случае от заката (Sunset) до восхода (Sunrise). Затем достаточно подключиться к среднему выходу ноды Big Timer, во включенном состоянии (ночью) нода будет каждую минуту отсылать 1, в выключенном (от восхода до заката) – 0 (рис. 37).

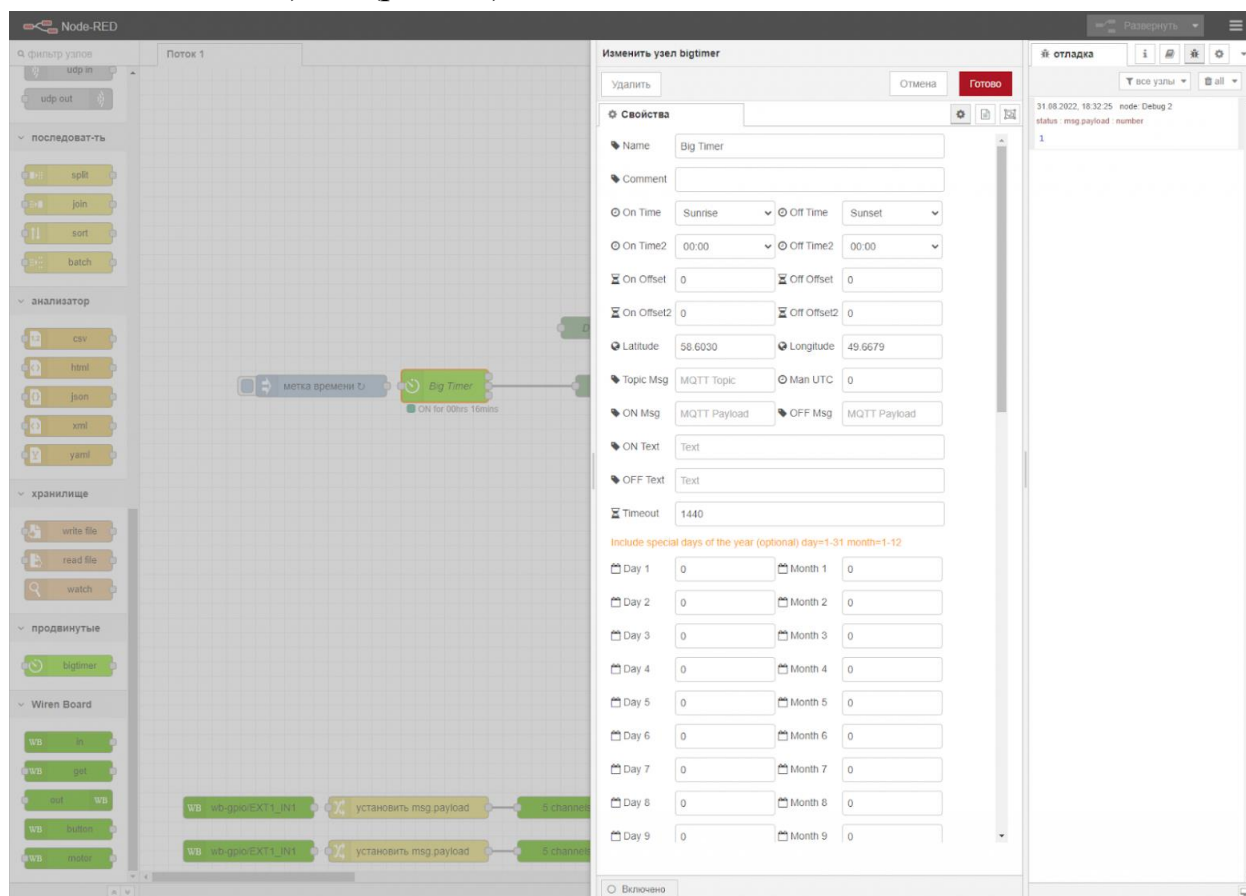


Рисунок 37. Свойства узла big time

Больше ничего в Big Timer настраивать не надо. Затем просто добавляем ноду wb-out с выключателем освещения, в нашем случае – лампочка K1 на модуле WB-MDM3 (рис. 38).

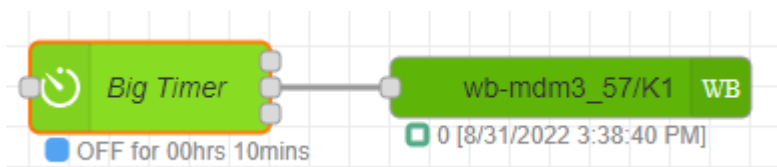


Рисунок 38. Размещение Bigtime на рабочей области

Более сложный вариант – настройка освещения по показаниям датчика освещения. Здесь воспользуемся датчиком WB-MSW v.3. У датчика нам нужен параметр освещенности Illuminance, его получим через ноду wb-input (рис. 39).

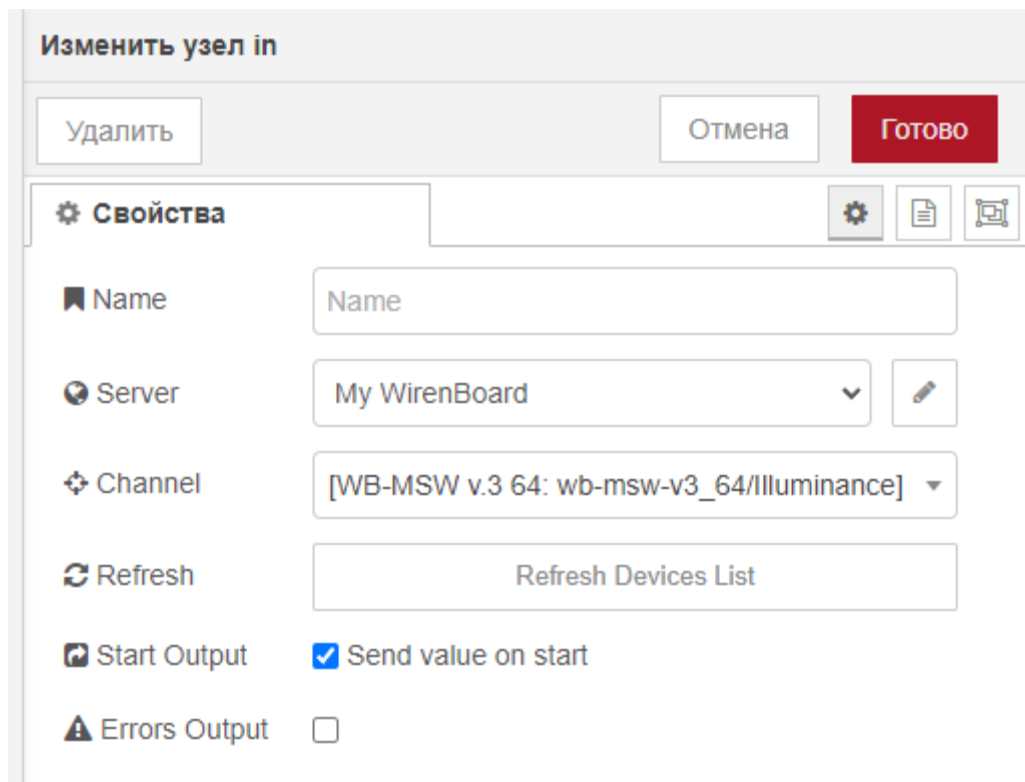


Рисунок 39. Свойства узла wb-input

Затем добавим ноду ветвления Switch, при освещенности меньше 10 будет включаться свет (нода change на 1), при освещенности больше 30 – выключаться свет (нода change на 0). Необходимо указать численный тип (рис. 40).

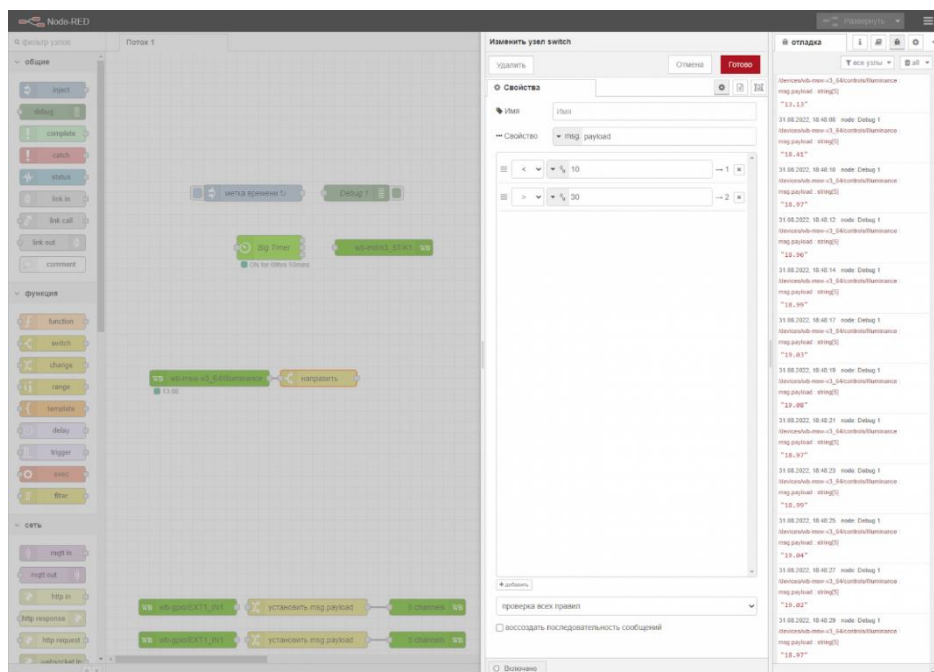


Рисунок 40. Свойства узла Switch

Далее добавляем ноды wb-out с включением света и изменение значения 0 или 1 через ноду change. В нашем случае – лампочка K1 на модуле WB-MDM3 (рис. 41).

Рисунок 41. Размещение модуля wb-mdm3

Для настройки освещения в помещении используем универсальную ноду Big Timer, для которой выставлю время включения с 7:00 утра до 23:00 вечера. При выставлении времени следует учитывать часовой пояс (рис. 42).

Рисунок 42. Свойства узла BigTime

Далее настраиваем ноду ветвления. Верхний путь будет выбран при 1 (днем), нижний – при 0 (ночью). Днем нужно включить ленту RGB и выставить белый свет. Ночью включается лента RGB и устанавливается красный свет. Включение ленты выполняется через ноду `wb-out` и устройство `WB-LED (WB-MRGBW-D)`, значение 1 соответствует включению, 0 – выключению (рис. 43-44).

Рисунок 43. Свойство узла out

Рисунок 44. Свойство узла change

Также через ноду wb-out задается оттенок RGB Palette. Здесь строка 255;255;255 – это белый цвет, а 255;0;0 – красный. Данные значения будут выставлять ноды change, после чего передавать его на устройство RGB Palette в ноду wb-out (рис. 45).

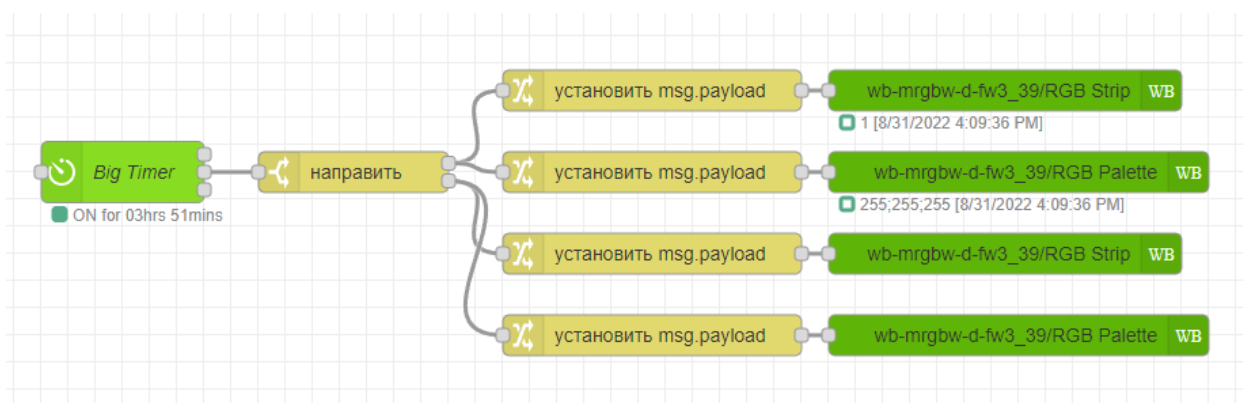


Рисунок 45. Размещение узлов и установка цветов

Настройка завершена.

1.6.3. Лабораторная работа «Автоматизированная система управления освещением в помещении»

Цель работы – автоматизация освещения внутри помещения. В качестве помещения выбран санузел. Источниками данных выступают датчик движения и датчик открытия двери. При открытии двери и наличии движения свет включается по таймеру на две минуты. Каждый раз, когда датчик фиксирует движение, таймер обновляется. При закрытой двери и наличии движения свет включается по таймеру на пятнадцать минут. При открытой двери и отсутствии движения свет горит в течение двух минут (по таймеру). Если дверь закрыта и отсутствует движение свет горит в течение трёх минут по таймеру. При возникновении движения таймер обнуляется.

Введём ноду Inject. Она будет каждую секунду опрашивать датчики. В качестве датчика двери используется тумблер QF3 тестового чемодана. Открытое состояние тумблера будет означать открытую дверь, закрытое – закрытую. В вашем случае наверняка будет использовать геркон с теми же двумя состояниями. Добавляем ноду wb.get для опроса нашего “геркона”, роль которого выполняет тумблер QF3. Затем выполняется ветвление. Верхняя ветка означает открытую дверь, нижняя – закрытую (рис. 46).

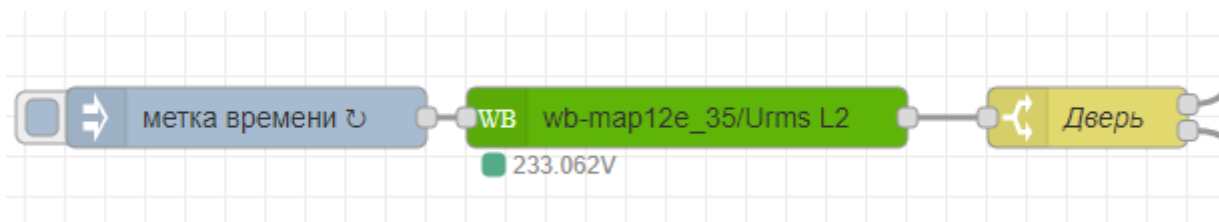


Рисунок 46. Первая половина цепи работы модуля

Затем нам нужна информация о движении. Для этого снимаются показания с датчика движения модуля WB-MSW v.3 через ноду `wb.get`. Если он выдает больше 20, то движение есть (верхняя ветка), если меньше 20 – движения нет (нижняя ветка). Чтобы предусмотреть все четыре возможных варианта, понадобятся две ноды `wb.get` по двум веткам, следующим от датчика двери (рис. 47).

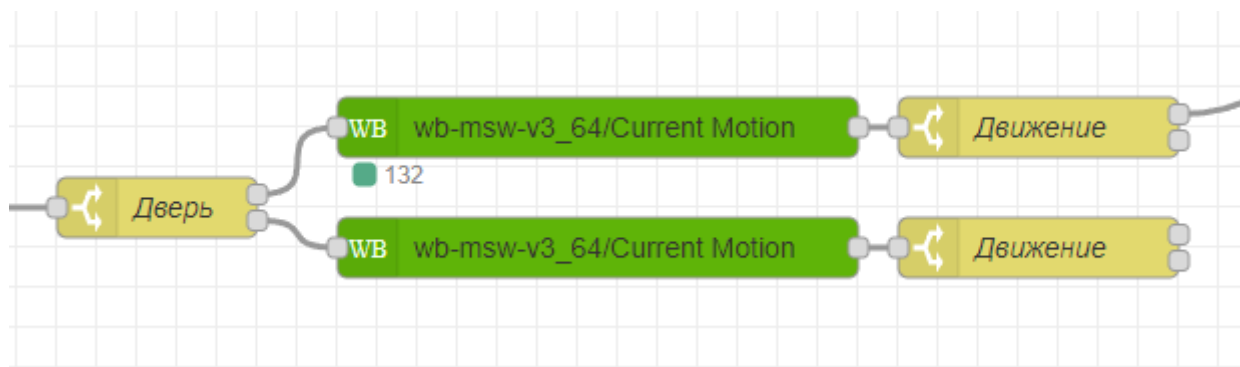


Рисунок 47. Разветвление положений двери

Самая верхняя ветка: дверь открыта, движение есть (зашли в ванную помыть руки). Здесь добавлена нода `Trigger`, которая будет отправлять 1 на лампу и ждать 2 минуты, после чего отправлять на лампу 0. При этом поставлена галочка «продлить при поступлении нового сообщения». Она позволит сбросить триггер при поступлении информации о движении, то есть увеличить время ожидания еще на две минуты (рис. 48).

Изменить узел trigger

Удалить

Отмена

Готово

⚙️ Свойства

⚙️ 📄 🖨️

Отправить

▼ 0₉ 1

затем

ждать ▼

2 мин ▼

☒ продлить при поступлении нового сообщения
 ☐ заменить задержку через msg.delay

затем
отправить

▼ 0₉ 0

☐ отправить второе сообщение на отдельный выход

Сбрасывать триггер, если:

- установлен msg.reset
- msg.payload равен

Обрабатывать

все сообщен ▼

🏷️ Имя

Рисунок 48. Свойства узла trigger

Остается подключить к триггеру лампу, и верхняя ветка готова. Следующая ветка: дверь открыта, движения нет (уходите из ванной не закрыв дверь). Здесь добавлена задержка 2 минуты с помощью ноды Delay. После этого опрашивается датчик движения, и если движения не будет, то лампа выключается. Повторный опрос нужен, чтобы избежать ситуации с

выключением света по верхней ветке, поскольку информация по нижней ветке будет идти с 2-минутной задержкой.

То есть две минуты назад движения не было, сработает нижняя ветка, даже если прямо сейчас движение появилось (рис. 49).

