Weidmüller

Starter-Kit | u-control

Application Note for a simple Temperature controller with MQTT Publisher in Node-RED

Abstract:
This Application Note describes the implementation of a demo application for the u-control Starter Kit IoT. The application utilizes all key components of the starter kit to build a simple temperature controller. This temperature controller is extended by a MQTT publisher to send sensor values to a public MQTT broker. The reception of these values with a MQTT subscriber is shown as well.

Hardware reference

No.	Component name	Article No.	Hardware / Firmware version
1	STARTERKIT-UC20-WL2000-IOT	2666060000	UC20-WL2000-IOT FW 1.10.0 or higher

Software reference

No.	Software name	Article No.	Software version
1	Recent web browser (Microsoft Edge, Firefox, Chrome)	-	-
2	Test broker (mosquitto.org)	-	-

File reference

No.	Name	Description	Version
1	-	-	-

Contact
Weidmüller Interface GmbH & Co. KG
Klingenbergstraße 26
32758 Detmold, Germany
www.weidmueller.com

For any further support please contact your local sales representative:
https://www.weidmueller.com/countries

Content

1 Warning and Disclaimer 4
2 Requirements 5
3 PLC setup in u-create web 6
3.1 Devices and I/O 6
3.2 Global Variables 10
3.3 Visualization 11
4 Creating a simple Temperature Controller in Node-RED 13
5 Publish the Controller's State to a MQTT Broker 20
6 Further Information 23

1 Warning and Disclaimer

Warning
Controls may fail in unsafe operating conditions, causing uncontrolled operation of the controlled devices. Such hazardous events can result in death and / or serious injury and / or property damage. Therefore, there must be safety equipment provided / electrical safety design or other redundant safety features that are independent from the automation system.

Disclaimer
This Application Note / Quick Start Guide / Example Program does not relieve you of the obligation to handle it safely during use, installation, operation and maintenance. Each user is responsible for the correct operation of his control system. By using this Application Note / Quick Start Guide / Example Program prepared by Weidmüller, you accept that Weidmüller cannot be held liable for any damage to property and / or personal injury that may occur because of the use.

Note
The given descriptions and examples do not represent any customer-specific solutions, they are simply intended to help for typical tasks. The user is responsible for the proper operation of the described products. Application notes / Quick Start Guides / Example Programs are not binding and do not claim to be complete in terms of configuration as well as any contingencies. By using this Application Note / Quick Start Guide / Example Program, you acknowledge that we cannot be held liable for any damages beyond the described liability regime. We reserve the right to make changes to this application note / quick start guide / example at any time without notice. In case of discrepancies between the proposals Application Notes / Quick Start Guides / Program Examples and other Weidmüller publications, like manuals, such contents have always more priority to the examples. We assume no liability for the information contained in this document. Our liability, for whatever legal reason, for damages caused using the examples, instructions, programs, project planning and performance data, etc. described in this Application Note / Quick Start Guide / Example is excluded.

Security notes
In order to protect equipment, systems, machines and networks against cyber threats, it is necessary to implement (and maintain) a complete state-of-the-art industrial security concept. The customer is responsible for preventing unauthorized access to his equipment, systems, machines and networks. Systems, machines and components should only be connected to the corporate network or the Internet if necessary and appropriate safeguards (such as firewalls and network segmentation) have been taken.

2 Requirements

The Quick Start Guide for the Starter Kit IoT (QSG0033) describes the initial setup and configuration of the u-control PLC as well as the download and usage of the application described in this Application Note. Further information and a link to the download section for the Quick Start Guide and example applications is given in Chapter 6. An internet connection is required, to push data to an IoT-service and to install additional nodes.

3 PLC setup in u-create web

To make use of the key components of the Starter Kit IoT, we implement a simple temperature controller. This chapter describes the hardware of the Starter Kit and how to map inputs / outputs to global variables using the browser-based engineering tool u-create web. Chapter 4 shows how to access these variables and the implementation of the temperature controller in Node-RED.

3.1 Devices and I/O

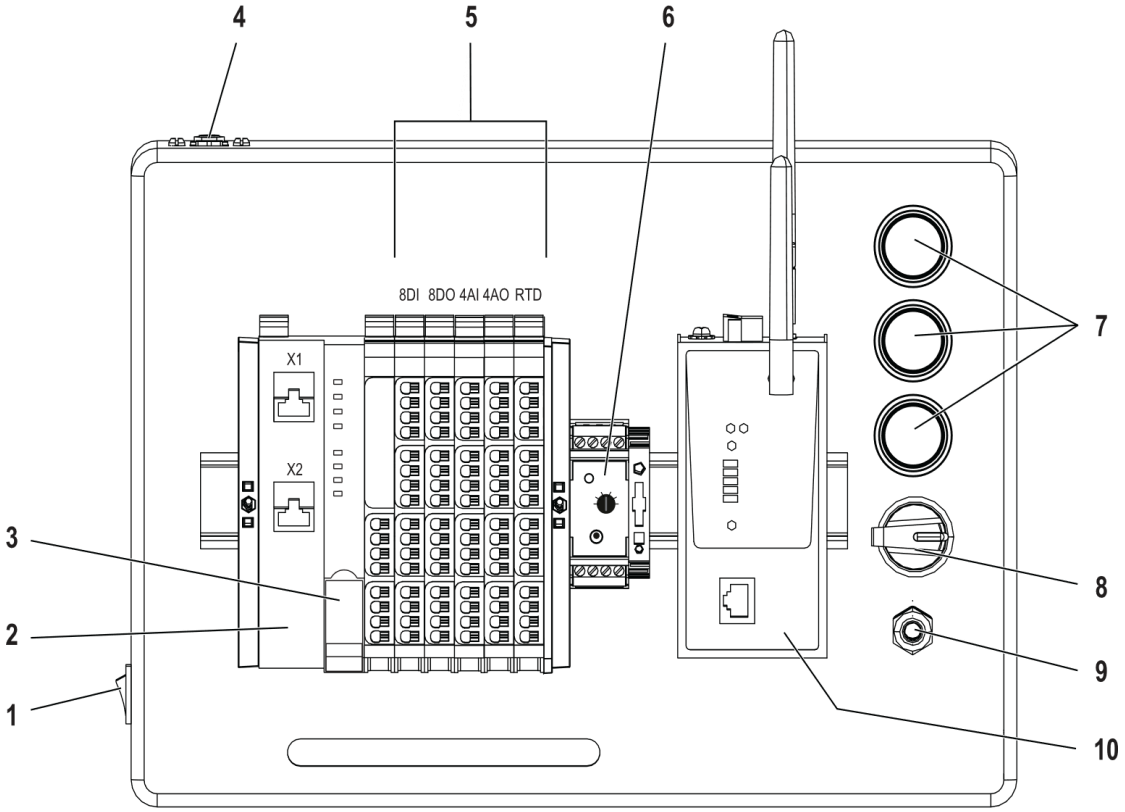


Figure 1: Overview of the starter kit hardware

The starter kit comes with five I/O modules (Pos. 5 above) providing different inputs and outputs:

Table 1: List of I/O modules that come with the starter kit

Module Type	Description	Order No. (link to product page)
UR20-8DI-P-2W	8 channel digital input module	1315180000
UR20-8DO-P	8 channel digital output module	1315240000
UR20-4AI-UI-16	4 channel analog input module	1315620000
UR20-4AO-UI-16	4 channel analog output module	1315680000
UR20-4AI-RTD-DIAG	4 channel temperature input module	1315700000

