# Практическая работа №3 – Обработка событий в системах Интернета вещей

## Правила-скрипты в WirenBoard

Для контроллера WirenBoard можно писать правила, например: "Если температура датчика меньше 18°C, включи нагреватель". Правила можно создать и редактировать на странице **Rules**. Правила пишутся на простом языке, похожем на JavaScript и позволяют создавать правила ("включай свет с 10:00 до 18:00") или виртуальные устройства (например, кнопка в интерфейсе, которая включает и отключает всё освещение в здании вместе).

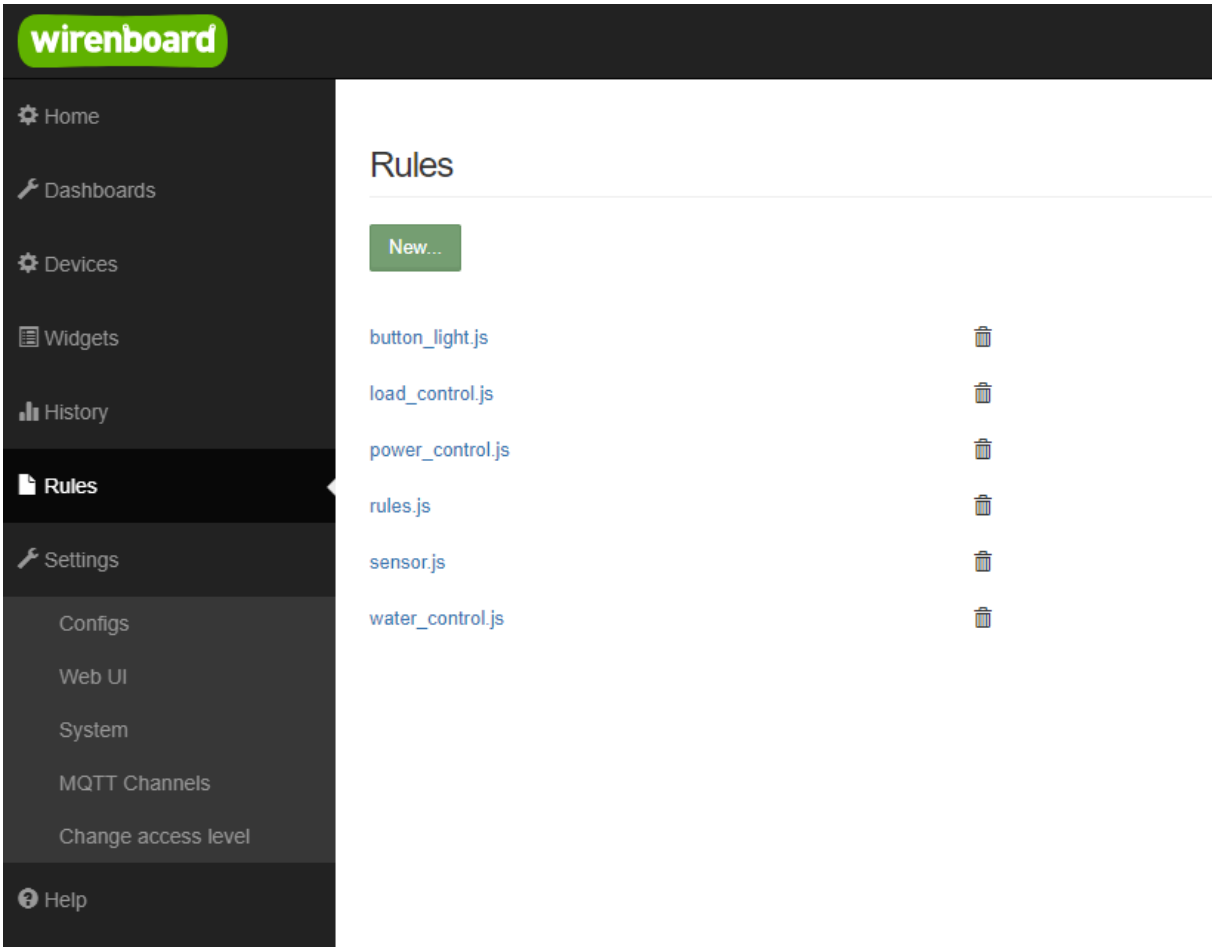


*Рисунок 1 – Окно правил с базовыми сценариями*

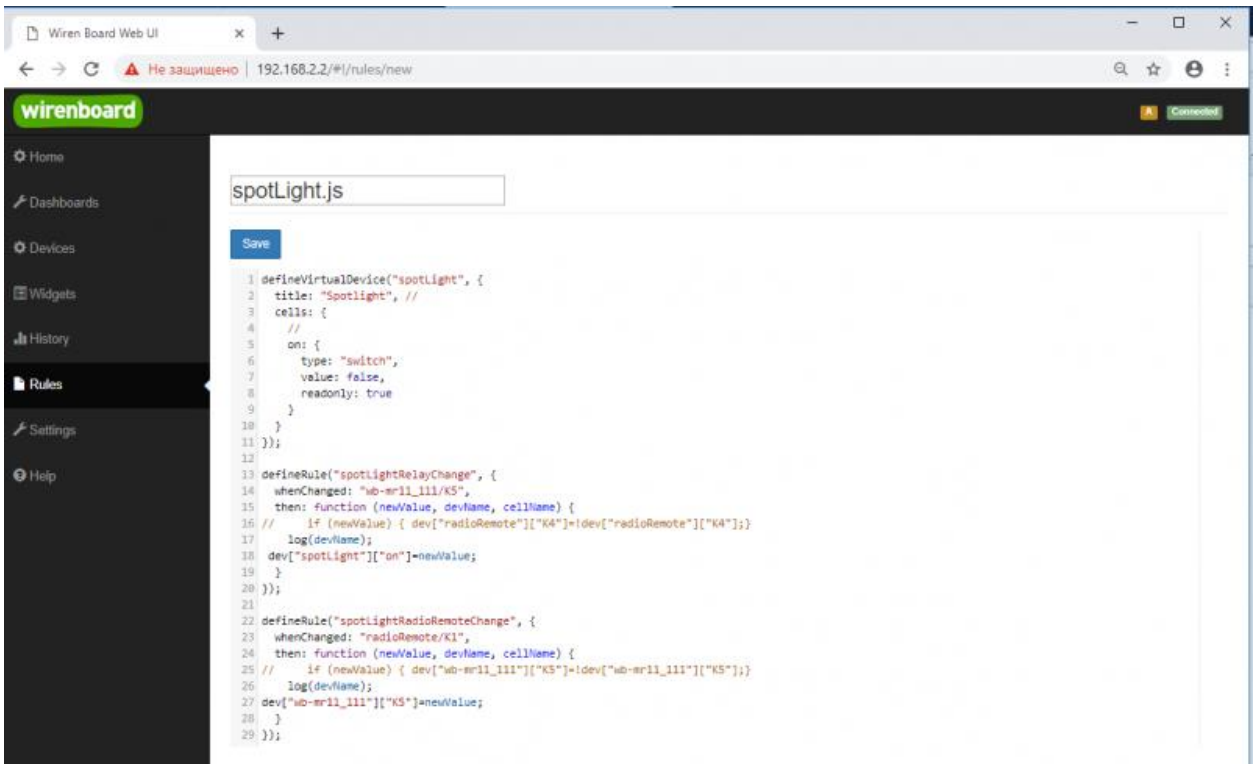


*Рисунок 2 – Скрипт, открытый для просмотра и редактирования*

## Создание и редактирование правил

Список файлов с правилами находится на странице *Rules* веб-интерфейса. Для редактирования правила нажмите на название файла.

Для создания нового правила, нажмите на *New...*, вверху введите название (латинские буквы и цифры, в качестве расширения укажите *.js*), в основное поле введите текст скрипта, и нажмите *Save* вверху.

Правило начинает сразу работать после сохранения, если в нём нет ошибок.

В одном файле можно хранить неограниченное количество правил. Но обычно в одном файле хранятся правила с близкими функциями.

### Пример правила

Для начала разберём простое правило "при превышении температуры - выключи обогреватель". Температуру получаем с датчика 1-Wire, обогреватель подключён к Реле 1 внешнего релейного модуля WB-MRM2.

```
defineRule("heater_control", { //название правила - "контроль обогревателя", может быть произвольным

  whenChanged: "wb-w1/28-0115a48fcfff", //при изменении состояния датчика 1-Wire с идентификатором 28-0115a48fcfff

  then: function (newValue, devName, cellName) { //выполняй следующие действия

    if ( newValue > 30) { //если температура датчика больше 30 градусов

      dev["wb-mrm2_130"]["Relay 1"] = false; //установи Реле 1 модуля WB-MRM2 с адресом 130 в состояние "выключено"

    } else {
```

```
        dev["wb-mrm2_130"]["Relay 1"] = true; //установи Реле 1 модуля WB-MRM2
с адресом 130 в состояние "включено"
    }
  }
});
```

