



# Tutorial for I/O Learning Card and Node-RED

*Your Gateway to Home and Industrial Automation*

*(Preliminary)*

**Note:** *This document contains the first three chapters of the Sequent Microsystems I/O Learning Card Tutorial. It has been designed so you can start learning about Node-RED using only your Raspberry Pi.*

*Once you receive your I/O Learning Card the full Tutorial will teach you how to Implement common tasks required for Home and Industrial Automation Systems*

# Learning Kit Workbook (version 1.3)

## Chapter 1 - Introduction

This tutorial will introduce you to the world of home and industrial automation. You will learn about widely used automation interfaces and about how to measure and control the world around you.

You live in a marvelous technical age. A very capable computer in the form of the Raspberry Pi is available at very low cost. Node-RED, the software you will use, is available at no cost! In this tutorial you will learn the basics of automation by using the Sequent Microsystems Learning kit, which contains a Raspberry Pi card and example sensors and actuators. You will build systems that demonstrate the basics of Node-RED and automation interfaces.

The tutorial is not a cookbook! You will be given some step-by-step instructions for building the example systems, but you will need to do your part. This means working carefully, being patient and stopping to think about what you are doing. To help you along you will find engineering puzzles<sup>1</sup> for you to solve. Some are easy, some are hard, but the purpose is to help you extend your knowledge.

### Before We Begin...

In this document you will find links (in blue) that will take you to websites with more information about topics you will be learning about. These links are there to help you learn. However, when we give you a link it does not mean that Sequent Microsystem necessarily endorses or recommends any products you may find advertised there.

You may find this workbook to be too easy or too hard. Well, it is hard to please everybody. We have put this workbook together to help you learn about home automation I/O, the Internet of Things, and Node-RED. Our audience is the non-professional person who has some technical background and wants to learn how to construct simple automation systems, for example, the hobbyist or the smart high schooler.

If you have absolutely no experience with programming and electronics you will find the going a bit difficult. In Appendix [\[redacted\]](#) you will find some resources to help you learn basic electronics and programming. Even if you have never programmed before you will find that with some patience you can build very interesting systems using Node-RED alone. If you know how to program and have built some circuits then you will find some of the idea here obvious, but you can always skip over anything you already know.

If you are really a hard-core programmer who just made a bundle selling your latest smartphone App, just jump to Appendix [\[redacted\]](#) and start building whatever you want in Python, Command Line scripting or whatever other language is your favorite. You will still find the I/O Learning Board useful because it gives you a full selection of the standard automation interfaces to prototype systems with.

### Purpose

We will show you how to use your I/O Learning kit to carry out some basic automation tasks. To do this you will be using programming language called [Node-RED](#).

---

<sup>1</sup> Puzzles? – well if they were called as “Exercises” would you do them or would you figure it was just more drudgery? Sorry, at this time there are no “answers in the back of the book”. If you find a neat solution to any of the puzzles, send it to us and we will add it to the collection.

## Learning Kit Workbook (version 1.3)

Your I/O Learning card kit will allow you to build and program a small system that can control devices like lights, motors, valves and so on. You will learn to control these devices by sensing the environment, making decisions and then sending a signal to the device, or devices, you wish to control.

### What You Will Learn

Our objective is to help you learn about two main ideas. First, you will learn about the different types of I/O signals that you can sense and control with the I/O Learning Card. Second, you will learn how to sense, process and control these signals using Node-RED, a programming language that you may find different from other languages you have used. Don't worry... if you are not interested in using Node-RED you can also use Python or even a Linux command line script in place of Node-RED. The appendices will give you all the information you need electrically and program-wise.

**The Learning I/O Card:** This card is like a box of chocolates; it has several different I/O interfaces you can sample. You will be able to sense contact closures, voltage levels and current levels, and you will be able to control contact closures, LEDs, voltage outputs, current outputs and a small motor. Each of these input and output types is compatible with widely used automation standards. Your I/O Learning Card also includes a standard RS-485 interface, which is a standard in Industrial Automation allowing you to communicate with devices over long distances and to link several systems with I/O Learning cards together.

**Node-RED:** This is a language developed specifically for programming Internet-of-Things (IoT) devices. It is likely that you will find this different from other languages you may have used. In the first place, it is primarily a graphic- based approach to programming, in which you program by connecting different types of nodes together with "wires" to build what are referred to in Node-RED as "flows". Second, the programming model is based on "events" and "messages". This is very different from languages like Python, C, C++ or Java, which you may have learned about already. You will find out that Node-RED is very powerful when it comes to constructing automation and control system.

When you have completed the exercises in this workbook you will know how to:

- Use seven different types of standard automation interfaces
- Write basic control programs in Node-RED
- Debug your project
- Build control systems using each of the available I/O connections
- Extend our demonstration projects to larger practical systems.

### Safety First!

Before you use your I/O Learning Card you should read and understand the following general safety guidelines, which are really just common sense. If you are an adult, we assume that you know what you are doing, but if you are not an adult remember this: nothing you can learn here is worth getting hurt for. So, keep the following in mind:

- Some of the components on this board can control high voltages (greater than 24 volts). Do not build setups that use high voltages unless you are certain that you know what you are doing.
- Automation means that your program can get along just fine without you watching it. Just make sure that whatever you decide to control works in a safe manner. Common sense should tell you not to turn on a space heater unless you know it is not going to start a fire. Remember,

## Learning Kit Workbook (version 1.3)

Grandma may have just fallen asleep while reading the newspaper, which fell out of her hands and onto the space heater. Your perfect Node-RED program is going to set the house on fire when it turns the space heater on from your cellphone. Be careful with what you control and remember the [Peter Parker Principle](#): “With great power comes great responsibility”.

- For crying out loud... always use your tools: cutters, knives, soldering iron etc., etc., etc. safely. You know you should always wear safety glasses when you cut wires and solder. Just do it!

### Setting Up Your System

This tutorial is based on using your I/O Learning card with a Raspberry Pi processor board. In this section you will attach the I/O learning card to your Raspberry Pi, load up a test program and make sure everything works.

#### Make Sure You Have Everything in the Kit

Let’s start by making sure you have everything in the I/O Learning Card kit. When you open it up you should find the following:

#### I/O Learning Card

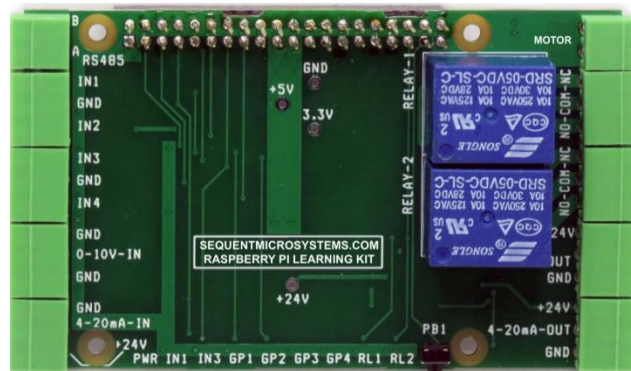


Figure 1-1: I/O Learning Card (Top View)

#### Connector Plugs

- Two 2-pin plugs for the micro-motor and RS-485 ports
- Four 3-pin plugs for the four inputs on the left side of the board
- Four 3-pin plugs for the four outputs on the right side of the board



Figure 1-2: Connector Plugs



## Learning Kit Workbook (version 1.3)

### Mounting Hardware



Figure 1-3: Mounting Hardware

What Else Will You Need?

#### *Things You Must Have*

**Raspberry Pi and Power Adapter:** The I/O learning card is compatible with Raspberry Pi 3 and 4. Of course, you will need a power adapter for your Raspberry Pi. We recommend using the Pi-4, which has more processing power.

**SD Card:** If it is not already included with your Raspberry Pi, you will need an SD card. We recommend that you purchase one that already includes the Linux based operating system of your choice. All examples in this tutorial are based on the Canakit Raspberry Pi 4 Starter kit running Raspbian Buster.

**Monitor:** You will need a monitor so that you can see what you are doing and, you will also need the right cable to connect the Raspberry Pi to the monitor. If you get desperate you can use the family's TV if it has an HDMI input. Just be sure that you are not keeping someone else from watching old reruns of Gun Smoke or the Secret Life of Hamsters.

**USB Keyboard:** Of course!

**USB Mouse:** Of course!

**Internet Connection:** Of course!

#### *Things That You Might Need*

- Flat screwdriver
- Philips screwdriver
- Insulated wire (20 gauge)
- Wire stripper/cutter

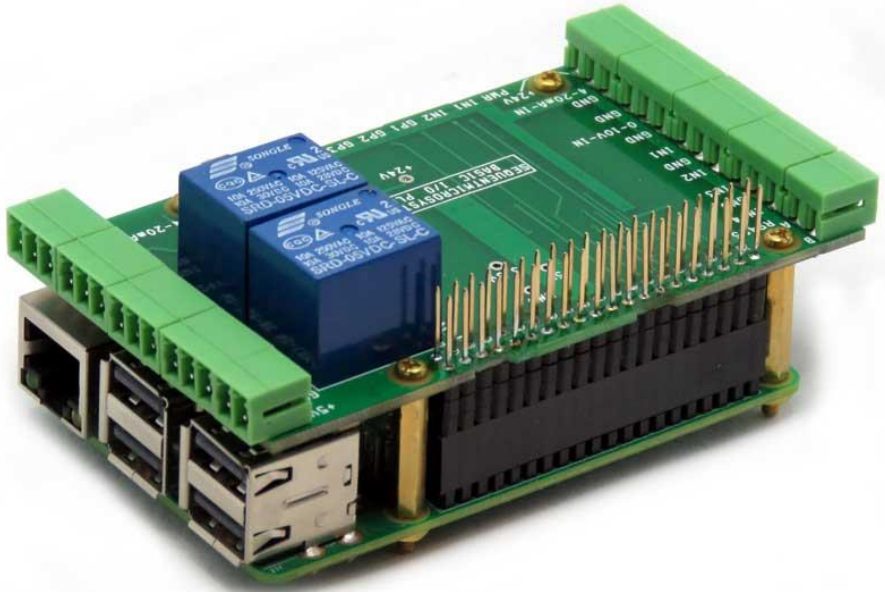
#### *Putting It All Together*

Now it is time to mount the I/O Learning Card on your Raspberry Pi. But before you do that, put together your Raspberry Pi, download the operating system and test it out. You can do this by playing some of the provided games. Make sure that everything is working without the I/O Learning Card. Your Raspberry Pi will come with instructions telling you how to set up your system and download the operating system. Be sure to test your Internet access.

Now that you have tested the basic Raspberry it is time to mount the I/O Learning Card. Be a smart engineer and unplug everything from the Raspberry Pi. If you mount the card with the power turned on you may see some pretty sparks followed by a cute little puff of smoke. If you see this take a moment to

## Learning Kit Workbook (version 1.3)

admire its fleeting beauty because you will probably need to buy another processor board or I/O Learning Card or both!



*Figure 1-4: Assembled Raspberry Pi with I/O Learning Card*

Once you have everything put together, but before you turn on the power do one final check. Make sure that nothing on the Raspberry Pi Card is touching the bottom of the I/O Learning Card. Be especially careful about the area where the I/O Learning Card over hangs the USB and Ethernet connections. You might consider placing some plastic tape on top of the plugs just to be sure. Okay, once you have done that, connect the monitor, keyboard, mouse and the power supply. Turn it on. If your system comes up, you are in business.

### Download the Software

Once you are happy that your Linux system on the Raspberry Pi is operating correctly it will be time to download the software. This software will have several components:

- Drivers for Node-RED, Python and the Linux Command Line
- Updates to the firmware on the I/O Learning Card
- Test Program to verify that your I/O Learning Card works properly

[ TBD - Downloading the Code ]

### Test Your System

It is always good engineering practice to test parts of your system before you jump in and build something. So, start by testing the I/O Learning Card right after you install it. Testing is simple and only

## Learning Kit Workbook (version 1.3)

takes a few minutes. Anytime you think there might be a problem with the card just go back and test it. Follow the instructions in Appendix [\[redacted\]](#).

Testing is based on the loopback cable (below). You can buy it with the Option Kit or make your own.

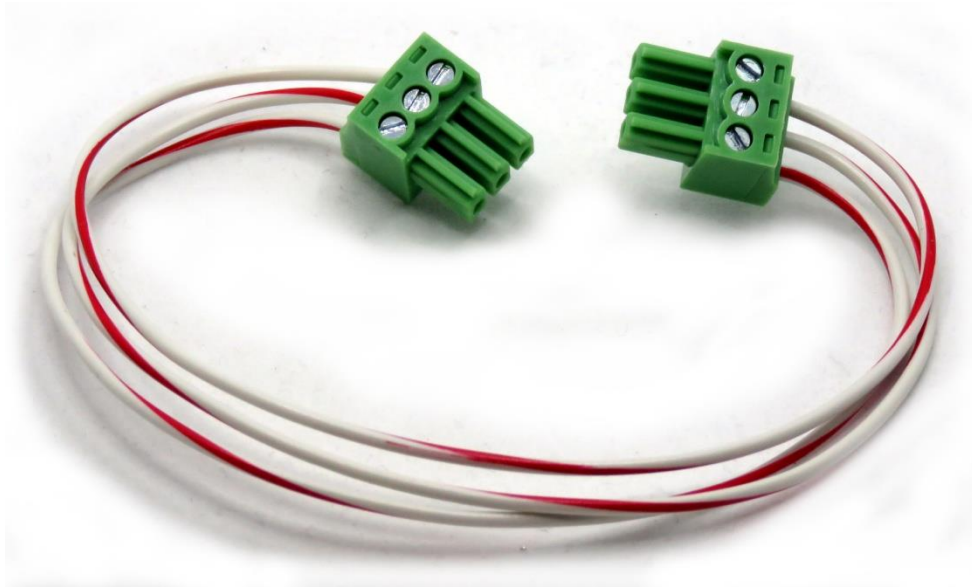


Figure 1-5: Loopback Cable

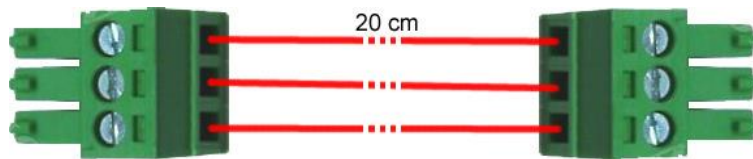


Figure 1-6: Measurements for Loopback Cable

Uh-Oh... It Doesn't Work! What Do I Do Now?

We test every I/O Learning Card before it leaves the factory, so your board should work. We do this using the same test program that you are using. If there is a problem with the card, you should be able to locate it using the test program. If you built your own loopback cable, be sure to check it with an ohmmeter.

### The I/O Learning Card

Let's talk about the I/O Learning Card, what it is, and what you can do with it. Below is an over head view of the card. The connectors on the left and right are where you will connect your sensors and

## Learning Kit Workbook (version 1.3)

actuators. Along the bottom edge you will find LEDs, some of which are programmable (GP1-GP4) and a pushbutton (PB1).

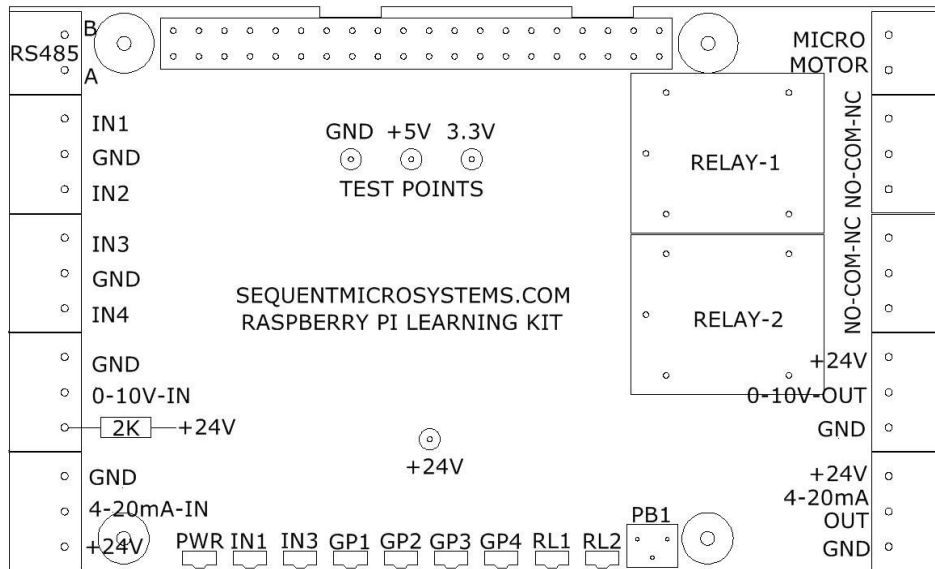


Figure 1-7: I/O Learning Card Connectors

The figure below is a block diagram of the I/O Learning card. You will find more details in the Appendix.

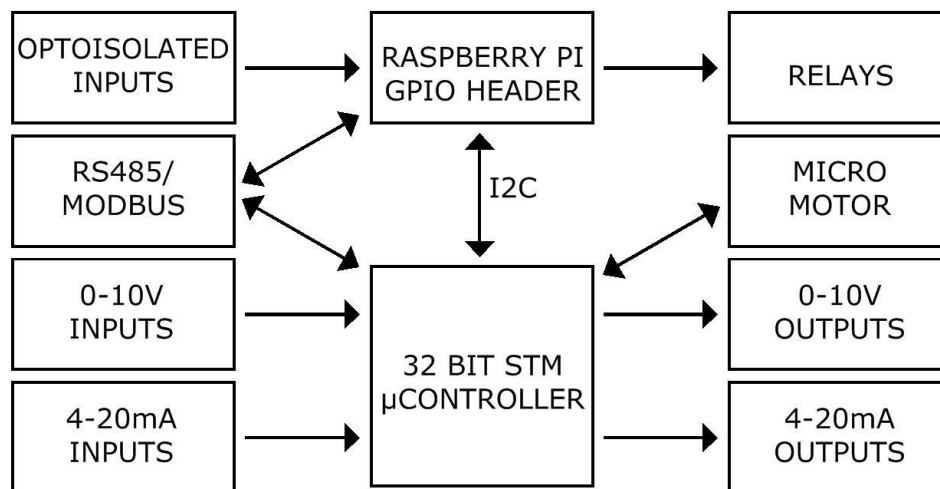


Figure 1-8: I/O Learning Card Block Diagram

### Node-RED – What’s it All About?

#### FBP – Flow Based Programming

If you already program then you are familiar with the dominant programming model, namely “imperative” programming. This includes many languages such as Python, C, and even the educational language Scratch. In the imperative programming model, you write programs as a list of instructions and

## Learning Kit Workbook (version 1.3)

the program is executed one instruction at a time. All your data is stored in a big memory and the program reaches into the memory and manipulates the data there. Imperative programming is very powerful and well suited for many applications, which is why it is the dominant way we do programming.

There are problems with the current programming approach when you are trying to build systems with many asynchronous events. An asynchronous event is any event, like a button push, temperature change and so on that can occur at any time. To handle this type of environment, which is typical of the so-called Internet of Things (IOT), you need to be a very clever programmer. One reason is that your program must continually check for these events. When you try to program in the imperative style it is easy to get lost trying to handle events efficiently and in a timely manner. It is basically technical juggling.

This is where a flow-based programming language, like Node-RED, comes to the rescue. In imperative languages you model processing as the sequential execution of a single stream of instructions. Instead, in Node-RED you set up a network of routines that are triggered by the appearance of data at their inputs and which in response produce output data. When tied together the collection of these little routines can handle many asynchronous events in a powerful and understandable way.

Before we get loaded down with lots of definitions, let's first think about a simple situation that you handle with the imperative programming approach. Then we will look at the same situation handled in a flow-based programming system. We won't even use programming here we will just use our imagination and think about everything in terms of a little office with tiny little people doing all the work.

### A Simple System – getting water to Aquaville

Here's the situation... (check out the figure below). The town of Aquaville is full of tiny people all from the same extended family, called Water. There's Jack Water, Jill Water, Shawna Water, Juana Water etc. Fortunately, for everybody it is no longer necessary to fetch water from the river or in the case of Jack and Jill to drag a bucket up hill to the well and risk breaking their crowns. With great foresight and some expense, the town put in a central water tank and was able to tap into the river a few miles away. Now when Juana needs water she just goes down to the tank and fills her bucket.

## Learning Kit Workbook (version 1.3)

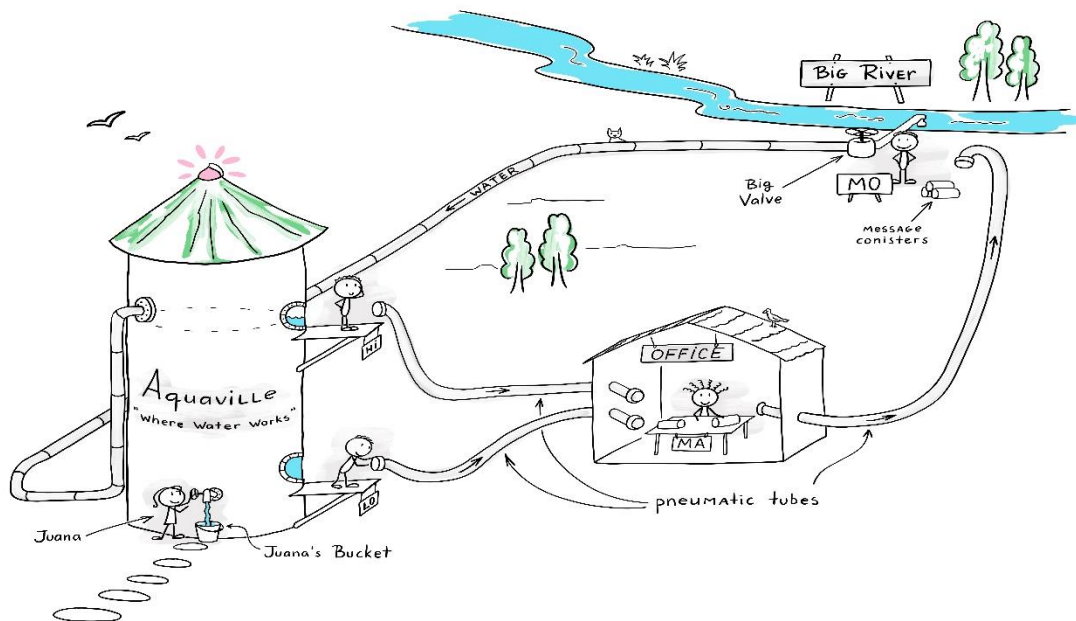


Figure 1-9: The Little Town of Aquaville

Of course, it would be a good thing if the tank always had water in it. This is no problem because Mo Water has control of a big valve that has two positions: open and shut. When the valve is open water from the river pours into the tank. In fact, it pours in so fast that the tank would overflow in a few minutes if the valve was left open. This would be a waste of water and Aquaville would flood. Unfortunately, Mo Water and his valve are located out by the river where he can't see the tank but, fortunately, there are two other folks, Hi Water and Lo Water, to help him out.

Hi Water is always watching a little window near the top of the tank, so she know when the water is too high. Her brother Lo Water is watching another window near the bottom of the tank to make sure the level is not too low. In charge of everything is Ma Water who was appointed by the Aquavillians to make sure there was always water, and that their feet did not get wet from an overflow. How is Ma going to keep the tank full but not too full?

### The Imperative Approach

The thirsty residents of Aquaville have asked you to get Ma Water organized. You are pretty good at writing programs, so you think to yourself... "Hmm, how about we have Ma run around and visit Lo, Hi and Mo in turn, checking up on things and telling them what to do." Ma is a little forgetful, so she has a little piece of paper to write things on. Maybe your instructions for Ma look like this and if Ma had taken programming in school, you might be able to write it out in Python or Scratch:

```
Write "close valve" on notepad
Visit Mo Water and show him the note pad
Erase the notepad
While the town is thirsty:
    Visit Lo Water
    If Lo Water says "I can't see any water in my window" then:
```

## Learning Kit Workbook (version 1.3)

*write "open valve" on the notepad.*

*Visit Hi Water*

*If Hi Water says "I see water in my window" then:*

*write "close valve on the notepad*

*Visit Mo Water*

*Show Mo the notepad and he will do what it says.*

*Erase the notepad*

Simple enough. Ma Water just has to run around all day checking with each of her children. It's exhausting, but the town has water. She's always running around even when Aguaville is snoozing or binge-watching old episodes of Hamster Wars. Of course, if Ma Water is not fast enough getting from one place to the other the town might still flood or go thirsty.

If Ma is also supervising the sewage department, the electrical department, the street sweeping department, the fire department and so on then the job is really complicated, and poor Ma Water spends all her time running from one place to the other. If you have to tell her how to handle all this, well the program is going to be very complex.

### The Flow Based Approach

Now let's look at the problem from a different perspective, the flow-based programming perspective. Here we are going to allow the events to drive the program. In this approach Ma Water just sits with her feet propped up on her desk reading the latest copy of Hydraulics World and doing the crossword puzzle. Instead of running around to visit her workers Hi, Lo and Mo she has a little pneumatic tube system connecting her to each one. In a [pneumatic tube](#) system you put slip of paper on a little round canister and with a puff of air you send the canister to the other end of the tube. Very efficient. You may have seen one in Home Depot or some other store. Long ago big cities, like London and Paris, even used a system like this to send mail across town. It is said that mail could be delivered within an hour.

When she first sits down Ma Water write "close valve" on a slip of paper and sends it through the pneumatic tube to Mo Water. Mo reads the note and shuts the valve.

Now, Ma leans back, puts her feet on the desk and falls asleep doing a crossword puzzle.

Boom! A message comes in and falls on her desk. The message is from Lo and it says: "I don't see any water". Ma quickly writes a new message saying: "open valve" and sends it to Mo Water. Mo gets the message and does what it says by opening the valve. Ma and Mo go back to sleep.

Ker-Thunk!. A new message falls on her desk. It's from Hi and says she sees water in her window. Ma writes "close valve" on a slip of paper and sends it to Mo. Thud. The message lands at his feet waking him up. After checking it out Mo does what it says and closes the valve. Mo goes back to sleep. Ma works on her crossword puzzle.

And so, it goes. All day long Mo and Ma just wait for messages. Poor Hi and Lo have to pay close attention to their tank windows, but maybe we can give them an alarm clock that wakes them up every five minutes. When this happens, they check their windows and depending on the conditions send a message to Ma. This way the whole family gets some sleep.



## Learning Kit Workbook (version 1.3)

You can represent this program graphically by just showing Hi, Lo, Ma and Mo and the pneumatic tubes connecting them. Of course, you will also have to write a few more instructions for each of them, like telling Hi and Lo how often to check their windows in the tank and telling Ma what to do with each incoming message. Here is what your system might look like:

You can see from this example that the process is driven by the events (water in the Hi's window, no water in Lo's window) and the messages (From Hi, to Ma: "I see water in my window", From Ma, to Mo: "close the valve", etc.). This is what flow-based programming is all about. When we start looking at an Internet of Things application you will see that this is just like the flow-based programming of Aquaville: everything is driven by messages and events.

### Node-RED Basics

Node-RED is a type of flow-based programming language. Better yet it is graphically based, so it is very pleasant to work with. A program in Node-RED looks a lot like figure above and is referred to as a "flow". The word "node" makes some sense because a node is a place in the flow where work gets done. In the homey Aquaville example, Mo, Ma, Hi and Lo are all "nodes". Strangely enough, the initials RED in "Node-RED" do not seem to stand for anything in particular (see [Why is it Called Node-RED](#)).

The basic components of Node-RED are nodes, wires, messages and flows. Nodes are simply places where information is originated, processed and consumed (Mo, Ma, Lo and Hi from Aquaville). Messages are packets of information that can be passed from one node to another. Think about letters or telegrams. This is done using "wires" which are just like the pneumatic tubes of Aquaville. The terminology "wire" for a connection between nodes makes sense because the purpose of wires it to define how messages pass between nodes, just like telegraph wires or telephone wires. Lay down some nodes, connect them up with wires and you have a "flow". Start it operating and an event will cause a message to be generated. The message will pass from the node where it originates to another node. At that node it will be processed or possibly cause some external action. It is almost like designing a circuit, except that the messages that can be passed on a wire in Node-RED can be very complex.

Here is what a Node-RED flow might look like for Aquaville.

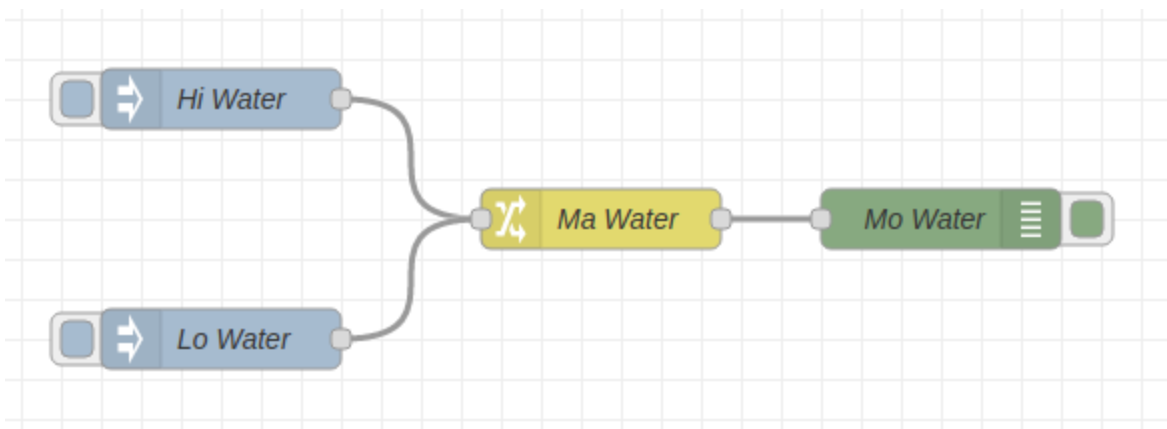


Figure 1-10: Node-RED Flow for Aquaville

## Learning Kit Workbook (version 1.3)

### Nodes

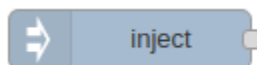
Let's talk about nodes.

Nodes are the places where things happen in Node-RED. There are many different types of nodes, but here we will be thinking about groups of nodes, namely, Input Nodes, Output Nodes and Processing Nodes. If you are a Node-RED expert you may have a different idea of how nodes might be grouped, but for the purposes of learning let's keep things simple. From these three groups of nodes, you can build very complex systems. Within each group of nodes there are many different sub-groups each pre-baked to carry out specific functions. Later you will find that you can even take off the covers of a node and tinker with the way it works. For some sub-groups of nodes, you can even write conventional programs to carry out complex processing.

Let's look at each kind of node in more detail. Nodes may have one or more "ports" on their left and right edges. Ports are places where you can connect wires to receive or send messages. Different kinds of nodes have different port configurations depending upon what purpose the node serves. As you will see the number of ports may change based on how you configure the node.

#### Input Nodes

Input Nodes receive events from the outside world and produce one or more messages related to these events. An event can be the push of a button, a change in temperature, a mouse click on the Node-RED screen or even a message from some other part of the Internet. When an event occurs the input node generates a message at one of its output ports and passes it over the attached wire to the next node. Here is a picture on a typical Node-RED input node, called an Inject node:

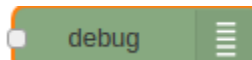


*Figure 1-11: Example of an input node*

Notice that there is a port only on the right side. This indicates that this particular node only produces messages in response to some external stimulus. There is a little more to this... you can configure some nodes to generate events periodically or even a single event at some future time.

#### Output Nodes

Output Nodes receive messages and generate events in the outside world. An output node may turn on a light, indicate a meter reading, produce a sound or even send an email message. The typical output node has an input port, but no output ports. When a message arrives at one of the input ports, the node examines the message and performs the requested action, i.e., turning on a light. Below is an example of a Node-RED output node.



*Figure 1-12: Example of output node*

#### Processing Nodes

Processing Nodes do the heavy lifting in Node-RED. Processing nodes usually have one or more input ports and one or more output ports. When a message arrives on one of its input ports the processing node wakes up and examines the message contents. Based on the contents, the processing node may

## Learning Kit Workbook (version 1.3)

process the message and produce a new message (or messages) to be passed to the one or more of the output ports. Processing nodes can do almost anything you can imagine. A simple processing node might convert temperature from Fahrenheit to Celsius. A more complex node might parse an email and find certain values in the email, for example a command to set the temperature of your furnace to a toasty 75 degrees (Fahrenheit not Celsius!). Here is an example of a simple processing node:



Figure 1-13: Example of processing node

### Messages

In Node-RED messages are the coin of the realm. If you think of messages as letters or telegrams, you will not be too far off the mark. Messages can contain almost anything, numbers, text, sound files, pictures, whatever. Messages may include other information, such as who sent them, what time they were sent and where they should go, just like a letter. Messages may also include keywords that identify various components of a message. You can use these keywords to reference different parts of the message. In this sense a message is almost like a filled-out form where each item is identified by a name. If you want to know more about messages, then check out Appendix [1-13](#).

### Flows

A flow is a collection of nodes connected by wires. In addition, a flow includes all the defined aspects of nodes. As you will find out later, that nodes can be configured to work in different ways. For example, one commonly used node is the Change node. This node examines incoming messages and modifies them according to various rules you define. It is almost as if there is a little clerk inside the node that copies the message over making changes that you told them to make (just like Ma Water in her Aquaville office). If the incoming message was an order the clerk might add up the total, compute the tax, work out the shipping costs and pass this modified form message on to both the shipping department and to the accounting department.

### Becoming Familiar with the Node-RED System

Here is what you will learn in the section:

- Loading up Node-RED
- Starting Node-RED
- Components of the Node-RED Independent Development Environment (IDE)
- A few simple Node-Red operations

Time to roll up your sleeves and start working with Node-RED. You will not need the I/O learning card yet, just your Raspberry Pi system. Before we build anything, let's just play with the system.

### Loading Up Node-RED

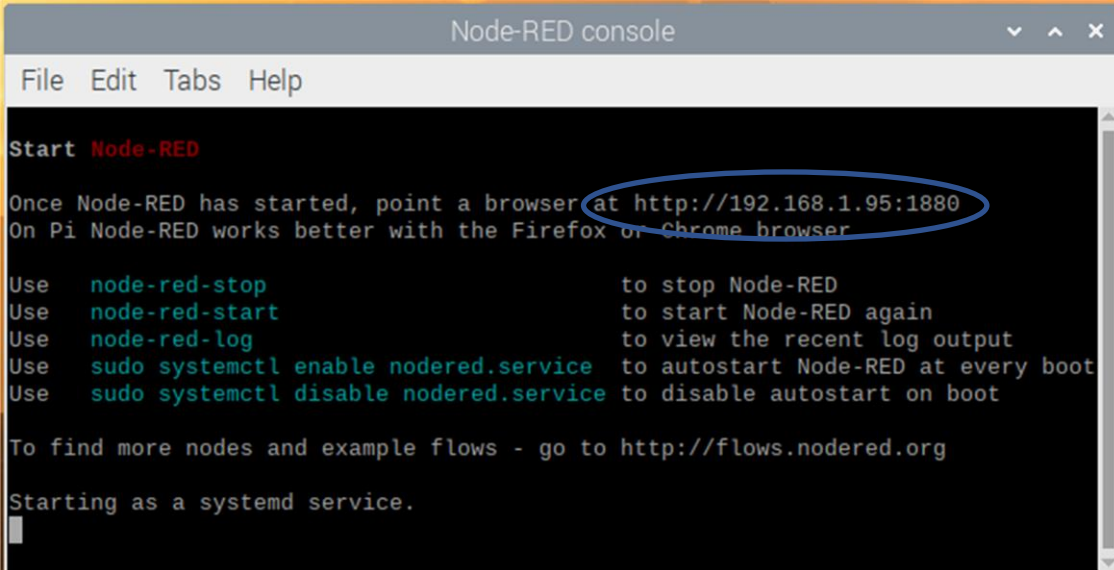
First, check your system. Because Node-RED is free it is very popular. As a result, you probably already have Node-RED on your system. If you do, then you should update it so that you are working with the very latest version.

If you do not have Node-RED on your system now then go to [Getting Started](#) at Node-RED and follow the instructions to download the latest version.

## Learning Kit Workbook (version 1.3)

### Starting Up Node-RED

Startup Node-RED. Do this just the way you would start any other program. When you start Node-RED you will see a window that looks like the one below.



```
Node-RED console
File Edit Tabs Help

Start Node-RED

Once Node-RED has started, point a browser at http://192.168.1.95:1880
On Pi Node-RED works better with the Firefox or Chrome browser

Use node-red-stop to stop Node-RED
Use node-red-start to start Node-RED again
Use node-red-log to view the recent log output
Use sudo systemctl enable nodered.service to autostart Node-RED at every boot
Use sudo systemctl disable nodered.service to disable autostart on boot

To find more nodes and example flows - go to http://flows.nodered.org

Starting as a systemd service.
```

Figure 1-14: Node-RED System Console

This is the Node-RED system console. However, what you really want is the Node-RED IDE, which is an Independent Development Environment where you can create flows graphically, and this is just plain fun! To get to the IDE hold down the control key and click the link circled in the figure above (the address is <http://192.168.1.95:1880>, which is on your local server). This will open your browser and you will see a window like figure below.

## Learning Kit Workbook (version 1.3)

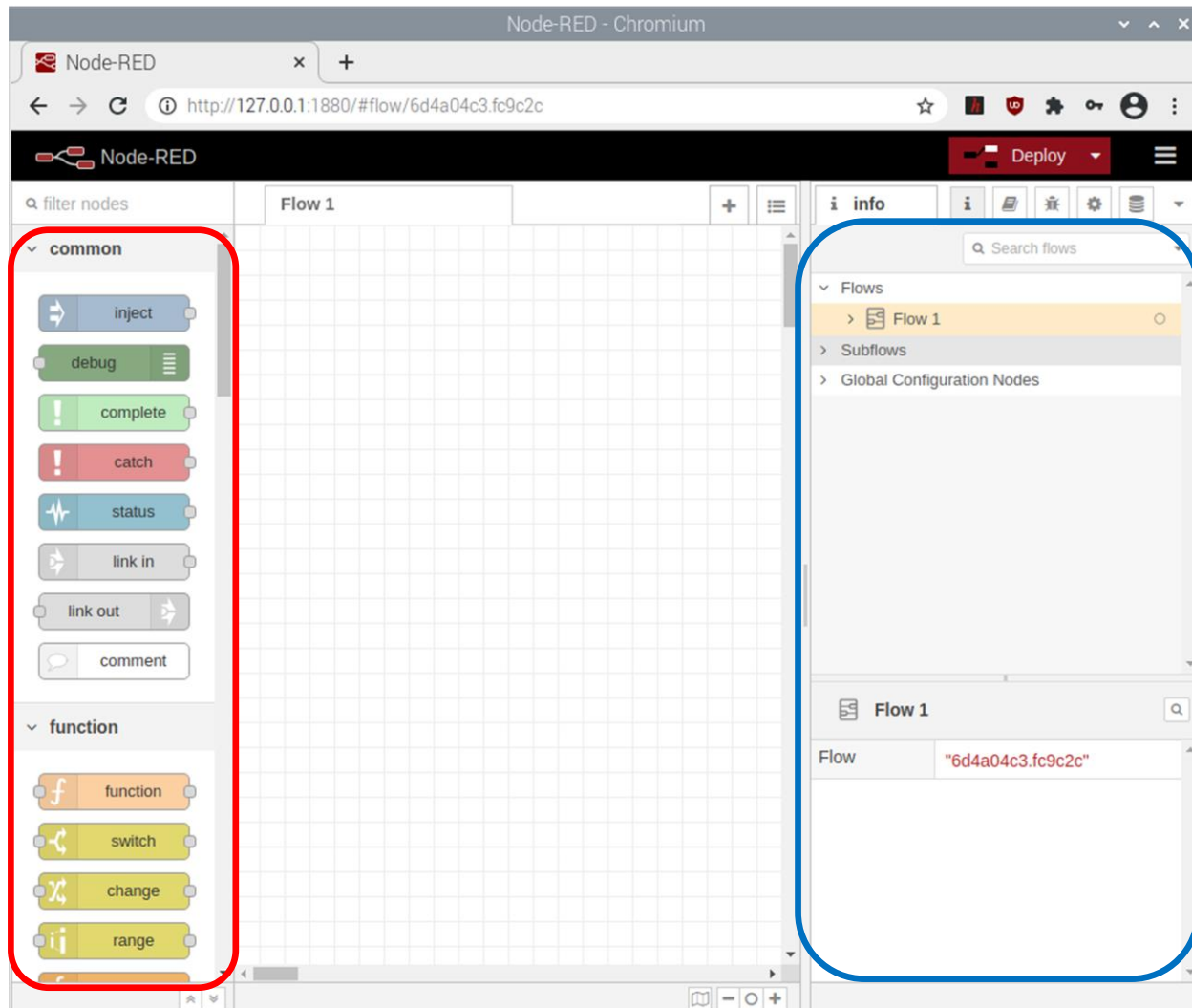


Figure 1-15: Node-RED IDE Components

What you are looking at is the Node-RED IDE (Integrated Development System) which is just a workbench with big box of tools and a bin full of parts to build your system with. At the left side you will see an area called the “palette” (outlined here in red), which is like a big bin full of part. In this case, the parts are different types of nodes arranged in groups. When you start, your palette will have a collection of commonly used nodes, but you can add to your palette at any time from nodes that other people have built. As you will learn later there are literally thousands of nodes available covering almost anything you can imagine. In a few more pages you will begin working with the I/O Learning card for your Raspberry Pi. This piece of hardware, like many other pieces of hardware you might use in the future, has its very own set of specialized nodes to make the task of implementing your ideas easy.

The big grid in the center is the “workspace”, which is where you will build your flows. You will do this by dragging nodes from the palette and dropping them into the workspace. Then using the mouse, you will connect the nodes with wires to define how messages are to be passed between the nodes. The workspace can hold many flows because you can add a new flow to the workspace by clicking the plus sign on the top right of the workspace. This will create a new, empty tab for a new flow. You will find

## Learning Kit Workbook (version 1.3)

out later that you can connect nodes located on different flows, so the overall flow can be as complex as you need but somewhat manageable because it is sub-divided into tabs. In fact, you can probably make a flow so complex that you will not understand it anymore... be careful.

On the right side of screen (outlined in blue), you will see the output area (called the sidebar) that contains several tabs representing different kinds of information you might be interested in. This is the set of tools you will use to determine if your flow is working correctly and, if not, what might have gone wrong.

At the top of the sidebar is a set of tabs as shown below.



Figure 1-16: Node-RED Sidebar

Although the tabs have tiny icons, you can always find out what a tab does by hovering the mouse over the tabs. Here is summary:

- Information – the tab with the “i” symbol – provides you with current information about an object you have selected in your workspace. For example, you can find out the unique serial number of a node this way.
- Help – the tab with the little book icon – gives you a help file for the selected object in your workspace. This is handy if you are trying to figure out how a node will handle a message.
- Debug Messages – the tab with the little bug icon. While your flow is running it will print out messages here, especially if something goes wrong. It is like a tiny flight recorder.
- Configuration Nodes – the tab with the wee gear. This shows you all the Configuration nodes in your flows. Configuration nodes allow you to establish useful parameters when your flows start. This will make sense later when we make use of Configuration nodes.
- Context Data – the tab with the picture of a stack of disks. Sometimes nodes want to share information outside of the messages that are passed. Context data definitions allow this. Context data is an advance topic we will cover later.

Take a few minutes and poke around on the Node-RED IDE. You will find that most things you want to do are intuitive. So, experiment away. Try to work with some nodes. Don't worry about building something that works, just play around with the IDE. Most things work the way you expect. So try the following:

- Drag a node from the palette to the workspace and drop it some place.
- Move the node around in the workspace.
- Drag in some other different nodes.
- Can you delete one of the nodes in the workspace.
- Can you figure out how to copy a node?
- Look at one of the nodes you dragged in. It should have a small bubble on the left or right side. Click and drag on the bubble. What happened?

## Learning Kit Workbook (version 1.3)

- Try clicking on a bubble of one node and drawing a wire from there to the bubble of another node. Can you connect them? If you could not, try different nodes and different bubbles on those nodes. Do you have some ideas about which nodes can be connected together and which can't?
- Some nodes have bubbles on the left, some on the right and some on both sides. Can you figure out which bubbles can be connected to each other?
- Try adding more than one wire to a node and connecting it to some other place. Can you do this? What are the restrictions, if any?
- Connect two nodes together with a wire. What happens if you move one of the nodes?
- Can you figure out how to create a new flow by creating a new tab in the workspace? Can you switch between the different tabs?

Did you figure it out or are you completely frustrated? Either way it is okay because sometimes you just have to try things. Now let's talk about how things are done in Node-RED.

*Placing a node in the workspace* – Easy enough. Just click on the node you want from the palette, hold down the left-hand mouse key (or the right mouse key if you and your mouse are lefthanded) and drag the node to where you want it. If you want to move it to a different place in the workspace, just click on it and drag it to its new home.

*Getting rid of a node* – Did you figure this one out. Click on the node to highlight it then hit the delete key. Kapow... goodbye node (and any wires connected to it)!

*Make a copy of a node* – Later you will find that you can change how a node works so that you can develop nodes that are specialized for your needs. Being able to copy a node will save you time and effort. This works just like cut and paste in a text editor or spreadsheet. Click on the node to highlight it then use *ctl-c* to copy and *ctl-v* to pop the copy onto the workspace wherever the cursor is currently pointing.

*Connecting nodes* – Did this drive you crazy? Sometimes it worked and sometimes it did not? To do this right you have to understand something about nodes. Nodes work from left to right. Technically, the bubbles on the left and right of a node are called "ports". There are two types of ports input ports and output ports. A port located on the left side of a node is an input port and can only receive messages from a connecting wire. Similarly, a port on the right side is an output port and is the point from which messages may be placed on a wire. A node may have no input ports (look at the Inject node, for example), but if it does have an input port it may only have one. Nodes may have one or more output ports on the right-hand side, but again some nodes, like the Debug node, have no output ports. And, of course, some nodes may have both an input port and an output port, like a Function node.

Here is the important thing to remember. A wire can connect only one input port to one output port. You cannot connect two output ports together or two input ports together. However, you can use two wires connect an input port of a node to two or more output ports on other nodes. What do you think happens if you do this? Also, you can connect multiple input nodes to one output port. Again, what do you think is going to happen to messages if you do this? Once you set up your first simple flow you can try these arrangements and see what happens.



## Learning Kit Workbook (version 1.3)

Creating a new flow – If you need a new place to create a flow, you simply click on the plus sign at the top right of the workspace. You can move between flows by clicking on the tab of the flow you want.

That's it. You now know enough to create your first flow.

## Learning Kit Workbook (version 1.3)

### Chapter 2 - Your First Flow – “Hello World”, What Else?

What you are going to learn:

- How to set up a simple flow
- How to use the Inject node
- How to use the Debug node
- How to change the way a node behaves
- How to change the name of a node
- How to name your flow
- How to leave a comment in your flow

#### Hello World – As Simple as it Gets

As is long standing tradition, your first program must be “Hello World” because if you don’t write this flow first you risk upsetting the delicate balance of force throughout the programming sub-space.

Start by bringing up a blank flow tab in your workspace. If you got here by experimenting around you might have a screen full of nodes, probably labeled Flow 1. Let’s get rid of it and start with a fresh screen. To do that, first create a new flow by clicking the plus-sign in the upper right of the workspace. Now if you have two tabs on the screen (one that you were experimenting with and one you just created) double click the tab that you were experimenting with and you should see a new window giving information on the flow, something like this:

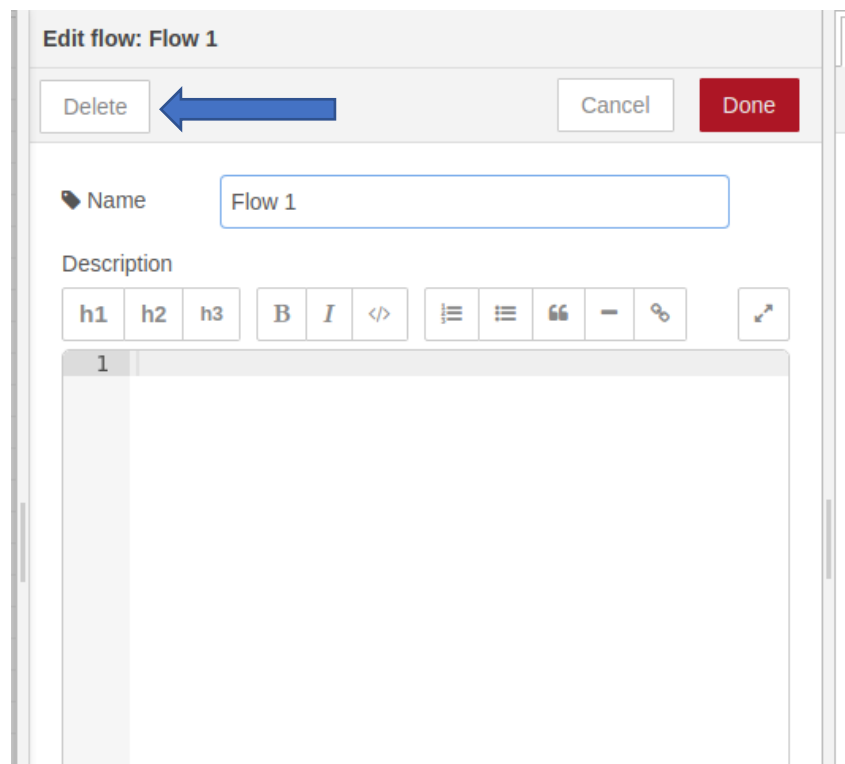


Figure 2-1: Deleting an Old Flow

## Learning Kit Workbook (version 1.3)

You will see that in the upper left there is a Delete button (see big blue arrow). Once you click this the flow for the tab is deleted and gone forever. Now, here is an important thing to know: if you have only one tab on the screen you cannot delete it, that is why you needed to create a new tab first. Also, we blew up the old flow you were experimenting with because if you leave old flows laying around, they might interfere with other work you are doing, and you will get strange and spooky results. What you just did above is make sure you have a completely new flow to work with.

Now that you have a brand-new workspace, you are going to bring in two nodes from the palette. The first is going to be an Inject node and the second is going to be a Debug node. You will see them at the top of palette as shown below:

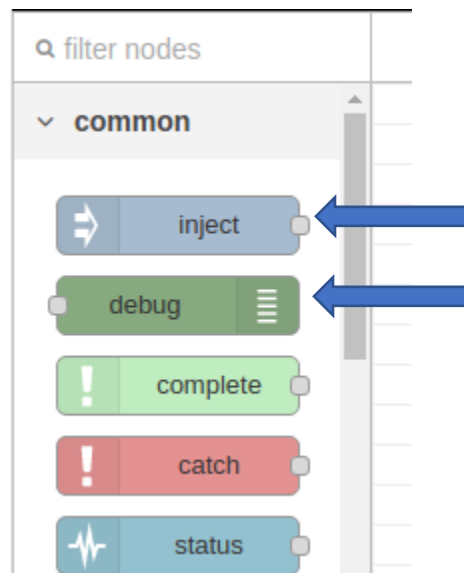


Figure 2-2: Inject and Debug Nodes in the Palette

Grab the inject node and drag it to the left side of your workspace (do this by clicking on the Inject node, holding down the mouse button, dragging the copy to where you want it and dropping it, remember? Click, hold, drag, drop. Next grab a debug node and drop it on the right side of your workspace. Then wire the two nodes together (click on the output port of the Inject node, drag the wire to the input port of the Debug node and let go). Your flow should look like this:

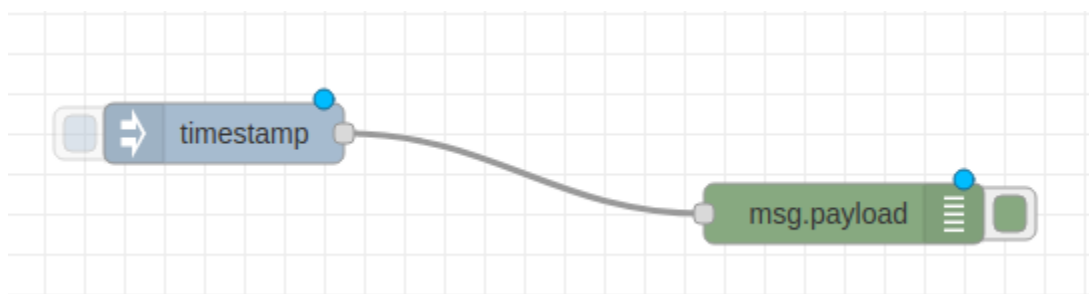


Figure 2-3: Simple First Flow before Deployment

## Learning Kit Workbook (version 1.3)

If you have sharp eyes, you might have noticed that the Inject node does not say “Inject” anymore, rather it says “Timestamp”. This is because all Inject nodes start off in life sending a timestamp message on their output port. Later you will find out that you can make the Inject node send any sort of message you want, which is a terrific help in debugging and testing.

Question: Why does the Inject node have an output on the right side of the little block and the Debug node have an input port on the left side? It is because by convention engineering drawing usually (well, almost always) show information flowing from left to right. Which brings us to...

---

**Engineering Tip # 1:** Since time immemorial, or at least since the invention of the telegraph (which ever came first) engineers have been drawing process diagrams where the inputs are on the left side and outputs are on the right side. Generally, air, water, coal, electrical signals, hamster chow pellets, or whatever else is being processed starts on the left side and moves thorough steps to the right side. The same applies to electrical diagrams and by extension to the kinds of message passing in the flows you are creating. Of course, Node-RED does not care where you put your nodes and what direction the message flow in, top to bottom, right to left, bottom to top, southwest to northeast, whatever you want, because the messages know where to go: they just flow from the output of one node to the input of the next node along the wires you drew. However, now is a good time to join the legions of engineers that went before you and set things up so that generally messages flow from left to right. It will make your flows easier for other to read. Actually, Node-RED does care a little bit because inputs are always on the left side of node and outputs are on the right side. So, if you try to swim upstream against tradition and set your flows up so that they work from right to left you will have an incomprehensible, tangled mess.

---

Your flow is in the workspace and is all wired up. Is it ready to go? Not quite... did you notice the little blue bubbles at the top of the two nodes? These indicate that these nodes have not been deployed. Before your flow can help you out you must “Deploy” it, which means starting it up, just like you might run other types of programs. Look at the top right of your screen and find the sidebar tabs.

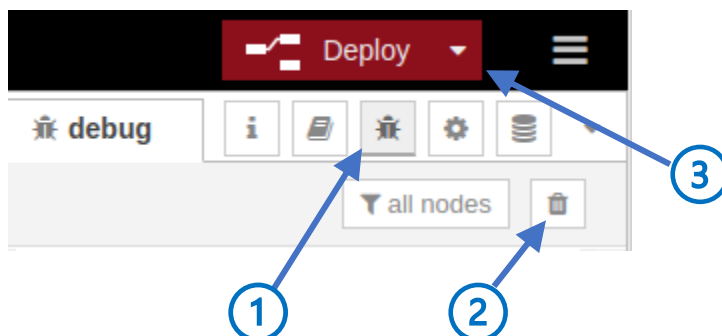


Figure 2-4: Deploying Your Flow

## Learning Kit Workbook (version 1.3)

- You will want to see the result of the flow. To do this go to the sidebar area on the right side of the screen and click on the tab that has the little picture of a bug (see ① on the figure above). When you do this, the area under the tab will show the debugging messages that occur in your flow. The Debugging tab is a tool you can use to inspect the operation of your flow, but right now we are going to use it as a makeshift display area.
- Next, clear the Debug output by clicking on the little trashcan at the top right of the output area (see ② on the figure above). The output area should now be empty, which will make it easier for you to see the results of your flow operating. If you don't do this the messages from your flow will probably be lost among messages from the operation of previous flows.
- If you make changes to your flow (even something as simple as moving a node to a new location), or if it is a new flow, you must deploy it for the flow to become active. To do this, click on the "Deploy" button ③ just above the top righthand side of your workspace. When you do, all the blue dots indicating nodes that have not been deployed will disappear and a message will drop down above your workspace hopefully say: "Successfully deployed". Your flow is now almost ready to test.

At this point your workspace should look like this (more or less):

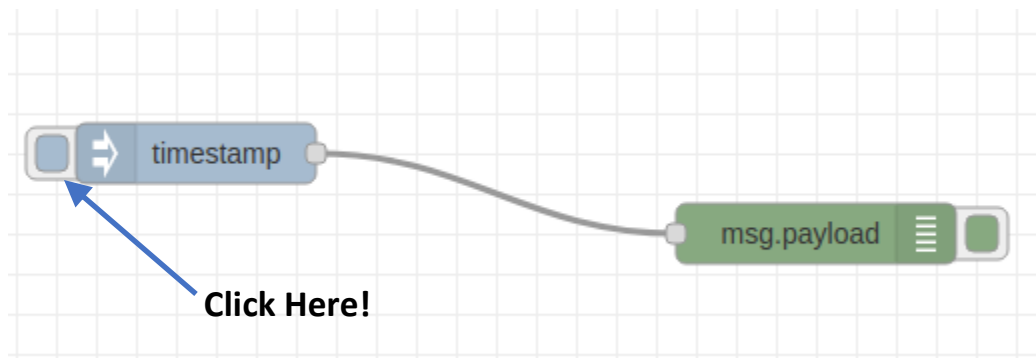


Figure 2-5: Deployed Flow

This is the big moment... take your mouse and click on the little tab sticking out of the left side of the Inject Node (Click Here!). If you set everything up properly you should see a message appear in the Debug output area on the right. It will look something like the figure below.

## Learning Kit Workbook (version 1.3)

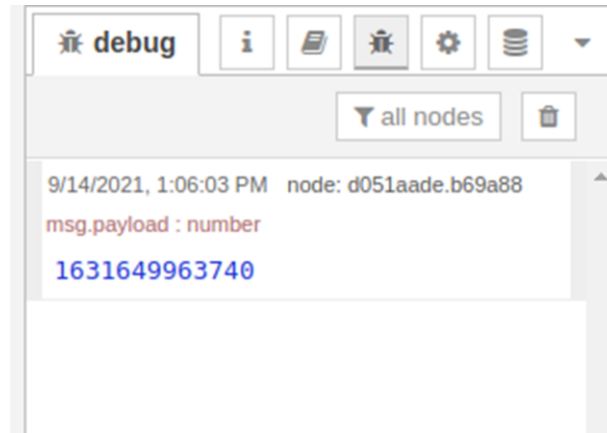


Figure 2-6: Debug Output for First Flow

Well, that is a little disappointing... the date, giant strings of numbers. No “Hello World” and just a little bit of rumbling in program sub-space because you are breaking the rules. What’s going on? Time for a little debugging.

---

**Engineering Tip # 2:** Debugging... As you do more and more with Node-RED, electronics, programming, robotics or any other task where you are building something you will find that, sadly and predictably, most great ideas do not work out the first time you try them. In fact, a great deal of engineering time is spent in testing and correcting your work. This is the process of debugging. We will talk quite a bit about this later because this is one of the things engineers must do, it is part of the practice of engineering. If you are skilled at debugging you will never be out of work, so start building good habits right now. Debugging is the process of removing “bugs” from your design and “bugs” are behaviors that you did not expect. The term “bug” is thought to have originated with Thomas Edison, but there are all sorts of apocryphal stories (yes, get the dictionary and look this word up, it will be on the SAT). Check out [https://en.wikipedia.org/wiki/Bug\\_\(engineering\)](https://en.wikipedia.org/wiki/Bug_(engineering)).

---

Debugging: the first thing to do is try to understand what is happening. Certainly, something happened because the Debug output shows a little message. In fact, this is the message received by the Debug node over the wire from the Inject node when you clicked on it. So, try clicking the Inject node again. What happened? You should have seen a message just like the first message, but just a tiny bit different.

The Debug Node is doing exactly what it is supposed to do, which is to print on the Debug output tab every message it receives at its input port. This is a very useful feature of Node-RED. The Debug message allows you to listen in on messages being passed around between the nodes. In this simple case the only message the Debug node is receiving is the message that is sent every time you click the Inject node

## Learning Kit Workbook (version 1.3)

Messages in Node-RED are a bit like telegrams or mail, that is, they contain a time, the name of the node who receives the message, possibly who sent the message and contents of the message. The contents of messages in Node-RED are referred to as the “payload”, which is similar to the terminology we use for rockets. The payload is the valuable part of the rocket that you want to get into orbit. In Node-RED the message payload is the valuable part that you want delivered somewhere, all the rest of the information you usually don’t care about unless something goes wrong, like right now. Let’s look at the message you just passed to the Debug node. Here is an example of a message from working with this simple flow, hopefully yours looks similar:

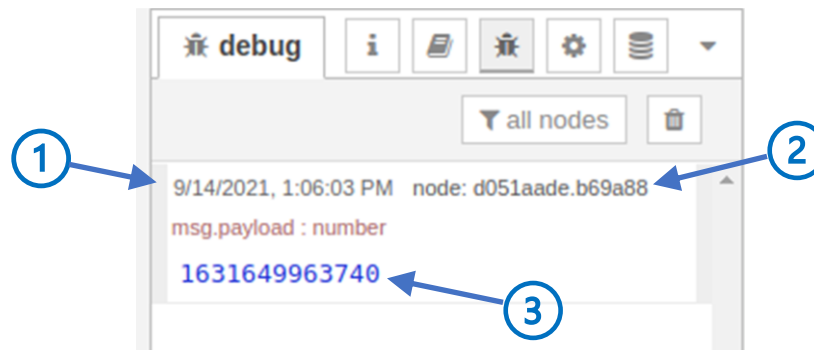


Figure 2-7: Debug Message from your First Flow

In the top right of the message are the date and time when the message was received by the Debug node (① in the figure). To the right of that is the word “node” followed by a big mysterious number (② in figure). This number is in [hexadecimal](#) and is a unique serial number identifying the node that received the message. In this case it is the serial number of the Debug node and is unique throughout your flow (and probably unique throughout the known universe). Thankfully, you do not need to check every node in your flow (or the known universe) to find out which node has the serial number. Instead click on the serial number and an animated outline will appear around the node that received the message, in this case the Debug Node. Try it!

The next line identifies the type of the message payload. In this case the payload is a number the value of which is shown directly below (③ in figure). What on earth is this number? It is the payload of the message, the important part, and it is a “timestamp”, which is a number that encodes the exact time the message was generated by the Inject node. This in turn, was the exact time that you clicked the tab on the Inject node. If it doesn’t look to you like a time of day that is because it is just a number representing the number of [milliseconds](#) since January 1<sup>st</sup>, 1970 at the [Prime Meridian](#) in Greenwich, England. Say what? That’s not very helpful. Well actually, it is pretty useful because you can do arithmetic on these timestamps if you want to find out how much time has passed between two points in time or find out what time it will be two weeks from now. But... not so helpful if you want to know what time it is in the real world.

Fortunately, Node-Red helps you out here. Put your cursor on the timestamp number (the blue number) in the message and click. Bingo, the time stamp is converted to a more useful format. Every



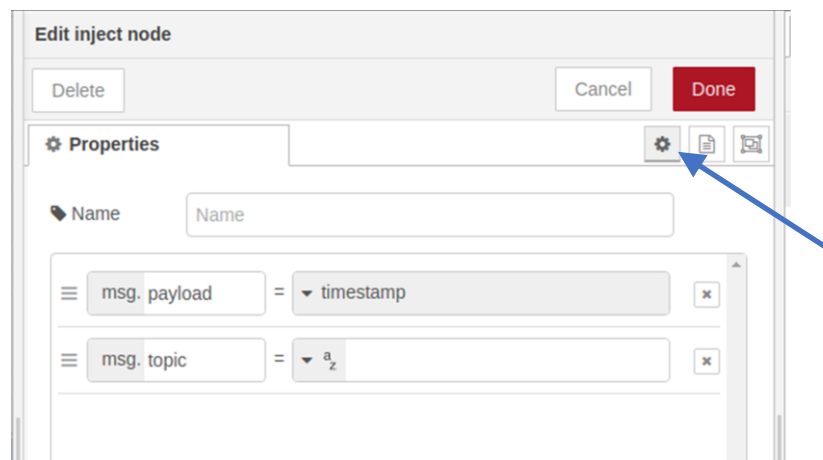
## Learning Kit Workbook (version 1.3)

time you click the timestamp it will be converted into a different format, so pick the one that makes sense to you. This will probably be the one that shows your local time.

By now you're thinking, "This is all very interesting, but really, I wanted to see "Hello World", especially because I can already feel little rips and tears in the sub-space of programming." Think! Where is that timestamp coming from? The answer is that it is coming from the Inject node. The Inject node is another useful tool. When you click on the tab, the Inject node will generate whatever message you would like. It just happens that the *default* message (that is the message you get if you don't do anything else) is the timestamp of the time when you clicked the Inject node. It is a bit like sending an envelope to someone with a note inside saying "I sent this to you at 1:06PM on 9/14/2021". Not all that helpful because they can see when you sent the letter by looking at the cancellation on the outside, like ① in the figure. What you really would like to do is change the message that is being sent so that it says, "Hello World".

When we first started talking about nodes, we mentioned something about being able to modify nodes. That is what you are going to do here, you are going to modify the Inject node so that it sends the message we want, namely "Hello World". To modify the node, you must take off the covers, so to speak, and tinker around with the message it generates. It's easy: try this.

Double click on the Inject node in your flow. This will open a little editor that will allow you to make changes to many aspects of your node, for example, the appearance, how it functions and the message it will send. Here is what you should see:



*Figure 2-8: Edit Window for Inject Node*

If you do not see this, select the Properties tab, which is the tab with the little gear symbol on it (see blue arrow). By now you have probably figured out that if you see an icon, like the little gear, and you don't know what it means, all you need to do is hover the cursor over the symbol and a little box will appear telling you what the symbol relates to. In this case, the little gear brings up the Properties window.

Now, you can change many aspects of the node, including the message contents. Keep in mind that different nodes will have different properties for you to fill in. For the Inject node you want to change

## Learning Kit Workbook (version 1.3)

the message because rather than sending the timestamp, you really want to send “Hello World!” to the next node.

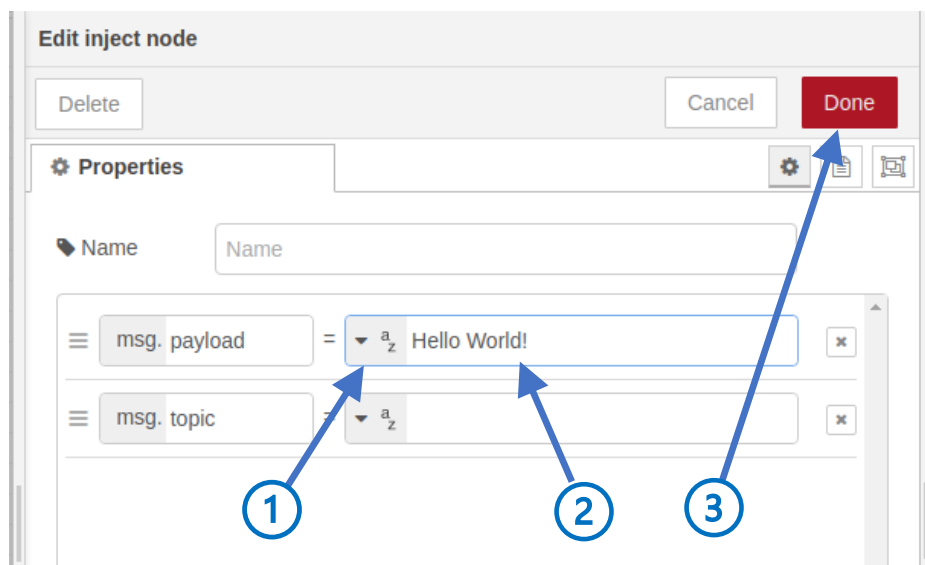


Figure 2-9: Inject Node After Editing

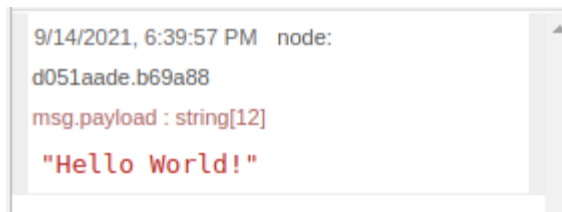
The line in the properties menu labeled *msg.payload* defines the payload of the message generated by the Inject node. Right now, it is showing that you are putting the timestamp in the payload, but let's change that. First, click on the dropdown button (① in figure above) to the right of the equal sign. This will open up a big list of options for the types of payloads you can send. Select “String” because you want to enter a string, like “Hello World!”, in the payload part of the message. Next, to the right fill in the value of the string, namely “Hello World!” (② in figure). You don't need to use quotes around the string because you already specified that the payload type was string. To finish up, click DONE (③ in figure).

Look at your Inject Node. Now it says “Hello World” inside the node. This is because the name of the node is just the message payload unless you specify otherwise. Later, we will talk about how to personalize the node, like changing the name, the color and so forth.

Remember, you have just changed one of the nodes in your flow, and this means that your flow is now out of sync with what you deployed earlier. You know this because the Inject node has a blue dot on the upper right of the node figure. This means you have made a change to the node that has not been deployed yet. Deploy the node again by clicking the Deploy button.

## Learning Kit Workbook (version 1.3)

Now it is time for a test. Select the debug output (the little bug icon) and clear the debug window (the little trash can). Now click the tab on the left side of your Hello World! flow. When you do this the node generates a message with a payload of “Hello World!” and sends it to the Debug node. The only job the Debug node has is to display the message. When you click the tab, you should see something similar to this:



```
9/14/2021, 6:39:57 PM node:
d051aade.b69a88
msg.payload : string[12]
"Hello World!"
```

Figure 2-10: Debug Output from Debug Flow

If you see the message, then Congratulations! You have just written your first flow, and you should feel good because programming sub-space it at peace again or at least until someone else tries to learn a new language.

Click the Hello World node a few more times. Each time you do this you should see a new debug message in the output. You can see the message is new because it will have a unique time and date on it.

### Dreaded Documentation

Time to move on to greater things. Before you do, think for a minute about whether or not other people will understand your flow if you leave it like it is. Which brings us to the topic of “Documentation”, perhaps one of the most despised words in the engineering world. It’s no fun, it’s boring, it’s tedious, but it is necessary.

---

**Engineering Tip # 3: Help other understand your work!** Now that you have a working flow you will want to save it for later use. Before you put it away, ask yourself this, “Am I going to remember what I did two years from now, or even next week?”. Your memory might be great, but what if someone else wants to use your flow and you are on an expedition snorkeling in search of sea hamsters in the South Pacific 500 miles from the nearest cell tower? What are your poor co-workers going to do? Better put some comments in your code so that others know what it is. Also, you might want to put in comments for anything unusual, clever, tricky, strange, obtuse or bizarre that you implemented in your flow.

---

## Learning Kit Workbook (version 1.3)

Before we leave this simple flow behind, let's do a few things to make it more understandable. The first thing to do is put a name on your flow tab that means something. When you open a new tab Node-RED just put "Flow 1" or Flow 2" and so on, in the tab name, which is not very helpful. Let's change it to "Hello World". Do this by double clicking on the name "Flow 1" in the tab, which will open a window where you can give the flow a new name by replacing "Flow 1" in the Name box (blue arrow) with "Hello World" (without the quotes, of course)

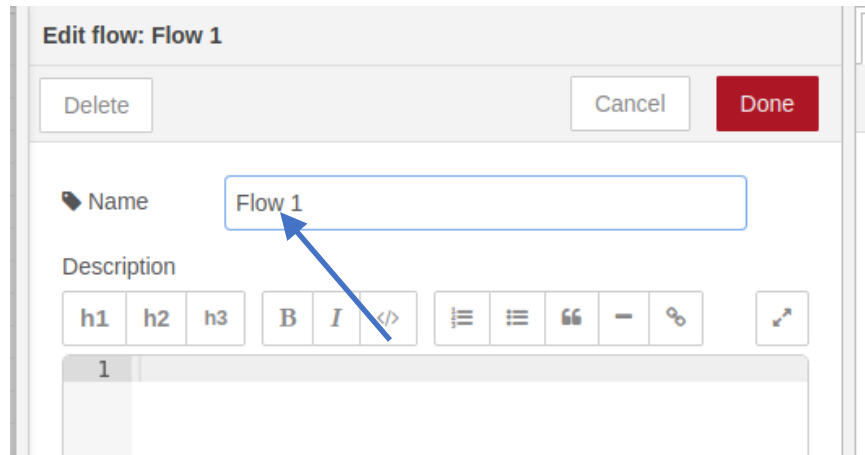


Figure 2-11: Editing the Flow Tab

Better document your code, which means leaving a description of what your code does, how to use it and how it works. Documenting your work can mean many things. Using clear and meaningful names for nodes and flows is one aspect. Another aspect is to add descriptive material to your flow. In Node-RED you can do this in several ways. One of the easiest approaches is to add comments to your code, which is done by inserting Comment nodes at appropriate places in you flow. A Comment node is a "dummy" node that has no input or output ports. Its only purpose is to display text explaining some aspect of your flow. You handle a Comment node just like any other node: drag and drop it where you want it to appear, like this:

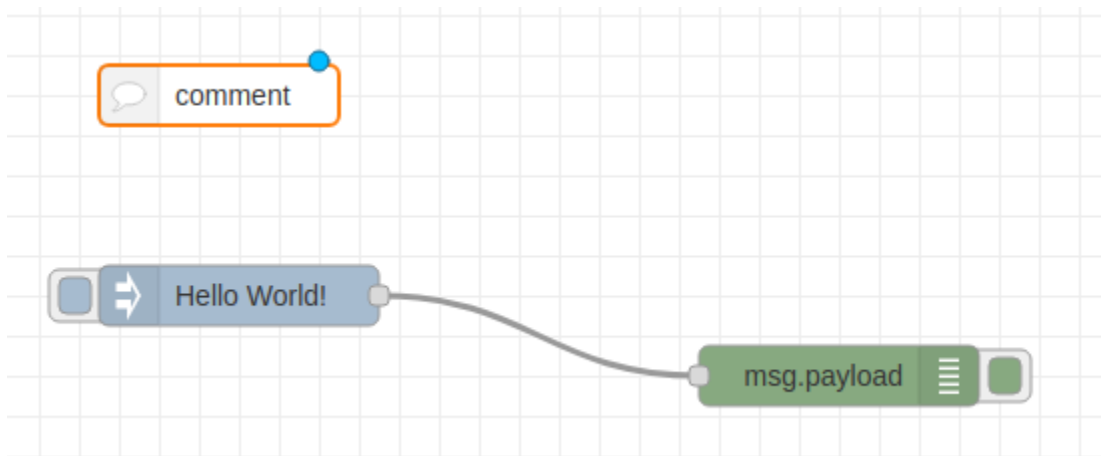


Figure 2-12: Hello World Flow with Comment Node

## Learning Kit Workbook (version 1.3)

Next, edit the Comment node to provide the reader with some useful information. Do this by double clicking the Comment node to open the description tab (see figure below). This tab contains a mini editor.

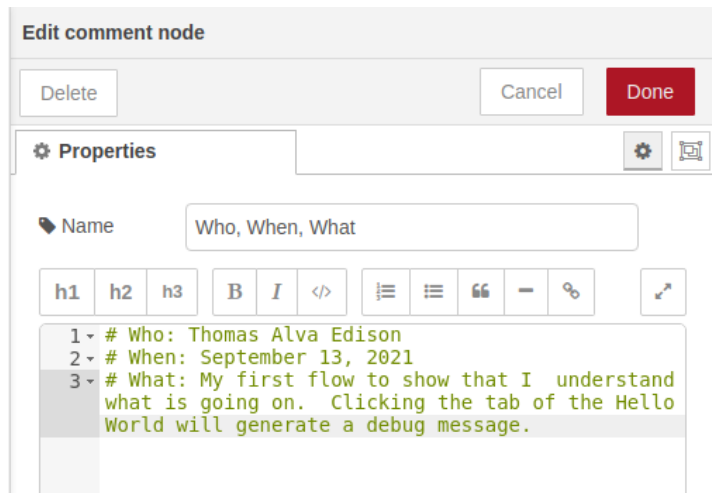


Figure 2-13: Comment Node Editor

The node editor allows you to write out somewhat sophisticated descriptions of your node. We will not go into the details of how to use the editor here, but you should know that descriptions, such as the one above, are written in the [Markdown](#) language, which is a very simple formatting system. Click the link in the last sentence for more details, or simply poke around at the editor and see what it can do. Briefly, to control formatting of the text you add characters, like # and \*, to the basic text. The editor does this for you. For example, if you highlight a word and click the big “B” button the editor will surround the word with two asterisks (e.g., **Who**) which means it will show up as a bold word in the information window. Below is an example of comment text as it appears in the editor.