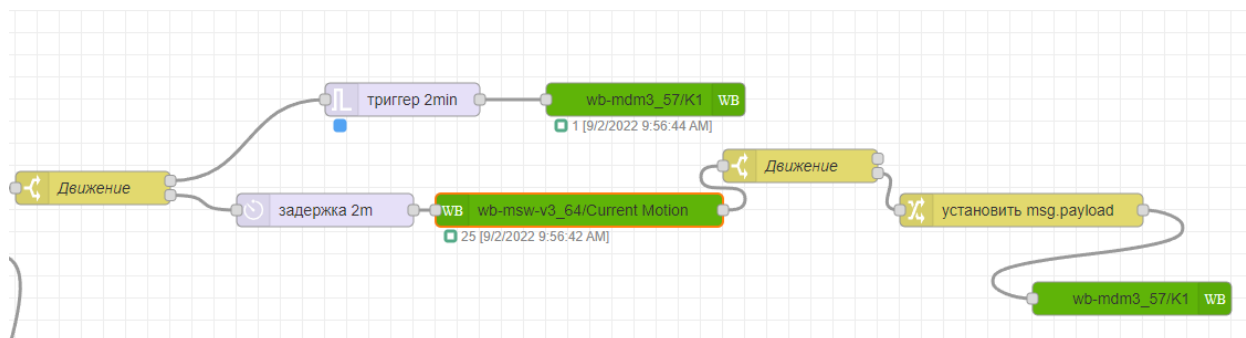


Рисунок 49. Реализация «открытой двери без движения»

Следующий вариант работы: дверь закрыта, движение есть (зашли в ванную помыться) - таймер на 15 минут. Здесь использована нода Trigger, но без продления при получении информации о движении. Все же 15 минут будет достаточно для всего. Через 15 минут свет выключается (рис. 50).

Рисунок 50. Свойства узла trigger для закрытой двери

Если же человек выйдет раньше, то здесь поможет четвертый сценарий, выключающий свет через 3 минуты после последнего определения движения. Меньше ставить не очень разумно, поскольку человек за шторкой может не генерировать движение каждую минуту, и здесь логика пойдет по верхней ветке с задержкой 15 минут (рис. 51).

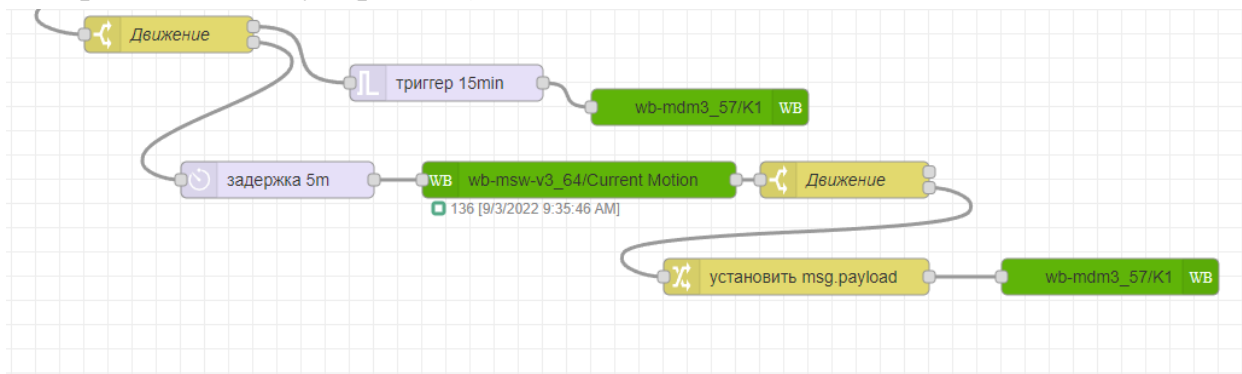


Рисунок 51. Реализация сценария выключения света при отсутствии движения

Однако существует особенность: если триггер сработал, но свет выключился по второму пути (человек вышел, потом снова зашел), то триггер может находиться в состоянии ожидания. Поэтому добавлен включение света по определению движения в верхние ветвления, чтобы избежать подобных ситуаций.

В результате мы получили следующую схему (рис. 52).

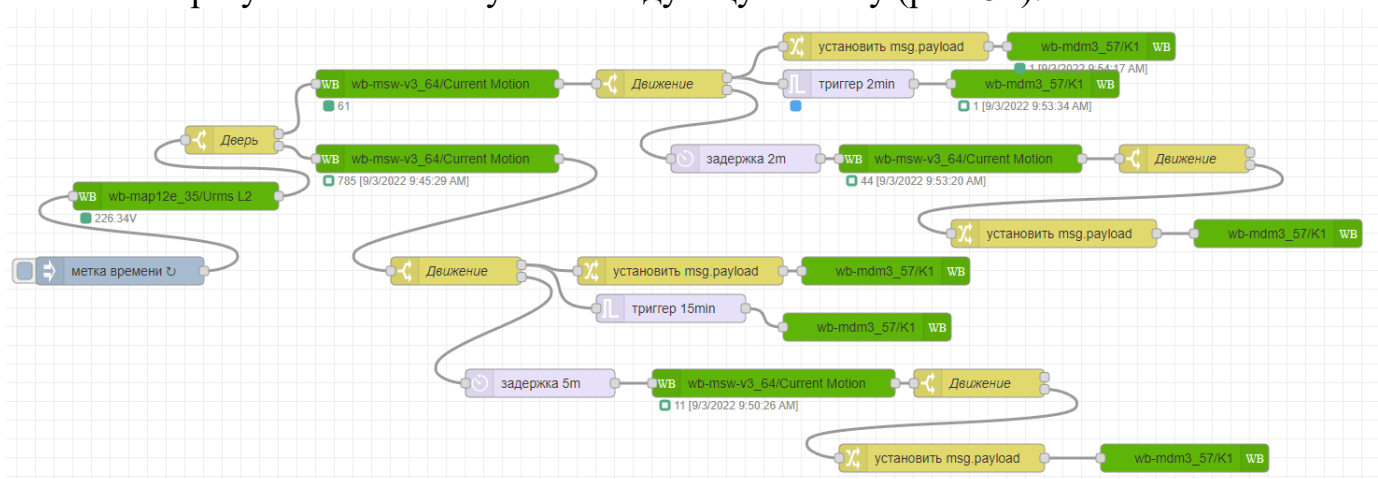


Рисунок 52. Итоговая схема работы сценария «Темной комнаты»

Со своей работой схема справляется, но ее можно оптимизировать или сделать эффективнее.

Часть II. ПРОГРАММИРОВАНИЕ В OpenHAB

2.1. ИНТЕРФЕЙС OpenHAB

OpenHAB – это программа, в которой реализуется электронная связь с интеллектуальными устройствами, пользователь может осуществить определённый набор команд, веб-страницам предоставляется доступ к определённой информации пользователя. В данной программе существует общий интерфейс, а также инструменты для взаимодействия со всеми устройствами. Для этого openHAB сегментирует и разделяет определенные функции и операции [6]. В таблице 1 дано общее описание наиболее важных концепций.

Таблица 1. Концепции openhab

Концепции	Значение
Привязки	Являются компонентом OpenHAB, который обеспечивает интерфейс для электронного взаимодействия с устройствами.
Вещи	Являются первым сгенерированным OpenHAB представлением устройств пользователя
Каналы	Представляют собой соединение OpenHAB (программное обеспечение) между «Вещами» и «Предметами»
Предметы	Представляют собой сгенерированное OpenHAB представление информации об устройствах
Правила	Которые выполняют автоматические действия (в простейшей форме: если произойдет «это», OpenHAB сделает «это»)

Концепции	Значение
Карта сайта	Это созданный OpenHAB пользовательский интерфейс (веб-сайт), который представляет информацию и позволяет взаимодействовать

Каналы

Каналы представляют собой логические взаимосвязи между Вещью и Предметом. С помощью Каналов определяется как Вещь может взаимодействовать с Предметом (и наоборот). Каналы формируются при определении вашей Вещи.

При формировании Вещи происходит образование Канала, к которому привязывается данная Вещь. Это способствует передаче информации от вещи к предмету (и наоборот) в OpenHAB.

Привязки

Привязки (Bindings) — это программные пакеты, которые устанавливаются пользователем в openHAB. Основная цель Bindings — установка соединения между устройством и вещью. Привязки организуют работу «умных» устройств и переводят все команды в openHAB и обратно.

Существует множество привязок для поддержки максимально возможного количества устройств разных производителей. Новые привязки регулярно добавляются по мере того, как разработчики интегрируют в openHAB больше устройств.

Для каждой привязки разработаны инструкции и примеры, которые включают руководство по настройке (если таковые имеются) самой привязки, определение вещей, поддерживаемых этой привязкой, и каналов, которые эти вещи предоставляют. В большинстве случаев описание также содержит полностью проработанный пример, включающий определение Вещей и их Каналов, Предметов, связанных с этими Каналами, и использование этих Предметов в карте сайта.

Правила

OpenHAB имеет высокоуровневый интегрированный, лёгкий, но мощный движок выполнения правил. «Правила» применяют для автоматизации процессов. Каждое правило запускается по определённому условию и выполняет сценарий, исполняющий задачи разного рода, такие как управление освещением, включение таймера, работа со счётчиком, управление температурным режимом.

Правила расположены в папке \$OPENHAB_CONF/rules. Демонстрационный комплект включает в себя файл demo.rules, в котором есть несколько базовых примеров. Задача может содержать несколько правил. Все правила, сохранённые в одном файле, будут выполняться в общем контексте, то есть они могут общаться и обмениваться переменными друг с другом. Поэтому имеет смысл иметь разные файлы правил для разных вариантов использования или категорий.

Определение на основе пользовательского интерфейса

Правила могут создаваться и редактироваться через пользовательский интерфейс UI. Редактор находится в разделе Settings -> Rules. Необходимо кликнуть на иконку «+» чтобы добавить правило, описать его название и поставить условие срабатывания.

В примере правило запускается при запуске openHAB для определения окружения. Если добавить второй триггер в раздел When, то будут учитываться срабатывания обоих триггеров (логическое ИЛИ). Если добавить дополнительное действие в Then, то оно будет выполняться последовательно. Если же добавить дополнительное условие в раздел «But only if», то оно будет работать совместно с остальными (рис. 53).

Unique ID 86e99503b8

Name Тестовое правило

Description

Reorder

When

When the system has reached start level 100
This triggers the rule if a given start level is reached by the system.

Add Trigger

Then

execute a given script
Allows the execution of a user-defined script.

Add Action

But only if

it is a weekend
Checks if the current day is on the weekend.

Add Condition

Tags

Semantic Class None

Add tag

Remove Rule

Рисунок 53. Меню тестового правила

При нажатии кнопки Show All открывается доступ к другим вариантам работы. За кнопкой. Show All зачастую скрываются очень полезные вещи. Например, при выборе триггера по элементу становятся доступны все элементы, а не только входящие в семантическую модель, как в базовом варианте выбора.

Кликните Add Action и нажмите Run Script. Выберите Rule DSL и вводите правила по образу, описанному в данном руководстве (рис. 54).

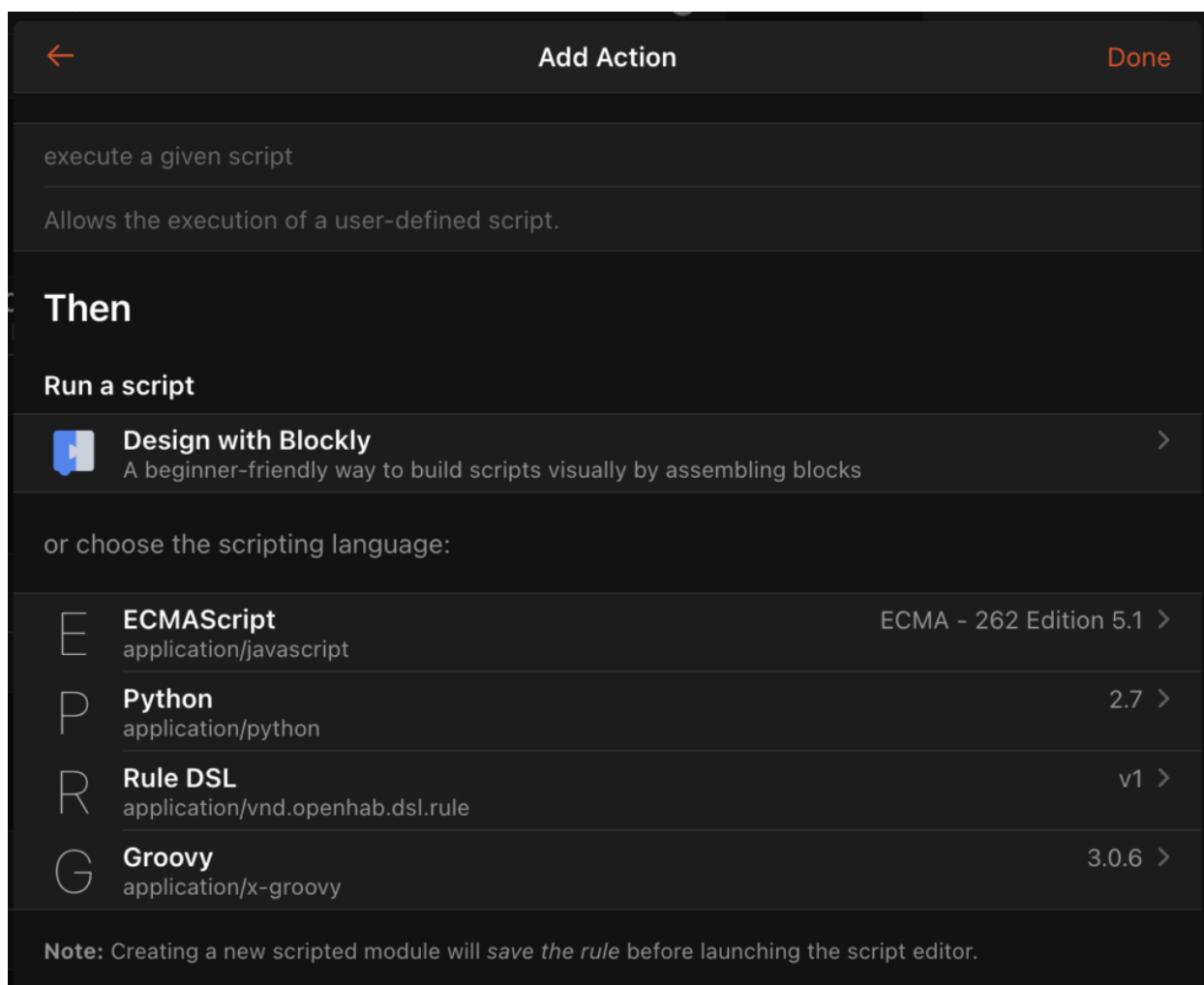


Рисунок 54. Меню добавления скрипта

Поддержка средств разработки (IDE)

Расширение openHAB VS Code Extension предлагает поддержку построения правил. Он включает в себя проверку синтаксиса и раскраску, проверку с маркерами ошибок, подсказки в контенте (Ctrl+Space), включая шаблоны и т. д. Это делает создание правил очень легким.

2.2. Синтаксис OpenHAB

Синтаксис правил основан на Xbase и в результате он во многом схож с Xtend, который также построен поверх Xbase. Файл правил — это текстовый файл, имеющий следующие разделы:

1. Импорт
2. Описание переменных
3. Правила

Раздел «Импорт» содержит инструкцию импорта. Как и в Java, импортированные типы становятся доступными без необходимости использования для них полного имени. Например: `import java.net.URI`

Несколько модулей импортированы по умолчанию, так что классы не требуется импортировать отдельно, как показано в таблице 2.

Таблица 2. Код импорта по умолчанию

```
org.openhab.core.items
org.openhab.core.persistence
org.openhab.core.library.types
org.openhab.core.library.items
org.openhab.model.script.actions
```

Часть под названием «**Описание переменных**» можно использовать для обозначения переменных, которые должны быть доступны для всех правил в этом файле. Пользователь задаёт переменные с начальными значениями или без них, как показано в таблице 3.

Таблица 3. Код объявления переменных

```
// a variable with an initial value. Note that the variable type is
automatically inferred
var counter = 0

// a read-only value, again the type is automatically inferred
val msg = "Здесь был Вася"

// an uninitialized variable where we have to provide the type (as it
cannot be inferred from an initial value)
var Number x
```

Часть под названием **Правила** содержит список правил. Каждое правило имеет определённый синтаксис. Данный синтаксис представлен в таблице 3.

Таблица 3. Код описания правила

```
rule "<RULE_NAME>"
when
    <TRIGGER_CONDITION> [or <TRIGGER_CONDITION2> [or ...]]
then
    <SCRIPT_BLOCK>
end
```

<RULE_NAME> — Каждое правило должно иметь уникальное имя. Имя пишется в кавычках. Рекомендуется использовать осмысленные названия правил.

<TRIGGER_CONDITION> — событие, при выполнении которого реализуется логика правила. Правило выполняется в ответ на одно или несколько условий триггера. Несколько условий разделяются ключевым словом «or».

<SCRIPT_BLOCK> — Содержит логику, которая должна быть выполнена при выполнении условия триггера, подробности о его синтаксисе см. в разделе сценария.

2.2.1. Триггеры правил

Прежде чем правило начнёт выполняться его необходимо активировать. Существует несколько категорий триггеров (активаторов) правил:

1. Триггеры по **элементам** (Items): реагируют на события на шине событий openHAB, т.е. команды и обновления статуса для элементов;
2. Триггеры **по группе**: реагируют на события на шине событий openHAB относящиеся к элементам входящим в указанную группу;
3. Триггеры по **времени**: срабатывают в определенное время, например, в полночь, каждый час и т. д.;
4. **Системные** триггеры: реагируют на определенные системные состояния;
5. Триггеры по **вещам** (Things): реагируют на статус вещи, т.е. переходы с ONLINE на OFFLINE и наоборот.

Триггеры по элементам

Необходимо считать показание статуса того или иного элемента и выполнить определённую команду в соответствии со статусом. Можно реагировать только на конкретную команду/статус или любое изменение. Синтаксис для всех этих случаев описан в таблице 4 (части в квадратных скобках являются необязательными).

Таблица 4. Код описания команд/статусов

Item <item> received command [<command>]
Item <item> received update [<state>]
Item <item> changed [from <state>] [to <state>]

Часть полей можно оставить пустыми при формировании правил через пользовательский интерфейс. Далее значение можно сравнить с числом в логическом блоке условий «But only if» ниже в форме. Эта функция может быть использована для реакции на выход за диапазон допустимых значений параметров с датчиков, например, превышение освещённости, температуры, расхода, уровня или влажность. Так же бывает полезно анализировать появление конкретного состояния элемента игнорируя его предыдущее состояние (для этого можно использовать конструкцию `changed to` или просто не указывать лишнее состояние).

Функции `command` и `update` различаются между собой следующим образом: `command` — управление исполнительными устройствами (освещением, воротами, жалюзи и т.п.), `update` — изменение состояния свойственное датчикам. При использовании срабатывания по `received command` правило может начать исполняться до того, как элемент поменяет своё состояние. Поэтому, если правилу важно знать, какая конкретно команда была отправлена, следует использовать неявную переменную `receivedCommand` вместо `<ItemName>.state`.