1. Первая строка - кодовое слово *defineRule* и название правила
2. Вторая строка - кодовое слово для определения, когда выполняется правило, - *whenChanged* - "при изменении параметра", далее название параметра, при изменении которого запустится правило - температура с датчика 1-Wire. Название параметра записывается в виде "Device/Control", где названия *Device* и *Control* для каждого параметра можно найти на странице *Settings* веб-интерфейса, в таблице *MQTT Channels*.
3. Третья строка - начало функции, которая будет исполняться
4. Четвертная строка - условие - "если значение температуры больше порогового, то ...". Значение параметра записывается в виде *dev[Device][Control]* - заметьте, оно отличается от вида записи параметра, при изменении которого запускается правило, потому что там речь идёт о *параметре*, а здесь - о *значении* того же параметра.
5. После выставлюется значения для реле в каждом случае - *false* - "выключено", *true* - "включено". Названия *Device* и *Control* для реле смотрим всё в той же таблице *MQTT Channels*, на странице *Settings* веб-интерфейса.

Ознакомиться с другими примерами правил можно здесь: https://wirenboard.com/wiki/Rule_Examples

### Типы правил defineRule

**whenChanged**

Правило срабатывает при любых изменениях значений параметров или функций В примере кнопка подключена ко входу A1 контроллера. При нажатии на кнопку срабатывает реле 1 устройства MRM2-mini, при отжатии реле возвращается в исходное состояние.

```
defineRule("test_rule", { //имя правила test_rule
  whenChanged: "wb-gpio/A1_IN",
  then: function (newValue, devName, cellName) {
    dev["wb-mrm2-mini_2"]["Relay 1"] = newValue;
  }
});
```

**asSoonAs**

Правила, задаваемые при помощи asSoonAs, срабатывают в случае, когда значение, возвращаемое функцией, заданной в asSoonAs, становится истинным при том, что при предыдущем просмотре данного правила оно было ложным.

```
defineRule({
  asSoonAs: function() {
```

```
    return dev["wb-gpio"]["A2_IN"];
  },
  then: function (newValue, devName, cellName) {
    dev["wb-mrm2-mini_2"]["Relay 2"] = true;
  }
});
```

При нажатии на кнопку, подключенную к входу A2, включаем реле, а при нажатии на кнопку, подключенную к входу A3 — выключаем.

**when**

Правила, задаваемые при помощи when срабатывают при каждом просмотре, при котором функция, заданная в when, возвращает истинное значение. При срабатывании правила выполняется функция, заданная в свойстве when.

```
defineVirtualDevice("test_button2", {
  title: "test_relay2",
  cells: {
    "switch_on": {
    type: "pushbutton",
    value: false,
    },
    "switch_off": {
    type: "pushbutton",
    value: false,
    }
  }
});

defineRule({
  when: function() {
    return dev["test_button2"]["switch_on"];
  },
  then: function (newValue, devName, cellName) {
    dev["wb-mrm2-mini_2"]["Relay 2"] = true;
  }
});

defineRule({
  when: function() {
    return dev["test_button2"]["switch_off"];
```

```
  },
  then: function (newValue, devName, cellName) {
    dev["wb-mrm2-mini_2"]["Relay 2"] = false;
  }
});
```

При нажатии на кнопку switch_on, включаем реле, а при нажатии на кнопку,switch_off — выключаем.

**cron-правила**

Отдельный тип правил - cron-правила. Такие правила задаются следующим образом:

```
defineRule("crontest_hourly", {
  when: cron("@hourly"),
  then: function () {
    log("@hourly rule fired");
  }
})
```

Вместо @hourly здесь можно задать любое выражение, допустимое в стандартном crontab, например, 00 00 20 * * (секунды минуты часы выполнять правило каждый день в 20:00). Помимо стандартных выражений допускается использование ряда расширений, см. описание формата выражений используемой cron-библиотеки.

### Массивы

Массив определяется короткой записью:

```
fruits = ["One", "Two", "Three"];
```

Обращение к элементу:

```
fruits[0]; //One
```

### Таймеры

**setTimeout();**

С помощью данного таймера можно отсрочить выполнение правила на определенное время. В примере после нажатия кнопки включается пищалка и затем выключается спустя 2 секунды.

```
defineVirtualDevice("test_buzzer", {
title: "Test Buzzer",
  cells: {
    enabled: {
      type: "pushbutton",
      value: false
```

```
    }
  }
});

defineRule({
  whenChanged: "test_buzzer/enabled",
    then: function (newValue, devName, cellName) {
    dev["buzzer"]["enabled"] = true;
    setTimeout(function () {
      dev["buzzer"]["enabled"] = false;
    }, 2000);
  }
});
```