When you click Done, the comment will say “Who, When, What” and if you double click on the comment node you can read the description. In a well-designed flow all nodes, not just the comment nodes, will have appropriate descriptions to help the user understand what the node does. In this way the flow is self-documenting meaning that folks don’t have to go someplace else to find out what is going on with your flow.

## Learning Kit Workbook (version 1.3)

You do not need to open the node directly to read the description. If you select the node (single click on the node) you can read the description in the information tab of the sidebar section (click the tab with the little “i”). Try it by clicking on the Flow Description comment node and then clicking the information tab. You should see something like this:

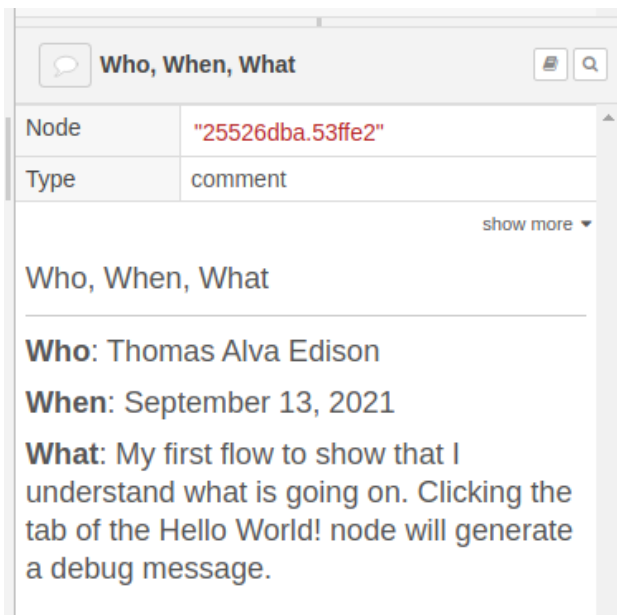


Figure 2-14: Information Display for Comment Node

Notice that the information display above no longer contains the weird Markdown characters and instead shows the formatting that those characters define.

### Packing up, Cleaning House and Moving On

We interrupt this program to bring you a public service announcement from the Better Engineering Council.

---

**Engineering Tip # 4 – Save your work early and often!** One good habit you should develop when you work with Node-RED, or any other language for that matter, is to save your work periodically. Don't just save it to the same file, but periodically save a copy to a file with a new name, like my-precious-project (backup 1).json. This way if you make a mistake or find out that a change you have made to your project is not working the way you expected, you can go back and start from the last working version. There are many systems (like [github](#)) to help you keep track of your work in an organized way, but for the time being we will just use a simple approach of saving a copy of the flow every now and again.

---

In Node-RED you do not save your work directly, as you might do in a text editor. Instead, you “export” it to a file and then save the file in any way that you wish to. To restore a flow you have saved, you “import” the file from your file system into the Node-RED workspace.

## Learning Kit Workbook (version 1.3)

### Putting It Away: Exporting a Flow

To export your flow first click on the tab of your Hello World flow to select it. Next, click on the menu symbol ① at the far right of the black bar above the workspace. When you do this a list of choices will drop down. Now click Export ②.

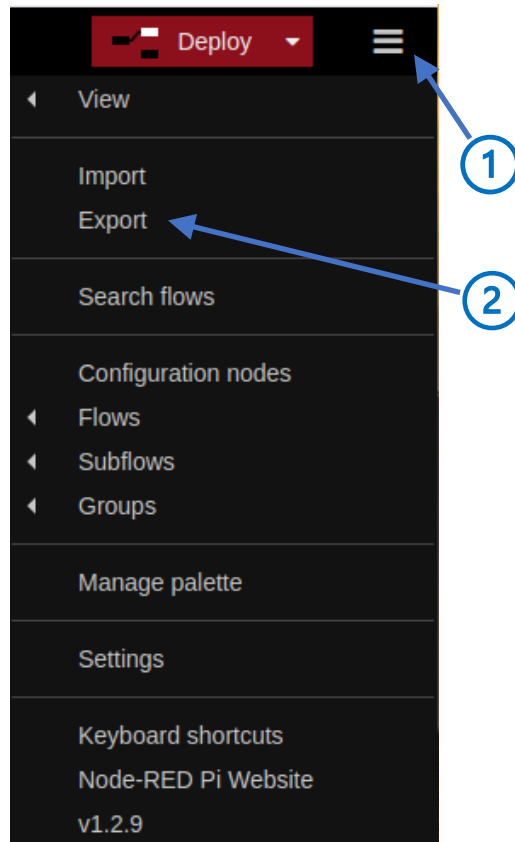


Figure 2-15: Exporting a Flow

After you click Export on the main menu you will get the window in the figure below. Your choices are given in the tabs at the top and include exporting only a node, exporting one particular flow or exporting all the flows in your workspace. Let's export your Hello World flow. Do this by selecting "current flow" from the tabs ①. When you do this, you will see the code for the Hello World flow in the window. It looks like a mess, but it is really a very organized description of your flow in [JSON](#), a sort of [lingua franca](#) for passing information between different systems. It is human readable, but to make sense of it you will need to learn much more about the underlying structure of Node-RED than we will cover here.



## Learning Kit Workbook (version 1.3)

Click Download ② and the current flow, Hello World, will be downloaded as a text file with the file extension .JSON. Now you can save the file some place safe. Notice that the downloaded file has a rather bland name, like flow(1).JSON. When you save the file, you will want to give it a descriptive name, like “Hello World.JSON” indicating what it is supposed to do. If you are making a backup copy you might give you file a name like *Hello World (bak 3).json*.

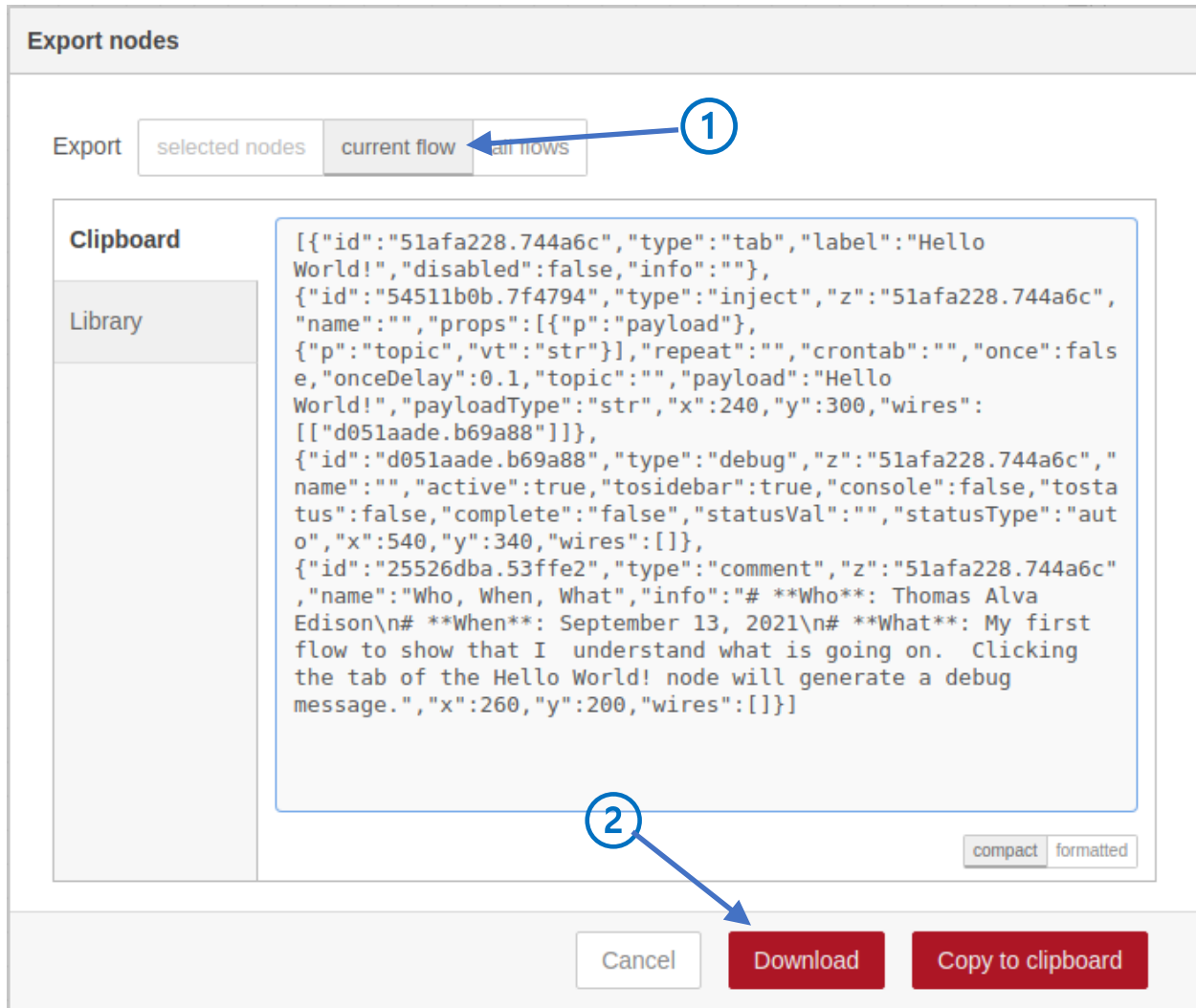


Figure 2-16: Downloading a Flow

If you are curious about the structure of the flow, click the formatted tab in the lower right corner and the JSON file will be laid out in a more understandable way. No need to do this, it is just if you are curious, which is almost always a good thing in engineering.

### Cleaning House: Deleting a Flow from Your Workspace

Okay... you have saved your precious first flow. Now let's clean up the workspace and see if we can get Hello World back. Here is how you delete a flow from your workspace, which is a bit non-intuitive. Strangely, you cannot delete a flow if it is the only one in your workspace. Thus, if you are trying to delete a particular flow and it is the only flow you must first create a new empty flow tab. You do this by using the technique we talked about earlier.

## Learning Kit Workbook (version 1.3)

First and foremost, make sure the flow you want to delete is selected. If you don't do this you might remove some other flow that you still want, so check it twice. You have been warned! Next...

Click the menu icon at the far right of the black header bar ①. This will drop down several options. Put the mouse on Flows and you should see this ② :

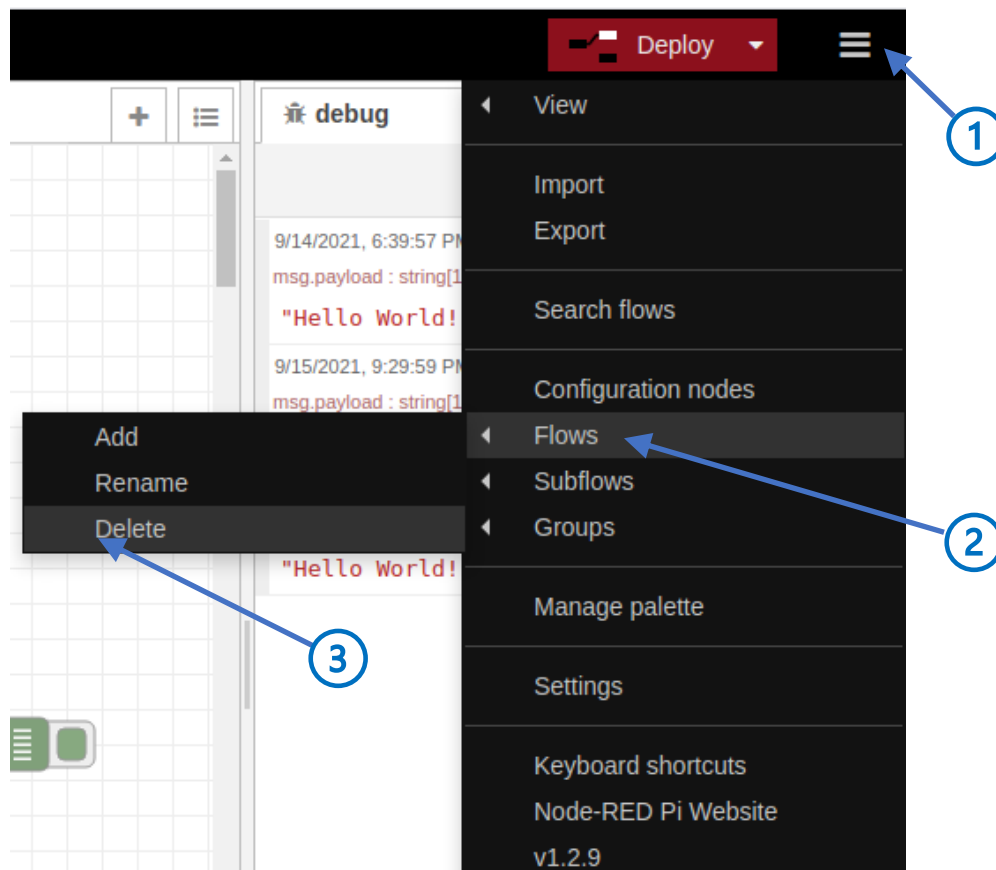


Figure 2-17: Deleting a Flow

Move the mouse over to “Delete” ③ and click. Poof! The flow has been sent to the great bit bucket in the sky and is gone forever. Of course, it's not really because you exported and saved it before you deleted it didn't you?

## Learning Kit Workbook (version 1.3)

### Bringing it Back – Importing a Flow

Time to see if you can retrieve your Hello World flow. The process is the opposite of exporting and is called “importing”. When you import a flow you have stored, you will simply be dropping it back in your workspace. You can also import nodes that you may have saved using a similar approach. Here is the magic formula: Click on the menu icon ① and select Import from the drop down ②.

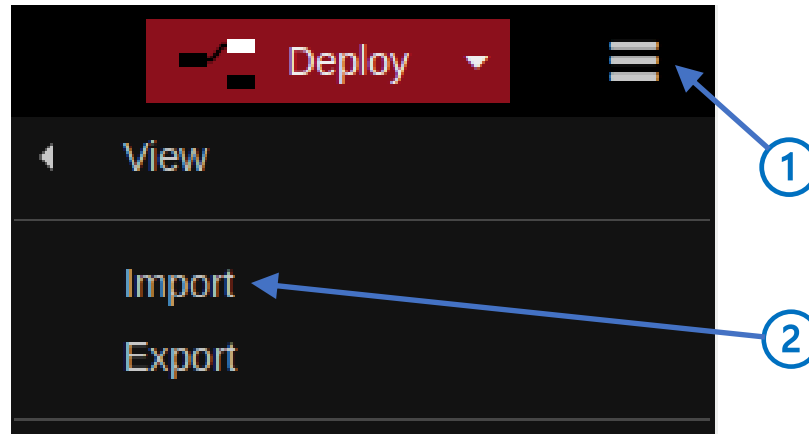


Figure 2-18: Importing a Flow

This will open a window like the one in figure below.

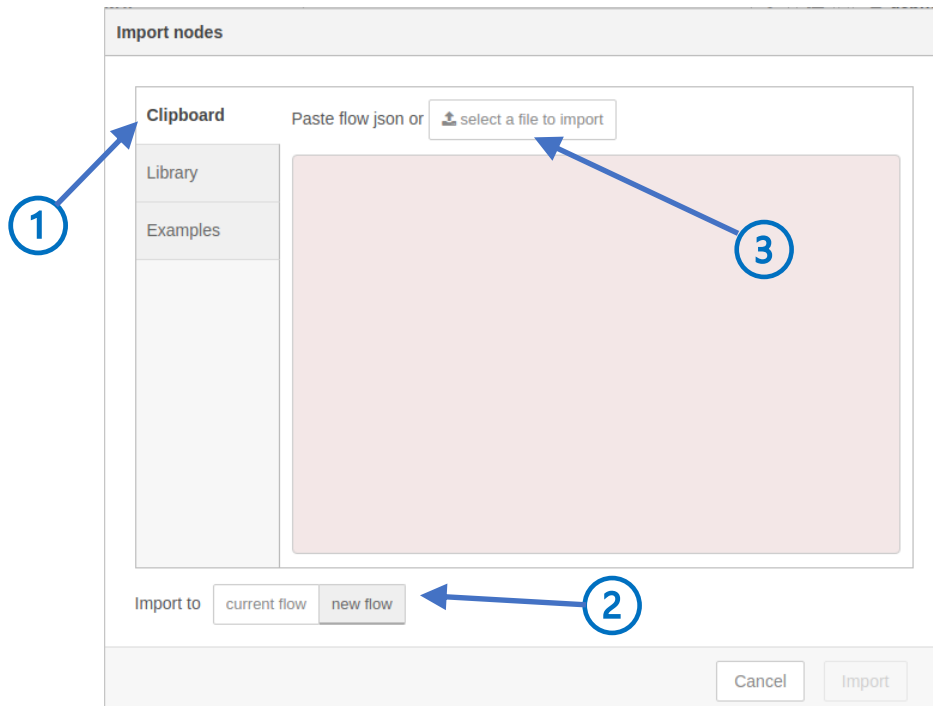


Figure 2-19: Importing a Flow

## Learning Kit Workbook (version 1.3)

First, you will need select Clipboard from the three tabs on the left ① . At the bottom you should select Import to new flow ②, otherwise you might accidentally overwrite one of your current flows. Now click on Select a file to import ③ near the top of the window and locate the file you want from your directory.

Use the file system on your particular system to locate and open the file you want to import. When you have done this you should see window full of text which represents the JSON code for the flow in the window as below. Click Import ① and voila! Your flow should be ready to use in a new tab. Easy! Don't

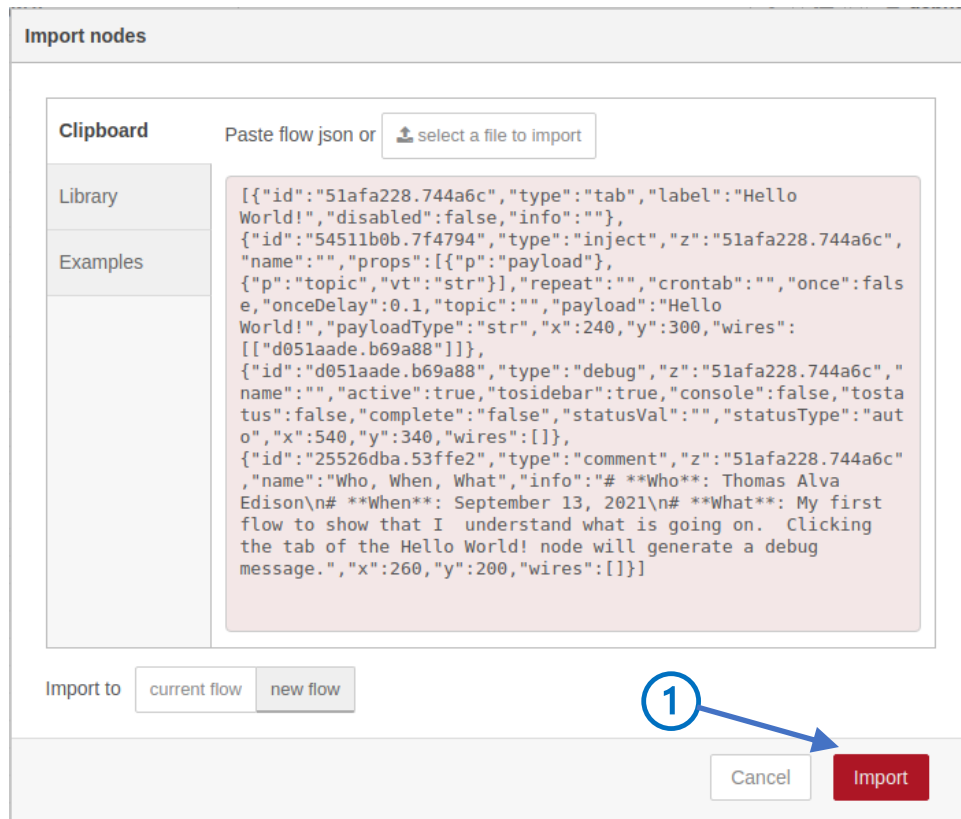


Figure 2-20: Importing the Hello World Flow

forget that before you can use the flow you must Deploy it.

### Graduation Time!

You have now used all the tools that you will need to start building flows with the I/O Learning card! Let's move on!

## Learning Kit Workbook (version 1.3)

### Chapter 3 - Using the Humble Pushbutton

What you will learn:

- Using the pushbutton on the Learning Kit Board
- Using Change Nodes
- Using Switch Nodes
- Simple Data Types
- Generating Sound
- Sending Emails
- Sending SMS Messages

#### Crawl, Walk Run

Now that you have written the mandatory “Hello World” flow it is time to move on to doing something at least semi-useful. Before we jump into projects that require external connections to the I/O Learning card, let’s see what we can do with just what is on the board. In fact, we are going to start with the simplest device of all, the humble pushbutton and see how far we can get with just this one device.

Crawl, walk, run... let’s start by crawling and after writing a few more flows you will be able to stand up and walk. Later you will run.

#### Crawling – Pushbutton to Debug Output

##### Inject and Debug Nodes

Back to the Hello World flow. It’s simple. Let’s see if you can modify this into something a bit more complex. Hmmmm... one push button. What could we do? How about this, let’s see if we can replace the Inject node with a pushbutton. Yes, it’s not that exciting, but ya gotta start somewhere. Before we start, think about what you would like the flow to do. A good objective would be to have the flow output Hello World when and only when the button is pushed. Let’s go, one step at a time. Now for a commercial break and a word from our sponsor The World Engineering and Programming Council...

---

**Engineering Tip # 5:** Keep a notebook! While it sometimes feels like it is slowing you down, in the end it will make you a much more effective engineer. An engineering notebook is a place to record what you are doing and ideas you have about future projects. The reason to write stuff down is that you will forget things, important things. Your notebook is the place to go when you forgot where, when and how you downloaded that really effective application, circuit, hamster wheel brake design, whatever.

Many very skilled engineers use a bound notebook and record everything on paper. If you do this keep the following in mind:

- Write your name on the inside cover, just like grade school. If you lose your notebook then there is at least the possibility it will be returned.
- Use ink. If you use pencil, you will start erasing things so that you can’t be sure of what you did. If you later work in some big company, you will probably be required to keep a notebook and you will be required to keep it in ink because sometimes your notebook is a record of when you invented something important.

## Learning Kit Workbook (version 1.3)

- Date everything. When you start the day write down the date. This will be a big help if you need to go find something in the past.
- Try to write neatly. Yes, it is exciting to get your program running, but if you just scribble down that web site name in your notebook you are never, ever going to find it again. Deep breath, exhale slowly, write carefully.
- Don't blot things out. If you make a mistake, don't scrawl all over it with your pen, but rather just put a single line through it. You will be surprised how many times you will go back to something you crossed out and later found out it was actually correct. If you obliterate it so that you can't read what you wrote, then it won't be much help.
- Follow your notes. One of the important things to keep in your notebook is procedures, that is, recipes for how to do something, like how to download a program from Github. You might do this a dozen or more times. After you write down a procedure, next time you need to do that procedure follow the steps in your notebook to make sure it is still correct. Make changes to reflect anything that is new or different.

A paper notebook is not the only way to keep notes, you can always keep notes using a word processing program or text editor. Just be sure you make your work permanent by saving it often and making backup copies. Using a remote service like Google Drive, Dropbox etc. is a good idea, although it does give one pause to imagine what these folks are doing with all that data they store for free.

---

OK, back to the task at hand. Start out this way. Follow the directions above and save your Hello World flow. Be sure to give it a descriptive name and if you are smart you will also write it down in your notebook so you can find it again. Now rename the Hello World flow in your workspace from "Hello World" to "Pushbutton". You still have the tiny flow from "Hello World" in your workspace, but next you are going to modify it a little bit at a time until the push button activates the Hello World output, thereby solidifying your command of the programming subspace.

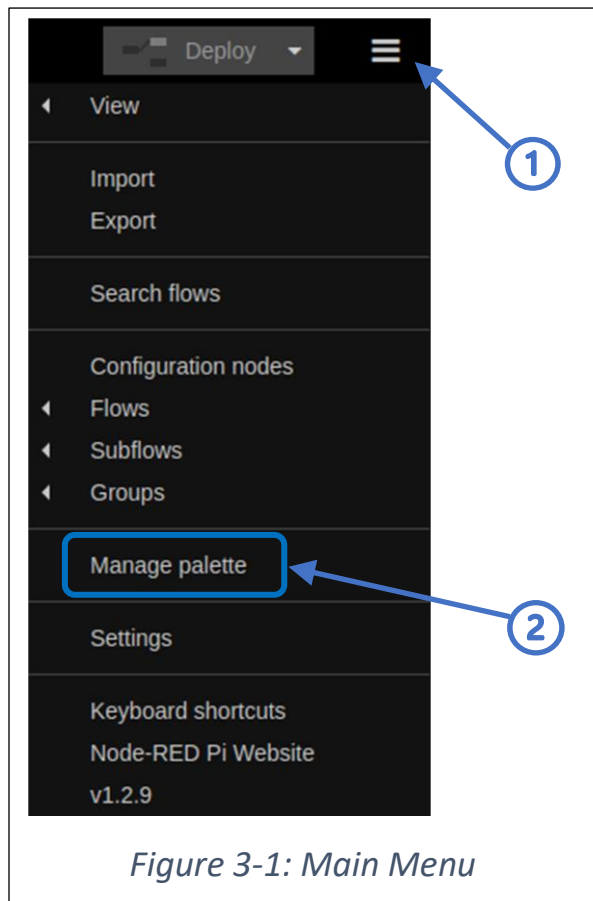
At this point you should have the Sequent Microsystems I/O Learning card installed on your Raspberry Pi. If not, go back and set that up.

**Note:** In this **Preliminary** version of the Tutorial you will not have received you I/O Learning card yet. When you need to use the pushbutton, we will show you how to proceed without the learning card pushbutton.

To make the task of writing a Node-RED programs for the I/O learning card Sequent Microsystems has developed special nodes for you to use. These nodes understand how the I/O Learning card works and make it easy for you to control its features. Now is the time to download them.

Proceed as follows. First, click the menu icon at the top left of your workspace (① in figure below). This will open the main menu. Now select "Manage Palette" from the dropdown (②).

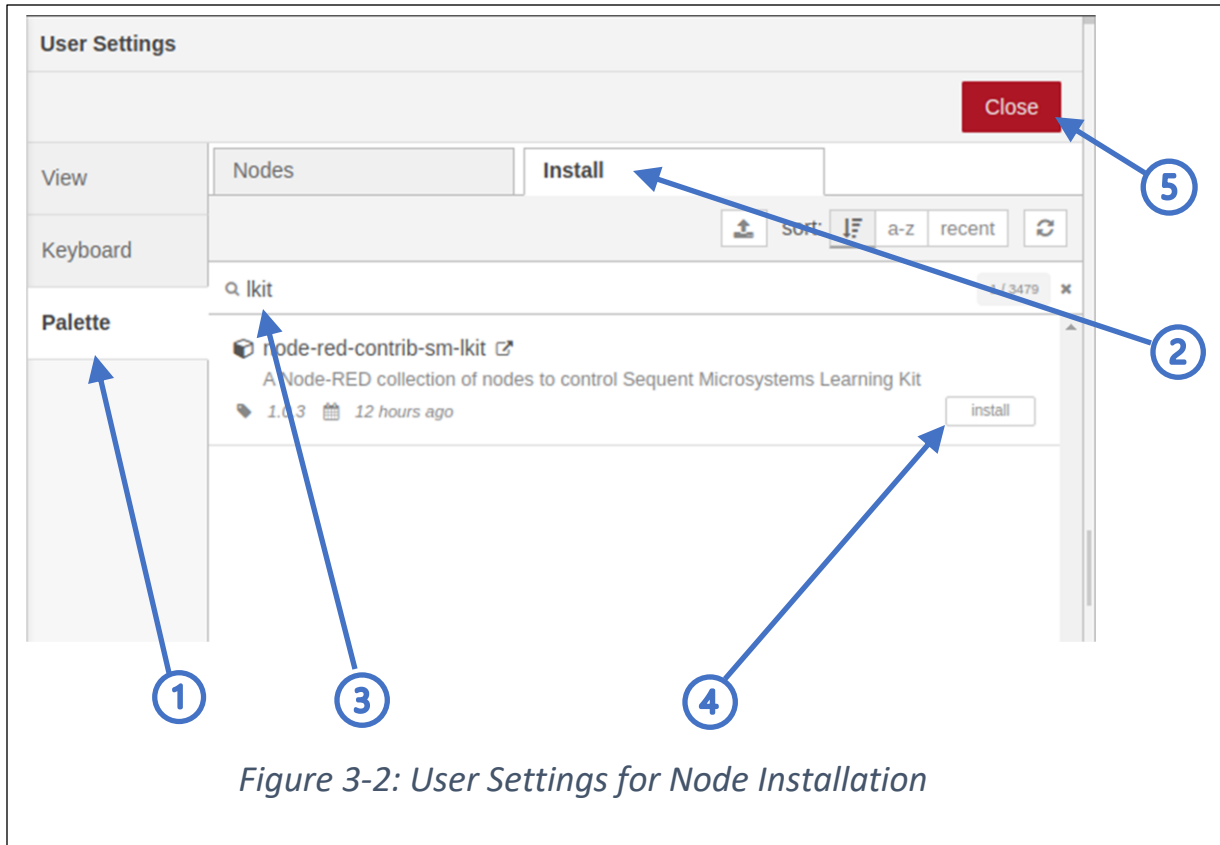
## Learning Kit Workbook (version 1.3)



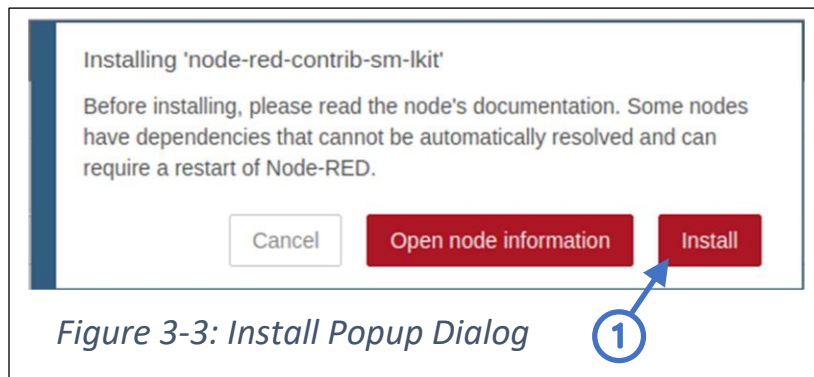
*Figure 3-1: Main Menu*

Once you have done this you will see a User Settings window, like the one below:

## Learning Kit Workbook (version 1.3)




Now, click the “Palette” tab on the left-hand side (① in figure). Then click the “Install” tab (②). Enter “lkit” in the search box (③). When you do this the window below the tab will become a buffet of nodes to pick from. In this case, you should see “node-red-contrib-sm-lkit”, which is a package of nodes for the I/O Learning card. Click the tiny “Install” button to the right of the description ④. This will cause a small dialog box to pop up showing that you are about to install the lkit nodes. Don’t pay any heed to the warning, it is for more complex installation. Instead, click “Install” ①.

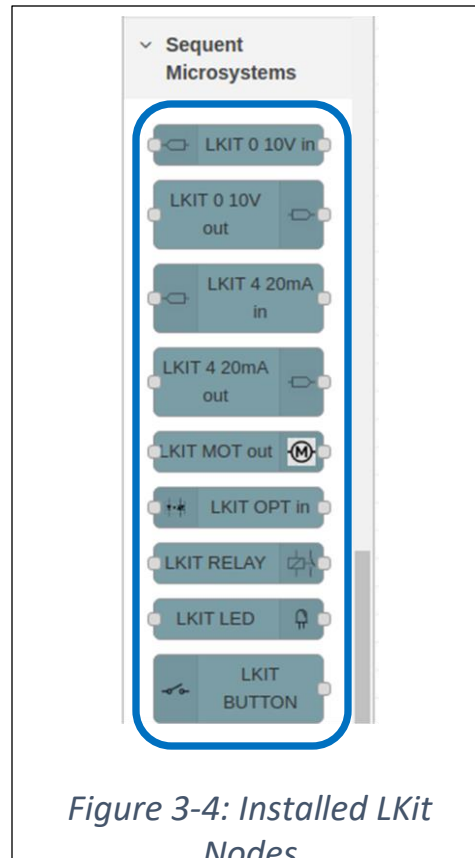





## Learning Kit Workbook (version 1.3)

Time will pass as the lkit nodes are installed. Take a break, have some coffee, take a nap, feed Mr. Nibbles. After a while the installation will be complete, and you can click the “Close” button .

To make sure that you have the nodes, scroll the palette down to the bottom and you should see a selection of nodes under the heading “Sequent Microsystems” as outlined in blue in the figure below.



Now you are ready to start using the I/O Learning card. While you are here, think for a minute about what you have just done. Node-RED has literally thousands of nodes that have been contributed by real Masters of the Programming Sub-Space from around the universe. When you enter a search term like “lkit” in the search box  you are asking for a list of all the contributions that match that term. Just remember, these are free contributions by the Node-RED community and because they are free you must always be aware that you get what you pay for. Set your expectations accordingly and always be careful about what you allow node you have downloaded to do. If a node starts asking for your bank account number and password, well that should be a warning sign, but then again it is amazing the risks people will take in the name of convenience.

### Working with the Pushbutton

Time to build a flow that works with the outside world, or at least something tangible on the I/O Learning card. The new guy on the block is going to be the pushbutton.

## Learning Kit Workbook (version 1.3)

First, find the pushbutton on your card. It is on the side opposite the big connector as shown below:

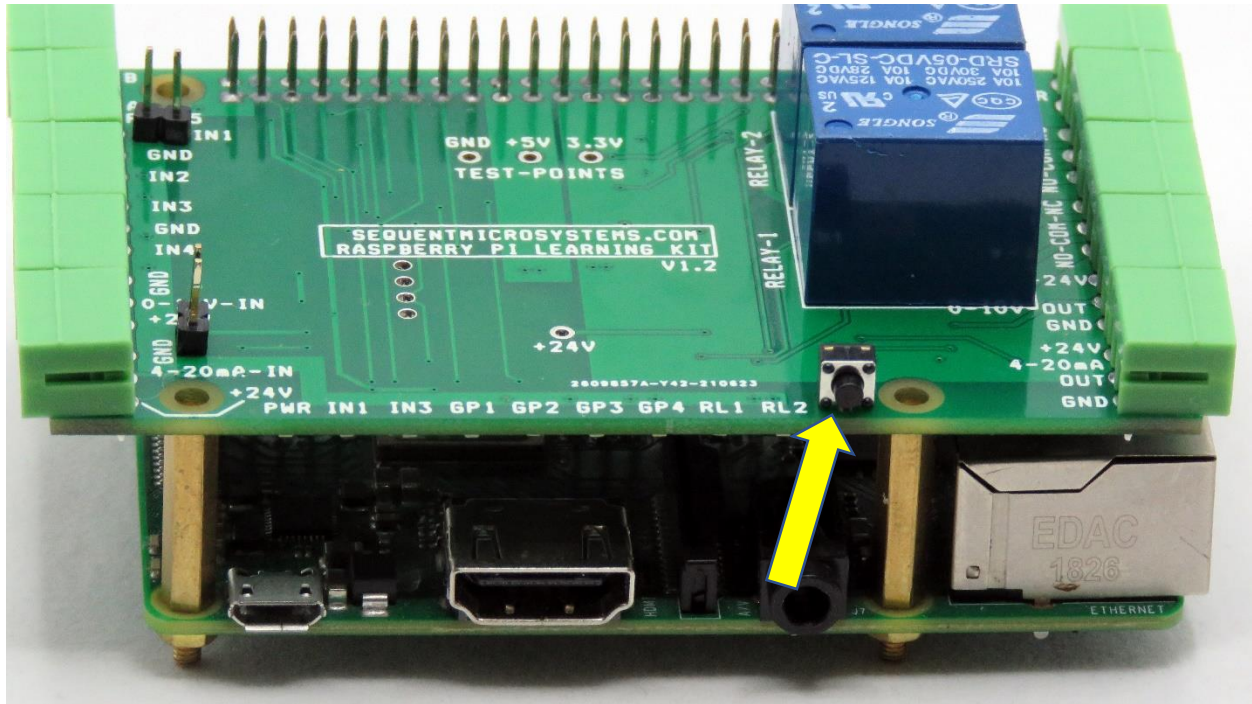


Figure 3-5: I/O Learning Card Pushbutton

The push button is that tiny little black dot on the edge (yellow arrow). It is no different than a doorbell button except that it is just very, very small.

Preliminary Version: In this preliminary version of the tutorial, we assume that you do not yet have your Sequent Microsystems I/O Learning card. Despair not! You can do everything in this preliminary tutorial by using a key on your keyboard as a substitute for the I/O Learning card pushbutton. Here's how:

Below you will be asked to load the LKit Button node. If you do not have the I/O learning card you should do every thing below, but when you are asked to down load the LKit Button node you should down load the LKit Key node instead. Here is what it looks like:



Figure 3-6: LKit Key Node

You can use this node exactly as you would the LKit Button node. The only difference is that instead of pushing the button on I/O Learning card you will push the grave/tilde key on the keyboard. It does not

## Learning Kit Workbook (version 1.3)

make any difference whether the shift key is pressed. Usually, this key is in the upper left corner of your keyboard.



Figure 3-7: Grave Accent and Tilde Key

You can use this key in the same way you would use the pushbutton in the examples below. We are using this specific key because it is probably the least used typewrite key; that's why it is so nice and shiny.

Instead of issuing messages by clicking the mouse on the Inject node, in this new flow you are going to issue them by pushing a button. Use your newly acquired knowledge of Node-RED to do the following:

- Bring in the Hello World Flow if it is not already on the screen. (if you forgot how to do this, see [\[redacted\]](#))
- Delete the Comment Node if there is one, you will document this flow using a different technique later (see [\[redacted\]](#))
- Deploy your Hello World Flow (see [\[redacted\]](#))
- Go to the debug screen (see [\[redacted\]](#))
- Click the Inject node and make sure that you see Hello World in the debug window (see [\[redacted\]](#))

Now that you are sure you have the right flow and that it works you can modify it using the following steps.

- Change the name of the flow on the tab to "Pushbutton Hello World". (see [\[redacted\]](#))
- Delete the Inject Node from the flow (see [\[redacted\]](#))
- Copy the LKit button node from the palette to your flow (see [\[redacted\]](#))



Figure 3-8: LKit Pushbutton Node

- Connect the pushbutton to the inject node (see [\[redacted\]](#)).
- Deploy your flow!

After you deploy your flow, you should see something similar to this in your workspace:

## Learning Kit Workbook (version 1.3)

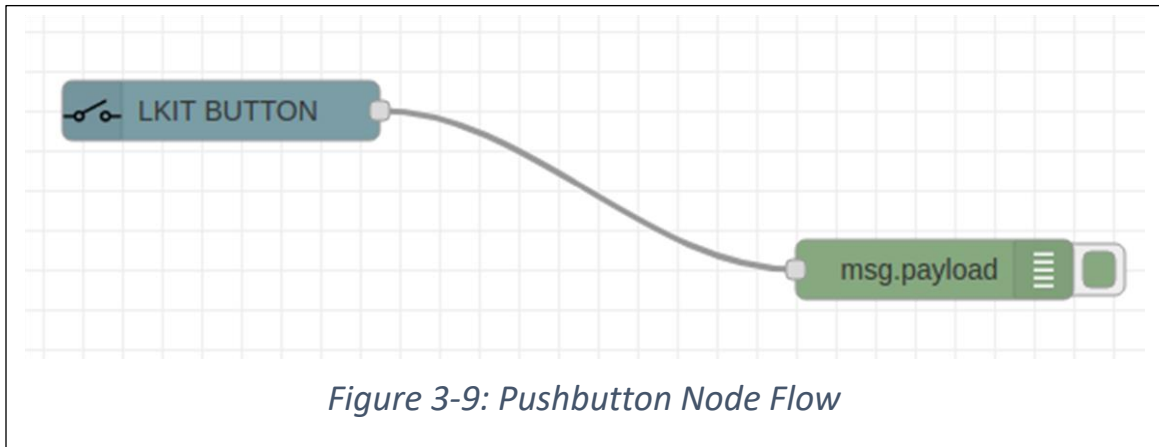


Figure 3-9: Pushbutton Node Flow

Open the debugging window (the little bug icon) and clear the debug window (the little trash can icon). And now for the big moment... while you are watching the debug window push the button and release it.

TA-DA! If everything worked right, then you should see a little set of messages like this:

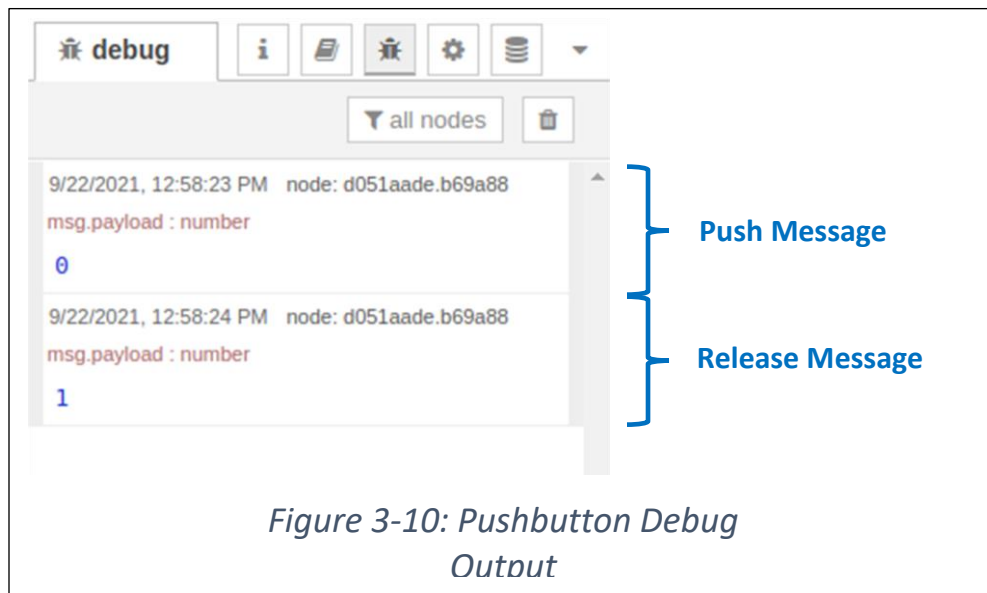


Figure 3-10: Pushbutton Debug Output

What happened? Is it what you expected? Every time you push and release the button you will get two messages. Try it again, but this time push the button and hold it down. Look at the message. It is telling you that the pushbutton node is sending message with a single number zero in the payload.

Your finger is getting tired, so release the pushbutton. Bingo! You get another message, but this time the payload has the number one in it.

## Learning Kit Workbook (version 1.3)

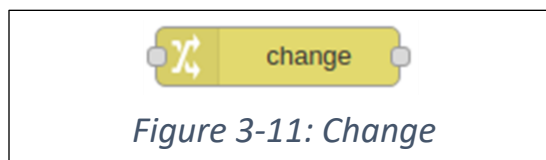
Try this a few times. Each push and release of the button should give you two new messages, one for the push and one for the release. In the case of the push button the 0 indicates the button has been pushed in and the 1 indicates that the button has been released. Maybe you think it should be the other way around and a push should cause a one and a release should cause a zero. If you are curious about why the pushbutton behaves the way it, you should go and read Card Electronics: Pushbutton section in the Appendix.

Hopefully, at this point everything is behaving as described above. You can also see the basic idea of Node-RED in this flow: Input nodes generate messages when something happens and send them to other nodes that act on them. All the pushbutton node is doing is sending a message every time the button is pushed or released. All the Debug node is doing is displaying the payload of the message it receives (along with the time and sending node). Simple!

The next step: let's see if we can get the button pushes to put up the "Hello World" message again. Specifically, we want the "Hello World" text to appear in the debug window only when we push the pushbutton, but not when we release it. Let's see if we can sneak up on the solution one step at a time (most of engineering is incremental, working from something that functions in a known manner toward something that functions the way you want it to).

Remember when we constructed the Hello World flow? We had the Inject node send the text "Hello World" in the payload and the Debug node dutifully displayed the payload. In our pushbutton flow, however, the Pushbutton node is only sending zero and one. What you might like to do is change the message so that the payload says something more interesting. Like "Hello World". In some nodes you can change the payload that is generated, but in the Pushbutton node is stubborn node and only wants to talk in ones and zeros. What are you going to do?

Time to go to the bench and bring another player into the game... the Change node! This node allows you to modify the payload of a message on the fly. A message comes in one side and the Change node checks the payload according to the rules you give it and pumps out a message on the output with a new payload. This is a very powerful concept, so pay attention.



## Learning Kit Workbook (version 1.3)

Go to the palette and under the “Function” category drag a Change node in and drop it right on top of the line connecting the Pushbutton node to the Debug Node. (Before you do this you might need to spread the Pushbutton node and the Debug Node apart so that the change node will fit in between). When you are done your workspace should something look like this.

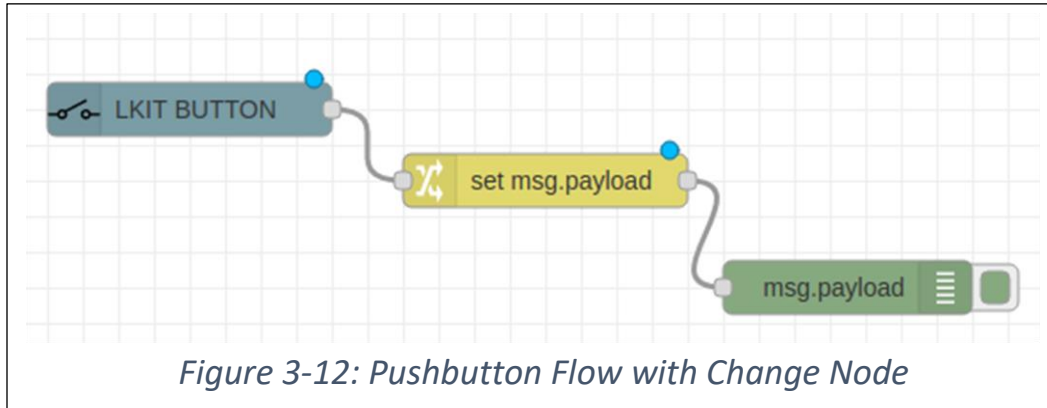


Figure 3-12: Pushbutton Flow with Change Node

The change node is going to inspect every arriving message and, if necessary, make changes that you specify to the message. Deploy your flow and give it a try by pushing and releasing the pushbutton. Look at the debug window. What do you get? Does it look like the messages below? Why? What do you think is happening? Yes, there is not much there. Look very closely at the debug messages next to the “message payload :” which indicates the “type” of the payload, in this case “string[0]”, which means that the message is a string with of length zero, which means the message is empty. Also look at the part of the message circled in blue. Just two quotes (“”). This is another indication that your message contains no characters.

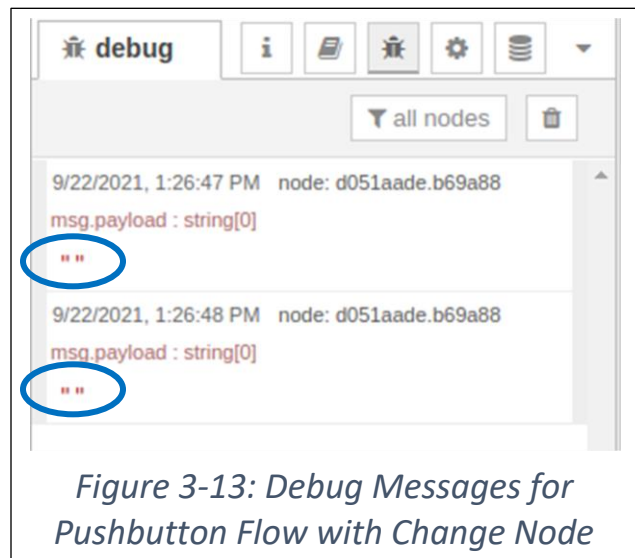




Figure 3-13: Debug Messages for Pushbutton Flow with Change Node

Whoa! Say what... “types”, “strings”, double quotes! What is all this stuff anyway? Don’t panic (yet). If you have done any work with programming languages then you will already have an idea of what a

## Learning Kit Workbook (version 1.3)

“type” is, what a string is and how string and number types differ from each other. If you have no idea what is going on with types, strings and numbers then read the Appendix on Types.

Briefly, a message payload contains information, sorry, you already knew that. Just as in the everyday world information comes in different forms. Think about this: someone tells you that they live at 924 Hamster Avenue and that there are 924 pellets left in the bag of premium pet chow. You know intuitively that 924 in a street address is different from the count of 924 pellets. The number 924 looks the same when you say it or write it out, but you know in your heart of hearts that they are not the same thing. For example, if you drop another pellet in the bag then you know that the bag now contains 925 pellets. However, you can't figure out the address of the next house on the street by adding one to 924. House number and counting number are two different things and cannot be handled in the same way. In the world of computer programming we say that they are of two different “types”. In the Node-RED examples here if a number is shown as 924 it is a number if it is surrounded by quotes, like this: “924”, then it is a string. Numbers and strings are just two of many “types” that are available in Node-RED (and many other programming languages).

A string is simply a sequence of characters, just like a strip of paper with characters written on it. In the example debug message in the figure above, the quotes indicate that the message payload is a string. The fact that there is nothing between the quotes means that the payload string has no characters, it is a string with 0 characters, in the real world a blank strip of paper. Go back to figure  from your first flow. Look carefully just above the payload and you will see string[12], which indicates that the message is a string of 12 characters. Now, look at the payload. It says, “Hello World!”. Count the characters between the quotes and you will find that there are twelve of them. Compare this to the message in figure  above.

For now, all you need to know is this: Node-RED data has different types, and you need to be careful how you handle each type. More details later.

## Learning Kit Workbook (version 1.3)

Back down to work. Sometimes it helps if you clear the debug window (click the little trash can icon) so that you can see each message on a clean sheet of pixels, so to speak. At this point we do not have exactly what we wanted. Now every time you push and release the button you get two “empty” messages, just “”, which indicates a message with no contents. It sure looks like the change node is taking each message and replacing the payload with nothing. Not all that helpful. What we really want is for the change node to replace the zero in the message it gets from push button node with “Hello World”. So, let’s open up the change node by double clicking on it. Presto!

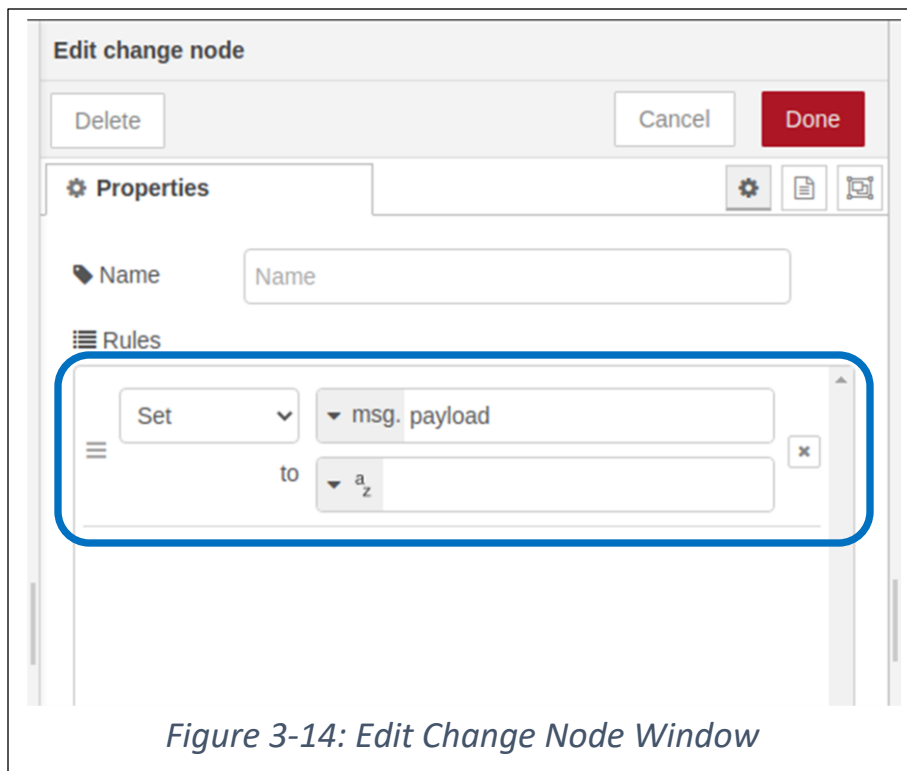


Figure 3-14: Edit Change Node Window

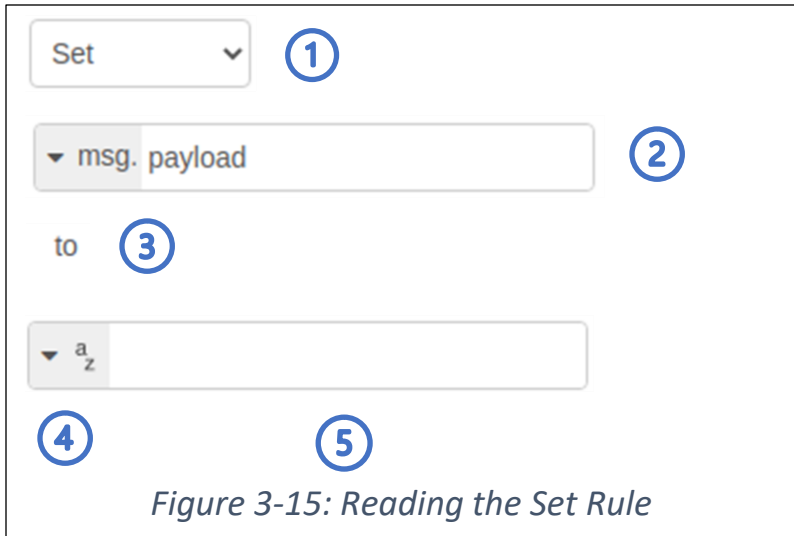
What you see here is a table with a single “rule” (see blue outline). A “rule” is just the description of how you want to modify the incoming message. As you will see there are different types of rules, and you can have as many rules as you want applied to the incoming message<sup>2</sup>. The “default” rule, that is the rule you get when you first drop the change node into your flow, is very simple. The figure below breaks out each part of the default rule”

---

<sup>2</sup> But be on your toes. When there are rules in a chain they are applied one at a time starting at the top of the table. If a rule changes a part of the message payload the next rule will act on the changed information, which may not be what you had in mind.



## Learning Kit Workbook (version 1.3)

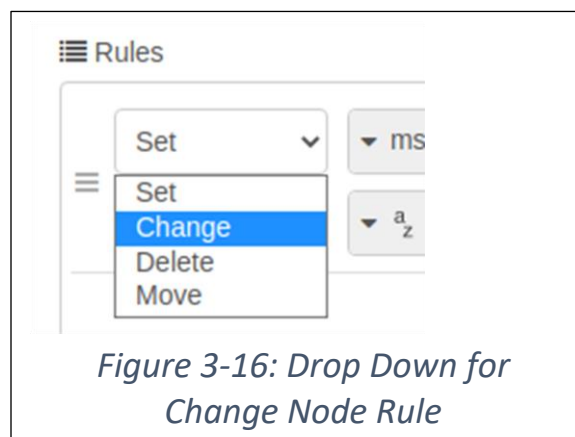


Here is what the rule says, step by step: ① “set” ② “the payload of the message” ③ “to” ④ “a new value whose type is a string” ⑤ “with the value of nothing”. In other words, whatever is in the payload of the message replace it with a string with no characters. For every message received this rule modifies the message and sends via the output port to the next node. Now, go back to the debug output for this flow (figure [\[redacted\]](#)) and see if you can explain the output you got.

What do you think you might do to get the change node to convert an incoming message with a payload of zero to an outgoing message with the string “Hello World” in it? Go ahead poke at the rule and the other features on the change node settings. You might make a mess of things, but you can always cancel the editing, or if worse comes to worse, you can delete the change node and start with a fresh change node. Sometimes, but not always, a fast way to learn about something is to poke at it with a small stick. Of course, you will want to use good engineering judgement. Poking at a sleeping hamster is okay, a sleeping cougar, not so much.

Enough poking, let’s try this recipe.

- Click on the little drop down that says “Set” [see figure at right]. A little menu will drop down saying “Set”, “Change”, “Delete” and “Move”.
- Click on “Change” because that certainly sounds like what you want to do, namely, change the message. You should get a new rule block that looks like that in figure [\[redacted\]](#).



## Learning Kit Workbook (version 1.3)

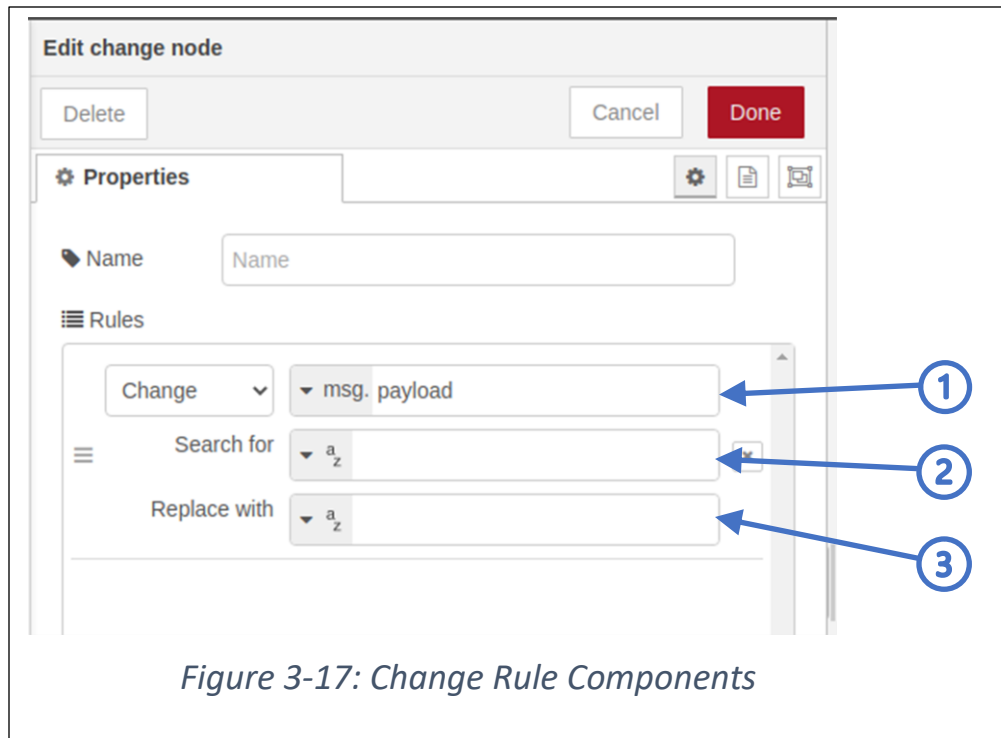
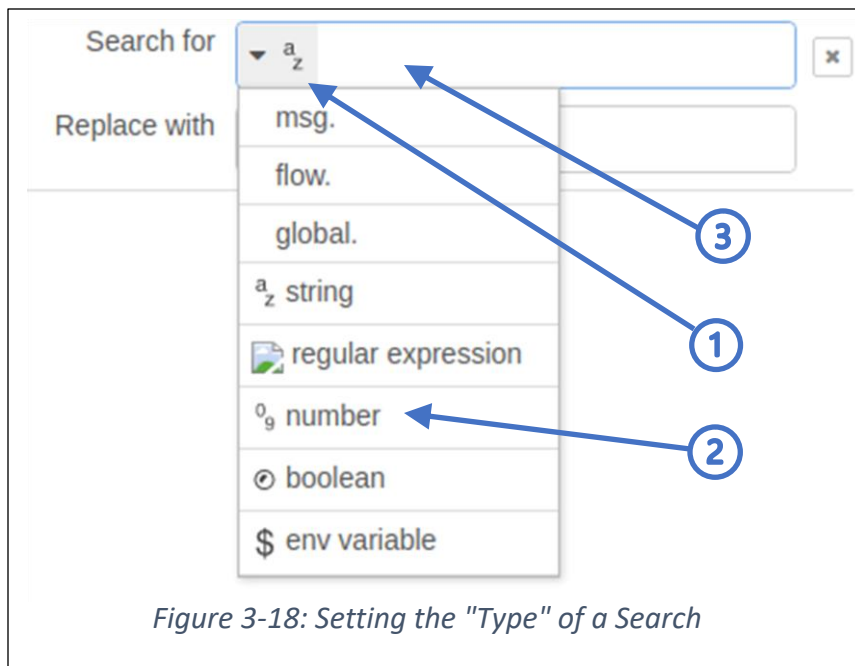


Figure 3-17: Change Rule Components

The rule just says, “**①** change the payload in the message in two steps: **②** search for something in the payload and, **③** replace it with something new”. When this process is done the Change node sends the new message to the node connected to its output port. What we want to do is to replace the number zero in the message with the message “Hello World”.

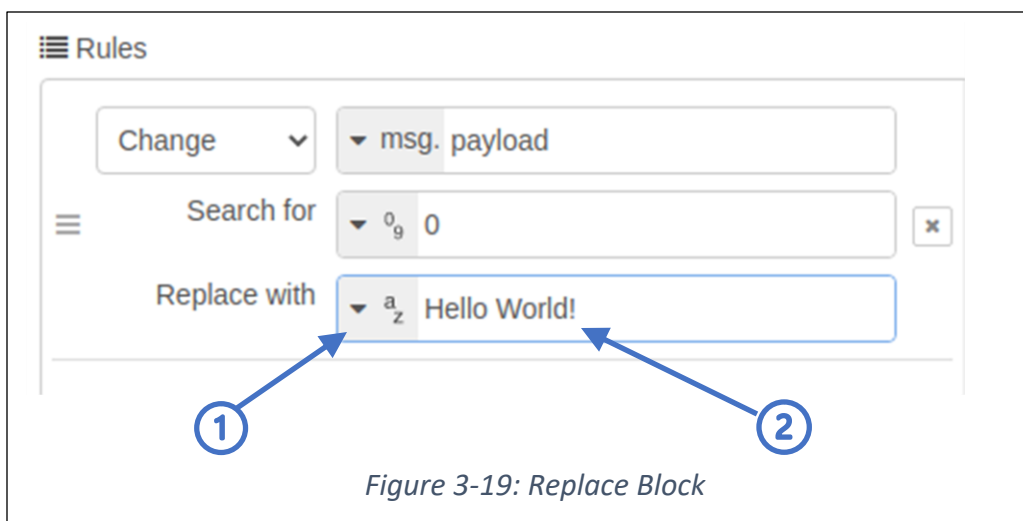
Now, back to the Change node. The payload of the incoming message is a number, it is either a one or a zero. If you look at incoming message, as we did above with the debug node (see fig. \_\_\_\_), you will see that when you push the button a zero (0) appears in the output. It does not have quotes around it, so you know for sure that its type is a number not a string and right above the 0 it says it's a number. So, in our rule we want to search for the number 0 and not the string “0”. To do this click the type dropdown (see **①** in figure \_\_\_\_ below) and you should see this:

## Learning Kit Workbook (version 1.3)



From the choices click on number (see ②) The little 0 and 9 digits indicate that you are going to search for a number type (squint, they really are a 0 and a 1). Now go to the box to the right of the type dropdown and ③ enter 0, which is the value of the payload you want to look for because it is the value the message contains when the button is pushed.

Next, let's set up the "replace with" value. We want to replace the number 0 in the payload with the string "Hello World". The type of the replacement is already a string. You can tell this because the type dropdown shows (see ① in figure \_\_\_), the microscopic a and z characters indicate that the type is a string, just like the 0 and 1 characters indicated the type in the "search for" block is a number. In the "Replace with" block ③ type in Hello World!. it is not necessary to surround Hello World! with quotes

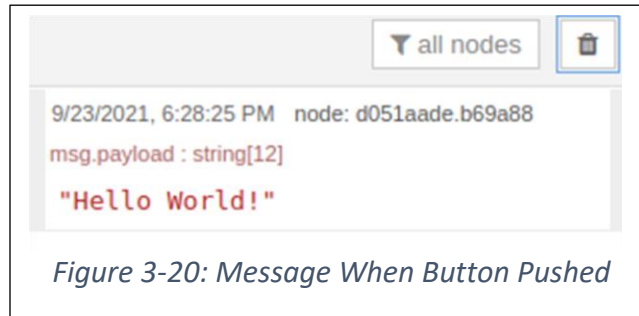


## Learning Kit Workbook (version 1.3)

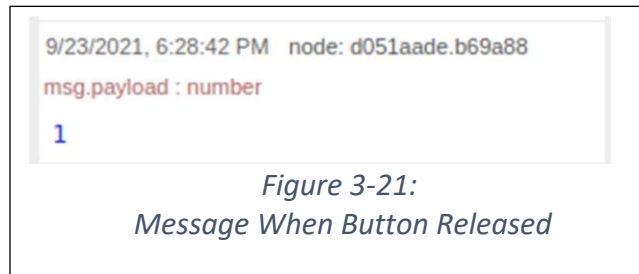
as you might do in other programming languages. In NodeRED the type dropdown ① indicates that Hello World is a string. This will be very clear in a moment. Finish up by clicking DONE.

Deploy it!

Now press the button and hold it down. What do you see? If everything is working, then you should see this:

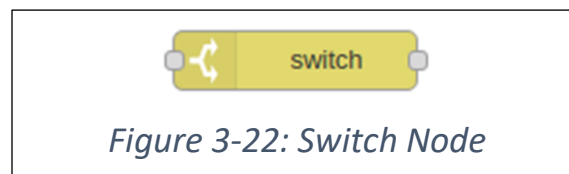


Looking pretty good! By now your finger is getting tired, so release the button. Whoa! What happened? It's another message. What is it where did it come from??? Look at the debug window closely you will see the message looks something like this:



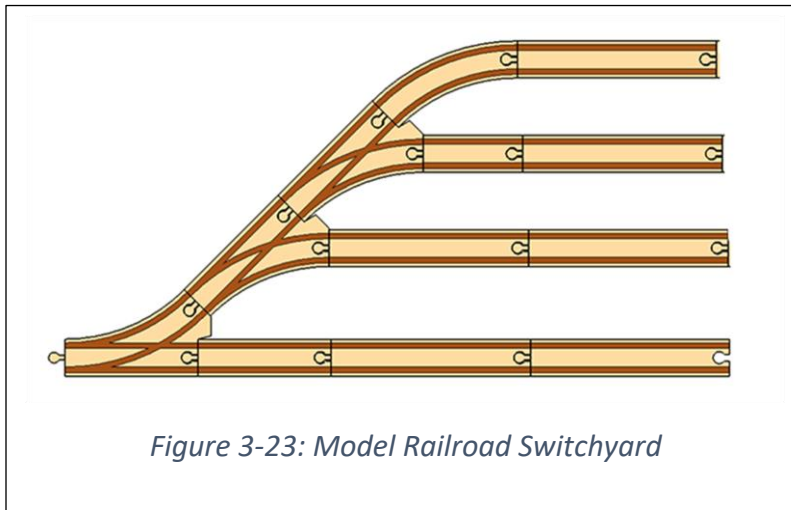
If you think about this for a minute you will realize that the number one is a message from the pushbutton that was passed through the change node without being affected. This is important to remember when you use a change node: any message that you do not explicitly change is passed to the output unchanged.

Suppose that you really don't want to see the message that occurs when you release the button. Happily, there is a way to destroy a message. You can do this by using the Switch Node.



## Learning Kit Workbook (version 1.3)

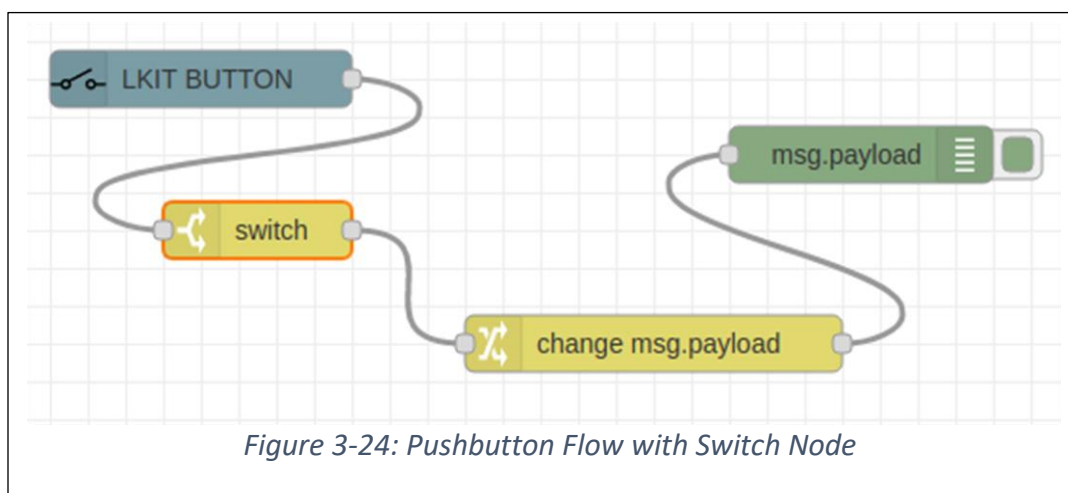
By now it might have occurred to you that Node-RED is like a little model railroad where small trains carry messages from one place to another. The nodes are stations, and the wires are the tracks. Wouldn't it be nice if you could switch the messages from one track to the next based on the contents



of the message? Well that is what the Switch node is for. It allows you to switch messages at the input port of a node to one of several output ports just like this model railroad switch yard.

The switch node is just like the switch in a train yard; it allows you to direct a message to different places based on the contents of the message. Let's see if the switch node is up to the task.

Drag the switch node from the palette and drop it right between the pushbutton node and the change node. If there is enough space it should connect itself up to both nodes. If not, you may need to delete the connection between the pushbutton node and then connect everything back up. When you are done it should look something like This:



Look closely at the Switch Node. It only has one output port, so it is not quite like the train yard shown in figure [3-23](#), but we are going to fix that in a moment.

## Learning Kit Workbook (version 1.3)

Now, you must tell the switch node what to do with incoming messages. In our flow the switch node receives one of two incoming messages: a message containing the number zero when the button is pushed and a message containing the number one when the button is released. Start by defining the first rule. Open the message node by double clicking and you will see this:

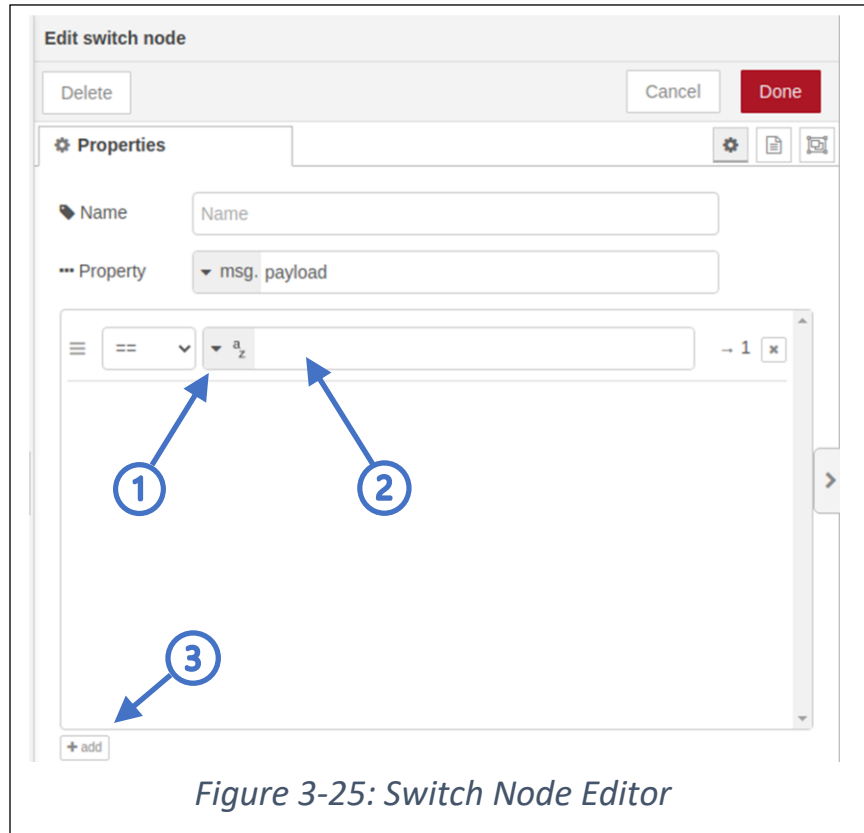


Figure 3-25: Switch Node Editor

There is one rule which is going to be activated when the message payload is equal to something, but what? Well, right now nothing, but you can change this. First ①, change the type in the rule from a string to a number as you did before ([reference to previous change of type](#)). Now, ② enter the number zero in the block next to the type. The definition should look like this:

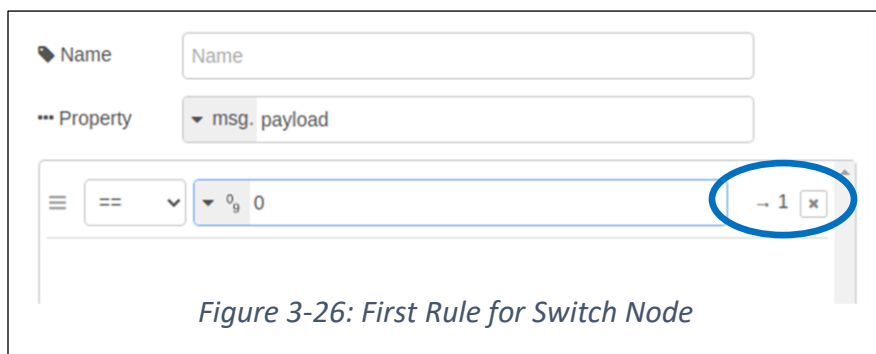


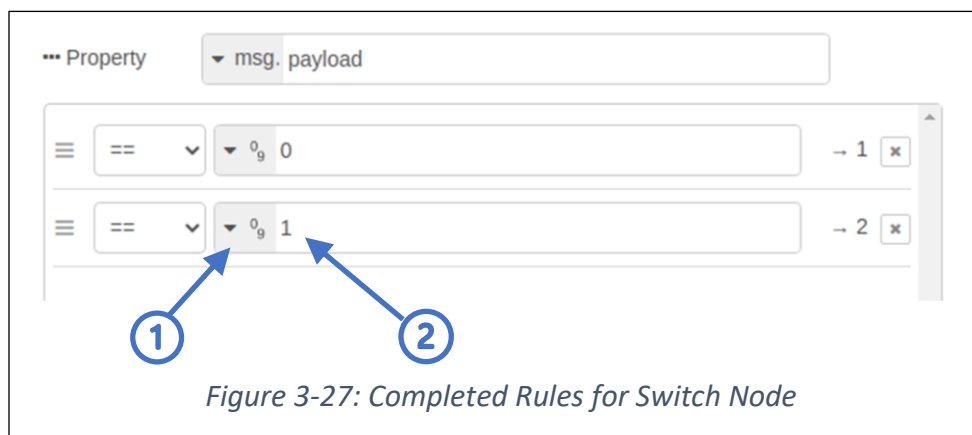
Figure 3-26: First Rule for Switch Node

## Learning Kit Workbook (version 1.3)

Stop for a moment and look at the rule. Look especially at the right side of the rule where there is a little arrow pointing to the number 1 (circled in blue). This is telling you that when a message comes in that has a payload with only the number zero in it the message will be sent to output port number 1. Right now, there is only one output, but we are going to change that by adding another rule.