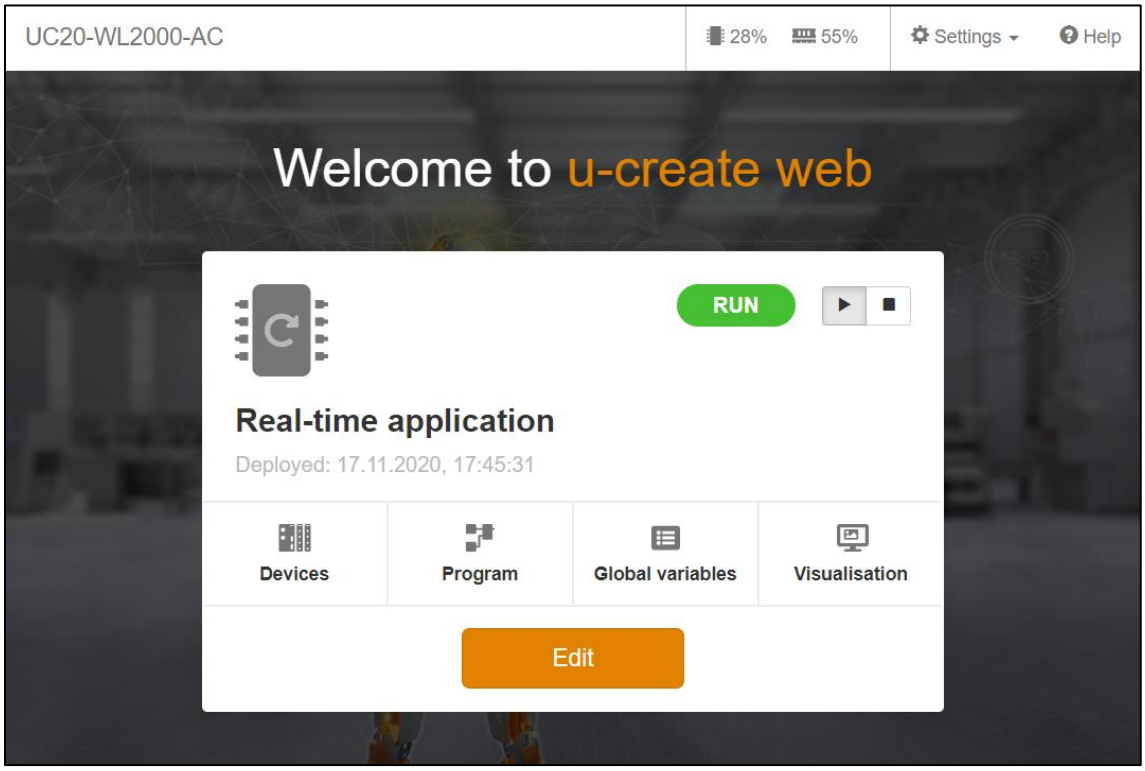


Figure 2: Welcome screen of u-create web, edit button highlighted

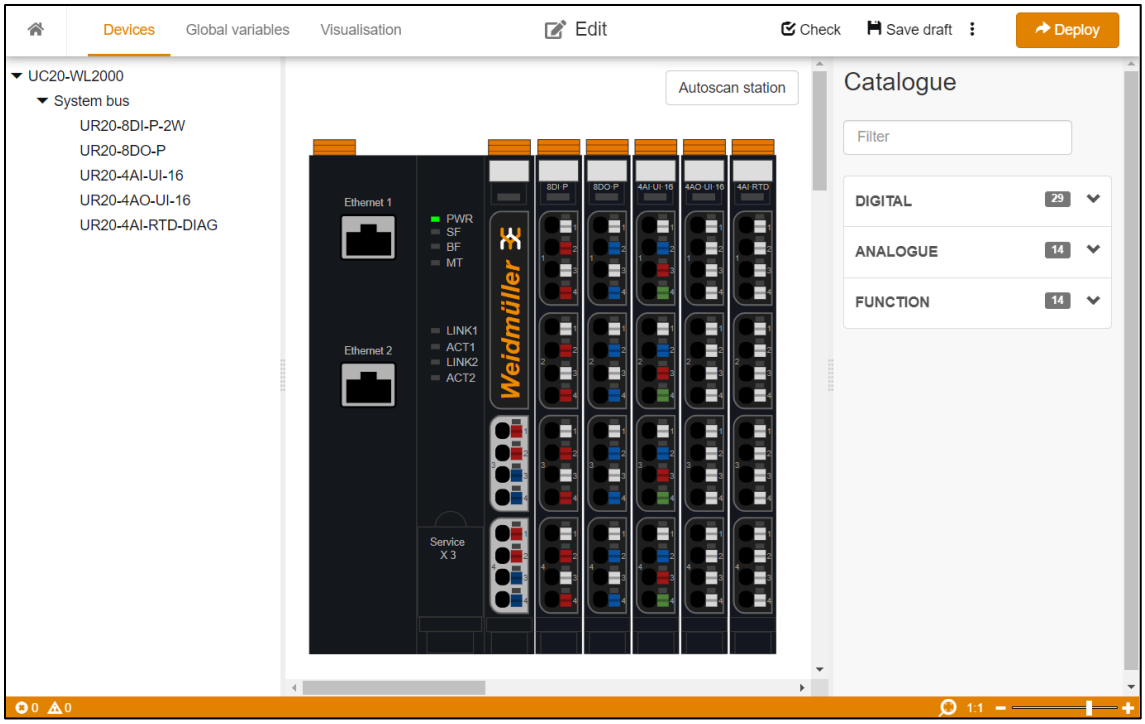


Figure 3: u-create web in planning (e.g. edit) mode, devices tab

The following devices are already connected to the I/O modules:

Table 2: List of the starter kit hardware and connections

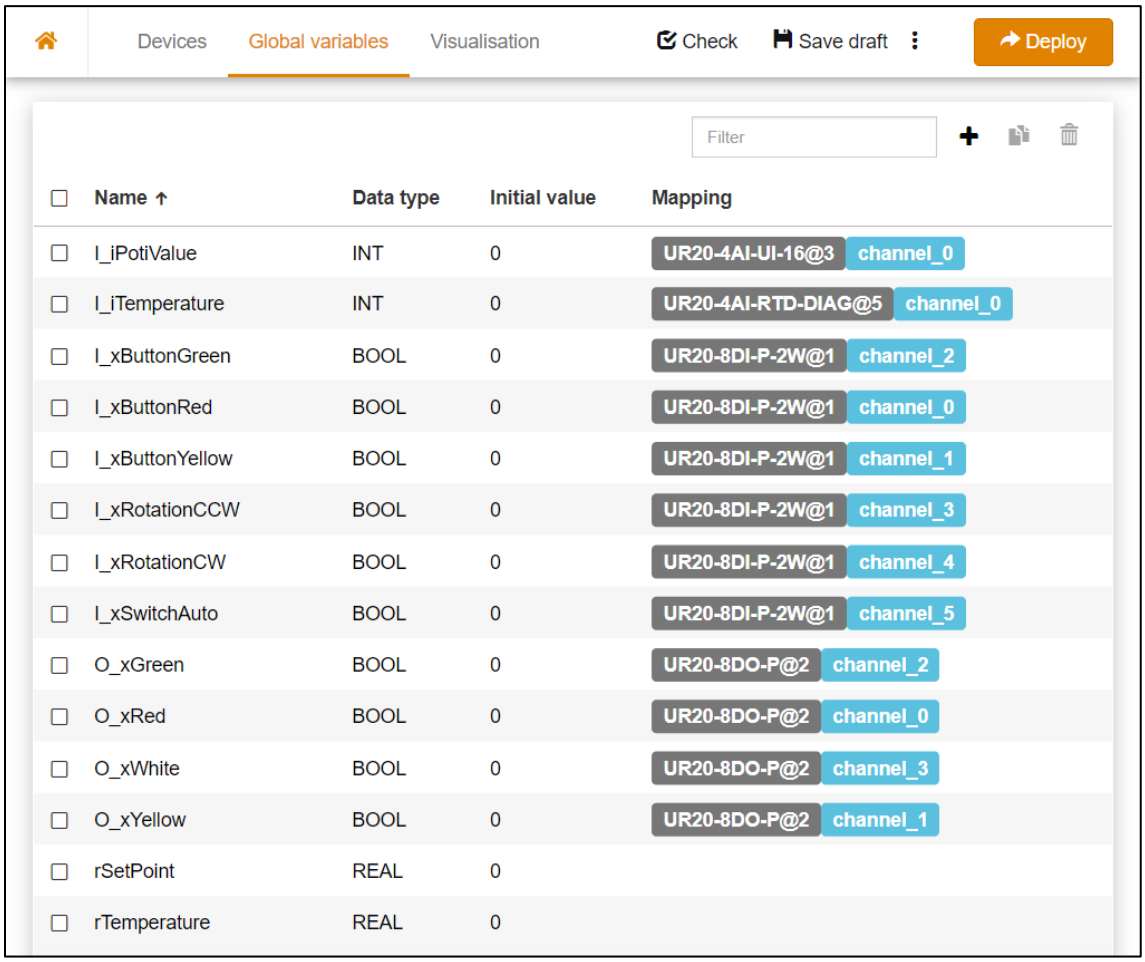
Pos		Hardware	I/O Module	Connection
7		Pushbutton red	UR20-8DI-P-2W	Channel 0 / DI 1.1
7		Pushbutton yellow	UR20-8DI-P-2W	Channel 1 / DI 1.3
7		Pushbutton green	UR20-8DI-P-2W	Channel 2 / DI 2.1
8		Rotary switch	UR20-8DI-P-2W	Channel 3 / DI 2.3
8		Rotary switch	UR20-8DI-P-2W	Channel 4 / DI 3.1
6	Hand/Auto	Analogue encoder	UR20-8DI-P-2W	Channel 5 / DI 3.3
7		LED red	UR20-8DO-P	Channel 0 / DO 1.1

