i-UG Open Source Education
for IBM i

Node-Red

LESSON 3: Lights-3

1. Creating a traffic light system

So, we can switch a light on and off and control the colours… from static instructions. Now for this next task, we will be stepping it up a few levels. We are going to create a traffic light indicator mechanism for a work process, based on data that will be randomly generated. Don't worry, it's simpler than it sounds.

The function of this is to use the lights as an indicator as to a given value. We will use a random number generator node to give us a number (representing a value given from any number of places), which will fall into 1 of 3 categories depending on what the number is, and each category will light up a different colour, green, yellow or red.

The usefulness of this function is limitless when applied to a work process, such as it being used to indicate too much water being put into a system, or a temperature measurement etc.

We will be using 5 nodes:

Gate Node:
This node can halt the payload of a flow depending on Boolean logic, acting as a gateway for information.

Function Node:
This node allows us to input code into our flow, making it more useful and specific.

Random Node:
This node will generate a random number between two given values when prompted by an inject node.

Delay Node:
The delay node will delay the transmission of a payload between nodes.

Debug Node:
This will allow us to see the payload from a node.

STAGE 1

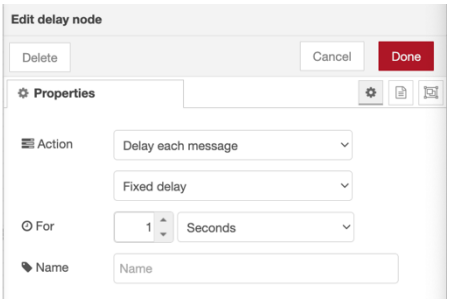
First, we will bring in an inject node, no changes need to be made to it as we are only using it to kick-start our flow.

Next, we will bring in 3 function nodes, connecting them to the inject node. Each one will be programmed for 1 of our 3 colours, so remember the colour information from the previous exercise. For each of the function nodes, you'll need to write the following code in the 'On Message' tab:

```
1. msg.payload = { "on": true, "bri": 100, "hue": n }   (Where n is the colour we want)
2. return msg;
```

This is the same for each node apart from the "hue", that will need to be changed for each colour. This will act the same as the specific colour inject node from before.

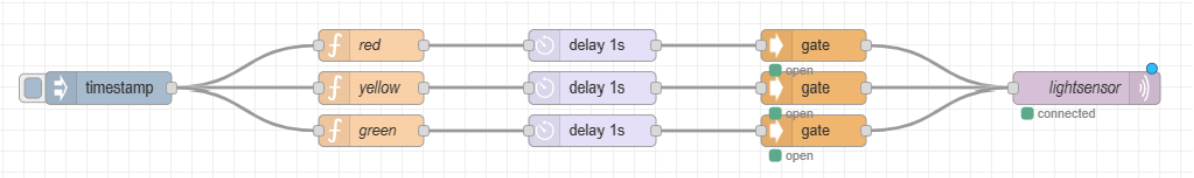
Next, bring a delay node that connects from each function node. Open the nodes and change the settings to match the image below.



The purpose of these nodes is to ensure that the order of payloads works so that the correct gate will open before the colour information is sent, otherwise all 3 colours may be sent at once, confusing the light.

Next are the gate nodes. Again, add a connecting gate node from each delay node. There should be a linear path from the inject node through to the delay and then gate nodes now. If you open the gate node you should see a list of commands that it can be given to change the status of the nodes. The control topic is the message that allows each command when delivered. For now, we can leave the gates as they are.

Finally, for this first stage, we need to add in the Philips hue node, connect all 3 gates to the node, ensure that the node is connected to the light (review the first task) and you should have something resembling this.



For the moment this flow wont function, as even when deployed, all gates will be opened, therefore the colour information from each colour will be sent through at once, which will obviously not work. We need to provide it with information which will cause the gates to open one at a time depending on the value. That's the next stage.