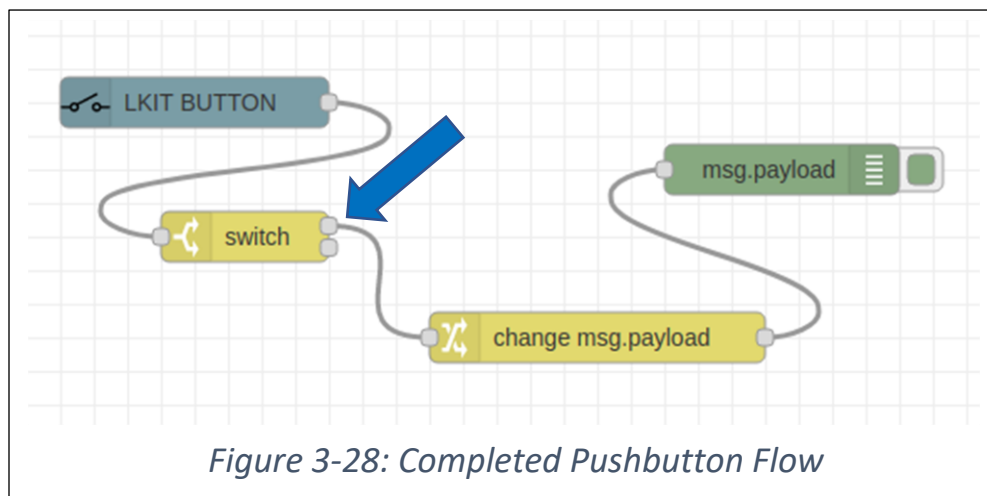
To add a new rule, click the little add button at the bottom of the rule window (③ in figure above). This will add another undefined rule block, but this time the rule is pointing to the number 2, which means that if the rule matches the message it will go to output port 2.

Do the same thing as you did before: ① change the type selection from string to number and then ② put the number 1 in the value window. When you are done the list of rules should look like this:



If you are satisfied that your rules are like the rules above, then click the DONE button.

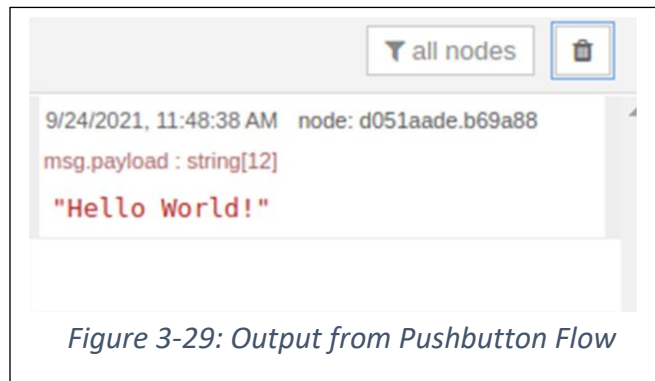
Now look at the switch node. Does it look different? If you did everything correctly, then the switch node will now have two output ports on the right side and your flow will look like this:



## Learning Kit Workbook (version 1.3)

Notice that the change node is connected to the top output bubble representing the original port (check out the blue arrow). The ports on switch nodes are numbered from top to bottom starting at 1, so the change node is connected to the number 1 port of the switch node. Remember that this where incoming messages are going to go if they have numeric value of 0.

Deploy it! Try it! Go ahead push the button and release it. Is it doing what you would expect? If you did everything correctly the debug output should look like this:



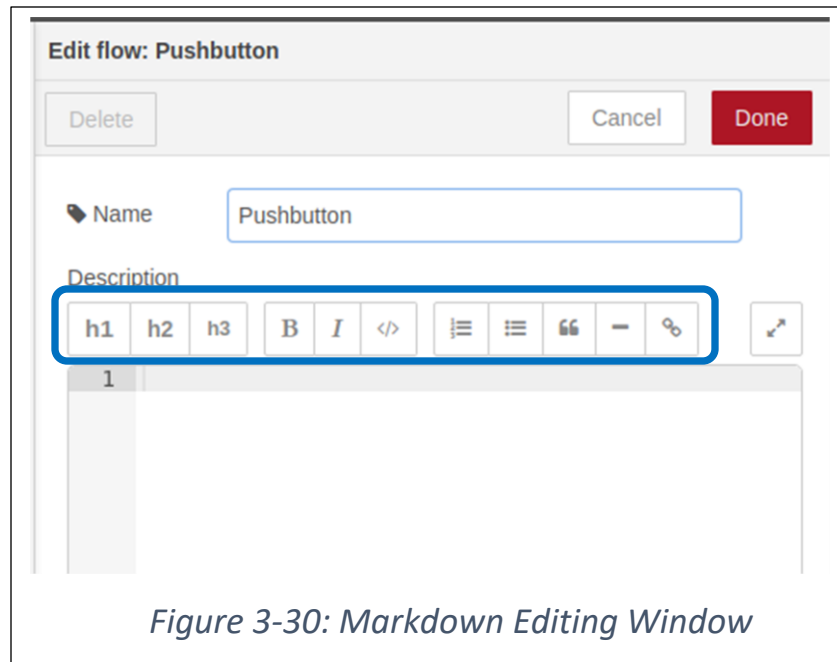
A Hello World message is only displayed when you push the button. Take a moment and see if you can explain why the flow behaves this way. What happened to the message from the pushbutton that was generated when the button was released? That's right, it went to terminal number 2 of the switch node and because nothing is connected to that terminal it is gone forever. You can check this out if you want to by hooking up a debug node to the second terminal. Push and release the pushbutton and see what happens. Was it what you expected?

At long last! The Hello World! flow that you wanted where pushing the button generates only one Hello World! message and nothing else. Hallelujah! Now for the part that separates real engineers from engineer wanna-bes... the documentation. Rather than cover this beautiful flow with comment nodes let's put in a description that is part of the flow.

Start off by double clicking on the tab at the top of your flow. This will open an editing window for your flow.



## Learning Kit Workbook (version 1.3)



*Figure 3-30: Markdown Editing Window*

You have seen this before when you were working with the comment node. Everything is the same, it is just a simple Markdown editor. With it you can define different headers, bold text, underlined text, bullet lists and so forth. In the Markdown editor you can either enter special characters to control formatting or you can use the formatting options shown in the blue box. For example, h1, h2 and h3 will set up formatting of headers at different levels. B and I will format your text as bold or italic. Below is example documentation for the Pushbutton Flow.

## Learning Kit Workbook (version 1.3)

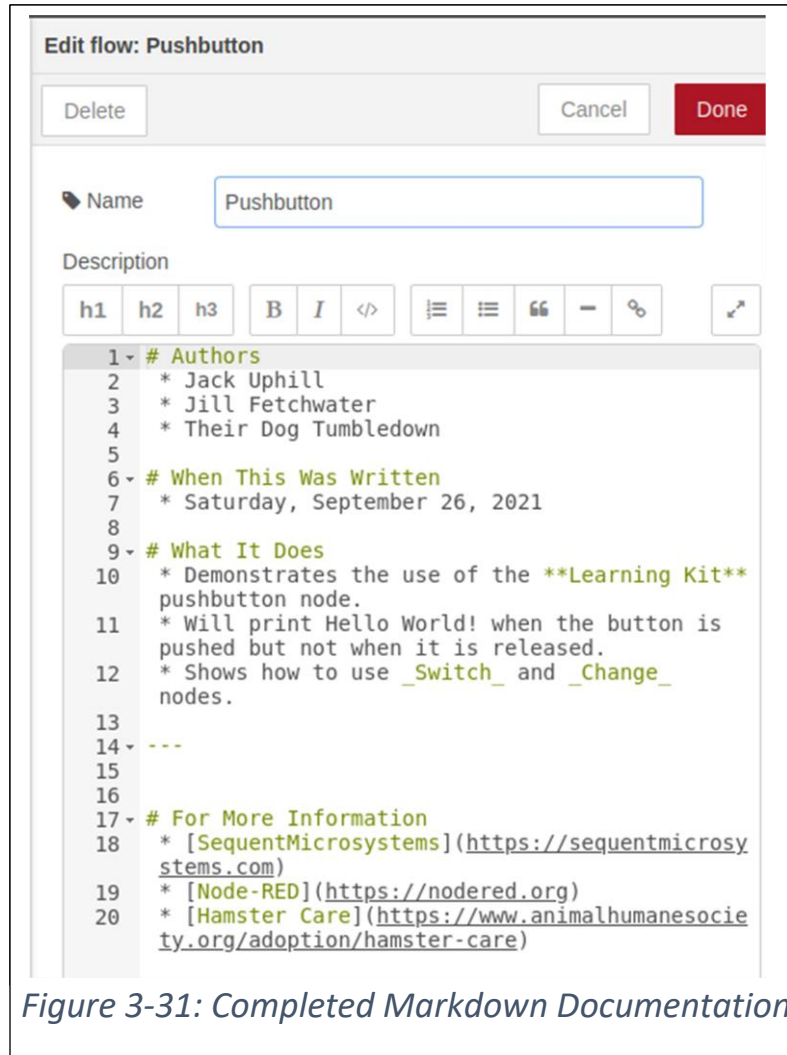
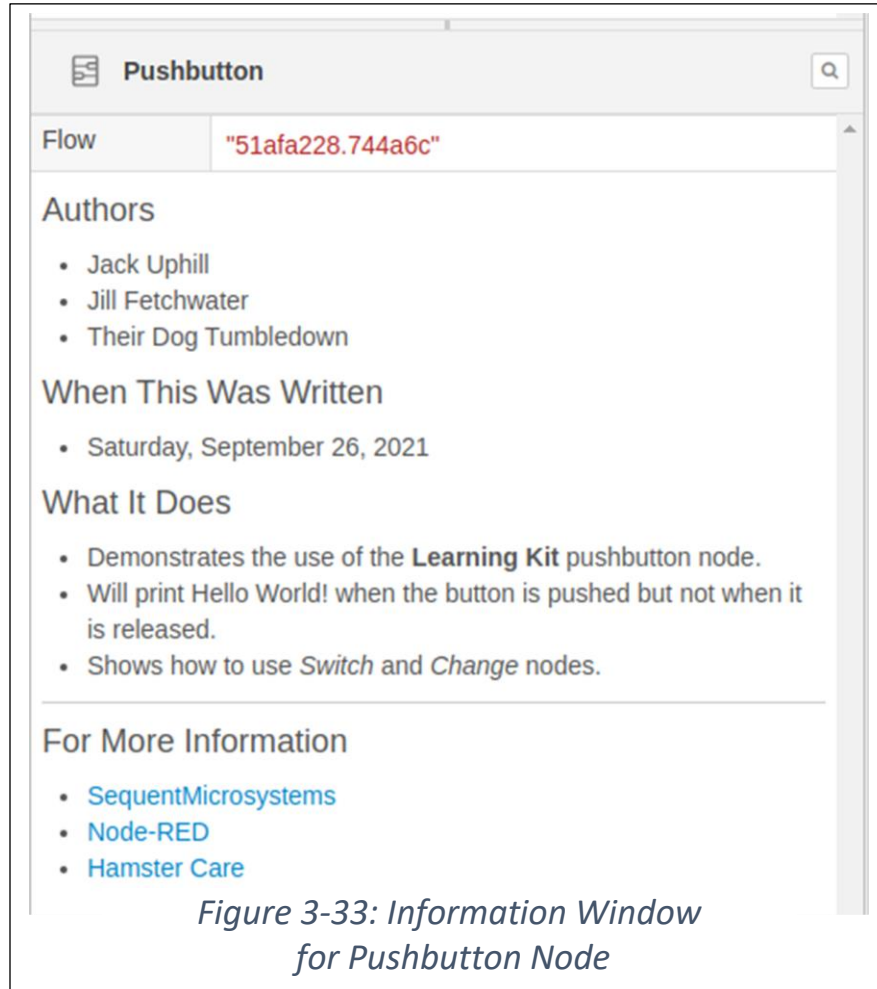


Figure 3-31: Completed Markdown Documentation

In the Markdown editor are a number of strange symbols, like #, ---, \* and so on. You can either type these in with your text or use the option buttons (h1, h2, ....) at the top of the window to format your text. What you type will remain in markdown format until you hit the DONE button. Then it will be included in the pushbutton flow as part of the information for the flow. To see the results, click on the information tab (I) in the sidebar. Beautiful! Now you can book your cruise on the SS Mesocricetus knowing that your poor colleagues back home have some idea about what is going on with your flow.

## Learning Kit Workbook (version 1.3)



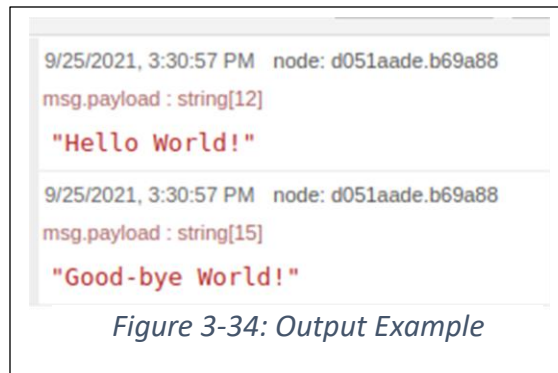
### Puzzles

Now that you have some experience see if you can solve these puzzles using what you have learned.

**Puzzle 1:** Can you think of a way to get the same behavior as in the example above, but by reversing the position of the change node and the switch node? In other words, the output of the push button node is going to go first to the change node and then the output of the change node is going to go to the switch node. Finally, one of the outputs of the switch node is going to go to the debug node. If you can get this to work, you will have demonstrated to yourself that there is more than one way to solve the same problem in Node-RED.

**Puzzle 2:** Can you set up a flow using so that World is printed when you push the button and Goodbye World is printed when you release the node? Your output should look something like this as you push and release the pushbutton:

## Learning Kit Workbook (version 1.3)



Do this only using Pushbutton, Change, Switch and Debug nodes, but you can make as many copies of each node type as you need. As above there are probably many different solutions. See if you can find one or more of them.

**Puzzle 3:** In section [\[redacted\]](#) you used the Inject node to create a message when you click on it. Open the inject node edit window (i.e. double click on an inject node). Look at the bottom and you will see some other options that you can use to configure the inject node. Using just the inject node and a debug node (like figure [\[redacted\]](#)) see if you can figure out how to make the inject node act on its own? Can you think of some way to use these features? Could you build a flow that would print the local time every minute in the debug window?

For example, could you use an inject node and debug node to create an alarm clock that printed “Rise and Shine Buttercup!” every morning at 7:30 AM? Could you think of a way to make a simple sprinkler timer that would water your favorite Ficus plant for 10 minutes every Wednesday (well, it is just going to print “water on” and “water off” 10 minutes apart on Wednesdays at 11:45 AM). Try making a simple flow that will wish you a Happy Birthday in the debug window at noon on your birthday.

**Puzzle 4:** Investigation for the Curious: Whenever you plop a node down in your workspace it is already configured for some behavior. Technically we call this the “default configuration”. In this chapter you have worked with two new nodes: the change node and the switch node. When you first brought them in to the workspace they were already set up for a basic operation. Strangely, there is no clear documentation (for shame) about what the default behavior is. However, if you are the curious type maybe you can figure out what the default behavior is. You have all the tools you need to do this, namely the Inject node and the Debug node. You can use the inject node to send different messages to a node and use the debug node see what sort of messages it generates (or maybe it does nothing!).

Load an inject node, Change node and Debug node into a fresh workspace. Open up the Change node and look at the default rule (you did this before here [\[redacted\]](#)). Try sending different messages to the Change node from the Inject node. What happened? Can you see a pattern? How would you describe the behavior of the change node in its default state?

Suppose you opened the Change node and deleted the default rule so that there are now no rules defined. How do you think it will behave? Try sending different messages to the Change node. What happened? Was it what you expected?

## Learning Kit Workbook (version 1.3)

Messy, messy, messy. Sometimes in engineering when you do not have all the information you need you must just roll up your sleeves and test the node, device, component or whatever in an organized way to find out what it does. Caution here... if the default behavior of something, like the Change node is not documented then that means it might change in the future. As a general rule you rely on undocumented behaviors at your peril.

**Puzzle 5:** Investigate the Switch Node. In particular when a switch node has only one output how does it handle various types of messages. What is the default behavior? Suppose you only define one message, what happens to all the other messages the switch node receives that are different? Can you give a clear description of what is happening?

**Puzzle 6:** Markdown for Dummies: The descriptions of nodes and flows in Node-RED are defined using the mini-Markdown editor. Markdown is a very simple language for defining text formatting and is quite popular for lightweight documentation. Open up the Markdown editor for flow (double click on the flow tab, remember?). Play with the editor. Try each of the formatting functions and see if you can understand what they are doing. Do you think you could write your class next essay in Markdown and print out the formatted results? You might need to get a Markdown viewer to do this. Try some simple examples in Markdown. Do you think it would be worth your time to learn Markdown?

**Puzzle 7:** Mr. Nibbles is eating way too much hamster chow. It is going to take a few more weeks to save up or an I/O Learning card, but you would like to work with this workbook now. Can you think of a way to replace the switch with two Inject nodes, so that when you click one it sends the number zero and when you click the other it sends the number one to the same place. Replace the Learning Kit pushbutton node with your new arrangement. Does it work? If you got it to work then you can build all the flows in this preliminary workbook.

### Walking – Pushbutton Says “Hello World!” Out Loud

#### The Node-RED Console

Time to stand up a walk! So far you have managed to do some simple things in the Node-RED IDE. The IDE is powerful, but it is just an editing and debugging tool. Once you deploy a flow you don't need the IDE anymore because the flow is operating directly on your Raspberry Pi. In fact, you can close the IDE and send debug results directly to the Node-RED console window. Later you will learn about fancy ways to display your data, but for now let's see if we can get by with just the Node-RED console.

Wait a minute, what Node-RED console? Where is it? What are we talking about here? Go waaaaaaay back and look at how you started up Node-RED (see section [\[redacted\]](#)). The Node-RED console has probably been sitting unobtrusively beneath your Node-RED IDE all along.

## Learning Kit Workbook (version 1.3)

If you are running the Node-RED IDE now look up at the top of your Raspberry Pi screen where you will find a list of the open windows. One of them will say “Node-RED Console”.

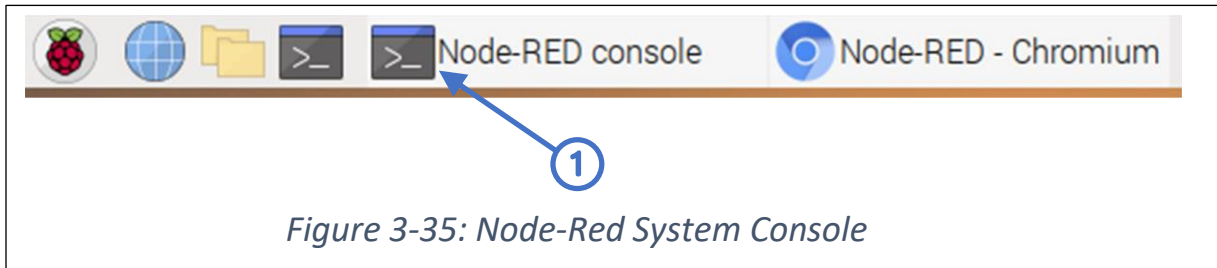


Figure 3-35: Node-Red System Console

Click on it and Raspberry Pi will bring it to the top of the windows heap on your screen.

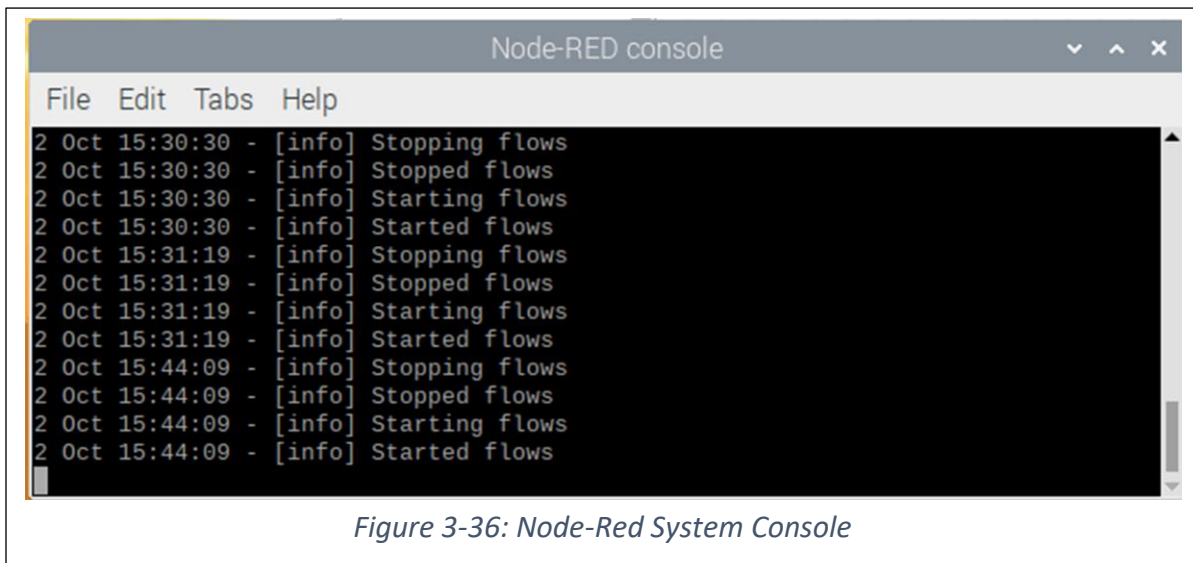


Figure 3-36: Node-Red System Console

The Node-RED console shows high level messages from Node-RED, like when a flow is started or stopped, which typically occurs when you deploy a flow, because you are stopping the previous flow and starting a new one. All of these messages are continually collected in the Node-RED console window, and you can scroll up and down through them. What is really happening here is that certain events that occur during the operation of your flows are printed out here. This process is called “logging” and is a very powerful debugging tool that we will discuss later. For the time being think of it as a little “flight recorder” or “black box<sup>3</sup>” that is keeping track what is happening in your flow. If you flow crashes and burns you can go back to the Node-RED console and review the messages and maybe find out what happened. Now we are going to use it to print out “Hello World!” when you press the pushbutton.

<sup>3</sup> Yes, yes, we all know that the black boxes on airplanes are really orange.

## Learning Kit Workbook (version 1.3)

Do this: go to your Pushbutton flow (or bring it back by importing it from where you saved it. You did save it didn't you?) Push the button and make sure the flow still shows "Hello World!" in the debug sidebar every time you push the little button.

Now pop open the Debug node (double click on it) and look at the options:

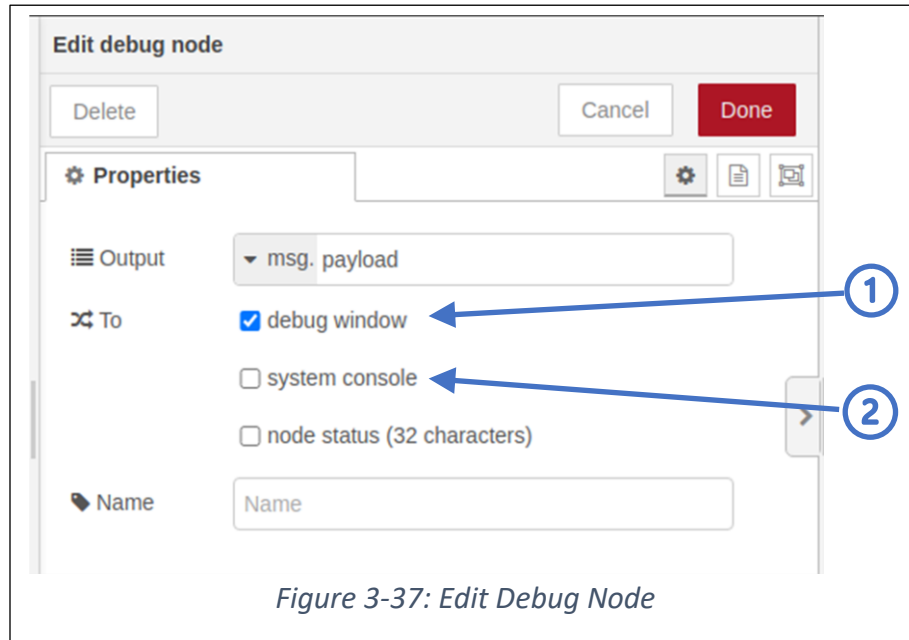
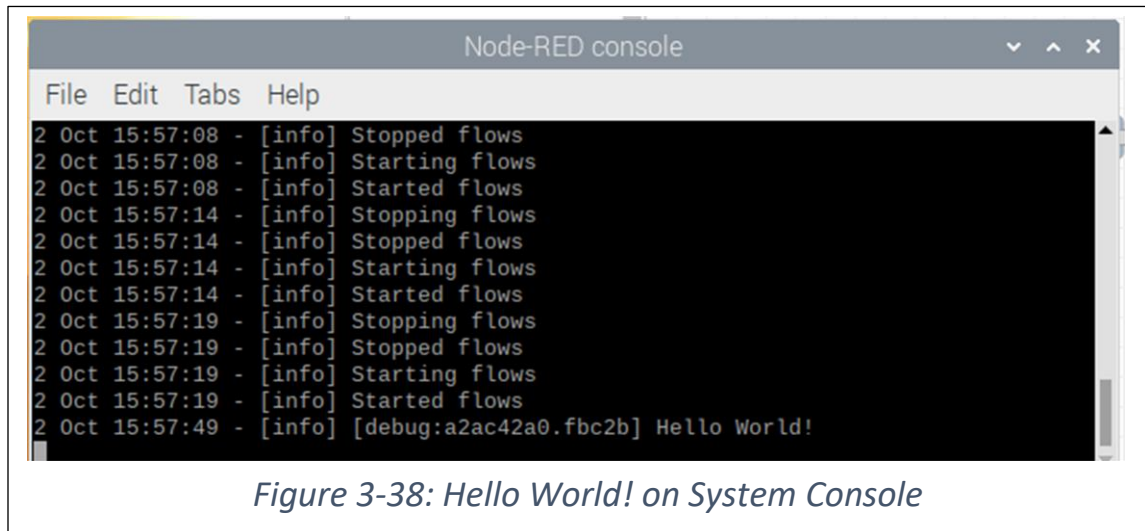


Figure 3-37: Edit Debug Node

You can send the output to different places, including the system console. Click on the box next to "debug window" to deselect it ①. Then click on "system console" to select it ②. Click DONE and don't forget to deploy your new flow.

## Learning Kit Workbook (version 1.3)

Press the button. What happened? Did you get the Hello World! message on the Node-RED console? If everything went according to plan you should see something like this at the bottom of your Node-RED console window.



Try this: close the IDE window and see if pushing the button still prints on the Node-RED console. If it does this means that your flow is still running. Nice!

### Making Noise – The Sound of Hello World!

Your Hello World! flow is slowly making contact with the outside world. How about using the push button to generate some sound? Easy to do. Let's see if you can modify the Pushbutton flow to say Hello World! rather than just printing it on the screen.

Your humble Raspberry Pi can produce sound, all you need is a headphone or external speaker, so, rummage around in the closet, find an old pair of ear buds and plug them in.

And now it is time for a word from our sponsor "The Engineering Good Practices Council":

---

**Engineering Tip # 6: Step by Step.** Adding sound to your program should be simple, however there are all sorts of things that can go wrong. First, stop and think about all the parts you are going to put together: headphones, a sound file, a new sound node and your old flow. If you put them all together, press the button and nothing happens you are going to have a big debugging problem on your hands. Instead, remember our motto: "Engineering is an incremental process". Don't build everything at once and try it out. If aeronautical engineers built a whole airplane at once it would probably fall out of the sky on its first flight, assuming that it did not just fall apart on the ground.

To avoid these sorts of expensive failures, engineers start by testing each part, then testing groups of parts and finally testing everything together. Sometimes, we refer to this by fancy names like "unit test", "sub-system test", "system test". Now there is not much to test for this little flow, but let's do



## Learning Kit Workbook (version 1.3)

things right. For example, that old pair of ear buds you found on the street last year after they were run over by a truck: do they work? That sound file you just downloaded: does it play? You get the idea.

- 
- Headphones – Check'em. Plug them into something you know works like your smart phone. Do they work? Great! Move on.
  - Raspberry Pi – Plug your headphone is and see if you can hear something. Maybe you could try playing a game you know generates sound. If you don't hear anything better stop here and figure out what is wrong. How you test your sound will depend upon what operating system you are using. For Raspbian (and probably all other Raspberry Pi systems) you can enter the command:

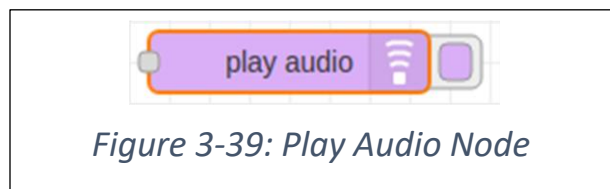
```
~ $ speaker-test -tsine
```

If your headphones and the Raspberry Pi is configured properly you will hear a nice tone (a sine wave) in both ears. If you don't hear anything, check the volume level on the main screen. Right click on the speaker icon and check the volume of the analog (headphone output) and HDMI (if you have speakers in your HDMI connected screen).

- Download the HelloWorld.wav file from the Sequent Microsystems site. **{We will need to set this up and provide some instructions}** and park it in a directory (write it down in your notebook!). Here we will park the sound file in /home/pi/sounds.
- Check that it works. This is easy to do by using the aplay application on the Raspbian OS (and probably all the other versions of Linux for Raspberry Pi.

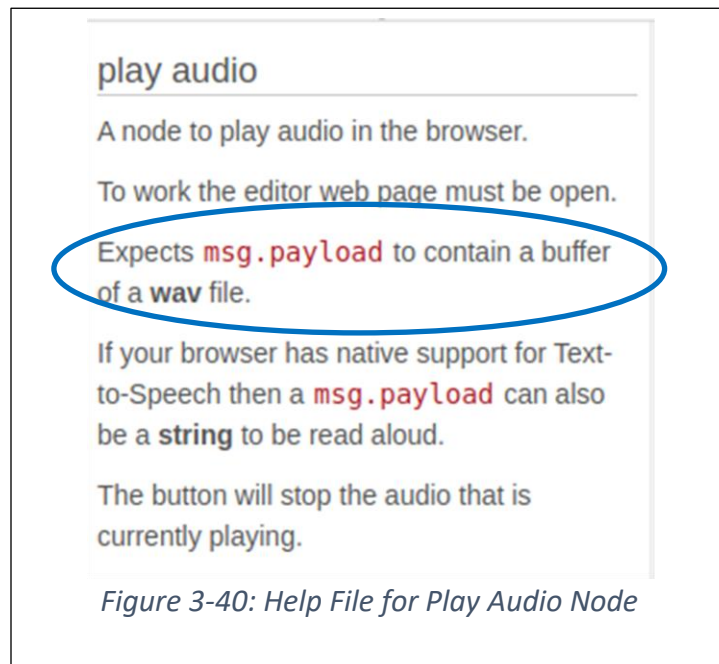
```
~ $ aplay /home/pi/sounds/HelloWorld.wav
```

- Once you have the sound playing you are ready to go.
- Scroll down on the palette and find the Audio Play node and drag it into your workspace,

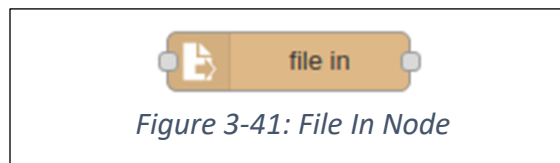


## Learning Kit Workbook (version 1.3)

- You might be thinking: “My, my this is something new. I wonder how it works.” You know how to find out: click on the node and then click tab with the book icon in the sidebar. This will give you the help file for the node.



- Check it out. What does this Play Audio node want as input? Right! It wants a stream of data from the sound file in .wav format (see the blue ellipse). This means you are going to need some way to read in your sound file, read it out, and connect it to the Play Audio node. Sure sounds hard. Time to go back to the palette and find a node to do the heavy lifting. How about the File In node?



- Bring it in and open up the help file (select the node, click the little book icon).
- It's always a good idea to check the help information for a node. Unfortunately, these descriptions can be a little thin on details, but at least you get some idea of what the node does. If there is not enough information, then you will just have to experiment around with the node. Happily, in this case the File In node looks like it will help us out.

## Learning Kit Workbook (version 1.3)

**file in**

Reads the contents of a file as either a string or binary buffer.

∨ Inputs

**filename** string  
if not set in the node configuration, this property sets the filename to read. ①

∨ Outputs

**payload** string | buffer  
The contents of the file as either a string or binary buffer. ②

**filename** string  
If not configured in the node, this optional property sets the name of the file to be read.

∨ Details

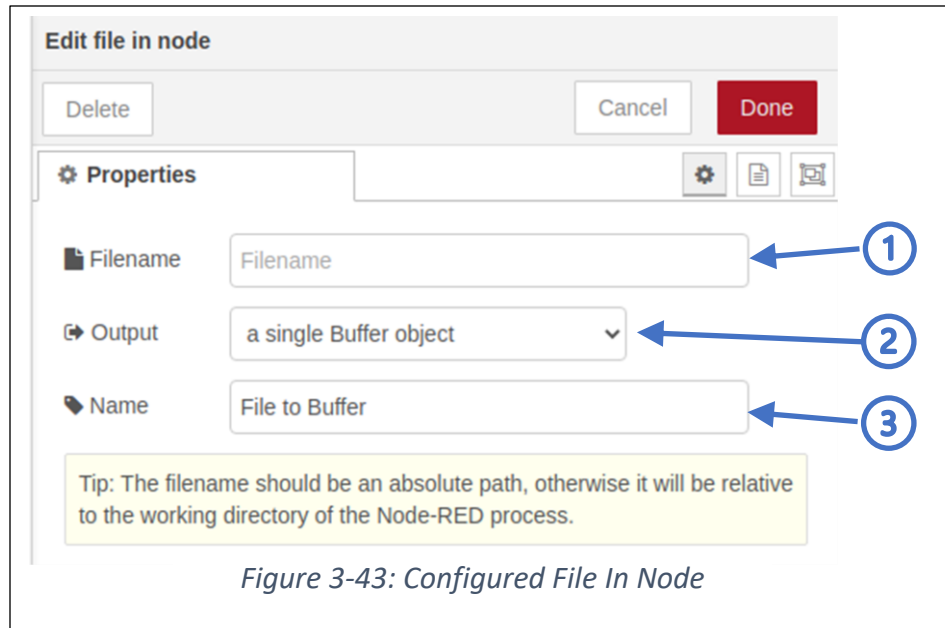
The filename should be an absolute path, otherwise it will be relative to the working directory of the Node-RED process.

*Figure 3-42: Help for File In Node*

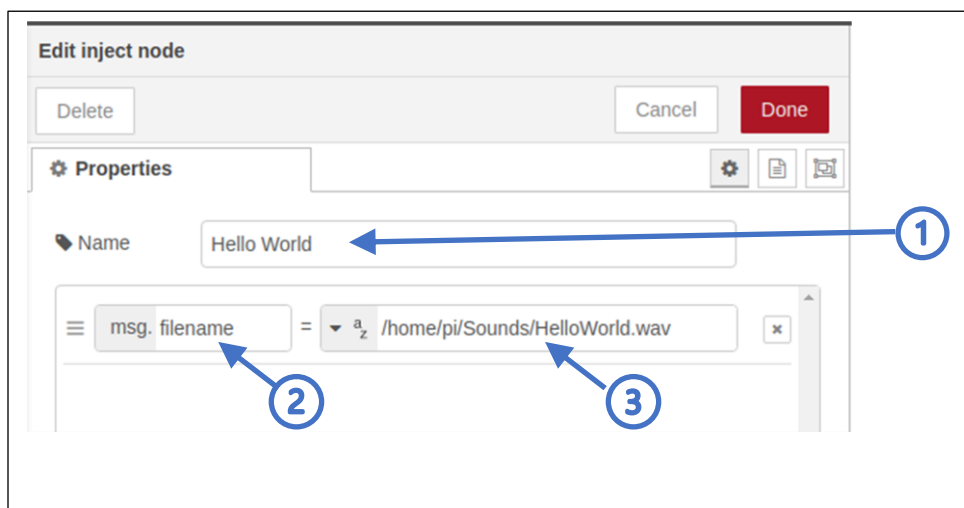
The input message should have a name of the file to read identified by the name “filename” ①. The output payload is going to be a buffer (i.e. a stream of data), just what the Play Audio node input is looking for ②. Hot Dog!

Notice under “Details” that the file name should be an absolute path (i.e. /home/pi/....) because otherwise it will be considered as a path name relative to the working directory of the Node-RED process, which is harder to locate. Now it is time to configure the File In node. Open the node by double clicking it and fill it in so that it matches the figure below.

## Learning Kit Workbook (version 1.3)

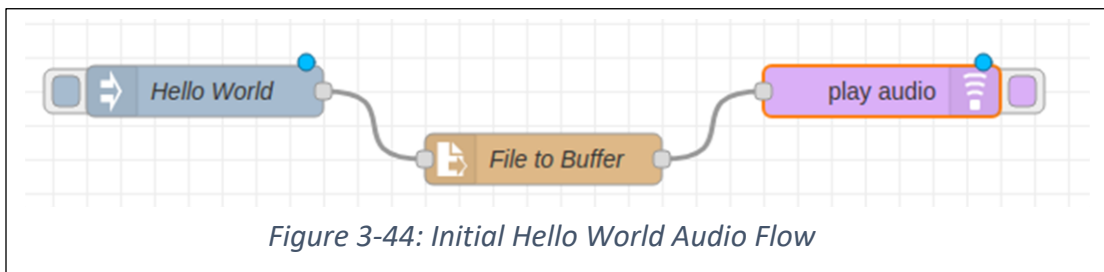


- Leave the Filename block ① blank because the filename is going to be specified by the message from the prior node. In the dropdown for output ② select “a single buffer object”. This means that the message from this node to the next node will be one “buffer” of data, basically a really long message. This buffer will be the contents of the sound file. Finally, fill in the Name block with a nice simple name ③. Click Done and you are ready to go.
- Bring in an Inject node. Double click the inject node to open the editor and configure your Inject node as shown in the figure below.



## Learning Kit Workbook (version 1.3)

- To configure the node, first delete the second rule which says “msg.topic”. You won’t need that now. Then, change the name of the inject node to Hello World ① because that is what it is going to do. Next, in the first rule ② change the property next to msg. from payload to filename. You need to do this so that the File In node can identify the filename property in the message. If you are confused or want to learn more go Appendix      Message Properties. Finally, ③ fill in the block next to msg.filename with the location of your sound file. Remember this must be an absolute path to the file. Click Done!
- Almost there... create your flow by wiring up the nodes as shown below.



- Now is the time! Deploy it! Click the tab on the left side of the Inject node and you should hear “Hello World”. When a file plays a little blue square will appear below the Play Audio node, and the file being played will appear below the File In node. The blue square will remain until the audio finishes playing. If something goes wrong a little red square will appear with an error message. The usual problem is that the file name you provided in the Inject node was not correct and there is no sound file there.

At this point you have a kind of prototype, it is not working with the pushbutton, but it is simple, and it works. When you first work with a new node, like Play Audio and File In, it is a good idea to start with a simple situation and expand from there because there are fewer places to make mistakes.

You already have a working flow using the push button that prints out a debug message. Let’s use parts of that flow to replace the Inject node in this flow with part of the pushbutton flow so that when you put the button your Raspberry Pi will introduce itself to the world. Here is the recipe..,

- Go to the tab with the flow you just completed (see figure     ) where clicking the Inject node plays the Hello World audio file.
- Import (see     ) the Flow you built earlier (figure     ) where pushing the button caused a debug message of “Hello World! to be printed). The flows will be on two separate tabs. Time to learn how to copy nodes from one tab to another. There are three ways to identify the nodes you want to copy:
  - Method 1 – hold down the control key and right click on each node you want to copy.
  - Method 2 – hold down the shift key and right click on a node and all nodes associated with that flow will be selected.

## Learning Kit Workbook (version 1.3)

- Method 3 – draw a box around the nodes you want to select by holding down the right mouse button and dragging over the nodes.
- Now that you have selected all the nodes from the Pushbutton to Debug flow type `ctl-c` to copy, click on the tab for the Inject to Play Audio flow and type `ctl-v` to insert it. You can then drag it to where you want to and click to drop it in place. You should have a workspace that looks like the figure below.

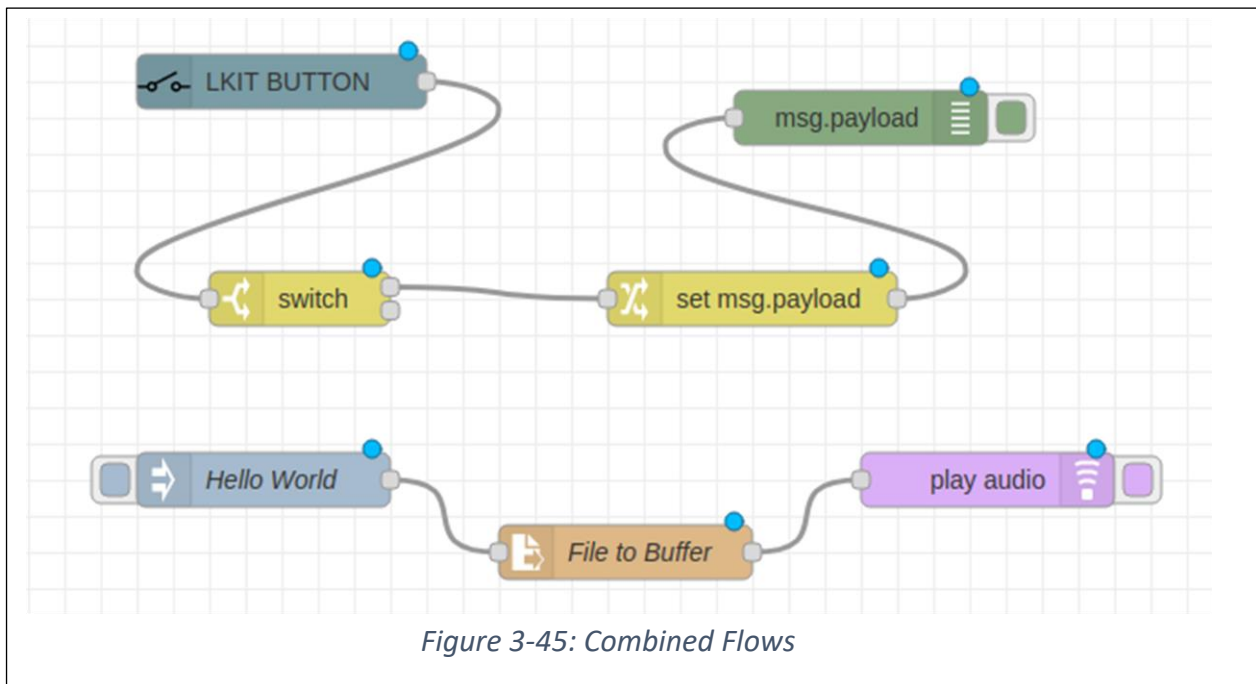

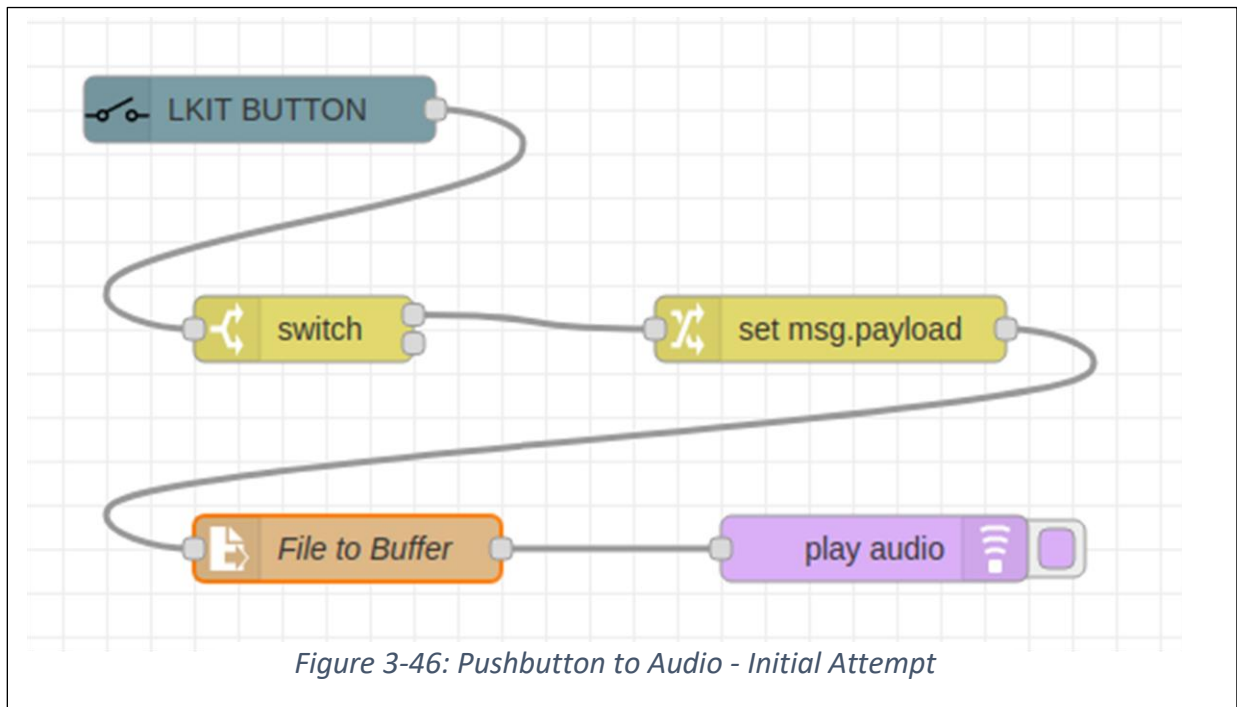


Figure 3-45: Combined Flows

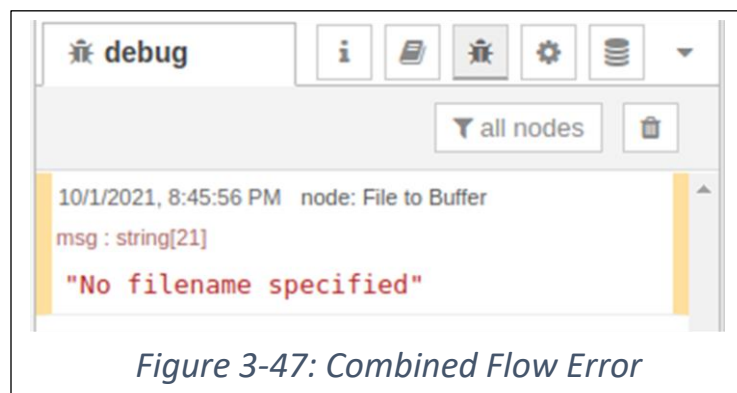
- Go back and delete tab with the Pushbutton to Debug flow. This is very important because if you deploy both flows there will be two copies of the Pushbutton to Debug flow present and you will get two messages every time you push the button, one from each flow.
- Go back to tab with the combine flow (figure ). At this point everything you need is in the same tab you just need to connect it up and make a few adjustments. Remember... small steps.
- Deploy the combined flow above and test each flow separately. Pushing the button should print "Hello World" in the debug sidebar window. Clicking the Inject node should still play "Hello World" over your headphones or speakers. Remember... test the piece like this before you hook them up.
- Stop for a moment... Deep Breath... Slow Exhale... imagine you are a beach with the sound of waves in the background. Now, think about how you might combine these two flows into one flow where pushing the button causes "Hello World" to play in the speaker. You know that every time you push the button (but not when you release it) the Change node labeled set msg.payload will send a payload to the debug node. You also know that the Inject node labeled Hello World is sending the filename to the File In Node (labeled File to Buffer). Doesn't it seem like connecting the output of the Change Node to the input of the File Input node is part of the solution. Make it so!

## Learning Kit Workbook (version 1.3)

- Delete the two unused nodes and your flow should look like this:



- Do you think this will work or not? Why? Can't hurt to try, so deploy it and push the button.
- Oops... did you get an error in the debug window. It looks like the File to Buffer node is complaining that it is not getting the file name. Did you notice that rather than some cryptic number that the debug message actually gave you the name of the node where the error occurred? That is why it is a good idea to name every node in some clear way. It will make debugging much easier.



- At this point you could try debugging with the Debug node. Alternatively, you could apply the most powerful debugging tool available... you could think about the problem a bit more. The error is that the File to Buffer node is not getting a file name at the input... Hmm... Aha! The

## Learning Kit Workbook (version 1.3)

Change node is sending the payload of the message to the File to Buffer node where it should be a message with the property `filename` set to the sound file name. Let's fix it<sup>4</sup>.

- Double click on the Change node (labeled `set msg.payload`) to open the node editor. Here is what you will see:

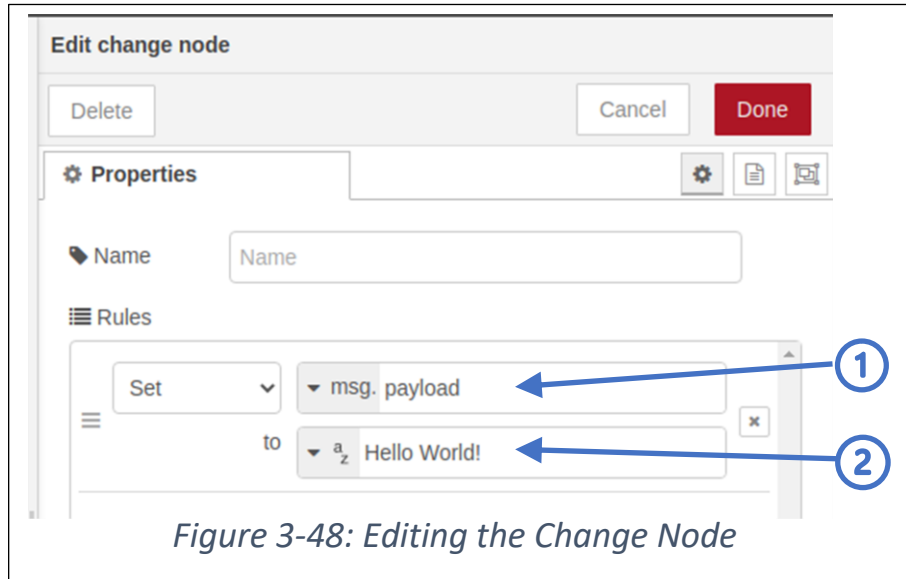


Figure 3-48: Editing the Change Node

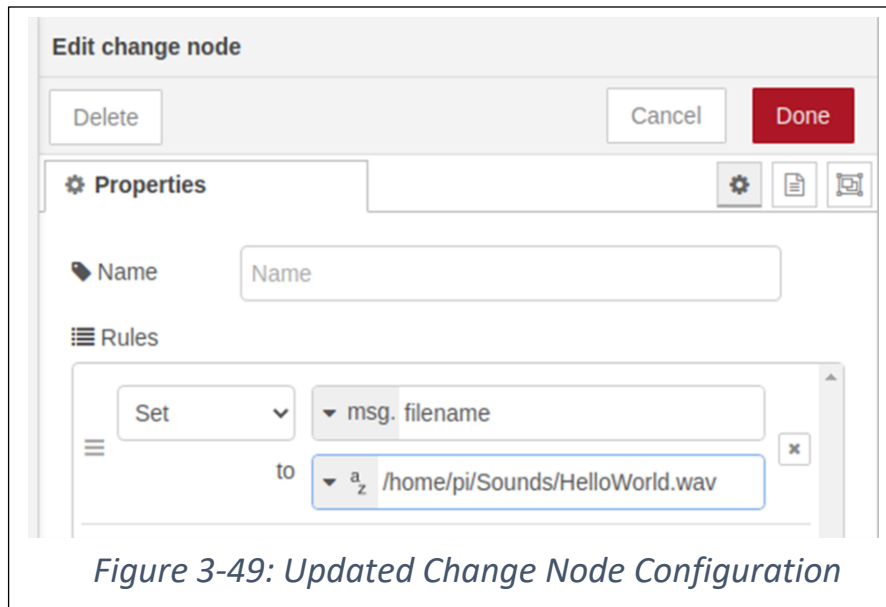
- First, change the message property (①) from payload to `filename`. Second, change the string from `Hello World!` to the file name where your audio file is, which may be different than what is shown below. Now your Change node configuration should look like the figure below:

---

<sup>4</sup> If you are curious, and you should be, you can verify this by simply attaching a debug node to the output of the change node and checking the message. You will see that the message contains a payload of the 1- character string "Hello World!"



## Learning Kit Workbook (version 1.3)



- Click Done! Deploy it! Push the button! Do you hear the sweet, sweet sound of Hello World! If so, congratulations! This is a complicated flow that uses almost everything you have learned. You are ready to start running.
- Don't forget – document your work. Now is the time to change node names to something descriptive. Write out a description of what you did so that when you need this flow next time you will be able to figure out what was going on. Save a copy in your archive.

### Puzzles for You to Solve

Sound is fun to work with. Here are some little puzzles for you to solve. There are many places on the Internet where you can find free sound samples. As always be careful, because if you sign up for something you might start getting lots of emails. A reliable place is the [BBC Sound Effects Library](#). Also [Free Special Effects](#) seems like a safe site with all sorts of sounds. If you want to edit sounds you can download [Audacity](#). It's free and works on all the platforms. [Text to speech translators if we can find a safe site that does not require registration]

**Puzzle #1: Sound Effects Box** – You must supply sound effect for a play. Build a flow to produce one of five sound effects on demand. Use a separate Inject node to trigger each sound when you click on it.

**Puzzle #2: Buzzer** – build a flow that will play a buzzer (or siren or whatever) sound as long as you hold down the button. You might need to learn more about the Play Audio node to do this. By changing the message topic you send to the Play Audio node you can start and stop the playing of a sound file.

**Puzzle #3: Old Fashioned Doorbell** – the old electric doorbells were set up so that when you pushed the button the bell would Ding and when you released the button the bell would Dong. Can you build a flow that does this?

## Learning Kit Workbook (version 1.3)

**Puzzle #4: Simple Alarm Clock** – remember how the Inject node can be programmed to produce a message at a particular time (see [\[redacted\]](#)). Can you build an alarm clock that will go off at a particular time and continue until you hit the pushbutton to stop it?

**Puzzle #5: Workday Alarm Clock** – Extend your alarm clock from Puzzle 4 to wake you up at a different time each day. So maybe you have to straggle out of bed at 6:30 AM Monday through Friday, but you can sleep until 8:00 AM on Saturday and Sunday. Can you build the flow? Do you trust your flow enough to use it? How are you going to test it? Can you play a particular alarm sound on weekdays and a different one to wake up to on the weekends!

**Puzzle #6: Your Idea** – Can you think of an interesting sound-based problem that will work with either the pushbutton, the Inject node or both.

Time to RUN!

### Running – Pushbutton Controls the World

#### Pushbutton Sends Email

With each flow you have created, the little pushbutton has touched a bit more of the wider world. Now it is time to run. Let's see if you can get the push of a button to send an email. You are now embarking on the real Internet of Things, IoT. If you can send an email whenever the button is pushed, then you can be anyplace in the world and know almost instantly about an event, a worldwide doorbell! This is the real deal because if you go on Amazon, you can buy a little button that will reorder a box of pet food<sup>5</sup> anytime you push it. You are now going to build something very similar.

Now for a word from one of our sponsors: Peter Parker Proverbs, Inc. Remember the Parker Principle from above (section [\[redacted\]](#))? Yeah, yeah great power, great responsibly, that sort of thing. Time to apply it. If you are going to send messages over the internet, make sure that you work carefully. Make sure you know where you email is going, don't send messages willy-nilly to any old internet address, try not to create a program that sends 10,000 message a minute.

Okay, enough said. First you are going to need an internet address to experiment with. It is easy to get a Gmail address from Google. If you are going to experiment you should use a newly created Gmail address rather than your own email address. The reason is that to make things easy we are going to cut a few corners on security. Your personal email should be as secure as you can make it, so for goodness sakes, don't compromise the email you use to do your banking, make airline reservations, order Hamster Habitats from Amazon and so forth. Use an email address that you can shut down any time.

---

<sup>5</sup> Important – if you ever purchase an Amazon Dash button for this purpose do not leave it in the cage with your pet. Otherwise, you might wake up some morning and find a trailer truck full of Nature Farm Hamster Treats parked outside your house, a \$14,930.43 charge on your Amazon account and a pet with a very sore paw.

## Learning Kit Workbook (version 1.3)

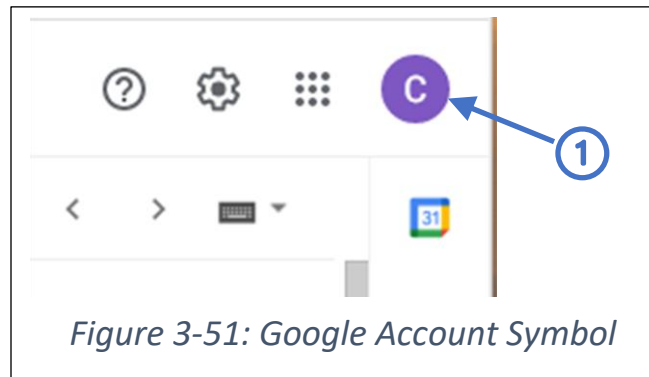
Create a Gmail account just for experimenting. No instructions here because it is easy to do and if you can't create a Gmail account on your own then you should not be trying this. Got your Gmail account?



Good, now you need to configure it so that you can use it from Node-RED. In this section we are going to do a very simple thing, namely send a Hello World! message from this new Gmail account to your personal email account. Believe it or not there is a Node-RED node just for this purpose, in fact it is right in the palette.

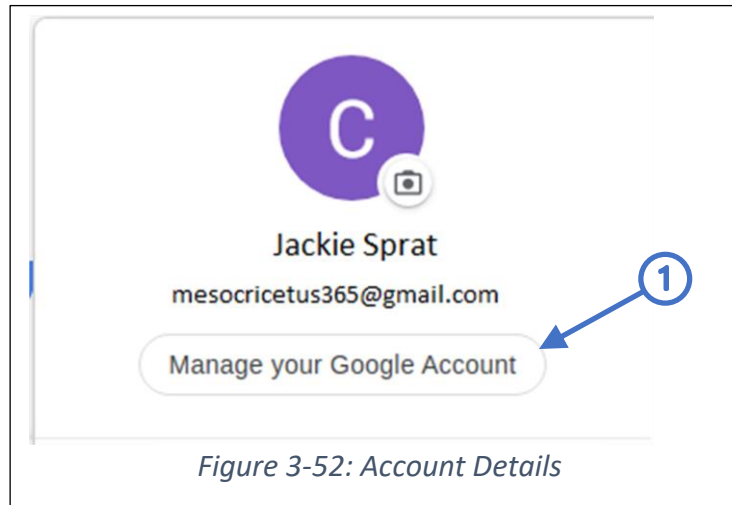
The problem is that it needs to work directly with your email account using your username and password. Normally, Gmail likes to know that the application that is telling it to send an email is trustworthy, and the work of thousands of careful, dedicated and well-fed programmers, something like Microsoft Outlook. Gmail does not consider the Email Send node above to be all that trustworthy. To make use of it in your new email account you will need to allow “less secure apps” access to your Gmail account.

Open your Gmail screen. In the upper right there will be an icon ① that represents your account (It will probably be different from the purple “C” shown here).

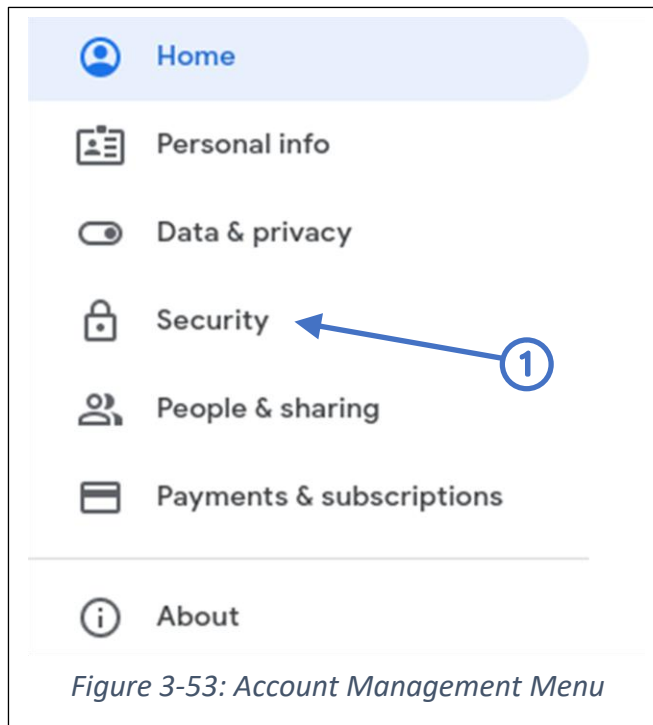


Click on the account symbol, which will open a window identifying your account. Now click ① on the “Manage your Google Account” button.

## Learning Kit Workbook (version 1.3)

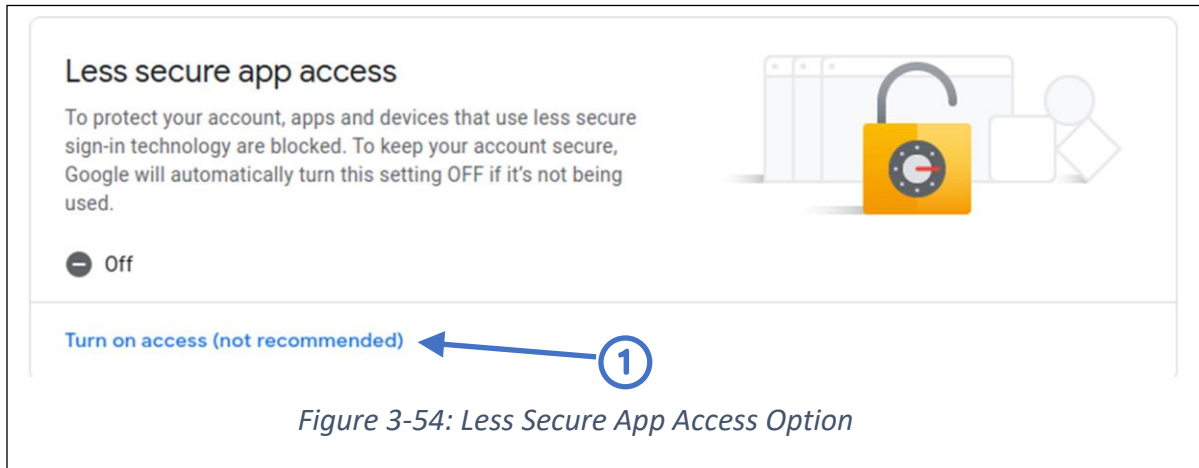


On the left side of the screen select ① “Security” from the menu.

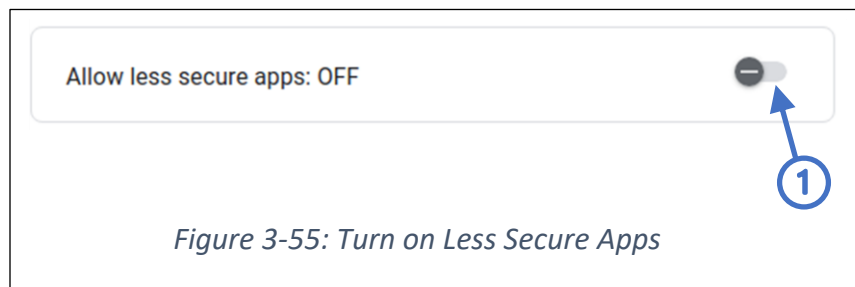


Scroll down until you see a block labeled “Less Secure App Access”;

## Learning Kit Workbook (version 1.3)

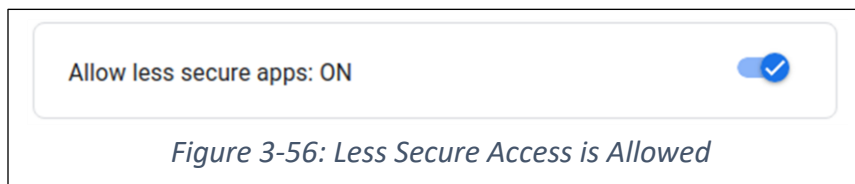


Click the link at the bottom ① “Turn on access (not recommended)”, which will open the screen below. Remember... Google does not recommend this, but they will allow you to do it, got it?



Click to the right of the check mark to turn on less secure access ①. Now you should see that the less secure app access is on.

The next time you return to your Gmail inbox you will probably find a security alert stating that your account now allows access by less secure apps. You should acknowledge this message. When you lower the security of your account it is not permanent. If your low security Node-RED app does not use your account for some period of time Google may turn the access off automatically.



## Learning Kit Workbook (version 1.3)

Remember: you have lowered the security of all applications on this Google account, so do not connect this account to any other account you have or store anything important on the Gdrive. You should make sure your password is at least 10 characters long and is just random characters. Don't use the name of your pet which is already posted on your Facebook page.

At this point you should have a pretty good idea what to do. Import your pushbutton flow that sends messages to the Debug node. All you need to do is replace the Debug node with the Email Send node. Well, you need to do a bit more because the Email Send node needs to know how to use your email server and you also need to build up the message you are going to send.

After you bring back your Pushbutton flow, deploy and test it to make sure it still works (remember Engineering Tip # [redacted]?) Now replace the Debug node and with the Send Email node, and while you are at it rename the flow to be "Gmail – Hello World". You should have something like the figure below. Don't deploy your flow yet because you need to make some changes. By the way, don't you just love the way the wires between nodes have that beautiful, curvy, almost sensuous look? When you are bored look up [Bézier curves](#) on Wikipedia.

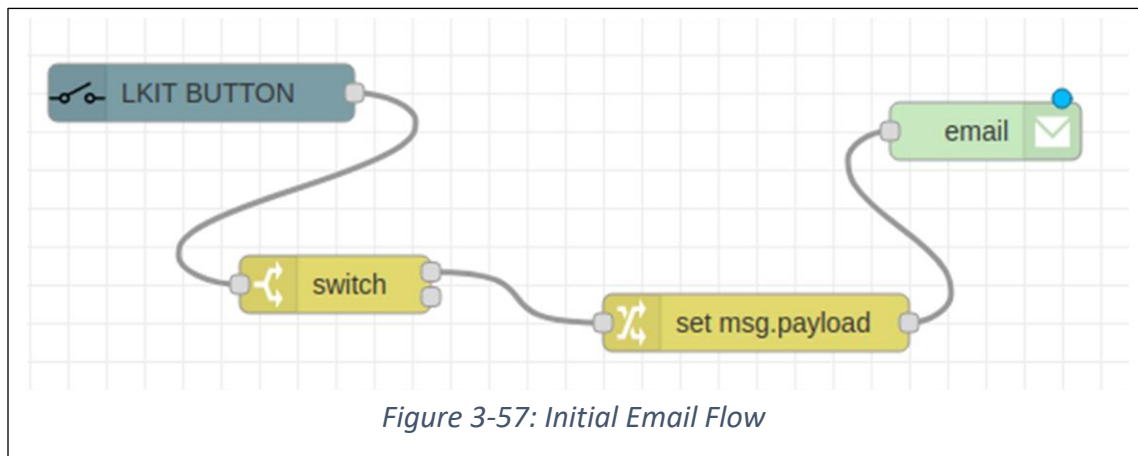
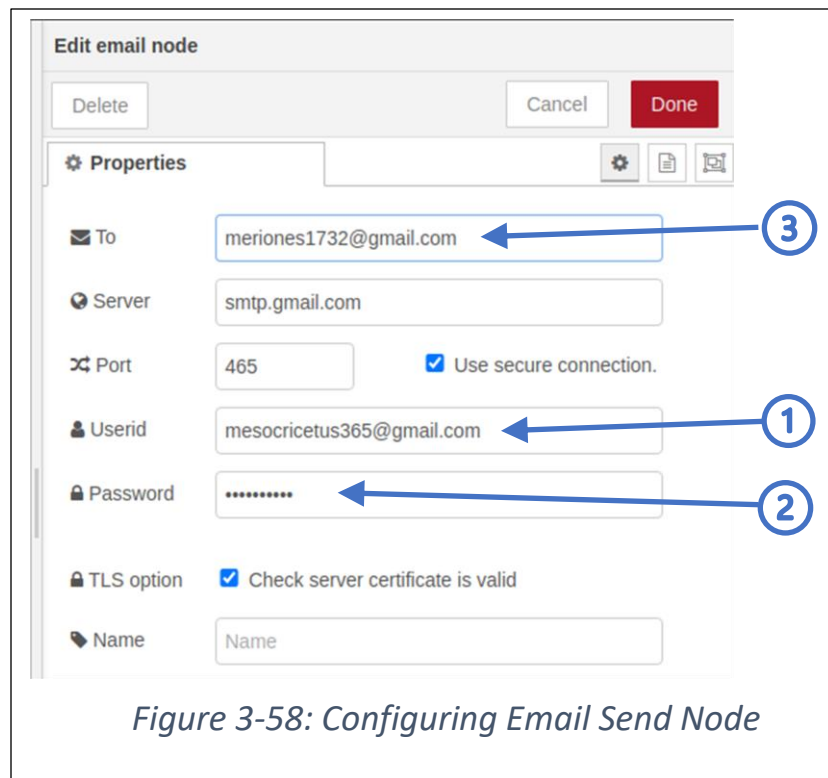


Figure 3-57: Initial Email Flow

## Learning Kit Workbook (version 1.3)

First, let's fix up the Send Email node so that it can work with your email server, which is the remote program that will send and receive your emails. As usual, double click on the Email Send node and it will open an editing window:



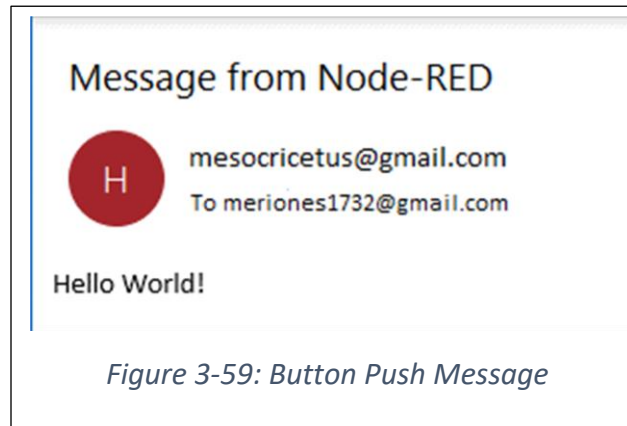
This window is set up for Gmail and you only need to fix a few things. Number one, put in your Userid ①. Number two, carefully fill in your password. You will not be able to see the password ②, so type carefully. Number three, enter the email address where you want your message to end up in the “To” box ③. That’s all there is to it!

Deploy it! Push the button! If everything worked out a little blue square will appear near the bottom left of the Email Send node that says in teeny-tiny print “sending”. If things don’t go right a red box will appear with the dreaded word “error”. Go back and check everything.

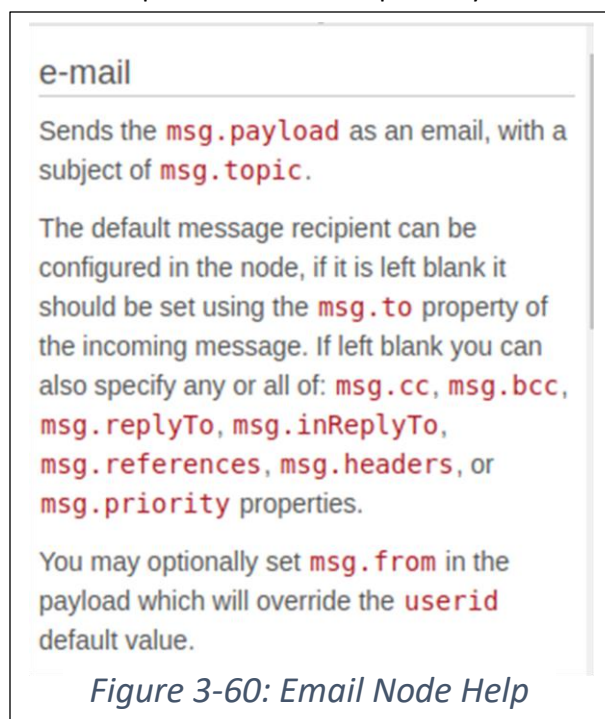
Assuming everything went right, go to you’re the email where you sent the message and see what you received. It will look something like the figure below. If you do not see your message the first thing to

## Learning Kit Workbook (version 1.3)

check is the junk folder. You may need to “whitelist” the address you are sending from to sneak by the spam filter on your email system.



Hmmm... the subject line is a little cryptic. Maybe you would like to change it. Maybe you would like to send the email to more than one person. Maybe you would like to put the name of your device in the window. Click on the Email Send node to select it and then open the Help window (tab with the little book) in the sidebar window. Here is part of what the help file says:



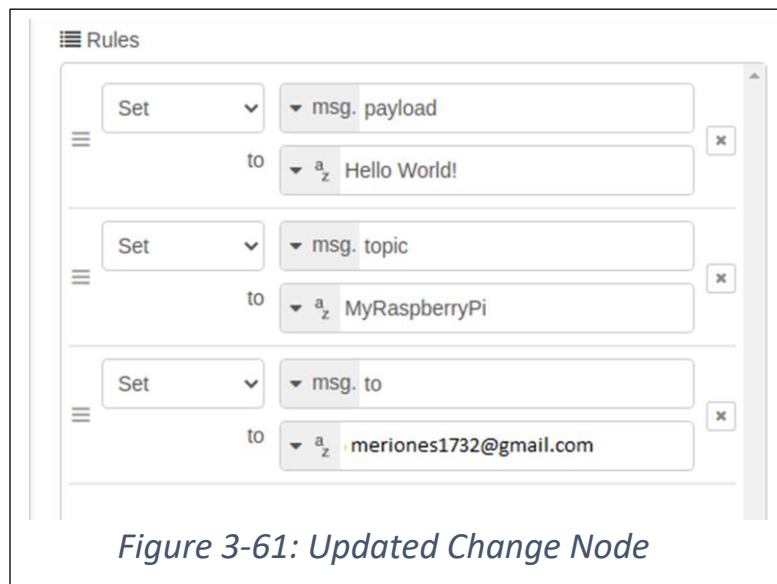


## Learning Kit Workbook (version 1.3)

Up to this point you might have been thinking that the message just has a payload, which you set to Hello World! The complete picture is that a message can have many components each identified by a name. As the figure above shows the message can contain “to” addresses (msg.to), “cc” addresses, (msg.cc) and a topic (msg.topic) among other things. You can adjust all these within the Change node so that the message the Email Send node receives can control more aspects of the email process.

We will give the email a topic, which will appear as the subject line. We will also set the “to” address in the change node rather than in the Email Send node. You can do all this by defining parts of the message payload in the Change node.

First, open the Email Send node and delete the address in the “To” block. You are going to supply this address in the message that the change node will generate and send to the Email node. This is convenient because all your email parameters are in one place, the Change node rules. Now, open the Change node and set up the rules as shown below for your specific situation.



*Figure 3-61: Updated Change Node*

Deploy! Push the button! Check your email! Do you see something like this?

## Learning Kit Workbook (version 1.3)



### Pushbutton Sends SMS Message

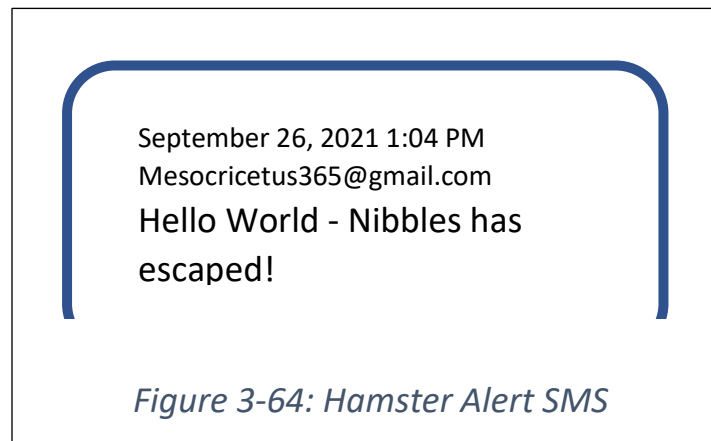
Are you thinking “Email is nice, but I am going to hook that button up to my hamster’s cage door so that I know when she kicks it open. I don’t want to wait for an email. I want to know right now!”? No problem, if you can send an email you can send an SMS message right to your phone. Many cellphone carriers have a way for you to send them an email and they will automatically forward it to your phone (or any other phone for that matter). All you need to do is go your carrier’s web site and find out the



email address. If you are using a US carrier you can go to [FreeCarrierLookup.com](http://FreeCarrierLookup.com) (Remember: they have ads and heaven only knows what sort of cookies they are going to leave on your system). Here is what the results look like:

## Learning Kit Workbook (version 1.3)

All you need to do is go to the Change node in your flow and plug in the SMS Gateway Address as your “msg.to” address and next time your pet pops the cage door in the middle of the night<sup>6</sup> you will get a text message. Give it a try! Nibbles is out:



Of course, if you want the Nibbles’ cage door to push the button on your I/O Learning card it is going to take some rather weird mechanical arrangement. Wouldn’t be nice if you could just run a wire from the cage door switch and connect it directly to the card? Stay tuned. One or two more chapters and you will be able to do this!