7		LED yellow	UR20-8DO-P	Channel 1 / DO 1.3
7		LED green	UR20-8DO-P	Channel 2 / DO 2.1
8		LED white	UR20-8DO-P	Channel 3 / DO 2.3
6	OUT (Y)	Analogue encoder	UR20-4AI-UI-16	Channel 0 / AI 1.1
6	IN (YR)	Analogue encoder	UR20-4AO-UI-16	Channel 0 / AO 1.1
6	IN (YGND)	Analogue encoder	UR20-4AO-UI-16	Channel 0 / AO 1.4
9	Pt100 +	Temperature sensor	UR20-4AI-RTD-DIAG	Channel 0 / AI 1.1
9	Pt100 -	Temperature sensor	UR20-4AI-RTD-DIAG	Channel 0 / AI 1.4

To make these devices accessible via Node-RED, the respective I/O's must be mapped to global variables in u-create web. To do this, open the planning view using the "Edit" button on the welcome page (Figure 2). The planning view will open on the "Devices" tab.

In case the shown configuration differs from the physical configuration of your starter kit (Figure 1 for reference), click on the "Autoscan station" button (Figure 3). Check the connections between modules if modules are not detected correctly.

3.2 Global Variables



Name	Data type	Initial value	Mapping
I_iPotiValue	INT	0	UR20-4AI-UI-16@3 channel_0
I_iTemperature	INT	0	UR20-4AI-RTD-DIAG@5 channel_0
I_xButtonGreen	BOOL	0	UR20-8DI-P-2W@1 channel_2
I_xButtonRed	BOOL	0	UR20-8DI-P-2W@1 channel_0
I_xButtonYellow	BOOL	0	UR20-8DI-P-2W@1 channel_1
I_xRotationCCW	BOOL	0	UR20-8DI-P-2W@1 channel_3
I_xRotationCW	BOOL	0	UR20-8DI-P-2W@1 channel_4
I_xSwitchAuto	BOOL	0	UR20-8DI-P-2W@1 channel_5
O_xGreen	BOOL	0	UR20-8DO-P@2 channel_2
O_xRed	BOOL	0	UR20-8DO-P@2 channel_0
O_xWhite	BOOL	0	UR20-8DO-P@2 channel_3
O_xYellow	BOOL	0	UR20-8DO-P@2 channel_1
rSetPoint	REAL	0	
rTemperature	REAL	0	

Figure 4: Planning view of global variables in u-create web

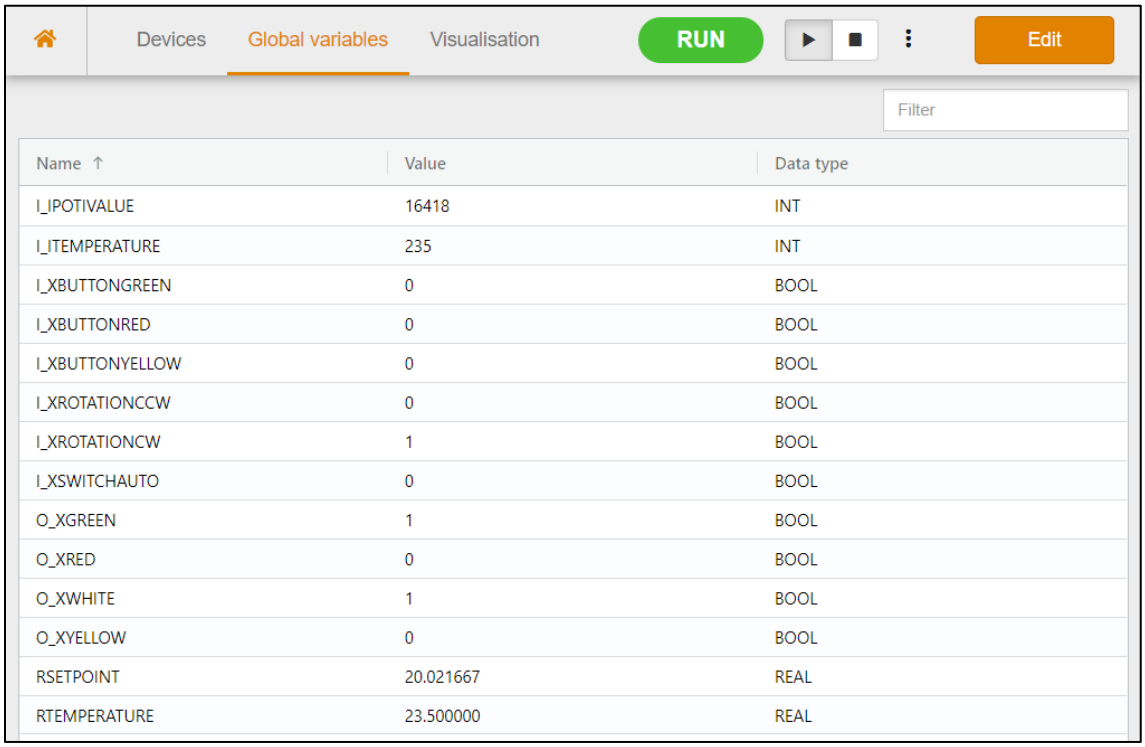
Figure 4 shows the “Global variables” tab of the planning view with all required variables already configured. All but two variables are mapped to an I/O. The variables “`rSetPoint`” and “`rTemperature`” will be used to display data from Node-RED on the visualisation integrated into u-create web and therefore are not mapped to an I/O.

Note the naming convention used in this example:

1. Variables are preceded by “I_” (or “O_”) if they are mapped to an input (or output).
2. Next comes a lowercase letter indicating the data type of the variable:
   - x: Bool
   - i: Integer
   - r: Real
3. Next comes the name. It represents the function or device the variable is mapped to.

This naming convention is by no means authoritative, it is a design decision meant to ease the use of these variables in Node-RED. There, variables are only identified by their name, their data type or mapping is not clear unless included in the name itself.

Internally, u-create web converts all variable names to uppercase. This can be seen in the live view shown in Figure 5. The live view allows to monitor the value of all variables.