STAGE 2

The first step of stage two is introducing our random number generator node. Connect this to the inject node, open it and insert our number range in the 'from' and 'to' boxes. Generally, you can set any range you like, depending on the task at the time. For our purposes, the range is 1-359, representing each hue code.

We will connect our Debug Node to this Random Node, so we can see what value has been generated. This will be crucial later to ensure the system is working as intended.

Remember, to view our debug log you may need to open the sidebar if it's not already open, to do so click on the arrow that emerges when hovering your mouse of the centre dash next to the scroll bar on the right log the screen (*fig 1*), then click the debug icon at the up of the sidebar (*fig 2*). When the flow is deployed and the inject button is clicked you should see a message like this (*fig 3*) with the random value.

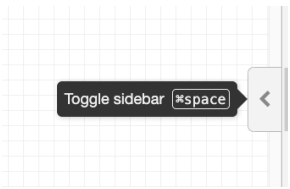


Fig.1

Fig.2

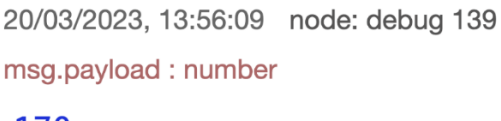
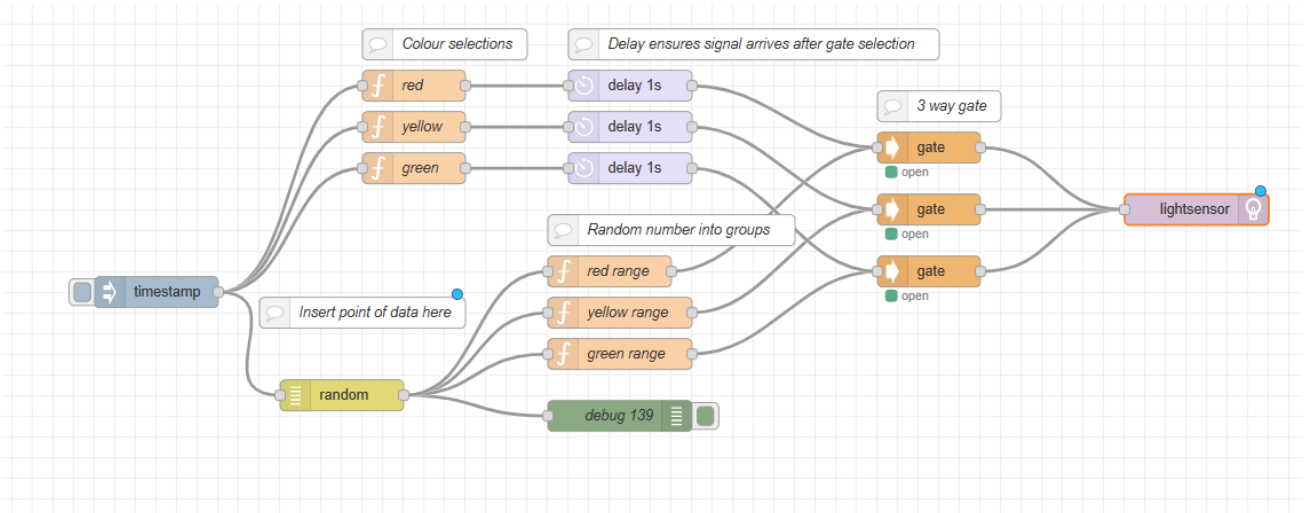


Fig.3

It now gets a bit more complex, but we'll break it down for you.

Add another 3 function nodes connecting from the random number node, in a moment, we are going to place a few lines of code into each of them that will place the random value we're given into one of 3 categories. Connect each Function node to a corresponding Gate node.

Our flow should now look something like this: -



Now, some code…

```
let payload = msg.payload;
let field = ""
if ((payload > 300 && payload < 359)) {
    field = ("open")
}

else {
    field = ("close")
}
msg.payload = field
msg.topic = "control"
return msg;
```

The code here can be copied into the 'On Message' tab of all 3 new Function Nodes.