Триггеры по группе

Как и в случае с триггерами на основе событий, описанными выше, вы можете прослушивать команды, обновления статуса или изменения статуса для членов данной группы. Вы также можете решить, хотите ли вы реагировать только на определенную команду / статус или любое изменение. Все неявные переменные заполняются с помощью элемента, вызвавшего событие. Неявная переменная `triggeringItem` заполняется элементом, который вызвал срабатывание правила. Подробное описание приведено в таблице 5.

Таблица 5. Код срабатывания правил

```
Member of <group> received command [<command>]
Member of <group> received update [<state>]
Member of <group> changed [from <state>] [to <state>]
```

Триггер по группе работает только с элементами, являющимися прямыми участниками группы. Он не реагирует на элементы вложенных групп. Так же как в случае с триггерами по событию при использовании `received command` правило может начать выполняться до того, как поменяется состояние. Если правилу необходимо знать, какая была отправлена команда, то нужно использовать неявную переменную `receivedCommand` вместо `<ItemName>.state`.

Триггеры по времени

Предопределенные выражения применяются для таймеров. Также можно использовать выражение cron:

Time is midnight

Time is noon

Time cron "<выражение cron>"

Выражение cron имеет форму шести или семи полей:

Секунды

Минуты

Часы

День месяца

Месяц

День недели

Год (необязательное поле)

Можно использовать FreeFormatter.com для создания выражений cron.

Системные триггеры

Описание системного триггера дано в таблице 6.

Таблица 6. Описание триггеров

Триггер	Описание
System started	System started запускается при запуске openHAB. В openHAB версии 2 System started также срабатывает после изменения файла правил, содержащего триггер System started, или после изменения элементов в файле .items.

Вы можете использовать системный триггер 'System started' для установки начальных значений если они ещё не установлены. Пример описан в таблице 7.

Таблица 7. Код реализации системного триггера

```
rule "Инициализация Speedtest"
when
    System started
then
    createTimer(now.plusSeconds(30), [|
        if      (Speedtest_Summary.state      ==      NULL      ||
Speedtest_Summary.state                      ==                      "")
Speedtest_Summary.postUpdate("unknown")
    |])
end
```

Триггеры по вещам

Правила позволяют совершать действия, основанные на обновлениях статуса или изменениях статуса описанных вещей (Things). Пользователь сам определяет необходимость отслеживать только определенный статус или любой статус. Синтаксис для этих случаев (части в квадратных скобках необязательны) представлен в таблице 8.

Таблица 8. Код отслеживания статуса вещей

```
Thing <thingUID> received update [<status>]
Thing <thingUID> changed [from <status>] [to <status>]
```

Статус, используемый в триггере и сценарии, представляет собой строку (без кавычек). Возможные значения статуса представлены в Thing Status. Для приобретения статуса объекта в сценарии необходимо зайти в «Действие по статусу объекта». Идентификатор thingUID присваивается Вещи вручную в конфигурации или автоматически во время автоматического нахождения. Вы можете найти его в пользовательском интерфейсе или с удаленной консоли Karaf. Например, одно устройство z-wave может быть «zwave: device: c5155aa4: node14». Необходимо помещать thingUID в кавычки если оно содержит специальные символы.

Триггеры по каналам

Триггерные каналы являются частью надстроек. Канал запуска обладает информацией о дискретных событиях, но не имеет непрерывную информацию о состоянии.

На основе правил могут совершаться действия по примеру триггерных событий, генерируемых этими каналами. Вы можете запрограммировать получение информации с определенного или абсолютно любого триггера, который предоставляет канал. Синтаксис для этих случаев выглядит следующим образом: необходимо помещать `triggerChannel` в кавычки, если он содержит специальные символы.

Channel "<triggerChannel>" triggered [<triggerEvent>]

`triggerChannel` — это идентификатор конкретного канала.

Информацию о создаваемых каналах можно найти в документации надстройки (Add-on). Не существует списка возможных значений `triggerEvent`, так как это зависит от конкретной реализации надстройки. Если необходимо знать какое именно событие произошло необходимо использовать неявную переменную `receivedEvent`, как показано в таблице 9.

Таблица 9. Код реализации триггера канала

```
rule "Начинаем пробуждение на рассвете"
when
    Channel "astro:sun:home:rise#event" triggered START
then
    ...
end
```

2.2.2. Сценарии

Язык сценариев аналогичен языку Xtend. Синтаксис аналогичен Java, но имеет много полезных функций, позволяющих писать краткий код. Это особенно эффективно при работе с коллекциями. Отличительной особенностью является то, что сценарии не надо компилировать. Они могут быть интерпретированы во время выполнения.

Программа `openHAB` предоставляет доступ: ко всем определенным элементам, чтобы пользователь мог получить к ним доступ по имени, ко всем перечисленным состояниям или командам, например, `ON`, `OFF`, `DOWN`, `INCREASE` и т. д., ко всем стандартным действиям.

Комбинируя эти функции достаточно просто написать программный код, представленный в таблице 10.

Таблица 10. Код изменение статуса командами

```
if (Temperature.state < 20) {
    Heating.sendCommand(ON)
}
```

Управление состояниями элементов

Правила в openHAB необходимы для того, чтобы пользователь мог контролировать и изменять состояние объекта [7]. Например, управлять системой освещённости помещения при определенных условиях. Следующие команды могут изменить значение или состояние элемента в рамках правил:

- `MyItem.postUpdate(<new_state>)` — позволяет изменить статус элемента, не вызывая каких-либо неявных действий. Может использоваться для отражения изменений, которые могут быть вызваны другими способами.
- `MyItem.sendCommand(<new_state>)` — позволяет пользователю изменить статус элемента и осуществить какое-либо действие, например, отправку команды на связанное устройство / привязку.

По отношению к событийному правилу управление в сценариях запускает команды `postUpdate` и `sendCommand`.

Осторожно: в большинстве случаев правило с триггером `received update` срабатывает впоследствии отправки команды `sendCommand` в следующих ситуациях:

- openHAB автоматически обновляет статус элементов, для которых определение элемента не содержит `autoupdate=»false»`;

Вещь отправляет элементу обновление статуса. Помимо конкретных команд управления методы `MyItem.sendCommand(<new_state>)` и `MyItem.postUpdate(<new_state>)` так же доступны и основные команды управления `sendCommand(MyItem, <new_state>)` и `postUpdate(MyItem, <new_state>)`. Обычно рекомендуется использовать конкретные команды.

Сравнение `MyItem.sendCommand(<new state>)` и `sendCommand(MyItem, <new state>)`

Использование команд `MyItem.sendCommand(<new_state>)` и `MyItem.postUpdate(<new_state>)` является приоритетным. В этих методах объекты могут быть разных типов. А методы `sendCommand(MyItem,`

"<new_state>") и `postUpdate (MyItem, "<new_state>")` напротив могут иметь в качестве аргумента только строку.

Это объяснимо близостью к языку Java. В Java и Rules DSL существуют два базовых типа. Это примитивы и объекты. Если названия типов после `val` или `var`, например `var int`, пишется со строчной буквы, то это примитивный тип. Если названия типов после `val` или `var` пишется с заглавной буквы, например, `var Number`, то это объект. Объекты являются более сложными элементами по сравнению с примитивами.

Существуют особые методы, применимые к объектам. Они позволяют автоматически выполнять необходимые преобразования типов. При использовании `MyItem.sendCommand(new_state)` или `MyItem.postUpdate(new_state)` в большинстве случаев `new_state` преобразуется в тип объекта `myItem`.

Команда `sendCommand(MyItem, new_state)` не обладает такой гибкостью. Например, если `new_state` имеет примитивный тип (например, `var int new_state = 3`) а `myItem` является объектом типа диммер:

- команда `sendCommand(MyItem, new_state)` **недопустима**;
- но команда `MyItem.sendCommand(new_state)` **будет работать**.

Использование `MyItem.postUpdate(new_state)` или `MyItem.sendCommand(new_state)` создают более стабильный код. Это, безусловно, лучший способ избежать большинства проблем. Данный синтаксис гарантирует, что любое преобразование (типизация) `new_state` исполнится наиболее подходящим для `myItem`. Исключение: действия полезны, когда имя элемента доступно только в виде строки. Например, если имя элемента для получения обновления или команды было вычислено в правиле путем построения строки, как в таблице 11.

Таблица 11. Код реализации стабильного преобразования

```
val index = 5
sendCommand("My_Lamp_" + index, ON)
```

Использование состояний элементов в правилах

Иногда необходимо найти другие данные из состояний элемента или сравнить состояния элемента с другими значениями. Состояние является важным параметром объекта в openHAB. Состояние элемента — это сам объект, к которому можно получить доступ с помощью `MyItem.state`. Полный и

актуальный список типов элементов, а также типы команд, которые может принимать каждый элемент, приведены в документации openHab для элементов. Важной является информация о типе состояния. Она необходима для применения состояние элемента в правилах. Также необходимо уметь преобразовать текущее состояние в другие типы состояний. И наоборот, чтобы использовать результат вычисления для изменения состояния элемента, может потребоваться его преобразование в подходящий тип.

Далее рассмотрено различие между типом команды и типом состояния. Тип состояния можно получить, добавив «тип» в конец типа команды. Например, цветовой элемент может получить с помощью команды OnOffType, IncreaseDecreaseType, PercentType или HSBType. В таблице 12 представлены команды, необходимые для работы с цветом.

Таблица 12. Код реализации изменения цвета различными типами команд

```
MyColorItem.sendCommand(ON)
MyColorItem.sendCommand(INCREASE)
MyColorItem.sendCommand(new PercentType(50))
MyColorItem.sendCommand(new HSBType(new DecimalType(123), new
PercentType(45), new PercentType(67)))
```

Ещё один способ указать или изменить состояние элемента — применение отформатированных строк. Элементы используют один тип, хотя многие из них принимают команды и обновления различных типов. Color Item из таблицы 12 будет принимать различные типы команд, но вернет только HSBType.

Группы могут иметь любой тип элемента. Внутреннее состояние группы будет соответствовать этому типу. Например, Group:Switch вернет OnOffType для своего состояния. Каждый тип состояния содержит некоторое количество удобных методов, которые могут упростить преобразование и вычисления. Существует несколько способов изучить эти методы. Один из способов заключается в применении openHAB VS Code Extension и нажатии комбинации клавиш <ctrl><space>. Это приведёт к показу списка доступных методов.

Можно рассмотреть JavaDocs для данного типа. Например, JavaDoc for HSBType показывает методы getRed, getBlue, и getGreen. Эти три метода могут быть вызваны в Rules-DSL без части своего имени, например, (MyColorItem.state as HSBType).red. Они получают состояние MyColorItem и приводят его к типу HSBType что бы можно было применять методы свойственные HSBType.

2.2.3. Работа с состояниями элементов: преобразования

Преобразование Item.state в String

Все состояния элемента можно преобразовать в строку, вызвав `MyItem.state.toString`.

Цветовой элемент

Цвет сохраняется как **HSBType**. HSB расшифровывается как оттенок (Hue), насыщенность (Saturation) и яркость (Brightness). Часто желаемый цвет отображается в виде значения RGB (красный, зеленый, синий). Код в таблице 13 используется для преобразования значения RGB в элемент цвета.

Таблица 13. Код отправки «цвета» в свойство объекта

```
import java.awt.Color

// Создаём элемент newColor
// где red, blue и green целые числа от 0 до 255
val newColor = new Color(red, blue, green)

// Сохраняем элемент
MyColorItem.sendCommand(new HSBType(newColor))
```

Отдельные значения цвета из **HSBType** извлекаются как **PercentType**. Следовательно, для получения 8-битного цветового канала RGB необходимо разделить **PercentType** на 100 и умножить на 255. Соответственно, для 16- или 32-битного представления **PercentType** необходимо разделить на 100 и умножить на 65535 или 4294967295. Пример преобразования для 8-битного представления показан в таблице 14.

Таблица 14. Правило преобразования цветов

```
// В теле правила
val red = (MyColorItem.state as HSBType).red / 100 * 255
val green = (MyColorItem.state as HSBType).green / 100 * 255
val blue = (MyColorItem.state as HSBType).blue / 100 * 255
```

Элемент Contact

Элемент типа контакт имеет тип **OpenCloseType**. **OpenCloseType** означает перечисление. Он может быть преобразован из «Открыто» и «Закрыто» в «1» и «0» при помощи простого кода: `val contactNum = if (MyContactItem.state == OPEN) 1 else 0`

Элемент DateTime

Элемент DateTime имеет тип DateTimeType, который внутри содержит ZonedDateTime объект Java. Подробное описание кода дано в таблице 15.

Таблица 15. Операции преобразования элемента DateTime

```
// Получение epoch (Unix-эпоха - число миллисекунд от 00:00:00 UTC 1
января 1970 года) из DateTimeType
val Number epoch = (MyDateTimeItem.state as
DateTimeType).zonedDateTime.toInstant.toEpochMilli

// Получение epoch bp Java ZonedDateTime
val Number nowEpoch = now.toInstant.toEpochMilli

// Преобразование DateTimeType в Java ZonedDateTime
val javaZonedDateTime = (MyDateTimeItem.state as
DateTimeType).zonedDateTime

// Преобразование Java ZonedDateTime в DateTimeType
val DateTimeType date = new DateTimeType(now)
```

Иногда требуется преобразовать временную метку в вид более удобный для конечного пользователя и / или сохранить ее в DateTimeType и DateTime Item. Возможность сделать это с помощью SimpleDateFormat представлена в таблице 16.

Таблица 16. Код преобразование времени в удобочитаемое

```
import java.time.format.DateTimeFormatter

// Преобразование Unix-эпохи в понятное людям время
val DateTimeFormatter formatter =
DateTimeFormatter.ofPattern("yyyy-MM-dd'T'HH:mm:ss.SSSZ")
val long epoch = now.toInstant.toEpochMilli
val ZonedDateTime zdt =
ZonedDateTime.ofInstant(Instant.ofEpochMilli(epoch),
ZoneOffset.UTC);
val String dateTimeString = zdt.format(formatter)

// Преобразование обратно в нормальное время DateTimeType
val DateTimeType dtt = DateTimeType.valueOf(dateTimeString)

// Преобразование состояние элемента типа DateTimeType в строку
```

```
val String datetime_string =
DateTime_Item.state.format("%1$td.%1$tm.%1$ty %1$tH:%1$tM")
```

ZonedDateTime предоставляет ряд полезных методов для сравнения даты и времени вместе и / или при помощи извлечения частей даты. Методы описаны в таблице 17.

Таблица 17. Методы **ZoneDateTimes**

```
// Проверка на корректность времени с типом
DateTimeType
if(now.isBefore((MyDateTimeItem.state as
DateTimeType).zonedDateTime)) ...

// А может быть всё уже пропало типа DateTimeType
if(now.isAfter((MyDateTimeItem.state as
DateTimeType).zonedDateTime)) ...

// узнаёт который час у типа DateTimeType
val hour = (MyDateTimeItem.state as
DateTimeType).zonedDateTime.hour
```

Элемент **Dimmer**

Элемент **Dimmer** (регулятор) имеет тип **PercentType**. **PercentType** можно преобразовать к типу **java.lang.Number** и обрабатываться как **java.lang.Number**. Элемент **Number** является любым типом числового значения. Язык правил поддерживает выполнение математических и логических операций с числами. Объект **Number** поддерживает методы, описанные в таблице 18 получения примитивных версий этого числа.

Таблица 18. Методы объекта **Number**

```
// Загружаем данные из элемента в переменную
val dimVal = MyDimmerItem.state as Number
// как целое число
val int dimAsInt = dimVal.intValue
// и как число с плавающей запятой
val float dimAsFloat = dimVal.floatValue
```

Примеры преобразования шестнадцатеричных значений приведены в таблице 19.

Таблица 19. Преобразование из или в шестнадцатеричные значения

```
// преобразовать hex_code (число, выраженное в шестнадцатеричной системе)
в числовой тип
val dimVal = Integer.parseInt(hex_code, 16) as Number
// для очень больших large_hex_codes другой вариант
val dimVal = Long.valueOf(large_hex_code, 16).longValue() as Number

// и дополнительный пример преобразования integer_value в строку hex_code
var String hex = Long.toHexString(integer_value);
```

Элемент Location

Элемент Location имеет тип **PointType**. PointType содержит два или три числа DecimalType. В данном элементе имеется возможность работать с широтой и долготой в градусах, а также высотой в метрах. Примеры работы Location показаны в таблице 20.

Таблица 20. Описание методов работы элемента Location

```
// Создание
val location = new PointType(new DecimalType(50.12345), new
DecimalType(10.12345))
// Создание из строки. ВНИМАНИЕ: нельзя ставить пробел после запятой
val PointType home = new PointType("12.121212,123.123123")

// Загрузка из элемента
val PointType location = Device_Coordinates.state as PointType
```

Числовые элементы

Числовые элементы имеют тип **DecimalType** или **QuantityType**, например, Number:Temperature. Пользователь может добавить размерность к числу при помощи символа |. Размерность указывается в кавычках если содержит символы, отличные от букв и цифр, например, 10|m, but 20|"km/h". Некоторые международно признанные единицы не требуют кавычек: °C, °F, Ω, °, %, m² and m³, например можно написать 20|«°C», но 20|°C тоже правильно. Типы DecimalType и QuantityType так же являются java.lang.Number и все преобразования описанные выше для Dimmer так же применимы к элементу Number. Часто используемые преобразования представлены в таблице 21.

Таблица 21. Код описания работы методов числовых элементов

```
//Для состояний DecimalType::
// преобразует целое число в строку содержащую шестнадцатиричный код
var String hex_code = Long.toHexString(integer_number)

// преобразует шестнадцатиричный код в объект типа Number
var myNumber = Integer.parseInt(hex_code, 16) as Number
// то же для large_hex_code
var myNumber = Long.parseLong(hex, 16) as Number

// преобразует hex_code в DecimalType
var          DecimalType          parsedResult          =          new
DecimalType(Long.parseLong(hex_code, 16))
// Для состояний QuantityType:
// описание переменной типа QuantityType
var myTemperature = 20|°C

// получить размерность элемента в виде текста
var myUnits = myTemperature.getUnit.toString // gives "°C"

// преобразование количественного состояния в различные единицы:
var fahrenheit = myTemperature.toUnit("°F") // will contain quantity
68°F

// количественное значение в десятичное
var myDecimal = new DecimalType(fahrenheit.doubleValue) //
myDecimal == 68
var myCentigrade = fahrenheit.toUnit("°C").toBigDecimal // 20

// доступ к скалярным значениям как int, double, float
var myInt = fahrenheit.intValue
var mydouble = fahrenheit.doubleValue
var myfloat = fahrenheit.floatValue

// проверка не является ли состояние элемента количеством
var isQuantity = myItem.state instanceof QuantityType

// сравнение количества
// Конструкция if (fahrenheit > 10) НЕ РАБОТАЕТ!!
if (fahrenheit > 10|°C) { logInfo("test", "It's warm.") }
```

Особенность `DecimalType` (относится и к `QuantityType`). Необходимо преобразовывать состояние типа `DecimalType` к `Number`, а не наоборот, чтобы не было ошибки: «неоднозначный вызов метода». Важно учитывать математическую составляющую, связанную с `QuantityType`. Хотя во многих

случаях пользователь имеет право свободно смешивать единицы, но есть ряд особенностей, указанных в таблице 22.