Name	Value	Data type
I_IPOTIVALUE	16418	INT
I_ITEMPERATURE	235	INT
I_XBUTTONGREEN	0	BOOL
I_XBUTTONRED	0	BOOL
I_XBUTTONYELLOW	0	BOOL
I_XROTATIONCCW	0	BOOL
I_XROTATIONCW	1	BOOL
I_XSWITCHAUTO	0	BOOL
O_XGREEN	1	BOOL
O_XRED	0	BOOL
O_XWHITE	1	BOOL
O_XYELLOW	0	BOOL
RSETPOINT	20.021667	REAL
RTEMPERATURE	23.500000	REAL

Figure 5: Live view of global variables in u-create web

3.3 Visualization

u-create web includes a powerful yet easy to use visualization tool. It allows the creation of and interaction with visualisations like shown in Figure 6 using only a web browser, even on mobile devices like tablets.

Figure 7 shows the configuration of one UI element in the planning view. Different elements provide different means of interaction. While a LED can only be used as an indicator (output) for the variable specified under “Data binding”, other elements like a button can be used as an input. The data binding of the UI elements shown here should be self-explanatory, given the descriptive names of the variables shown previously in Figure 4.

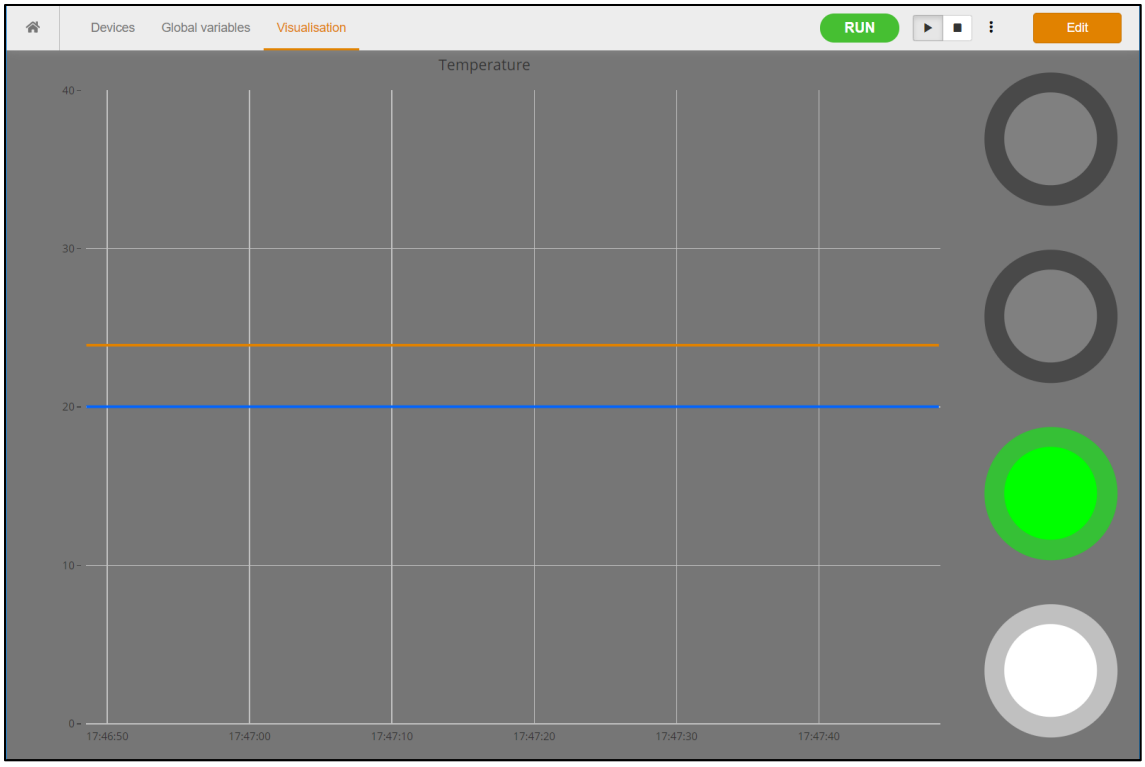


Figure 6: Live view of the visualisation in u-create web

The goal of the visualisation shown here is to represent the physical LEDs of the starter kit and to provide a live view of the actual temperature and the set point of the controller.

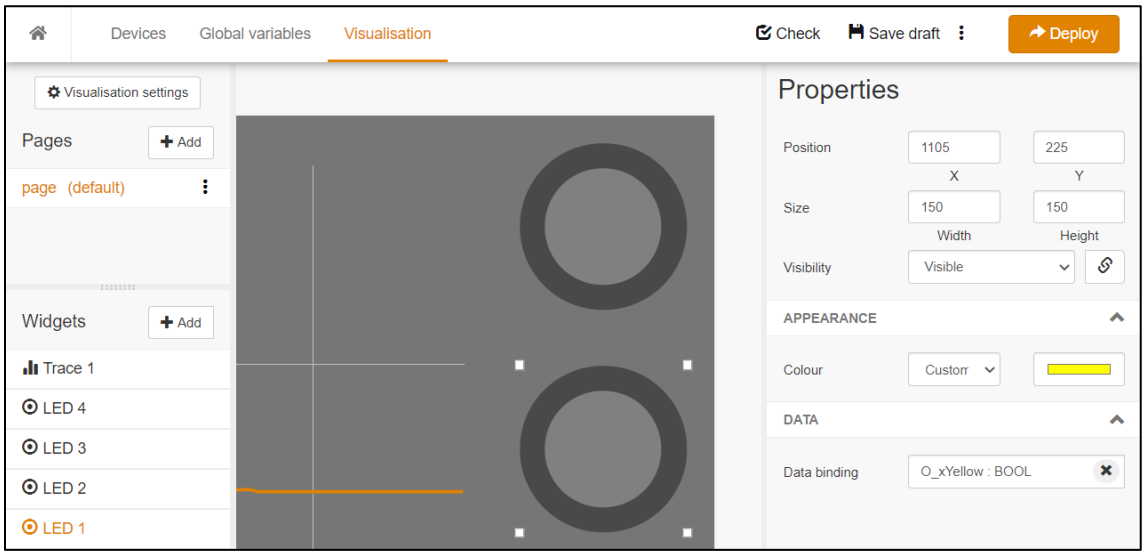


Figure 7: Planning view of the visualisation in u-create web

4 Creating a simple Temperature Controller in Node-RED

Now that the I/O are mapped to global variables and the visualisation has been created, we will focus on the implementation of the actual application in Node-RED. To access Node-RED click on the respective button of the u-create web welcome page (see Figure 8).

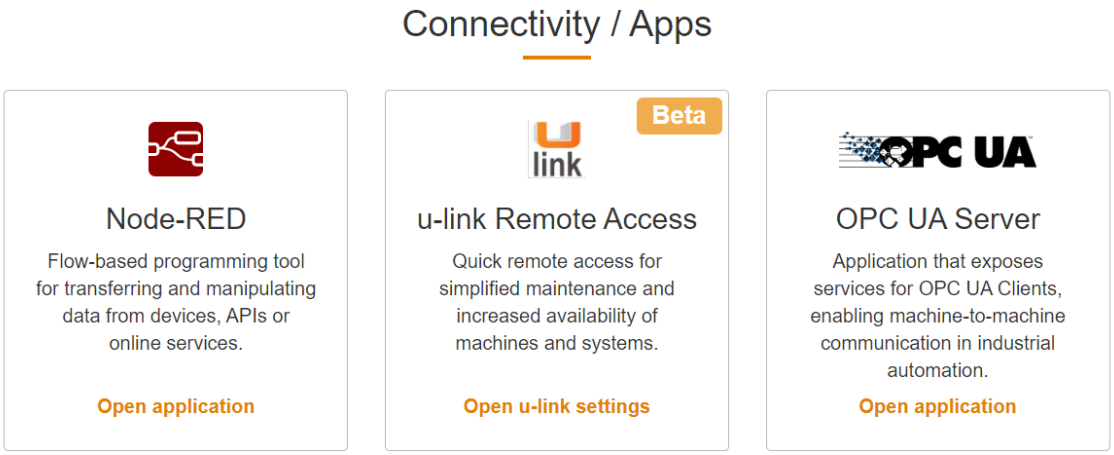


Figure 8: Application menu in u-create web

Node-RED is a browser based, visual programming tool which builds on Node.js. Programs are implemented as “flows”. Nodes can have a single input and multiple outputs to propagate events and information through the flow. Figure 9 shows the complete flow for the temperature controller.