N.B. It is vital that all of the punctuation is EXACT in this JavaScript as any small error will result in the code (and the Node) not functioning!

So, essentially, we will set each Function Node to operate in the same way, but although they will all look the same, the crucial elements that will change are the numbers in green, these represent the range for each category and will be different for each of these new Function Nodes.

Split whatever range you have chosen into 3 and for each node give one of these 3 ranges.

Look carefully now at the code we have given you. There is a amore detailed explanation of the code at the end of this section.

We are setting up a field called 'payload' and populating it with the contents of the Message Payload

let payload = msg.payload;

We are also setting up a working variable called 'field' and setting it to blank

let field = " "

Then we are checking what the payload coming in from the Random Node is. If it is higher than the low value for this Function Node and lower than the high value for this Function Node, then we will put "open" into the variable called 'field'.

If ((payload > 300 && payload < 359)) { field = ("open") }

The 'else' statement is just that, if it has failed to fall within the range for this field, then we do the 'else' statement, which is to set the 'field' value to "close"

else { field = ("close") }

We end by now putting the working variable 'field' into the msg.payload, so that this can be passed to the next node. We will also tell the next node (the Gate Node) that we want to control it by setting the msg.topic to "control".

msg.payload = field

msg.topic = "control"

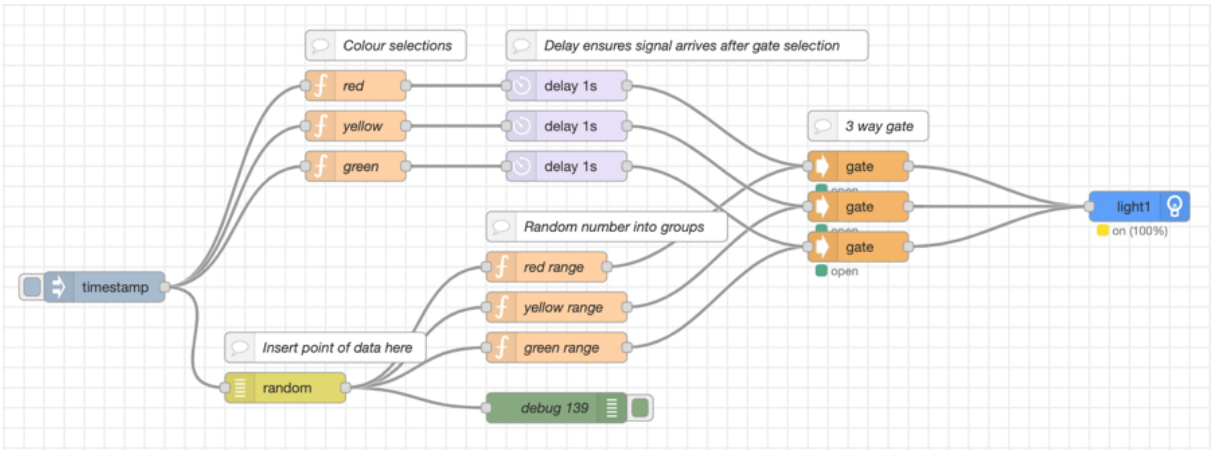
If the value is within the range on a given node, it will open the gate with the corresponding colour, if it's not within the range, it will close the gate, stopping that colour from being presented.

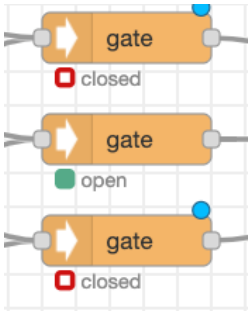
At this stage, although we are using the range 1 – 359, these are only used to create groups. We will use the actual colours we set in the earlier 3 Function Nodes. For now, set one Function Node to be within the range 0 – 199, set the next node to be between 200 and 300 and the last Function Node to be between 300 and 359.