## Chapter 4 - Let There be Light – LEDs

What you will learn:

- Using on-board LEDs
- Delay Node
- Subflows
- Link In and Link Out nodes
- Context Variables
- Set up initial conditions
- Loops
- Link In and Link Out Nodes
- Random Number Node
- Controlling Operations from Email Messages
- Retrieving Information from the Internet
- Range Node

---

<sup>6</sup> Hamsters are nocturnal animals, and just like students, they would rather sleep during the day and play at night.

# Chapter 4 LEDs - Preliminary (V0.0)

## Introduction

Your Learning card contains four onboard Light Emitting Diodes (LEDs). LEDs are very simple devices. On the Learning card the only thing you can do with the LEDs is to turn them on and off. Just like with the pushbutton in the previous chapter let's take these simple devices and see how far you can push them. As usual you will start with simple steps and proceed to some quite complicated setups. While you do this you will learn more and more about the capabilities of Node-RED, which is the important objective of this chapter.

## Crawling Once Again – Controlling One LED

### Locating the LEDs

First things first... find the LEDs on your Learning Card. Take a look at the card on the edge that has the pushbutton. You will see nine LEDs and on the top of the card in tiny little print you will see the names of the LEDs (check out Figure 4-1).

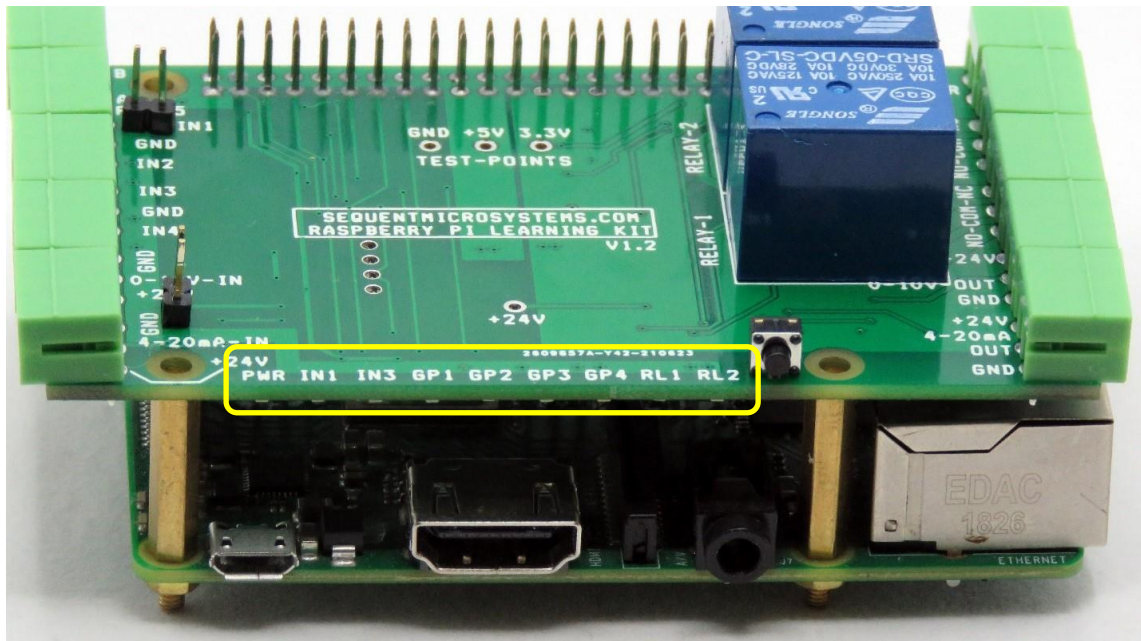


Figure 4-1: LEDs on Edge of Learning Card

The LEDs are green and are located on the bottom side of the Learning card, and although the light shines out from the side you will find that you can easily see the lit LEDs through the board, so the visibility is very good. The Learning Card has nine LEDs:

Label	Purpose
PWR	Flashes when power is on
IN1	ON when input circuit 1 is closed (see Chapter 5)
IN3	ON when input circuit 3 is closed (see Chapter 5)
GP1	User controlled 1
GP2	User controlled 2
GP3	User controlled 3

## Chapter 4 LEDs - Preliminary (V0.0)

GP4	User controlled 4
RL1	ON when relay 1 is activates (see Chapter 6)
RL2	ON when relay 2 is activates (see Chapter 6)

LEDs GP1 to GP3 are available for you to control. The other LEDs are control in response to inputs or to the operation of the relays, which means that you cannot explicitly control them (more on this later).

### Controlling the GP1 to GP4 LEDs

You can control the GP1 to GP4 LEDs directly from Node-RED. You do this by using the LKit LED node (Figure 4-2).

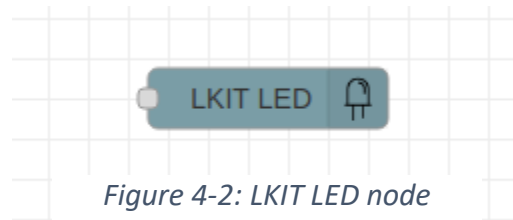


Figure 4-2: LKIT LED node

The LKit LED node is an output node and has only one port, the input port. LEDs may be controlled in two ways depending upon the mode of operation you select. An LKit LED node can be placed in one of two modes “single” and “group”. In the “single mode” the node only controls one LED, whereas, in “group” mode the node controls all four GP1 to GP4 LEDs at once.

Start by bring in a copy of the LKit LED node and dropping it in the middle of your workspace. Now... Ooops! Time for a commercial break from the Council of Good Engineering Practice.

---

**Engineering Tip # 7: Read the Documentation!!** Yes, really, take the time to look at the documentation. Whenever you start to use a new node, new subroutine, new chip, new component, new language or new type of hamster water dispenser, it pays to at least glance at the documentation. Read it and make sure you really understand what it is saying. This will save you from making your life miserable when you hook something up wrong and spend all day improving your debugging skills. To paraphrase a famous author<sup>7</sup>: “An afternoon of debugging can often save you 15 minutes of reading the documentation”. Also, remember to get the latest documentation if it is available. Many projects have foundered on the rocks of ignominy when the engineers were using out of date documentation for design. Don’t let this happen to you. On the other hand, don’t trust the documentation until you have made sure that the

---

<sup>7</sup> Frank Westheimer – American chemist (1912- 2007): “a couple of months in the laboratory can frequently save you an afternoon in the library”

## Chapter 4 LEDs - Preliminary (V0.0)

behavior of your node, circuit, subroutine etc. behaves the way the documentation says it should. Be wary, be cautious, be skeptical. And now back to our normal programming...

You have done this before... click on the LKit LED node and then click on the Help icon (the cute little book) in the sidebar. This will bring up the information about how to use the node. Sadly, you will find that many contributed Node-RED nodes are a bit thin on documentation, but, happily for you, the Sequent Microsystems Learning Kit nodes come with adequate documentation (at least we think it does, but you be the judge). Here is what you should see:

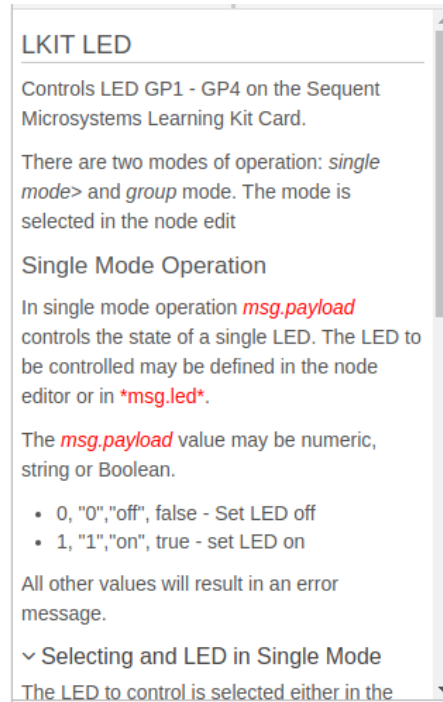


Figure 4-3: LKIT LED Help Window (Partial)

### LED Node – “Single” Mode Operation

Start simple! Hook up the LKit LED node to an Inject node and see if you can control just one LED, the one named GP1. You are going to be using the “single” mode where the node only controls one LED. Here is how to do it.

- Bring in a copy of the LKit LED node from the palette.
- Double click to open the node editor
- Set the mode to “single” because you just want to control one LED rather than all four (In a new copy of the node it will already be set because it is the default, but it is always good practice to check).
- Set the LED to 1 indicating that you will be controlling the LED labeled GP1.
- Change the name of the node to LED 1 (Why not? It will make things easier to understand)

Now you should see this configuration for the LED Node:

## Chapter 4 LEDs - Preliminary (V0.0)

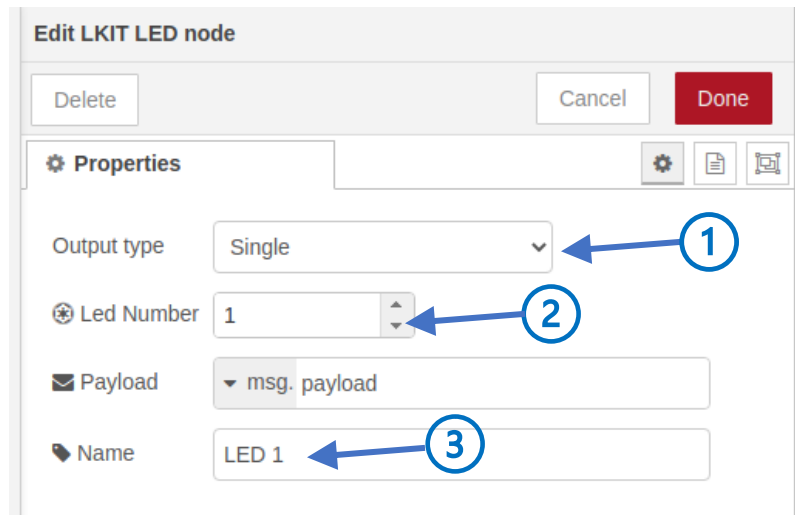


Figure 4-4: LKIT LED Edit Window

- Click Done to close the editor for the LED node.
- Relax.....
- Now Focus!
- Now bring in an Inject Node and drop it to the left of the LKit LED node.
- Connect the two nodes
- Double click on the Inject node to open the node editor
- Change the name of your Inject Node to "ON" ①.
- Delete the *msg.topic* rule because you will not need it here ②.
- Edit the *msg.payload* rule so that the payload is type number with value 1 ③.

## Chapter 4 LEDs - Preliminary (V0.0)

When you are done the Inject node edit should look like this.

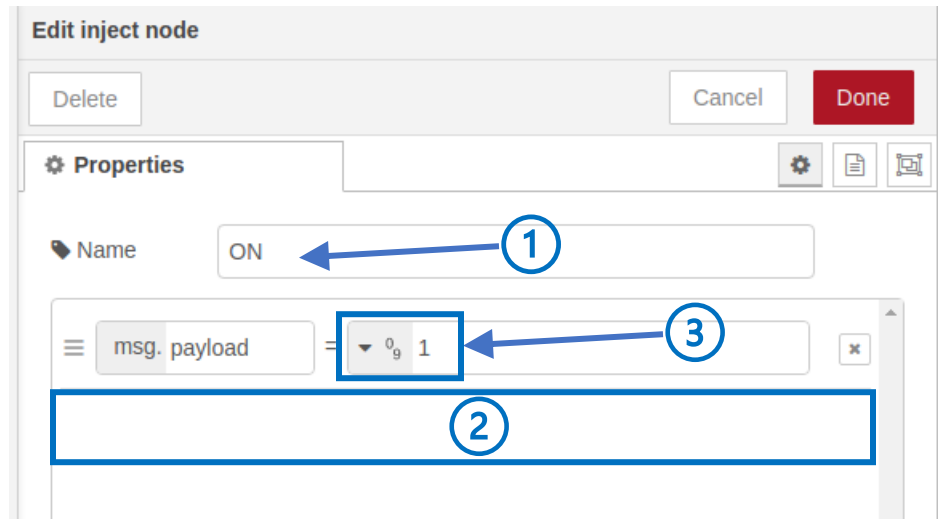


Figure 4-5: ON Inject Node Edit

- Click DONE to close the editor and...

**Deploy It!** Yes, Yes, Yes, don't forget this step!

Click on the ON Inject node. What happened? Did LED GP1 turn on or was it on already and nothing happened?

Okay, so maybe you need a way to turn the LED off. To do this add another Inject node and wire it to the input of the LKit LED node (See Figure 4-7)

Open the node editor for this new node by double clicking it and edit it the way you did for the ON Inject node above:

- Change the node name to "OFF".
- Delete the *msg.topic* rule. (Don't need it here)
- Set the *msg.payload* rule so that the payload is type number with value 0
- When you are done it should look like this:



## Chapter 4 LEDs - Preliminary (V0.0)

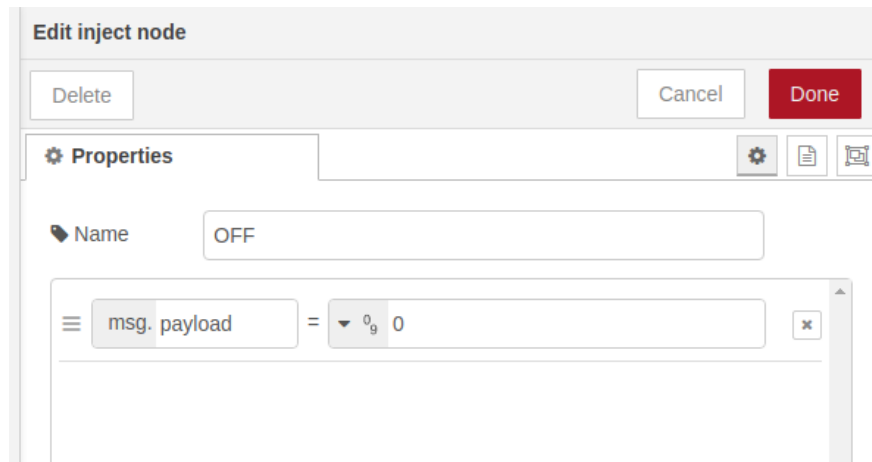


Figure 4-6: OFF Inject Node Edit

- Close the editor (click DONE)
- Your finished flow should look like this:

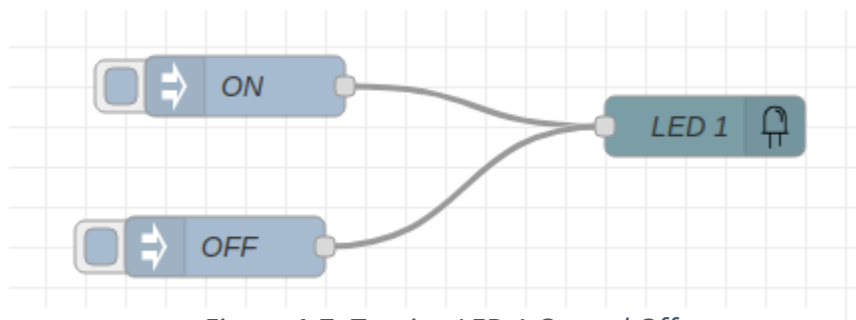


Figure 4-7: Turning LED 1 On and Off

Deeeeeeplooooy It!

Now when you click the ON Inject node the LED 1 should turn on and when you click the OFF LED node the LED should turn off. Is this how your flow is behaving? If not, then you will need to drag out the Debug node and look at the messages you are sending the node. If you hook a debug node up to the ON and OFF nodes you should see a message every time you click one of the nodes in the debug sidebar (Remember? The tab with the little bug on it.) This message should be either a 0 or a 1 in the form of a number. Also, check that the configuration of the LKit LED node is set to control LED 1 (GP1).

If everything is working, go into the LKit LED node and change the LED to 2. Deploy it. Click OFF and ON and see if LED 2 is changing. Now rinse and repeat for LED 3 and LED 4.

Okaaaaay. Now you probably just turned each LED on and off a few times. Did you try to see whether you could turn on one LED and then turn another one on and off? This is the essence of the “single”

## Chapter 4 LEDs - Preliminary (V0.0)

mode that you are currently using with the LKit LED node: each LED works independently of the other LEDs.

Try this:

- Modify the LKit LED node of your flow so that it controls LED 1.
- Make a copy of your first flow and drop it below the first flow (*ctl-c, ctl-v*)
- Modify the LKit LED node of the second flow so that it controls LED 2
- While you are at it open each LKit LED node and change the names to something like LED 1 and LED 2, just so that you know which is which.
- DEPLOY it for heaven's sake!

Your workspace should look something like the figure below.

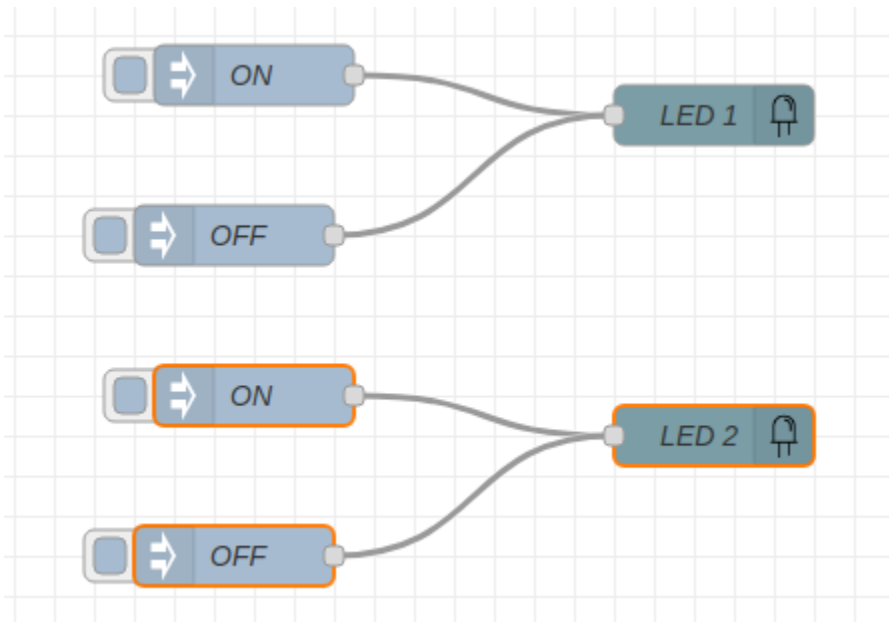


Figure 4-8: Controlling Two LED Nodes

Experiment with turning the LEDs on and off. Can you create every pattern of two LEDs? Here is a little test table:

LED 2	LED 1	Did It Work?
OFF	OFF	
OFF	ON	
ON	OFF	
ON	ON	

When you are working on your flow, it is always important to have an idea of how you are going to test it completely. Does the table above cover all the possibilities? What would the table look like if you were to build a flow like the one above but for four LEDs? Okay, extend the flow above to control all four GP1 to G4 LEDs and test it. If your flow had 10 LED how many rows would there be in your test

## Chapter 4 LEDs - Preliminary (V0.0)

table? Sorry, no extra credit for your answer, just the satisfying feeling of accomplishment. That's enough, isn't it?

### Many Outputs to One Input

Inputs can receive messages from multiple sources. You can see this in the flows you just completed because the LKit LED node is receiving messages from two different Inject nodes. There is no limit to how many other nodes a node can receive messages from other than making your flow so complicated that even you, the creator, cannot understand it anymore.

Take your last flow and modify it so that it looks like the figure below. All you have to do is move the wires around.

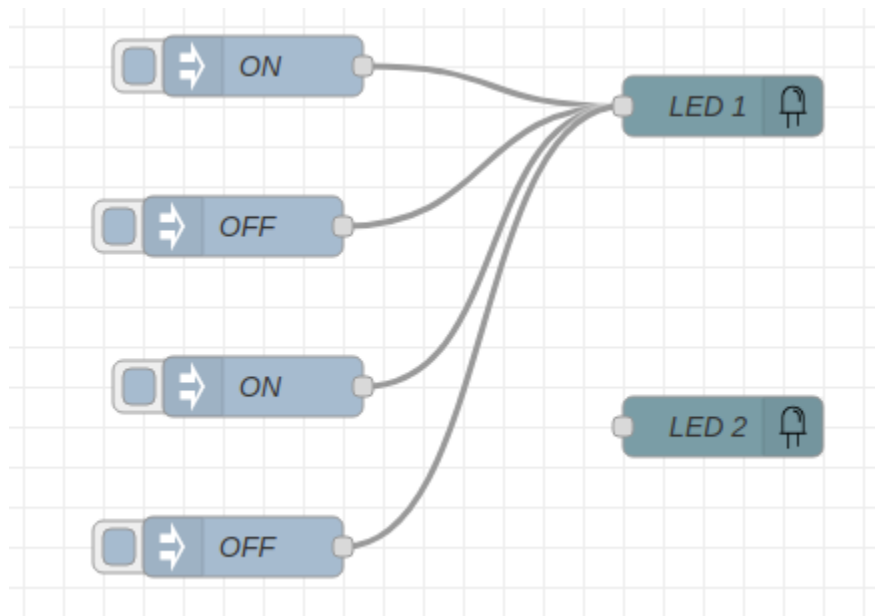


Figure 4-9: Four Nodes Controlling One LED

Note: the LED 2 node is not connected to anything, so it is not involved in the flow.

Ask yourself this: "Is this going to work? If it does work, what is going to happen?" Take a mental break and think about this. Breath. Center. Think. Okay... click the Inject nodes. Does it work the way you thought it would? If not, first ask yourself, "Did I deploy the flow after I made the changes?" Oh, yeah! Those little blue dots mean you did not deploy the flow. Try again. If it still does not work, then you should drag out a Debug node and see if the messages going into the LKit LED node for LED 1 are what you thought they should be. Just remember that you cannot connect the Debug node directly to the input of the LKit LED node for LED 1, which would certainly be convenient. Rather you, must connect it to the outputs of all four Inject nodes. Messy, but at least you can see what is going on. Here's what your debugging arrangement might look like:

## Chapter 4 LEDs - Preliminary (V0.0)

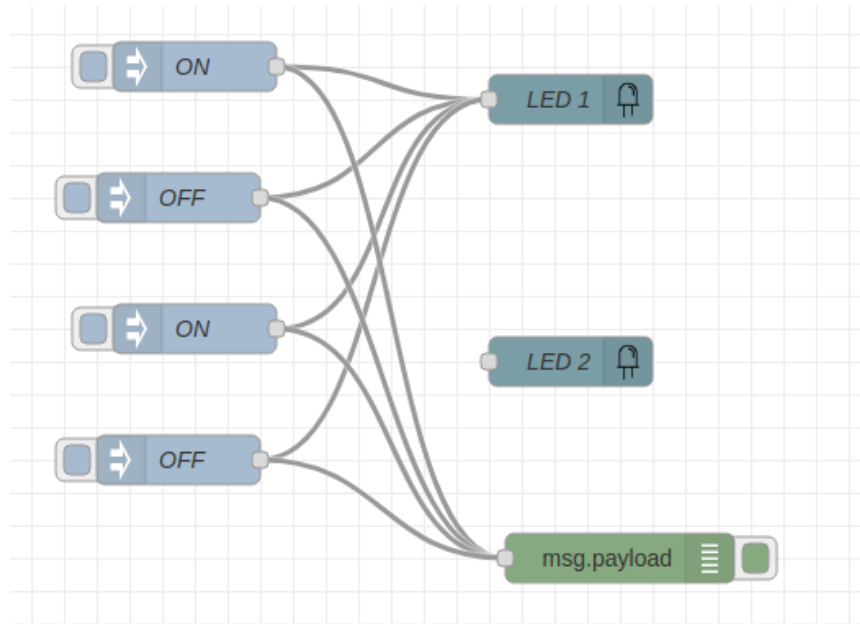


Figure 4-10: Capturing Messages

Make sure all the messages going to the LKit LED node are what you expect.

What have you learned from this exercise? If you did not know this before then you now know that you can connect as many wires as you want to a single input node as long as the wires all come from the outputs of other nodes. Good to know, important to know, don't forget.

### One Output to Many Inputs

Before we leave this simple system behind, let's see if it is possible to connect a single output to the input of more than one node. Do you think this is possible? How do you think it will behave?

Try this set up.

- Open a new tab
- Drag in one LKit LED node and drop it on the right side of your flow.
- Open the LKit LED node and make the following edits:
  - Set the Name Field to LED 1
  - Set the LED field to 1
- Now make three more copies of this node one below the other
- Edit Each of these nodes so that it controls a different LED and just to make things clear, change the name of each LED node to be the number of the LED
- Now copy the ON and OFF Inject nodes from your previous flow (Figure 4-10).

## Chapter 4 LEDs - Preliminary (V0.0)

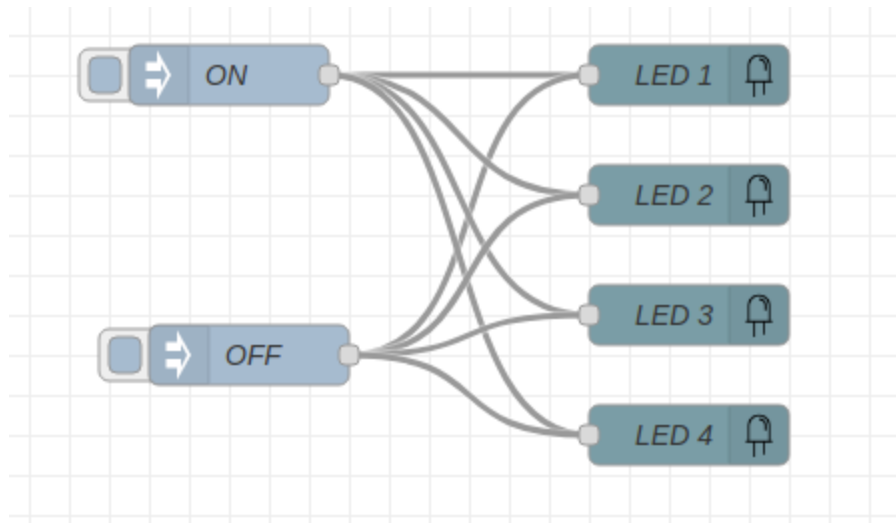


Figure 4-11: A Node Controls Multiple Nodes

Once you have a flow that looks like the figure above **D...E...P...L...O...Y I...T...!**

Now what do you think will happen if you click the ON Inject node? Try it. Was it what you expected or are you totally confused? Now click the OFF Inject node. Did it work the way you expected?

### LED Node - Group Mode Operation

Now that you have the “single” mode operation down it is time to explore how “group” mode works. In this mode your LED Node will control all four LEDs at once. This is handy if you want to show a binary number or maybe a moving meter, like the LED sound level meter on a piece of audio equipment.

When you select group mode for the LED node the way you use the *msg.payload* property changes. In “single” mode the value of the *msg.payload* is either 0 or 1 depending upon whether you want the selected LED to be on or off. However, in “group” mode the *msg.payload* is a number between 0 and 15 which and is the decimal representation of a [binary number](#) to be displayed. The number zero means all LEDs are off and the number 15 means all the LEDs are on. On the Learning card GP4 LED is the least significant bit of the binary number and the GP1 LED is the most significant bit. This means that if you look at the board edge in GP4 will be on the righthand side and GP1 will be on the left. This is the natural way to look at a binary number. Of course, everything will be backwards if you look at the LEDs through the top of the board from the other edge.

Here is a table showing how the decimal number in the *msg.payload* property translates to a particular LED pattern as you view it from the edge of the board.

<i>msg.payload</i>	LED Pattern
0	○ ○ ○ ○
1	○ ○ ○ ●
2	○ ○ ● ○
3	○ ○ ● ●
4	○ ● ○ ○
5	○ ● ○ ●

## Chapter 4 LEDs - Preliminary (V0.0)

6	○ ● ● ● ○
7	○ ● ● ● ●
8	● ○ ○ ○ ○
9	● ○ ○ ○ ●
10	● ○ ● ● ○
11	● ○ ● ● ●
12	● ● ○ ○ ○
13	● ● ○ ● ●
14	● ● ● ● ○
15	● ● ● ● ●

As is tradition, at least in this tutorial, whenever something new comes along we will begin with a simple flow so that you can check your understanding. With that in mind set up a simple flow like the one in the figure below (which you have done before)

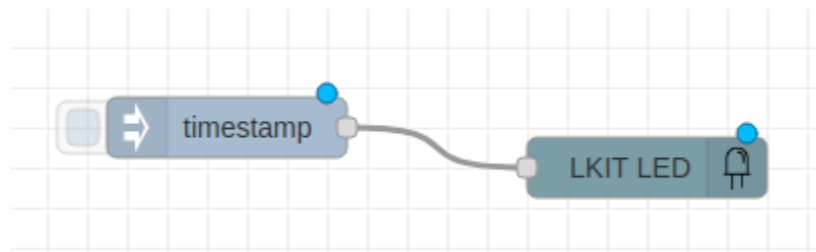


Figure 4-12: LED Group Mode - starting flow

Now, edit each node in this basic arrangement...

- Edit the LED Node (double click on the LED Node)
- Change the node name to “ALL LEDs” so that you know it is different from what you did before.
- Change mode to “group”
- Click DONE.
- It should look like this:

## Chapter 4 LEDs - Preliminary (V0.0)

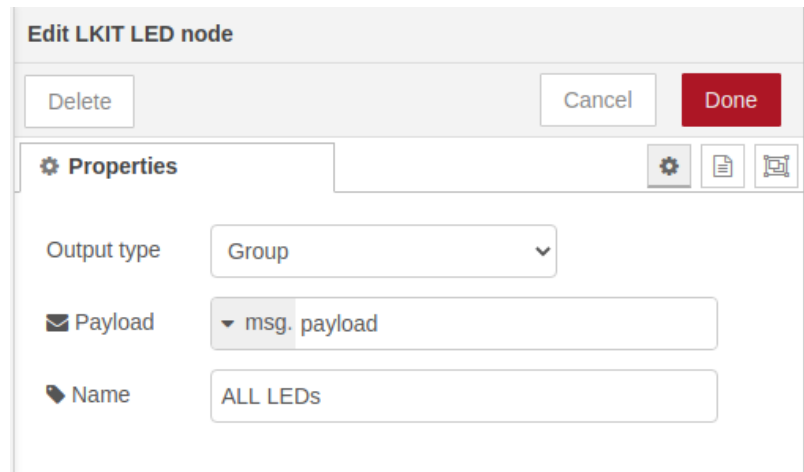


Figure 4-13: LED Node Edit

- Edit the Inject node (double click on it)
- Delete the *msg.topic* rule (click the little X beside the rule)
- Change the *msg.payload* to:
  - Type of number (select it from the drop down)
  - A payload value of 5
- When you are done your edit window should look like this:

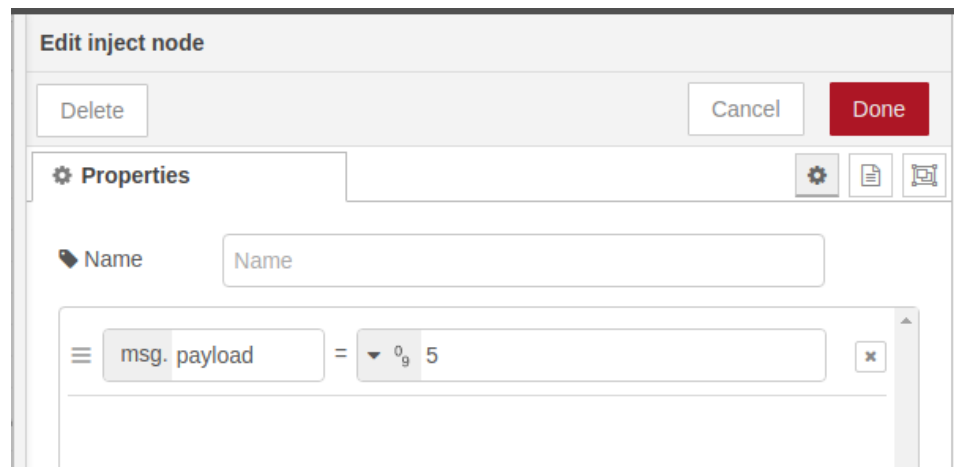


Figure 4-14: Inject Node Edit

- Click DONE.

**DEPLOY IT!** Once you do you should see this flow:

## Chapter 4 LEDs - Preliminary (V0.0)

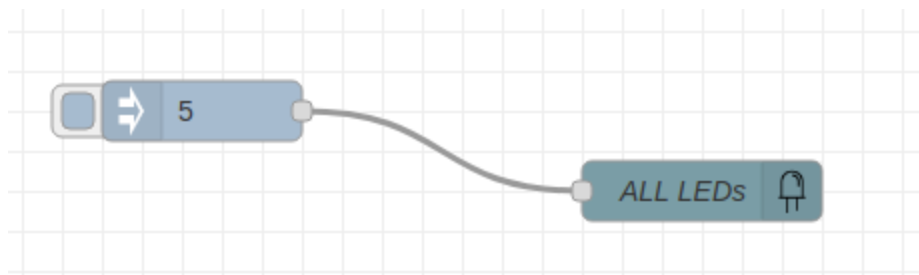


Figure 4-15: Group Mode Example - Completed Flow

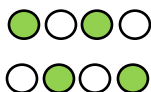
Now click on the tab of the inject node. What pattern did you get in the LEDs? Is it the same as the pattern shown in the table for the number 5? If not, it is time to debug. You know what to do... check the configuration of each node in the node edit window. If that looks good attach a debug node to the output of the Inject node and make sure the *msg.payload* is 5 and that it is numeric.

Once everything is working properly, open up the Inject node again and change it to another number, like 10 or 3 or 8. Deploy it and check the results when you click the Inject node. Compare your results to the table. If this works then try adding another Inject node to the input of the LED node. Configure this node the way you did above, but with a different number. Deploy it (always). Now click one Inject node and then the other. Does the light pattern change the way you expect? Add one more Inject node and configure it with a different number. Deploy it. How does it behave?

### Simple LED Puzzles

Time to demonstrate<sup>8</sup> your prowess in using the group mode for the LED node.

LED Puzzle # 1 - **Warning Lights** – You’ve seen them on the highway when the patrol car pulls over a miscreant<sup>9</sup>, hopefully not you. There are four lights in a row on top of the patrol car and they flash back and forth between these two patterns:



(Yeah..., as if tiny flashing green LEDs on the Learning card are going to get anybody’s attention, but ya gotta use whatcha got on hand.)

Your task: using two inject nodes and one LED Node set up a flow so that when you click one inject node you get one pattern and when you click the inject node you get the other pattern. In another few pages you will learn how to automate this, so practice on this little example now.

Once you finish this puzzle build a flow to do the same thing, but this time do not use the LED group mode but rather use the LED single mode. This means you will need two Inject nodes and four LED nodes. Now, ask yourself this: “Self, I need to turn ON some LEDs and turn OFF others. How in the world

<sup>8</sup> Demonstrate – if the word had been “test” then your hands would have become sweaty, your lips would have started trembling and your brain would have frozen up like a block of ice at 74 N,94 W on December 21st.

<sup>9</sup> Miscreant – Yes, it will be on the SATs.



## Chapter 4 LEDs - Preliminary (V0.0)

am I going to do that with one Inject Node?" (Hint – you might need to use one of the other nodes you have learned about, like... the Change<sup>10</sup> node.)

Here is an example of how you might create the same function, but individual LED nodes in single mode. Can you figure out how this works and build your own copy? How does the complexity of this arrangement compare to using the group mode?

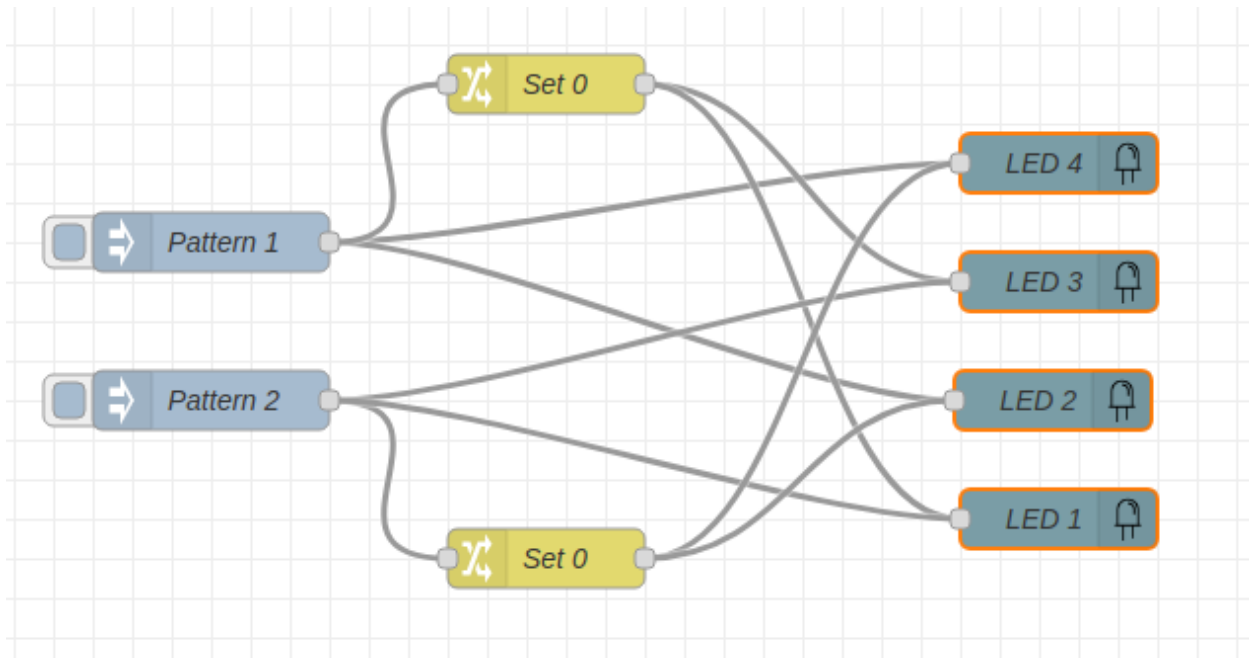


Figure 4-16: Flashing Light Example in Single Mode

**LED Puzzle # 2 - Thermometer** – Well, not quite, but it is a starting point and later, when you are running, you will turn it into a real one. With four LEDs you can make a little bar display that will display the magnitude of the value. Later you will use a real input source to control this mini meter, but for now you will just be demonstrating the concept using Inject nodes. You will want to set things up so that you have five Inject nodes arranged from top to bottom. When you click the particular node, you will get the pattern (seen from the edge) shown in the table below.

---

<sup>10</sup> If you have some experience in logic circuit design, you will see that you can use a Change node as a kind of inverter. There are several ideas from circuit design that apply in the Node-RED environment because messages are a bit like signals. Just remember that Node-RED is a different kind of beast and something that works well in logic design might be a little weird to use in Node-RED.

## Chapter 4 LEDs - Preliminary (V0.0)

Inject Node	Pattern
A	●●●●
B	●●●○
C	●●○○
D	●○○○
E	○○○○

Here is what your flow might look like, but because you are now an expert in using the Inject and LED nodes you will know how to configure these nodes without being given the details.

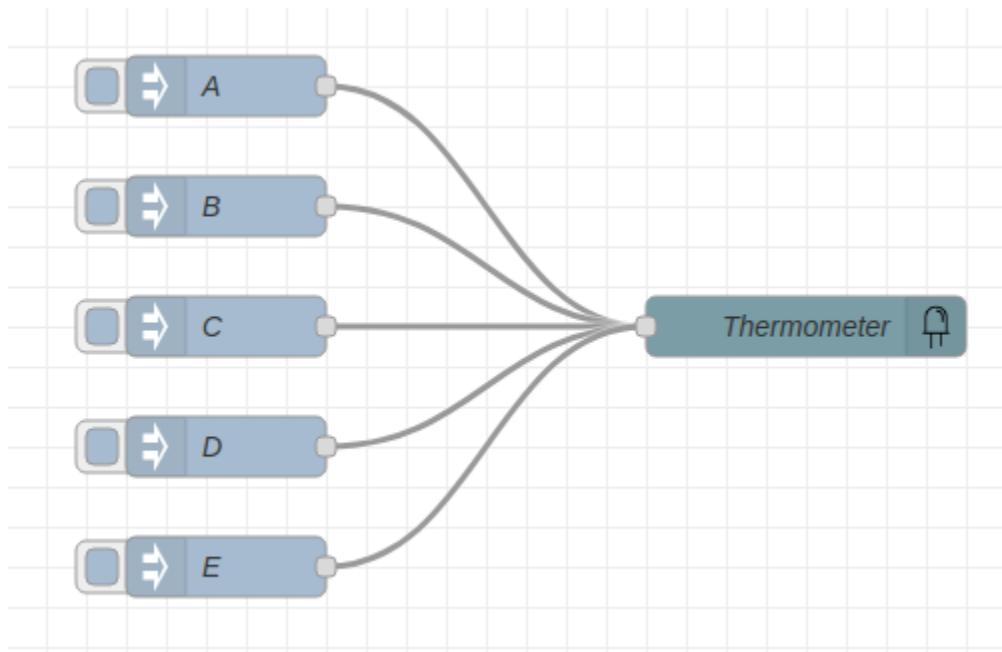


Figure 4-17: Simple Thermometer Flow

Okay, now you have built a simple demonstration. How about extending it so that the display shows when the temperature is within a given range? In a few more chapters you will find out how to measure temperature, but for now just enter the temperature using number using an Inject node. Yes, it is clunky, but it's a start.

When you click the Inject node you are going to generate a message with a number representing the temperature. Your flow should convert the temperature into an LED pattern indicating the range the temperature falls into according to this table:

Temperature	Pattern
>90	●●●●
70-90	●●●○
50-70	●●○○
32-50	●○○○
< 32	○○○○

## Chapter 4 LEDs - Preliminary (V0.0)

To get you started here is what the flow should look like:

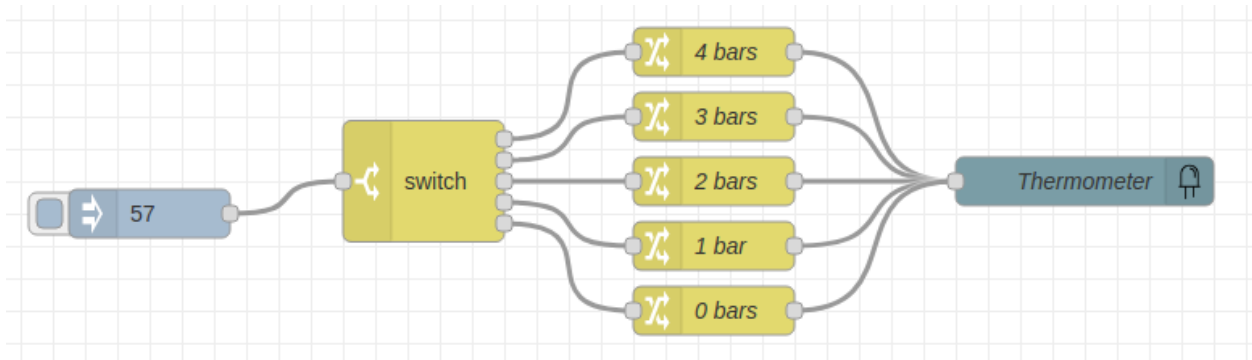


Figure 4-18: Thermometer Flow with Switch Node

You will enter a temperature by putting it in the Inject node. The Switch node will figure out the range and the nodes labeled “0 Bars” to “4 Bars” are change nodes that will control the LEDs. You know how to configure the Inject, Change and LED nodes. Here is the configuration of the Switch node:

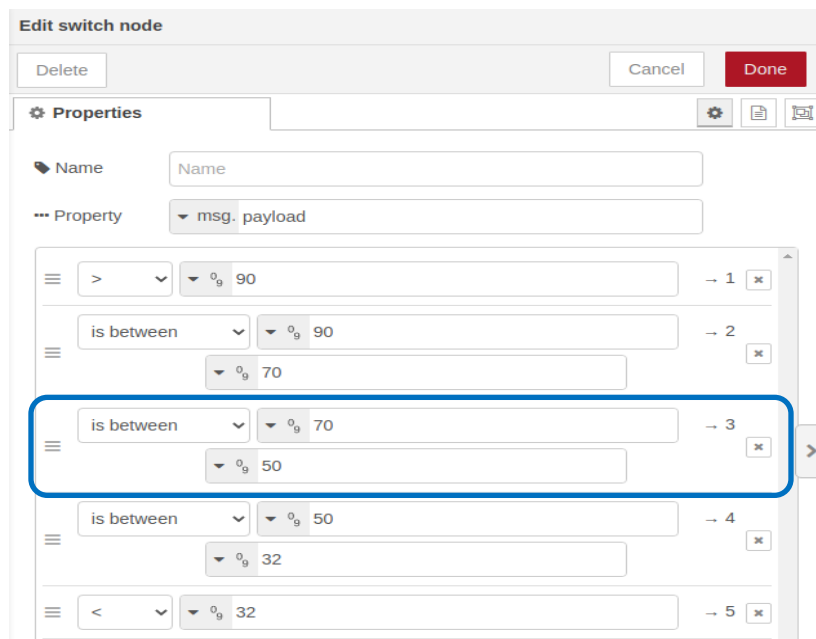


Figure 4-19: Thermometer Flow Switch Node Configuration

Your previous use of the switch node was very simple (see Chapter 3). This configuration is more complex. The message you are sending from the Inject node is a temperature value. The Switch node is going to figure out which range the temperature is in and then pass the message to one (and only one) of the Change nodes that will in turn generate a message to the LEDs.

## Chapter 4 LEDs - Preliminary (V0.0)

There are five rules in this Switch node. A rule can have many options, which you can see if you click on the drop down, like this:

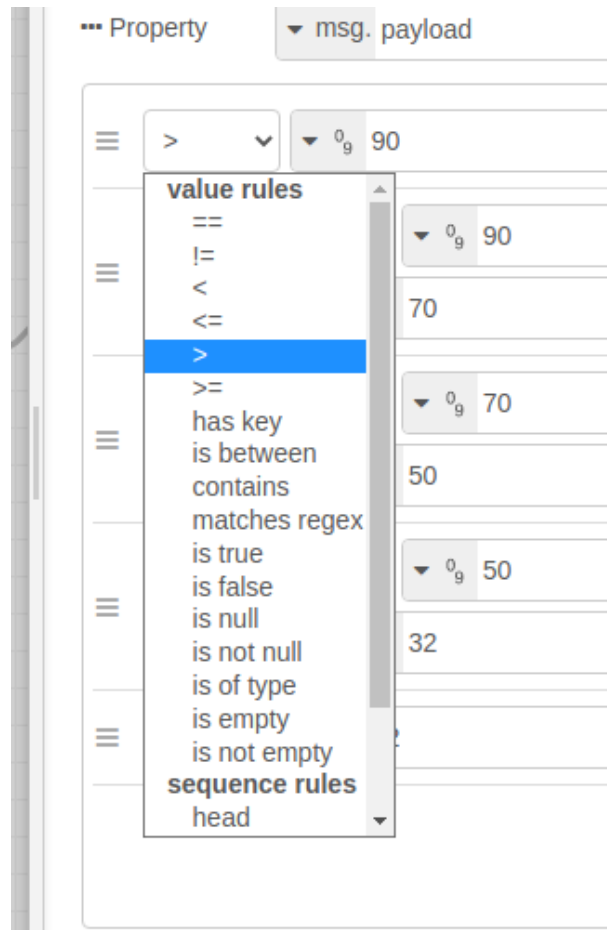


Figure 4-20: Switch Node Options

In this case you want to compare the number received from the Inject node (the faux-temperature) to different ranges. For example, the rule for output 3 (circled in Figure 4-19) checks if the temperature is between 70 and 50. If it is then the message is routed to the third output port from the top of the node. See if you can determine what rules 1 and 5 are doing?

Build up your flow based on Figure 4-18 and configure each node. Set a temperature in the inject node and please, please, please DEPLOY IT! Click the inject node. Did it work? Change the temperature in the Inject node to a different value. Remember, after you change the temperature value in the Inject node you must **DEPLOY IT** for the change to take effect. Try various temperature values. Try some negative values. Try five different values each of which should produce one of the five LED patterns from the table.

Now if only you could measure a real temperature value. Be patient you will get there in Chapter 7. However, later in this chapter you will learn how to pick up the outside temperature over the Internet.

Enough with the crawling, time to stand up and start...

## Chapter 4 LEDs - Preliminary (V0.0)

### Walking with LEDs – Pushbuttons, Delays, Subflows and Loops

Inject nodes are great for debugging and experimentation, but not really that great for real world use. After all, if Mr. Nibbles breaks out of his cage, we really can expect him to click the “Cage Door Open” Inject node on the way out. How about replacing the Inject nodes in some of the flow you developed above with the pushbutton from Chapter 3? Not super exciting, but it’s a start.

What was that??? You hear a little voice in your head saying start simple? Good, the subliminal messages from dozens of whispering hamsters inhabiting this tutorial are finally getting through. Start Simple.

What could be simpler than one push button and one LED? But before you press on, it is time for a commercial break brought to you by Rodent World – A Store for All Your Hamster Need!

---

**Engineering Tip # 8 - Cook Up a Spec!** One of the seductive things about all the wonderful tools at your disposal is that it is very tempting to just jump into build a flow and think things out on the fly. However, resist the Sirens’<sup>11</sup> call, slow down and think carefully about what you want to do. For simple things, like this flow, you can probably figure everything out with a quick sketch on the back of an old gum wrapper. However, in the larger world of engineering, where you are working with colleagues, it is always helpful (nay, necessary) to have a written specification for what you want to do. Mostly, this is so everybody is on the same page(s), so to speak. A good specification tells the reader how everything is going to behave, including all the strange usages that should not happen but probably will.

Returning now to our regularly scheduled broadcast...

---

How about a quick sketch of what you want to do in this flow? Like this perhaps:

- The name of the flow will be “Button to LED”
- Pushing the button should turn on LED 1.
- While you hold the button, down LED 1 should stay on
- When you let go of the button LED 1 should turn off
- As long as you are not pushing the button LED 1 should remain off.
- Whatever happens with LED 1 the other LEDs should not change.

Not bad, seems to cover all the cases. Well, not quite... What about when you first turn on the system or start up Node-RED? Should LED be ON or OFF? Probably it should be OFF, but let’s ignore this for the time being because you do not, yet, have the tools to set the startup conditions.

With this sketch in mind, it’s time to get down to work.

- Be smart. Save your current flows.

---

<sup>11</sup> Sirens – No, not the noise makers on top of police cars and ambulances. Rather, the [Sirens](#), who were mythological creatures in the ancient Greek world whose beautiful singing would lure sailors to their death. Listen to those thoughts about how easy it is to build a flow without any planning and your flow will lead you down to the dark pit of total confusion. Take time to plan, Sailor!

## Chapter 4 LEDs - Preliminary (V0.0)

- Then open a new flow and name it something like “Button to LED” per the specification.
- Delete all the other flows so that you are only working with one flow<sup>12</sup>.
- Drag and drop the LED node on the right side
- Drag and drop the LKIT Button node on the left side.
- Connect them with a wire.

Now pause and think for a minute... You know how the pushbutton node works and what sort of messages it generates. You also know how the LED node responds to messages. How are you going to configure your nodes? For the LED node your first question might be: “Am I going to use single mode or group mode?” How are you going to decide? Well, it sure sounds like a job for a node in single mode because in the specification only one LED is changing and all the other LEDs stay the same. So...

- Open the editor for the LED node and set the mode to single
- Name the node “LED 1” or something you think is descriptive.
- Set the node number to 1

The Edit window for the LED node should look something like this:

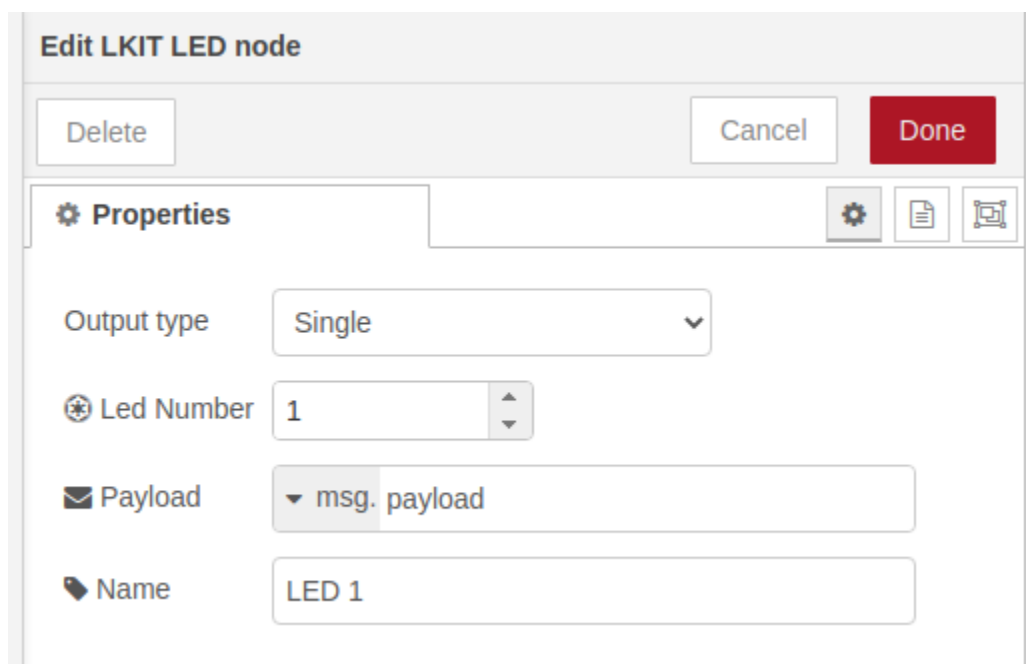


Figure 4-21: LED Node Edit in Single Mode

<sup>12</sup> Always be careful with other flows you have open because when you push the button there might be a node on some other flow that is “listening” to the button. If you click on a flow tab and look at the bottom you will find a very tiny button that allows you to disable the flow. Once you do your flow will look a bit sepulchral because everything will be grayed out. This is a useful feature allowing you to break down your testing and debugging, or keep an old flow around for reference.

## Chapter 4 LEDs - Preliminary (V0.0)

- What does the LED node want to receive as a message? If you are not sure you can select the LED node and open the Help tab in the sidebar (the little book icon). Right... The payload controls the state of the LED: number 1 means ON and number 0 means OFF.
- Sounds good!
- Your flow should look like this:

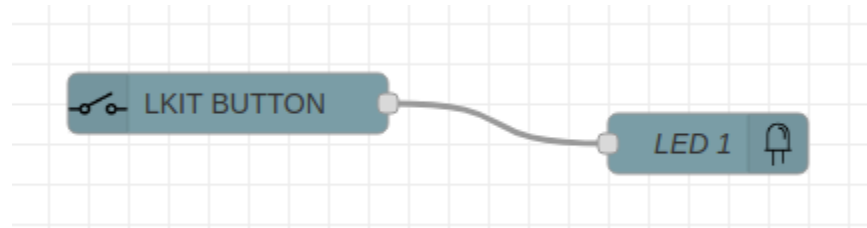


Figure 4-22: Button to LED - Initial Flow

- DEPLOY or FAIL!

Testing, testing, testing: Push the button. What happens to LED 1? Say whaaaat! It turns OFF? Check your specification. When you push the button LED 1 should turn on. Opps. Stop for a moment and think: “Hmmmmm... how did that pushbutton work? Go back to Chapter 3 (or select the Button node and select the Help Tab). Yes, indeed, when you push the button the message payload has the number 0 in it, which turns off the LED. If you don’t trust the documentation (and you probably should be wary) you can use a Debug node to check the output of the Button node.

How are you going to fix the problem? Check the palette out. Maybe the change node will work.

- Insert a Change node between the Button node and the LED node.
- It looks like you are going to have to change the number 0 to a 1 and the number 1 to a 0. Your first impulse is probably to write two rules in the change node. One to search for a 0 and replace it with a 1 and another rule to search for a 1 and replace it with a zero. How confident are you that this will work? Did you read the footnote back on page 48? Okay, you are uber-confident so give it a try.
- Double click on the change node.
- Change the name to something like “1 to 0; 0 to 1” because that is what you want to do.
- Fix the first rule so that it searches for the number 1 in the payload and changes it to the number 0. (You’ve done this before in Chapter 3)
- Add another rule and set this one to change the number 0 to the number 1 like you did above.

The edit window in the change node should look like this:

## Chapter 4 LEDs - Preliminary (V0.0)

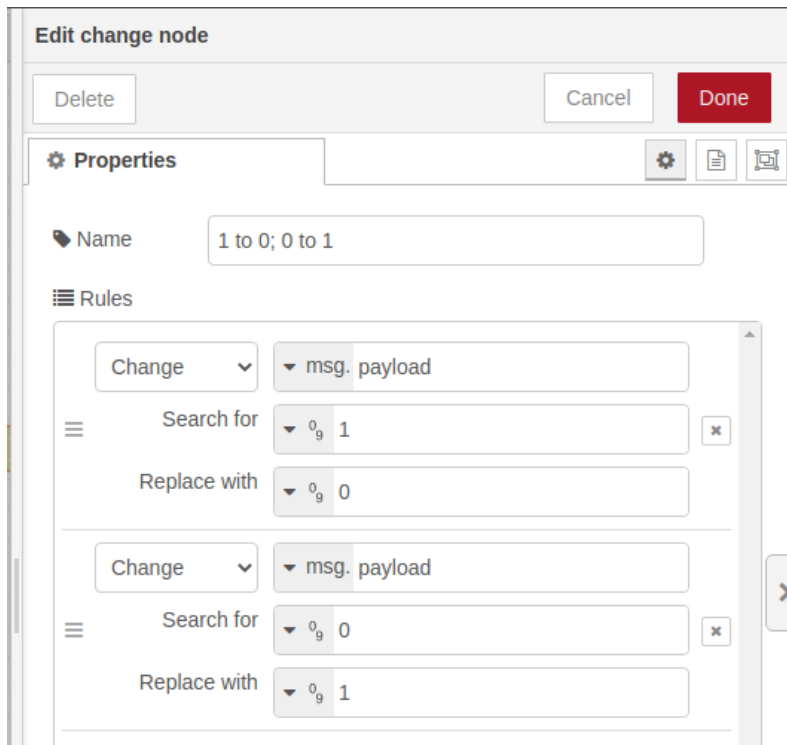


Figure 4-23: Change Edit Window - First Attempt

- Click DONE and see if your flow looks like this:

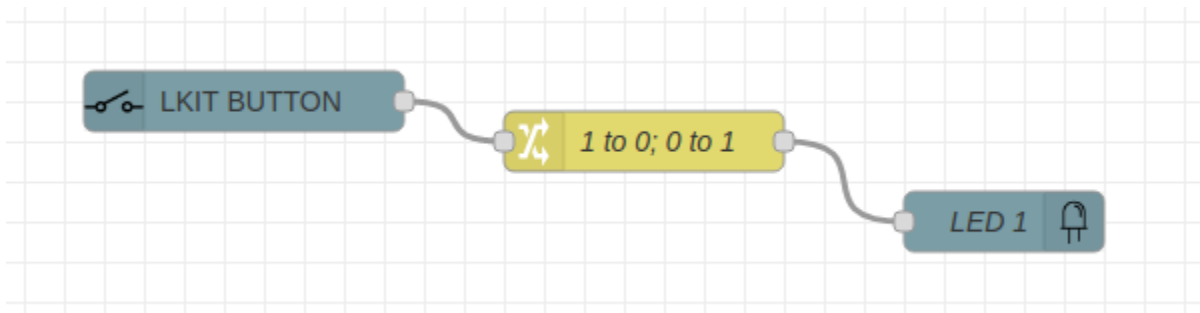


Figure 4-24: Button to LED Flow - First Attempt

- **Deploy** that flow and see what happens.

Did it work? Probably not. Depending upon the order in which you wrote the rules in the Change node you either turned LED 1 on one time or turned it off one time and then it was stuck. You were drawn into the whirlpool by the Siren call of Expediency, and now you are in the Underworld of Doubt.

What went wrong? The answer is that the Change node may not behave exactly like you thought it would (or should). Here is the problem: rules in the Change node are executed one at a time from the



## Chapter 4 LEDs - Preliminary (V0.0)

top of the list to the bottom of the list. Importantly, every rule is executed. In this flow each rule is applied to the `msg.payload` property and changes that property. Effectively, the Change node grabs the payload, puts it on the operating table and knocks it out with some anesthesia<sup>13</sup>. Then the Change node brings out a scalpel and goes to work. Let's listen in.

**Dr. Change Node:** "Okay, I am now applying the first rule. No problem, cut out this nasty number 1 and stitch in a 0. Alright, second rule: look for a 0 and change it to a 1. Looky here! there is the 0 I just put in, but the second rule says to cut it out and stich in a 1. Great. I just follow the rules. Top to bottom. Out with the 0 and in with the 1. Nice job! All done. You can close Nurse."

**Nurse:** "Yes Doctor, but that sure was a pointless operation"

**Dr. Change Node:** "We just follow the rules here, all the rules."

All that work and all that happened was that the payload is still the same except it is a little groggy from the anesthesia. Beware, this is how the Change node works. Every rule is carried out, in order from top to bottom on the payload and if you are not careful you can undo something you did in a previous rule.

Here is one way to fix the problem. Use three rules as shown below:

The screenshot shows the 'Edit change node' dialog box. At the top, there are buttons for 'Delete', 'Cancel', and 'Done'. Below that is a 'Properties' section with a gear icon and a text input field containing '1 to 0; 0 to 1'. The main section is titled 'Rules' and contains three rule configurations. Each rule has a 'Change' dropdown set to 'msg. payload', a 'Search for' field, and a 'Replace with' field. The first rule searches for '1' and replaces it with '0'. The second rule searches for '0' and replaces it with '1'. The third rule searches for '2' and replaces it with '0'.

Figure 4-25: Button to LED Edit - Second Attempt

<sup>13</sup> Look it up and while you are at it look up the origin of the word.

## Chapter 4 LEDs - Preliminary (V0.0)

Here's what is going to happen. The first rule is checked. If the payload contains a 0 then nothing happens. However, if it contains a 1 then that is converted to the number 2. At this point the payload will contain either the number 0 or the number 2. Now the second rule is checked. If the payload contains a 2 then nothing happens, but if it contains a 0 then that will be changed to a 1. At this point the payload has either the number 1 or the number 2 in it. Finally, the last rule searches for the number 2 and converts it to a 1. Of course, if there is no number 2 in the payload nothing happens. Read this over a few times and make sure you understand it because you will need it later.

- Go back and fix the rules so that the Change node edit window looks like this.
- Click DONE
- *Deeeemploy it!*

Test your flow. Does everything work like the specification says? If so, pat yourself on the back and have a cup of coffee. Otherwise... no coffee for you. Well, you know what to do by now: get a Debug node off the shelf and see what messages you are sending, read the Help notes, think, take a break, think some more.

By now you are thinking: That is one hard to understand Change node. There has gotta be a better way". Yes, there are other better approaches, although some of them would require Java programming. Here is one solution that does not require any programming it uses three nodes instead of one, but it is better in the sense that it is easier to understand.

Suppose you took the messages from the pushbutton node and separated them using a Switch node. The messages with the number 0 would go one way and the messages with number 1 would go the other way. Now that the messages are traveling on separate wires you can change each one individually and then merge the two results back into one message stream to send to the LED. Here is what this would look like as a flow:

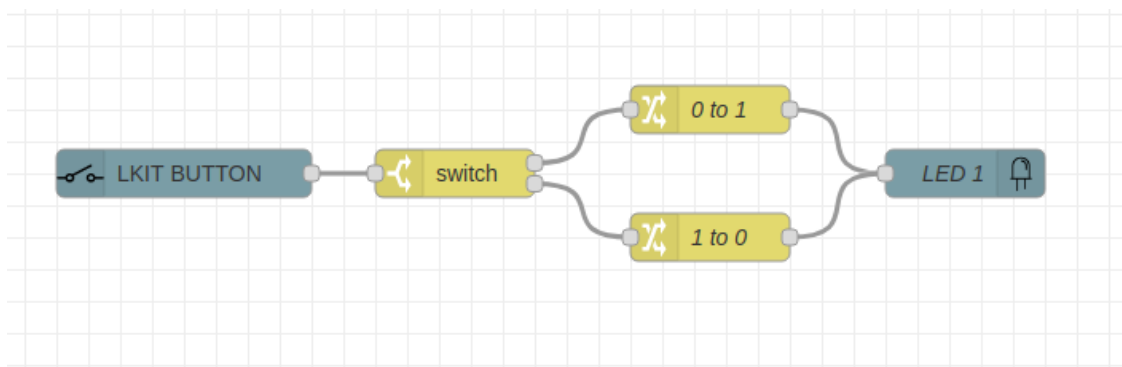


Figure 4-26: Button to LED with Switch Node

In the figure above each of the Change nodes (the yellow nodes on the right) have been given a name that reflects what it does (e.g. one change a 0 to a 1 and the other changes 1 to a zero). At this point you

## Chapter 4 LEDs - Preliminary (V0.0)

should know enough to put this flow together and configure the Switch and Change nodes. Only one rule is needed in each Change node because the nodes only receive a specific payload value.

Here is another way to think about what is happening. Imagine once more that the messages are in little freight trains carrying a payload. In this flow every time the pushbutton is pressed or released a train leaves the pushbutton node with either a number 0 or a number 1 in the payload freight car. When it reaches the Switch node those trains with a 0 (zero) are going to be switched to the upper track and those with a 1 (one) are going to be switched to the lower track. Now the Change nodes only need to deal with one value. The upper Change node, named “0 to 1” only has to change the payload value from a 0 to a 1. Easy. The lower Change node just changes a 1 to a 0. Also, easy. Finally, the tracks (the wires) join together, and all the trains proceed to the LED node.

You have the knowledge now to build the flow in Figure 4-26 and to configure the Switch and Change nodes. Give it a try<sup>14</sup>. Once you get it working come back, celebrate briefly and continue reading.

If you are really confused and discombobulated with the flow above here is how the Switch and Changes nodes are configured.

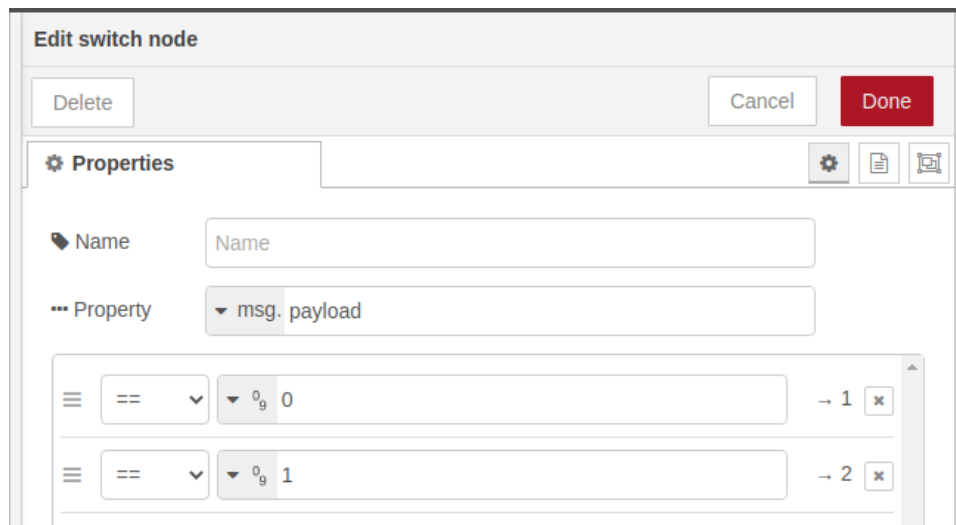


Figure 4-27: Button to LED - Switch Node Edit

---

<sup>14</sup> For the Change node you can use a “change” rule, but you know that each change node only receives one kind of payload, so you can simply use a “set” rule to simply set the output payload to the opposite of the expected input payload.

## Chapter 4 LEDs - Preliminary (V0.0)

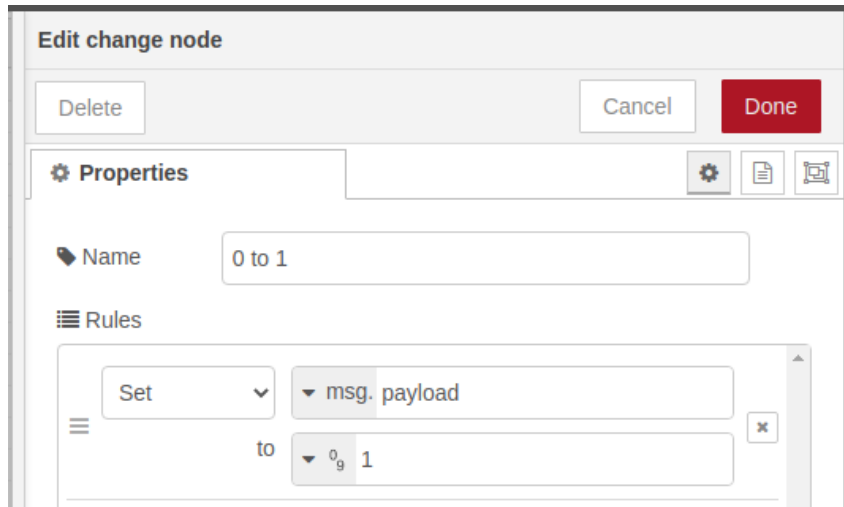


Figure 4-29: Button to LED - 0 to 1 Node Edit

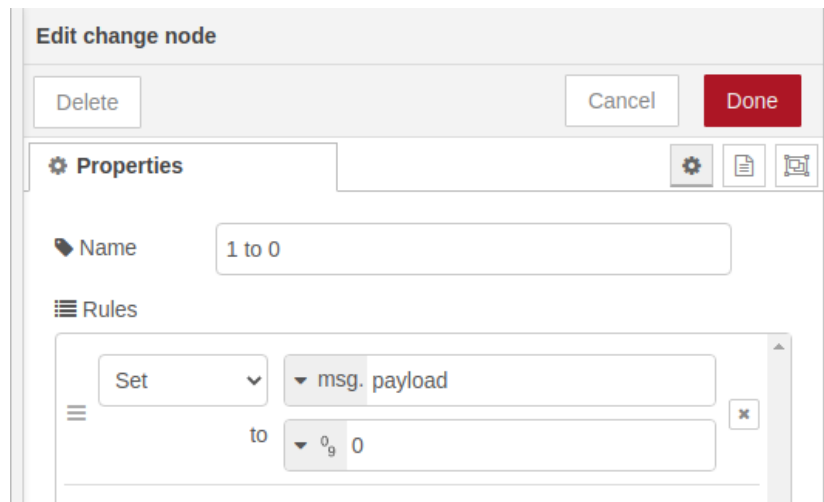


Figure 4-28: Button to LED - 1 to 0 Node Edit

Make sure that this flow is working properly because it is going to be used in the next example.

Subflows – making things simpler since 2016<sup>15</sup>

Look at the flow in Figure 4-26. My Gosh, it is only a pushbutton and an LED but there are three more nodes just to flip the value of the payload in the message. Messy, complicated, and confusing when all you are trying to do is to make the LED light up when you push the button. Wouldn't life be much easier if a node to flip the value already existed, and maybe you can imagine a use for it somewhere in the future. Even better, wouldn't it be nice if you could build this node yourself. Well, you can by using *subflows*.

<sup>15</sup> 2016 – when IBM donated their Node-RED development environment to the Open Source JS Foundation.

## Chapter 4 LEDs - Preliminary (V0.0)

In Node-RED a subflow is a single node that represents a flow composed of several other nodes. Subflows offer several advantages:

- They reduce the visual complexity of your flows.
- They make your flow easier to understand.
- They allow you to encapsulate a small flow so that you can reuse it in other flows.
- They allow you to build and test a flow and then reuse it, secure in the knowledge that nothing has changed, which might not be the case if you were to recreate a simple flow from memory.

If you are familiar with conventional programming languages, you might think of subflows as something similar to (but not exactly the same as) a subroutine or a macro.

Time to look at an example, which will make the concept much clearer. Let's begin with the flow from Figure 4-26 where the pushbutton controls LED 1. Here is the figure reproduced below. The Switch node and the two Change nodes perform a simple but useful function, namely changing a 0 to a 1 or a 1 to a zero. Let's make them into a single node that changes 1 to 0 and 0 to 1.

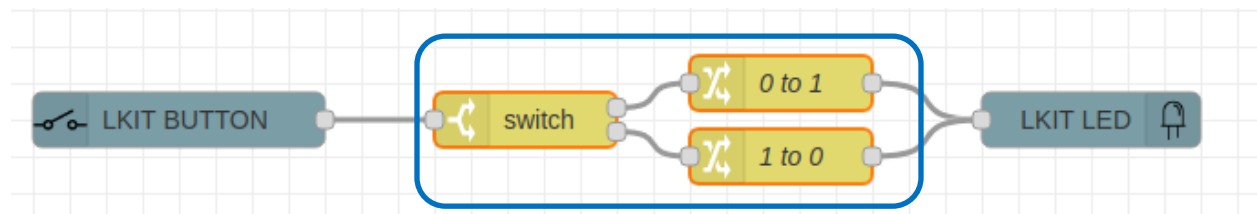


Figure 4-30: Original Button to LED Flow

Follow the recipe, please:

- Select the three nodes outlined. You can do this by holding down the Shift key and clicking on each node in turn.
- Click on the main menu icon ① (See figure below).
- Select "Subflows" ②.
- Click "Create Subflow" ③.

## Chapter 4 LEDs - Preliminary (V0.0)

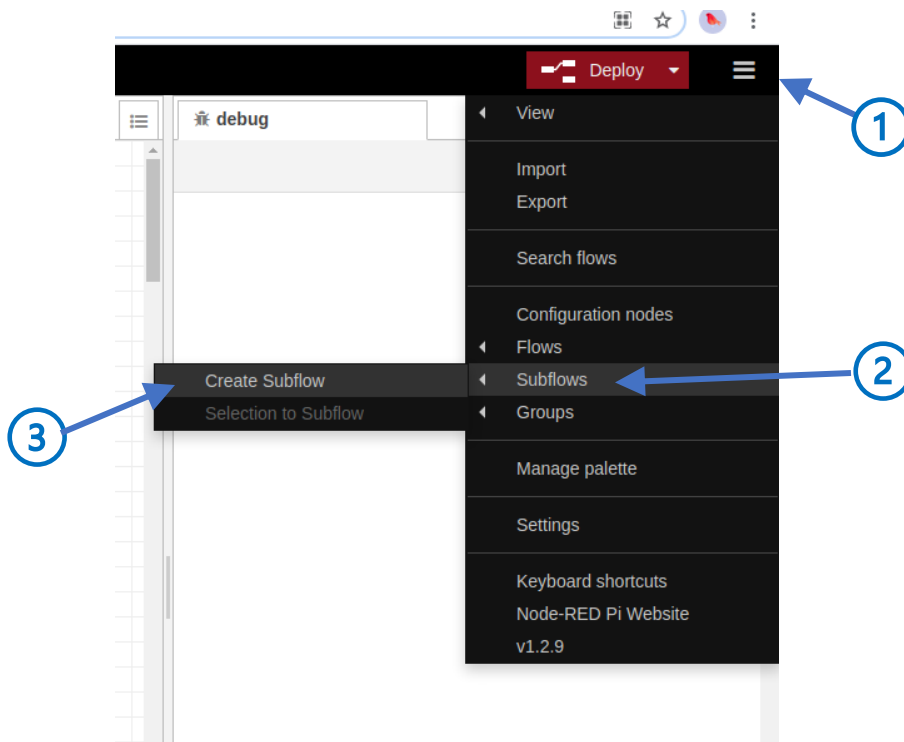


Figure 4-32: Creating a Subflow

Now you will have a new tab specifically for creating subflows like what is shown below.