Таблица 22. Особенности использования числовых элементов

```
// сложим количественные переменные разных размерностей
var miles = 2|mi
var metres = 10|m
var distance = miles + metres // результат 2.0062 mi
// Размерность у результата та же что у первого слагаемого

var area = metres * metres // результат 100 m²
// Новые подходящие единицы используются для результата

var fahr = 68|°F
var centi = 1|°C
var sumTemps = fahr + centi // результат 101.80 °F
// Результат может показаться неожиданным...
// Однако, температура всегда считается абсолютной, а не интервалом или
приращением.
// 1°C был преобразован в 33.8°F, а не в интервал 1.8°F

// Для этого есть математический трюк
var increment = fahr + centi - 0|°C // результат 69.80 °F
// "выделение нуля" фиксирует смещение между разными размерностями
```

Элемент Player

Элемент проигрыватель (player) позволяет управлять проигрыванием (например аудиоплеера). Для этого применяются следующие команды: Play, Pause, Next, Previous, Rewind и Fastforward. Player имеет три типа, указанные в таблице 23, с определёнными командами.

Таблица 23. Состояния и команды проигрывателя

Тип Состояния	Команды
PlayPauseType	PLAY, PAUSE
RewindFastforwardType	REWIND, FASTFORWARD
NextPreviousType	NEXT, PREVIOUS

Эти три типа могут быть преобразованы по типу контакта из Открыто и Замкнуто в 1 и 0 при помощи этого простого скопированного кода (из раздела про контакты). Получение состояния из элемента представляется в виде: `val int Playing = if (MyPlayerItem.state == PLAY) 1 else 0`.

Тип Rollershutter (жалюзи)

В дополнение к командам свойственным Dimmer элемент Rollershutter понимает StopMoveType с командами STOP и MOVE.

Элемент Строка (String)

Для изменения состояние элемента StringType используется метод toString. Получения строки из элемента представляется как: `val stateAsString = MyStringItem.state.toString`. Если элемент возвращает строку, содержащую значение в виде шестнадцатеричного числа, его можно преобразовать в целое число из шестнадцатеричного числа так: `val itemvalue = new java.math.BigDecimal(Integer.parseInt(myHexValue, 16))`.

Элемент Выключатель (Switch)

Элемент Switch имеет OnOffType. OnOffType обозначает перечисление. Можно преобразовать из ВКЛ и ВЫКЛ в 1 и 0 с помощью кода, эквивалентного следующему: `val SwitchNum = if (MySwitchItem.state == ON) 1 else 0`.

Важно понимать различие между объектами и примитивами. В языках Java и Rules DSL реализована концепция наследования. Особенностью является то, что наследование применимо только к объектам и не применимо к примитивам. Примерами примитивов являются integer и boolean. Наследование позволяет взять существующий тип объекта, называемый классом, и добавить к нему, чтобы превратить его во что-то другое. Это «нечто иное» становится Дочерним от исходного Класса, родителем. Ребенок по-прежнему может делать все, что может делать родитель. Базовый класс верхнего уровня для всех объектов в Java и DSL правил называется просто Object.

Дополнительно в классе Object реализован метод с именем toString. А поскольку Object является родительским для всех объектов, ВСЕ классы также реализуют toString метод. Однако примитивы не наследуются от Object. Они ни от чего не наследуются, и у них вообще нет никаких методов, включая отсутствие метода toString.

Объекты имеют гораздо более обширное количество методов преобразования типов [8]. Прimitives не поддерживают преобразование типов. Это важно учитывать при попытке использовать результат вычисления. Действие `sendCommand` является универсальным. Оно может работать со всеми типами элементов. Действия поддерживают только два аргумента `String`, но при этом все объекты будут поддерживать преобразование `toString.sendCommand (MyItem, new_state)`. Автоматически применяется `MyItem.toString` – метод для преобразования `MyItem` в `String`. Существует возможность подобного преобразования второго аргумента, если `new_state` это еще не `String`. Однако, если второй аргумент является примитивом, а не объектом, он не несет метода `toString`. Таким образом, правила DSL не могут быть преобразованы `new_state` в строку. Важно понимать, что применение `sendCommand (MyItem, primitive)` при использовании примитива в качестве второго аргумента почти всегда приводит к ошибке.

Синтаксис для общего и специфического применения различается и представлен в таблице 24 ниже.

Таблица 24. Действия и методы отправки команд

Generic (Action)	Specific (Method)
<code>postUpdate(MyItem, new_state)</code>	<code>MyItem.postUpdate(new_state)</code>
<code>sendCommand(MyItem, new_state)</code>	<code>MyItem.sendCommand(new_state)</code>

Преимущество использования объектов над примитивами заключается в изменении типов, которые автоматически вызываются объектом в зависимости от контекста. Использование метода `MyItems.sendCommand()`, которым владеет `MyItem`, будет использовать `sendCommand` метод, который подходит для выполнения необходимых преобразований типов. Например, `NumberItem` класс имеет `sendCommand(int)`, `sendCommand(long)`, `sendCommand(float)`, `sendCommand(double)`, `sendCommand(Number)`, `sendCommand(DecimalType)`, и `sendCommand(String)` методы. Каждый из этих отдельных методов индивидуально написан для обработки различных типов объектов. `MyItem` автоматически применит метод в зависимости от типа аргумента.

Неявные переменные внутри выполняемого блока

Помимо неявно доступных переменных для элементов и команд или состояний, правила могут иметь дополнительные предварительно определенные переменные в зависимости от их триггеров:

- `receivedCommand` — неявно доступная переменная в любом правиле, имеющем минимум один триггер командного события;
- `previousState` — неявно доступная переменная в любом правиле, имеющем хотя бы один триггер события изменения статуса;
- `newState` — неявно доступная переменная в любом правиле, имеющем хотя бы один триггер события обновления или изменения статуса;
- `triggeringItemName` — неявно доступная переменная в любом правиле, которое имеет хотя бы одно обновление статуса, изменение статуса или триггер командного события;
- `triggeringItem` — неявно доступная переменная в любом правиле, имеющем триггер;
- `receivedEvent` — неявно доступная переменная в любом правиле с канальным триггером.

Раннее возвращение

Можно заранее вернуться из правила, не выполняя остальные инструкции, подобные описанному в таблице 25.

Таблица 25. Досрочное возвращение из правила

```
if (Temperature.state > 20) {  
    return;  
}  
Heating.sendCommand(ON)
```

Следует обратить внимание на точку с запятой после оператора `return`, который завершает команду без дополнительного аргумента.

Защита конкуренции

Если правило срабатывает при событиях пользовательского интерфейса, может потребоваться защита от параллелизма. Описание приведено в таблице 26.

Таблица 26. Защита от параллелизма

```
import java.util.concurrent.locks.ReentrantLock

val ReentrantLock lock = new ReentrantLock()

rule ConcurrentCode
when
    Item Dummy received update
then
    lock.lock()
    try {
        // выполнение действий
    } finally {
        lock.unlock()
    }
end
```

Трансформации

Сервисы openHAB Transformation применяются в правилах для преобразования / перевода / преобразования данных. Общий синтаксис представляется в виде: `transform("<transformation-identifier>", "<transf. expression or transf. file name>", <input-data or variable>)`, где:

- `<transformation-identifier>` — сокращенный идентификатор функции трансформации
- `<transf. expression or transf. file name>` — специальная функция трансформации
- `<input-data or variable>` — данные для преобразования ДОЛЖНЫ иметь тип данных String.

Примеры трансформаций приведены в таблице 27.

Таблица 27. Трансформации данных

```
var condition = transform("MAP", "window_esp.map", "CLOSED")
var temperature = transform("JSONPATH", "$.temperature",
jsonstring)
var fahrenheit = transform("JS", "convert-C-to-F.js",
temperature)
```

В методе `transform` пользователь может изменить заданное значение, и, если это не удаётся, исходное значение останется неизменным. Существует

метод `transformRaw`, в котором недопустимы `TransformationException` символы Синтаксис имеет следующий вид: `transformRaw("<transformation-identifier>", "<transf. expression or transf. file name>", <input-data or variable>)`. Примеры показаны в таблице 28.

Таблица 28. Метод трансформ

```
try {
    var temperature = transformRaw("JSONPATH", "$.temperature",
jsonstring)
}
catch(TransformationException e) {
    logError("Error", "Я написал неверное правило: " +
e.getMessage)
}
finally {
    // всегда запускается даже если произошла ошибка! Можно использовать
для очистки
}
```

Запись журналов

Пользователь может создавать сообщения журнала из ваших правил, чтобы облегчить отладку. Существуют методы ведения журнала, доступные из правил пользователя, подписи Java, описанные в таблице 27.

Таблица 27. Сообщения журнала

```
logDebug(String loggerName, String format, Object... args)
logInfo(String loggerName, String format, Object... args)
logWarn(String loggerName, String format, Object... args)
logError(String loggerName, String format, Object... args)
```

В каждом случае `loggerName` параметр объединяется со строкой `org.openhab.core.model.script` для создания имени регистратора `log4j`. Например, если файл правил содержит следующее сообщение журнала: `logDebug("kitchen", "Kitchen light turned on")`, для появления уведомления на консоли пользователя можно сделать следующие настройки журнала: `log:set DEBUG org.openhab.core.model.script.kitchen`.

Примеры правил

В таблице 28, приведены некоторые примеры общих правил.

Таблица 28. Примеры правил

```

var Number counter

// установка начальных значений счётчиков
// можно это делать и в объявлении переменной
rule "Поехали"
when
    System started
then
    counter = 0
end

// Увеличиваем счётчик в полночь
rule "а счётчик-то тикает"
when
    Time cron "0 0 0 * * ?"
then
    counter = counter + 1
end

// в полдень доложить о количестве суток автономного полёта!
rule "Сообщить сколько дней"
when
    Time is noon or
    Item AnnounceButton received command ON
then
    say("Всё хорошо уже " + counter + " дней")
end

// меняет счётчик по команде от начальства
rule "Улучшить счётчик"
when
    Item SetCounterItem received command
then
    counter = receivedCommand as DecimalType
end

// включает свет, когда один из нескольких датчиков движения получил
команду ВКЛ, выключает его, когда один из нескольких полученных команд ВЫКЛ.
rule "Прожектор в лицо"
when
    Member of MotionSensors received command
then
    if(receivedCommand == ON) Light.sendCommand(ON)
    else Light.sendCommand(OFF)
end

```

```
end

rule "Побудка на рассвете"
when
    Channel "astro:sun:home:rise#event" triggered
then
    switch(receivedEvent) {
        case "START": {
            Light.sendCommand(ON)
        }
    }
end
```

2.3. Проектирование интерфейсов дашбордов с помощью HABPanel

Одним из преимуществ программы openHAB является удобный пользовательский интерфейс HABPanel. С помощью него можно повысить удобство информационных панелей, например, для настенных планшетов. Эти информационные панели можно создавать в интерактивном режиме во встроенном конструкторе, а не с помощью файлов конфигурации.

Несмотря на схожесть, информационные панели и карты сайта HABPanel представляет собой отдельный элемент и может быть создана независимо. Однако необходимо учесть, что информационные панели и карты сайта HABPanel могут работать с одними и теми же элементами. Демонстрационный установочный пакет, доступный для установки при первом запуске openHAB, определяет ряд элементов-примеров и настраивает HABPanel с полным набором информационных панелей для демонстрации конечному пользователю. Он такой же, как установленный на демонстрационном сервере openHAB, и его можно модифицировать, не рискуя ничего сломать, что также является большим преимуществом openHAB.

2.3.1. Концепции

НАВPanel имеет собственную терминологию сущностей, представленную на рис. 55.

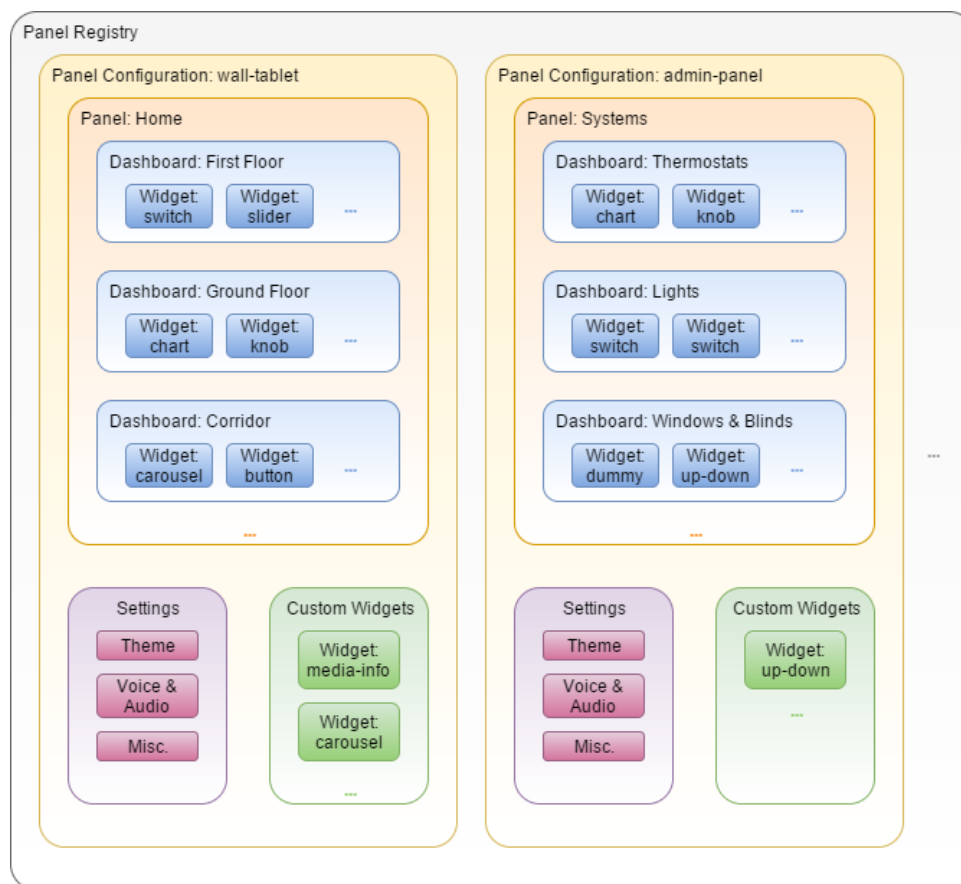


Рисунок 55. Сущности HabPanel

Реестр **Panel** — это центральное хранилище, используемое НАВPanel в openHAB. Данный элемент имеет в своём составе несколько **конфигураций Panel**.

Конфигурация **панели** — это контейнер, содержащий **панель** вместе с ее **настройками** и определением **пользовательских виджетов**. Любое устройство, на котором работает НАВPanel (настольный браузер, планшет и т.д.), имеет активную конфигурацию панели и отображает связанную с ней панель.

Панель — это набор **информационных панелей** (или страниц), к которым имеют доступ конечные пользователи. В них есть функция переключения с помощью меню.

Информационная панель состоит из обособленных элементов, называемых **виджетами**. Виджеты расположены на поверхности панели во

время разработки. Существует несколько типов встроенных *стандартных виджетов*, настраиваемых отдельно, а также администратор может разрабатывать (или импортировать) *пользовательские виджеты*.

2.3.2. О сохранении данных

По умолчанию при запуске НАВPanel в новом браузере или на новом устройстве будет отображаться руководство, позволяющее пользователю начать с нуля или переключиться на ранее определенную конфигурацию панели. **Пока не будет создана (или выбрана) конфигурация панели, НАВPanel будет работать в режиме «локального хранилища» для этого устройства. Следовательно, настройки будут сохраняться только в локальном хранилище браузера, данные не будут сохраняться на сервере.**

Если пользователь установит активную конфигурацию панели, то каждое изменение на устройстве будет обновлять конфигурацию панели на сервере. Это позволяет совместно использовать конфигурацию панели между устройствами, потому что другие браузеры и устройства, использующие эту конфигурацию панели, увидят изменения при обновлении страницы. Такие данные могут быть использованы, например, для удобного создания панели на компьютере с целью дальнейшего использования ее на планшете.

Чтобы переключиться с локального хранилища на конфигурацию панели, размещенной на сервере, перейдите на страницу настроек из главного меню или боковой панели (см. ниже). Список конфигураций панели будет представлен в разделе «*Текущая конфигурация хранилища*»; если доступна только опция «*Локальное хранилище*», **нажмите на ссылку «Сохранить текущую конфигурацию» в новой конфигурации панели**, дайте ей имя для идентификации, и она должна быть добавлена в список. Радиокнопка также автоматически проверяется, что означает, что теперь она является активной конфигурацией панели.

Даже при наличии активной конфигурации панели НАВPanel использует хранилище браузера для синхронизации локально управляемой копии. С помощью ссылки «**Изменить конфигурацию локальной панели (только для экспертов)**» под параметром **конфигурации** хранилища «*Локальное хранилище*» на экране настроек необработанную структуру конфигурации панели можно проверить, изменить и экспортировать или импортировать из/в файл.json. Это также альтернативный способ резервного копирования, восстановления и совместного использования конфигурации.

Для сохранения данных на сервере openHAB необходимо применить переменные конфигурации службы в HABPanel. Для доступа к данным применяют пользовательский интерфейс Paper (Configuration > Services > UI > HABPanel > Configure) или консоль openHAB Karaf:

```
openhab> config:edit org.openhab.habpanel
```

```
openhab> config:property-get <property>
```

Определены следующие свойства:

- `panelsRegistry`: содержит весь реестр в JSON. Доступ к нему осуществляется через приложение. Возможно редактирование вручную, но оно не рекомендуется;
- `lockEditing`: при включении все экземпляры HABPanel будут скрывать свои функции редактирования (необходимо обновление страницы). Когда панели готовы и стабильны, рекомендуется включить этот параметр, чтобы конечные пользователи не могли легко изменить их;
- `initialPanelConfig`: если этот параметр не установлен, и предыдущая локальная конфигурация не обнаружена, начальный вариант будет отображаться до тех пор, пока не будут добавлены некоторые панели мониторинга или не будет выбрана конфигурация панели. Данный параметр даёт возможность обойти обучение и работать непосредственно с существующей конфигурацией панели с заданным именем.