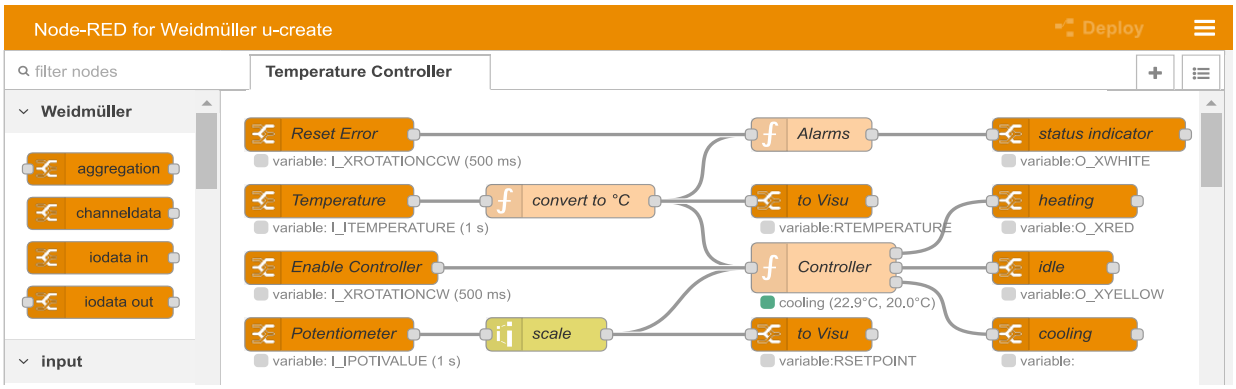


Figure 9: The flow in Node-RED, a simple temperature controller that lights the LEDs of the starter kit

Although the appearance of a Node-RED flow might resemble a functional block diagram (FBD, IEC 61131-3), Node-RED does not come with logic blocks. While a few third-party libraries offer Boolean logic elements, we will implement the temperature controller in JavaScript.

To interact with the controller, we will use the following user inputs:

 Reset Alarm Condition

 Controller off

 Controller on

 Temperature set point [10°C, 30°C]

The controllers state is indicated by the LEDs:

 Heating

 Idle

 Cooling

 On: Controller active; Flashing: Alarm triggered

To interact with the physical process – in this case the temperature sensor, the user inputs, and the LEDs – Weidmüller provides the “`iodata-in`” and “`iodata-out`” nodes. These are preinstalled on the u-control IoT and u-control Web controllers. We use a total of ten “`iodata`” nodes, four “`iodata-in`” and six “`iodata-out`”. Figure 10 shows the configuration of one “`iodata-in`” node. We use the node in the “Single Variable” mode, where it polls one specific variable with a defined rate and outputs it’s value.

Please note that using multiple “`iodata-in`” nodes in this mode can be less performant than using a single “`iodata-in`” node in the mode “All Variables”. The latter is advised if a multitude of global variables shall be read in Node-RED.

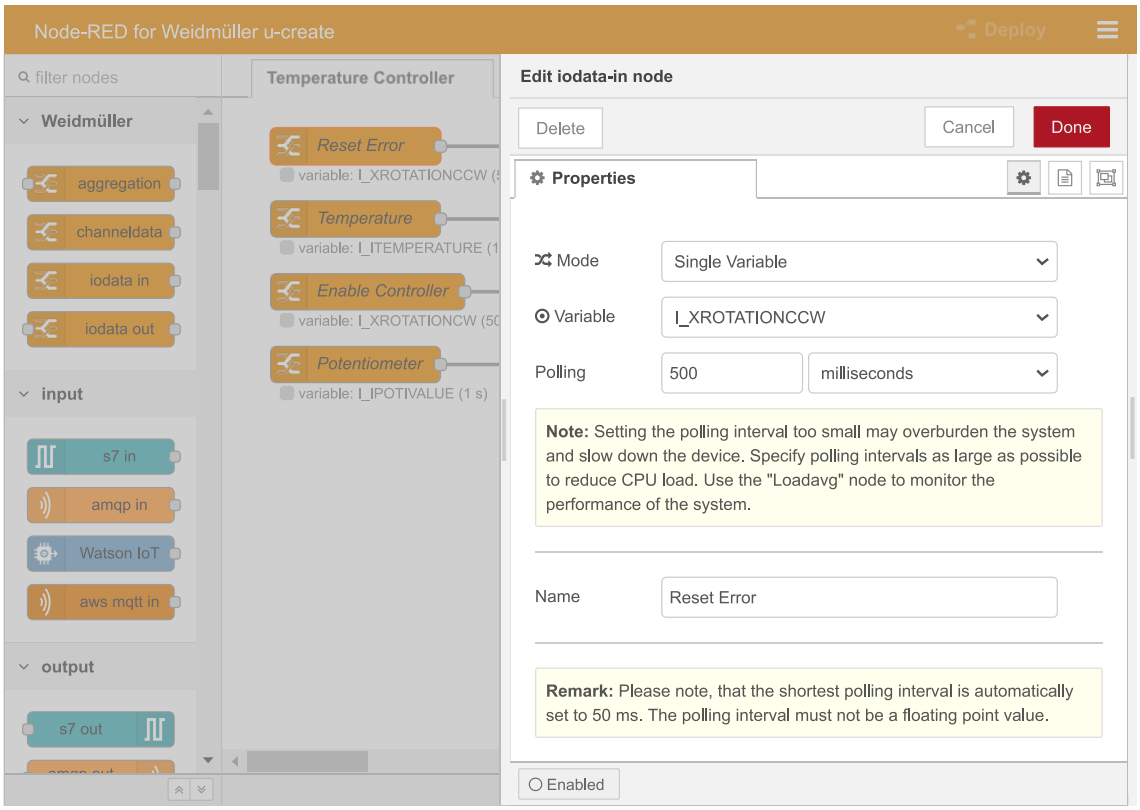


Figure 10: Configuration of the iodata-in node with a polling rate and mapping to a global variable

The "`iodata-out`" nodes configuration is analogue to the "`iodata-in`" node, with the exception that no polling rate can be specified (see Figure 11). Once the node receives a message with a valid value, it will update the global variable. The structure of these messages will be described later in this chapter.

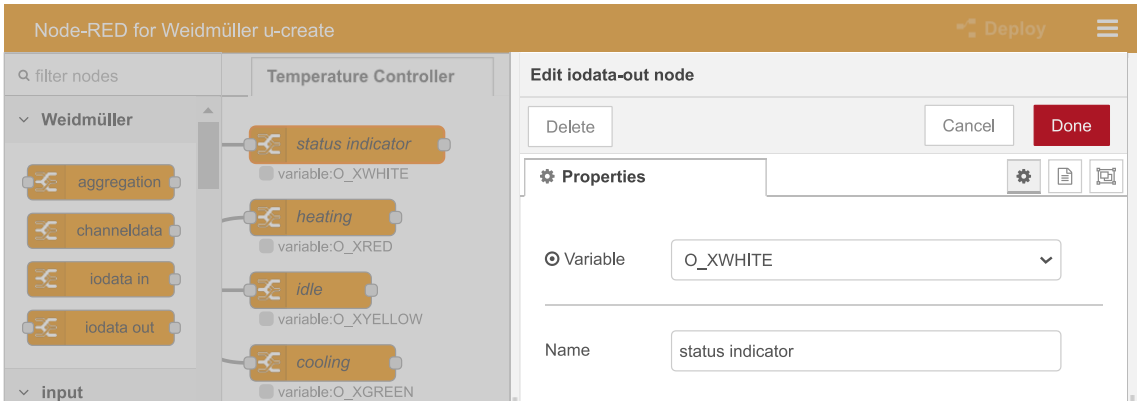


Figure 11: Configuration of the iodata-out node and mapping to a global variable

Node-RED provides a node to map an input value to a target range, the range node. Figure 12 shows the configuration of the range node used to map the input from the analogue encoder to a range of 10 to 30 for the temperature set point. Note that the input range is 0 to 32767 with 32767 being the maximum value of a signed 16-bit integer, the datatype of the variable "`I_IPOTIVALUE`".