**setInterval();**

```
defineVirtualDevice("test_buzzer", {
  title: "Test Buzzer",
  cells: {
    enabled: {
      type: "pushbutton",
      value: false
    }
  }
});



var test_interval = null;

defineRule({
  whenChanged: "test_buzzer/enabled",
  then: function (newValue, devName, cellName) {
    var n = 0;
    if (dev["test_buzzer"]["enabled"]){
      test_interval = setInterval(function () {
        dev["buzzer"]["enabled"] = !dev["buzzer"]["enabled"];
        n = n+1;
        if (n >= 10){
          clearInterval(test_interval);
        }
```

```
      }, 500);
    }
  }
});
```

**startTimer();**

Запускает однократный таймер. При срабатывании таймера происходит просмотр правил, при этом timers.<name>.firing для этого таймера становится истинным на время этого просмотра.

```
defineVirtualDevice("test_buzzer", {
  title: "Test Buzzer",
  cells: {
    enabled: {
      type: "switch",
      value: false
    }
  }
});

defineRule("1",{
  asSoonAs: function () {
    return dev["test_buzzer"]["enabled"] ;
  },
  then: function () {
    startTimer("one_second", 1000);
    dev["buzzer"]["enabled"] = true;//включаем пищалку
  }
});

defineRule("2",{
  when: function () { return timers.one_second.firing; },
  then: function () {
    dev["buzzer"]["enabled"] = false;//выключаем пищалку
    dev["test_buzzer"]["enabled"] = false;
  }
});
```

**startTicker();**

Запускает периодический таймер с указанным интервалом. В примере правило 1 запускает таймер с интервалом 1 сек. Вызывая срабатывание правила 2. Метод stop() приводит к остановке таймера.

```
defineVirtualDevice("test_buzzer", {
  title: "Test Buzzer",
  cells: {
    enabled: {
      type: "switch",
      value: false
    }
  }
});


defineRule("1",{
  asSoonAs: function () {
    return dev["test_buzzer"]["enabled"] ;
  },
  then: function () {
    startTicker("one_second", 1000);
  }
});
defineRule("2",{
  when: function () { return timers.one_second.firing; },
  then: function () {
    dev["buzzer"]["enabled"] = !dev["buzzer"]["enabled"];
    if (dev["test_buzzer"]["enabled"] == false){
      timers.one_second.stop();
    }
  }
});
```

### Виртуальные устройства defineVirtualDevice

Правила делятся на два типа: непосредственно правила (defineRule) и виртуальные устройства (defineVirtualDevice).

Виртуальные устройства — это появляющиеся в веб-интерфейсе новые элементы управления - например, кнопка-выключатель, которая на самом деле выключает два устройства одновременно. Она не привязана напрямую ни к какому физическому устройству, а действия при её нажатии определяются написанным вами скриптом. При

написании скрипта вы можете создать виртуальное устройство для включения/выключения тех или иных управляющих алгоритмов и установки их параметров.

```
defineVirtualDevice("wb-1", {
  title: "Отопление",
  cells: {
    "Температура": {
    type: "range",
    max: 24,
    value: 20,
    },
    "Вкл.": {
    type: "switch",
    value: false,
    }
  }
});
```

**Типы устройств (type):**

**switch** — переключатель. Может принимать значения true или false. По-умолчанию доступен для изменения пользователем.

**wo-switch** — переключатель, аналог switch. Может принимать значения true или false. По-умолчанию не доступен для изменения пользователем.

**pushbutton** — кнопка. Может принимать значения true. По-умолчанию доступна для нажатия.

**range** — ползунок. Может принимать значения от 0 до max. По-умолчанию доступен для изменения пользователем.

**rgb** — специальный тип для задания цвета. Кодируется 3 числами от 0 до 255, разделенными точкой с запятой. Например „255;0;0“ задает красный цвет. По-умолчанию доступен для изменения пользователем.