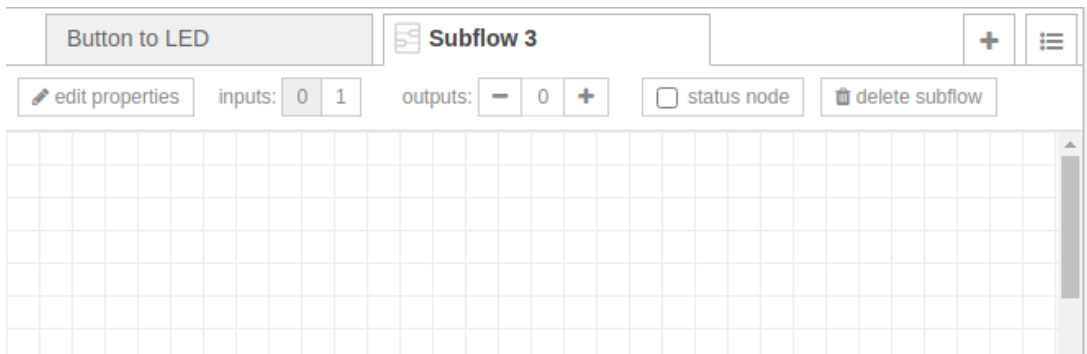


Figure 4-31: New Subflow Tab

Now use `ctl-v` to insert the three nodes you copied from Figure 4-30 into the workspace of this subflow tab. Like this:

## Chapter 4 LEDs - Preliminary (V0.0)

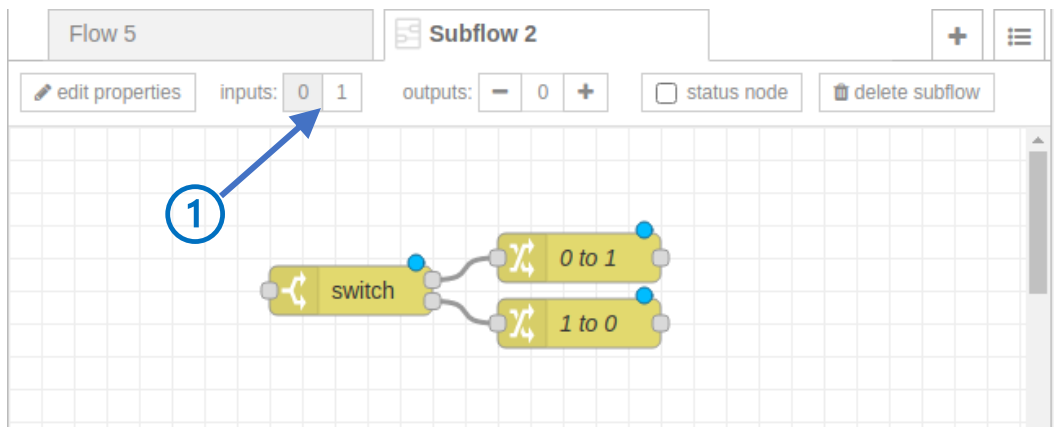


Figure 4-33: Subflow Tab with Copied nodes

The only thing left to do is to add an input port and an output port. Remember, in Node-RED a node may have zero or one input ports, but it may have as many output ports as you need, including no output port. With this in mind...

- Click on the 1 next to “inputs” at the top of tab ①. This will add an input node to the tab workspace, thusly:

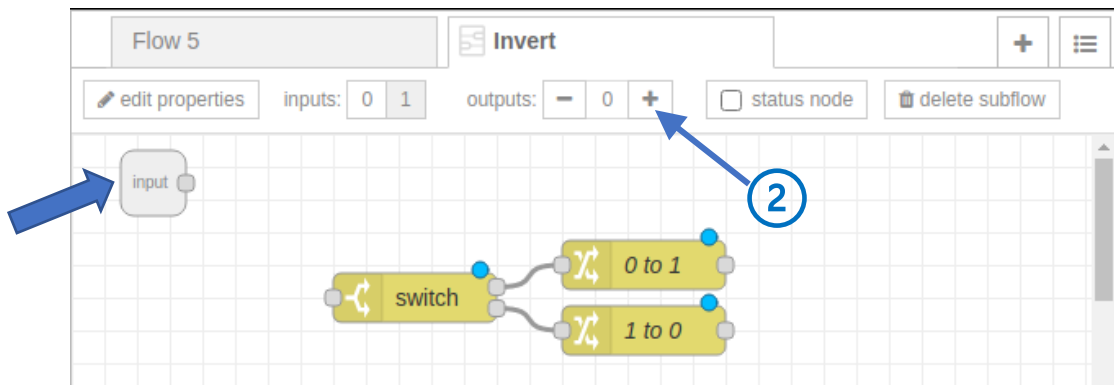


Figure 4-34: Subflow with Added Input Port (Blue Arrow)

- The subflow will have one output port. Click on the plus sign (+) by “outputs” to add one output port ②. Now you will have the following workspace:

## Chapter 4 LEDs - Preliminary (V0.0)

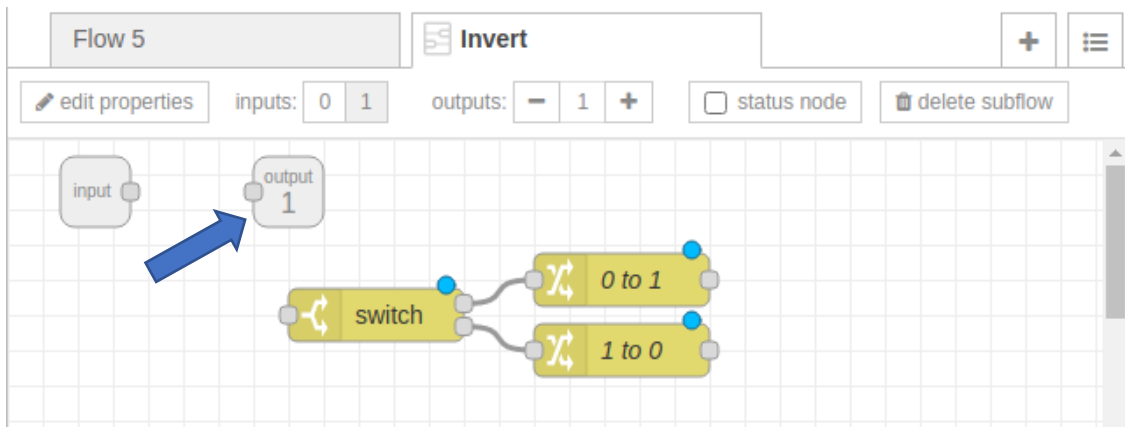


Figure 4-35: Subflow with Added Output Node (Blue Arrow)

The input and output nodes are not connected, but you have probably figure out what to do. In case, you are just getting back from the hamster farm, here are the steps:

- Connect the output port of the Input node to the input port of the switch node.
- Connect the output ports of the two change nodes (0 to 1 and 1 to 0) to the input port of the output node (got it?).
- Move the input and output nodes to a nice sensible position (outputs on the right and inputs on the left, right?)

Your subflow should look more or less like this:

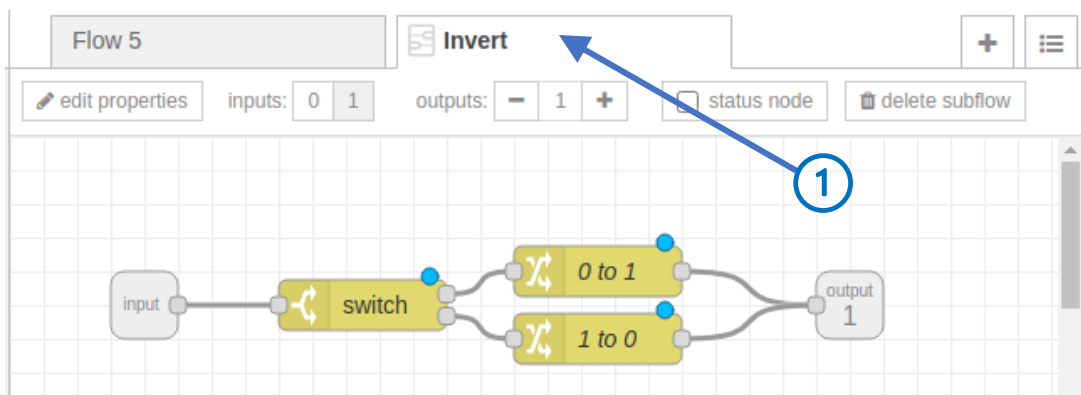


Figure 4-36: Completed Invert Subflow

The little blue dots resting so comfortably on top of the Switch and Change nodes mean you have not yet deployed the flow. Now would be a good time to:

### o **DEPLOY IT!**

Are you done? Not quite. To make your shiny new subflow useful you should really do a few more things: Give it a name, set up the Help tab and maybe perk up the appearance. Here's how.

First, give it a name. If you don't do anything then Node-RED will be happy to call it something very descriptive, like "Subflow 8". You and any future users might like something better, so...



## Chapter 4 LEDs - Preliminary (V0.0)

- Double click on the tab ①. This will open the “Edit subflow template” as below:

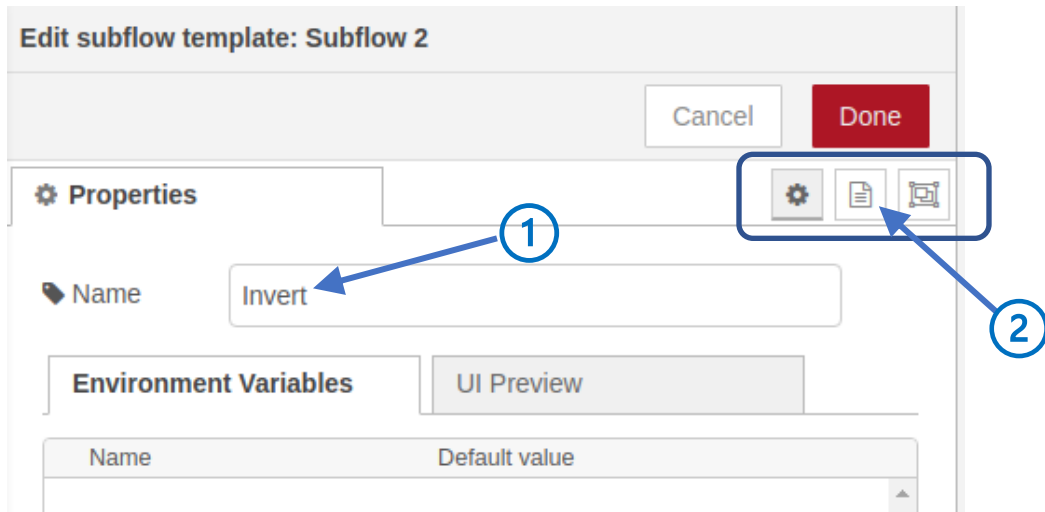


Figure 4-37: Subflow Editing - Changing the Name

- Click in the Name box and enter a name the makes sense, like “Invert” ①.
- The blue box in the figure above outlines the different tabs in the subflow edit dialog. Right now, you are dealing with the “Properties” tab.
- Click on the little document icon ② to select the “Description” to bring up the node description for editing:

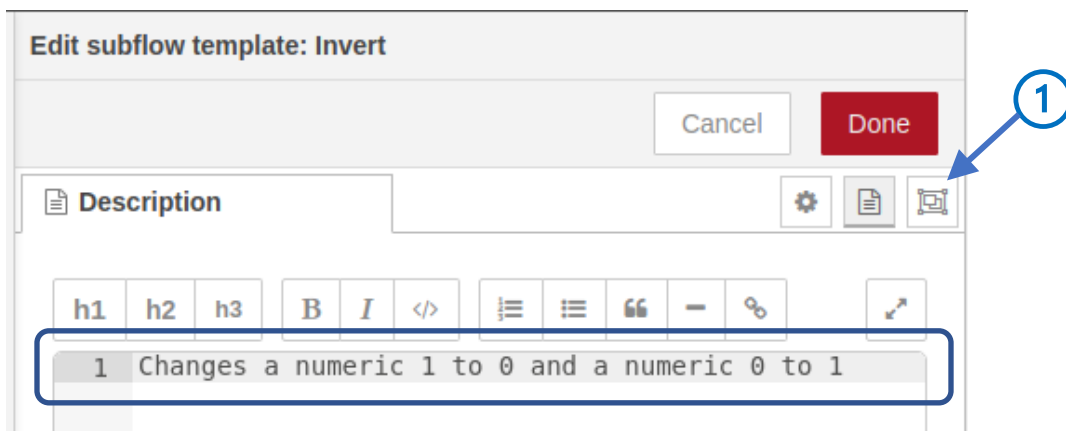


Figure 4-38: Subflow Editing - Adding a Description

- Type in a description of the node as in the blue box above. Anytime someone uses your subflow they will be able to see this description under the Help tab. Remember documentation is the Golden Rule of engineering. Help other so that they will help you, maybe.<sup>16</sup>

If you would like your node to stand out from the crowd, try changing the appearance. Do this by:

<sup>16</sup> Or as Yogi Berra allegedly said: “Always go to other people’s funerals, otherwise they won’t come to yours.”

## Chapter 4 LEDs - Preliminary (V0.0)

- Clicking on the “Appearance” tab ① to bring up the Appearance dialog in the subflow edit window as shown below:

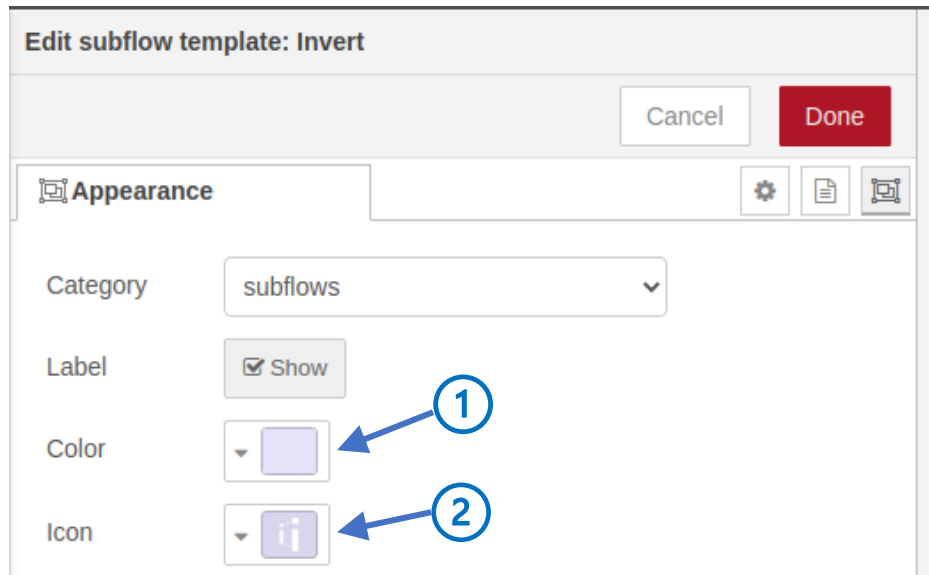


Figure 4-39: Subflow Editing - Changing the Appearance

- Use the “Color” dropdown to select a pleasing color for your subflow ①. Lavender will do.
- Use the “Icon” dropdown to add a small icon to represent your flow ②.
- Click DONE!

Yes, DONE! Your subflow is ready to use.

Go back to your original Button to LED flow (Figure 4-30) replace the three nodes in the middle with the Invert Node, like this:

- Delete the three node in the middle (Switch node and the “0 to 1” and “1 to 0” change nodes.
- Go to the top of the Palette menu and drag the Invert node between the Button and LED nodes.
- Wire up the nodes as in the figure below.
- DEPLOY IT!
- TEST IT!

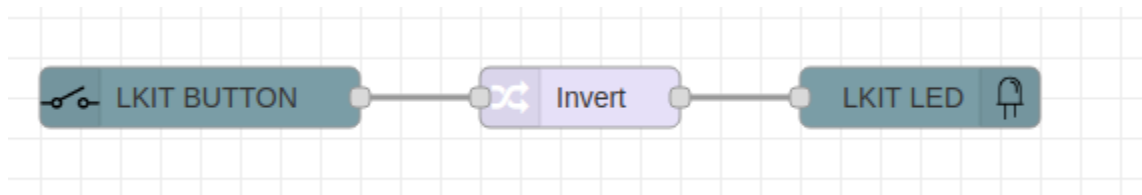


Figure 4-40: Button to LED Flow with Invert Node

Be sure to export and save your subflow for the Invert node you have just created. You will need it later.

## Chapter 4 LEDs - Preliminary (V0.0)

### Delays and Loops!

It's time to revisit the Warning Lights puzzle from above and see if you can automate it. Below is one possible solution using four LED nodes in single mode. There are no details here, but you have the knowledge and the awesome power of the Invert node. See if you can recreate this flow. Remember the point is that when you push the button one pair of LEDs (GP1 and GP3) is on and when you release the button the other pair (GP2 and GP4) is on.

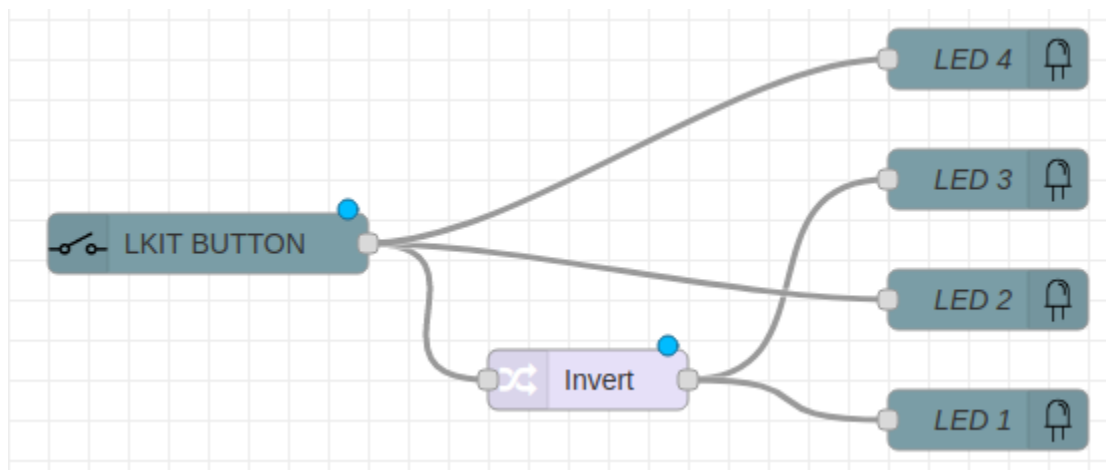


Figure 4-41: Warning Light Example

Deploy your flow (because there are little blue dots infesting the nodes) and push the button. Does it work?

Great, except that it seems a bit less than useful that someone has keep pushing the button to make the lights flash. Wouldn't it be nice if this were automated? To automate this flow all you need to do is build up a flow that generates messages with alternating 1 and 0 payload values spaced apart in time by, say, two second. If you could do this, then you could replace the Button Node with your new flow.

The first step is to see if you can generate a sequence of messages with a payload containing a value of numeric 1. A new message should occur every four seconds.

- Add a new blank tab to your workspace.
- Delete all the other tabs (of course, you will save the flows on any interesting tabs first).
- Pull in an Inject node and drop it in your workspace.
- Be brave! Open the Help file for the Inject node and see if you can understand the all the functions this node might be able to perform.

## Chapter 4 LEDs - Preliminary (V0.0)

- Now that you are thoroughly familiar with the Inject node capabilities open the Properties menu (double click on the Inject node)

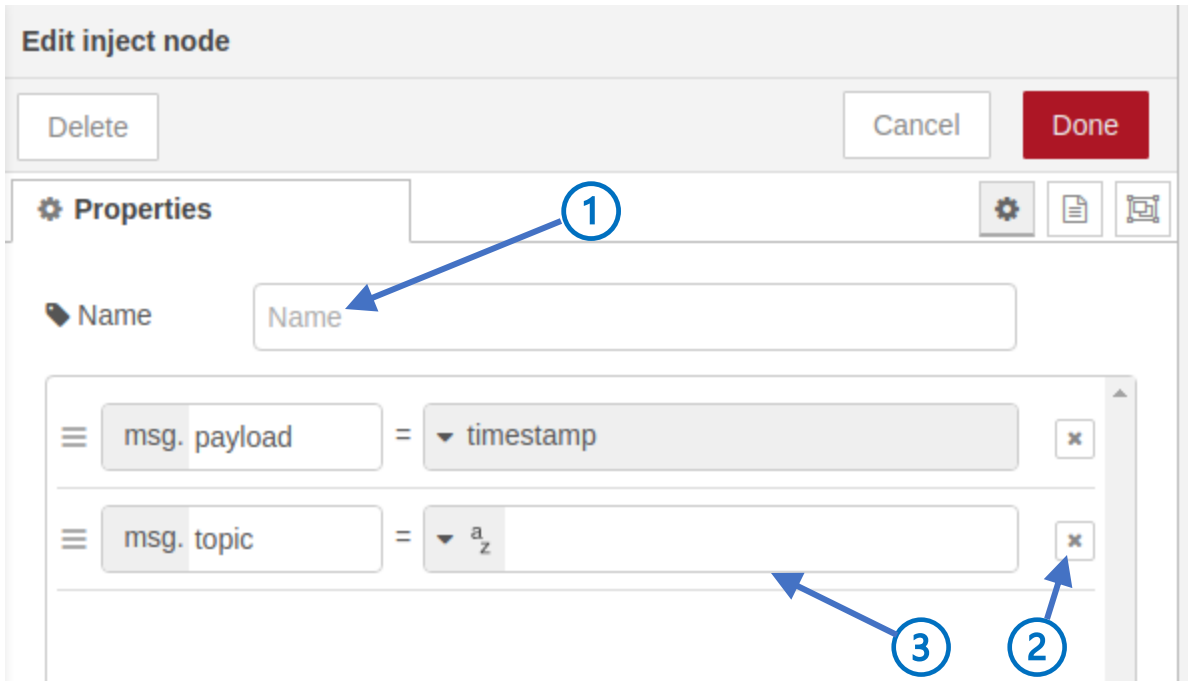


Figure 4-42: Edit Inject Node Rules

- First change the name to something descriptive, like “Generate 1s” ① (yes, you do not need to do this, but help out those who will come after you by making your flow a little less murky).
- Delete the “topic” rule ② because it is not needed.
- Change the payload to be a numeric 1 ③.

Now your Inject node edit window should look like this:

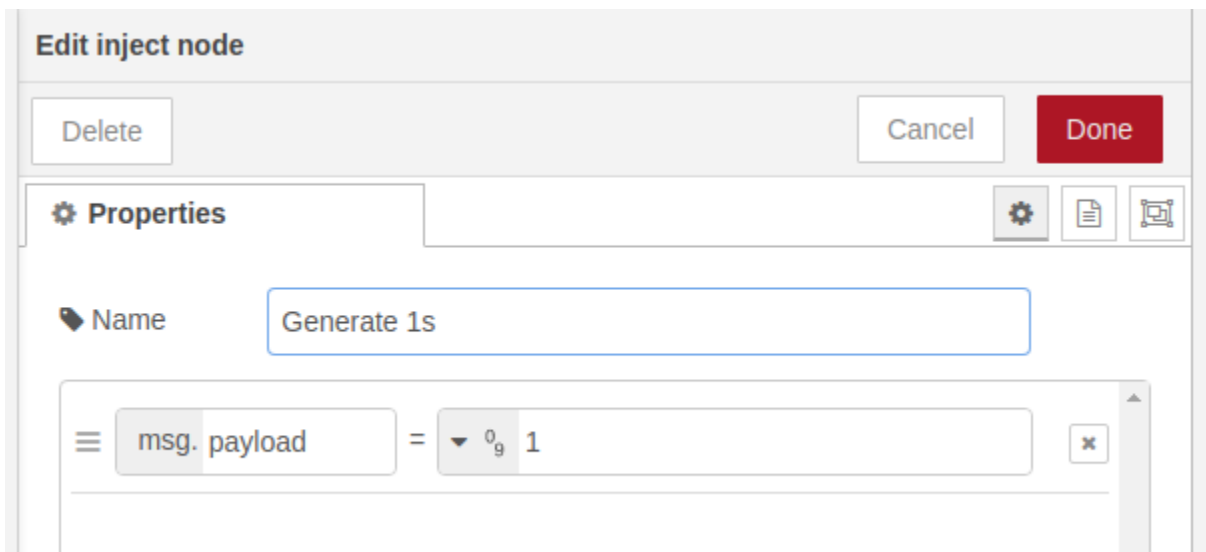


Figure 4-43: Inject Node After Editing

## Chapter 4 LEDs - Preliminary (V0.0)

Now scroll down and look at the bottom of the Properties Edit window.

- If you read the help menu for the Inject node you will know that the Inject node has several functions. We are interested in the function that generates messages automatically.
- To use this feature, click the “Repeat” dropdown menu ①, and select “Interval” ②.

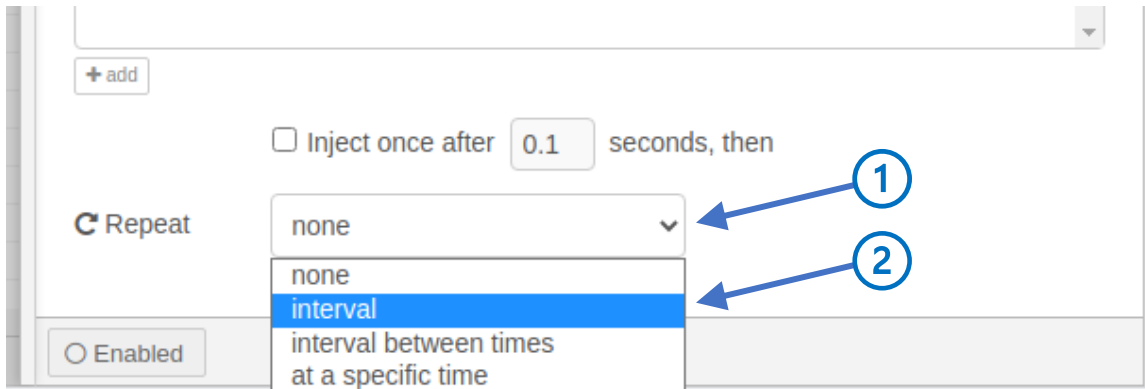


Figure 4-44: Edit Inject Node Repeat Menu

This will open a little selection block where you can select a time interval, either with the up/down arrows or by directly typing a value into the box, as in the figure below:

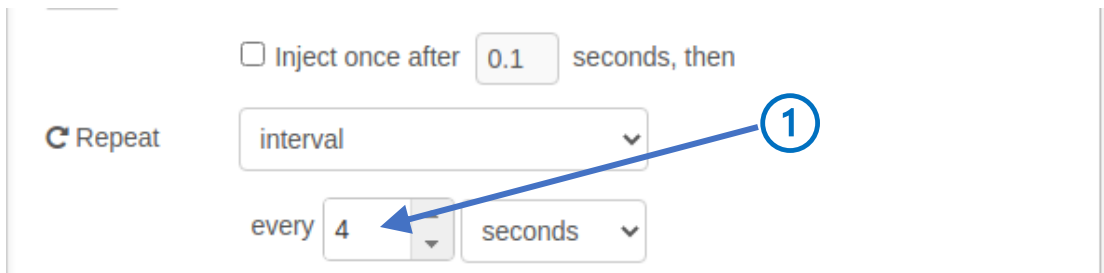


Figure 4-45: Setting Inject Node Repeat

- Set the interval to four seconds ①.
- Click DONE

What have you done? You have set out on the road to automation because now the Inject node will generate a new timestamp message every four seconds (forever!). To see this in action add a Debug node and connect it to the Inject node, like this:

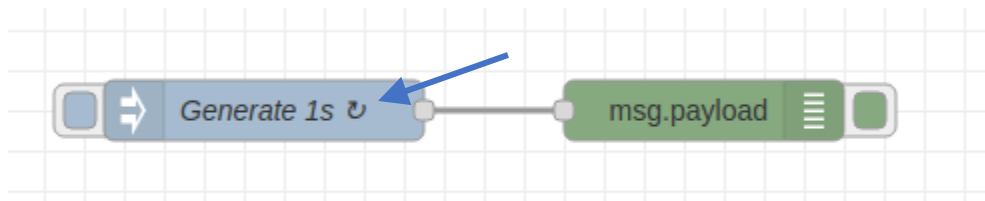


Figure 4-46: Generation of Repeated Messages

## Chapter 4 LEDs - Preliminary (V0.0)

Get out a magnifying glass and look carefully at the Inject node. You should see a little circular arrow (see blue arrow in figure above). This indicates that the node will repeatedly generating messages.

Deeeploy it!

If you did everything correctly then when you look in the debug window in the sidebar you should see a new Debug message every four seconds. You will need the magnifying glass again to see the time (blue box) but check what the time difference is between each message. It should be about four seconds as below:

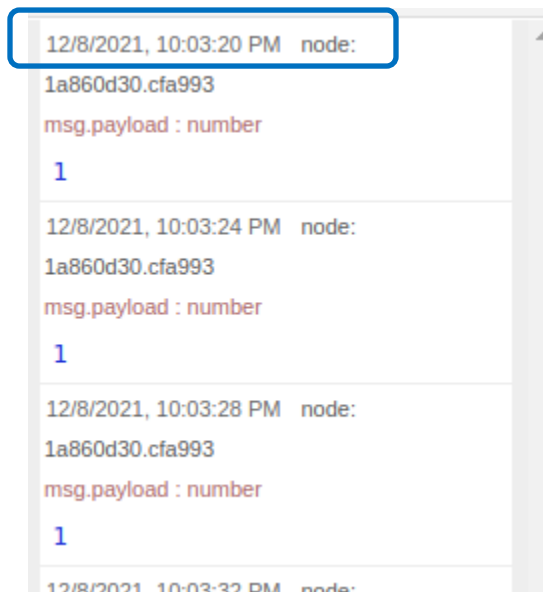


Figure 4-47: A One Output Every Four Seconds

You now have in your hands a flow that will generate a sequence of messages four seconds apart. This should feel very close to what you need to automate the Warning Light flow. By now you realize why you cannot hook this up directly to your Warning Light flow. Because it is only generating the same message over and over you do not have a way to change the pattern. Think about this. Suppose that after a message in which a numeric 1 was generated you could wait for two seconds and generate a message with a numeric zero in it. With a sequence of messages with 0s and 1s spaced two seconds apart you could use the message with the one to turn on one set of lights and message with zero to turn on the other set of lights. One way to do this is with a Delay node. Time to call up another node from the reserve squad and see if it is up to the task.

### The Delay Node

Start with an empty workspace by: Saving your flows, adding a new empty flow tab and deleting the others. Ready to go!

- In your new workspace drag a Delay node from the palette and drop it in your flow.

## Chapter 4 LEDs - Preliminary (V0.0)



Figure 4-48: Delay Node Fresh Off the Palette

Note: When you first bring it into play Delay node is labeled with “Delay 5s” because the default delay is 5 seconds.

As many a hamster has said: “The best way to learn about the Delay node is to download a copy and check out the help file.” (Select the node and click on the tab with the little book icon). Here are what we are interested in: the simplest function of the delay node is to receive a message, delay it for a specified amount of time and then release it to the output.

As always remember “SSS” – Start Simple Stupid<sup>17</sup> and play with the node in a controlled environment until you understand how it works. Let’s see if we can delay timestamp messages. Augment the four second flow above (Figure 4-46) in your workspace by bringing in an Invert and Delay Node, as shown.

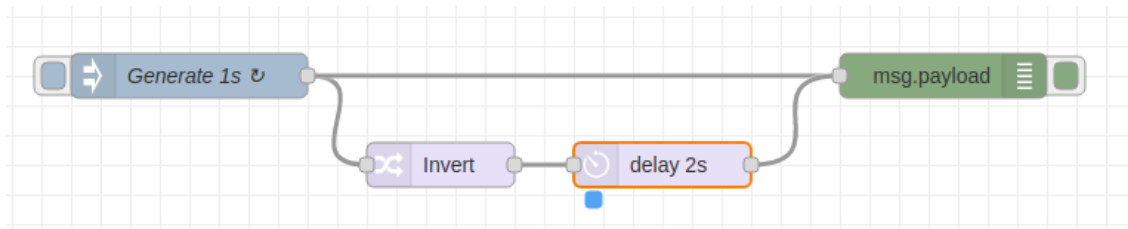


Figure 4-49: Flow to Generate Alternating 0 and 1 Payloads

The Inject node (“Generate 1s”) and the Debug node are the same as in Figure 4-46. The two added nodes (Invert and Delay) are going to produce the messages with a payload of 0 in between the messages with a payload of 1.

Configure the delay node for a two second delay.

- Double click the delay node to open the properties menu
- Set the delay for 2 seconds.
- Click DONE

Your edit window for the delay node should look like this:

---

<sup>17</sup> SSS – and her cousin KISS = “Keep It Simple Stupid”

## Chapter 4 LEDs - Preliminary (V0.0)

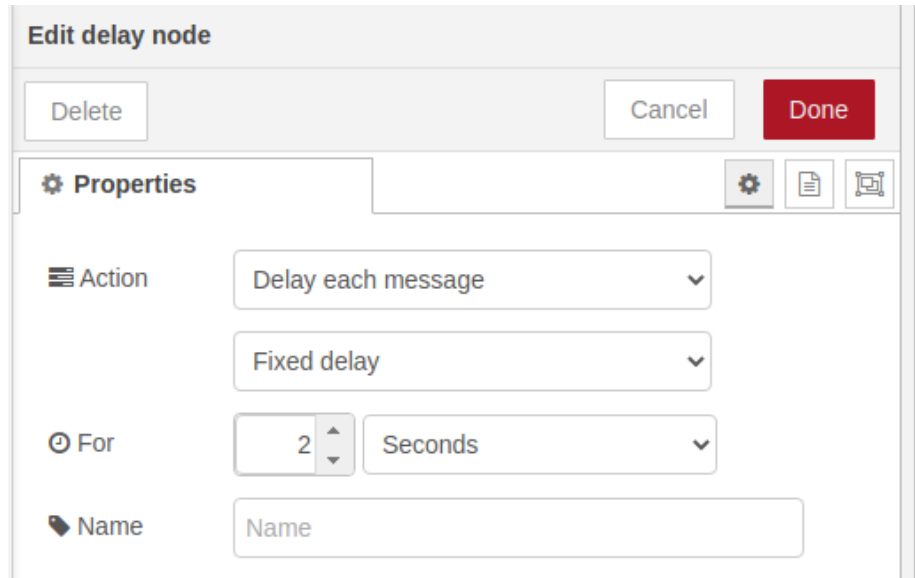


Figure 4-50: Delay Node After Editing

Look at the figure above and think about how it might behave. [time passes as the neurons in your brain fire.....] Did you figure it out? Compare your idea to this description:

“Once you deploy the node the Inject node (“Generate 1s) will generate a message with a 1 in the payload every four seconds and send it to both the Debug node and the Invert Node. The Debug node will print the message. The other copy of the Inject message goes through the Invert node where the payload is changed to 0. This message is delayed for two seconds and then sent to the Debug node where it is output in the debug window. The cycle will repeat every four seconds.”

If you think the flow will do this, then **DEPLOY IT!**

In the debug window you should see a message every two seconds (check the time stamp) the payload will alternate between 1 and 0 as in this figure:



## Chapter 4 LEDs - Preliminary (V0.0)

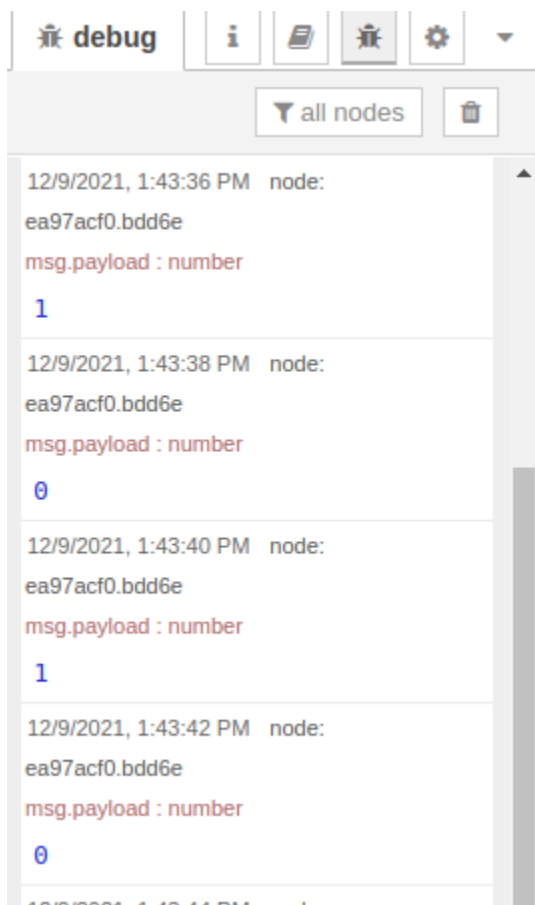


Figure 4-51: Alternating Ones and Zeros in Payload two seconds Apart

Did the flow behave as described above? If not put on you're thinking cap you ordered from HamsterHats.com and using some extra debug nodes to trace the messages.

Once you have convinced yourself that your flow is working it is time to combine it with the Warning Light Flow from Figure 4-41. The Alternating 1 and 0 flow is going to generate a stream of payloads with alternating ones and zero. In the Warning Light flow the pushbutton did this, albeit manually. It sure looks like it would be simple to combine the two flows. Start here

- Open a new tab and import your Warning Light flow (Figure 4-40)
- Copy all the nodes from your Warning Light Flow to The Alternating 1 and 0 flow from Figure 4-41.
- Drop them in the Warning Light flow at the bottom.

You should now have something like this figure:

## Chapter 4 LEDs - Preliminary (V0.0)

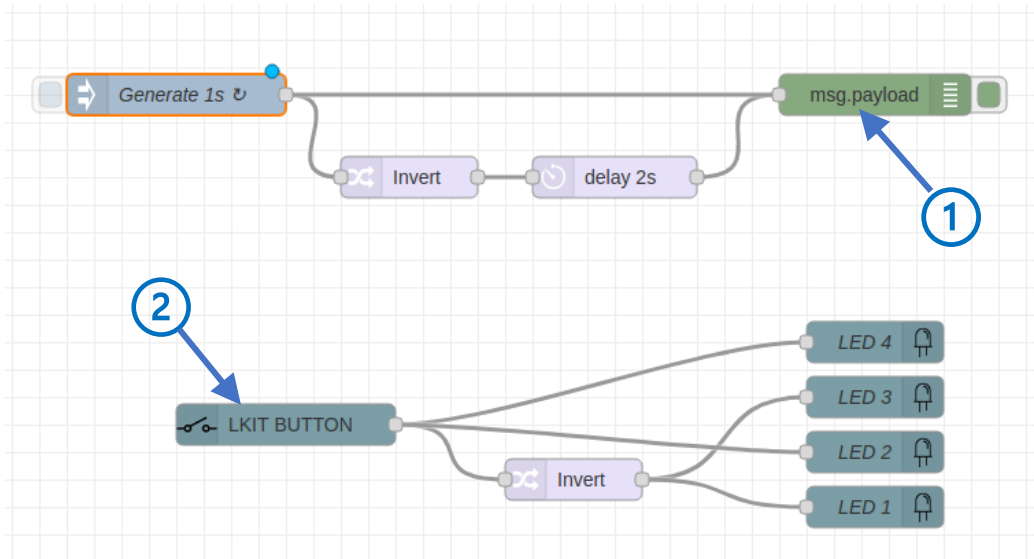


Figure 4-52: Combined Alternate and Warning Light Flows

At this point you just want to take all the messages arriving at the Debug node ① and send them to all the places that want messages from the Button node to go ②. What a mess, because you are going to have to rewire everything, which brings with it the possibility of mistakes. There are several ways to accomplish what you want without making a tangle of your flow. Also, by keeping the Alternating 1 and 0 flow and the Warning Light flow separated graphically it is easier to understand what is happening.

Go back to the Node-RED palette and find the Link In and Link Out nodes (see figure below)

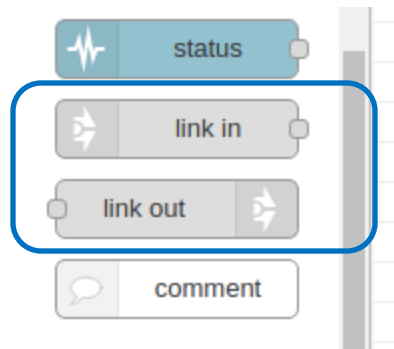


Figure 4-53: Link In and Link Out Nodes in Palette

The Link Out node allows you to collect messages from multiple nodes in your flow and send them over a “virtual” link<sup>18</sup> to an identified Link In Node, where they can be distributed to multiple other nodes. What we want to do here is to replace the Debug node in the upper flow with a Link Out node and replace the Button node in the lower node with a Link In node. Then you will connect them together.

- Replace the Debug node with a Link Out node ①.

<sup>18</sup> Think “imaginary” link although the results are real and concrete enough.

## Chapter 4 LEDs - Preliminary (V0.0)

- Restore all the wires from the Generate 1s node and the Delay node to the Link Out node.
- Replace the Button node with a Link In node ②.
- Restore all the wires from the Link Out node to the Invert node and the LED 4 and LED 2.

Your flow should now look like this:

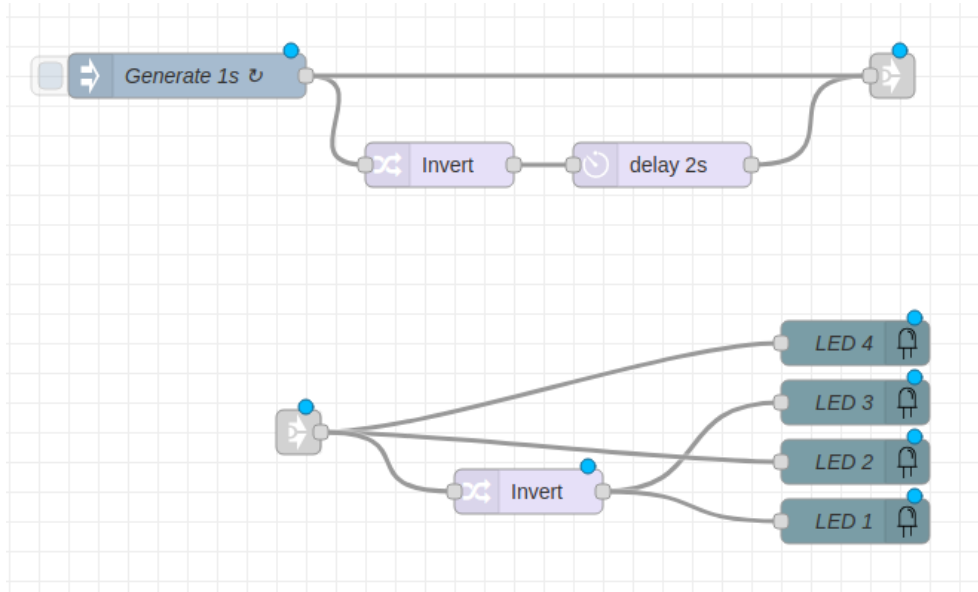


Figure 4-54: Combined Flows with Link Nodes

At this point you are probably thinking: “Great! But how do these links know to talk to each other?” Answer: you must tell them how to connect. The easiest way is to give each link a name. Start by:

- Double clicking the Link Out node and giving it a descriptive name like “Off-On” ① because the messages it sends are going to determine when the warning LEDs change.

When you are done the Link Out Edit window will look like this:

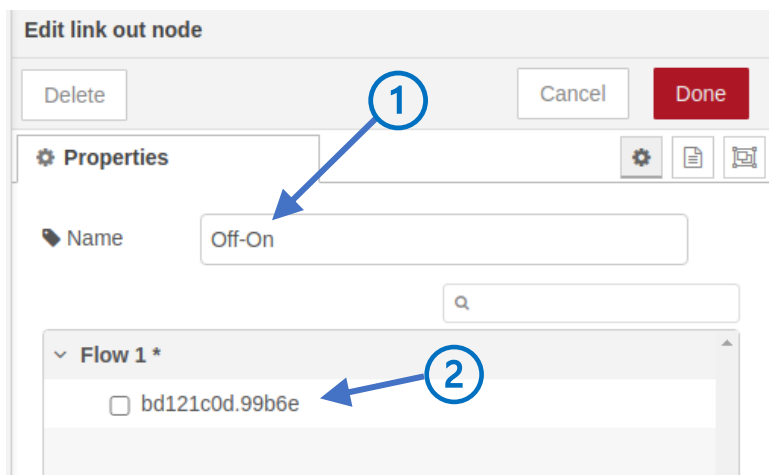


Figure 4-55: Link Out Node Edit Window

## Chapter 4 LEDs - Preliminary (V0.0)

When you finish editing look at the area below the name box ②. This is a list (in this case a very small list with one item) of all the Link In nodes in the flow that you can connect to. The number is a bit cryptic because it is the unique<sup>19</sup> ID number for the Link In nodes in the flow. But ignore this for now. Instead:

- Click DONE
- Double click the Link In node and give it a descriptive name like “LED Control” ① because the messages it receives are going to determine which light are on or off.

You should have something like this:

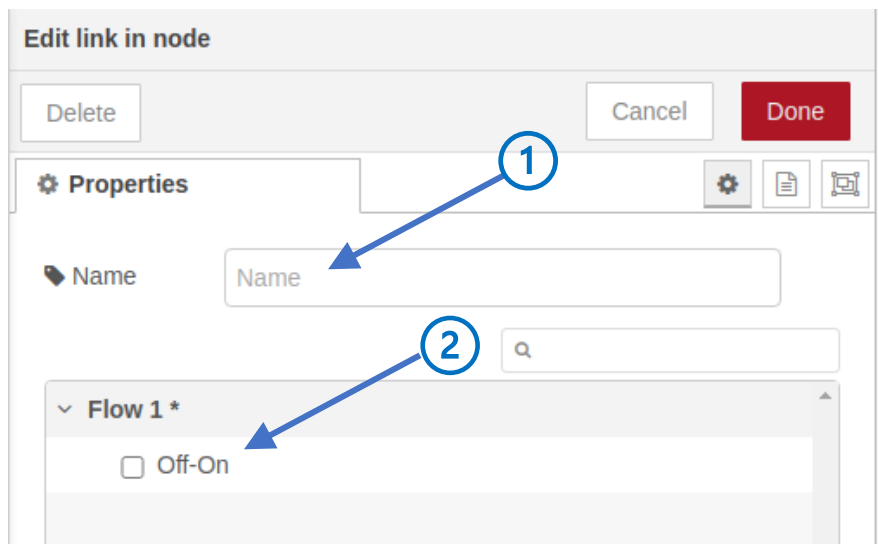


Figure 4-56: Link In Node Edit Window

Look down below the name at the list of nodes. This is the list of all the Link Out nodes that you can connect to this Link In node. Better yet because you gave the Link Out node a name, “Off-On”, you do not need to deal with cryptically unique numbers, you can just deal with the names of the nodes. You want to connect the Link Out node named “Off-On”, so...

- Click the check box next to the name “Off-On” ②.

You should now see this in the edit window:

---

<sup>19</sup> Not quite unique because the numbers are assigned at random when the node is created. But, close enough, because there are  $2^{32}$  different number to represent nodes and  $2^{20}$  flows that further identify the node. Close enough unique for use in this universe. However, things will become problematic if we living in a multi-verse.

## Chapter 4 LEDs - Preliminary (V0.0)

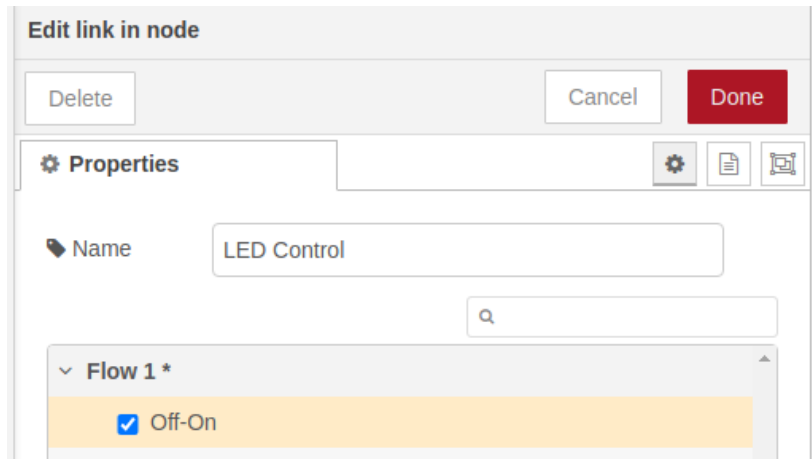


Figure 4-57: Link In Editing - Completed

- Click DONE.
- Go back to the Link Out node and double click it to open the edit window.
- Check the box next to “LED Control” to indicate that you want the Link Out node to connect to the Link In node named “LED Control”
- Click DONE

In your workspace click on either the Link Out or Link In node. This will bring up a dashed gray line indicating that any message received by the Link Out node will travel over the wire represented by the dashed line to the attached Link In node and thence to any nodes that output port of the Link In node is connected to.

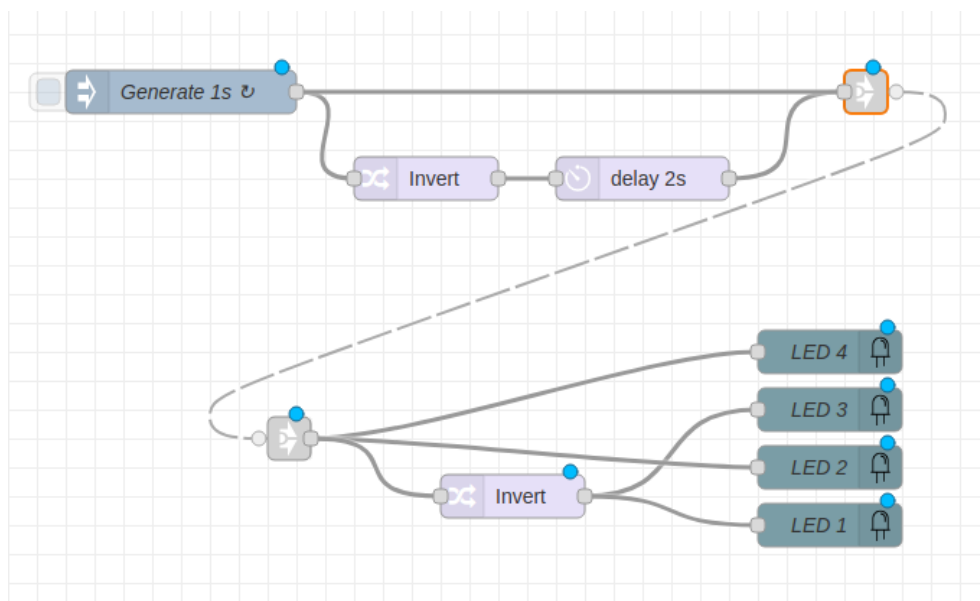


Figure 4-58: Completed Warning Light Flow

## Chapter 4 LEDs - Preliminary (V0.0)

### Deploy It because the blue dots are annoying.

If you made no missteps, you will now see that the LEDs alternate between the two patterns every two seconds. If you think that this is not going to get very much attention, go back and adjust the time interval in the Generate 1s node to, say, one second. Then adjust the delay in the Delay node to be half that value or 0.5 seconds<sup>20</sup>. Deploy and enjoy the micro-light show.

**LED Puzzle # 3 - Links from Different Tabs** – In the above example flow try moving the lower flow to a different flow tab. How does that affect the link nodes? Are they still connected? Deploy the flow and see if it still works. Click a link tab on one of the flows. Can you tell what node (or nodes) it is connected to?

**LED Puzzle # 4 - LED Control in Group Mode** – Go back to the LED group mode flow shown in LED Puzzle #1 (of course you saved it). Bring that flow in and substitute it for the lower flow. Make the necessary modification to link this flow to the upper flow. Deploy it. Does it work the same way as the original flow?

**LED Puzzle # 5 - Null Nodes** – rather than use link nodes to wire the two flows together you might use some sort of “null” node which just takes messages in and passes them to the output. However, in Node-RED there is no such thing as a “Null node”. With your now substantial (really!) knowledge of Node-RED you should be able to think of at least six ways to make a “Null Node”. See if you can create three examples. Make your examples into a subflow labeled “Null”

- Test each of your “Null” nodes.
- Substitute one of your Null node creations for each Link Out and Link In node in Figure 4-58.
- Deploy the flow
- Did it work?

Here is what a Null Node should look like:

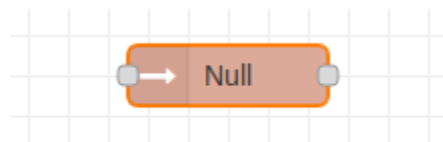


Figure 4-59: Null Node

### Beware! Loops Ahead

It is now time to take up the subject of “Loops”. Node-RED is an event-based system, where an event, usually in the external world, occurs and triggers a cascade of messages that result in actions. This is part of the charm of Node-RED. In some of the recent experiments you have done you have been able to trigger repetitive actions using the Inject node, however, using delay elements and loops you can arrange for messages to circulate on their own and trigger continuous actions. Before we proceed however this station brings you the following public service announcement:

---

<sup>20</sup> The delay value should always be less than the rate at which messages are generated. Otherwise, you will be issuing the delayed messages after the next generated message. Experiment with the values and see what happens.

## Chapter 4 LEDs - Preliminary (V0.0)

---

**Engineering Tip # 9 - Beware of the Loop!** Loops are the powerhouse of many programming languages. The difference between a calculator and a computer is that while the calculator can plod through a single calculation a computer can chain calculations together and repetitively perform them on different data. This wonderful capability comes to you courtesy of loops, which are nothing more than sections of code that run over and over again until they accomplish some purpose. Powerful, but remember the Parker Principle from an earlier discussion or to paraphrase “Chainsaw are terrific tools but use with care.” Probably more programs have foundered on simple mistakes in loop construction than any other cause. Mistakes in controlling when loops stop, where they stop or that they even stop at all is the cause of many problems. Work carefully and the loop will be your friend, one moment of inattention and your loopy flows will crush your program.

---

### Let the Light Shows Begin!

In the following examples and puzzles you will be working with simple loops to automate the display of patterns on the LEDs. As is usual let’s start with something simple.

Specification for Set LED 1:

1. The flow has two Inject nodes: CLEAR and START.
2. Clicking on CLEAR turns all the LEDs off.
3. Clicking on START turns on only LED 1.

STOP! Think! Can you build the flow just from the specification above? Give it a try.

Does your flow look more or less like this?



Figure 4-60: Flow to Turn on LED 1

Good – Deploy It!

Test it against the specification above. Can you clear all the LEDs? Does clicking START turn on only LED 1? Can you turn it off with the CLEAR node?

## Chapter 4 LEDs - Preliminary (V0.0)

If you had trouble here is how each node is configured.

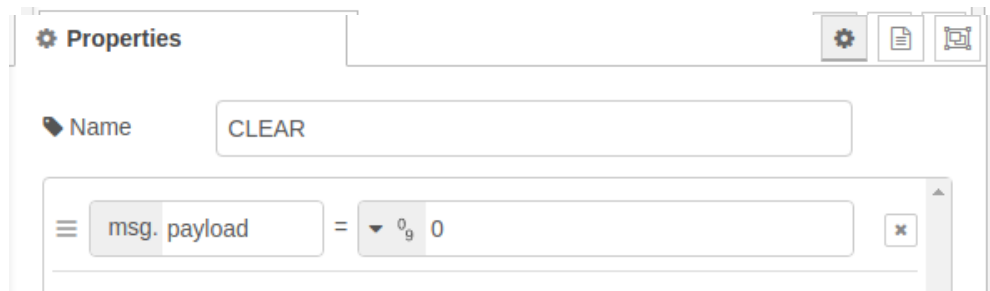


Figure 4-61: Configuration of CLEAR Node

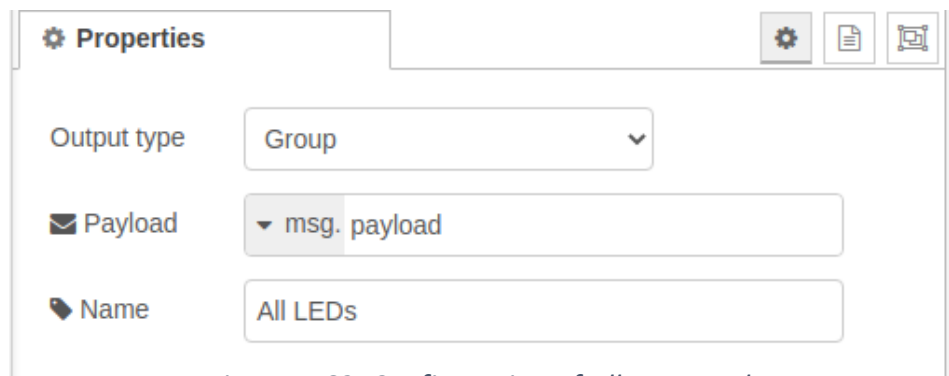


Figure 4-62: Configuration of All LEDs Node

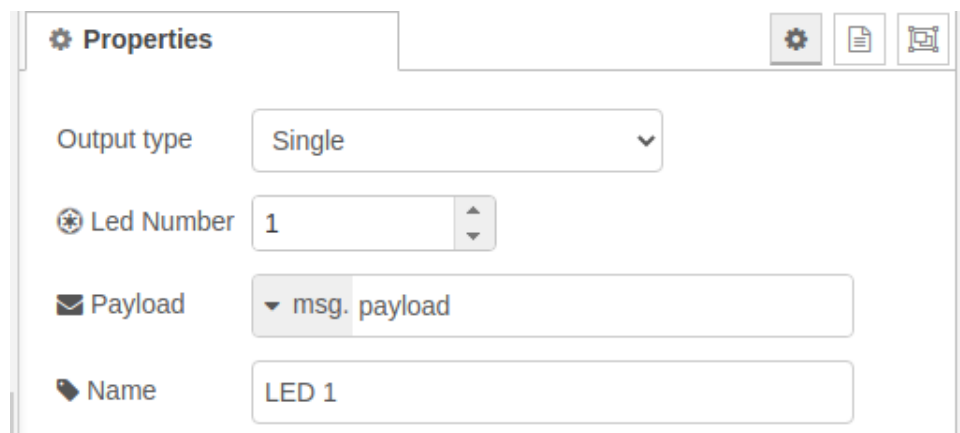


Figure 4-64: Configuration of LED 1 Node

Next step. Consider this extension to the specification:

4. After one second LED 1 turns off.

Give it some thought. This sure sounds like the kind of game the Delay node would like to play in. Where would it go? How would you use the output from the Delay node? How would the output of the delay node turn off LED 1?



## Chapter 4 LEDs - Preliminary (V0.0)

See if you can modify the flow to meet the new requirement. Deploy it! Test it! Success or dismal failure?

Did you get something like this?

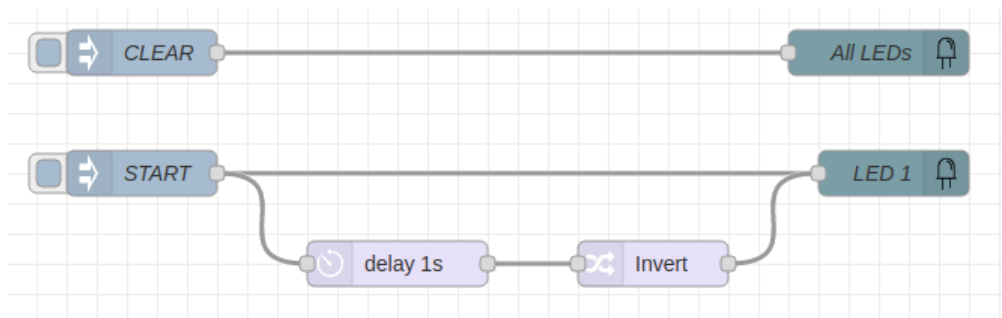


Figure 4-65: Delayed Off Flow

The purpose of the CLEAR subflow is simply to turn all the LEDs off so you can see what is happening. When you do click the start node a message with a payload of numeric 1 goes to both the LED 1 and the Delay nodes. In the first case it turns LED 1 on and in the second case the message sits in the delay node, twiddling its thumbs for 1 second before moving on. So, LED 1 turns on and time passes. After one second the message passes through the Invert node and the payload becomes a 0. When this message hits the LED 1 node it's lights out.

Well, not bad. At least you have a one second Flash, unfortunately, it is only one flash. Can you think of a way to keep the LED flashing without click on the mouse all day?

How about this: suppose you took the message that comes out of the Invert node and sent it back around to the input of the Delay node? Wouldn't that be almost the same as clicking the Start node again. Should you try it? Or, would all life in the known Universe end if you did such a crazy thing. Think about it... no more rainbows, no more clouds, no more planets, no more galaxies and no more unicorns. Should you take a chance? Yeah, what the heck, probably it will be OK and if it does destroy the Universe who's to know<sup>21</sup>.

---

<sup>21</sup> During the testing of the first atomic bomb there were some scientists who thought that the test might set the oxygen in the atmosphere on fire [[Bethe, Teller, Trinity and the End of the Earth](#)] Such an outcome would have ended all life as we know it, but fear not, the anaerobic bacteria would probably have survived and after a few billion years we'd be back.

## Chapter 4 LEDs - Preliminary (V0.0)

Make the change so the flow looks like this:

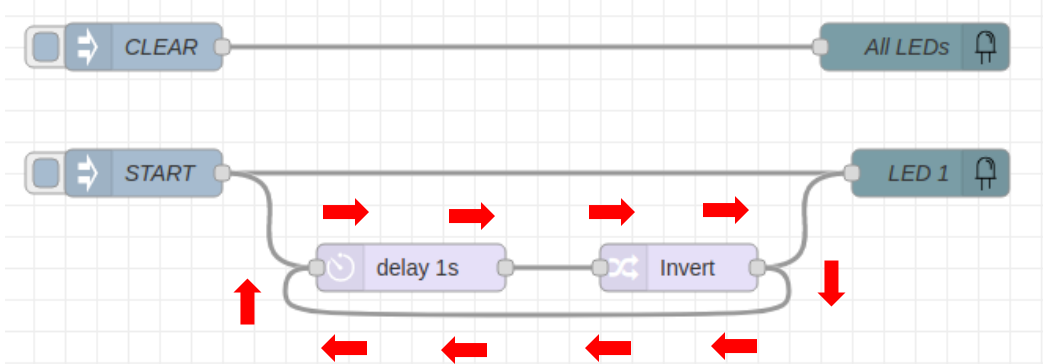


Figure 4-66: Simple Loop Flow

Deploy it if you dare!

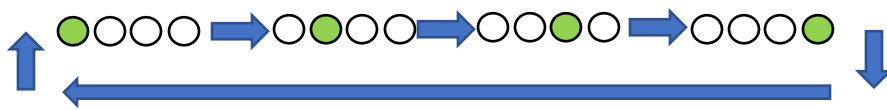
Whew, close one, fortunately, the Universe did not end<sup>22</sup> and LED 1 is happily flashing away... forever. Yes, the message controlling the LED will happily circulate forever between the Delay and Invert nodes. In the chapter on relays, you will learn how to terminate loops. Until then here is a hack, yes, a hack right here in the middle of this very scientific and technical tutorial. Well, use what ya got! The simple hack is this: move one of the nodes a tiny bit until the little blue dot showing it is undeployed appears. Then if you deploy your flow again the current flow will be terminated. Weird and ugly, but it does the job for now<sup>23</sup>.

**LED Puzzle # 6 - Flash them all** – Modify the flow above to flash all the lights on and off together. Remember the Invert node you made earlier will only change a zero to a one and vice versa. You must either replace this node or accommodate it in some way. Also, you will want to change the LED 1 node to use the group mode so that you can control all the LEDs at once.

### Shifty Lights

Let's work on something more complicated. Here is the specification:

1. There is one Inject node labeled "START"
2. When you click start all the LEDs are cleared.
3. One second only LED 1 turns on.
4. The sequence proceeds as below every second, forever.



<sup>22</sup> But if you are not careful you could blow up your Node-RED program. If you play around with loops, you will find that you cannot connect the output of a node directly to its own input. This is a bit of a safety feature of Node-RED. Imagine what you happen if you connected the output of a change node to its input. Messages would start flying around between the input and the output about as fast as they could. After a few minutes the CPU in the Raspberry Pi would start to glow red and then after an impressive flash and a small puff of smoke you would need to buy a new card. Actually, not. The Raspberry Pi is smart enough to shut down before it melts.

<sup>23</sup> The location of the node on the workspace is part of the node definition. Moving the node a little bit changes the node definition and Node-RED takes this as a cue that the flow has changed, which it has, but not in any significant way.

## Chapter 4 LEDs - Preliminary (V0.0)

There are several ways to meet the specification. You could set up a huge loop to controls each LED individually or you could use the LEDs in group mode as in the following.

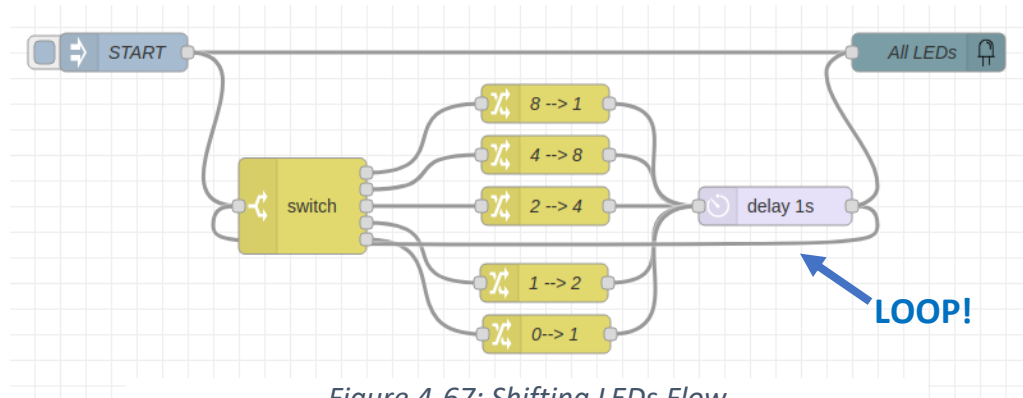


Figure 4-67: Shifting LEDs Flow

Notice that here the delay is moved to a slightly different place. The overall function is the same, but it is easier to build and understand if you wire the change nodes to one place, namely the delay node. See if you can understand how this flow works and then...

Build up the flow. You have enough skill and knowledge now to build a flow just from the specification and maybe a hint or two about how the flow should look. Note the loop from the output of the delay node to the input of the switch node. It is a bit hard to see. In Node-RED you can connect a wire from one node to another, but you can't control how Node-RED routes the wires. This means that a loop is not always obvious or is blocked by other nodes. Sometimes you can move the nodes around a bit to emphasize the loop or the intended flow of information.

## Chapter 4 LEDs - Preliminary (V0.0)

If you are having trouble understanding the flow here is the configuration of the Switch node.

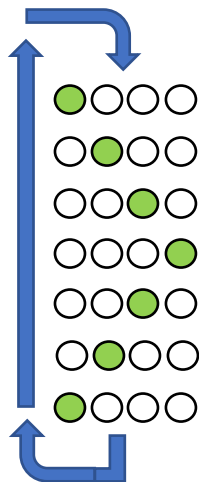


Figure 4-68: Shifting Lights Flow Switch Node Edit

Deploy your flow and see how it works.

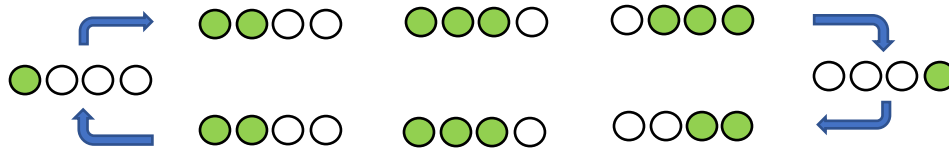
**Puzzle # 7 - Shift the other way** – In the flow above the LEDs shift from left to right. If you were able to get the flow working then see if you can modify the movement of the lights so that the flow in the opposite direction, i.e., from right to left.

**Puzzle # 8 - Ping-Pong Lights** – Bounce a single lighted LED back and forth between LED 1 and LED 4. Here's the pattern.



## Chapter 4 LEDs - Preliminary (V0.0)

**Puzzle # 9 - K.I.T.T. Lights** – Back in the old days (1982) there was a TV series called Knight Rider featuring a cyber-intelligent car called [KITT](#) (Knight Industries Two Thousand). As per the usual Hollywood trope<sup>24</sup>, it was the product of some eccentric millionaire’s imagination and engaged in bring the bad guys and gals to justice. The dashboard had all sorts of lights and TV screens (imagine that). On the front of the car were 8 lights that moved from side to side to express whatever emotions the cyber-car was currently experiencing. You only have four LEDs so here is an approximation of the pattern.



Probably the lights should shift every half second. Can you build it?

### Science Fiction Lights

Now for something interesting and a little strange. Suppose you were sitting at your desk and a famous movie director knocked on the door. You open the door, and she rushes in with a request: “Look, we are in big trouble. Tomorrow we are shooting a big Sci Fi spectacular, and our spaceship needs some flashing lights in the command center. Just four little green LEDs will do<sup>25</sup>. Can you help us?” Cool, but what are you going to do?

You’ve seen it every time you watch any Sci Fi movie, especially from the 1960s. In the background on the panels are lots of randomly flashing lights. You can probably do the same thing. There are at least two approaches. You could put a random four-bit pattern up on the four LEDs every second, however, it would look a bit stiff because LEDs would turn on and off in groups every second (see the next section). Another approach would be to turn each LED on and off for random lengths of time. But, my goodness, how would you do such a thing? Turns out that you already have a node to do this, the trusty Delay Node.

Open up a tab and drop a delay node on it. Double click the delay node to open the properties menu. Notice that the default condition (the configuration when you first drop a node in your workspace) is a fixed delay. But wait a minute, that box is a dropdown menu. Open it and you will see that you can select a random delay. It does not get much better than this: you can delay messages for a random<sup>26</sup> amount of time. If these messages control the LEDs, then you are all set.

<sup>24</sup> Trope – this word will be on the SAT.

<sup>25</sup> Apparently, it is not going to be a big budget block buster.

<sup>26</sup> Technically there is a little elf inside the node that rolls some dice every time a message comes in and then, using a tiny stopwatch, hangs on to the message for the length of time determined by the dice.

## Chapter 4 LEDs - Preliminary (V0.0)

Select “Random Delay” from the dropdown menu and you will see this”

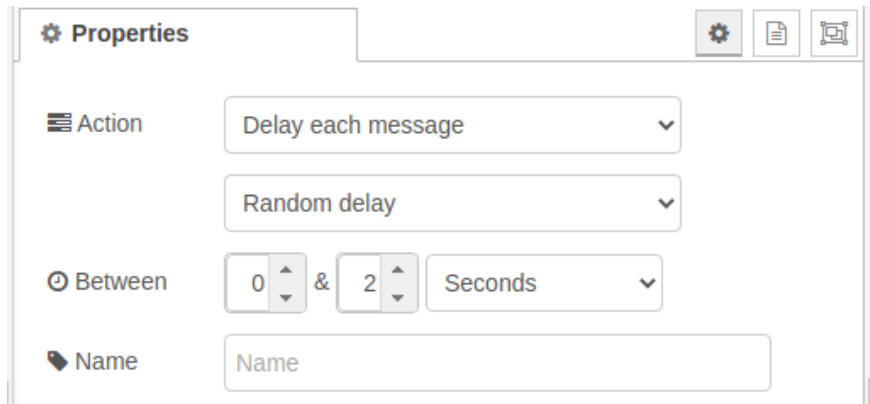


Figure 4-69: Delay Node Edit for Random Delay

You can set a delay range and the message will be delayed by some random time selected from within that range.

By now you know how to start: SIMPLE! Set up a flow like the one below:

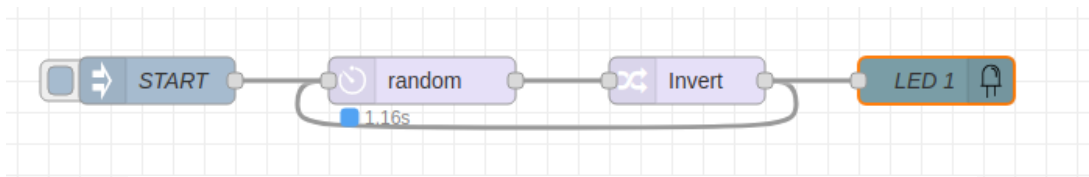


Figure 4-70: Random On and Off for One LED

Set the random delay to be between 0 and 2 seconds. Deploy it and see what happens. Notice that whenever a message is delayed a little blue square appears below the node telling you what the delay time is.

Now extend this simple flow to cover the other three LEDs. Something like this:

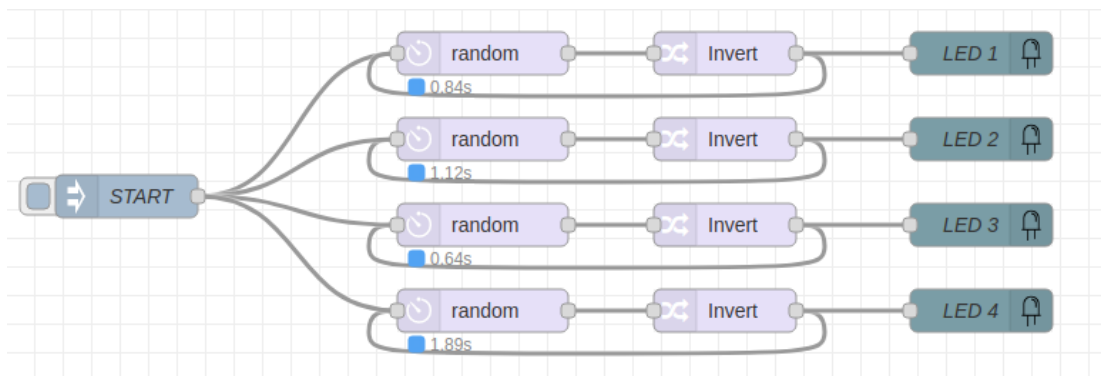


Figure 4-71: Random Flashing of Four LEDs

No assembly instructions included because if you are able to get the random flashing of a single LED to work you will be able to put this flow together and Deeepppplllllooooy it!

## Chapter 4 LEDs - Preliminary (V0.0)

**Puzzle # 10 - Radiation simulation** – you did such a great job on the SciFi front panel that the director asked you to create the sound effects for a Geiger counter. You know how to play a sound. Can you build a flow so that each time a lamp turns on or off a click is played in your headphones or on your speaker? You know how to make a message play a sound, so it should be straight forward. Try it out. Does it sound like a Geiger counter?

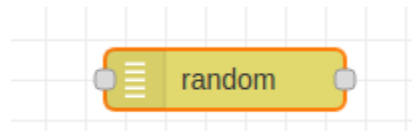
### Rolling the Die

If you are a Dungeons and Dragons player, you know that some games use a 16- sided die. On the Learning Card you have four LEDs, just enough to display all the numbers between 0 and 15. Let's build an electronic version.

Specification:

- An inject node labeled ROLL
- When clicked the LEDs go blank for one second and then display a random number between 0 and 15 until the inject node is clicked again. Yes, it is a little hard to distinguish the blank period from 0, but it's the best we can do here.

Where are the random numbers going to come from? Well, of course, there's a node for that: the Random node.



*Figure 4-72: Random Node*

Listen to the hamsters: read the Help file!

To make sure you understand how the Random node works follow the SSS principle. How about a flow like this?