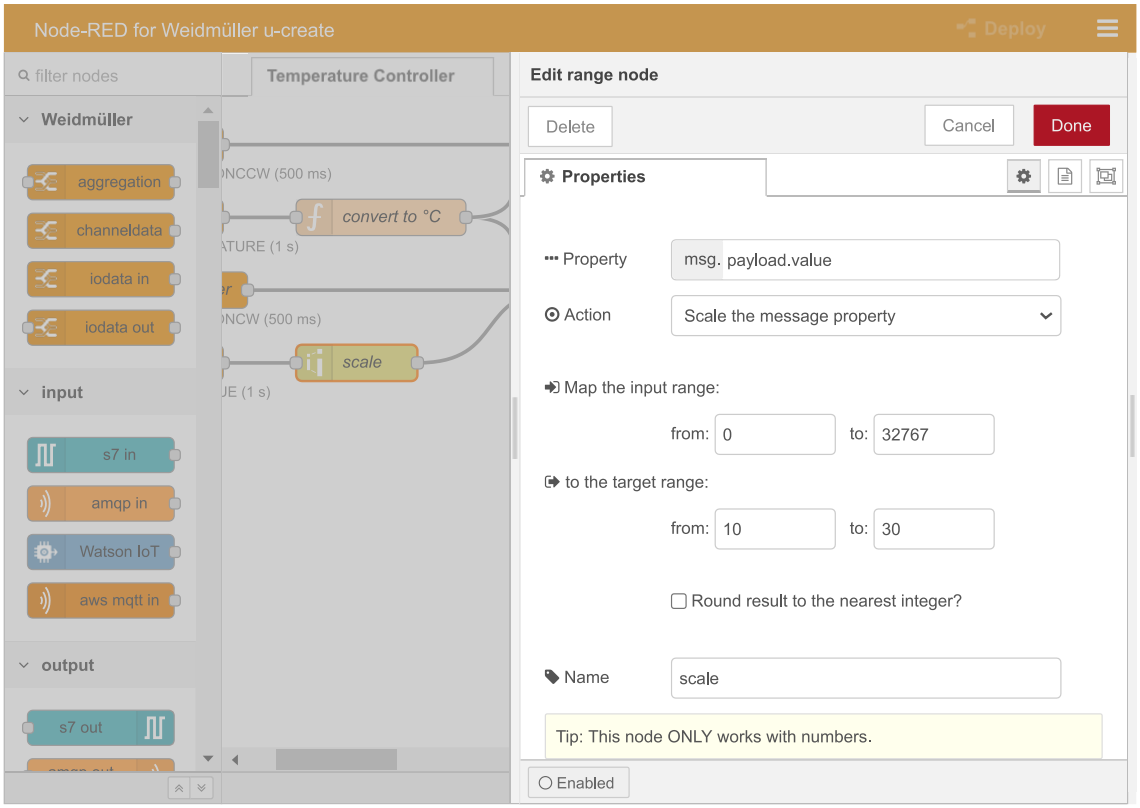


Figure 12: Configuration of the range node, to map the analog input from the rotary encoder to the range of 10 to 30

The temperature sensor reading is given in 0.1 degrees centigrade, 123 equals 12.3°C. We will use a function node and one line of JavaScript (JS) to divide the reading by ten, see Listing 1.

```
msg.payload.value = parseInt(msg.payload.value) / 10


return msg
```

Listing 1: Code of the very simple "convert to °C" function node

The sensor reading is stored in the input message under "`msg.payload.value`" as a string. We parse this string to an integer and divide this integer by 10, effectively turning it into a float in the JS. This float then is stored back in "`msg.payload.value`" and the whole message is returned.

The temperature is used by both, the “Alarms” function node, and the “Controller” function node. Additionally, it is written to the global variable “RTEMPERATURE” using an “`iodata-out`”, to make it available for the visualisation.

```
let ledState;


switch(msg.payload.name){
    case "I_ITEMPERATURE":
        if (msg.payload.value > 30 || msg.payload.value < 10)
            flow.set("tempAlarm", true)
        break;


    case "I_XROTATIONCCW":
        if (msg.payload.value == 1)
            flow.set("tempAlarm", false)


        if (flow.get("tempControllerRun") && !flow.get("tempAlarm")) {
            ledState = true
        } else if (flow.get("tempAlarm")) {
            ledState = !context.get("ledState") || 0
        } else {
            ledState = false
        }


        context.set("ledState", ledState)
        msg = {"payload": {"value": ledState }}


        return msg
}
```

Listing 2: Code of the “Alarms” function node that monitors the rotary switch and the actual temperature

Listing 2 shows the implementation of the “Alarms” node. This node monitors the temperature sensor reading and disables the temperature controller if the reading is out of the range 10°C to 30°C. The alarm state can be reset using the rotational switch turned to the north-east / CCW position. Nodes in Node-RED can only have a single input. Since this node accepts messages from two different “`iodata-in`” nodes, we must distinguish between messages containing the temperature sensor value and messages containing the state of the rotary switch.

The “`iodata-in`” node stores the name of the global variable it polled under “`msg.payload.name`”, which we evaluate using a switch-statement.

For the temperature reading, the case-statement is very simple. “`msg.payload.value`” is compared against the limits of the allowed range. If it breaches those limits, the alarm-state is triggered and stored as a “context” in Node-RED using “`flow.set()`”. Other nodes like the “Controller” node can read this context using “`flow.get()`”.

For the rotational switch, the case statement is a little bit more involved. First, the state of the switch is evaluated. If it’s in the CCW position, the alarm will be reset. Second, we use the poll rate of the iodata-in node associated with the switch as the update rate for the state of the white LED that is part of the switch. The LED can have three states, on, off and blinking. In case the temperature controller is running, and no alarm is triggered, the LED is on. If the alarm is active, the LED is blinking. If the controller is not running, the LED is off.

To make the LED blink without using any loops or wait-statements, we store the LEDs last state in the nodes context with “`context.set()`”, similar to the flows context used before. This allows us to simply invert the last known state without reading the LEDs state with an “`iodata-in`” node. The result is a LED that changes state each 500ms (the poll rate of the “`iodata-in`” node associated with the rotational switch).

The “Controller” nodes code is structured similarly, as can be seen in Listing 3. First “`msg.payload.name`” is evaluated using a switch-statement. “`msg.payload.value`” is stored as a node-context for the user-definable deadband, the actual temperature and temperature set point. The controller’s activation state (running / not running) is stored as a flow-context to make it available to the “Alarms” node.

The controller itself has three operating points, heating, cooling and idle. These are triggered by the difference between the actual temperature and the set point. If the difference lies within the deadband, the controller is idle. The width of the deadband defaults to 2°C. It can be configured by sending a message with “deadband” in “`msg.payload.name`” and the desired value in “`msg.payload.value`” to the controller node.