**alarm** — индикатор. Может принимать значения true или false.

**text** — текстовое поле. По-умолчанию не доступно для редактирования пользователем.

**value** — значение с плавающей точкой. По-умолчанию не доступно для редактирования пользователем.

Так же существуют еще 14 специальных типов. Все они аналогичны value, но имеют соответствующие подписи в интерфейсе.

| Type | meta/type | units | value format |
|---|---|---|---|
| Temperature | temperature | °C | float |

| | | | |
|---|---|---|---|
| Relative humidity | rel_humidity | %, RH | float, 0 - 100 |
| Atmospheric pressure | atmospheric_pressure | millibar (100 Pa) | float |
| Precipitation rate (rainfall rate) | rainfall | mm per hour | float |
| Wind speed | wind_speed | m/s | float |
| Power | power | watt | float |
| Power consumption | power_consumption | kWh | float |
| Voltage | voltage | volts | float |
| Water flow | water_flow | m^3 / hour | float |
| Water total consumption | water_consumption | m^3 | float |
| Resistance | resistance | resistance | float |
| Gas concentration | concentration | ppm | float (unsigned) |
| Heat power | heat_power | Gcal / hour | float |
| Heat energy | heat_energy | Gcal | float |

**value:**

Обязательное поле. При создании устройства в первый раз его значение будет установлено в значение по-умолчанию. В дальнейшем значения сохраняются в специальное хранилище в постоянной памяти и восстанавливаются при загрузке сценария.

**forceDefault:**

Если необходимо каждый раз при перезагрузке скрипта восстанавливать строго определённое значение (т.е. не восстанавливать предыдущее сохранённое), нужно добавить в описание контрола поле **forceDefault: true**

**max:**

Для параметра типа **range** может задавать его максимально допустимое значение.

**readonly:**

Когда задано истинное значение — устройство становится не доступным для редактирования пользователем. Если надо предоставить пользователю возможность редактировать значение, следует добавить в описание **readonly: false**

### Сообщения в лог

В зависимости от функции сообщение классифицируется как отладочное (debug), информационное (info), предупреждение (warning) или сообщение об ошибке (error).

```
defineVirtualDevice("test_buzzer", {
  title: "Test Buzzer",
  cells: {
    enabled: {
      type: "pushbutton",
      value: false
    }
  }
});

defineRule({
  whenChanged: "test_buzzer/enabled",
  then: function (newValue, devName, cellName) {
    log.info('info');
    log.debug('debug');
    log.error('error');
    log.warning('warning');
    log('log!'); //сокращение для log.info();
    debug('deb!'); //сокращение для log.debug();
  }
});
```

Выполнение произвольной команды runShellCommand();

```
defineVirtualDevice("test_button", {
  title: "Test Button",
  cells: {
    enabled: {
      type: "pushbutton",
      value: false
    }
  }
});

defineRule({
  whenChanged: "test_button/enabled",
  then: function (newValue, devName, cellName) {
    runShellCommand("mosquitto_pub -t
'/devices/test_button/controls/enabled/meta/readonly' -r -m 1");
    setTimeout(function () {
```

```
        runShellCommand("mosquitto_pub -t
'/devices/test_button/controls/enabled/meta/readonly' -r -m 0");
    }, 5000);
  }
});
```

## Задание практической работы №3

1. Разработайте сценарии для обработки событий согласно вариантам и приведите в отчете его листинг с комментариями:

**Первый** пункт выполняется на wb-dev-kit v2 (серебристый), **второй** – на wb-dev-kit v3 (чёрный).

Для второго пункта может быть полезна формула для перевода значения из одного диапазона в другой $y = (x - x_{min}) \cdot \frac{y_{max} - y_{min}}{x_{max} - x_{min}} + y_{min}$ , где

$x$ – исходное значение, которое переводим,

$x_{\min}, x_{\max}$ – границы исходного диапазона,

$y_{\min}, y_{\max}$ – границы нового диапазона,

$y$ – результат (значение в новом диапазоне).

| № варианта | Сценарии |
|---|---|
| **1** | 1. Включение и выключение воды по датчику движения<br>2. Изменение яркости лампы от уровня шума в помещении |
| **2** | 1. Включение и изменение цвета диодной ленты по кнопкам<br>2. Изменение высоты звукового сигнала от скорости вращения вентилятора |
| **3** | 1. Изменение цвета диодной ленты по концентрации CO2<br>(зеленый цвет – концентрация в норме, красный – повышена)<br>2. Изменение скорости вращения вентилятора от концентрации углекислого газа |
| **4** | 1. Включение и выключения световых индикаторов по кнопкам<br>2. Изменение яркости RGB ленты в зависимости от интенсивности движения |
| **5** | 1. Включение и выключение вентилятора по датчику движения<br>2. Изменение высоты звукового сигнала от яркости лампы |
| **6** | 1. Включение и выключение вентилятора по температуре<br>2. Изменение яркости RGB ленты от потребляемой мощности чемодана |
| **7** | 1. Включение и выключения вентилятора по концентрации CO2<br>2. Изменение яркости RGB ленты от освещённости |

2. Не забудьте выгрузить сценарий на личное устройство, чтобы избежать его потери