2.3.3. Основные элементы и функции интерфейса

Главное меню

Главное меню — это домашняя страница HABPanel. В нём расположены плитки со ссылками на информационные панели и значок для переключения между различными режимами работы: запуском и редактированием (если режим редактирования доступен) (рис. 56).

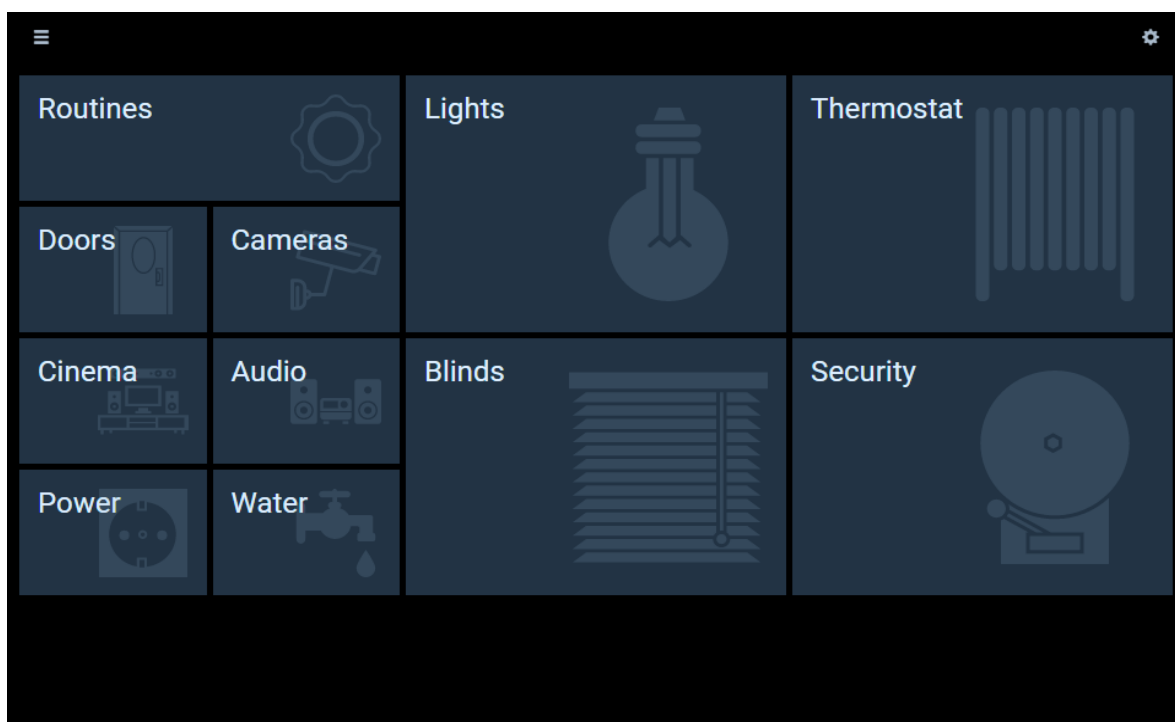


Рисунок 56. Домашняя страница HabPanel

Для переключения между двумя режимами можно нажать на значок шестеренки в правом верхнем углу (рис. 57).



Рисунок 57 – Переключение между режимами запуска и редактирования

Существует ряд функций в режиме редактирования:

- Во-первых, пользователь может добавить новую пустую панель мониторинга по ссылке «**Добавить новую панель мониторинга**»;

- Существует возможность перемещения на экран настроек (например, чтобы переключиться с локального хранилища на конфигурацию панели, управляемой сервером). Пользователь может кликнуть на ссылку «Дополнительные настройки»;

- Пользователь имеет возможность настроить число столбцов сетки плиток главного меню с помощью ползунка от до 6;

- Переместить значки со стрелками в верхнем левом углу каждой плитки, чтобы изменить её расположение;

- Увеличить или уменьшить размер плитки с помощью шеврона (треугольника) в правом нижнем углу каждой плитки;

- Настроить плитки и сами информационные панели с помощью значков шестеренки в правом верхнем углу каждой плитки;

- Получить доступ к конструктору панели мониторинга, кликнув внутри плитки.

Конфигурация при нажатии на значок шестеренки плитки содержит следующие настройки, приведённые в таблице 28.

Таблица 28. Описание конфигурации настройки панели

Параметр	Описание
Имя	Название панели мониторинга, отображаемое на плитке
Фоновый URL-адрес	URL-адрес фонового изображения <i>для плитки главного меню.</i>
Значок фона	Совокупность иконок и значок, показываемый на плитке в качестве фона
Центральный фон по горизонтали	Если этот флажок не установлен, фон выравнивается по правому краю плитки; при проверке центрируется

Параметр	Описание
Значок	Значок, связанный с приборной панелью, в настоящее время используется только в боковом ящике.
Размер (значок)	<i>(в настоящее время не используется)</i>
Цвет текста заголовка	Цвет названия дашборда на плитке
Продвинутая вкладка	Содержит нестабильные или неподдерживаемые настройки (только для опытных пользователей).
Вкладка пользовательских виджетов	Эта экспериментальная функция (открывает новое окно) позволяет заменить определенные части панели инструментов пользовательским виджетом: плитку главного меню, меню ящика и элемент панели инструментов.

Также имеется кнопка **Удалить**, которая удалит всю панель инструментов и ее содержимое. Данное действие происходит без задания дополнительного вопроса пользователю и не может быть отменено.

Боковой ящик

Доступ к боковому ящику можно получить с любого экрана, проведя пальцем или перетаскив вправо (на большинстве элементов, где этот жест не конфликтует), или с помощью трёх полосок ≡ в верхнем левом углу (рис. 58).

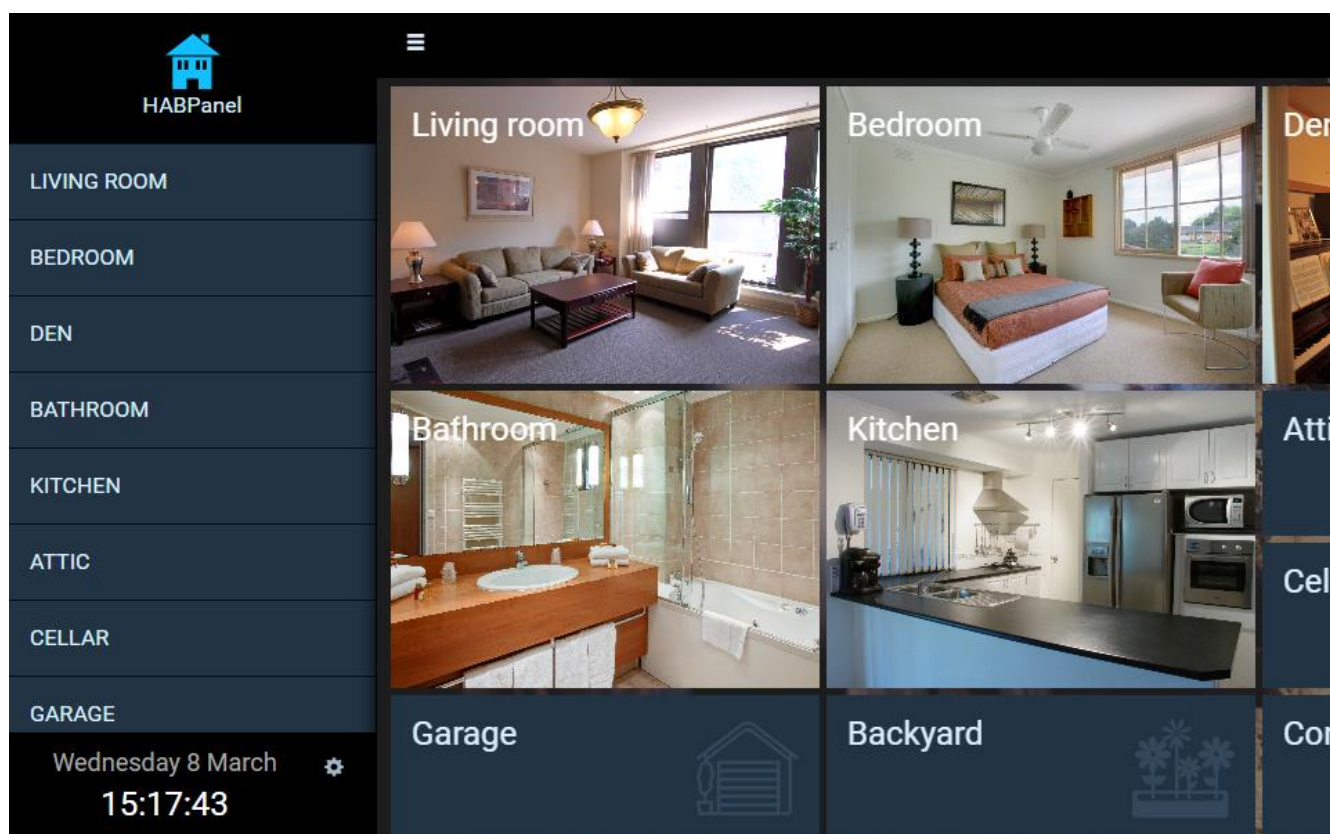


Рисунок 58. Боковой ящик HubPanel

Данный элемент имеет в своём составе три основные части:

- Первый элемент – это заголовок. Если пользователь кликает по нему, то он **возвращается** в главное меню.
- Далее идёт список **дашбордов** для быстрого перемещения между дашбордами без возврата в главное меню. Представлены в порядке меню (отсортированы по строке, затем по столбцу).
- Нижний **колонтитул** применяется для показа текущей даты и времени, а также ссылки на экран настроек.

Дизайнер приборной панели

В конструкторе панели управления можно добавлять виджеты, размещать их, увеличивать или уменьшать и всячески настраивать. Заполнители отображаются там, где фактически должны находиться виджеты: на работающей панели инструментов (рис. 59).

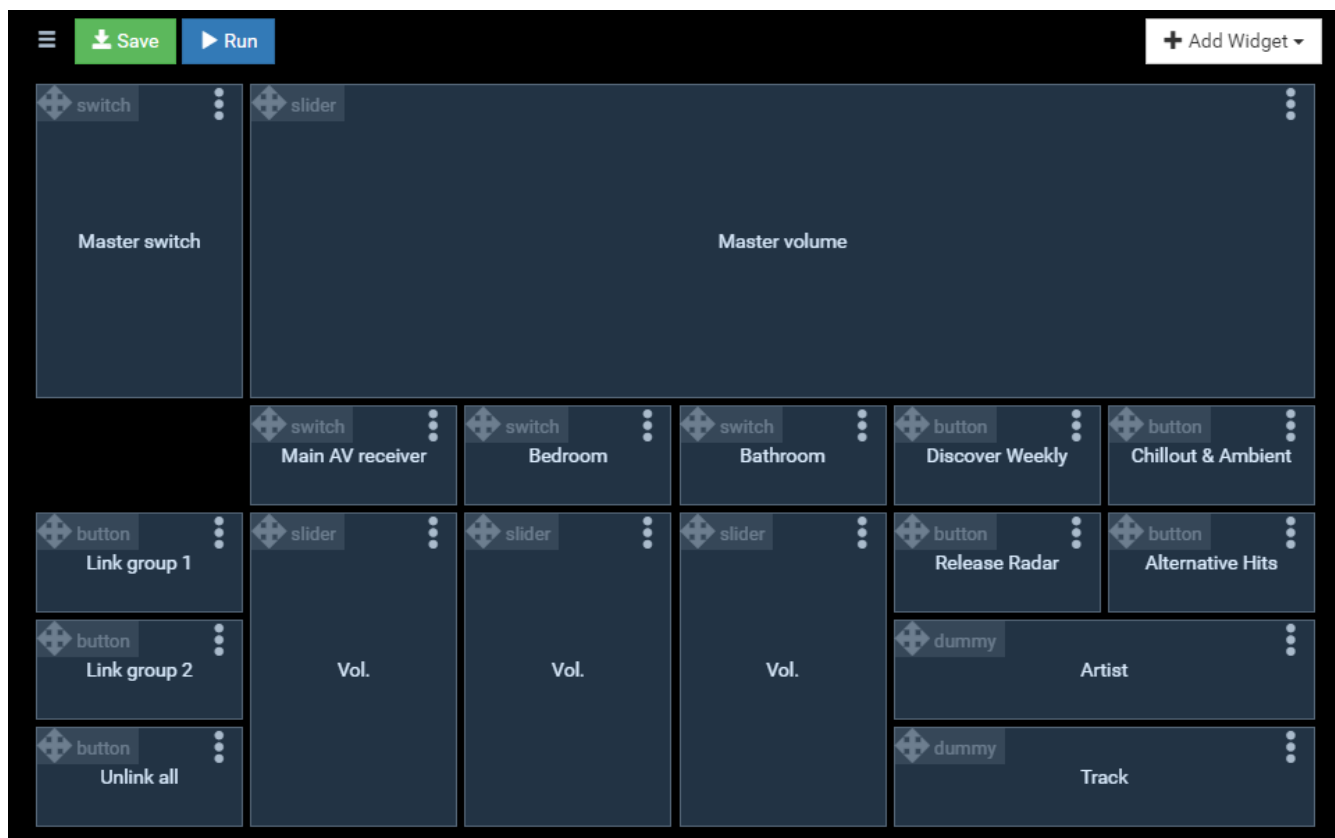


Рисунок 59. Приборная панель

Чтобы добавить виджет, необходимо кликнуть по кнопке «Добавить виджет» и выбрать один из стандартных виджетов или возможных пользовательских виджетов в конфигурации панели (рис. 60).

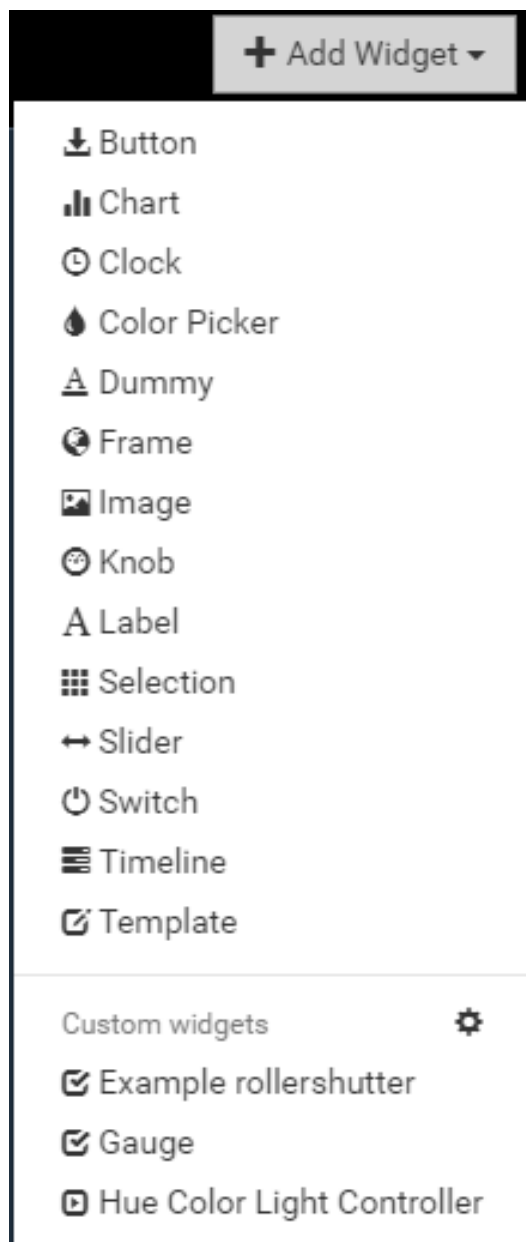


Рисунок 60. Меню добавления нового виджета на приборную панель

Используйте заголовок заполнителя виджета (со значком с четырьмя стрелками и типом виджета), чтобы переместить виджет. Перемещая виджет поверх других виджетов, не отталкивайте их, чтобы освободить место (в отличие от плиток главного меню). Перед перемещением необходимо знать, что для вашего виджета достаточно места (рис. 61).

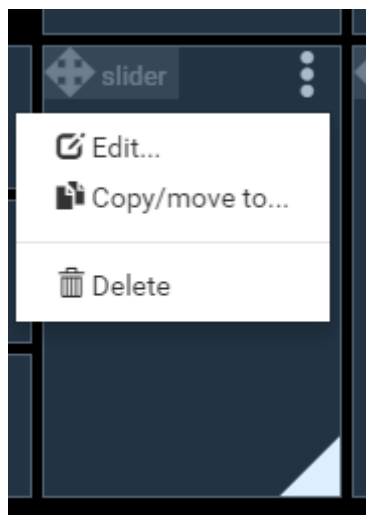


Рисунок 61. Меню передвижения виджета

При наведении указателя мыши на заполнитель (или касании его внутри, если используется сенсорный интерфейс), в правом нижнем углу появляется шеврон, позволяющий изменить его размер.

Для вызова контекстного меню необходимо нажать на значок с многоточием \therefore . В данном меню есть параметры:

- **Редактировать...:** показывает диалоговое окно конфигурации виджета. Параметры, доступные в диалоговом окне, зависят от типа виджета и подробно описаны в разделе Виджеты;
- **Копировать/переместить в... :** показывает диалоговое окно, позволяющее копировать виджет с его конфигурацией на текущую или другую панель мониторинга или переместить его на другую панель (целевой виджет сохранит размер исходного виджета, но будет размещен там, где есть свободное место;
- **Удалить:** удаляет виджет.

Изменения в панели управления не сохраняются автоматически. Для этого существует кнопка **Сохранить** в заголовке, чтобы зафиксировать изменения в конфигурации панели (или в локальном хранилище). Кнопка **«Выполнить»** также сохраняет, а затем запускает панель инструментов.

Запуск дашбордов

Когда панель мониторинга работает, с виджетами можно взаимодействовать, а отправленные сервером события принимаются при

обновлении состояний элементов, поэтому виджеты автоматически обновляются в HAVPanel (рис. 62).