The controller node has three outputs, one for each operating point. To control these three outputs, we return a message-array. The array contains one JSON-object for each output. In case of the operating point heating the local variable “heating” equals true, “idle” and “cooling” are false. In this case the first output emits a message with “`msg.payload.value`” = 1, the second and third output emit a message with “`msg.payload.value`” = 0. The “`iodata-out`” nodes write these values to the global variables mapped to the respective LEDs, which are switched on or off accordingly.

```
let mode = "idle"
switch(msg.payload.name){
    case "deadband":
        context.set("tempControllerDeadband", msg.payload.value)
        break;
    case "I_ITEMPERATURE":
        context.set("tempActualValue", msg.payload.value);
        break;
    case "I_IPOTIVALUE":
        context.set("tempSetPoint", msg.payload.value.toFixed(1))
        break;
    case "I_XROTATIONCW":
        flow.set("tempControllerRun", msg.payload.value == 1)
        break;
}

if(flow.get("tempControllerRun") && !flow.get("tempAlarm")) {
    let deadband  = context.get("tempControllerDeadband") || 2
    let actualValue = context.get("tempActualValue")
    let setPoint    = context.get("tempSetPoint")

    if (setPoint - actualValue > deadband / 2) {
        mode = "heating"

    } else if (actualValue - setPoint > deadband / 2) {
        mode = "cooling"

    } else {
        mode = "idle"
    }
} else {
    mode = "stop"
}

msg = [{"payload": {"value": mode == "heating"}},
       {"payload": {"value": mode == "idle"}},
       {"payload": {"value": mode == "cooling"}}]

return msg;
```

Listing 3: The "Controller" function node controls the LEDs as a function of the actual temperature and set point

5 Publish the Controller's State to a MQTT Broker

The second part of this demo applications implementation is the connection to a MQTT broker. This shall demonstrate the usage of an IoT protocol like MQTT in Node-RED, using the built-in MQTT publisher node. Figure 13 shows the original temperature controller flow with three "out" nodes added to it. These connect to the "in" node of a new flow named IoT.

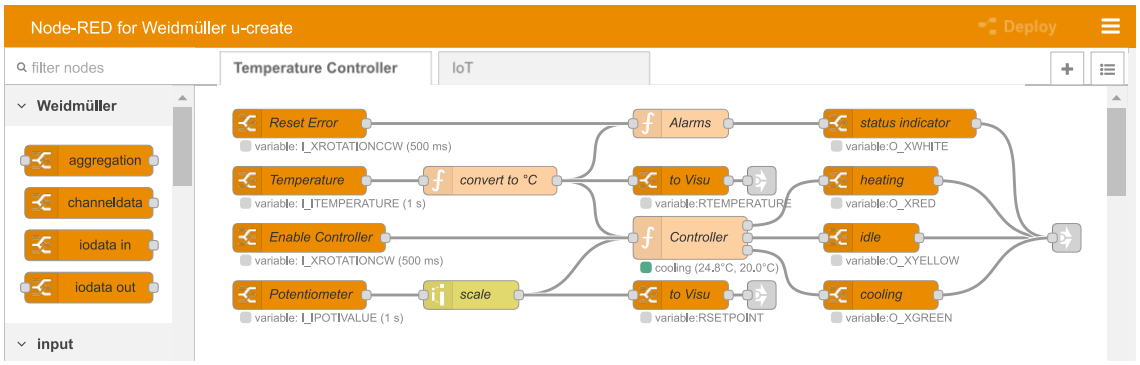


Figure 13: The temperature controller flow with three "out" nodes connecting it to the IoT flow

The IoT flow is a simple data pipeline consisting of four key nodes, as shown in Figure 14.

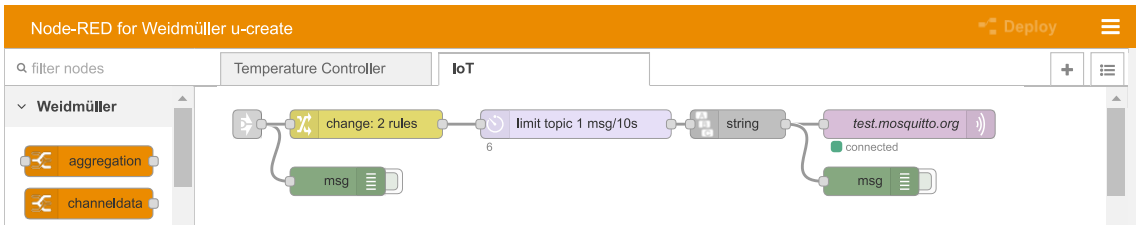


Figure 14: The IoT flow with a MQTT publisher node

To re-arrange the internal structure of each message to a format that suits the MQTT publisher node we use the change node. Figure 15 shows the configuration.

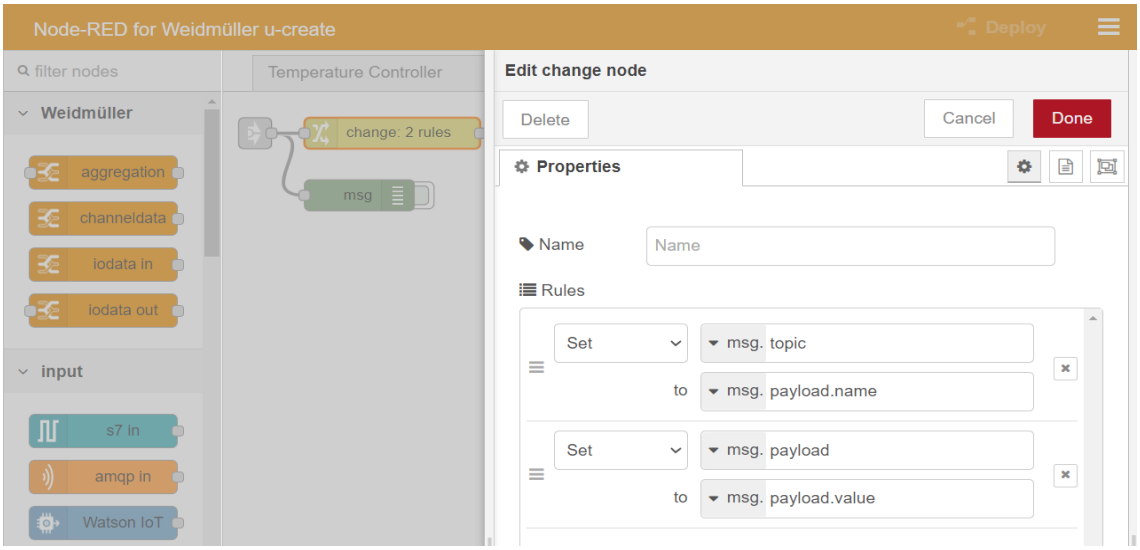


Figure 15: The change node is used to re-arrange the message structure

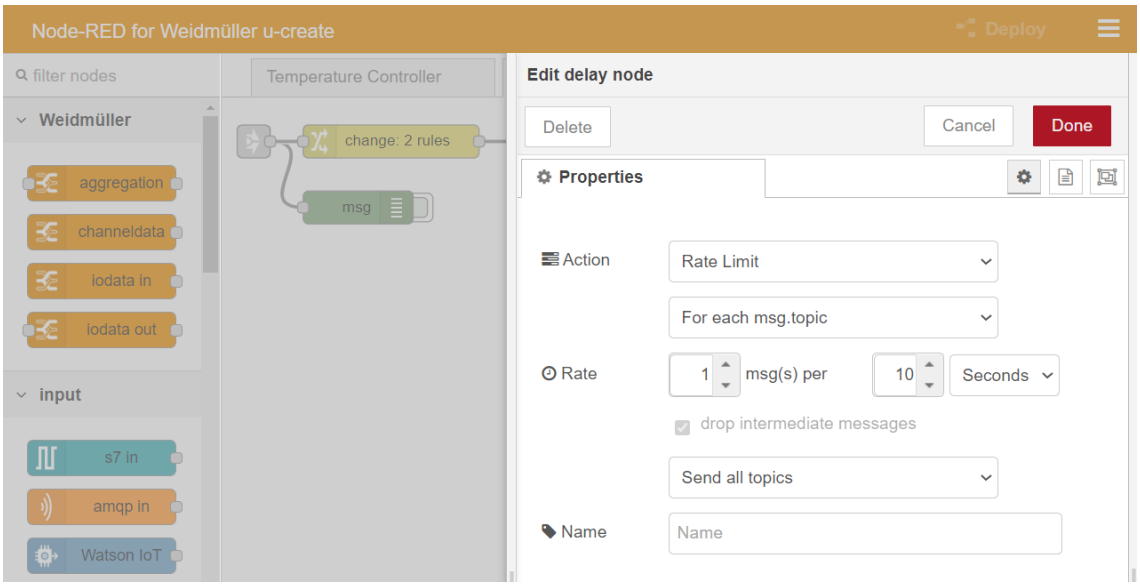


Figure 16: The delay node is used to limit the rate at which messages are send to the MQTT broker

Since we are using a public, free-to-use IoT broker, we want to make sure we practice fair use. This means that we must limit the number of messages we send to the broker to a sensible amount, in this case only one message every ten seconds for each variable. This can be done by using the delay node in the mode “Rate Limit”, as shown in Figure 16.