## Chapter 4 LEDs - Preliminary (V0.0)

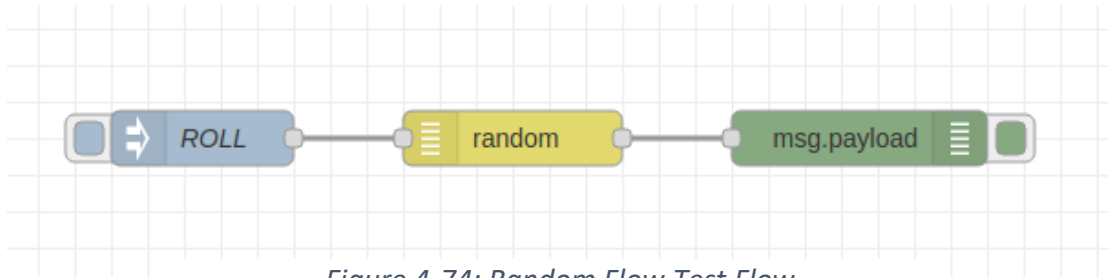


Figure 4-74: Random Flow Test Flow

Click the inject node labeled ROLL a few times and see what sort of numbers you get. Open the Random node for editing by double clicking on it. You will see that the default setting for the node is to create a random number between 1 and 10 every time it receives a message.

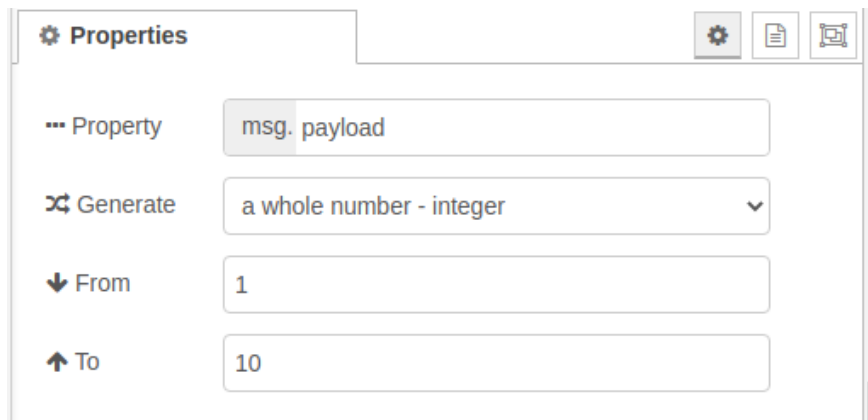


Figure 4-73: Random Node Edit Dialogue

Now you know what to do. Fix up the random node so that it generates numbers between 0 and 15 and send the result to the LEDs. This looks like a good time to use the LEDs in group mode. Now all you have to do is figure out how to turn off the LEDs for one second before you present the answer.

Does your flow look like this?

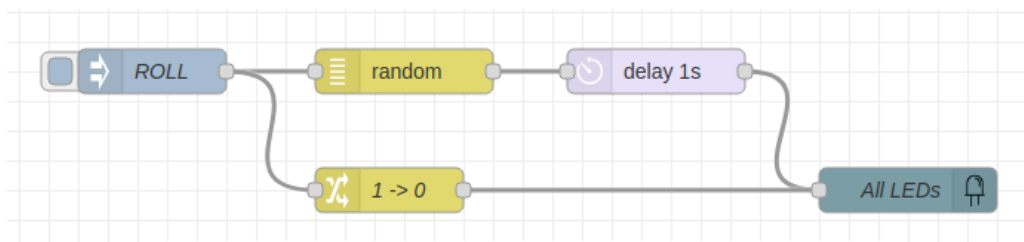


Figure 4-75: Random LED Flow

Deploy your flow and see if it meets the specification. Be sure to check that there is a 1 second break between the display of the results.

**Puzzle # 11 - Pushbutton Die** – modify the rolling die example so that pushing the button on the learning card roles the die. Just remember that the Button Node generates two messages: one when the button is pushed and one when it is released. You will need to get rid of the second message.



## Chapter 4 LEDs - Preliminary (V0.0)

**Puzzle # 12 - Sci Fi Lights – Take Two** – Modify the Die flow above so that it once started it loops forever it places a random number on the four LEDs once each second. How does the display compare to the one for the Sci Fi Lights above? Which one looks more like a real science fiction console?

Crawling, Walking... enough! Now it is time to Run with the LEDs.

### Running with LEDs

#### Email To LEDs

Let's start with a relatively simple project in which you will control the LEDs from Email messages. Yes, of course it is a little weird, and there are better ways to connecting to the outside world. However, you should give it a try because it will be good practice.

You can control the Learning Card from Email messages. Right now, the main issue is to keep the Email messages simple so that you can analyze them with simple nodes, like the Switch and Change nodes. How about this for a specification:

(In the following the quotes around text identify the text and are not included in the text of the email).

- The subject line will contain "LED Control". (Because later you might want to use different subject lines to direct commands to other Learning Card output types, like relays).
- The email body will provide the particulars of LED control and may contain any of the following command strings.
  - "LED1:ON", "LED1:OFF", "LED2:ON", ... "LED4:OFF"
  - "GRP:0", "GRP:1", "GRP:2", ... "GRP:15"
- Strings starting "LED" indicate control of a single LED identified by the number and set to either on or off as indicated.
- Strings starting with "GRP" indicate control of the LEDs as a group where the number in the string is sent to the LEDs as a group mode command.
- All command strings must be on a single line in the email body.
- All commands in a line will be executed, but the order of execution is arbitrary.

This is a bit tricky, but then you are running now so chase the solution down.

First things first. How does one even get an email message into Node-RED? By now you know the answer: There's a node for that! Check out the palette and locate the Email input node that looks like this:



Figure 4-76: Email Node

Note: the email node has either one or no input port depending on how it is configured. Here you will be configuring the node to have an input port so that you can control testing. More later.

Read the Help File. Otherwise, you will be running in the dark and that's a good way to trip.

From the Help file you should have the following ideas in mind.

## Chapter 4 LEDs - Preliminary (V0.0)

- The email node periodically (or on demand) checks an email server (like Gmail.com) and fetches any messages there that have not been marked as read.
- The *msg.topic* property contains the subject line of the email.
- The *msg.payload* property contains the email text.

To work with the Email node, you will need to set up an email account. The example here is based on Gmail because it is free and if anything goes wrong, like you start getting hundreds of emails for hamster exercise wheels, you can toss the account into the waste bin.

The other issue is that you will need to lower the security setting for the account because you will be accessing the account with an untrusted application. You are bypassing some of the security checks so be sure that this email is not linked to any of your important accounts. Be safe.

Based on the specification above you want to set up a flow that periodically checks for new email messages; if it finds one it must parse the subject line and the email body and act on any information it finds there. “Parse” is just a fancy word for analyze, but in the world of computer science one meaning is to locate and identify specific substrings in a larger string. For example, here you will want to find a string like “LED:1” somewhere in the email body.

It certainly seems like this is going to be a complex flow, so listen the hamsters and start simple. You will want to build of the pieces of the flow individually, test each piece, combine the pieces and then test the overall flow. The first thing to do (after reading the Help file, of course) is just to see if you can fetch an email message and show it in the debug sidebar window. How about a flow like this?

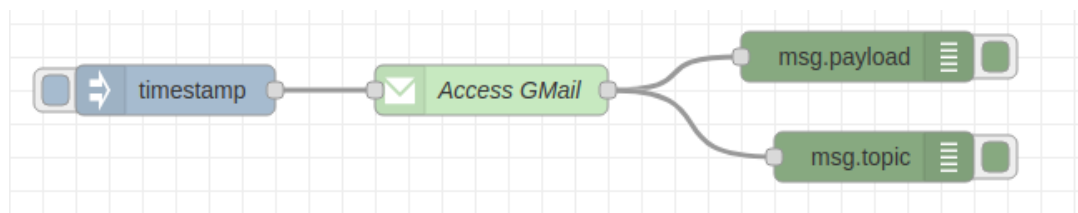


Figure 4-77: Simple Email Flow for Testing

In the Email message you are going to need to see both the email text and the subject line, so that is why there are two Debug nodes.

Now comes the tricky part: configuration. Check out the figure below.

## Chapter 4 LEDs - Preliminary (V0.0)

The screenshot shows the 'Properties' panel for an email node in Node-RED. The settings are as follows:

- Get mail: when triggered
- Protocol: IMAP
- Use SSL?:
- Start TLS?: never
- Server: imap.gmail.com
- Port: 993
- Userid: \*\*\*\*\*@gmail.com
- Password: .....
- Folder: INBOX
- Disposition: Mark Read
- Criteria: Unseen
- Name: Access Gmail

Figure 4-78: Email Node Configuration

Look at each part of the configuration from top to bottom.

- “Get Mail” – here you are going to use “when triggered” – This means that when the Email node receives a message it will contact the server to get your mail.
- “Protocol” – “IMAP” is the default protocol and will work with Gmail.
- “Use SSL” – “never” is the default and will work with Gmail.
- “Server” – “imap.gmail.com” is the default because almost everybody is using Gmail. If you use a different server, you will need to find out the address of your email server.
- “Port” – “993” for Gmail, your email server may be different.
- “Userid” – plug in your email address.
- “Password” – plug in your password. You will need to renew the userID and password in this node if you stop Node-RED and restart later. This is so that your password is not just sitting in the node description, which anybody with access to your computer can read.
- “Folder” – “INBOX” is the default and is the place where this node will look for emails.
- “Disposition” – “Mark Read” is the default. This tells the email server that once the email has been downloaded to mark it as “Read”. This is important because if you do not mark the emails you download as “Read” then you will end up reading the same email again and again.

## Chapter 4 LEDs - Preliminary (V0.0)

- “Criteria” – the default is “Unseen”. This indicates to the email server that you only want messages that you have not yet seen, in other words all the messages that have not been read yet.

If this does not make sense to you think about the email server as a post office, but one that stores all your mail forever. In this post office you do not take your mail away, you only get to take away a copy. This would not work in real life because post offices would need to have huge warehouses to store all the mail and rows of copy machines to make the copies. In the digital world it is no problem because storing bits is easy and making copies is even easier.

Keeping in mind this strange “store and copy” post office and think about the configuration this way.

You (thinking to yourself): “There’s the one o’clock whistle, I’d better go to the post office and check for mail” (Get mail when triggered)

You (thinking some more): “At the post office they are very formal, I’s better say please and thank you.” (Protocol)

You (thinking, thinking, thinking): “Normally, I’d use a translator because to keep things really secure the clerks in this post office only speak Navaho<sup>27</sup>. But I’m going to take a chance today ask them to use English.” (SSL and start TLS).

You (still thinking away): “Let’s see the post office is at Gmail Avenue” (Server)

You (more thinking): “I always got to counter 993.” (Port)

You walk to the post office and step up to the counter...

Clerk: “May I have your name and see your driver’s license please”

You: “Certainly I’m Harold Amster and here’s my license.” (Userid and password)

Clerk: “Thank you sir, how may I help you

You: “I’d like to get a copy of some of the messages I’ve received that are stored in the INBOX folder.” (Folder)

You: “After you make a copy of the messages for me could you stamp them all as “READ” in big red letters?” (Disposition).

Clerk: “I can do that for you”

You: “Please only make copies of those messages that I have not seen yet, so don’t make copies of any messages marked as READ.” (Criteria)

Clerk: “No problem Mr. Amster. Here are your copies. Have a nice day.”

You: “Thank you. It’s been my pleasure”

---

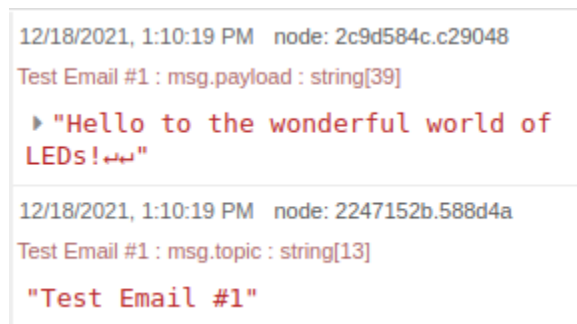
<sup>27</sup> During the Second World War US Marines in the Pacific used Native American language speaker to keep important radio and telephone messages secret. The code was never broken. See [Code Talkers](#).

## Chapter 4 LEDs - Preliminary (V0.0)

Important: Remember you must lower the security of your email account to allow for access by Node-RED. Follow the directions from Chapter 3 for the Pushbutton to Email example.

Ready to go? Deploy your flow. If you have not configured the Email node properly you may get messages in the Debug sidebar that will give you some idea of what went wrong.

- Now go to a different email account and send a message to your new account with the subject line of "Test Email #1" and a text body of "Hello to the wonderful world of LEDs!"
- Go back to your Node-RED flow.
- Select and clear the Debug sidebar.
- Click the Inject node to trigger the Email node.
- After a short delay you should see your email in the Debug sidebar.



```
12/18/2021, 1:10:19 PM node: 2c9d584c.c29048
Test Email #1 : msg.payload : string[39]
▶ "Hello to the wonderful world of LEDs!"

12/18/2021, 1:10:19 PM node: 2247152b.588d4a
Test Email #1 : msg.topic : string[13]
"Test Email #1"
```

Figure 4-79: Received Test Message

If things did not work out the way you expected check the tiny little messages that appear under the Email node as the system cycles. These track the progress of your interaction with the email server. If you see a message in red that will be a clue to where things got tangled up.

The figure above shows the received test message. Now...

- Clear the Debug messages
- Click on the Inject node to cycle the email access again.

Did anything happen? You should not have received any new messages because you told the email server to mark your last message as "READ" and you are only requesting copies new, unread messages to be sent to you.

## Chapter 4 LEDs - Preliminary (V0.0)

Try this: send a message in the format of the specification. Make the subject line “LED Control”, but without the quotes and make the text “LED1:ON”, again without the quotes. It should look like this when you fetch it using your Node-RED flow.

```
12/18/2021, 8:57:04 PM node: 2c9d584c.c29048
LED Control : msg.payload : string[9]
▶ "LED1:ON"

12/18/2021, 8:57:04 PM node: 2247152b.588d4a
LED Control : msg.topic : string[11]
"LED Control"
```

Figure 4-80: Example of Received LED Control Message

The next step is to put together a flow that will fetch the email and separate out those email messages that relate to LED control. These are the messages you are going to process further. You can modify the flow from Figure 4-77 into the flow shown below.

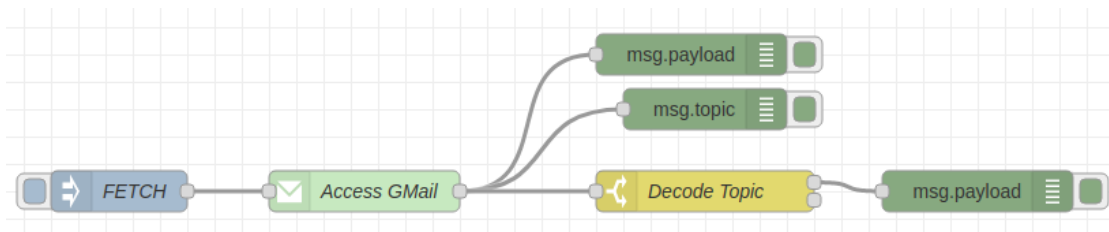


Figure 4-81: Topic Decode Flow

In this flow the Switch node labeled “Decode Topic” is going to check the subject line (i.e. the topic) of every email received for the string “LED control” and if it finds such a string it will send the message to the Debug node where the message text will be printed.

Here is the configuration of the Decode Topic change node:

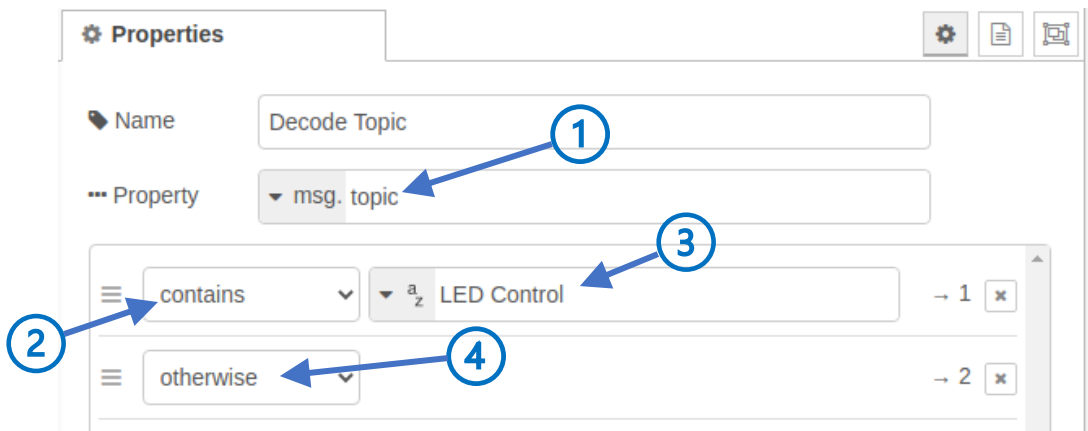


Figure 4-82: Decode Topic Node Configuration

## Chapter 4 LEDs - Preliminary (V0.0)

Look carefully at this configuration because it is a bit different from other Switch node configurations you may have used. The property that you want to examine is the `msg.topic` ① rather than the `msg.payload` because the `msg.topic` is the subject line of the email. The first rule uses the comparison action “contains” ② and the comparison string “LED Control” ③. This rule indicates that if the subject line of the email (i.e. `msg.topic`) contains the string “LED Control” anywhere in the subject line string then the messages should be sent to output 1. Note that the second rule “Otherwise” ④ indicates that any messages that do not meet rule 1 should be sent to output 2 and good-bye. The overall effect of this switch node is to isolate those emails with the subject line “LED Control” from all the other emails you might receive for vitamin supplements, investment opportunities, pet food and from widows of Nigerian military officers who have problems obtaining their inheritance and need your help.

Cook up a few email messages, some with “LED Control” in the subject line and some without. Send them and then pick them up with the flow in Figure 4-81. Those messages with “LED Control” in the subject line should pass the message to the Debug node connected to output 1 of the Decode Topic node.

Next Step: Separate single mode control messages from group commands. The email text commands of the form “LEDx:ON” and “LEDx:OFF” (where x is the LED number) must be separated from those commands of the form “GRP:y” where y is the group code. This is easy to do with another switch node that determines whether a string contains one of the commands by looking for “LED” or “GRP”.

Extend the flow above with another Switch node (Separate messages by type) as in the figure below:

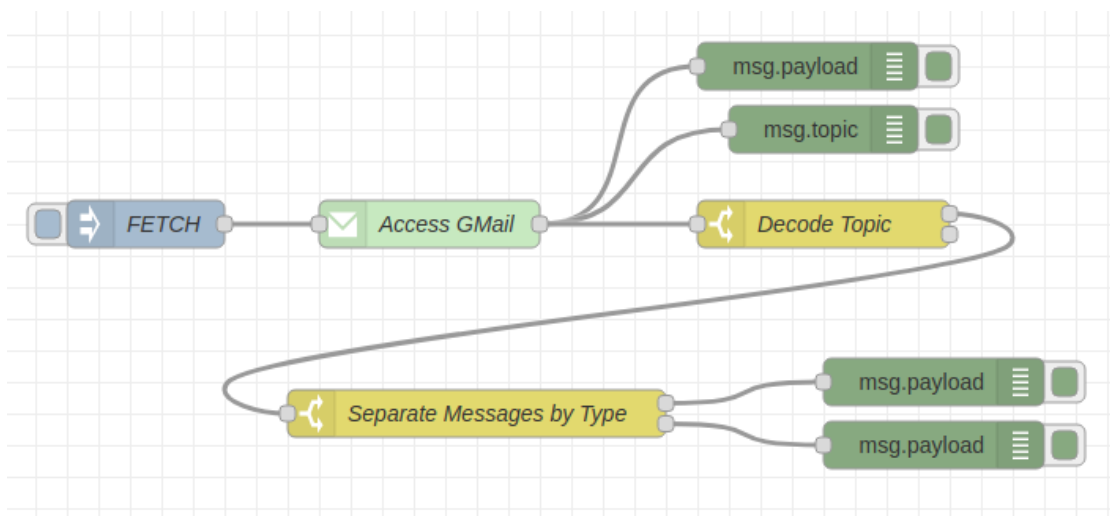


Figure 4-83: Separate Messages by Type Flow

Here is the configuration of the Switch Node (Separate messages by type).

## Chapter 4 LEDs - Preliminary (V0.0)

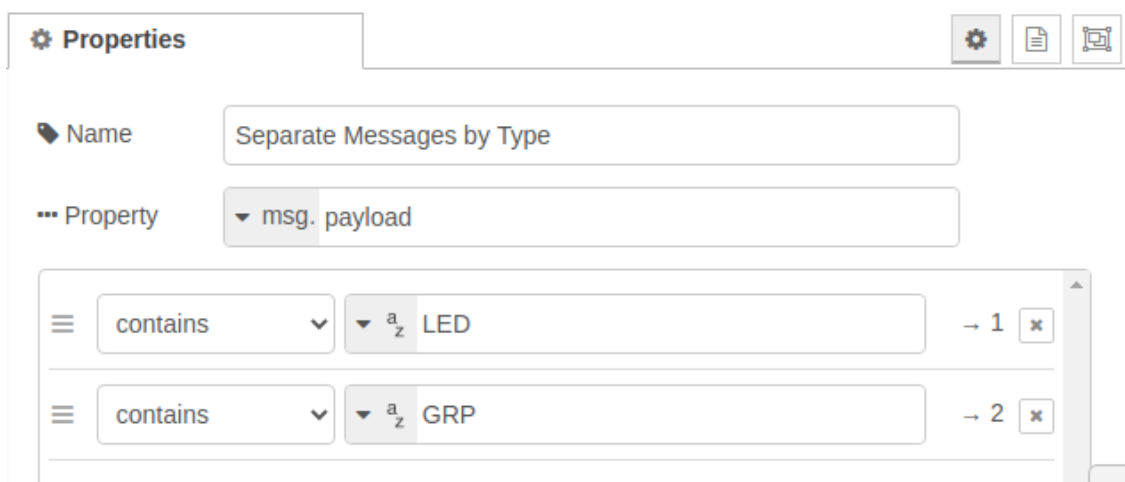


Figure 4-84: Separate Messages by Type Node Configuration

The operation here is simple. The first rule checks the payload (email text) for any string containing “LED” and sends the message to output 1. Similarly, the second rule checks the payload for any string containing “GRP” and sends it to output 2. There is something unusual here. The default configuration of the Switch node is to apply all the rules. If more than one rule applies the same message is sent to all the outputs where the rules apply. Thus if a `msg.payload` contains both “LED” and “GRP” each output will get a copy of the input message. This is a bit different from the train switch yard analog discussed in Chapter 3. It is more like a magic train switch yard where you can duplicate trains and send them to different tracks. Keep this in mind.

Once you get this flow set up, deploy it and try a few test Email messages. In some include an email text with, say, “LED1:ON”. In others include “GRP:7”. In another message include both “LED3:OFF” and “GRP:10”. Finally try a message that contains no substring of “LED” or “GRP”. Check the results and see what the Debug output shows. Pay special attention to the test case where the message contains both “LED” and “GRP”. In this case you should see the message twice in the output sidebar.

If all this works it is time to build flows that will decode each message to find the commands and then carry them out. The approach here is going to be very heavy handed. There are much better ways to do this, but they require unstanding of advanced topics like “regular expressions” and Javascript. Because this tutorial is trying to work with simple, introductory techniques you will be doing the command decoding the hard way, with heavy stone axes.



## Chapter 4 LEDs - Preliminary (V0.0)

As usual the best idea is to start off with a simple example and then build it out. To do this open a new tab and build up a flow like this.

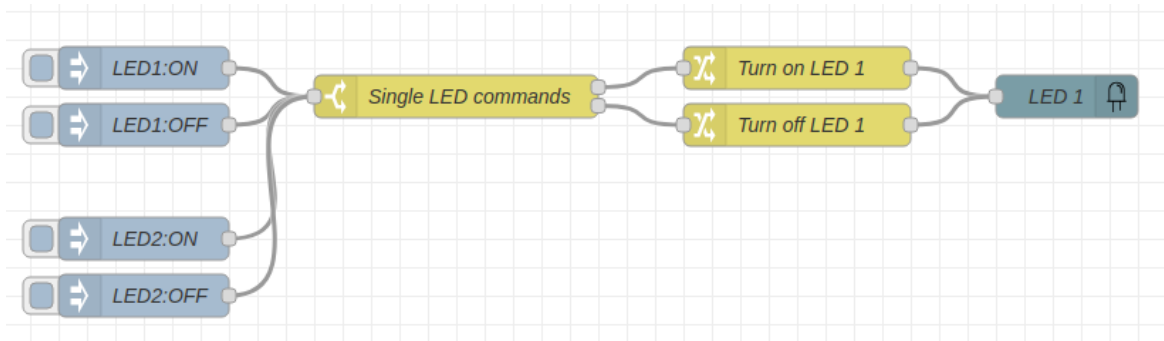


Figure 4-85: Single Mode Decode Test

The purpose of this flow is to make sure your understanding of single mode decoding is correct. On the left are Inject nodes (like LED1 ON) that inject a payload with text for single mode commands. The switch node (Single LED Commands) selects out messages containing “LED1:ON” and LED1:OFF” commands. The two change nodes (e.g. Turn on LED 1 and Turn off LED 1) simply generate a payload with a one or a zero to send to the LED 1 node.

On the left side are two Inject nodes (LED2 ON and LED2 OFF) that inject messages that should have no effect on LED 1 because the switch node will not recognize them. You can add other similar inject nodes to test whether their messages will change LED 1.

Here is how the Switch node (Single LED commands) is configured.

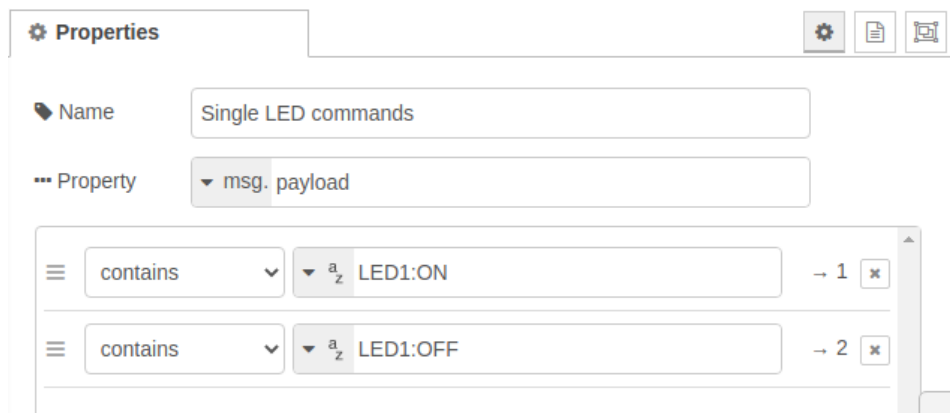


Figure 4-86: Single LED Command Node Configuration

Once you have this test flow working expand it to cover all eight of the single mode LED commands. You should have something that looks like this:

## Chapter 4 LEDs - Preliminary (V0.0)

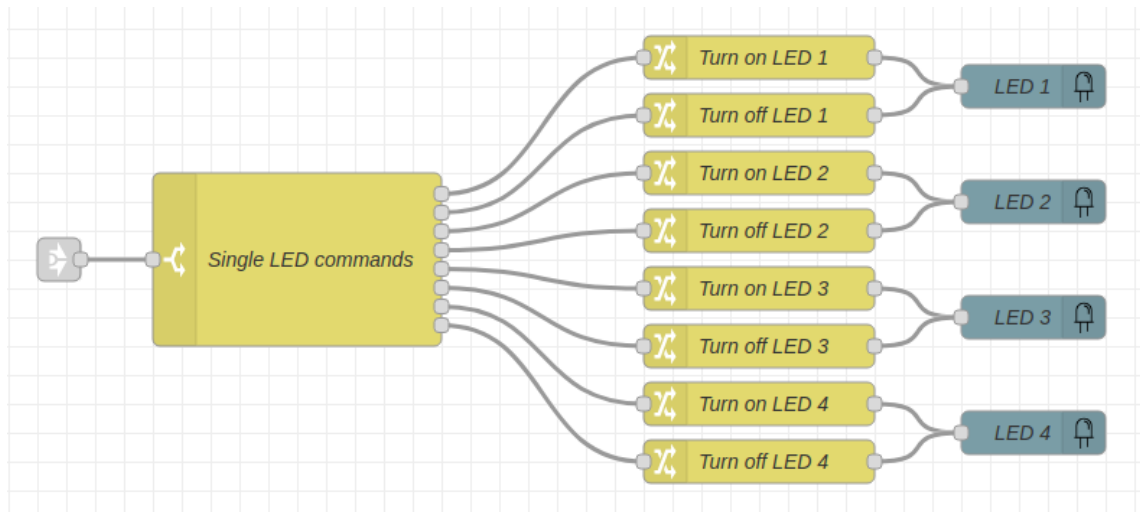


Figure 4-87: Single Mode LED Command Decode

This flow is just an expansion of the flow in Figure 4-85 above. On left side is a Link In node that you will later use to connect to the email access flow.

Onward, now it is time to build something similar for the group mode LED commands. It will be very similar to the flow above except it will be used to grab the group mode commands. Start simple and build a small test flow.

There is also a small issue here. Commands line "GRP:14" and "GRP:1" are very similar. Even worse the string "GRP:1" is contained within "GRP:14" (and "GRP:13", "GRP:12 ...). This means that you will need to do some fancy foot work to tell the commands apart.

As is traditional. start with a simple flow to see if it is possible to translate the commands to numbers. How about this?

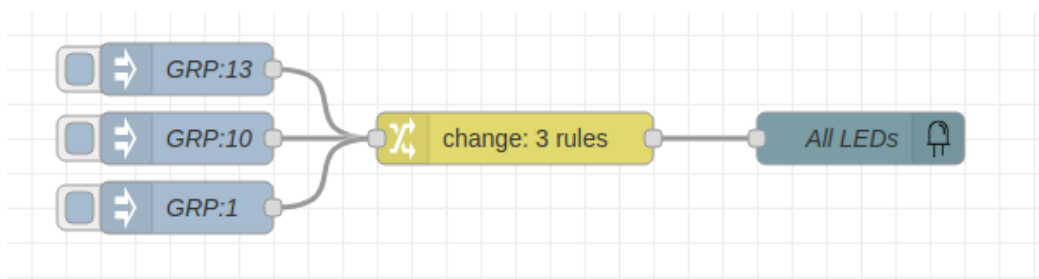


Figure 4-88: Group Mode Test

Here the Inject nodes on the left are test commands. The purpose of the change node in the center is to convert commands to the corresponding numbers. Usually, using the change node to convert between strings and numbers is straightforward but here there is a problem because "GRP:13", for example, contains the string "GRP:1". If you look for "GRP:1" it will match "GRP:13" also. The solution is to make sure you check for the longer string first. Once the longer string is converted the payload no longer

## Chapter 4 LEDs - Preliminary (V0.0)

contains a string but rather contains a number, so any other rules in the Change Node will not apply. This is a bit like the scheme you used to construct the Invert node earlier in this chapter.

Here is what the Change node in the example above looks like:

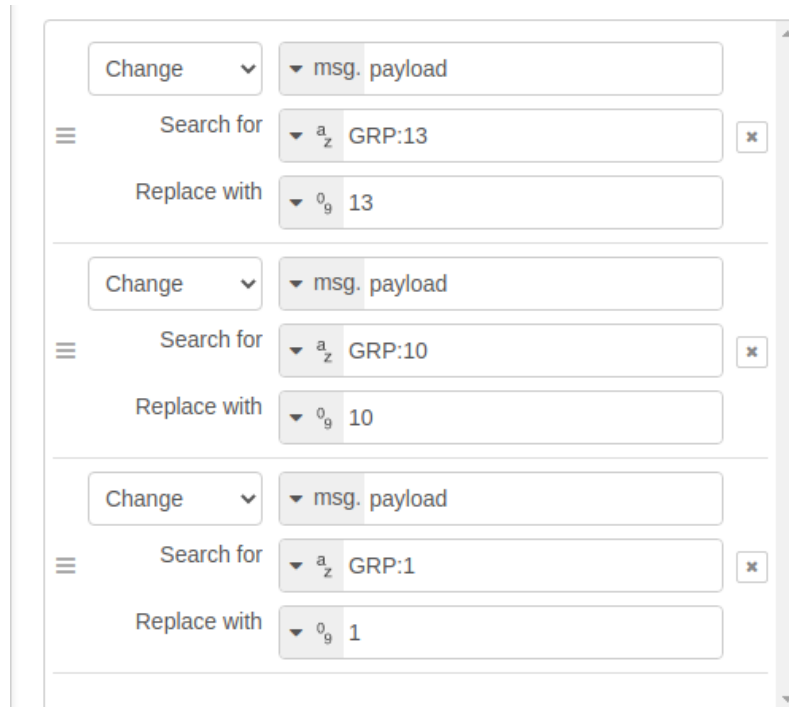


Figure 4-89: Change Node Configuration for Group Test

In the Change node configuration each rule tests for one command and converts it to the corresponding number. Notice that in the configuration the commands “GRP:13” and “GRP:10” come before the command “GRP:1”. As long as all the longer strings of “GRP:15” to “GRP:10” are at the top of the rule list everything will be fine. If you don’t believe this try putting the rule for “GRP:1” at the top of the list, deploy it, and see if the flow still works properly.

All that is needed now is to expand the Change node rules to cover all 16 group mode commands. This is very tedious, but necessary. Also you will want to put a Link In node in place of the Inject nodes so that you can connect all three flows together. Your final flow for decoding the group mode commands should look like this.



Figure 4-90: Group Mode Decoding Flow

## Chapter 4 LEDs - Preliminary (V0.0)

Time to go back and fix up the flow that does the topic detection and group decoding (Figure 4-83). This is going to involve getting rid of the Inject and Debug nodes, adding in the Email Receive node and adding links to connect to the two other flows. Get out your scissors and paste and build up the flow below from nodes you already have.

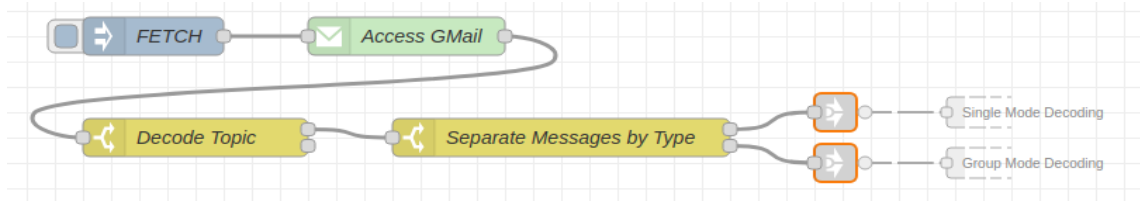


Figure 4-91: Email Topic Decoding with Links

The next step is to connect up flows. You can do this by putting all the flows on one page, or you can use the links to connect the flows. If you use links then review what you did previously in Figure 4-58. When you use the links here are a few things that can help out. First, if you hover over a link you can see the name of the link, if you gave it one. Second, if you select a link, as in the figure above, gray lines and text will appear showing you what flows that contain nodes it is connected to. Then you can go to that tab and see which node is involved. Here is an example,

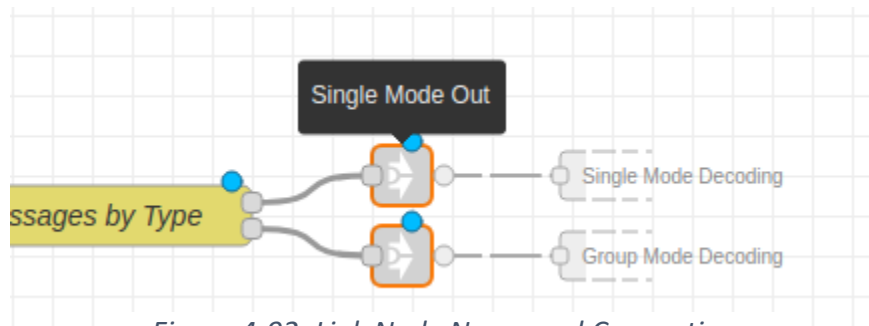


Figure 4-92: Link Node Name and Connection

For completeness here are the flows for the single mode decode and the group mode decode showing the links:

## Chapter 4 LEDs - Preliminary (V0.0)

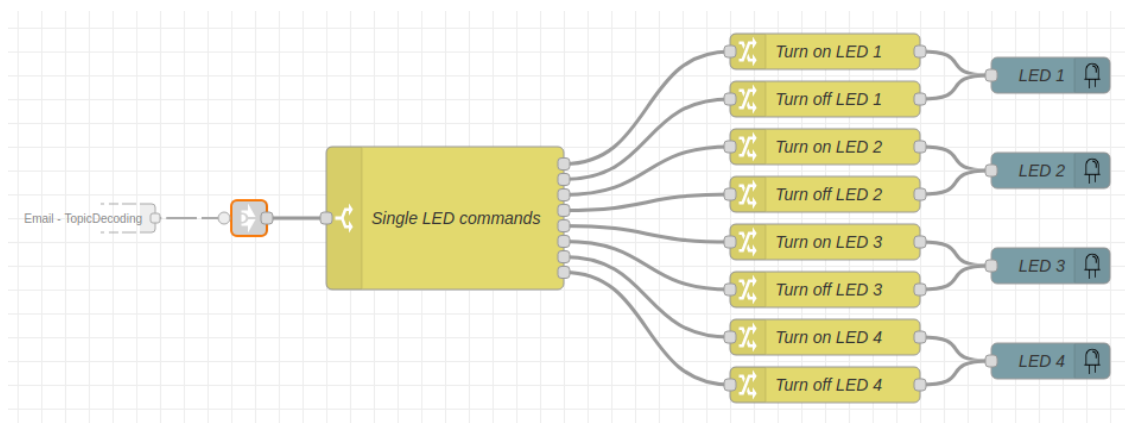


Figure 4-94: Single Mode Decode with Link

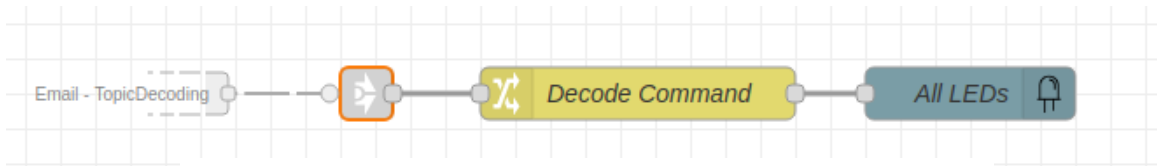


Figure 4-93: Group Mode Decode with Link

If you want to put your email flows to work you will need to change the FETCH node in the so that it injects a message periodically activating the Access Gmail and fetching any new messages. You probably do not want to fetch email messages any more frequently than every two minutes (see Figure 4-45 and Figure 4-46 to review how to inject messages periodically). You can also modify the Access Gmail node so that it scans for emails periodically by selecting the “Get mail” configuration from “when triggered” to “automatically”.

**Puzzle # 13 - Different Commands** – The commands used in the example above, such as LED1”ON are just arbitrary. You could use commands like “GATE OPEN”, “GATE CLOSED” to turn an LED on or off. Modify your flow so that it uses commands of your choice to control the LEDs in single mode.

**Puzzle # 14 - You’ve Got Mail** – Read the Help file carefully. See if you can work out a way to determine when you have mail that has not been read yet. If you can figure this out build a flow that will check your email file periodically and turn on an LED when you have mail. The LED should stay on until you have read the email the way you usually do. Go for the Gold: work out a way to see if you have unread messages from four of your friends. Turn on an different LED for each friend if you have unread emails from them. Be careful, don’t mark the emails as “read” otherwise, you might not know which are the new emails.

**Puzzle # 15 - Phone a Friend** – From Chapter 3 you know how to generate an email when the button is pushed. That email could be an LED command like LED1:ON. Now work with a friend who has a Rapsberry Pi and Learning Card to build a system where pushing a button on one system will light up an LED on the other system. The LED should stay on for 10 seconds and then turn off. Not an easy project, but you have all the knowledge you need to pull it off.

A Final Note: This use of email to control LEDs is a bit strange because of the time delays invovled. There is also another problem: it is “brittle”, a word that is used to discuss applications that can fail easily.

## Chapter 4 LEDs - Preliminary (V0.0)

Decoding is done here by requiring exact command matches. If you use a small letter in a command instead of a capital letter it will not be detected. Similarly for the topic decoding. There are better ways to do the decoding, but they depend upon advanced techniques, like regular expressions and JavaScript, in a word “real” programming.

### Weather to LEDs

Here is a project that will allow you to display the real time temperature for any place on the earth on the four little LEDs of the Learning Card.

“How in the world (so to speak) is this going to be done”, you might ask. Easy... there’s a node for that, no kidding there really is. This example will demonstrate one of the key features of Node-RED and its associated libraries of nodes. Namely, how information from the internet may be captured and used to control your environment. In this case, it is the lowly LED, but it could be any of the other ports on the Learning card as you will find out in later chapters.

You will make use of two nodes that are designed to pick up information from OpenWeatherMap.com and use that information to drive a display. OpenWeatherMap.com captures information from a range of sources and integrates it into weather forecasts. In your case you will be using its ability to provide current weather conditions for your location.

Cost: for any reasonable use that you might make of the service it will be free. With the free service you will be limited to 1,000,000 accesses per month and no more than 60 accesses per minute. You should be good unless you write some loop that gets out of control.

Later you will download the nodes specifically designed for accessing OpenWeatherMap.org. However, to make use of the nodes you must have an account. Once you have an account you will be able to create and download an “API<sup>28</sup> key”. This is a type of password that controls net access to the service. Without this key, which identifies you, the service might be overwhelmed with requests from bad actors. Getting an API key from OpenWeatherMaps.org requires that you provide your email address. Think carefully about this before you proceed, just as you should for any service you sign up for. Once you give someone your email address you might be flooded with all sorts of emails for free hamster giveaways and then bombarded with ads for required accessories, like tiny Santa hats. Consider yourself warned.

As an extra precaution you might consider setting up a new and unique Gmail account just for experimenting with the OpenWeatherMap node. The accounts are free and if you start to have trouble with an email flood, for example, you can always pull the plug on the account.

---

<sup>28</sup> API – Application Program Interface. Your Node-RED flow is an application program. The API is a mechanism to access a remote resource over the internet and to control that resource. Specifically, you are going to access the API for OpenWeatherMap.com and ask it to look out the window and tell you what the weather is. Fortunately, all the hard work is hidden in the OpenWeatherMap nodes you will download.

## Chapter 4 LEDs - Preliminary (V0.0)

Open your Web Browser and go to [openweathermap.org](https://openweathermap.org). As usual the instructions here are brief (no pictures) because if you cannot set up access to this service with only the instructions below, then you probably should not be using it.

- Click on the API tab at the top of the home page.
- Click the Subscribe button under “Current Weather Data”.
- On the Pricing page you are going to use the Free option. Under “Free” click “Get API Key”
- You will be taken to the “Create Account” page. Fill it out as requested and prove that you are not a robot.
- You will receive an email to verify that you really are not a robot.
- Once you have verified that you are flesh and blood rather than nuts and bolts you will be taken to a general services page at OpenWeatherMap.org.
- Click on API Keys. This will take you to a page with your assigned API key. It is long and it is not only unique in our galaxy and the known Universe, but also in all the parallel and multiverses surrounding us. Copy it and save it for later in a file. If you lose the API key, you can always log on and retrieve it.

Now let’s see if we can put this to good use. You will be downloading two nodes that allow your Node-RED flow to find out the weather for any point on the earth. The nodes do this by accessing the database at OpenWeatherMap.org. One node provides weather periodically (every 10 minutes) and the other will provide weather when it is triggered by a message.

First, download the nodes. Do this by clicking on the main menu and selecting “Manage Palette”.

- Select the “Install Tab” ①.
- In the search box type in “openweather” ②. This will display a selection of nodes downloads to choose from. Select the one labeled “node-red-node-openweathermap”.
- Click “install” ③.

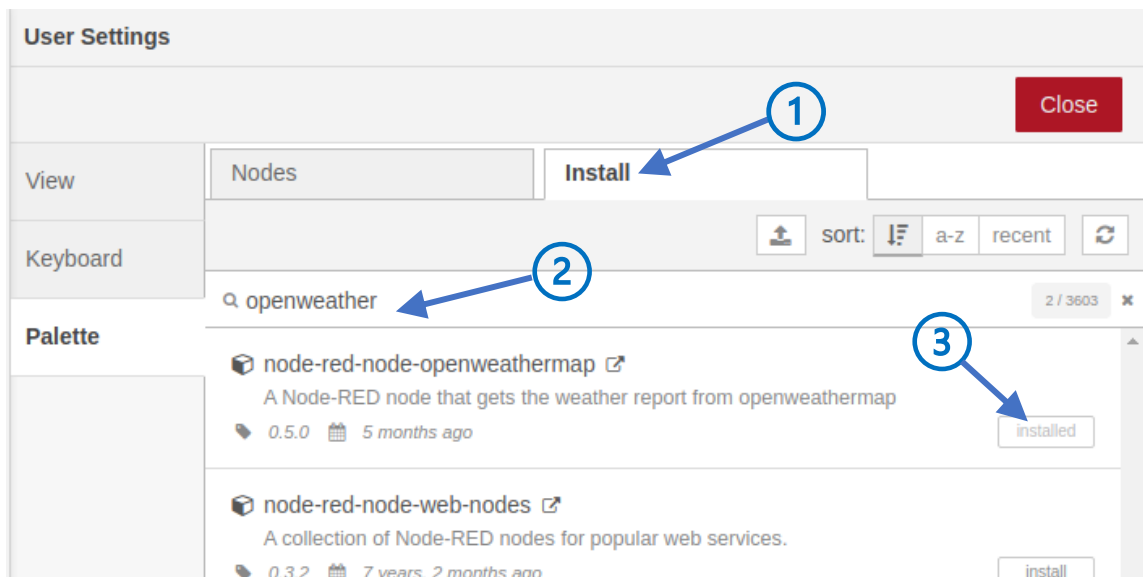


Figure 4-95: Downloading OpenWeatherMap Nodes

## Chapter 4 LEDs - Preliminary (V0.0)

The two new nodes will be added to your palette under the title “Weather” probably near the bottom of the palette as below:

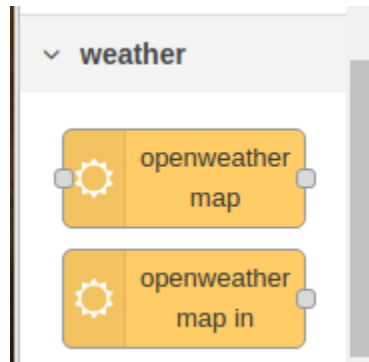


Figure 4-96: Weather Nodes in the Palette

Following the KISS principle begin by building the simplest flow possible to see how the nodes work. Start with the node that only has an output. This is the node that provides a weather report automatically and periodically.

- Clear the workspace by saving any flows you care about, adding a new flow tab and then deleting all the other tabs. Doing this prevents interference from other flows you might have laying around because when you click Deploy all the flows in your workspace are activated.
- Drop in the Openweathermap In node (the one that only has an output) and connect it to a Debug node as in the figure below.

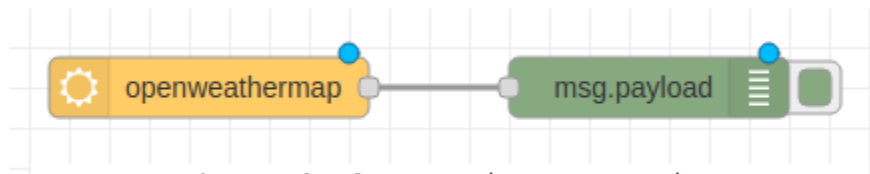


Figure 4-97: OpenWeatherMap Test Flow

As usual, listen to the wisdom of whispering hamsters and read the help file for the Openweathermap In node. Or you can ignore the wisdom and jump right in, get your feet tangled up and fall flat on your face. Your choice.

*[Time passes while you heed the whispered wisdom and read the Help File]*

Configure the node.

- Double click on the node to open the properties dialog.
- Enter your API key ① from above.
- The dropdown box below the language box should say “Current weather for” ②.



## Chapter 4 LEDs - Preliminary (V0.0)

- Enter the name of your city ③.<sup>29</sup>
- Enter your country ④.
- Give the node a name ⑤.
- Click DONE.

Figure 4-98: Configuring the OpenWeatherMap In Node

**Deploy** your flow and if you did everything correctly you will immediately get a weather report in the debug side bar that looks like the figure. If you made a mistake, you will get an error message, especially if the system is not able to identify the city.

---

<sup>29</sup> Surprisingly, there does not appear to be a way to enter your state, province, shire, parish or whatever. If you live in a fairly large city, like Cupertino, CA (World Headquarters of Sequent Microsystems) you will be okay. Once you get your first weather report you can check the latitude and longitude and see if you have the right city. If not, you will need to put in the coordinates of your location instead. You can just ask Google for the coordinates of your city.

## Chapter 4 LEDs - Preliminary (V0.0)

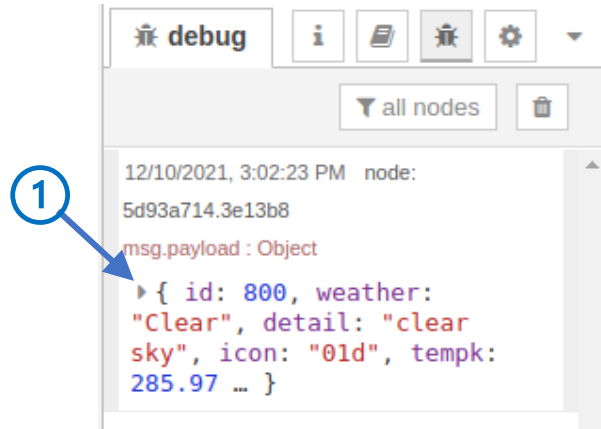



Figure 4-99: Initial Weather Report for Cupertino, California

## Chapter 4 LEDs - Preliminary (V0.0)

Normally, debug window shows only a compressed view of the weather report. Click on the tiny arrow  and you will see an expanded view of the report.

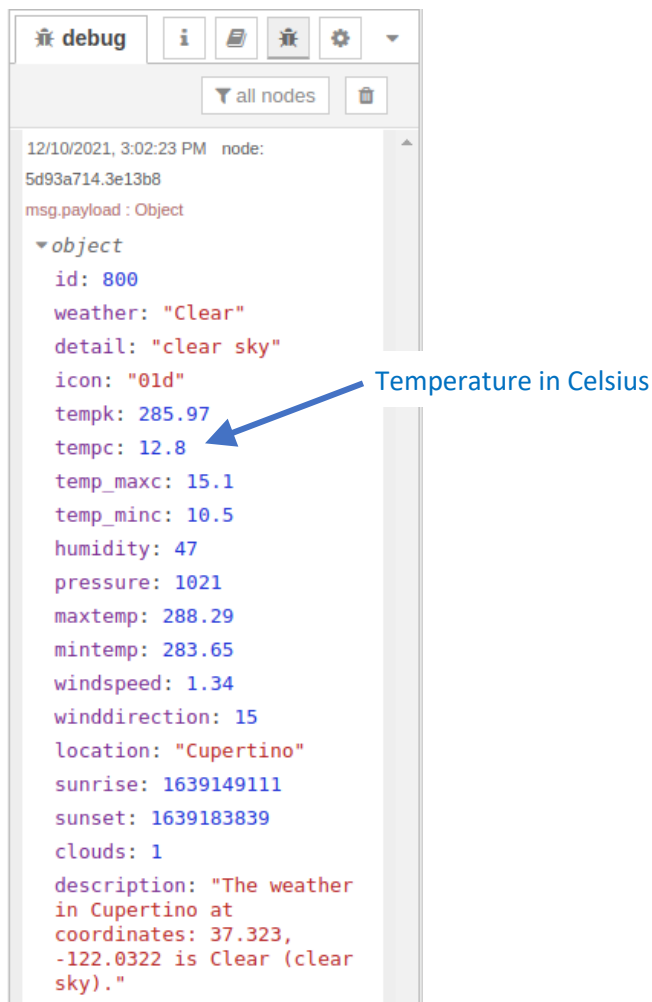


Figure 4-100: Expanded Weather

Each piece of information is neatly identified with a name followed by the data. In some cases, you can click on the data and change the format to one that suits you, like local time rather than milliseconds since the birth of Unix.

### Internet Thermometer

Before you can connect to the Thermometer Display you will need to get the temperature data from the OpenWeatherMap In node. The payload from this node contains an object. This object is just a collection of information representing the current weather. Pretend for now that it is a paper form where the names of fields in the form refer to weather parameters and the data next to the names is the current data you want. Conveniently, the OpenWeatherMap In node does all the heavy lifting for you. It calls up the OpenWeatherMap.org site and asks for the current weather. Then it fills out a form with temperature, wind speed, wind direction, humidity and so forth. After that it stuffs the completed

## Chapter 4 LEDs - Preliminary (V0.0)

form into the payload and hands it to you. If you know the name of the box containing the information you want, you can use a Change node to form a new message with that data in it.

- Grab a Change node and connect it to the output of the OpenWeatherMap In node.
- For good measure attach a Debug node to the output of the Change Node.

You want your flow to look like this.

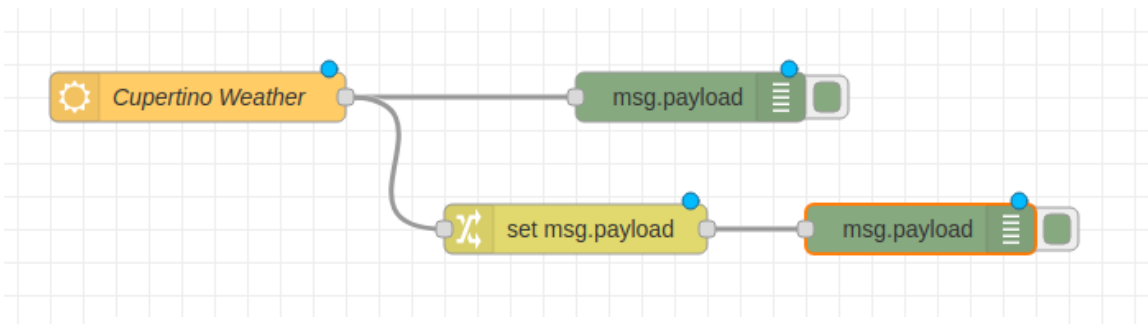


Figure 4-101: Weather Flow with Change Node

- Double click on the Change Node to open its properties dialog.
- Cook up a rule that will set the output `msg.payload` to the value in `msg.payload.tempc` ①. “msg.payload” has the weather “object”. By appending “tempc” to “msg.payload” you are asking the Change node to open up the payload and find that part of the weather report named “tempc” (the current temperature in Celsius). The Change node does this for you and puts the result in the payload of the output message it sends onward.
- Change the name of the Change node to something intelligent ②.

Your Change node edit dialog should look like the figure below when you are done.

## Chapter 4 LEDs - Preliminary (V0.0)

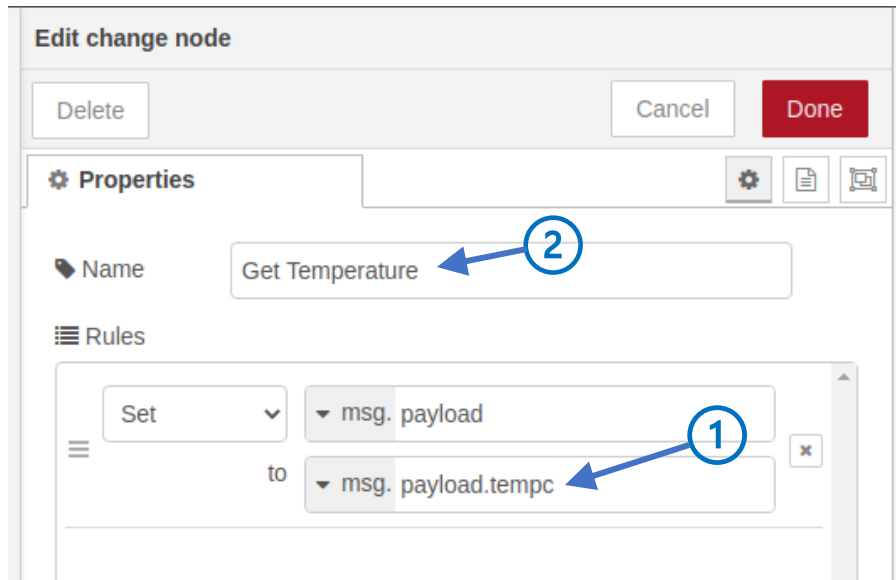


Figure 4-102: Configured Change Node

- Click Done.

### DEPLOY AND ENJOY!

As soon as you deploy the flow it will retrieve a weather report and print it. It will also print the output of the Change node (now called “Get Temperature”). If everything went according to plan at the top of the debug output you should see this:



Figure 4-103: Temperature in Degrees Celsius

You might click on the little triangle next to “Object” to expand the view (blue circle). Check to see if the value in “tempk” is the same as your output from the Change node (in the blue box).

## Chapter 4 LEDs - Preliminary (V0.0)

Next consideration – connecting to the Thermometer Display flow.

Your Thermometer Display works in Fahrenheit but if you look at the debug output from the OpenWeatherMap In node you will realize that the current local temperature “tempc” is in degrees Celsius. This means you will need to do a little conversion. But fear not... as always: “There’s a node for that!”

- Go to the main menu and click on the “Manage Palette” option.
- Find the Range node

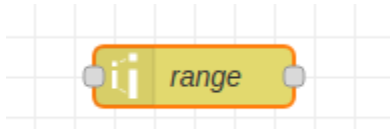


Figure 4-104: Range Node

- Read the Help file for goodness sakes!

Briefly, the range node maps numerical values from one range to another range using linear scaling. This will work perfectly for converting from degrees Celsius to degrees Fahrenheit. You don’t even need to know the conversion formula because the range node will figure all that out for you.

- Wire the input of the Range node to the output of the OpenWeatherMap node.
- For good measure attach a Debug node to the output of the range node so that you can check its operation.

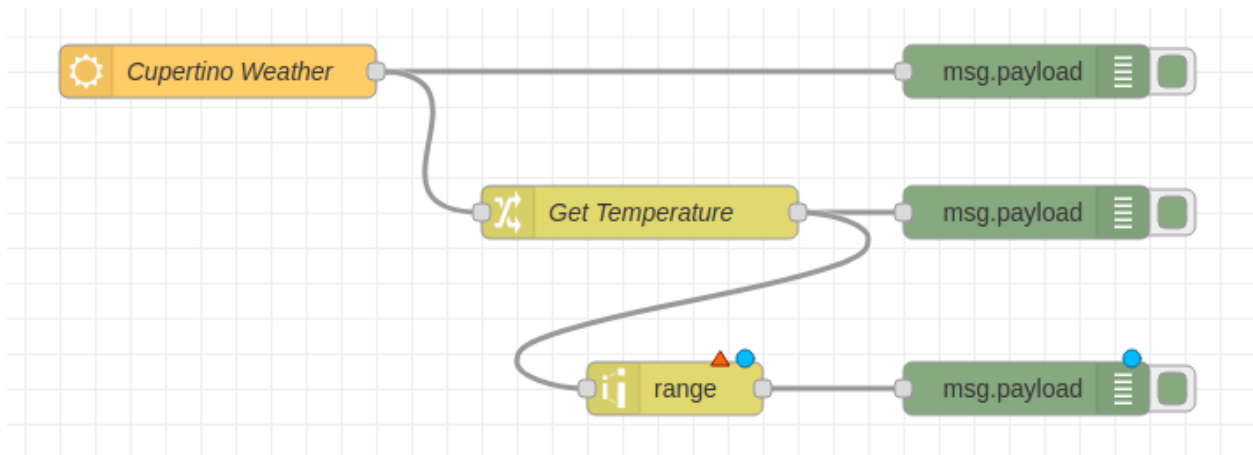


Figure 4-105: Weather Flow with Range Node

Did you notice the little red triangle above the Range node? This means that the node is useless until it is configured. Unlike other nodes you have used, the Range node has no default setting. After all, what

## Chapter 4 LEDs - Preliminary (V0.0)

would be a reasonable setting? If you try to deploy before you configure the Range node you will get an error message. Let's fix it!

- Double click the range node to open the edit dialog
- Give the node a useful, descriptive name ①.
- Set the input temperature range as shown below ②. Here you want to pick range limits that cover a reasonable temperature range in your location. Remember that this range is in Celsius.
- Set the output temperature range ③. The values you put in here must be the Fahrenheit values corresponding to the Celsius values in the input temperature range. Check the conversion very carefully because if you do not set these values correctly the conversion will be wrong.
- Click Done!

The screenshot shows the configuration dialog for a Range node. The 'Property' field is set to 'msg. payload'. The 'Action' is 'Scale the message property'. Under 'Map the input range:', the 'from' field is '-10' and the 'to' field is '40'. Under 'to the target range:', the 'from' field is '14' and the 'to' field is '104'. The 'Round result to the nearest integer?' checkbox is unchecked. The 'Name' field is 'C to F'. Blue circles with numbers 1, 2, and 3 are overlaid on the image, with arrows pointing to the Name field, the input range fields, and the target range fields respectively.

Figure 4-106: Configuring Range Node

DEPLOY your newborn flow!

The output should look something like shown below. Check to be sure that the “tempc” value and the output from the Range node in Fahrenheit agree mathematically (just ask Google to translate Celsius to Fahrenheit).

## Chapter 4 LEDs - Preliminary (V0.0)

```
12/11/2021, 12:50:54 PM node:
5d93a714.3e13b8
msg.payload : Object
  > { id: 801, weather:
    "Clouds", detail: "few
    clouds", icon: "02d", tempk:
    286.64 ... }

12/11/2021, 12:50:54 PM node:
ccd4237c.9f1b5
msg.payload : number
13.4

12/11/2021, 12:50:54 PM node:
98d517f0.d4ab68
msg.payload : number
56.12
```

Figure 4-107: Current Temperature Output

If everything looks good at this point it is time to bring in the temperature display flow and connect it up.

In a previous discussion you built an LED display that would act like a thermometer (Figure 4-18). Now you can get the data to display directly from the internet.

Proceed like this:

- Open a new flow Tab
- Import that Thermometer Display flow shown in Figure 4-18.
- Copy the temperature flow over to the Weather flow
- Delete anything connected to the input of the Switch node (like the Inject node you used for testing).
- Connect the output of the “C to F” node to the input of the Switch node.

Does your flow look like the figure below?



## Chapter 4 LEDs - Preliminary (V0.0)

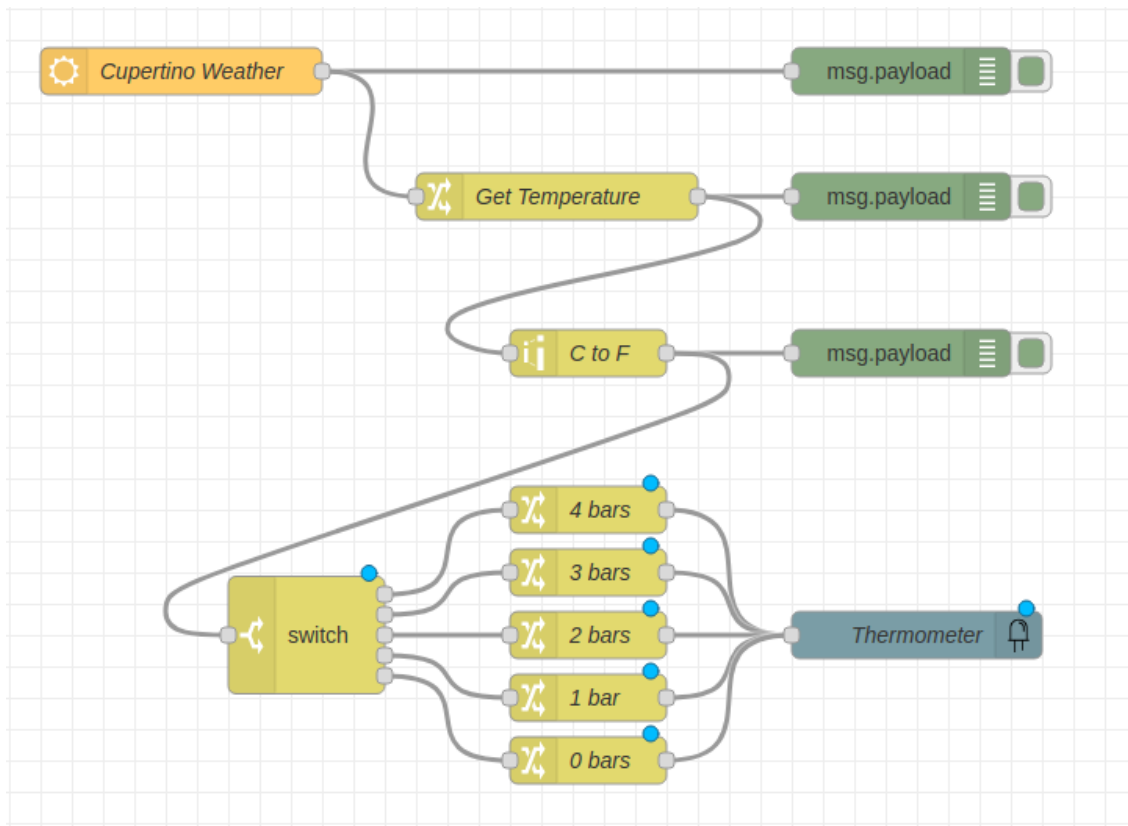


Figure 4-108: Completed Weather Flow

### DEPLOY!

If everything was done correctly the OpenWeatherMap node will read the current temperature, convert it and pass the result to the Thermometer display. Check the temperature value in the Debug window and make sure that the correct number of LEDs are lit. Leave your flow running and see if the number of bars changes when the temperature passes from, say, two bars to three bars.

Note: depending on where you live and the time of year you might need to adjust the parameters in the Switch node of the Thermometer Display so that the display changes once or twice a day. For example, you could set up the Switch node so that the temperature values are spaced on 10 degrees apart. In a later chapter you will learn to build better displays.

And now it's Puzzle Time!

**Puzzle # 16 - What Should I Wear?** – Modify the Weather Flow so that a different LED light up to show you what to wear outdoors. For example, you might divide the four lights so that they show: Shorts, Long Pants, Coat, Overcoat. This should be simple because you only need to adjust the thresholds in the Switch node.

## Chapter 4 LEDs - Preliminary (V0.0)

**Puzzle # 17 - Tell Me What To Wear** – Pull together all your knowledge of the pushbutton, sound, LEDs and the weather to build a bedside device that will tell you what to wear depending upon the weather. It should work this way: when you push the pushbutton it will tell you out loud what to wear based on the current temperature and weather. Something like: “Wear a raincoat Fool!”, “Wear your overcoat, Frosty!”, and so forth. You might need to combine several parts of the OpenWeatherMap report to determine which advice to speak out.

**Puzzle # 18 - Wind Direction** – The Openweathermap In node also provides wind direction. Modify the Weather Thermometer above so that one of the four LEDs light to indicate the general direction of the wind (North, South, East, West) or you might consider lighting two adjacent LEDs if the wind is from NE, SE, SW, NW in addition to the single LED.

**Puzzle # 19 - Wind Speed** – Enhance the Wind Direction indicator to show both the direction and the wind speed by flashing the LEDs at a rate that is proportional to the wind speed. If the wind speed is below some threshold, then all the LEDs should be off.

**Puzzle # 20 - General Weather** – The Openweathermap In node provides a short summary of the weather. Develop a way to show this summary on the LEDs. To do this you will need to research the possible text strings provided by the service. Then you will need to develop logic that condenses the weather into a code to display on the LEDs. You might consider using flashing the LEDs to provide some additional information.

**Puzzle # 21 - Porch Light** – Pretend that the LEDs are your porch light (not a very helpful porch light, but nonetheless it’s what you have to work with). Design a flow that will turn your porch light on at sunset and off at sunrise. “What!?!”, you ask. “How is my flow going to know when it is nighttime?” Well, of course, there’s a node for that. You are on your own here... look around and see if you can find a node that signals sunrise and sunset. You might start by looking in the Manage Palette section as you did above for the thermometer flow. Or look on the internet. Check out the OpenWeatherMap data, which may include the sunrise and sunset times. There are several nodes that work like the OpenWeatherMap nodes and tell you when the sun rises and sets in your city. Once you find one you know what to do: read the help file, build a simple system to show that you know how to use the node, build the final system, test everything out. Good Luck!

## Learning Kit Workbook (version 1.3)

### Chapter 5 – Opto-Isolated Inputs

What you will learn:

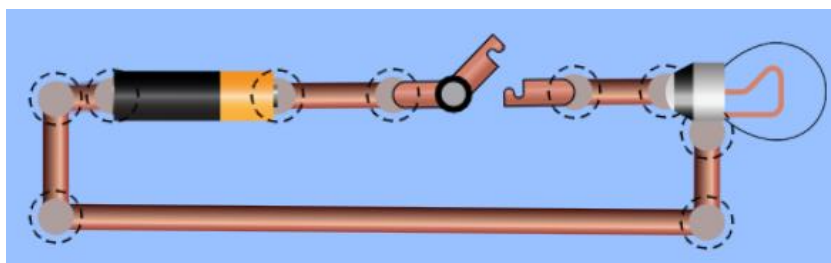
- OPTO Node
- Using switches as input
- Context Data Variables
- Stopping Loops
- Counting
- Simple JSONata Expressions
- User Interface Widgets – **Coming Soon**

#### Introduction

It's time to jump off the Learning Card and start interacting directly with the world. In keeping with our SSS policy let's begin with the switch, the simplest of all electronic devices. Switches come in an amazing array of sizes, shapes, functions and capabilities, but they all do the same things, which is to turn a mechanical action into the opening and closing of a circuit.

If you are know a bit about electronics and circuits you should skip ahead or risk boredom.

Circuit, circus, circle, curriculum, circumnavigate all from or related to the Latin word "circum" which means to go around<sup>30</sup>. In a circuit electrical current flows in a loop unless the loop is physically broken



*Figure 5-1: Flashlight Circuit*

someplace, in which case the current does not flow. The circuit shown below is just about as simple as a circuit can be and if you look at it for a minute you will realize it is the circuit that could be a flashlight.

In order from left to right is the Battery, Switch and Lightbulb. In the figure the switch is "open" meaning that no current can flow in the loop. "Close" the switch and current can flow and the lamp will light up. If you get the Phet Simulator from the University of Colorado ([www.phet.colorado.edu](http://www.phet.colorado.edu)) you can build circuits like this and learn basic electricity. It's free, and it's fun to play with. Phet only supports basic circuits (batteries, switches, lamps, resistors), but you can learn quite a bit if you are just getting started in circuits.

---

<sup>30</sup> [Eytmologeek.com](http://Eytmologeek.com) – an interesting (and apparently safe site) giving the etymology of many words in the form of charts. This site integrates etymological data from other sites. Use with care. You get what you pay for and your milage may vary.

## Learning Kit Workbook (version 1.3)

Click on the switch in the simulator and you can turn on the lightbulb because now the switch is “closed” and the circuit forms a complete loop from the tip of the battery to the back of the battery. The electrons are happy that they can run around this loop (like the hamster on an exercise wheel) and as they pass through the lightbulb, they are pleased to give up a little energy and heat up the filament to produce a little light, and in the case of incandescent bulbs they produce a lot of heat too.

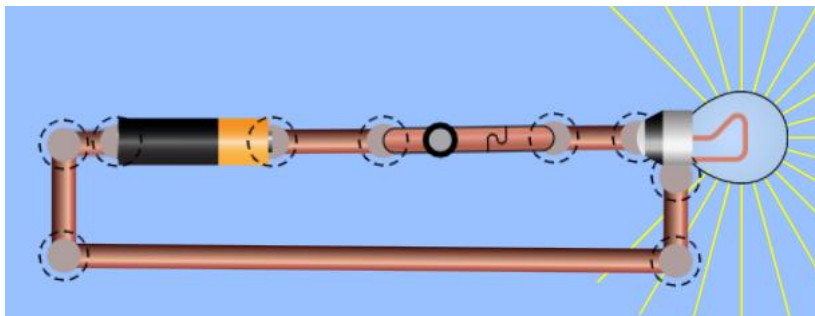


Figure 5-2: Flashlight Circuit with Switch Closed

To make these concepts simpler engineers usually draw the circuits as “schematic” where a specific physical component is replaced by a generic symbol. In the circuit below the switch, battery and bulb from Figure 5-1 have been replaced by standardized symbols, which look almost like the physical element they represent<sup>31</sup>.

Light switches in your house let you manually control the flow of current. Similarly in your car, flashlight, doorbell, remote control and so forth. However, there are also all sorts of switches that can sense the environment like a thermostat that are activated by temperature, humidistats that are activated by humidity and float switches activated by water level. All use the same principle: one condition (e.g., temperature too low) closes the switch and another (temperature high enough) open the switch.



Figure 5-3: Schematic Drawing of Flashlight



---

<sup>31</sup> The simple symbols in the figures have been around since the dawn of the electrical age. The symbols for the switch and light bulb are obvious because they look like very much like the physical item. But what of the battery? That symbol derives from the very early form of batteries which were simply two different metal plates separated by a chemical compound (salt water will do). The batteries of today are still formed from two plates separated by a chemical compound, but they are much more sophisticated. A word of warning to the curious. Do not take batteries apart unless you know what you are doing. The ordinary batteries from the grocery store have caustic chemicals which are incompatible with your skin, eyes, nose, mouth etc. Lithium batteries are even worse. If you pry one of these apart it will most likely burst into flame. Being doused in caustic chemicals is bad enough, but if they are also on fire... well you get the idea. For this reason, you cannot check lithium batteries in your airline luggage.

## Learning Kit Workbook (version 1.3)

Switches and Where to Find Them – The Learning card has four optically isolated switch inputs. If you want to build some of the flows in this chapter you might need as many as four switches or push buttons. For switches you can go the local hardware store and buy ordinary wall switches for about a dollar. If you want to have pushbuttons stay away from hardware store because doorbell buttons are very expensive (\$10 each!). You can go online and order pushbuttons for about a dollar each if you buy 10 of them and they might even come prewired. You know where to look.

### Connecting Switches to the Learning Card

Your learning card has four optically isolated connections for switches referred to as IN1, IN2, IN3 and IN4. You connect to the input using two three-pin connector plugs (see figure ). The plugs attach to the board through the connectors on the card edge as shown below ().

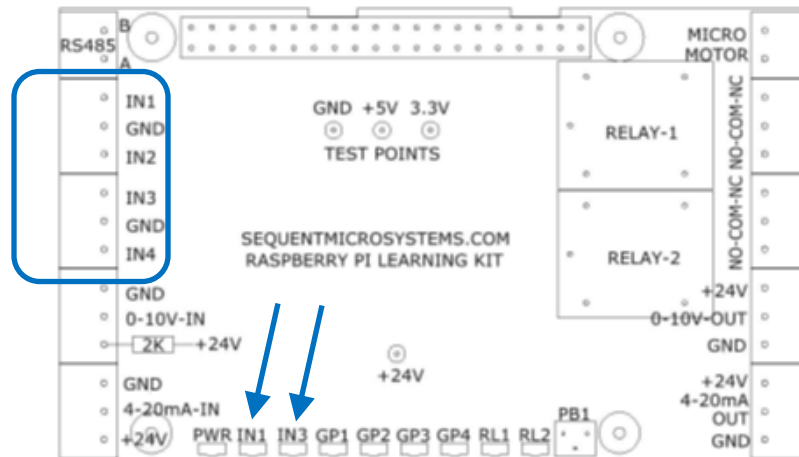


Figure 5-4: Learning Card OPTO Input Connections

A switch is wired between one of the inputs (e.g., IN1 and ground as shown schematically in this figure:

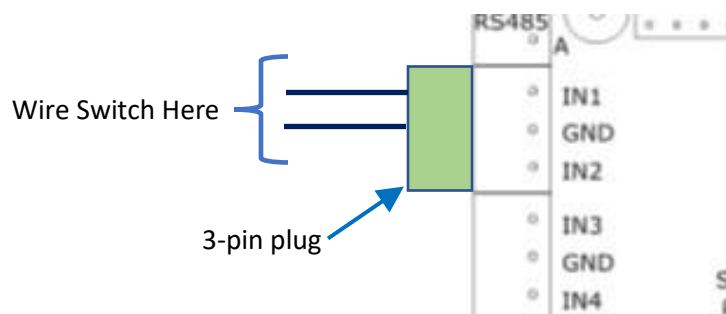
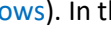


Figure 5-5: Wiring a Switch to OPTO Input IN1

Here is a little detail you should know about the Learning card. Closing the circuit on OPTO inputs IN1 and IN3 will activate LEDs labeled IN1 and IN3, respectively (). In the examples below you will be using OPTO input IN2 so that you can avoid confusion when you use an OPTO input to turn on an LED. Avoiding confusion always makes your day better.