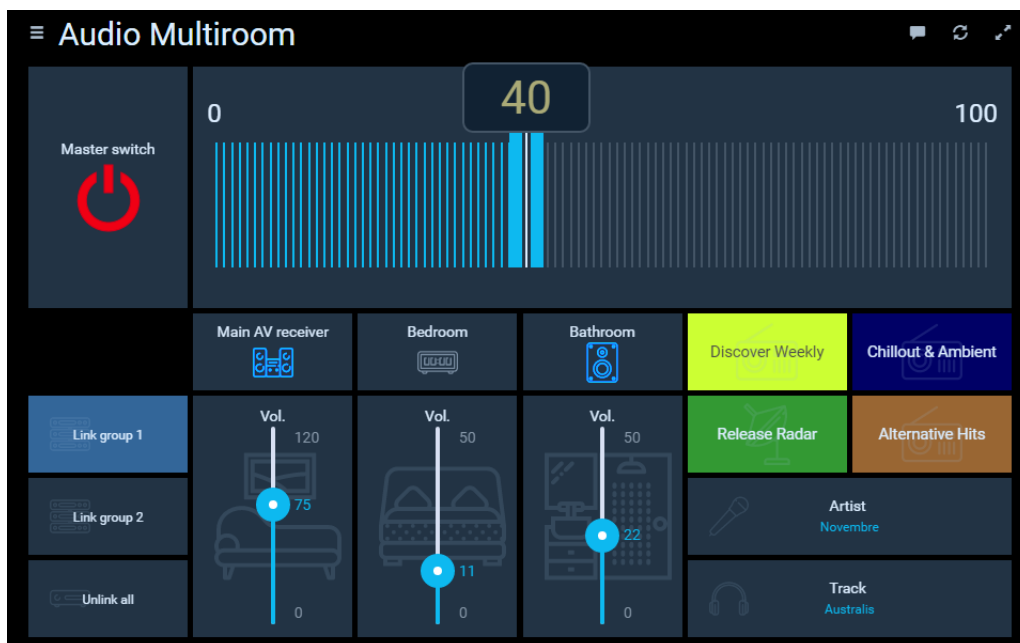


Рисунок 62. Работающая панель дэшбордов

С помощью значков в правом верхнем углу пользователь может исполнить ряд операций:

- речевое всплывающее окно активирует функцию распознавания речи и отправляет результаты в виде текста интерпретатору человеческого языка openHAV по умолчанию. Это подразумевает некоторую настройку на сервере, и этот значок может не отображаться, если браузер не поддерживает распознавание голоса. Его отображение в той или иной части экрана можно настроить в конфигурации панели;
- кнопка обновления позволяет HAVPanel распознать текущее состояние всех элементов;
- полноэкранный значок позволяет браузеру перейти в полноэкранный режим, если он поддерживается.

При использовании информационной панели пользователь может написать ? kiosk=onURL-адрес в веб-браузере, чтобы переключиться в «режим киоска». В этом режиме заголовок, гамбургер-меню и панель инструментов будут скрыты, а боковая панель будет отключена, поэтому у конечного пользователя не будет простого способа переключиться на другой экран. Этот режим удобен для отображения полноэкранного пользовательского интерфейса для стационарного планшета.

Дополнительные возможности и настройки

Помимо описанной выше конфигурации хранилища, экран настроек содержит несколько настроек, которые сохраняются как часть конфигурации панели (то есть задаются отдельно). Список настроен приведён в таблице 29.

Таблица 29. Отдельно задающиеся настройки панели

Параметр	Описание
Название панели	Задаёт имя панели, которое будет написано в заголовке бокового ящика.
Тема	НАВPanel имеет ряд стандартных встроенных тем. Темы не изменяются пользователем.
Фоновая картинка	Устанавливает указанный URL-адрес в качестве фонового изображения для всей панели. Необходимо учесть, что фоновое изображение лучше всего работает с полупрозрачной темой
Дополнительная таблица стилей	Можно написать относительный URL-адрес дополнительного файла CSS, который будет включен и может использоваться для переопределения стилей, определенных темой.
Изображение заголовка ящика	Пользователь может ввести URL-адрес изображения (ожидаемая ширина: 265 пикселей), которое заменит заголовок в боковом ящике.
Скрыть нижний колонтитул ящика	Пользователь может удалить нижнюю черную часть бокового ящика, содержащую дату и время.
Скрыть кнопки панели инструментов (озвучивание/обновление/ полноэкранный режим)	Пользователь может установить эти флажки, чтобы скрыть соответствующую кнопку в правом верхнем углу заголовка панели инструментов по умолчанию.
Показать часы в шапке	При активировании флажка, часы появятся в главном меню и на информационных панелях.
Формат часов заголовка (появляется, если установлен)	Используется для установки формата часов в шапке. По умолчанию HH:mm

флажок «Показывать часы в заголовке»)	
Запретить прокрутку (когда не редактируется)	Когда эта функция включена, прокрутка приборной панели на планшете невозможна (и предотвращает эффект «эластичного» подпрыгивания на iOS/Safari)
Управление > (пользовательские виджеты)	Переход к списку определений пользовательских виджетов для конфигурации активной панели.
Голос	Пользователь может выбрать голос из обнаруженного списка для преобразования текста в речь*
Произнесите новое значение следующего элемента, когда оно изменится	Когда выбранный элемент String изменится на новый текст, НАВPanel будет использовать механизм преобразования текста в речь браузера и выбранный голос, чтобы прочитать его вслух*
Отображение плавающей кнопки речи в нижней части экрана	Пользователь может применить альтернативный стиль для кнопки «Говорить» (голосовой ввод) на информационных панелях.
Когда этот элемент изменится на имя панели мониторинга, переключитесь на него.	Это позволяет управлять отображаемой в данный момент информационной панелью с помощью элемента openHAB.

*Примечание: функция преобразования текста в речь, представленная в НАВPanel, не связана со службами TTS, определенными на сервере openHAB, и они несовместимы (поэтому требуется элемент String, а say() функция не может использоваться). Однако НАВPanel будет воспроизводить аудио, передаваемое через приемник webaudio, включая устный текст.

2.4. Стандартные виджеты

В программе имеется ряд встроенных виджетов:

Манекен

Так называемый фиктивный виджет (название которого объясняется историческими причинами — он развился из первого разработанного виджета) отображает текущее состояние элемента без какой-либо интерактивности, вместе с меткой и необязательным значком (рис. 63).

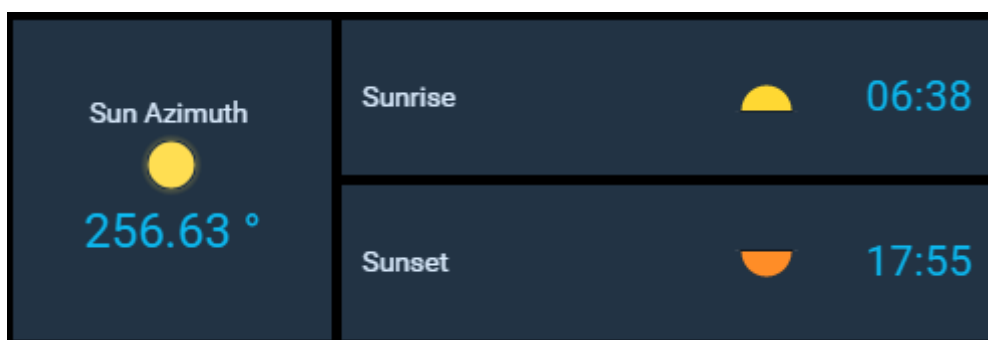


Рисунок 63. Виджет «Манекен»

Переключатель

Виджет переключения используется для управления элементом Switch в openHAB. Он показывает пользователю своё состояние и может переключать его между ON и OFF (рис. 64).

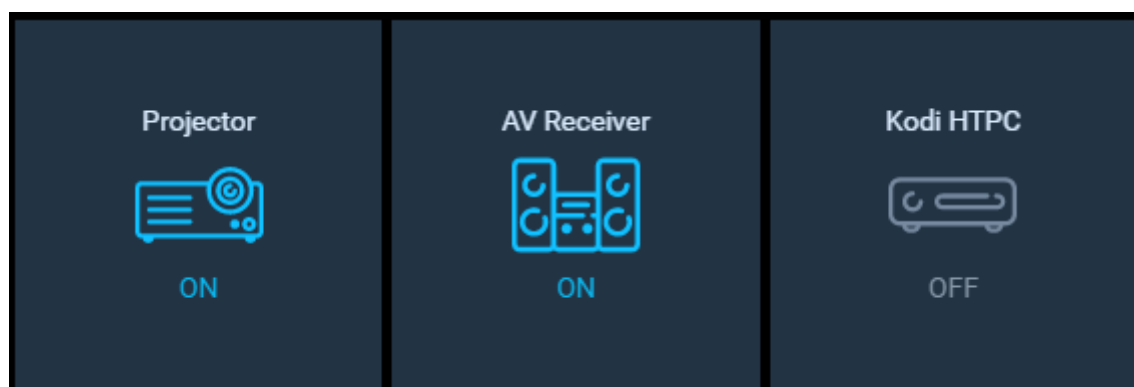


Рисунок 64. Виджет «Переключатель»

Этикетка

Данный виджет показывает фиксированный текст и имеет несколько вариантов внешнего вида. Данные варианты задаются цветом или шрифтом. Например, его можно использовать в качестве заголовка для группы виджетов под ним (рис. 65).



Рисунок 65. Виджет «Этикетка»

Кнопка

Виджет кнопки можно кликнуть (или коснуться), и он будет выполнять действие. Он может изменить свои цвета в зависимости от состояния основного элемента (рис. 66).

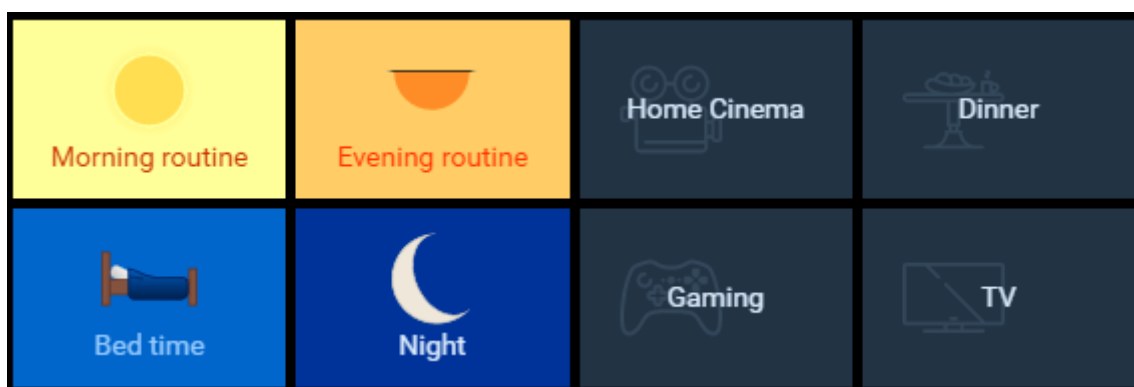


Рисунок 66. Виджет «Кнопка»

Несколько кнопок часто используются вместе, чтобы представить различные варианты элемента.

Слайдер

Виджет ползунка может показывать состояние и обновлять числовые элементы в определённом диапазоне значений. Варианты изменения его внешнего вида и поведения приведены на рис. 67.

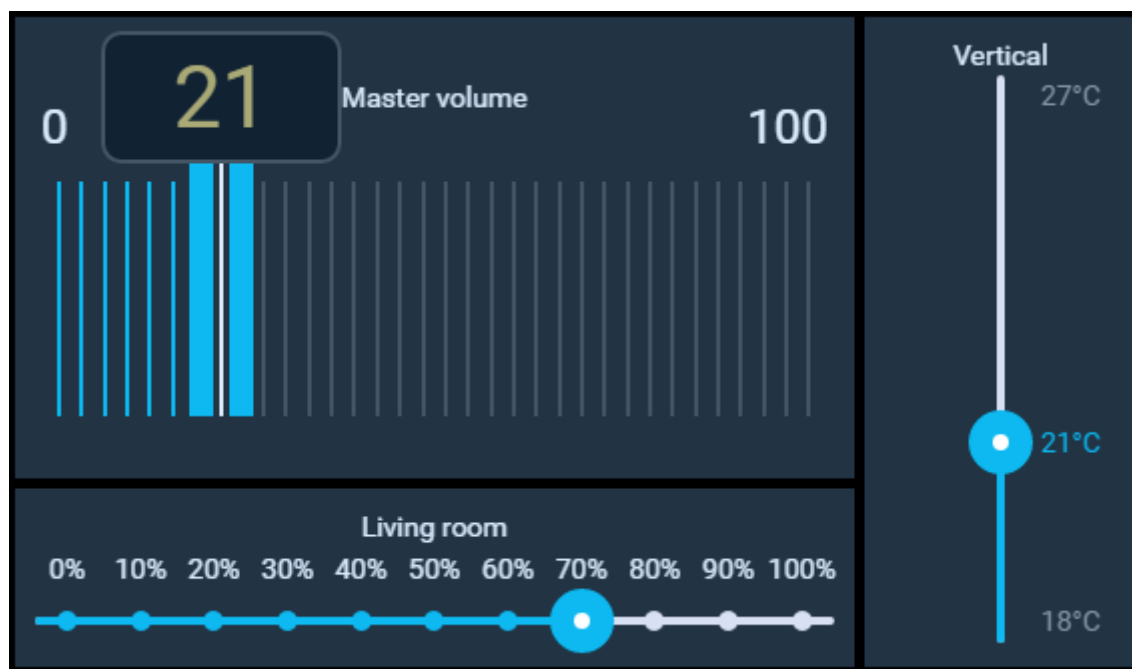


Рисунок 67. Виджет «Слайдер»

Ручка

Виджет-ручка, по сути, похож на ползунок, но вращается. С помощью него пользователь может изменить внешний вид и поведение элемента (рис. 68).

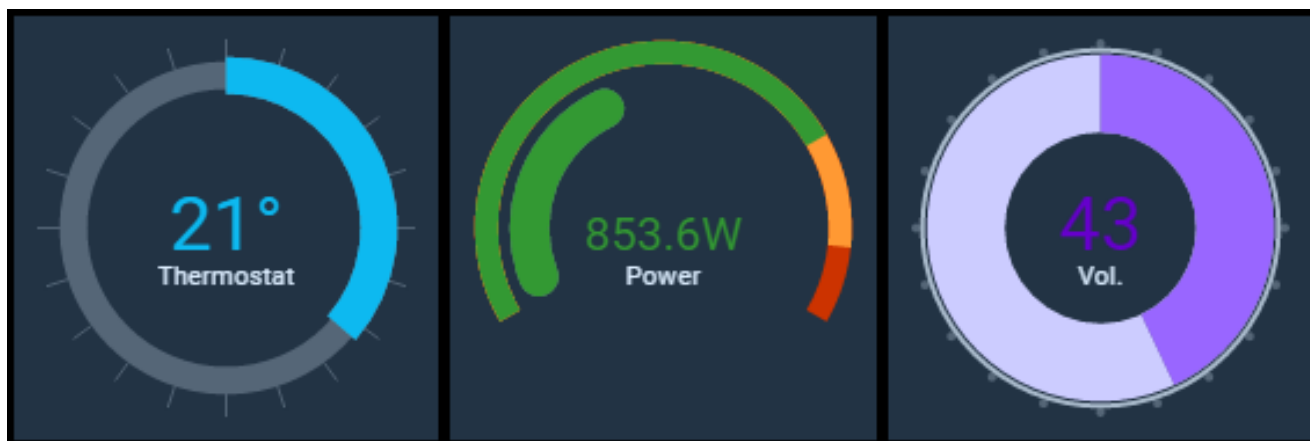


Рисунок 68 – Виджет «Ручка»

Отбор

Виджет выбора показывает текущее состояние элемента, как фиктивный виджет, за исключением того, что он открывает меню или сетку автоматически или ручную настроенных вариантов для отправки команд этому элементу. Доступны различные варианты отображения (рис. 69).

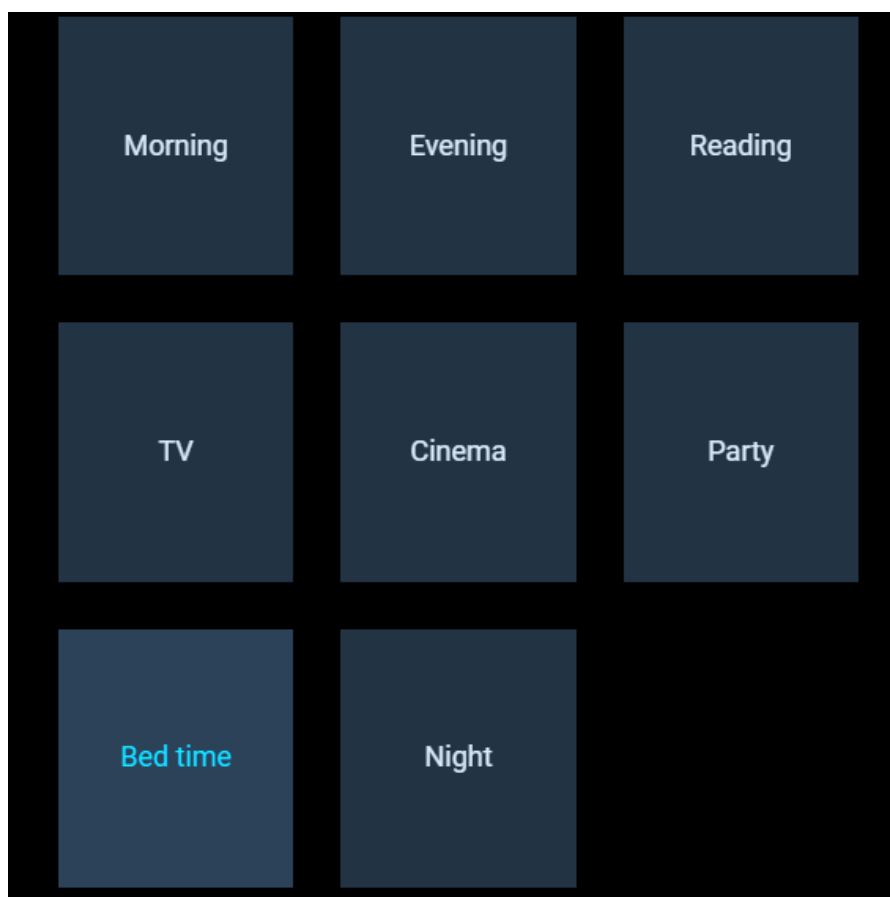


Рисунок 69. Виджет «Отбор»

Палитра цветов (colorpicker)

Виджет настройки цвета предлагает несколько способов отображения и обновления состояния элемента цвета в программе openHAB (рис. 70).