All will become clear…

Now, each time you hit the Inject Node button, the Random Node will present up a new random number.

If you've done everything correctly then you should have a flow like this. And you should be able to tell which gate is open by the indicators below each gate, check that the gate that's opening corresponds with the value and its range in the debug log. And, of course, have a look at the lights!





N.B. Don't forget the 

button!

Java Script Code Breakdown

OK, now we are going to explore the power of using small snips of JavaScript to really extend what we can do with Node-RED in order to meet our objectives.

Let's look at what we are trying to achieve here…

```
let payload = msg.payload;
let field = ""
if ((payload > 300 && payload < 359)) {
    field = ("open")
}

else {
    field = ("close")
}
msg.payload = field
msg.topic = "control"
return msg;
```

Taking each line in turn…

1. let payload = msg.payload;

   The 'let' keyword allows us to define a variable that we can use in our node. It allows us to define the variable and set the value at the start of the node. We will use it in its simplest form here but later you will learn how to make these variables persistent across Nodes, across Flows and Globally, but for now, we will just keep this local.

   In this case we are setting the value of our newly created variable 'payload' to the value that has been passed to this node in the msg.payload value. In this example, the value has been generated by the Random Node and is passed here via the link.

2. let field = ""

   Here, we are creating a new working variable called 'field' and we are setting its starting value to blank (" ")

3. if ((payload > 300 && payload < 359)) { field = ("open") } …

   Now, some logic. We want to know if the value passed in the msg.payload from the Random Node is within a certain range. So, in English, if the value is between 300 and 359, then we want to 'do' something. The something that we want to do is to start the process of setting the switch in the next node to 'open' as the value falls within the range. So, we will set the variable 'field' to "open".

This statement crosses line 3, 4 and 5 for this test, but in reality, it is also linked to the next statement, which allows is to do something 'else' if the value is not in this range, as we will see later.

6. This line is blank. This often aids the readability of the code but in Node-RED (As well as many other tools and languages) gets ignored by the operating system.

7. **else { field = ("close") }**

   Now, if the value in the previous 'if' statement is false, i.e. the value in msg.payload is not in between 300 and 359, we want to do something 'else'. The something that we want to do is to make sure the value we will ultimately pass to the following Gate Node is set to be closed. We will set the variable 'field' to "close". Again, this statement extends over lines 7, 8 and 9.

N.B If you are familiar with programming, you will understand how important the syntax in each statement is. If not, please TAKE CAREFUL NOTE of the use and position of each of the brackets () {} [] and all punctuation "" '' ; etc. as these are CRUCIAL to the code working.

Tip: There are plenty of examples available in the GitHub and generally through Google.

10. **msg.payload = field.**

    In the lines above, we have decided whether the value coming from the Random Node is within the range 300 to 359 or not, and we have set the variable 'field' to either "open" or "close". This statement now sets the msg.payload that we will pass to the next node. We are setting thisto the value of the 'field' variable we have been working with.

11. **msg.topic = "control"**

    As well as passing the msg.payload on to the next node, we can pass many other values, some of which are controlled values, such as 'topic'.

    We have elected to set this field to "control" because the next node – the Gate Node – is looking for an instruction on what is to be done. It wants to know if it is one of its standard control functions. Once it knows this, it will read the value passed in the msg.payload field and execute the control function (See Gate Node later on in the education).

13. **return, msg;**

    We always make sure at the end of the program that we are returning the Message (msg) which will contain all the elements we have used, like payload and topic.

And with that, the lesson is concluded. If you didn't manage to make one of the flows work then you can copy one of the flows from our GitHub repository here, and see where you went wrong. These functions, although they seem simple, are the backbone of many useful applications of Node-Red for major and small businesses.

https://github.com/JoshRyanEOG/i-UG_Education_Node-Red