## Learning Kit Workbook (version 1.3)

Below is a simple schematic of the IN1 circuit, which is representative of the other three inputs. Each input is connected to a separate opto-isolator chips which provides an electrical separation between the switch and the microcontroller on the Learning Card. An opto-isolator is a simple device composed of an LED and a photo transistor. When current flows through the LED it produces light which activates the photo transistor. This in turn allows a current to flow through the phototransistor that activates an input to the microcontroller.

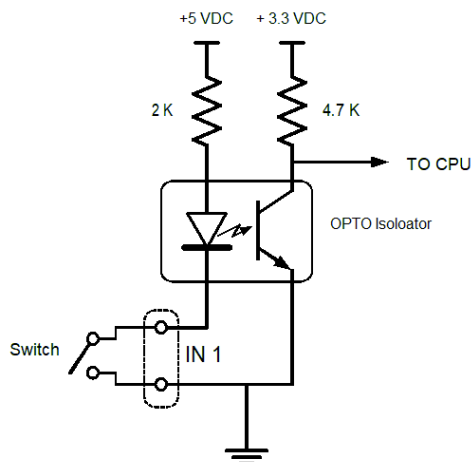


Figure 5-6: OPTO Input for IN 1

When the switch is open no current flows in the LED side of the opto-isolator and therefore no light is generated by the LED inside the chip. As a result, no current flows in the phototransistor and the input to the micro controller is at a high voltage. If you close a switch connected to IN1 then current flows through the LED, light is generated and consequently current flows in the phototransistor. When this happens the input to the microcontroller goes to a low voltage level.

That's all there is to it. The microcontroller on the Learning card simply senses whether an input is connected to ground (switch closed) or not (switch open).

And now you have come to the point where you need to think about safety for the first time. This is not just about keeping you safe, but about keeping the Learning card from turning into a brick. Read on!

### Be Careful!

There are some rules you should follow to prevent harm to your system.

- DO NOT apply voltages to the opto-isolated inputs. – If you apply voltage to an input, you may damage your Learning Card or your Raspberry Pi (or worse, both). The opto-isolated inputs are only intended to detect switch closures.
- Connect the INx input and corresponding ground only to the switch. – The circuit to your switch must be completely isolated from all other circuits. Do not connect the ground to any other ground circuit. If you do this, you risk damage to your Learning Card.
- Keep connections to the switch as short as you can. – The opto-isolated inputs are not intended for long runs of wire because they are susceptible to electrical interference from nearby

## Learning Kit Workbook (version 1.3)

electrical devices, like motors, dimmer switches and so forth. A practical limit would be a distance of tens of feet rather than hundreds of feet.

- Use twisted pair wire to connect your switch. – For simple test purposes where the switch is only a few feet from the Learning card you can use ordinary wire to make the connections. However, for longer runs (over say five feet) you should use two wires that are twisted together (as you might expect this is called a ‘twisted pair’). You can buy the wire already twisted or make your own by hand. A twisted pair of wires helps to prevent stray electrical signals from being detected by your Learning Card.



*Figure 5-7: Twisted Pair*

### Crawling with the Switches

By now you know the drill: read the documentation and start simple. Save all your flows (or at least the ones you really care about), add a new flow tab, delete your old flows. Go to the Palette and under Sequent Microsystems and grab an LKIT OPTO In Node.



*Figure 5-8: LKit OPTO Node (Input Mode)*

Now is the perfect time to review the Help file by selecting the node and opening the Help tab (the little book icon). Here is part of what you will see:

## Learning Kit Workbook (version 1.3)

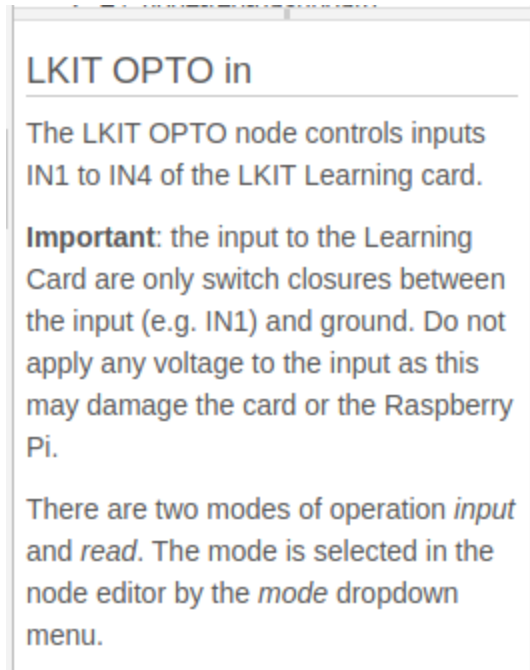


Figure 5-9: OPTO Node Help File (Partial)

The first thing to notice is that the node has two modes: *input* and *read*. When you load the node, it will be in the *input* mode, but you can switch to the other mode in the Mode dropdown. *Input* mode means that the node will generate a message whenever the selected input (e.g., IN1) changes. When you select *read* mode the node will change visually to add an input port. When this port receives a message (any type of message) the node will generate a message representing the current state of the selected input.

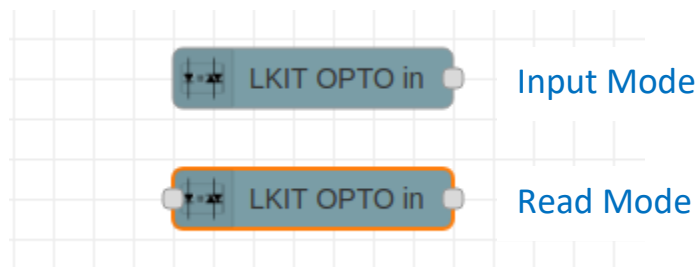


Figure 5-10: Two OPTO Nodes in Different

### Input Mode Operation

When first loaded the default mode of the OPTO node is Input. This means that when selected input changes the node will generate a message where the *msg.payload* value is either Boolean *true* or Boolean *false*. That's it, that's all she wrote. The node generates a Boolean *true* whenever the selected input changes from open to closed and conversely it generates a Boolean *false* whenever the input transitions from closed to open.

Time to try it out.



## Learning Kit Workbook (version 1.3)

- Bring in a Debug node.
- Wire the output of the OPTO node to the input of the Debug Node.
- Open the OPTO node editor and make sure that input channel 2 is selected. This channel corresponds to the IN2 input.
- Give your node a useful name, like “IN 2”

Your flow might look something like this:

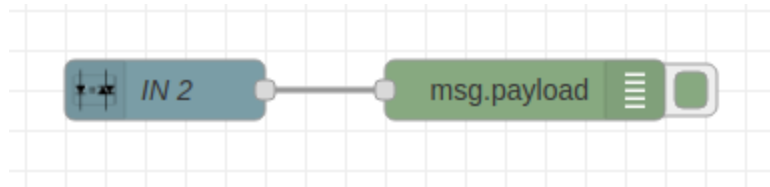


Figure 5-11: Simple Test Flow for OPTO Input 2

Your node edit window should look like this:

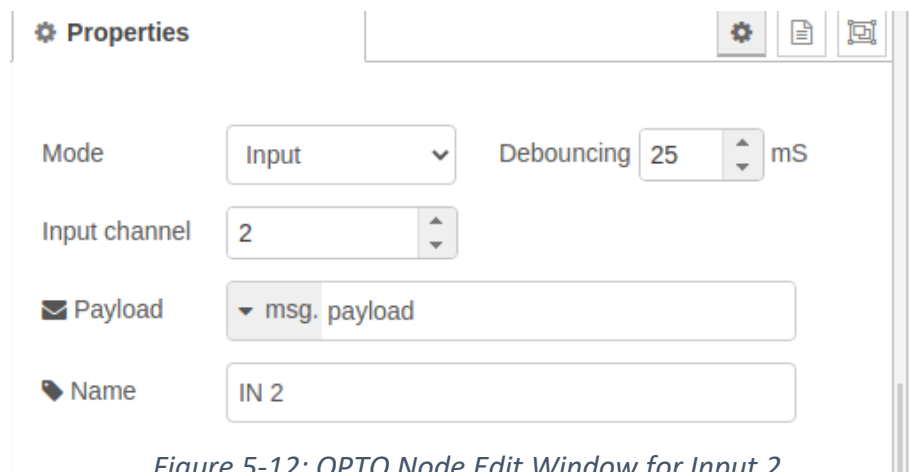


Figure 5-12: OPTO Node Edit Window for Input 2

- Click Done
- **DEPLOY IT!!!**

Push the button or flip the switch depending on what you wired to IN2. When the switch closes the circuit, you should see a debug message of *true* and when you open the circuit you should see a message of *false*. Does your Debug window show this?

## Learning Kit Workbook (version 1.3)

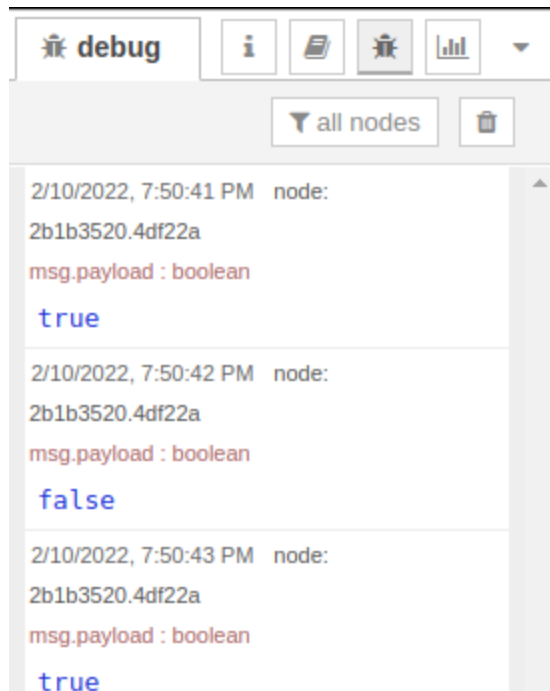


Figure 5-13: Debug Results for OPTO Node Operation

Go back up to Figure 5-12 and look for “Debouncing” selection menu. Debouncing? What the heck is that and what exactly is bouncing? Before you find out you must watch this short commercial brought to you by Real World Engineering, Inc. – “Where perfection meets reality”

---

**Engineering Tip # 10 - Nothing is Perfect!** In school or even reading on your own you have probably learned about all sorts of electrical devices: resistors, batteries, lightbulbs and maybe switches. Usually when you study these devices and make calculations you are either explicitly or implicitly assuming that they are perfect. Think about a battery. It has a voltage; you connect up to it and it does its job. Well, from your own experience in the real world you know this is not strictly true.

That little AA battery sitting on the bench is not exactly 1.5 volts and the A battery next to it does not have exactly the same voltage. The voltage may change with temperature. The voltage will change when you attach a lightbulb to the battery. As the battery is used the voltage will drop. Etc. Etc. Etc. The battery in the schematic drawing is assumed to be an ideal device, but the battery in the wild is less than ideal.

One of the most important engineering ideas is this: that real world devices are not perfect, and you must always be aware of their limitations.

---

This brings us to “debouncing”. The switch seems so simple: it is open or closed. End of story. But... nothing is perfect, especially a mechanical device like a switch. When you close the switch, it does not close immediately and perfectly. Worse yet it may close and then open for a short time and then close and then open... until finally it settles into a closed state. In the world of perfection, the switch closes or

## Learning Kit Workbook (version 1.3)

opens once each time you flip it, however, in the world that you live in the contacts actually bounce a little before they settle into their final position.

If you were to measure the voltage your switch was controlling while it went from closed to open you would very likely see something like this:

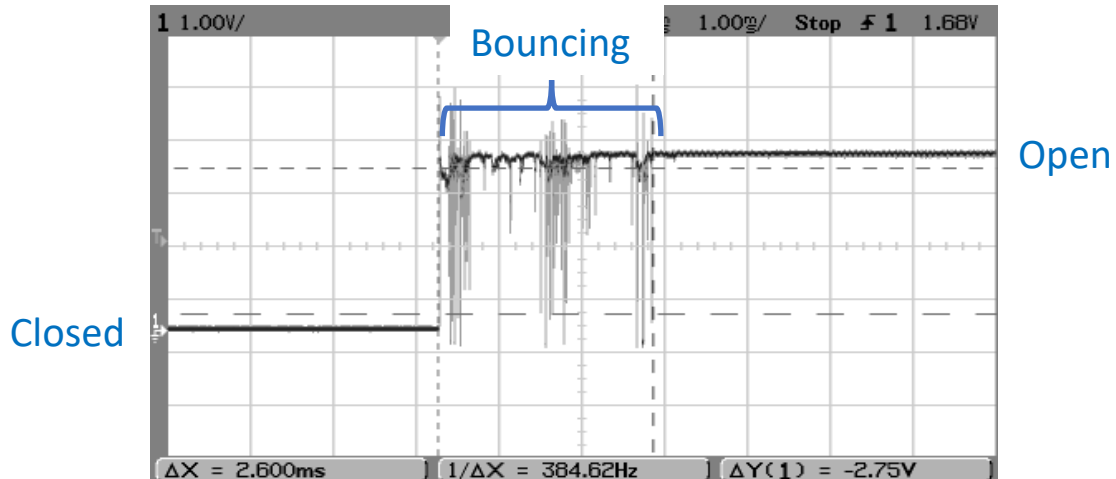


Figure 5-14: Switch Bounce (from Wikipedia)

In this diagram it takes several milliseconds for the switch to open and stay opened. If you are controlling a lightbulb in your house this shortcoming is not important. However, the Raspberry Pi is thinking in terms of microseconds, so it can see every one of these bounces. This means if you do nothing then you might get a several *true* and *false* messages every time you flip your switch. Not so good.

This is where “debouncing” saves the day. The OPTO node is designed to ignore all those little bounces. It does this in a very simple way. When the node detects the first change in state (say from closed to open) it says “Hmmm. Looks like the switch is opening. But I’ve been fooled before by all these bounces. I think it is better to wait and check the switch again in 25 milliseconds. By that time the switch will have settled down and I can send that value in the output message”. The Debouncing parameter in the OPTO node editor allows you to adjust the debounce time. The default value of 25 milliseconds is good for most applications.

**Always remember:** “No device is perfect”, and plan accordingly.

**OPTO Puzzle # 1 – I don’t believe any of this!** Well, you should be skeptical. Change the value of the debounce to 0 msec in the OPTO node. Then open and close your switch a few times. Instead of a nice transition from *true* to *false* you will likely see several even though you just opened or closed the switch once. Or if you believe the Wikipedia is the fount of all knowledge then go [here](#) and read the section on Contact Bounce.

Moving on -

The OPTO inputs are no different from the push button on the learning card in terms of what they can do. Anyplace you were using a button in Chapter 3 or 4 you can use one or more of the inputs to control

## Learning Kit Workbook (version 1.3)

the function if you set it up properly. Just remember that the LKit Button node puts out numeric 0 and 1 while the OPTO node puts out Boolean *true* and *false*. Also remember to make sure the polarity the message is what you expect when you push a button or close a switch. For the Button node pushing the button generates a 0, but for the OPTO inputs closing the circuit generates a *true*. If you want to directly substitute one of the OPTO inputs for the push button you will need to make some adjustments in your flow.

Time to experiment with the OPTO inputs. Set up a flow like this.

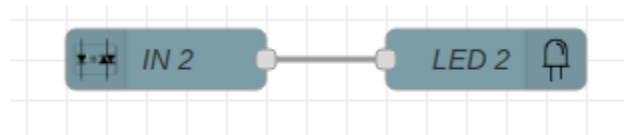


Figure 5-15: OPTO Input 2 to LED 2

You should be able to set up this flow on your own. Note that the output of the OPTO node is Boolean *true* and *false*, whereas the output of the button node is numeric 1 or 0. Fortunately, the LED node understands both messages.

To take full advantage of the four OPTO inputs in the examples below you will need to connect a switch or button to each of the four inputs. You might consider building a small box with four push buttons in it to connect to IN1 through IN4.

Here is a flow to control all four LEDs from the OPTO inputs.

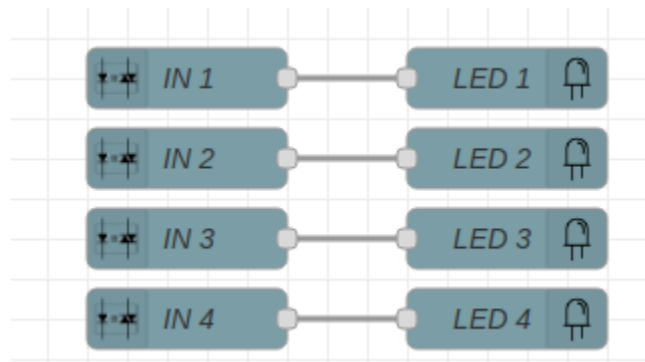


Figure 5-16: Controlling Four LEDs from Four

One little detail: the IN1 and IN3 inputs each activate an LED separate from the GP1 to GP4 LEDs. When you close the circuit on IN1 or IN3 you will see two LEDs turn on: one from your Node-RED flow and another that is hardwired to the input.

**OPTO Puzzle #2 - Sound Effects Redux.** Go back to Chapter 3 and look at Button Puzzle # 1 about sound effects. Can you implement this so that each sound effect is played when a button is pushed and continues to play until the button is released? Remember to make the necessary adjustments between

## Learning Kit Workbook (version 1.3)

the LKit Button node and the OPTO In node. Substitute a single musical node for each button (try C E G C, for example). Can you play a tune? Can you work out a way to play two notes at one time?

### Read Mode Operation

Read mode provides an explicit means for you to read the state of an input. In this mode a message giving the state of a particular input is only generated when the node receives a message. This provides a way for you to read the state of a switch on demand. A simple (but somewhat weird) example would be to read a switch in response to an email message and return the results in an email. As you will see later, this mode is also useful for reading the initial state of an OPTO node when you deploy a flow.

Begin with the flow from Figure 5-15.

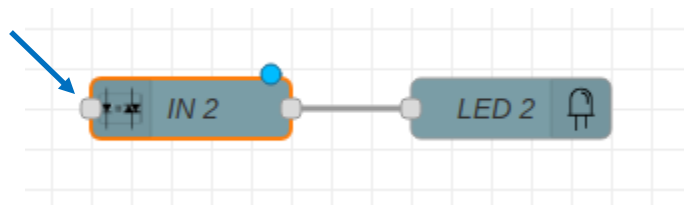
- Open the node editor for the IN 2 node and change the Mode dropdown to “Read” (blue rectangle). Now the edit window should look like this:



Figure 5-17: OPTO Node Read Mode Edit Window

- Click Done

After you click Done you might notice something interesting: the OPTO node changes to display an input port (blue arrow), like this.



In read mode the reception of a message causes the node to read the current state of an input and provide an output message where *msg.payload* reflects the current state of the input. In this example flow the input is IN2. The *msg.payload* of the message you use to trigger the node is arbitrary, in fact sending the node an empty string works just fine. Next up...

- Add an Inject node
  - labeled “Read IN2” and

## Learning Kit Workbook (version 1.3)

- that sends an empty string as a payload
- Wire the Inject node to the input of the node IN 2. The result will be the flow shown below.

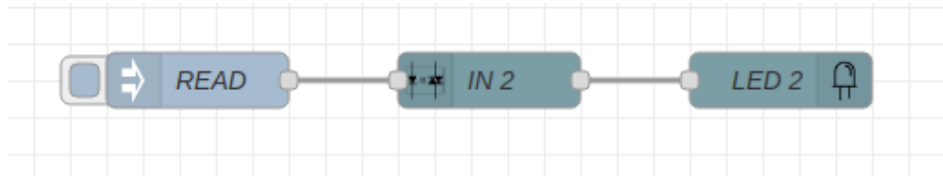


Figure 5-18: Simple Read Mode Flow

- Click Done.

### DEPLOY!!

Pause and think for a moment about what this flow is going to do. First try pushing and releasing the button (of flipping the switch) attached to IN2. Does LED 2 change? Should it, or shouldn't it?

Now press and hold the button on IN2 and click the “Red IN 2” inject node. Did LED 2 change? Now release the button. Did LED 2 change? Did you expect it to? Click the “READ” Inject node again. What happened?

If everything worked as expected the LED should only change when you click the “Read IN 2” Inject Node. This flow simply reads the state of the input whenever the Inject node is clicked.

Next up... modify the flow from Figure 5-18 so that when you click an Inject node the state of each input is read and passed to the corresponding LED. Maybe your flow will look like this. No recipe here for how to configure the nodes because you already know how.

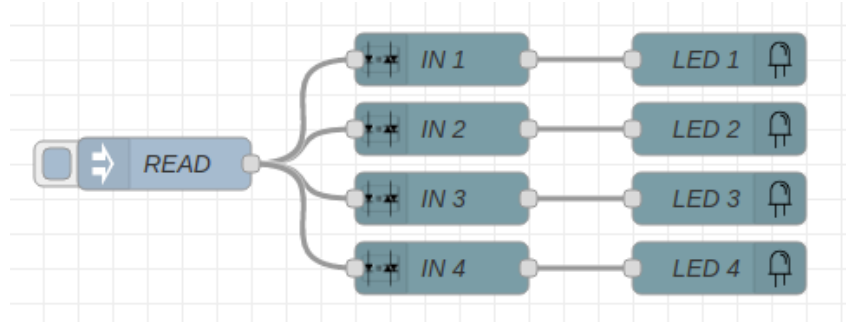


Figure 5-19: Copy All Inputs to LEDs

Ready to walk with the OPTOs?

### Walking with Switches – Initialization and Variables

Let's think about the following specification, which you will see later involves some new ideas.

- This flow will be named “Toggle LED 2 with IN2”.
- When the flow is deployed it will turn off LED 2.

## Learning Kit Workbook (version 1.3)

- When input channel 2<sup>32</sup> is closed LED 2 will change states. In other words, if LED 2 was off when IN2 is closed it will turn on. If it was on when IN2 is closed it will run off.
- When input channel 2 is not open (no connection to ground) nothing happens.

So, the action here is simple: pressing a button connected to input channel 2 will toggle the “state” of LED2. It is necessary start LED 2 in a known state, that is why it is turned off when the flow is deployed.

First, clear the deck: save all your old flow (or at least the ones you love and respect), add a brand new, sparkling clean, flow and then delete all the old flows. This way your old flows will not interfere with the operation of your new flow.

### Initialization

Now, let’s break this down a bit. The first problem is how do you make sure that LED 2 is turned off when you start the flow. Of course, you could do this manually with a simple flow, but the specification is saying that it must happen when the flow is deployed. But, how? Well, as you might expect, there is a node for that! In fact, it is one you know already, the Inject node. It’s just a matter of proper configuration.

As you flows become more complicated you will find that you will need to set up some sort of initial conditions so that your flow behaves the same way every time you deploy it. As you progress in your engineering career you will find that improper initialization is a big source of program misbehavior (and also of circuit misbehavior). Typically, if you do not initialize your flow properly it might work fine this time but fail the next. And there is this consider.

---

**Engineering Tip # 10 - Don’t Stake Your Life on Default Values!** When you create a variable (as you will do shortly) it is very tempting to just depend on their “default” values. Many programming languages will try to be helpful and set up some “nice” and logical default values for data types that you declare. Usually, something like zero or null. In some languages the default value will be “not defined” which will cause trouble down the line, but at least you know why the problem arose. Save yourself heartache, headache and stomachache by always establishing the initial value of data that you create.

---

From our specification about we know that when the flow is deployed it must clear LED 2. You can do this using the Inject node, but before working directly with the LEDs start by setting up a little test tube flow and experiment with just an Inject and Debug node. Your object is to get the Debug node to print the message “Let’s go” whenever the flow is deployed.

---

<sup>32</sup> We will use LED 2 and input channel 2 because input channel 1 also turns on an LED and this would be a bit confusing.

## Learning Kit Workbook (version 1.3)

Your undeployed test tube flow should look like this:

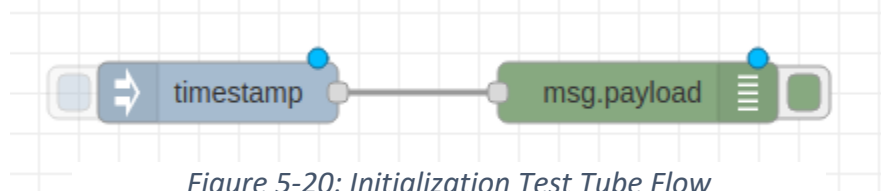


Figure 5-20: Initialization Test Tube Flow

Open up the Inject node and set it up as below (see blue arrows):

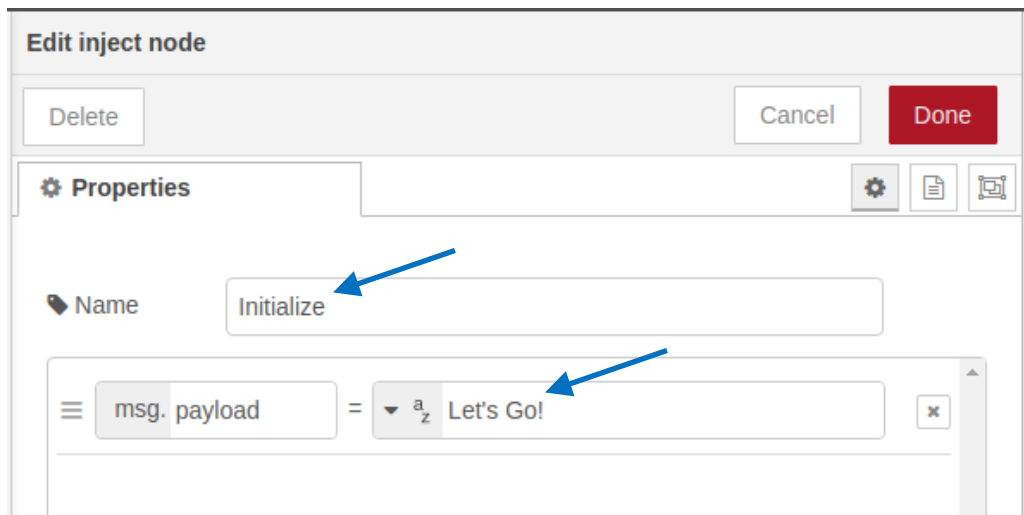


Figure 5-21: Initial Inject Node Edit

Deploy your flow. If everything is correct the Debug window will show “Let’s Go!” every time you click the Inject node. You are now ready to do something different.

Open the Inject node again and look carefully at the bottom of the edit dialog. You should see this:

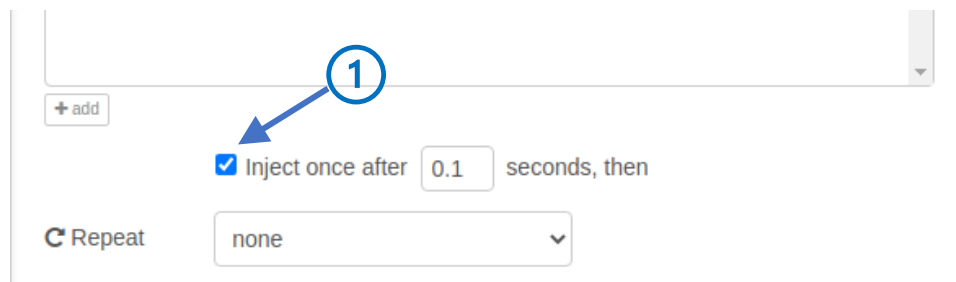


Figure 5-22: Setting Up Inject node for Initialization

- Put a check mark in the box labeled “Inject once after 0.1 seconds” ①.
- Leave the Repeat dropdown as “none” (because you don’t want this action to keep repeating).



## Learning Kit Workbook (version 1.3)

By doing this you are telling the Inject node to generate a payload with the message “Let’s Go!” one tenth of a second after deployment and then do nothing else. This is very powerful because with this set up you can generate a message shortly after your flow is deployed and that message can trigger other actions.

- Open the Debug Tab in the sidebar and clear it.

### Deploy Your Flow!

You should see “Let’s Go!” in the debug sidebar as soon as you deployed your flow.

Next, let’s see if you can clear all the LEDs on deployment. Change the message to clear all the LEDs (i.e. send a numeric zero in the payload) and replace the Debug node with an LKIT LED node. Configure your LKit LED node to group mode. Your flow should look like this now:

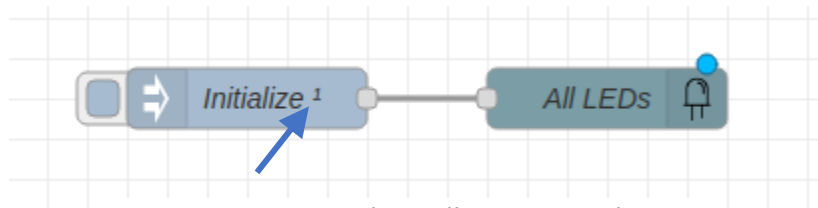


Figure 5-23: Clear All LED on Deploy

Look very closely at the Inject node. The little number 1 (blue arrow) means that the inject node will be activated once when you deploy your flow.

Think for a minute... “How am I going to test this flow?”. Probably you should add an Inject node that will set all the LEDs so you can then test your initialization. Edit your flow so that it looks like this:

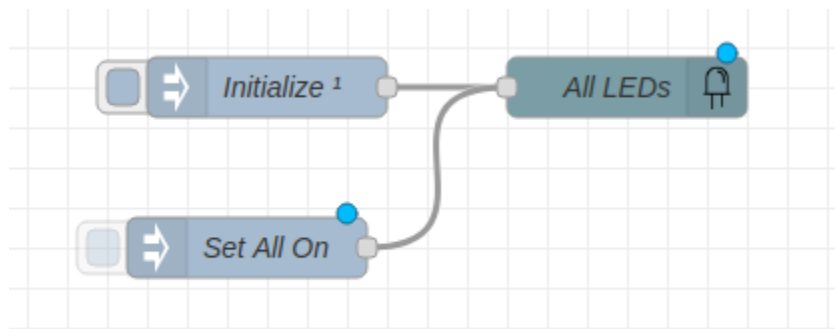


Figure 5-24: Initialization Flow Ready for Testing

You know how to do this without a recipe: open up the Inject node, name it “Set All On”, and set the payload to numeric 15.

**Deploy it Now!**

## Learning Kit Workbook (version 1.3)

You can test the basic operation by clicking on each inject node<sup>33</sup>, and if you have everything configured properly all the LEDs will turn on or off.

Time to test initialization. Click the “Set All On” Inject node, which should turn all LEDs on. When all the LEDs are on move one of the nodes a bit until the blue “Deploy Me” dot appears. Deploy the flow anew. If everything is working correctly the LEDs should go out proving that the Initialization is working properly.

Good work... you have shown that you can generate a message when a flow is deployed and use that message to set the initial state of the LEDs. Time to move on to...

### Context Data Variables

Now that you can handle the first part of the Toggle LED specification it is time to think about the second part, namely, how to toggle LED 2 from on to off and off to on when the IN2 input closes. There are several things to consider.

First, let us assume here that you are going to connect a pushbutton switch to input channel 2 and that pressing the button will close the circuit<sup>34</sup>. Remember that the OPTO node is going to generate a message when the switch is pressed (*true*) and when it is released (*false*). That means that you will need to work out a flow that will isolate the “button pressed” messages and ignore the “button released” messages by sending them off to the data trashcan. Easily done like below:

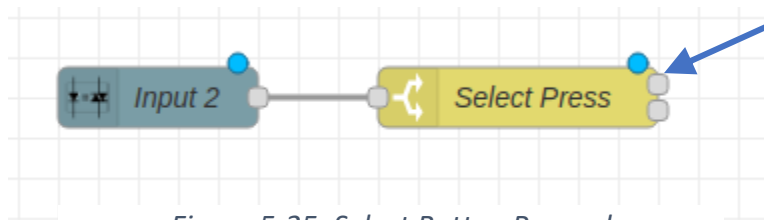


Figure 5-25: Select Button Pressed

When the button is pressed the message will appear on the top output port (blue arrow) using the Switch node configuration below.

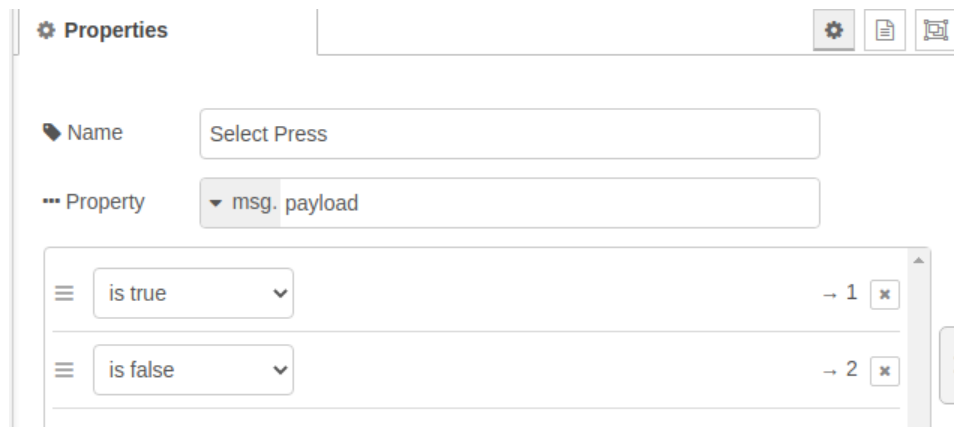


Figure 5-26: Switch Node Configuration

<sup>33</sup> Setting the Inject node to send a message at deployment does not prevent you from clicking on it also.

<sup>34</sup> You can use a switch if you wish to, but you will have to turn it on and off to simulate the push button.

## Learning Kit Workbook (version 1.3)

Second, how are you going to toggle the LED? Give this some thought. Remember, at this point you have amassed very considerable knowledge of Node-RED, how many of its common nodes work and how to construct flows. Really, take a break, get some coffee, take out a sheet of paper and see if you can think of a solution. Sleep on it, but not for too long.

Is your brain tired? It should be. If you thought about how to toggle the LED, you might have come to the conclusion that it is impossible because you don't know what state the LED is currently in. Of course, the LED knows because it is either on or off, but you have no way of saying: "Please, oh please LED 2, tell me whether you are on or off." What you need is a way to remember what you told the LED node to do last time you sent it a message. The problem is there is no obvious way to save a message so you can use it later<sup>35</sup>. This is where the concept of a **context data variable** comes into play.

"Variables" are staple of nearly every programming language, so if later you are going to plunge into a language like C, Python or Java then knowing about variables and how to apply them will be very useful.

In Node-RED you can store information outside of messages in what is referred to as "context data". For the problem you are working on now you could use context data to store the current state of LED 2 (on or off) when you set it. Then when the next button press comes you could use this stored data to decide what to do next. If the context data says the LED is currently on you can turn it off and vice-versa.

Node-RED stores context data independently of messages. Messages move from one node to the next, but context data is always held in a fixed place and is available for use by the nodes you specify.

If you already have experience with storing data in variables for other languages than you will probably want to hit fast forward now because the following discussion will seem very elementary. If not, then here are the basics.

In Node-RED context data is used to give "context" to the operation of a node, which is certainly one of the more circular definitions you have encountered lately. Imagine that context data is stored as a pile of envelopes each with a slip of paper inside. Now imagine that each envelope has a label on it, like "sum", "date-today" or, more to the point, "LED2State". The slip of paper inside each envelope is going to hold the current value of some data that you are interested in keeping track of. More specifically, imagine that the slip of paper inside the envelope with the label "LED2State" has "LED 2 is on" written on it to indicate the current state of LED 2.

Daydream some more: you are inside a node and a message arrives saying "button pressed on Input channel 2". If you could ask for the contents of the envelope labeled "LED2State" then you could decide what to do next. In this example, you ask for the slip of paper in the envelope labeled "LED2State" and it says: "LED 2 is on". With this information in hand, you know what to do next (1) erase the message on the slip of paper and (2) write "LED 2 is off", (3) put the slip of paper back in the envelope and (4) send a message to turn LED 2 off. Similarly, if the slip of paper said, "LED 2 is off" you would (1) erase the

---

<sup>35</sup> Full Disclosure – there is a way to keep a message around for later use, but it is very non-intuitive. Later you will learn about the Join node, which is a node that lets you compose new messages from several other messages. The Join node could be used to accomplish the toggle function but doing so is more complicated and not as easy to understand as using a variable. Besides in your engineering future you will be earning more \$\$\$ from knowing about variables than you will from knowing how the Join node works.

## Learning Kit Workbook (version 1.3)

message on the slip of paper and (2) write “LED 2 is on”, (3) put the slip paper back in the envelope and (4) send a message to turn LED 2 on. The key is that being able to store and access information about whether the LED is on or off gives “context” to how your node is going to operate.

Messages come and messages go, but context data is always there for you!

Time to see if the scheme above can be cast into a usable flow. Here is what the final flow is going to look like before deployment:

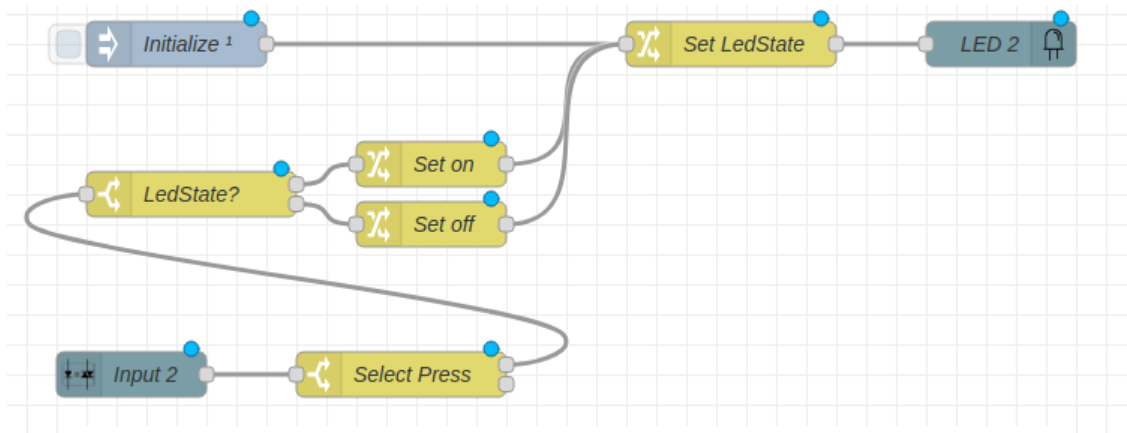


Figure 5-27: Toggle LED Flow

The top three nodes handle the initialization, setting the LED state variable and setting the LED on or off. The middle three nodes toggle of the LED state, and the bottom two nodes deal with detecting the input and selecting out only the button press action.

Let’s look at the configuration of the top three nodes first. Here is the configuration of the Initialize node, which is very similar to what you saw above in *Figure 5-21* and *Figure 5-22*

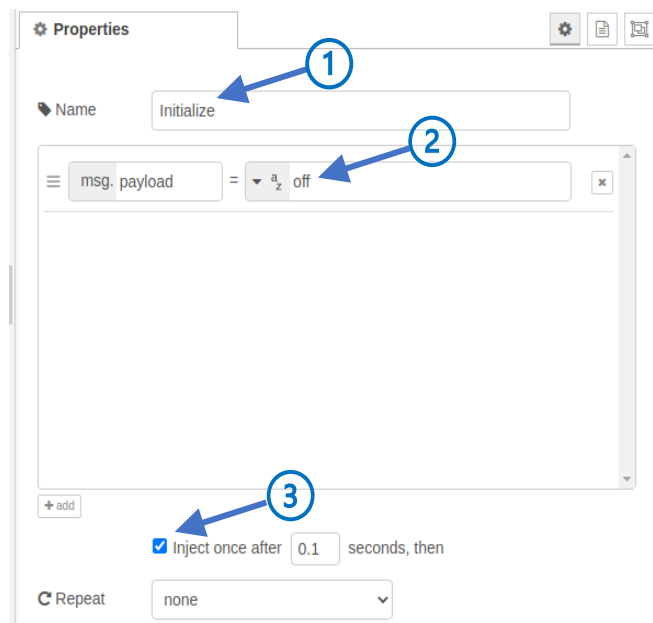


Figure 5-28: Initialize Flow Configuration

## Learning Kit Workbook (version 1.3)

- Set the name of the node to “Initialize” ① because that is what the node does, and you want others to know what you have in mind!
- Set the payload to the string “off” ② because this is the command to turn an LED off<sup>36</sup>.
- Check the box next to “Inject once...” ③ because you want to inject this message when the flow is deployed in order to initialize the LED and the LED state variable (see below).

Here is the configuration for the Set LedState node.

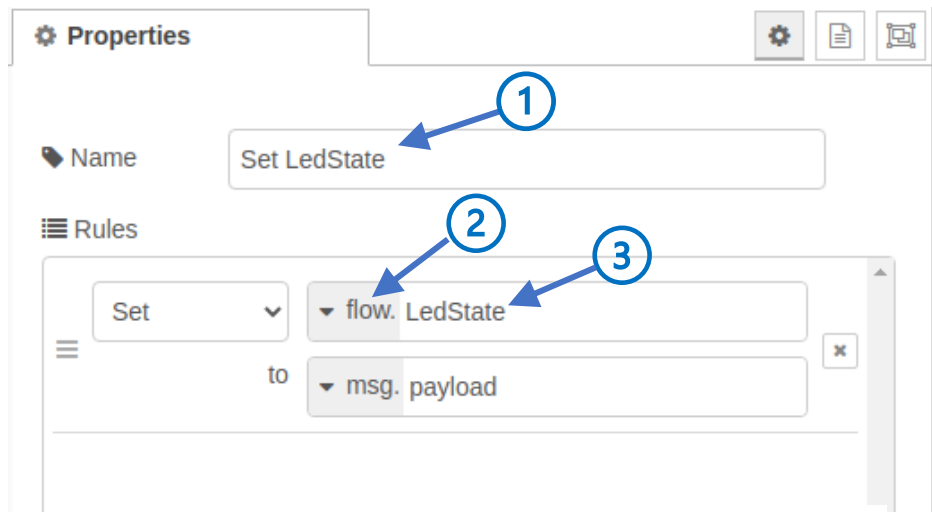


Figure 5-29: Set LedState Node Configuration

This node is a Change node, and the configuration steps are:

- Set the name of the node to “Set LedState” ① because the purpose of this node is to handle the setting of the variable called LedState.
- Using the dropdown menu select “flow.” ② . This is the place where the context data for this flow is stored.
- Give the context data the name “LedState” ③ so that you can refer to it from other nodes in this flow.

Now for some details that you *must* keep in mind:

**Names** – In the homey example above about envelopes with names on them you probably thought that you could refer to an envelope named “LedState” by any of the names that have similar spelling, like “LEDState”, “ledstate”, “LEDstate” or even “LeDsTaTe”. After all its the same letters and all the names

sound the same. **BEWARE!** The rules for names in Node-RED area very strict:

- Names can only contain letters, digits, underscores and dollar signs.
- Names must begin with a letter (so, “1stStreet” is not a good name)

---

<sup>36</sup> Remember – there are several different payload commands that can turn an LED off including the number 0 and the strings “0” and “off”.

## Learning Kit Workbook (version 1.3)

- Names are case sensitive. This is very important because it means that “LEDState” and “LedState” are two different names and refer to different context data variables. A very common programming error (and one that has been made a shameful number of times in constructing this Tutorial) is to get the capitalization of a variable name wrong. One slip of the shift key and you will have two different variables with names that look almost the same. In some type fonts it is also very easy to confuse 0 (zero) with O (oh) and I (capital i) with 1 (one) and even G with 6. Be careful.

**Scope** – This is an advanced programming concept that fortunately you will only need to deal with in a limited way in Node-RED. “Scope” refers to which nodes in your Node-RED workspace may interact with a particular context variable. There are three ways that a variable can be made “visible” to nodes.

- “node” – only one node can work with the variable. Usually this is restricted to a Function node, so you will not be concerned with this now.
- “flow” – only nodes on the same flow tab can work with the variable. This is what you will be using the most frequently.
- “global” – any node on any flow in your workspace can work with the variable. Very useful for larger projects where you have several flow tabs, but like chainsaws, global variables have seductive power that obscures how dangerous they are.

In the node configuration for the “Set LedState” shown above in Figure 5-29 the context variable “LedState” is given the scope “flow” this means that every node in the flow shown in Figure 5-27 can look at and a possibly change its value. In particular, the node “Set LedState” sets the value of the variable and the Switch node “LedState?” can look at the value to make a decision about where to send the message it receives.

Take a look dropdown menu (② in Figure 5-29). If you open the dropdown menu you will see this:

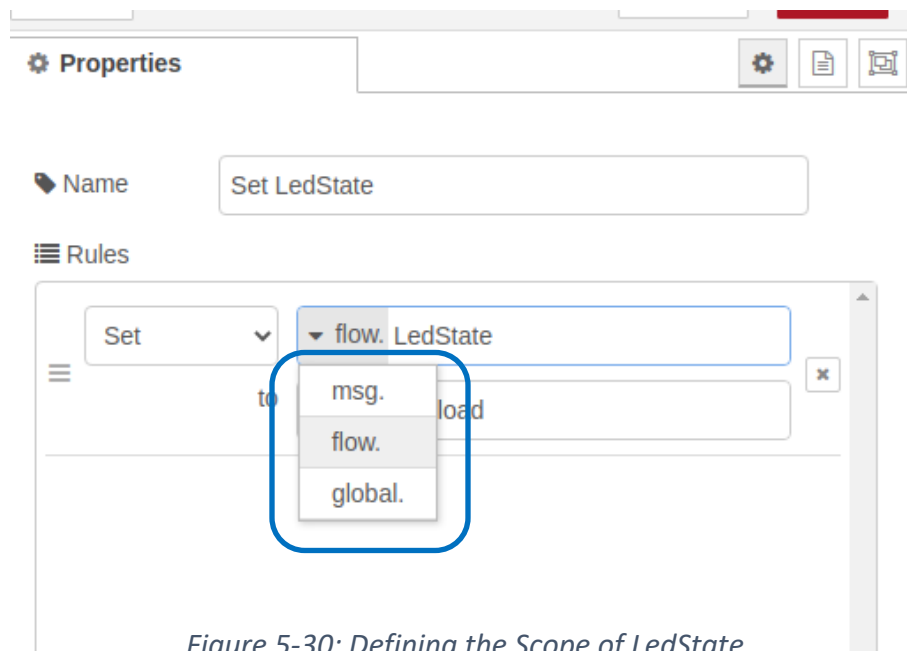


Figure 5-30: Defining the Scope of LedState

## Learning Kit Workbook (version 1.3)

Normally you would select “msg.” because you want to set the value of the payload or some other aspect of the message. However, if you want to set up a context variable, like “LedState”, then you would select either “flow.” or “global.”. Because we have only one flow in our workspace we will use “flow.”.

Return your attention to the “Set LedState” Change node for a moment. The Change node configuration (Figure 5-29 above) sets the context data for LedState to the value of *msg.payload*, but it does not change the value of the *msg.payload*. The message came into the node with *msg.payload* set to “off” and left the same way. That is what turns off LED 2 when you deploy your flow. If you are not sure about this then pull up a Debug node and look at the message going to the LED node.

And for completeness here is the configuration of the “LED 2” node:

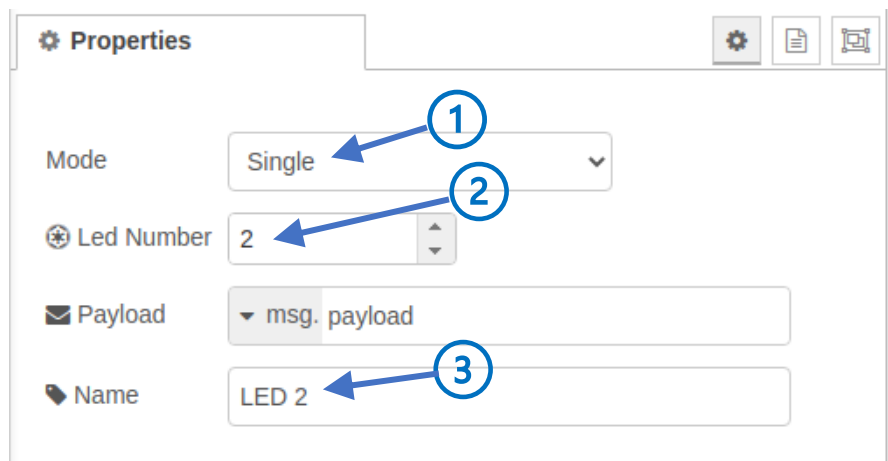


Figure 5-31: LED Node Configuration

- Set the mode to “single” ① because you only want to control LED 2.
- Select Led Number 2 ② because that’s what the specification says to control.
- Give the node a useful name ③, like “LED 2” because you should always be polite to your compatriots in the engineering profession.

Now turn your attention to the middle three nodes in the flow of Figure 5-27 reproduced below for your edification and convenience.

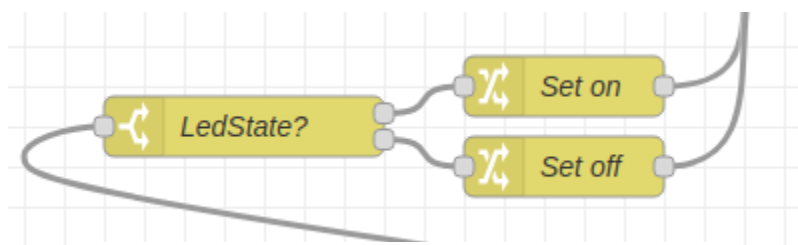
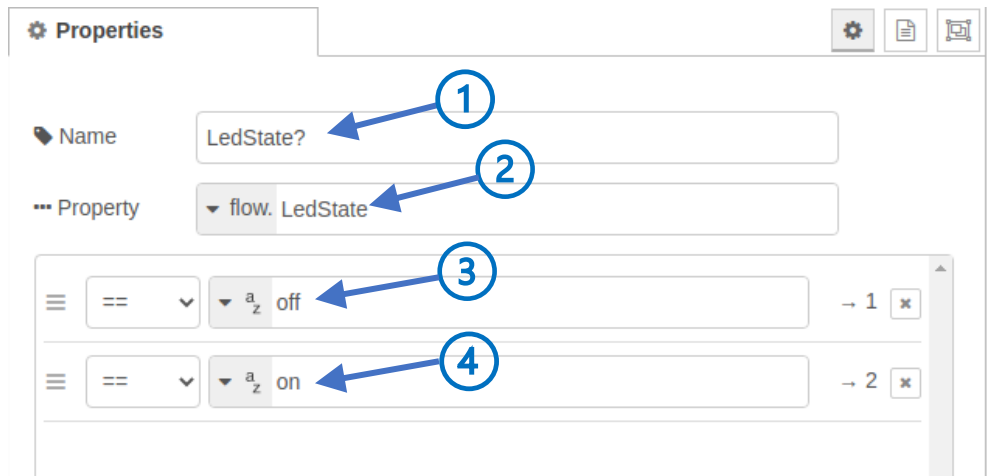


Figure 5-32: Middle Nodes of Toggle Flow

## Learning Kit Workbook (version 1.3)

Let's first look at the Switch node labeled "LedState?". The purpose of this node is to check the current value of the context data for "LedState" and based on its value ("on" or "off") to send the message to one of the two output ports. Here is the configuration:



Look at each part of this configuration<sup>37</sup>.

- Set the name of the node to "LedState?" ①. The use of the question mark indicates that the node is asking the question "what is the current value of LedState?"
- Set the property to be tested to *flow.LedState* ②, which is the context data that indicates the current state (off or on) of the LED.
- The first rule ③ will check whether *flow.LedState* is equal to the string "off". If it is then the *msg.payload* will be sent to output port 1.
- The second rule ④ check whether *flow.LedState* is equal to the string "on", in which case the *msg.payload* from the input port will be sent to output port 2.

When you look at the Switch node ("LedState?") in Figure 5-27 it can be a bit difficult to determine what rule is associated with each port, without opening up the node. Here's a Node-RED trick: simply hover the cursor over the output port and Node-RED will show you the rule, like this:

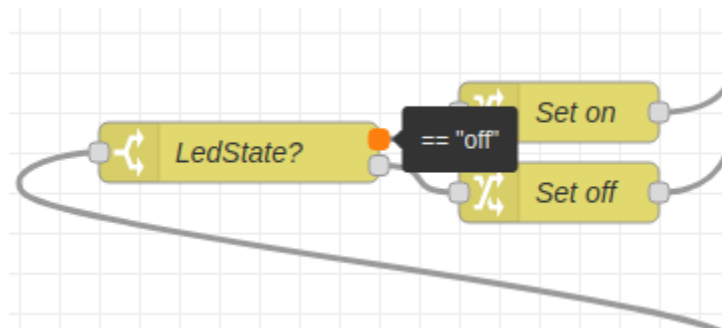


Figure 5-33: Hovering Over and Output Shows the Rule

<sup>37</sup> If you are familiar with digital logic design, you might recognize that the Switch node ("LedState?") arrangement used in the Toggle flow resembles a multiplexer, a common digital design element that sends an input signal to different outputs depending upon another signal.



## Learning Kit Workbook (version 1.3)

In this example when you hover over the top node (orange dot) a little window appears telling you that that output corresponds to the first rule where the *flow.LedState* is compared to “off” (black box). Very handy!

The configuration for the “Set On” and “Set Off” nodes are Change nodes programmed as you would expect: to simply set the *msg.payload* to the strings “on” or “off” as determined by the output of the “LedState?” node.

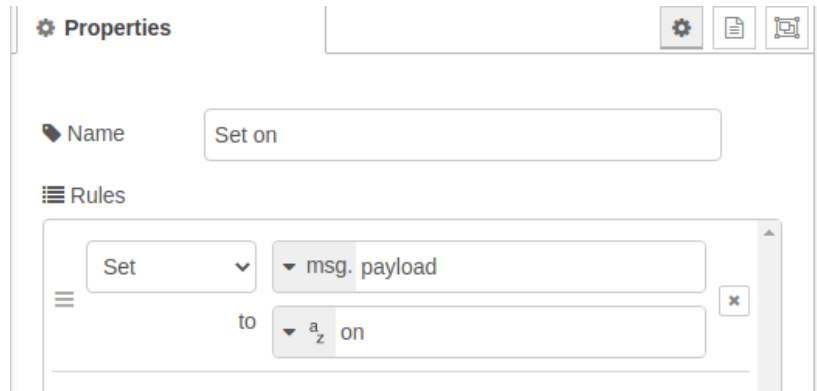


Figure 5-34: Set On Node Configuration

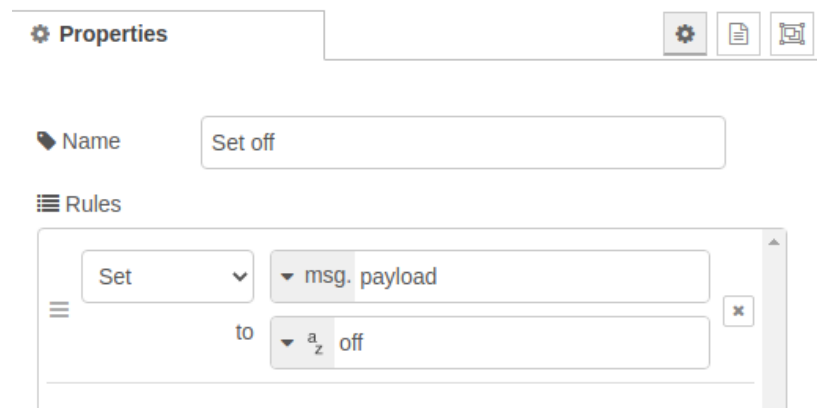


Figure 5-35: Set Off Node Configuration

Set up the flow as shown in Figure 5-27 above, DEPLOY IT and see if it behaves according to the specification. When you press the button connected to IN2 the state of LED 2 should alternate between off and on.

Study this flow carefully because it is about as simple a flow as you will find that uses context data and this is an important and powerful Node-RED concept. If you are confused about how the flow works pull in a few Debug nodes and look at the messages flowing between nodes.

To make the idea of context data more concrete look at this annotated diagram of the Toggle flow. On it you will find a little block labeled *flow.LedState*, which is the context data. This is shown outside of the nodes because it is not stored in any particular node. Rather nodes, like the “LedState?” may reference

## Learning Kit Workbook (version 1.3)

the `flow.LedState`. It like a little public library in your flow where nodes can check out a copy of the data and, if necessary, modify the data and send it back to the library<sup>38</sup>.

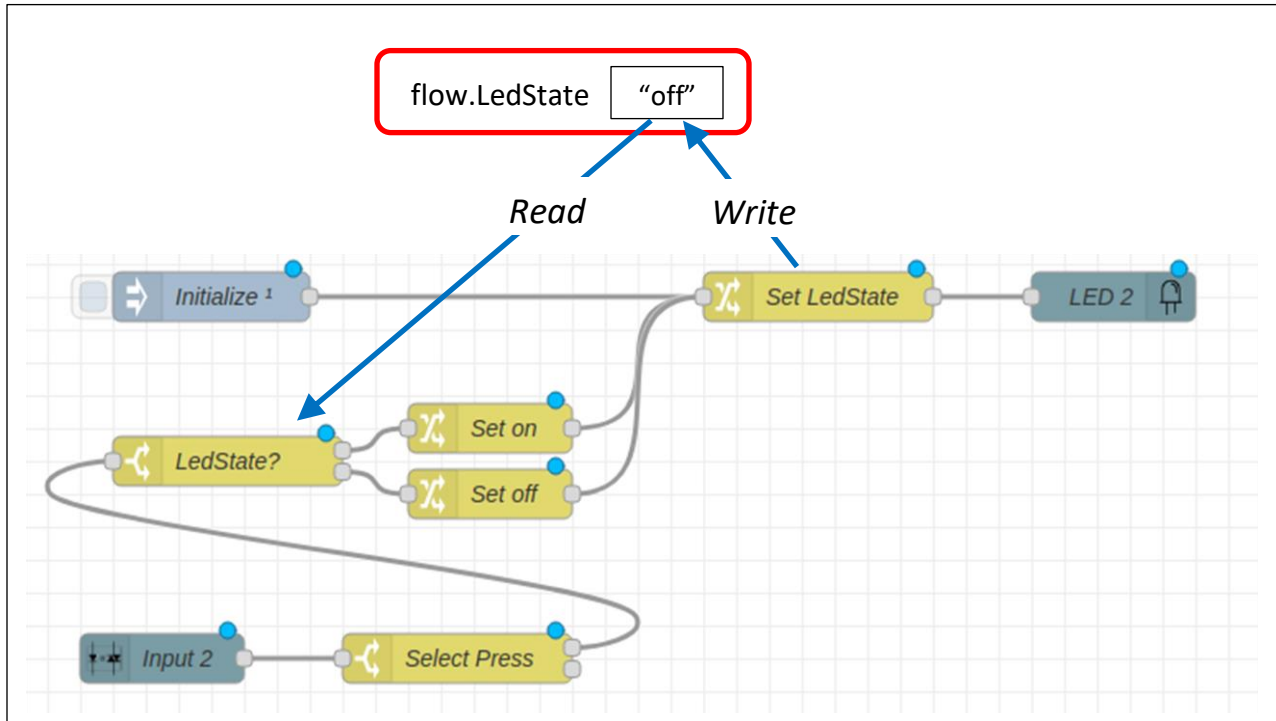


Figure 5-36: Context Data for Toggle Flow

In the figure above the context data “LedState” is accessed by two nodes, although any other node in this flow could make use of the data if necessary. The Change node, “Set LedState”, writes the value of the context data (in this example the string “off”) while the Switch node, “LedState?”, reads the context data. Later you will learn about situations where a node can read the context data, modify it and write it back.

### Working with Context Data

Node-RED provides you with the ability to examine the current state of context data. This is a powerful debugging tool you should know. You can do this by opening the context-data tab in the sidebar. At the

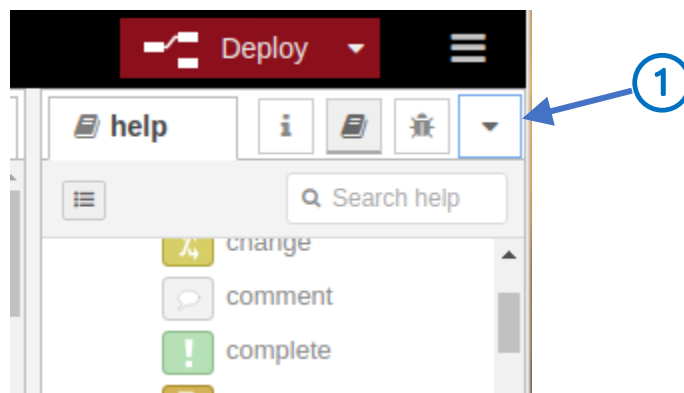


Figure 5-37: Sidebar Tabs for Narrow Window

<sup>38</sup> Please, please, please do not modify books you check out from a real library!!

## Learning Kit Workbook (version 1.3)

top of the sidebar there are four or five tabs that you can use to open different sidebar views. However, if your sidebar area is narrow only two or three of these tabs will show up. This means that the tab for the context-data might be covered up. Here is a typical view of the side bar area when the width is small.

As you can see only three tabs are visible (Information, Help and Debug) in this view. However, if you click the dropdown ① a menu will open showing you the other sidebar tabs, including the context data as shown below.

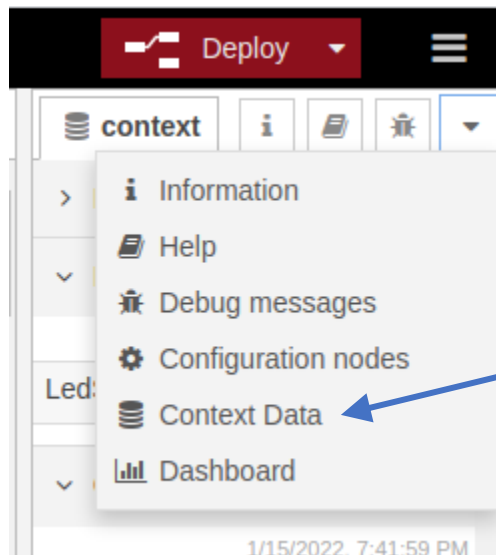


Figure 5-38: Sidebar Dropdown Menu

Click on the “Context Data” entry (blue arrow) and the context data window will open. If by fortunate chance your sidebar window is wide you will see all the context data tab which has an icon that looks like a disk drive sized for use by hamsters:

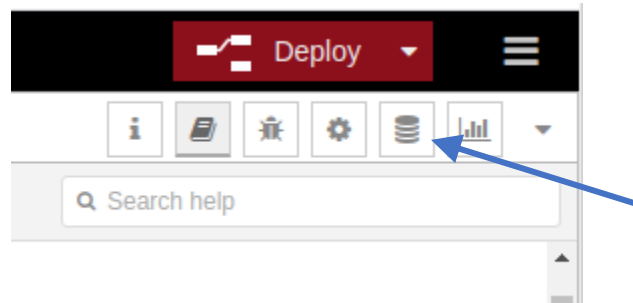


Figure 5-39: Wide Sidebar with All Tabs

No matter how you get to it, once you open the context data tab you will see something like this:

## Learning Kit Workbook (version 1.3)

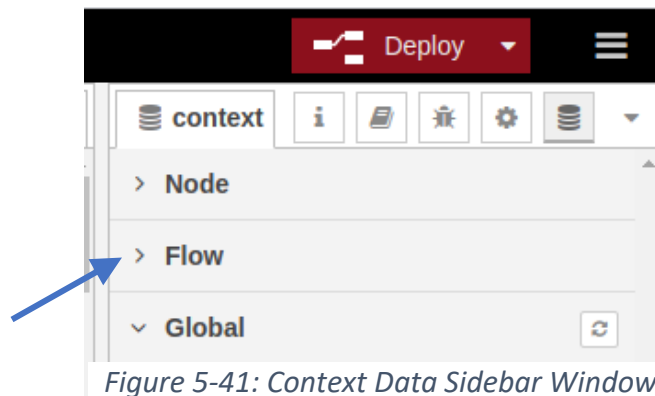


Figure 5-41: Context Data Sidebar Window

This window is showing your dropdowns for Node, Flow and Global context data. Click on the arrow next to “Flow” (blue arrow) and you will be able to look at all the context variables on the current flow. In the case of the Toggle flow, there is only one context variable, namely, *LedState*:

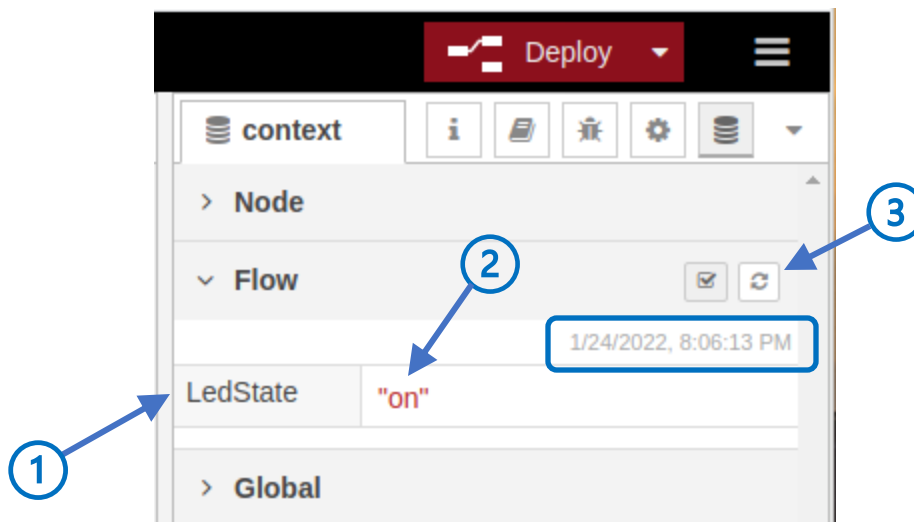


Figure 5-40: Context Data for Toggle Flow

Next to the name of the context data variable ① you will find the value ②. **Pay close attention...** the value you see next to the name of the context data variable may **NOT** be the current value of that variable. Rather it more like a snapshot, showing the value of the variable at the last point in time that you refreshed the context data, not the last time that Node-RED changed the value. Look above the variable and you will see a time stamp (blue rectangle). This is the last time you updated the variable. If you want to take a snapshot of the current value of the context data for your flow you must click the update button ③. When you do the value will be updated along with the timestamp. Keep this in mind otherwise you will be very confused when you debug the context variables in your flow.

### Controlling Loops – Starting and Stopping

Previously you learned about setting up loops (See Chapter 4), however, all these loops ran forever once you started them. Now is the time to tame them. The situation is going to be simple. You will use a

## Learning Kit Workbook (version 1.3)

switch connected to input IN 2 to control a loop that flashes LED 2. You already know how to set up a simple flashing light loop, and you can see the results easily.

Start with a specification:

- When IN 2 is closed LED 2 flashes.
- When IN 2 is open LED 2 stops flashing.
- When the flow is deployed LED 2 will behave according to the current state of IN 2.
- When flashing the LED is on for one half second and off for one half second.

As you start to think about the specification above you might notice aspects that are not mentioned explicitly. For example, if IN 2 is closed and the light is flashing, and you open IN 2 the flashing will stop, but is LED 2 left on or off when the flashing stops? This is not necessarily an oversight in the specification, but rather is something that is left open for you to decide. Sometimes these “loose ends” on a specification are referred to as “degrees of freedom”. A really well written specification will call out the degrees of freedom so that you may take advantage of them to simplify your design.

Another unmentioned aspect of the specification is what will be connected to IN 2. Since it is not specified you may connect any sort of switch or contact closure you desire as long as it can open and close IN 2. In particular, you may use a button, or a switch. Since the specification says nothing about IN 1 you can even use an SPDT switch connected between IN 2, IN 1 and the central ground pin.

Per the usual, you should begin by breaking the problem down into smaller pieces so that you can build and test the pieces and then integrate them a bit at a time into the final solution. Engineering is about imagining solutions, planning and careful execution. There are usually many different answers to a particular problem. Your task is to find one of them and, if possible, identify one that is the best (or at least good) by some measure, like cost, weight, speed, flexibility, fuel economy, beauty etc. etc. etc.

Here is one suggestion on how to proceed, but you might have other ideas that are as good or, hopefully better.

- Build up a flow that will flash LED 2 (you have done this before, so part of the solution is already at hand (see Chapter 4).
- Work out a way to start and stop the flashing (a good idea might be to start by using Inject nodes in place of switches because they are easier to deal with and control).
- Extend your solution to work with the IN 2 switch (or pushbutton)
- Finally, work out a way to detect the IN 2 switch position when the flow is initialized.

You have already built up a flashing LED loop. Use that as a basis for your design. Here is one possibility:

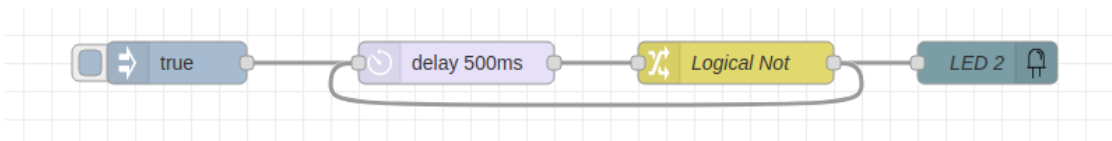


Figure 5-42: Flashing LED Loop

This should look very similar to the flow in Chapter 4. However, there are a few differences. The flashing time is one half second on (500 msec) and one half second off. Also, each time around the loop you must change the command you send to LED 2. In the previous example you were sending LED 2 a

## Learning Kit Workbook (version 1.3)

numeric 1 or a numeric 0. However, for this flow we will use Boolean *true* and *false* to control LED 2 because that is what the IN 2 input channel provides, and it will simplify things. So instead of using the Invert Node you built up before you will need to develop a new node, the “Logical Not<sup>39</sup>” node that will convert *true* to *false* and *false* to *true*. You can use a change node to do this, and you can use a similar technique to perform the exchange of *true* and *false*.

Now it is time for a coffee break<sup>40</sup>. Come back in 15 minutes think about the flow and how you might configure the nodes. Give it a try and see if the LED flashes as you expect. Once you hit the Inject node the LED is going to flash forever, which is OK because the next step in the plan is break the loop.

If your flashing light flow works then terrific and move on. If not, you can start by debugging it with judicious placement of Debug nodes. If that doesn't work, try taking the “Logical Not” node out by itself and hook up an Inject node and a Debug node and see if it is playing nicely and doing what you want. This is the most complicated node and the most likely node to have a configuration mistake. If all else fails, look at these configuration windows further down for each node.

Speaking of Debug nodes, here is another subtle feature of Node-RED: you can disable/enable Debug nodes on an individual basis. If you have done some simple programming in, say, Python you will know that one quite powerful technique is to put print statements in your code to track what is happening. Then when you verify some part of the code is working you “comment out” that print statement. You can do the same thing in Node-RED with Debug nodes.

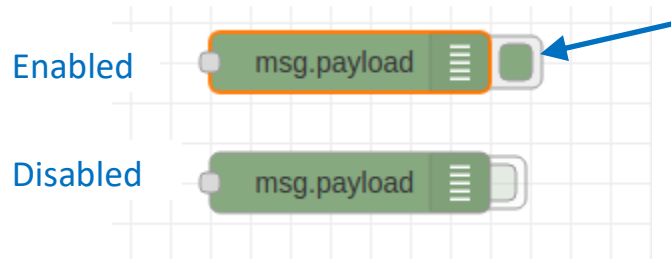


Figure 5-43: Debug Nodes (Enabled and Disabled)

In the figure above are two Debug nodes. When you drop a debug node into your flow it is Enabled and will send messages it receives to the Debug sidebar window. However, if you click the little green tab on the right side (blue arrow) it will change to the form shown below where the tab is grayed out and tucked under the node. In this mode the Debug node is deactivated and will not pass messages to the Debug sidebar window. Click the tab on the disabled Debug node and it will be enabled again. This means you can scatter as many Debug nodes as you want in your flow and enable or disable them as required to focus in on a problem. When you are satisfied that your flow is working you can delete them all.

<sup>39</sup> Logical Not – in Boolean logic (or Boolean Algebra) a function that converts true to false and false to true is called a “Not” or ‘complement’ function. Brush up on Boolean Logic [here](#).

<sup>40</sup> The mind is a strange thing. You can think and think and think about a problem. Then you take a break and BOOM the solution appears in a flash. There even seems to be some science behind this phenomenon: active thinking blocks subconscious thought. It really is true that many good ideas come to folks in the shower, because they are completely distracted from the problem. Ah... coffee breaks.

## Learning Kit Workbook (version 1.3)

Onward...

Here are the configurations of the nodes in Figure 5-42. The first, is an Inject node that will push a message with the `msg.payload` of boolean `true` into the flow to kick things off when you click it.

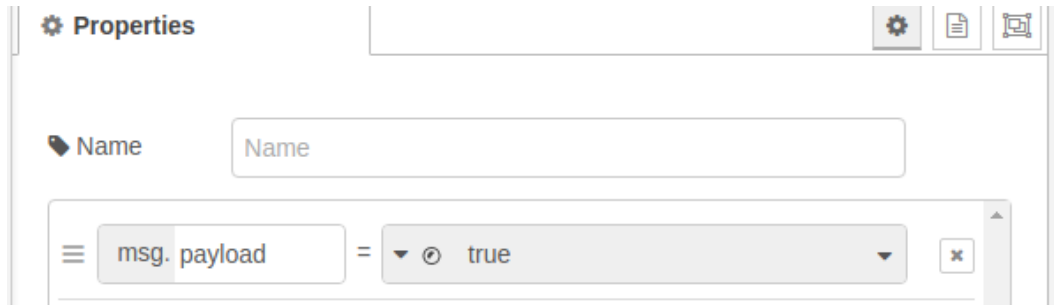


Figure 5-44: Inject Node Configuration

Next is the Delay Node, which delays the message for 500 milliseconds.