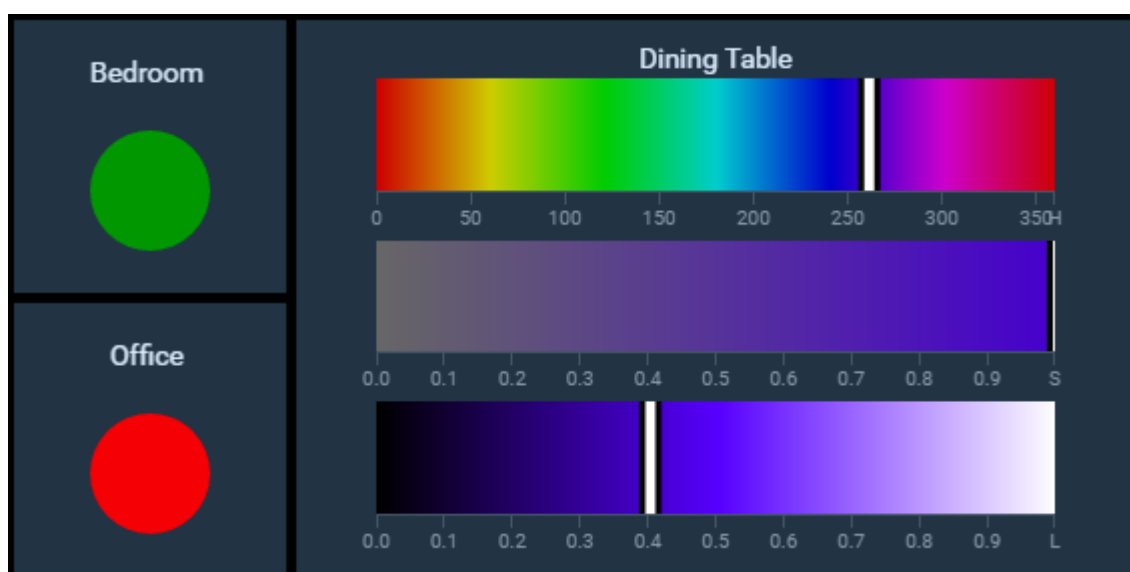


Рисунок 70. Виджет «Палитра»

Изображение

Виджет изображения может показывать картинку напрямую или через элемент строки openHAB. Также есть возможность обновлять его через равные промежутки времени (рис. 71).



Рисунок 71. Виджет «Изображение»

Рамка

Виджет фрейма отображает внешнюю веб-страницу в формате HTML `<iframe>` (рис. 72).

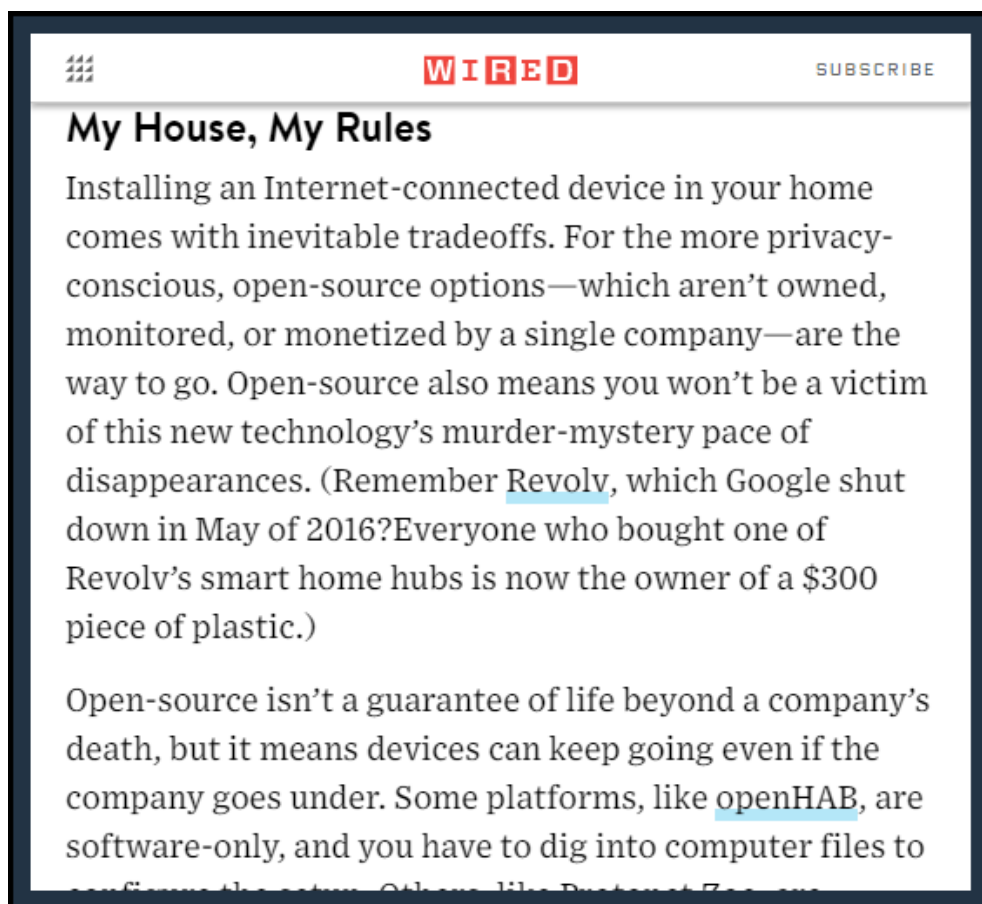


Рисунок 72. Виджет «Рамка»

Часы

Виджет показывает время с помощью аналоговых или цифровых часов. Его можно применить для отображения даты (рис. 73).

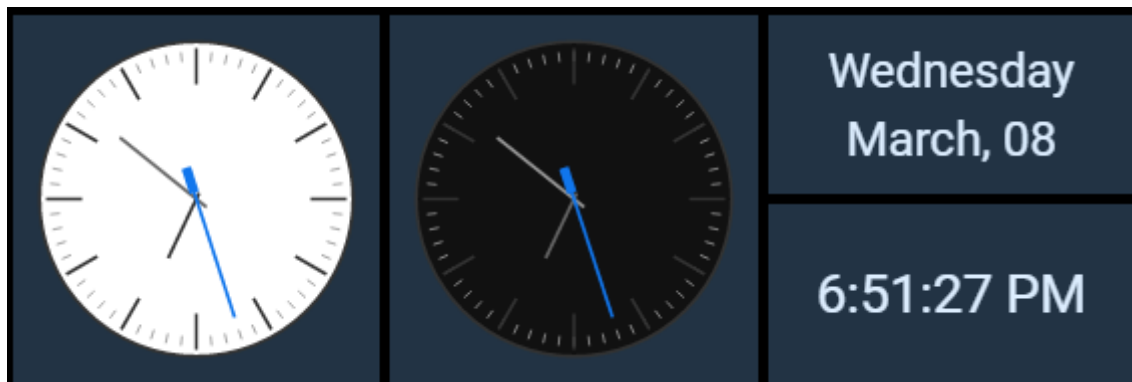


Рисунок 73. Виджет «Часы»

Диаграмма

Виджет диаграммы можно применить для отображения статистических данных за заданный период времени. Он может показать изображения диаграмм, созданных сервером (рис. 74).

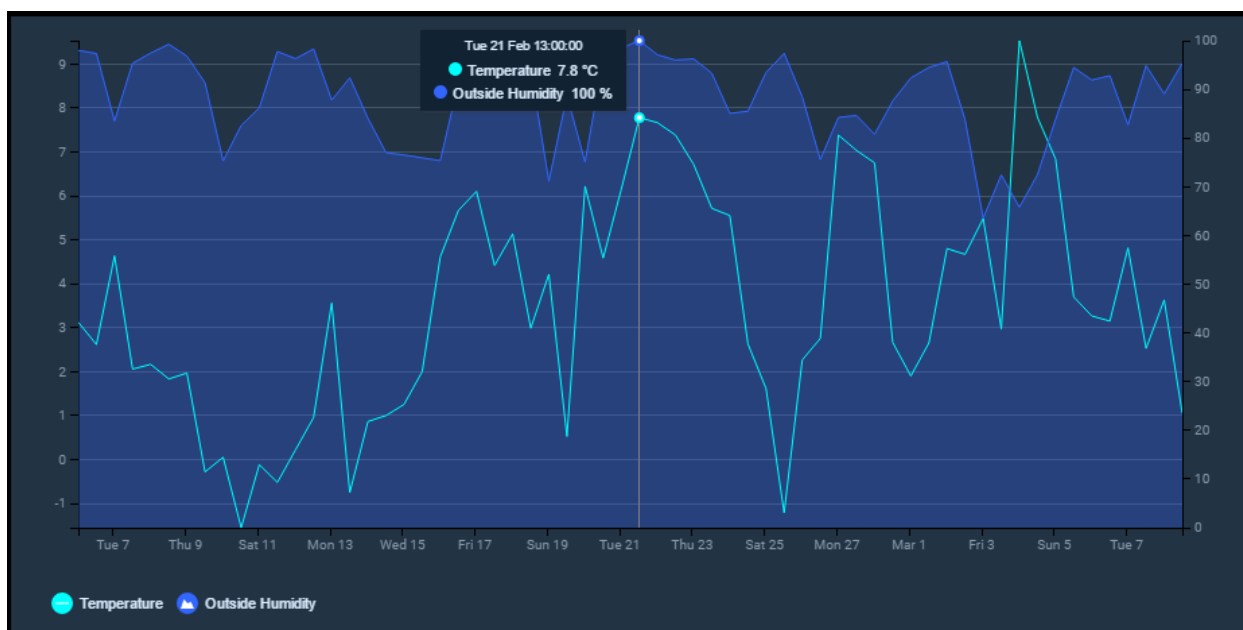


Рисунок 74. Виджет «Диаграмма»

Хронология (временная шкала)

Виджет временной шкалы подобен виджету диаграммы для нечисловых элементов. Он может отображать несколько «дорожек» элементов с фрагментами с цветовой кодировкой, представляющими изменения их состояния в течение выбранного периода. При наведении или касании внутри цветного

фрагмента отображаются сведения о состоянии элемента в данный момент (рис. 75).

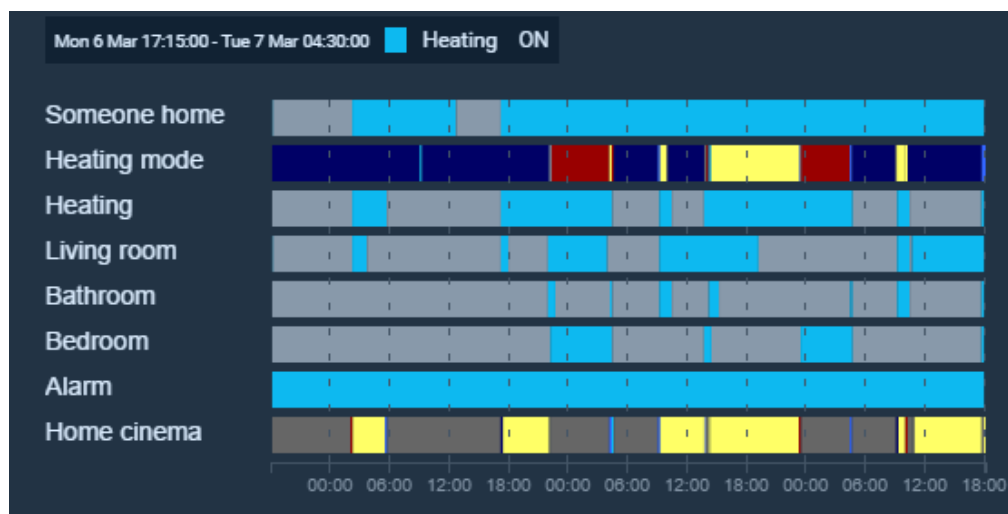


Рисунок 75. Виджет «Хронология»

Шаблон

Виджет шаблона позволяет отображать и размещать настроенный пользователем HTML-шаблон AngularJS внутри границ виджета. Данный шаблон имеет несколько вспомогательных функций и других средств для работы с openHAB.

Имея ограниченный набор навыков веб-разработки, пользователь может легко приступить к созданию собственных виджетов.

Шаблоны имеют ограничения и не обладают мобильностью, которая может понадобиться пользователю. Поэтому рациональным решением является создание собственного виджета.

Пользовательские виджеты

Пользовательские виджеты основаны на виджете шаблона, но предназначены для повторного использования, совместного использования и настройки. Пользовательский виджет — это шаблон AngularJS с соответствующим набором параметров конфигурации. Его можно добавить на информационные панели и настроить индивидуально, как встроенный виджет. Определения пользовательских виджетов хранятся в реестре на уровне конфигурации панели. Следовательно, каждая конфигурация панели имеет свои собственные настраиваемые виджеты.

Управление пользовательскими виджетами

Список пользовательских виджетов есть в конструкторе панели инструментов (щелкните/коснитесь значка шестеренки в раскрывающемся меню «Добавить виджет»). Также его можно найти с помощью кнопки «Управление» на экране настроек.

Пользовательские виджеты из списка можно создавать или импортировать из имеющегося файла .json.

Виджеты также могут быть получены из надстройки openHAB или пакета OSGi.: это «глобальные виджеты». Они не могут быть изменены или удалены (но могут быть клонированы, а затем изменены) и доступны для всех конфигураций панелей.

Контекстное меню : можно использовать для выполнения операций с виджетом: глобально предоставленные виджеты можно только клонировать, а определенные вручную или импортированные виджеты можно экспортировать или удалить.

Дизайнер виджетов

При нажатии на пользовательское определение виджета открывается конструктор виджетов. Он содержит три вкладки:

Код: Эта вкладка представляет собой редактор кода шаблона. Вы можете использовать сочетание клавиш Ctrl-S (или Cmd-S), чтобы сохранить виджет при редактировании кода.

Настройки: на этой вкладке находятся общие настройки виджета и структура настроек конфигурации, которые необходимо определить. Щелкните параметр «Добавить», чтобы добавить новый параметр конфигурации. Каждый параметр конфигурации должен иметь тип, технический идентификатор и другие необязательные атрибуты. Каждый тип параметра определяет элемент пользовательского интерфейса, представленный в конструкторе панели мониторинга при настройке экземпляров пользовательского виджета. Используйте кнопки со стрелками, чтобы перемещать параметры конфигурации вверх или вниз, и значок корзины, чтобы удалить их. При создании экземпляра значение параметров конфигурации устанавливается в области действия шаблона как `config.<setting_id>` (за исключением тех, которые относятся к типу `Icon`, которые определяют дополнительное значение, имя набора значков, как `config.<setting_id>_iconset`).

Предварительный просмотр: при переходе на эту вкладку тестовый экземпляр виджета отображается на пустой тестовой панели. Используйте ползунки, чтобы изменить размер виджета, чтобы просмотреть его в разных размерах. Если он определяет параметры конфигурации, они, скорее всего, должны быть установлены для этого предварительного просмотра с помощью значка шестеренки: это вызовет диалоговое окно конфигурации экземпляра виджета, как оно будет отображаться в дизайнера панели инструментов.

Не забудьте сохранить изменения кнопкой **Сохранить**.

2.5 Лабораторная работа «Автоматизация уличного освещения в программе OpenHAB»

Цель работы заключается в создании автоматизированной системы управления уличным освещением. Рассмотрим уличное освещение в городской среде. Чтобы сделать более комфортными условия пребывания людей в городе, а также повысить безопасность движения транспорта на дорогах, внедряют системы автоматизированного освещения. Освещение может включаться по показаниям датчика или по известному времени восхода и заката. Источником данных был выбран датчик света. Каждый раз, когда значение освещённости становится меньше заданной величины, освещение включается. При превышении данного показателя освещение автоматически отключается.

Алгоритм работы следующий:

- 1.Создайте новый проект и разместите в нем окружение моделируемого пространства (рис. 76).



Рисунок 76. Городская среда

2. Установите IP адрес и порт MQTT брокера в панели глобальных настроек в компоненте Параметры текущего соединения (рис. 77).

▼ Параметры текущего соединения	
IP адрес	192.168.1.58
Порт	1883

Рисунок 77. Параметры текущего соединения

3. Разместите датчик света (рис. 78), установите статус датчика ВКЛ и пропишите Event топик В компоненте MQTT - Показание: matt:topic:main:light_sensor:value/out.

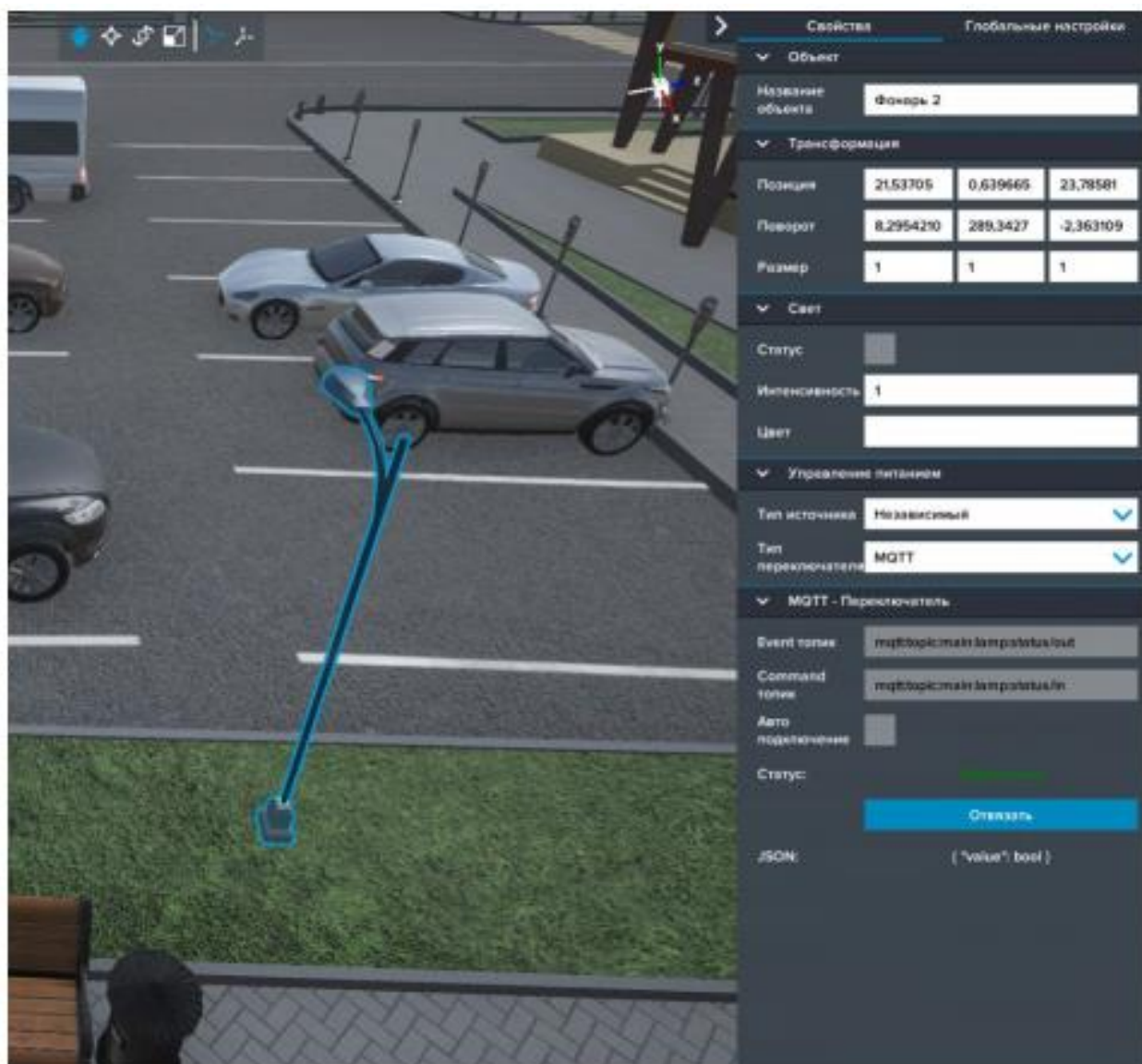


Рисунок 79 Добавление объекта освещения

7. В появившемся компоненте MQTT — Переключатель пропишите следующие топики:

a. Event топик - `mqtt:topic:main:lamp:status/out`,

b. Command топик - `mqtt:topic:main:lamp:status/in`

8. Нажмите на кнопку Связать и дождитесь установления статуса

Подключено.