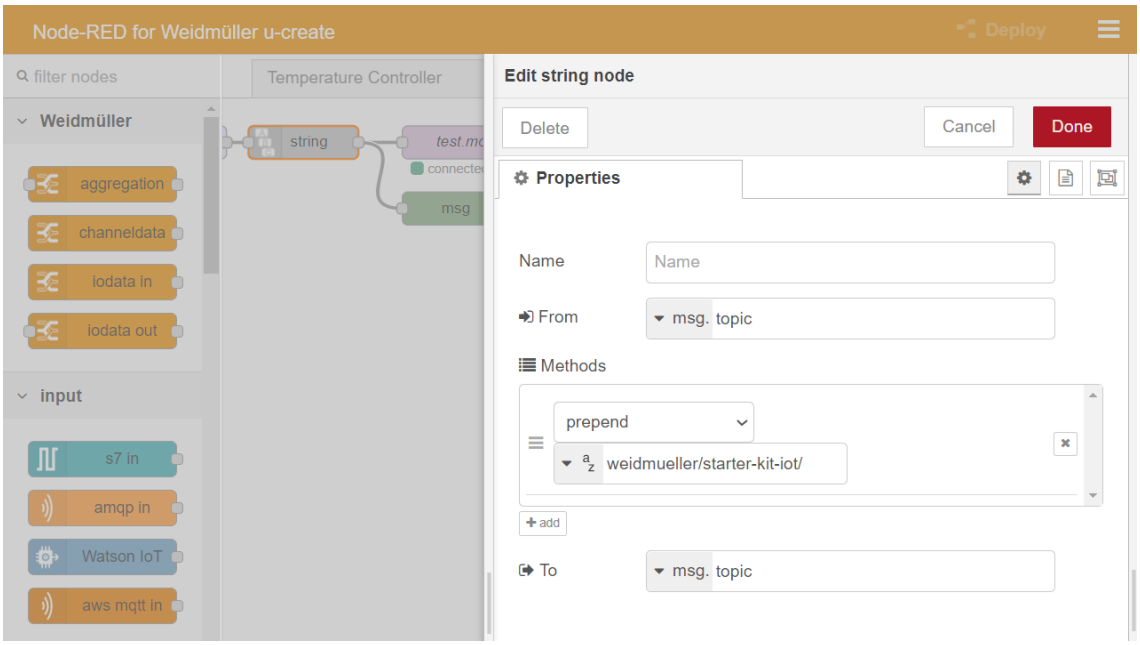


Figure 17: The string node is used to prepend a topic hierarchy to the variable name

The MQTT protocol works with topics that can be ordered in a hierarchy. While we could publish each variable under the topic “`variableName`”, e.g. “`RTEMPERATURE`” for the actual temperature, it is recommended to publish them as subtopics of a parent topic. We use “weidmueller/starter-kit-iot/” as the parent topic. To do this in Node-RED, the third party “string” node can be used. Figure 17 shows its configuration.

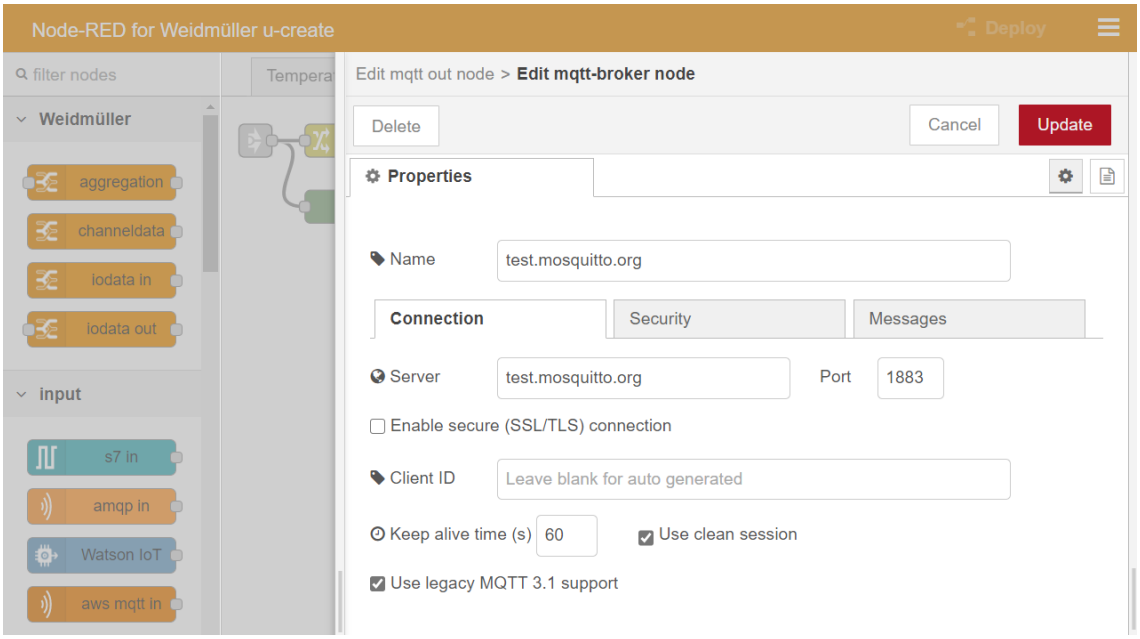


Figure 18: The mqtt-broker settings of the mqtt out node, here the mosquitto.org public test broker is used

The MQTT out node needs minimal configuration. A MQTT broker must be configured, in this case we use the public test broker of mosquitto.org. This broker can be used free of charge and allows unencrypted connections which is ideal for this simple example application. The configuration of the MQTT broker is shown in Figure 18.

To validate that our MQTT publisher is actually sending data to the MQTT broker, we need to subscribe to one or more of the topics with a MQTT subscriber. We use the free and open source Eclipse Mosquitto™ subscriber “mosquitto_sub”.

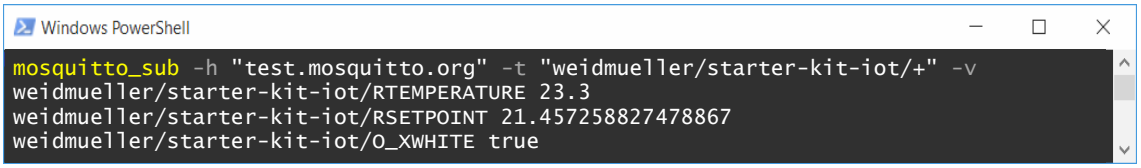


Figure 19: The Mosquitto MQTT subscriber command line utility (command can be copied)

Figure 19 shows “mosquitto_sub” subscribed to the wildcard topic “weidmueller/starter-kit-iot/+“. The “+” means that we are subscribed to every subtopic on this hierarchy level, as can be seen by the different topics printed out on the console.

6 Further Information

More information, manuals, example projects, Quick Start Guides, Application Notes you can find on our website. The following resources provide a good starting point.

QSG0033: Quick Start Guide for Starter-Kit IoT + Node-RED

Support and Downloads for Starter Kits, including Example Applications and Flows
https://www.weidmueller.com/int/service/support_for_u_control_starter_kits.jsp

UC20-WL2000-IOT General Information
https://catalog.weidmueller.com/procat/Product.jsp?productId=(%5b1334990000%5d)

MQTT General Information
https://mqtt.org/

Eclipse Mosquitto™ MQTT tools
https://mosquitto.org/