9. Откройте в браузере OpenNav.

10. В панели Things добавьте новую вещь Generic MQTT Thing, которая будет связываться с виртуальным датчиком света.

11. Установите следующие параметры (рис. 80):

a. Unique ID: `light_sensor`

b. Label: `Light Sensor`

Рисунок 80 Параметры датчика света

12. После создания датчика света, перейдите во вкладку channels и добавьте новый канал со следующими параметрами (рис. 81):

- a.Channel Identifier: light_sensor_value
- b.Label: Значение
- c.Channel type: Number Value
- d.MQTT State Topic: mqtt:topic:main:light_sensor:value/out
- e.Incoming Value Transformations: JSONPATH:\$.value
- f.Outgoing Value Format: {"value":%s}

Рисунок 81 Добавление нового канала

13.далее добавьте новый Item к созданному каналу со следующими Параметрами (рис. 82):

a.Name: LightSensor_Value

b.Label: Значение датчика света

c.Type: Number

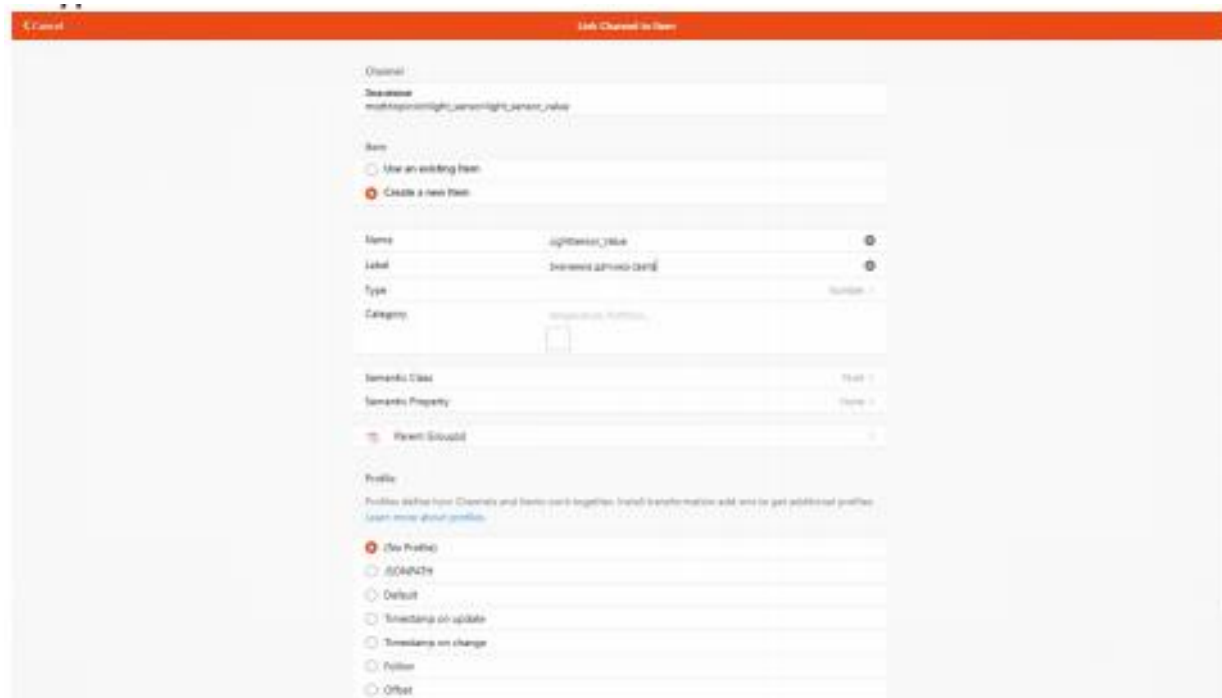


Рисунок 82 Добавление item

14.Убедитесь, что значение от виртуального датчика передается в Item.

15.Далее необходимо создать новую вещь (рис. 83), которая будет связываться с виртуальным переключателем освещения, со следующими параметрами:

a.Unique ID: lamp

b.Label: Lamp

16.Добавьте в вещь новый канал со следующими параметрами:

a.Channel Identifier: lamp_status

b.Label: Статус

c.Channel type: On/Off Switch

d.MQTT State Topic: mqtt:topic:main:lamp:status/out

e.MQTT Command Topic: mqtt:topic:main:lamp:status/in

f.Custom On/Open Value: true

g.Custom Off/Closed Value: false

h.Incoming Value Transformations: JSONPATH:\$.value

i.Outgoing Value Format: {"value":%s}

The screenshot shows the 'Add Channel' interface. At the top, there's an orange bar with the text 'Add Channel'. Below it, the 'Name' field contains 'lampstatus' and the 'Chasic' field contains 'Chasic'. The 'Channel type' section lists various options, with 'On/Off Switch' selected. The 'Configuration' section is expanded, showing a 'Show advanced' checkbox. It contains four fields: 'MQTT Data Topic' (mqtttopic/main/lampstatus/out), 'MQTT Command Topic' (mqtttopic/main/lampstatus/in), 'Custom On/Off Value' (true), and 'Custom On/Off State Value' (false). Each field has a small circular icon to its right.

Рисунок 83 Добавление новой вещи

17.Далее добавьте новый Item к созданному каналу со следующими параметрами (рис. 84):

- a.Name: Lamp_Status
- b.Label: Статус переключателя
- c.Type: Switch

18.Убедитесь, что статус от виртуального освещения передается в Item.

19.Далее необходимо создать новое правило во панели Rule, со следующими параметрами:

- a.Unique ID: light_auto
- b.Label: Автоматизация освещения

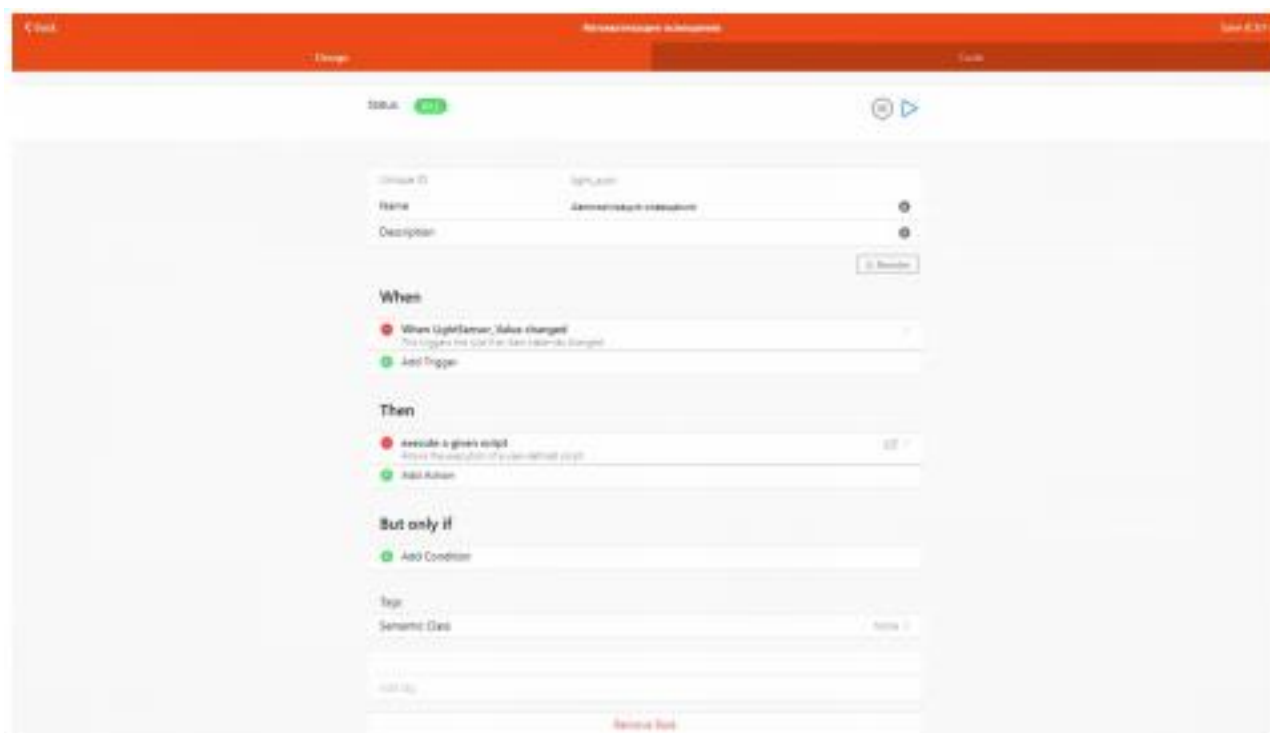


Рисунок 84 Добавление item к созданному каналу

20. Установите срабатывание правила на триггер. Для этого в пункте When добавьте триггер Item Event, выберите Значение датчика света, установите пункт срабатывания changed.

21. В пункт Then добавьте действие Run Script и выберите Design with Blockly и соберите следующий скрипт (рис. 85):



Рисунок 85 Скрипт

22. После сохранения правила, перейдите в виртуальный проект и начните изменять значение глобального освещения. При значении показания датчика света меньше 10000 освещение должно включаться. При значении больше 10000 выключаться (рис. 86).

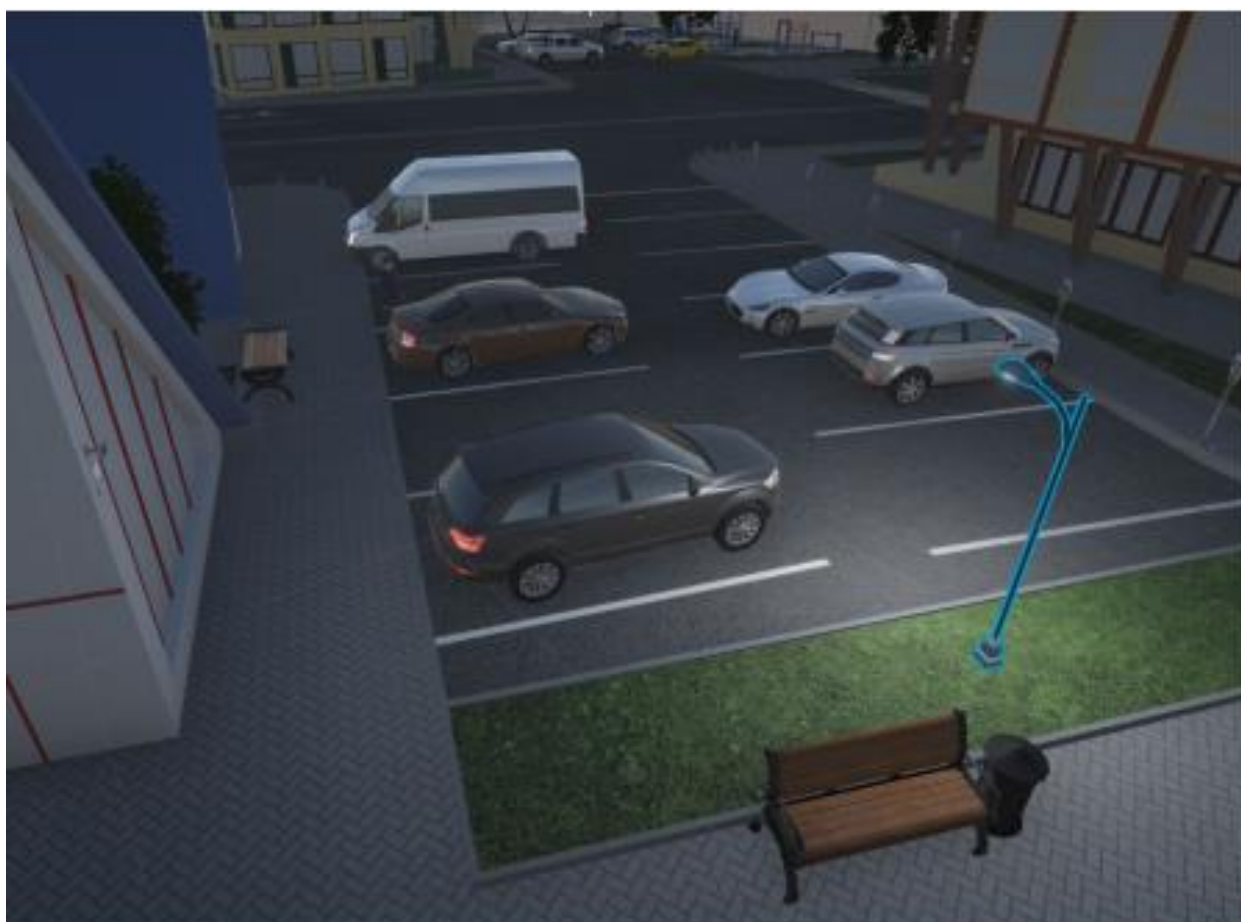


Рисунок 86 Работа датчика света

23.Далее перейдите в НАВPanel и настройте дашборд (рис.87).

24.Добавьте виджет Модель для вывода значения датчика света. В настройках установите следующие параметры:

- а.Имя: Показания датчика света
- б.openHAB Item: LightSensor_Value
- с.Формат: %.2f

Параметры Модели

Имя: Показания датчика света

openHAB Item: # LightSensor_Value

Размер шрифта:

Единица:

Формат: %.2f

☐ Использовать формат с сервера, если доступно

Иконка фона: Выберите набор иконок ▾

Иконка: Выберите набор иконок ▾

Расположение: ☐ Значение справа

Удалить Отмена Сохранить

Рисунок 87 Настройка дашборда

25.Добавьте виджет Переключатель для управления освещением со следующими параметрами (рис. 88):

a.Имя: Освещение

b.openHAB: Lamp_Status

Настройки переключателя

Имя: Освещение

openHAB Item: Lamp_Status

Настройки отображения: ☐ Скрыть имя ☐ Скрыть иконку ☐ Скрыть ON/OFF

Иконка фона: Выберите набор иконок ▾

Иконка: Выберите набор иконок ▾

Удалить Отмена Сохранить

Рисунок 88 Виджет Переключатель

26.Убедитесь, что на дашборд выводится верное показание виртуального датчика света и идёт управление освещением (рис. 89).

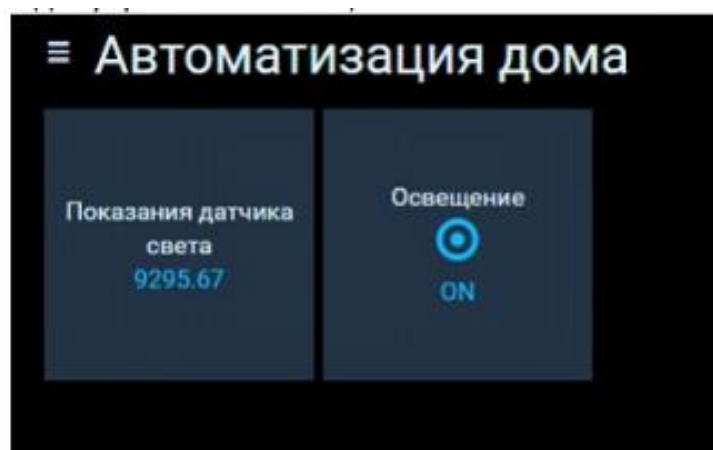


Рисунок 89 Дашборд

27. Проверьте работу системы автоматизированного управления освещением на лабораторном стенде и в виртуальном лабораторном комплексе ProgramLab.

Список литературы

1. Капля, В. И. Модели и методы искусственного интеллекта : электронное учебное пособие / В. И. Капля, Е. В. Капля ; Министерство науки и высшего образования Российской Федерации, Волгоградский государственный технический университет, Волжский политехнический институт (филиал). – Волжский : Волгоградский государственный технический университет, 2019. – 80 с.
2. Системы управления технологическими процессами и информационные технологии : учебное пособие для вузов / В. В. Троценко, В. К. Федоров, А. И. Забудский, В. В. Комендантов. — 2-е изд., испр. и доп. — Москва : Издательство Юрайт, 2022. — 136 с. — (Высшее образование). — ISBN 978-5-534-09938-6. — Текст : электронный // Образовательная платформа Юрайт [сайт].
3. Adamu Zungeru M., Chuma J., Lebekwe C., Phalaagae P., Gaboitaolelwe J. Green Internet of Things Sensor Networks: Applications, Communication Technologies, and Security Challenges / Springer, 2020. — 135 p.
4. Ahad Md Atiqur Rahman, Antar Anindya Das, Ahmed Masud. IoT Sensor-Based Activity Recognition: Human Activity Recognition / Springer, 2021. — 214 p.
5. Adryan Boris, Obermaier Dominik, Fremantle Paul. The Technical Foundations of IoT / Artech House, 2017. — 480 p.
6. Ruecker Bernd. Practical Process Automation: Orchestration and Integration in Microservices and Cloud Native Architectures / O'Reilly Media, 2021. — 294 p.
7. Alcaraz C. (Ed.) Security and Privacy Trends in the Industrial Internet of Things / Springer, 2019. — 312 p.
8. Modrzyk Nicolas. Real-Time IoT Imaging with Deep Neural Networks: Using Java on the Raspberry Pi 4 / Apress, 2020. — 241 p.

Сведения об авторе

Закожурников Сергей Сергеевич, к.т.н., доцент кафедры ВМиП ИПТИП.