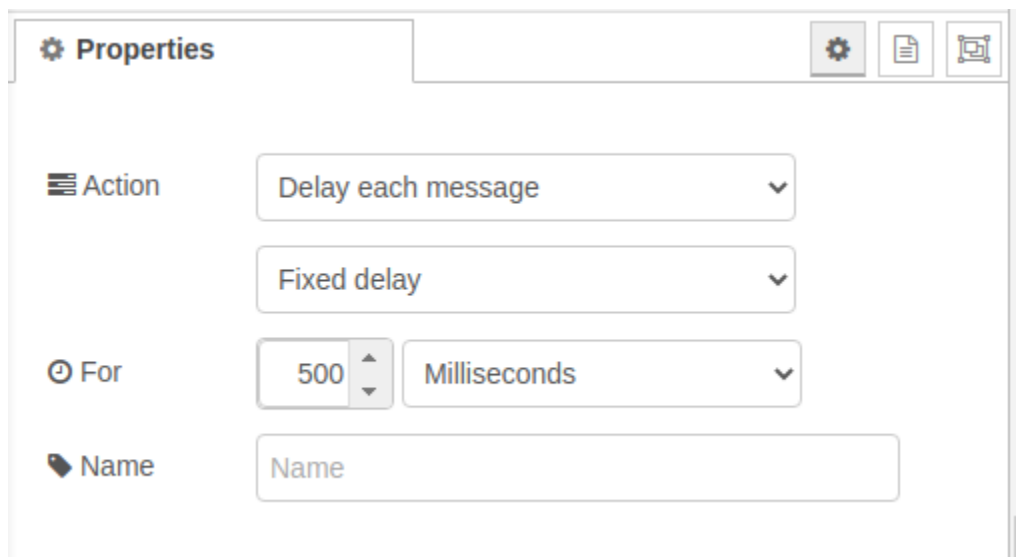


Figure 5-45: Delay Node Configuration

Now for the difficult node, the “Logical Not” node, which is a Change node with three rules:

## Learning Kit Workbook (version 1.3)

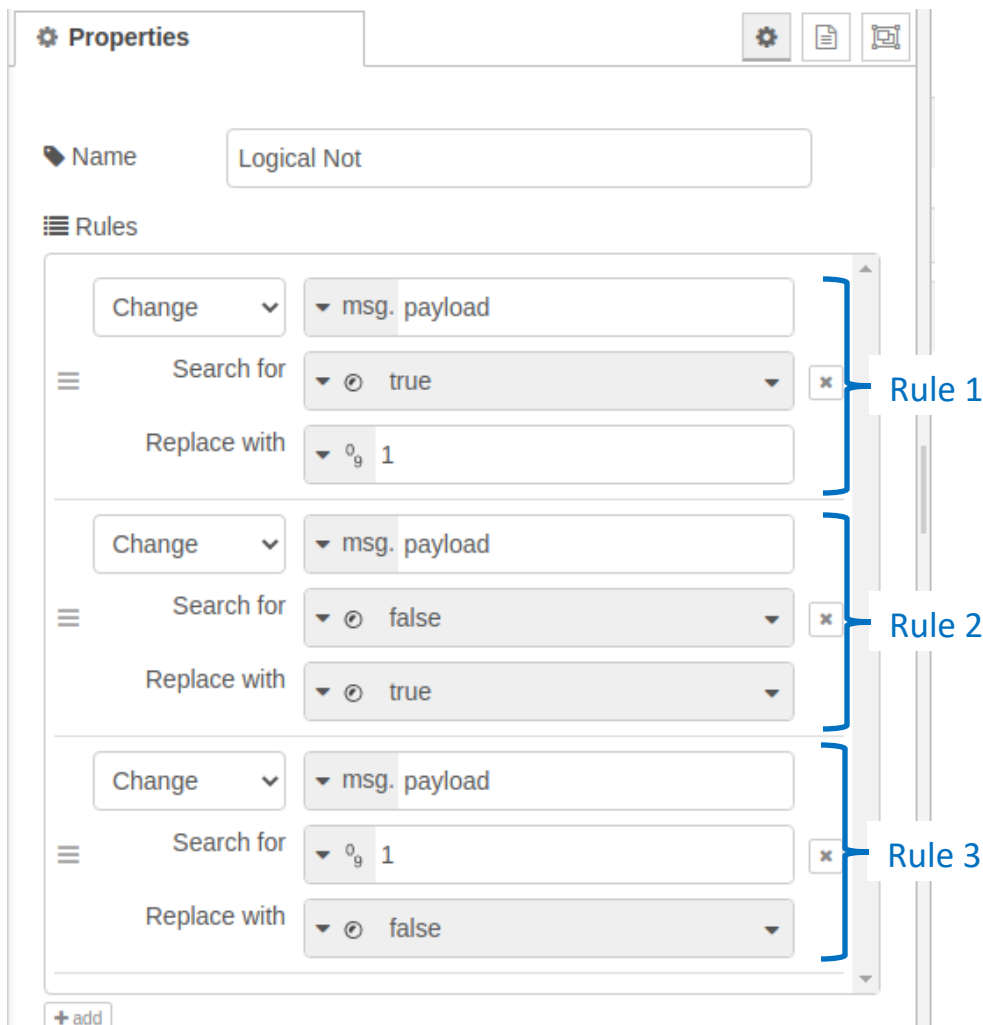


Figure 5-46: Logical Not Node Configuration

Even though all you want to do is to change Boolean *true* to *false* and *false* to *true* you cannot do this with two simple rules for the reason discussed in Chapter 4. Importantly, every rule specified in the Change node is applied to the payload starting with **Rule 1** and ending with **Rule 3**. If you simply changed Boolean *true* to *false* in **Rule 1** then **Rule 2** would convert it back to Boolean *true*. To avoid this problem Rule 1 converts Boolean *true* to a different, temporary value, namely, numeric 1<sup>41</sup>. Next is **Rule 2**, which converts a Boolean *false* in the payload to Boolean *true*. Finally, if the input payload had been Boolean *true* **Rule 3** will convert the temporary value of numeric 1 to Boolean *false*. If you are still confused about how this Change node works pull the node out by itself, connect an Inject node to the input and a Debug node to the output and try different payload values, like *true*, *false*, numeric 1, numeric 15, and maybe a string or two. Admittedly this three-rule approach is murky, but just remember every rule is applied to the *msg.payload* value in sequence and if rule changes the *msg.payload* to a value that will trigger a following rule then that following rule will apply also.

<sup>41</sup> It is not important what temporary value you choose as long as it is different from boolean *true* so that Rule 2 does not apply to the new *msg.payload* value.



## Learning Kit Workbook (version 1.3)

The LED node configuration for LED 2 is simple:

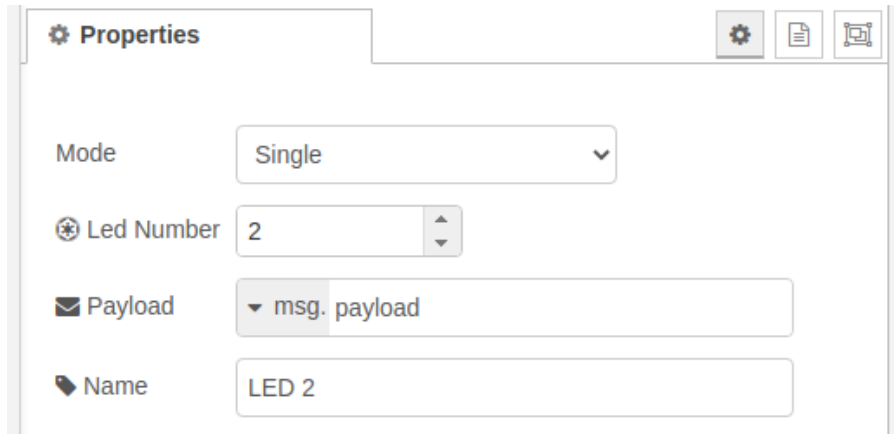


Figure 5-47: LED Node Configuration

The mode is “single” because you want to control only LED 2.

### **DEPLOY YOUR FLOW!**

When you click the Inject node you are injecting a single message with the payload Boolean *true*. After a delay of a half second (500 milliseconds) the payload value is changed from Boolean *true* to *false*. This goes to the LED 2 node and turns the LED off (Remember: payload values of *false*, 0, “0” and “off” will all turn the LED off). The new payload value of *false* loops back to the Delay node where it passes a pleasant half second resting up before moving on to the Logical Not node. Here is converted from its current value of *false* to *true* and passed to the LED which then turns on. The message also loops back to the Delay node for another snooze and so on forever.

Now for the big question: how do you break the loop to stop the LED from flashing? Think about this for a few moments... what sort of node might be used to shunt the message out of the loop so that it does not circulate any longer. Thinking... thinking... thinking... Ah Ha! How about the Switch node? A message goes in and depending on the conditions you have defined the message can go out one of two different output ports. How about an arrangement like this?

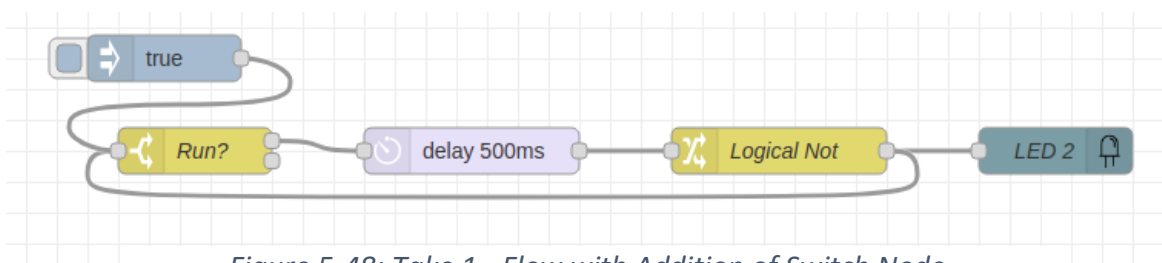


Figure 5-48: Take 1 - Flow with Addition of Switch Node

The “Run?” node is a Switch node with two outputs. Assume someplace there is a context data variable named “Run” which can take on a value of *true* or *false*. If the Switch node can consult this context data variable, then it would be able to decide whether the input message should go to the upper output port and continue circulating or go to the lower output port and be lost forever in the great bit bucket in the

## Learning Kit Workbook (version 1.3)

sky. Under one condition the message circulates and the LED flashes and under the other circumstance the message stops circulating, and the LED will stop flashing. With all of this in mind your “Run?” Switch node configuration should look like this:

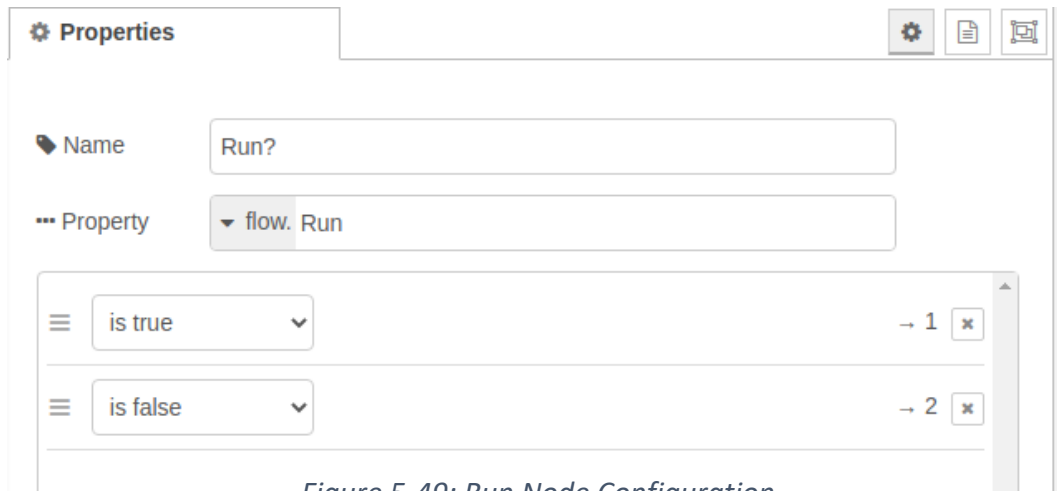


Figure 5-49: Run Node Configuration

Simply put, when an input message arrives the “Run?” node checks the value of the context data variable Run and directs the input message to output 1 if the value of Run is Boolean *true* and to output 2 if it is Boolean *false*.

If you deploy the flow as it stands now you will find that it does not work because the context data variable, Run, does not yet exist. So, your next step is to set up some way to define and control the context data variable. You can see if the Run context data has been defined by opening the context data tab and then opening the “Flow” dropdown. Don’t forget to refresh the “Flow” dropdown

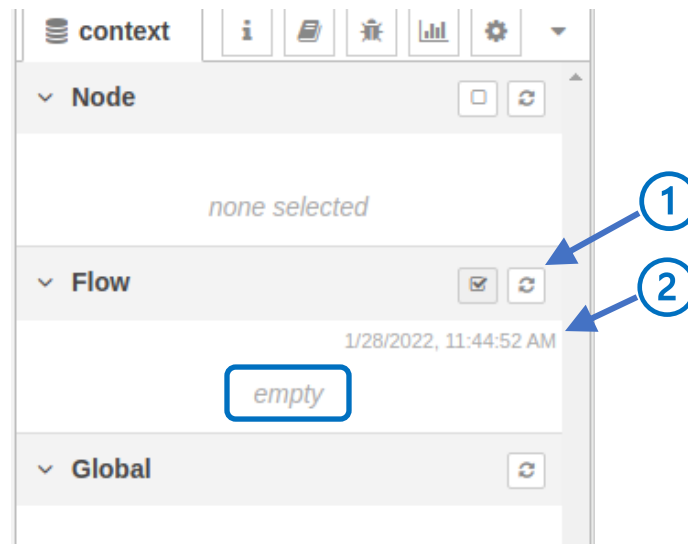


Figure 5-50: Context Data Example

Above is an example of context data for Flow related variables. It is empty (blue box) indicating that there is currently no defined context data for your flow. Remember, it is important to refresh the

## Learning Kit Workbook (version 1.3)

context data window ① before you rely on it. In the Flow dropdown there is a timestamp ② that tells you the last time your context data was updated.

Next step – enhance your flow to control the context data variable Run as in the figure below, which you will want to include in your flow from Figure 5-42. This is important because the context data for Run is only available within the flow where it is “defined”. Add the nodes below to your flow.

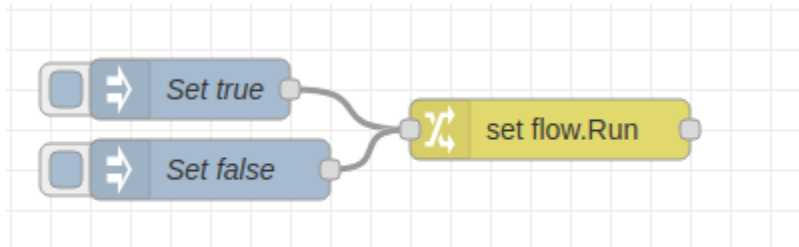


Figure 5-51: Defining and Controlling Context Data

- Configure one Inject node to generate a *msg.payload* with a value of Boolean *true*.
- Give that node the name “Set true” so you will know what does.
- Configure the other Inject to generate a *msg.payload* with a value of Boolean *false*.
- Give that node the name “Set false”.

The Change node is going to set the Context Data variable “Run” to whatever the *msg.payload* value is. Here is what the configuration should look like:

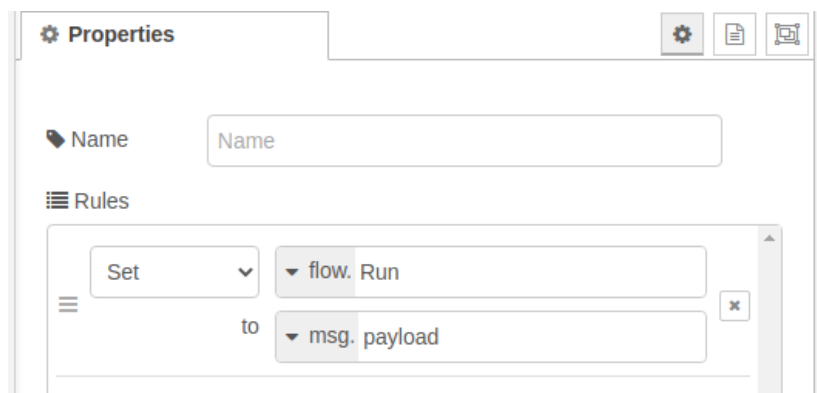


Figure 5-52: Configuration of Change Node to Set Run Variable

### Deeeeeeeply IT!

Now open the Context Data tab in the side bar and open the Flow dropdown (see Figure 5-50). Click the refresh icon ① in that figure. Notice that the Flow dropdown is still empty even though you have mentioned the variable “Run” in your “set Flow.Run” node ( Figure 5-52, above). You might be asking yourself, “What’s going on here?” The answer is that in Node-RED a context data variable does not spring into existence merely because you mention it in some node or other in your flow. Rather it will only become real the first time you assign a value to it.

To test out this little flow do this:

## Learning Kit Workbook (version 1.3)

- Click the “Set true” Inject node in Figure 5-51.
- Now go back to the Context Data Tab, open the Flow dropdown (if it is not already open) and click the refresh icon. Presto! The Run context data is there, and it is set to *true*.
- Click the “Set false” Inject node. The value in the Context Data window will not change, until you:
- Click the refresh icon again.

Try this a few times and make sure you understand what is going on.

Ready for the big test?

- Click the “Set true” Inject node, refresh the Context Data window and make sure that the value in the Run context data variable is *true*.
- Click the “true” Inject node on your previous flow.

Does LED 2 flash per the specification? If so great. If not, then time to debug. You know the drill already: wire up some Debug nodes and make sure that the messages are what you think they should be.

- Click the “Set False” Inject node, refresh the Context Data window and make sure that the value in the Run context data variable is *true*.

Did LED 2 stop flashing? If so: perfect, if not: debug time.

What do you suppose is the message that appears at the output port of “set flow.Run” node in Figure 5-51? Connect a debug node to the output port and find out. Like this:

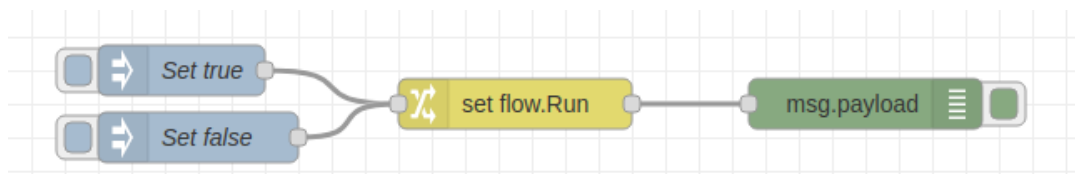


Figure 5-53: Message Output Test Flow

Deploy this test flow!

- Click the “Set true” Inject node. What is the message that appears in the Debug sidebar?
- Click the “Set false” Inject node. What message is output?

Interesting isn't? The “set flow.Run” uses the payload to set the Context Data variable Run and simply passes the payload to the output. It's almost as if the node is not there? Maybe you can use this to your advantage.

How about this:

- Delete the Debug node.
- Delete the “true” Inject node in your flashing LED flow.
- Connect the output of the “set flow.Run” node to the input of the “Run?” Switch node.

## Learning Kit Workbook (version 1.3)

Your new flow should look like this:

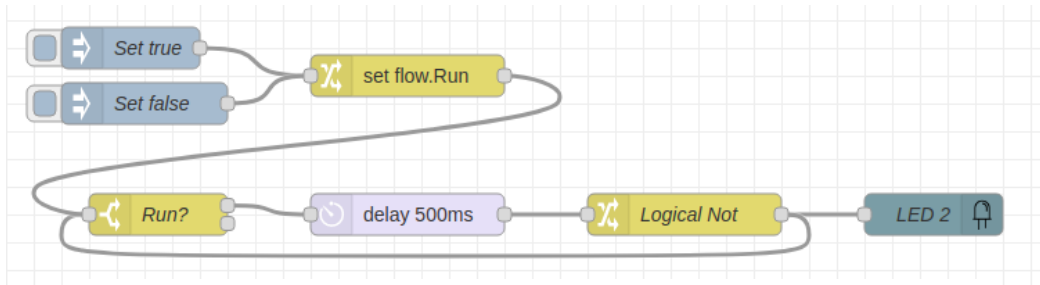


Figure 5-54: Take 2 - Flashing LED Flow with Start and Stop

### DEPLOY THE FLOW!

Now try clicking on the “Set true” and “Set false” Inject nodes. How does the flow behave? Is it close to the specification above? What part of the specification does the flow meet? Where is the flow deficient?

One aspect that is missing is using a switch connected to IN 2 to control the flow. Look back at the help file for the opto-isolated inputs. Closing the switch produces a message with a payload of *true* and opening the switch produces a payload of *false*. Fabulous! You can just replace the two inject nodes with the OPTO node like this:

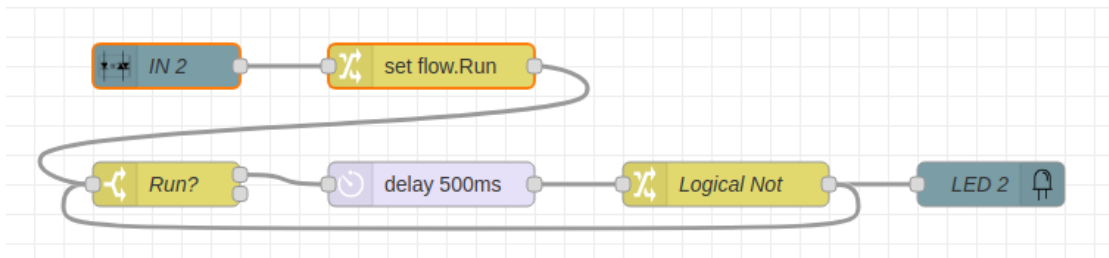


Figure 5-55: Take 3 - Flashing LED Flow with Switch Control

- Configure the OPTO node with name “IN 2” because that is the input you are using.
- Select “input” mode because you want a message each time the switch changes position.
- Select “input channel” 2.

When you are done the OPTO node configuration should look like this:

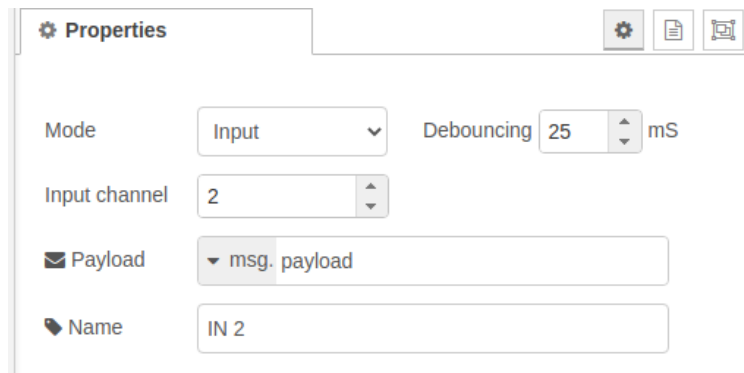


Figure 5-56: OPTO Node Configuration

DEPLOY, for goodness sakes!

## Learning Kit Workbook (version 1.3)

To test this flow, you must have a switch (or a pushbutton) connected to IN 2. If you did everything correctly when the switch is in one position LED 2 should flash and in the other position LED should stop flashing<sup>42</sup>.

- **Everything works:** take a walk, enjoy the sunshine.
- **Flow does not work:** time to put your debugging skills to the test. Carefully check the configuration of each node. Try using Debug nodes to see where the messages are going astray. You can even try injecting a message like Boolean *true* or *false* into the flow to see what happens. Just remember to Deploy the flow every time you make a change. Otherwise, you will never walk in the sunshine again.

Go back and check the specification. Test each aspect of the specification. Is anything missing?

Ah, yes! When the flow is deployed LED should flash or not flash depending upon the position of the switch. You may have noticed this deficiency because when you deployed the flow of Figure 5-55 the flow did not start flashing even though the switch was in the “flashing” position.

Think for a moment, appeal to the wisdom of the hamsters. How are you going to get the flow to start properly upon deployment? Look back at how you did initialization in previous flows (e.g., Figure 5-27). All you need to do is read the state of the switch on IN 2 when the flow is deployed and use that to start the flow. Would a flow like this work?

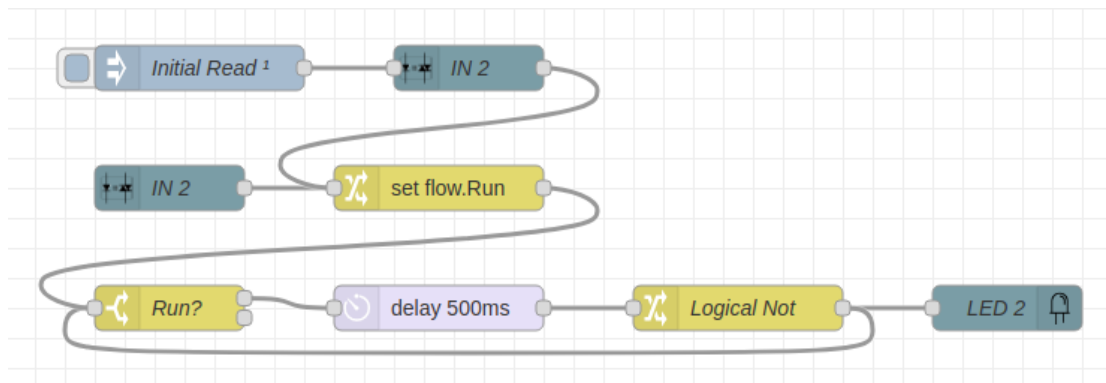


Figure 5-57: Take 4 - Flashing Light Flow with Initialization

See if you can set up the “Initial Read” node (an Inject node) and the top “IN 2” node (an OPTO node) yourself. The “Initial Read” node generates a message when the flow is deployed. This message triggers the read of the IN 2 input, which in turn provides a message with the current switch state. If after thinking about how to configure the nodes your brain overheats and your thought processes grind to a halt look at the node configurations below.

<sup>42</sup> Whether LED 2 is on or off when you flip the switch to the non-flashing position will depend upon exactly when you flip the switch. Fortunately, the specification does not require that the LED be on or off when it stops flashing.

## Learning Kit Workbook (version 1.3)

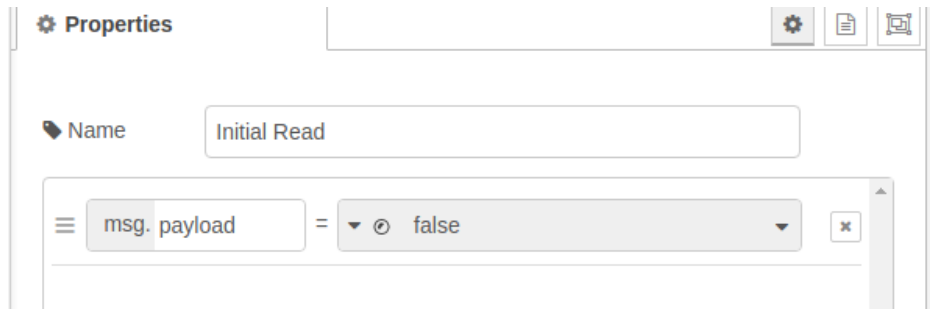


Figure 5-59: Inject Node (Top)

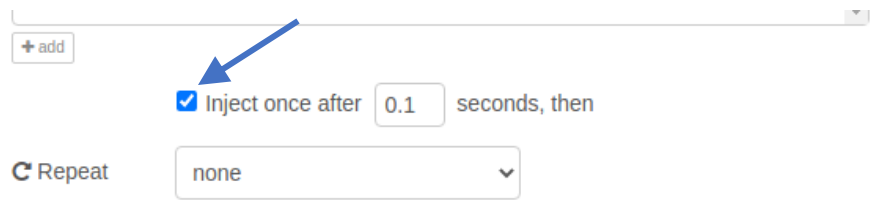


Figure 5-58: Inject Node (Bottom)

For the Inject node you must set the payload and name as in Figure 5-59, but you must also check the box (blue arrow) so that the node will inject the message once just after deployment as in Figure 5-58, directly above.

Here is the configuration of the OPTO node at the top of the flow:

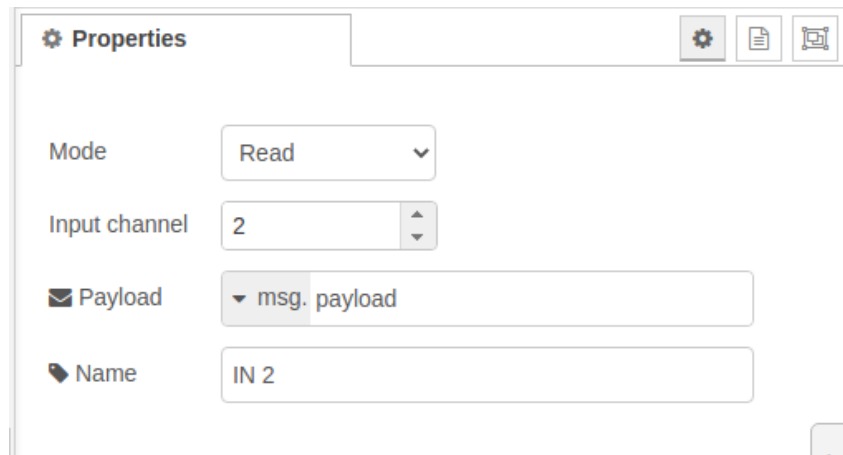


Figure 5-60: OPTO Node Configuration for Read

### Controlling Loops – Counting

Now you know how to start and stop the operation of a loop with a pushbutton or switch. Think for a moment about this very general, and somewhat incomplete, specification:

- When IN 2 is pushed LED 2 flashes 10 times and stops.
- Each flash is one half second on and one half second off.

## Learning Kit Workbook (version 1.3)

Can you think of a way to do this, that does not involve a huge chain for delays (well, after all a chain of 20 delays is very messy)? Can you think of a way to increment a number in `msg.payload` in a loop until it reaches some number, like 10 and then terminates<sup>43</sup> the loop?

Or how about this, make the problem much simpler and see if it is possible to do something small:

- Output the numeric values from 1 to 5 once each second in order in the debug window, and
- Stop

Well, there are only five things to output, so you could just use a big stack of Debug and Delay nodes. However, what if your specification included this additional limitation:

- You can use only a single Delay node of one second.

This means you will need to use a loop. Suppose the number that you want to print out is in the `msg.payload`. How are you going to add one to the number<sup>44</sup> each time through the loop?

Give this some thought. Sleep on it. Mull it over. Phone a friend. Ask the audience.

One possibility that might occur to you is to set up a huge Switch node with 6 output outputs combined with a Change node on each output. If a message came in with a value numeric 0 you could send the message to one of the outputs and then the Change node on that output would change the value from 0 to 1, 1 to 2, and so on.

Here is what that kind of flow might look like:

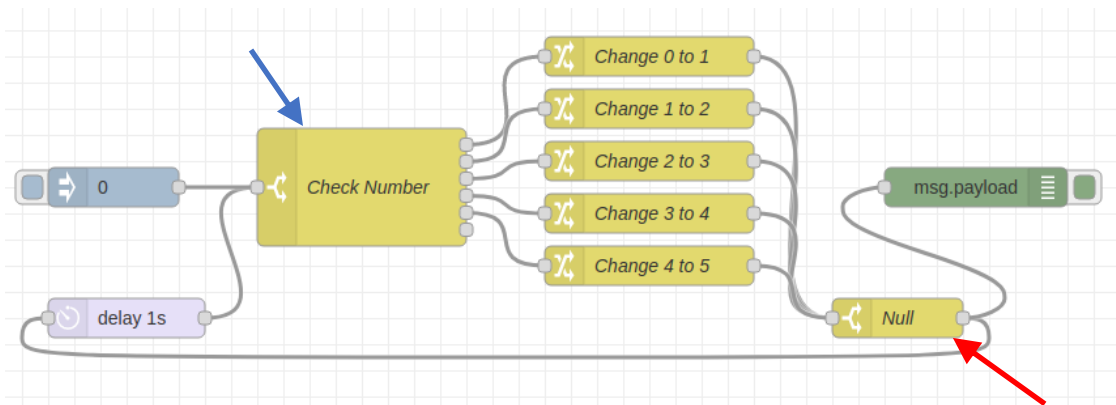


Figure 5-61: Counting Flow Based on a Switch Node

Check out the Switch Node labeled “Check Number” (blue arrow). The purpose of this node is to separate messages with different numbers. Then the attached change nodes increment the number by one using a simple set option. Notice the empty output port on the “Check Number” node. This is where the message goes when it contains numeric 5. It’s the end of the line because the message just vanishes into the ether, gone, never more to be seen and the output messages stop.

Here is what the configuration looks like.

<sup>43</sup> Terminate – computer programming speak for stopping a loop.

<sup>44</sup> “add one to a number” – programmers say “increment the number” or if you subtract one each time they say “decrement the number”.



## Learning Kit Workbook (version 1.3)

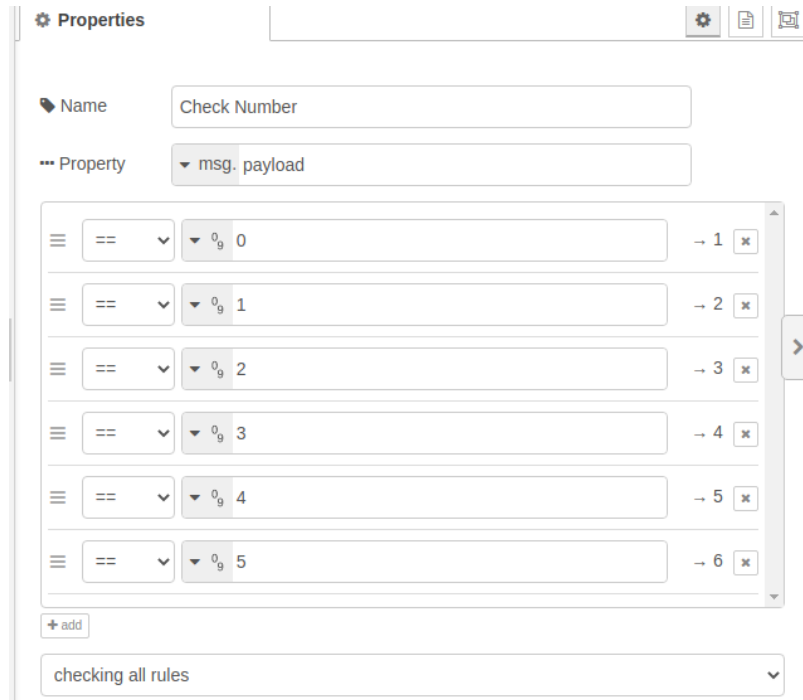


Figure 5-62: Configuration of Switch Node

Also check out the node labeled “Null” (red arrow). This node does nothing but pass the incoming message to the output. In its nothingness it serves two purposes here: (1) to allow each change node to connect to the Delay node without having a big tangle of lines and, (2) it makes the loop clearer because it positions the loop below the Delay node. The “Null” node is a Switch node with only one rule as shown below. Sometimes judicious use of an artifact like the Null node can make your flow easier to understand. Editorial Rant: wouldn't it be nice if Node-RED included a Null node? In any case, you can always roll your own.

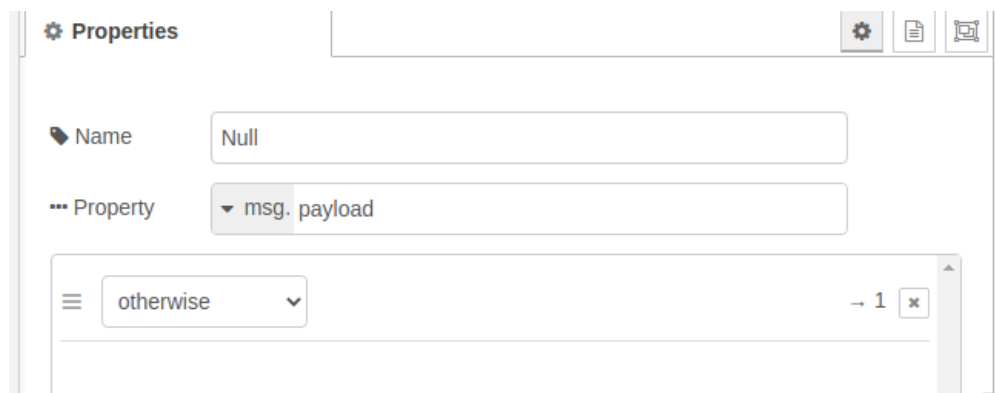


Figure 5-63: Configuration of Null Node.

You should be able to build this flow up on your own. Try it. Deploy it, Run it. Does it output the numbers one through five and stop?

## Learning Kit Workbook (version 1.3)

If you were very clever you could set up a single Change Node to do the same things as the Switch and Change nodes in Figure 5-61, but you would need to carefully arrange the order of the change rules so that a following rule does not modify the results of a previous rule<sup>45</sup>. Here is the equivalent flow using a single Change Node.

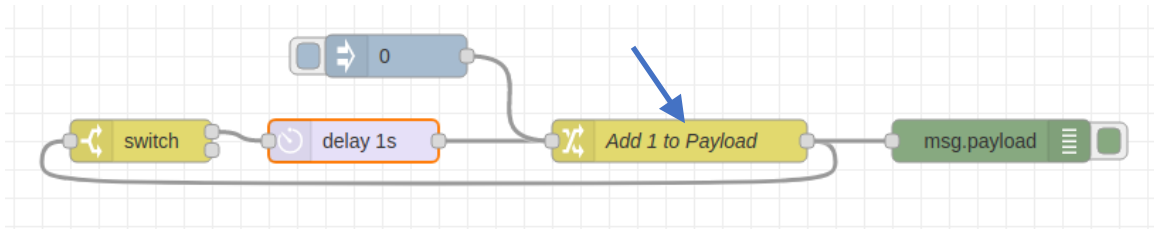


Figure 5-64: Counting Loop with a Change Node

One thing you might say about this flow is that it is certainly a lot simpler looking than the flow in Figure 5-61. No big switch node followed by a stack of change nodes. This is because the complexity is in the “Add 1 to Payload” Change node (blue arrow). However, if you crack open the node you will see this:

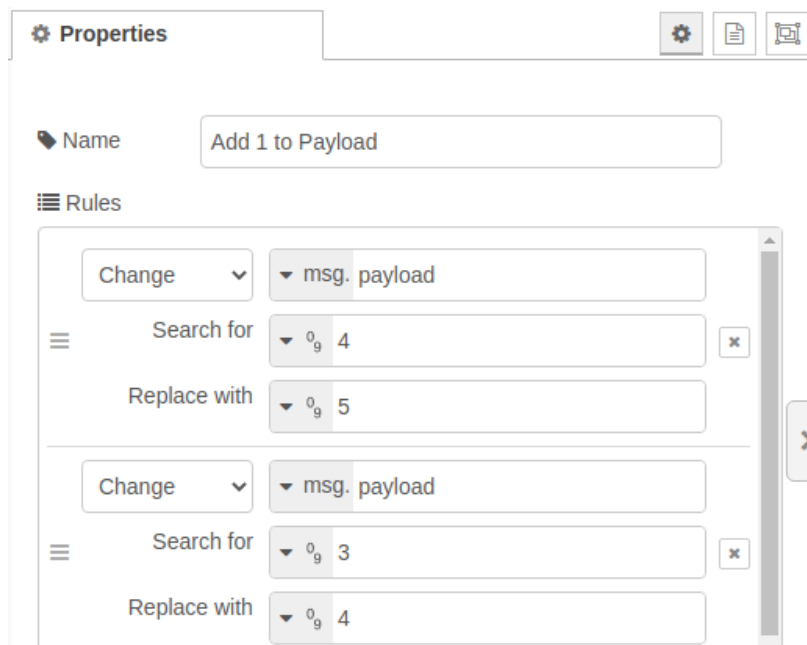


Figure 5-65: Configuration of Change Node (Partial)

The configuration in the figure above is only part of the story. There are a total of five rules, one to increment each payload value, and remember, the rules have to be arranged so that the largest value is modified first. Otherwise, the Change node will mangle your data beyond recognition.

The little Switch node with two outputs detects the end of the loop and if the incoming message contains numeric 5 it shunts the message to the unconnected output an “Adios Message”. Below is the configuration.

<sup>45</sup> For example, the first rule might change 4 to 5, the second rule would change 3 to 4 and so on so that only one of the rules applies each time a message is received.

## Learning Kit Workbook (version 1.3)

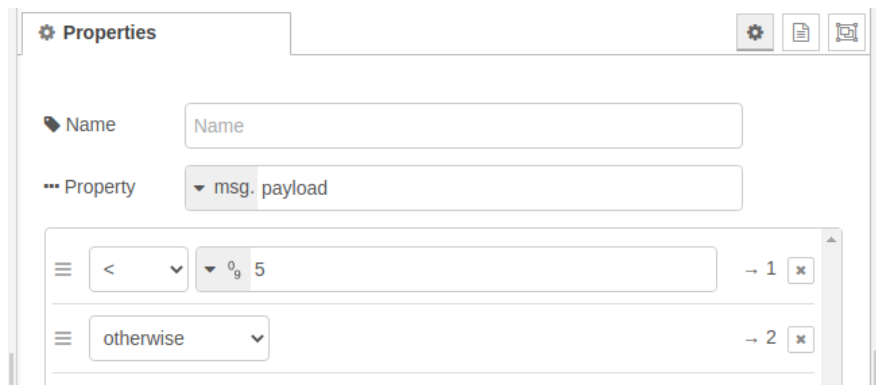


Figure 5-66: Configuration of Switch Node

Not to bad, if you have had the patience to work out some of these flow you will have learned something about building looping flows tht will run for a certain number of times and then stop. However, both of the techniques above are very clunky, which is a precise engineering term for, impractical. Suppose you wanted the loop to run for 20 times, or even 10 times. Building such a flow as above would be really tedious and quite prone to errors. After all every one of the 10 or 20 rules in the Switch or Change nodes must be exactly right or disaster will ensue.

There must be a better way. In fact, there is and it does not require learning about a new node, just how to use the nodes you already have in a different way.

### Simple Expressions – Something You Can Count On.

Typically programming languages allow you to write algebraic like “equations”<sup>46</sup> to process data. Node-RED is no different. The Inject, Switch and Change nodes (among others) allow you to perform simple arithmetic, logical and string manipulations on context data and data in messages. This is very handy and allows you to greatly extend what you can do in Node-RED without having to master Java Script.

The mechanism that supports this is called “JSONata”<sup>47</sup>, which is a simple language that allows you to write “expressions” to transform and combine data. Here you will be using some basic, nay trivial, JSONata expressions, but once you get a handle on the basics you will probably want to look [here](#) and learn more.

You are already familiar with expressions, unless you are a child prodigy in third grade and just happened to develop and unquenchable interest in Node-RED thereby skipping algebra. From algebra<sup>48</sup> you are familiar with statements like:

- $x + 1$
- $3x + 5$
- $2y + 5x + 7$

<sup>46</sup> They look like equations but be warned they are not equations in the algebraic senses. Rather they are just directions on how to process data.

<sup>47</sup> JSONata – JSON refers to the underlying data structure of Node-RED nodes and messages. JSONata is an expression language that allows you to manipulate a JSON “object”.

<sup>48</sup> Take home assignment: research the origin of this word. You might be surprised.

## Learning Kit Workbook (version 1.3)

And so on and so on. These are expressions, which are simply written statements that tell you how to combine the value of variables ( $x$ ,  $y$ ) with the value of constants (2, 3, 5, 7) to get a value. Trivially, if someone told you that the current value of  $x$  is 37 and asked you what the value of the expression  $x + 1$  was you would instantly say 38. Similarly for the other expressions shown above. Someone gives you the current values of the variables and, presto, you can evaluate the expression and give them the value of said expression. This is something you did all the time when you started taking algebra. Probably you have fond memories of quiz after test after exam where you were asked to evaluate an expression given some values. It's the same here, except that your friend JSONata is going to do all the heavy lifting for you.

Expressions in JSONata look and feel just like algebraic expressions, so if you know how to express something algebraically you are ready to use JSONata. However, you must keep a few things in mind:

- Variables in an expression are not limited to  $x$ ,  $y$ ,  $z$  and all those bland letters you remember from algebra. Instead, a variable may have any of valid data context variable name (see Context Data Variables). So, "LedNumber" is just as good as "x" and much clearer.
- In place of traditional algebraic symbols  $+$ ,  $-$ ,  $\times$ , and  $\div$  for add, subtract, multiply and divide, you will use  $+$ ,  $-$ ,  $*$  and  $/$ <sup>49</sup>.
- Use parenthesis as you would in algebra. The usual algebraic rules apply to evaluation, so in something like "5 \* x +3" the multiplication is performed first and then the addition.

Question: What do you do next?

Answer: Start simple!

Open a new flow, save any flows you love and cherish, delete the old flows and build a flow like this:

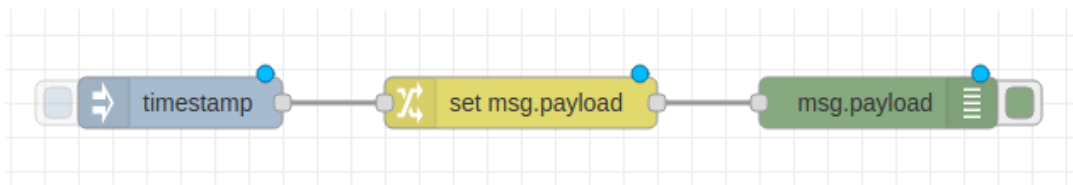


Figure 5-67: Do Nothing Flow – Take 1

Don't bother to deploy this flow because it is not going to do anything useful (or if you are the curious type, think about what it might do, deploy it and see if it worked the way you thought it would.)

Next, configure the Inject node and the Change node as follows.

- Name the Inject node "Inject 1" because that is what it is going to do.
- Get rid of the topic rule because it is not needed.
- Change the payload to be numeric 1

---

<sup>49</sup> The use of  $*$  for multiply and  $/$  for divide harkens back to prehistoric times in computer science (the 1950s) when the only display available was a teletype and the only symbols available were those on a standard typewriter keyboard. Can't use "x" for multiply because that might be the name of a variable. No, " $\div$ " on the standard keyboard, so "/" is the next best thing.

## Learning Kit Workbook (version 1.3)

Your edit window should look like this.



Figure 5-68: Inject Node Configuration

- Click Done.

Open the Change node and...

- Name this node "Do Nothing" because that is what it is going to do.
- Add one rule that sets the `msg.payload` to the `msg.payload`. In other words, the payload is going to pass straight through the Change Node.

Your Change edit window should now look like this:

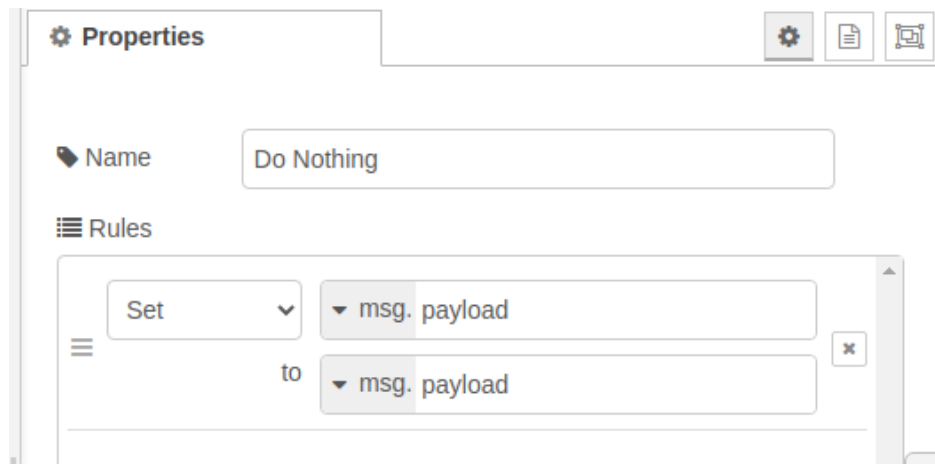


Figure 5-69: "Do Nothing" Change Node Configuration

- Click Done.

Does your flow look like this now?

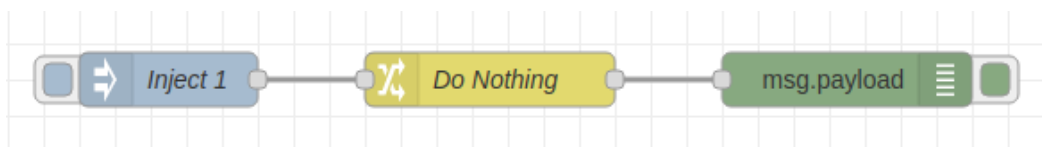


Figure 5-70: Do Nothing Flow – Take 2

## Learning Kit Workbook (version 1.3)

Wonderful. **Enjoyment comes from Deployment, so DEPLOY NOW!**

What do you expect to see in the Debug sidebar window each time you click on the “Inject 1” node. Try it. Were you right? If you saw a numeric 1 every time, then you did everything correctly. Otherwise, visit Debug Town for a few minutes and get it right.

Okay, time to modify this flow so that when you click the inject node the numeric 1 in the *msg.payload* is updated to numeric 2. You will do this by modifying the Change Node to add one to the *msg.payload* using a JSONata expression.

- Open the Change node.
- Change the name to “Add One” because that is its new goal in life.
- Click on the “To” Dropdown and look at the selections (below)
- Click on the JSONata icon<sup>50</sup> (blue arrow). Important don’t be led astray by the JSON icon just because it has the word JSON in it. You really want the icon by the [blue arrow](#).

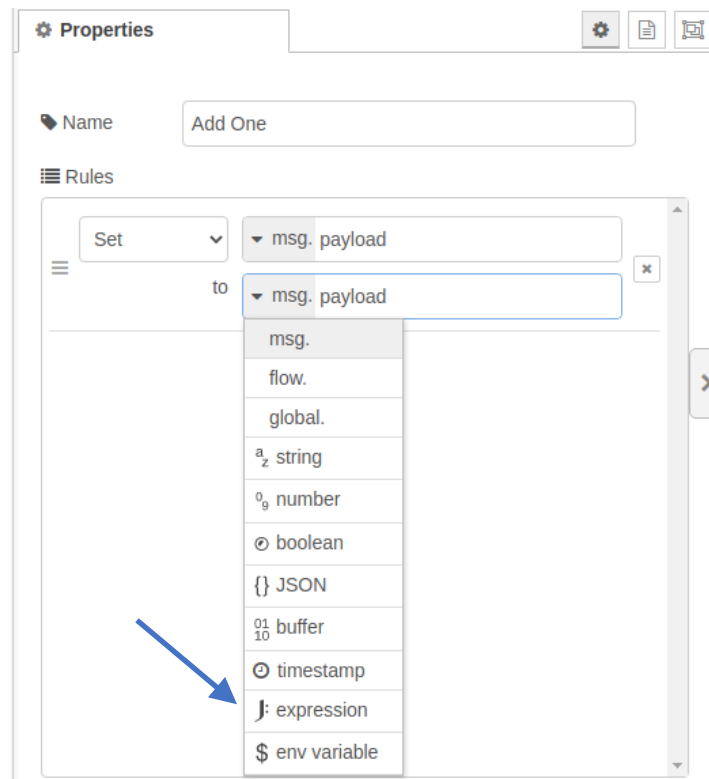


Figure 5-71: To Dropdown Showing JSONata Selection

Next:

---

<sup>50</sup> Very cleverly the Icon for JSONata looks bit like a bass clef sign on a musical staff, making the connection between JSON and Sonata.

## Learning Kit Workbook (version 1.3)

- Go to the box to the right of the “To” drop down and enter “payload + 1” (without the quotes, of course). Your final configuration for the change node should look like the figure below.

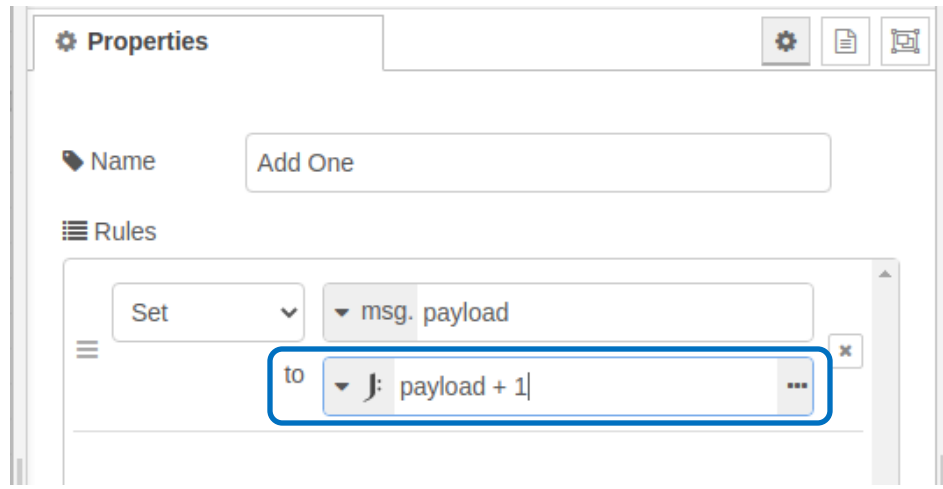


Figure 5-72: Change Node to Add One to the Payload

Before you click Done take a look at the configuration, especially the rule in the [blue box](#). Here is what the rule is saying that the Change node is going to do when it receives a message.

- Get the incoming message.
- Take the value of the message payload.
- Add one to the value of the payload.
- Put the new value into the message payload.
- Send it to the output.

In other words, the Change node is like a little factory that can modify the incoming payload and send it onward. In this case the modification is to add one to the payload. In computer programming terms here is what the rule says:

*“SET msg.payload TO msg.payload plus 1”*

Or in more colloquial terms:

*“Add one to the value of msg.payload”*

Perfect! Now for the big test:

- Click Done

### **Deploy or never see the sun again!**

Clear the Debug sidebar window. Click the inject node. What do you see? If everything went properly you should see the number 2 in the Debug sidebar window. Change the value in the Inject node to some other number, like 5, 37, -19, 14.2. Deploy and try your flow again. It should always add 1 to whatever numeric value is in the payload of the message it receives.

## Learning Kit Workbook (version 1.3)

If you are really curious you might try changing the Inject payload to the string “5”, or Boolean *true*. Deploy and test your flow. You should get a big red warning message in the Debug sidebar window because the JSONata expression is expecting the payload value to be numeric. This is because the little sprite inside Node-RED that evaluates JSONata expressions expects to see a numeric value on both sides of the plus sign. Nothing else will do. Plug in a string, a Boolean or something else and you will get the equivalent of ticket for being a public nuisance and not following the rules. For shame!

Now that your “Add One” node seems to work take it and substitute it for the “Add One to Payload” node in Figure 5-64 and you will have the flowing flow.

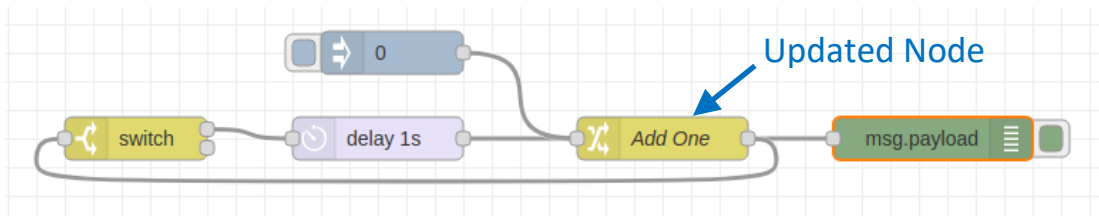


Figure 5-73: Updated Counting Flow

Time to **DEPLOY!**

Click the Inject node and you should see the numbers one through five printed out in the Debug sidebar window.

Now that you have seen how to write a simple expression you can probably imagine how to write more complex expressions. Not only can you modify the *msg.payload* value, but you can also modify other message values (e.g., *msg.led*) and you can perform arithmetic operations on context data variables. You just have to identify them in the expression.

**OPTO Puzzle # 3 - Going Down.** Modify the flow in Figure 5-73, above, to print out numbers in the debug window starting at 5, counting down to 1 and then stopping.

**OPTO Puzzle # 4 - Temperature Change** – Construct a node to convert a number representing degrees Fahrenheit to degrees Celsius. Test it out. If you built the Local Temperature flow from Chapter 3 see if you can substitute it for the Range node.

Let’s wrap this up. Using the flow from Figure 5-73 try to extend it to meet the specification at the beginning of this sections. Remember when you push the button on IN2 LED 2 should flash 10 times and stop. Here is what you must do:

- Modify the flow to count from 1 to 10
- Substitute a pushbutton on input 2 for the Inject node
- Substitute a flow for the Debug node that flashes LED

*Modify the flow count to 10* – Easy to do. Open the Switch node and change the termination condition to 10 as below.



## Learning Kit Workbook (version 1.3)

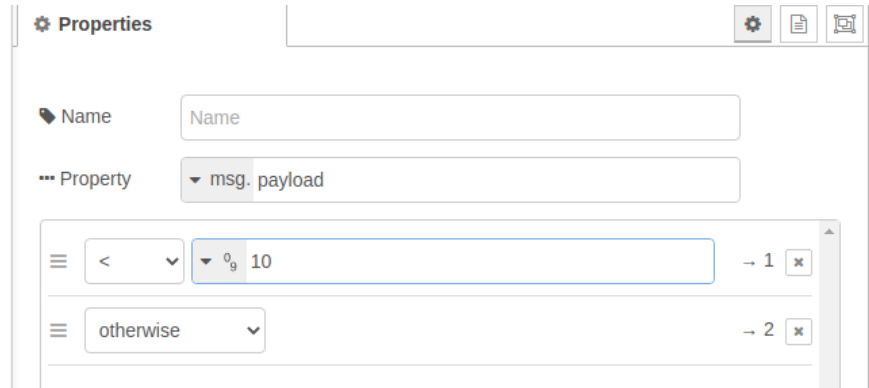


Figure 5-74: Switch Node Configuration

**Deploy** and check to make sure the flow counts from 1 to 10.

*Substitute IN2 for the Inject node* – replace the Inject node with an OPTO node configured as below. You will also need to include a switch node to make sure you only capture the button press and finally you will need to have a Change node to generate the value of numeric 0 to kick the loop off. Below is what the sub-flow will look like. The output (a numeric 0) will go to the input of the “Add One” node.

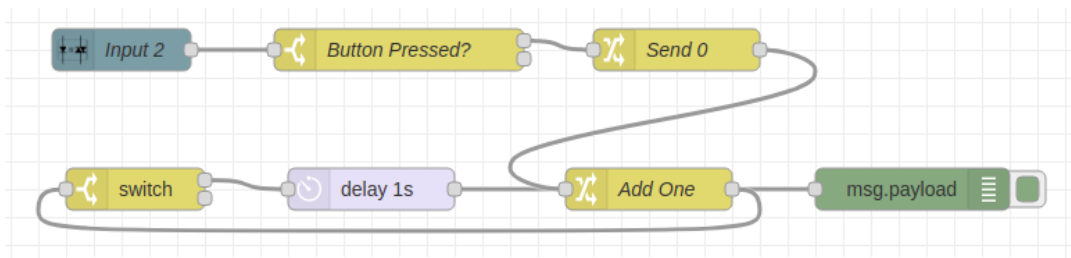


Figure 5-75: Flashing LED with Added Input Flow (Top)

Try setting the three nodes at the top of the flow on your own. If you run aground, then look at the configurations below.



Figure 5-76: OPTO Node Configuration

## Learning Kit Workbook (version 1.3)

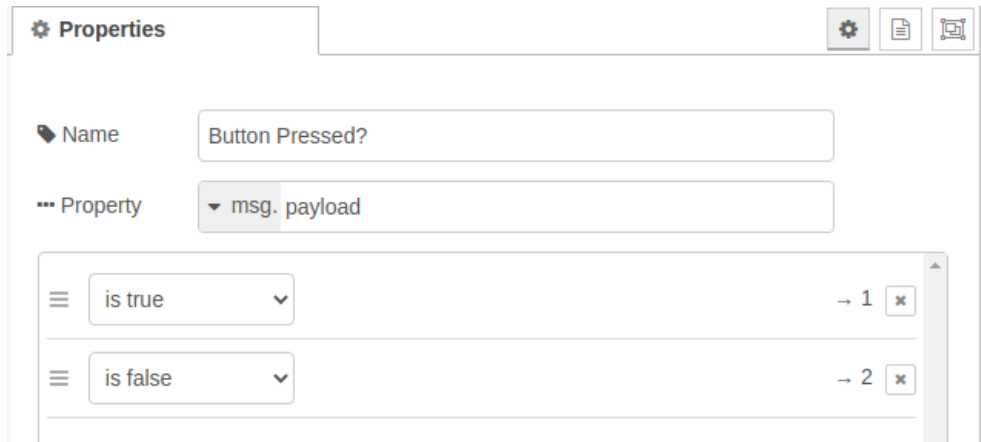


Figure 5-78: Switch Node Configuration (Button Pressed?)

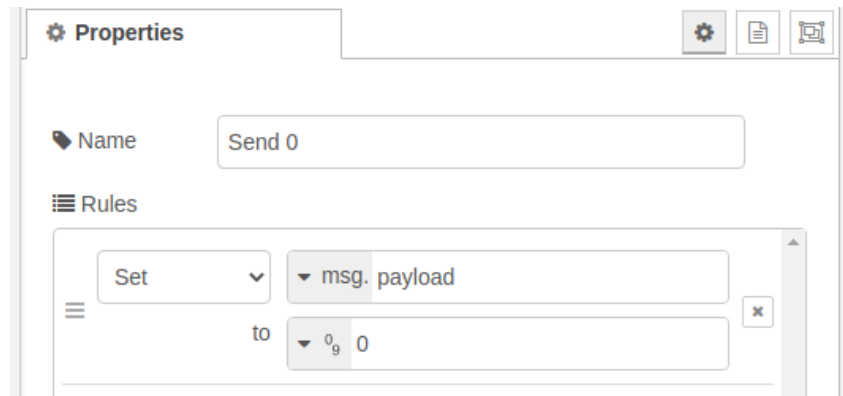


Figure 5-77: Send 0 Node Configuration

Deploy your flow and check to see that pushing the button on IN2 outputs the numbers from 1 to 10.

Add a sub-flow to flash the LED – All that remains is to replace the Debug node with a sub-flow that will flash LED 2 each time a new number is generated. You will first need to change the number in the payload to a numeric 1 to turn the LED on and then one half second later change the message to numeric 0 to turn the LED off. Here is what the final flow should look like.

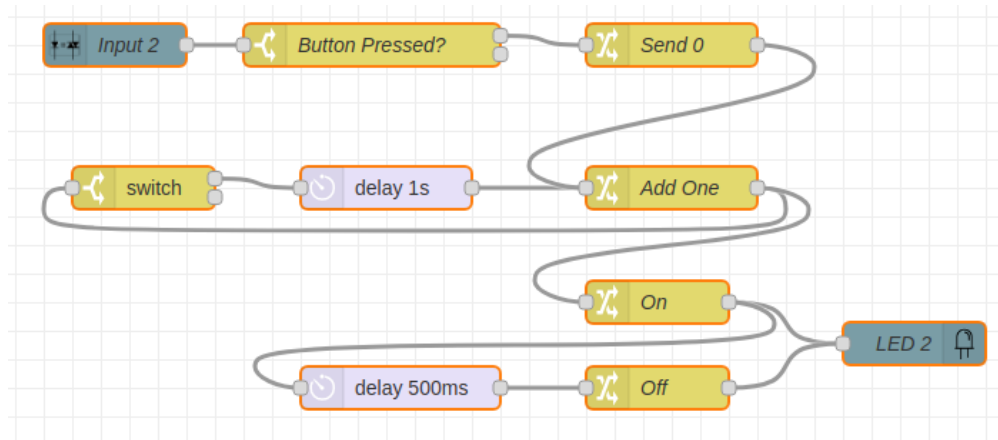


Figure 5-79: Completed Flashing LED Flow

## Learning Kit Workbook (version 1.3)

Again – you can set this up on your own, but if you begin to despair here are the configurations for each added node.

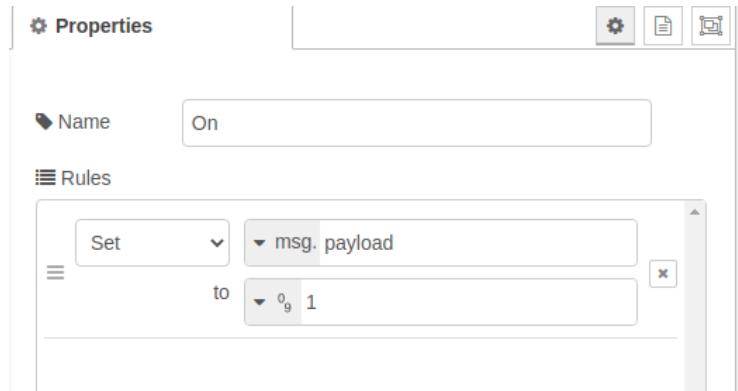


Figure 5-80: On Node Configuration

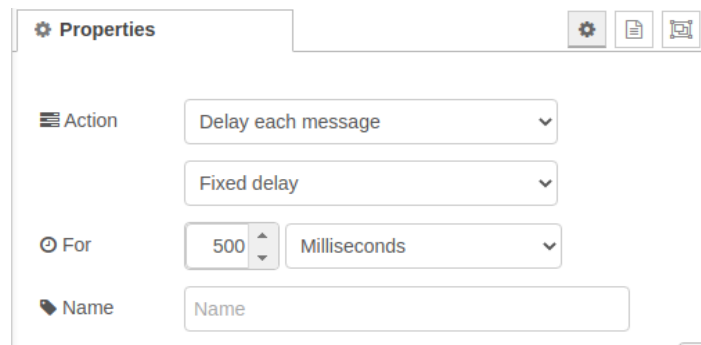


Figure 5-81: Delay 500 msec Configuration

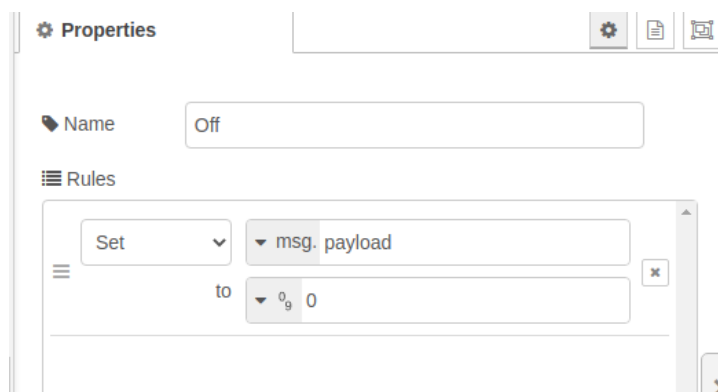


Figure 5-82: Off Node Configuration

## Learning Kit Workbook (version 1.3)

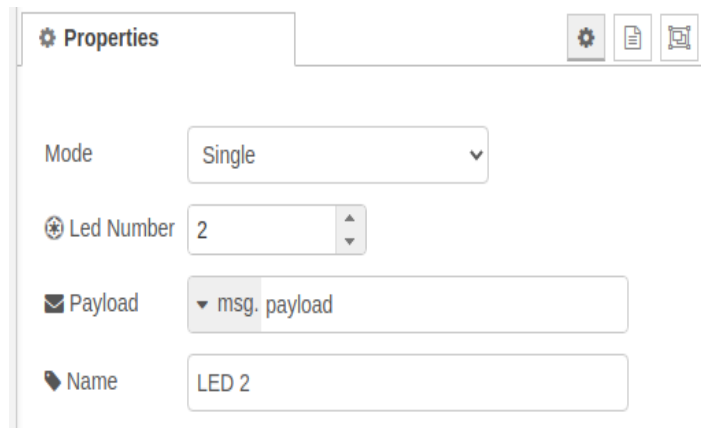


Figure 5-83: LED 2 Node Configuration

When testing your flow remember that the LED might be on initially, in which case you will only see nine flashes instead of 10. If this offends your sensibilities (and maybe it should because it is a bit of a hole in the specification) then you might consider these extensions to the flow of Figure 5-79.

- Add a sub-flow that turns off LED 2 when you push the IN2 button and delays a half second before sending the message on. Or,
- Add a node to clear LED 2 upon deployment.

You should also consider that if you push the button connected to LED 2 before the flashing sequence terminates you will have two messages circulating, which will cause all sorts of confusion in the flow. Therefore:

**Puzzle # 5 - Only One at a Time Please!** – Devise a flow to go between the IN2 button and the remainder of the flow in Figure 5-79 that prevents additional button pushes from initiating the flow until the flow completes flashing LED 2. You will probably need to use a context data variable to block any button push messages that occur before the flow ends. You can set the variable when the button is pressed and reset it with a message from the Switch node output that is not connected.

### Running with the Switches

#### Simple Kitchen Timer

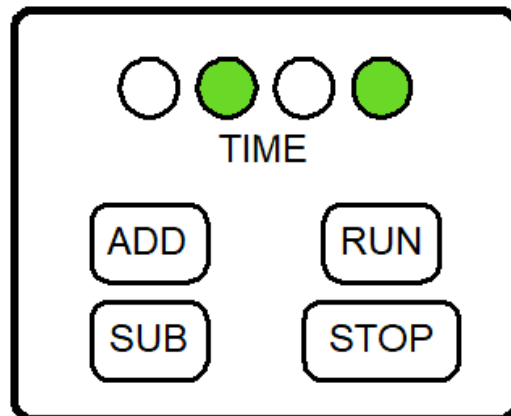
Time to pull all of your knowledge of OPTO nodes, expressions and context data variables together and demonstrate your supreme mastery of Node-RED. Here you will build a simple kitchen timer. In fact, it is so simple that it will only allow you to time your hardboiled eggs for between 0 and 15 seconds! Here is the specification complete with handy number so that you can refer back to them later.

- 1) LEDs GP! To GP4 will display the time remaining in binary.
- 2) All four pushbuttons will be used as follows:
  - IN1 – Stop Timer.
  - IN2 – Start Timer.
  - IN3 – Subtract one second from time remaining.
  - IN4 – Add one second to time remaining.
  - Action occurs when button is pushed.
- 3) IN4 will not add time beyond 15 seconds
- 4) IN3 will not subtract time below zero

## Learning Kit Workbook (version 1.3)

- 5) When the timer is running it will count down the time once each second
- 6) When timer reaches zero all LEDs will flash five times
- 7) The flash rate will be half second on and half second off.
- 8) Time may only be changed (add or subtract) when timer is stopped. If the time is changed while the timer is running the results are indeterminate.
- 9) When deployed the timer will be stopped and show zero time.

Here is an illustration of the timer.



*Figure 5-84: Timer User Interface*

To build and test this flow you will need four pushbuttons, although you could use a single pole double throw (SPDT) switch for the RUN/STOP function because only one input can be used at a time.

This is the most complex project you have encountered so far, so you will need to apply all your knowledge and good engineering habits to it. You will need to think first and program later. To increase your likelihood of success you will need to work incrementally.

---

**Engineering Tip # 11 - Test As You Build** – The great temptation in programming is to get a flash of inspiration and then jump in and start laying down code (or in the case of Node-RED, nodes and wires). Resist. Start with a good plan. More importantly work in a methodical manner.

Most projects are built up from small pieces and even these pieces might be composed of yet tinier parts. Do not rush to construct your whole design all at once. You will end up with a tangle of nodes and wires that you will take an infinite amount of time to debug. Also resist the temptation to build up a bunch of small functional units and then just stitch them together. Building smaller functional units first is a good idea, but you must test each such functional unit first before you go on to bigger things.

It is good practice to break your overall design into smaller sub-designs. It is also important to build tests for each sub-design so that you verify that the sub-design works before you incorporate it into the final design. Once you plug your sub-design into the larger system it will be much more difficult to locate any bugs because there are many more moving parts, so to speak. Good engineers break the development process into pieces, and they test each piece before they begin using it with other pieces. Sometimes testing at the lower level is referred to as “unit testing” and testing at the level of the completed system is

## Learning Kit Workbook (version 1.3)

obviously called “system testing”. Just because a “unit” of code passes a “unit test” does not necessarily mean that it is bug free and will play nicely with others, but at least you will not be trying to locate a simple bug in a much more complex environment.

---

### Step 1 – Initial Design

On any complicated design the first thing to do is to think about how to break your problem down. Here we have a specification, so that is very helpful. Sometimes you need to mull the problem over for several days to get the creative engine inside to rev up and start generating possible solutions. It is usually helpful to sketch out a block diagram or chart that shows the basic components of your design.

After several cups of coffee and a lot of cogitation you might conclude that this project should be broken into smaller, more manageable flows each with its own tab:

- Initialization of the system upon deployment.
- Sensing the buttons.
- Keeping track of time.
- Flashing the lights when timing is complete.

### Keeping Track of Time

You could start into this problem any place, but maybe keeping track of time would be a good place to begin. You might imagine that inside your flow there would be a place that keeps track of how much time is remaining on the timer. You are going to increase and decrease this value with two of the buttons (Add One Second, Subtract One Second) and you are going start and stop the count down with the other two buttons (Start and Stop). This certainly sounds like a loop of some sort. Further, there are two modes of operation the timer is either running (that is, it is counting down) or it is stopped. When the time is stopped you can add or subtract one second from the time remaining. Also, when the timer gets to zero it is going to stop (you can worry about the flashing later).

After you write enough flows (or programs) you will begin to get a feel for what sort of information you need to store. Here, it is more or less clear that you will need some sort of context data variable to keep track of the time remaining. While the timer is running you could certainly keep track of time in the payload, but what would you do when the timer is stopped? You still have to remember the time. The problem is that it would be hard to do this in the payload of a message because you would need to keep the message circulating to retain the value of the current time. Using a context variable means you could hold on to the value even when the flow has not active messages zipping around inside.

Here is another decision: what type of context data variable to use, node, flow or global? Here you must make a big design decision: “Do I use one giant flow, or do I break the problem down into several flows and link them together?” You already have a rough breakdown of your problems, so maybe there should be a flow for each part of the problem. In this tutorial there is a very practical reason to use multiple flows: the flows must be readable when they are turned in to figures! Here the big design decision is to go with multiple flows stitched together with Link In and Link Out nodes. Once you decide to do this you will probably be using global context data variables because the value of the variables will likely need to be accessed from several flows.

## Learning Kit Workbook (version 1.3)

Now that you have thought the approach over you might try to clarify your ideas by making a quick sketch on the back of an envelope.

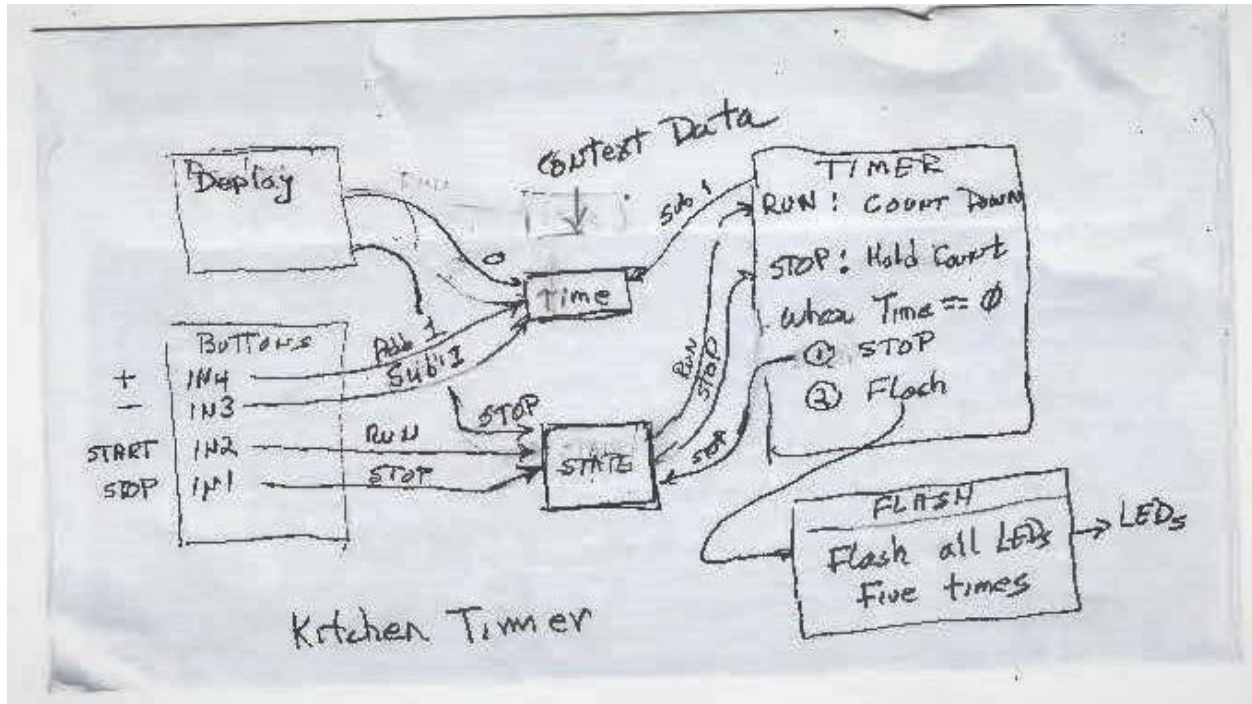


Figure 5-85: Sketch of Possible Approach

### Step 2 – Start Small

Clear the decks: add a new flow, save any old flows you have laying around and delete everything so that nothing in your workspace interferes with your new design.

Where to start? Where to start? Where to start? This is always an important decision<sup>51</sup>.

Start with the context data variables. From the sketch above you are going to need one called “Time” to hold the current time left, and you will need another called “State<sup>52</sup>” to tell whether the timer is running or stopped. For the “State” variable you could use the strings “RUN” and “STOP” to indicate the current state. Now some folks might suggest the Boolean *true* and *false* to represent running and stopped. Or you could use 0 and 1 the same way. However, if you use strings then it will make a lot more sense when you go to debug your flow.

The place to start with context data is in the deployment because you will surely want to set the initial value of these as soon as you deploy your flow. Create the flow below in a tab labeled “Deployment”.

<sup>51</sup> “Begin at the beginning,” the King said, very gravely, “and go on till you come to the end: then stop.” (Lewis Carroll – Alice in Wonderland)

<sup>52</sup> “State” – a common noun in Programmer-Speak. It just means the current condition of some aspect of the design, e.g., the state of the light is “on” or “off”, the state of valve is “open” or “closed”.

## Learning Kit Workbook (version 1.3)

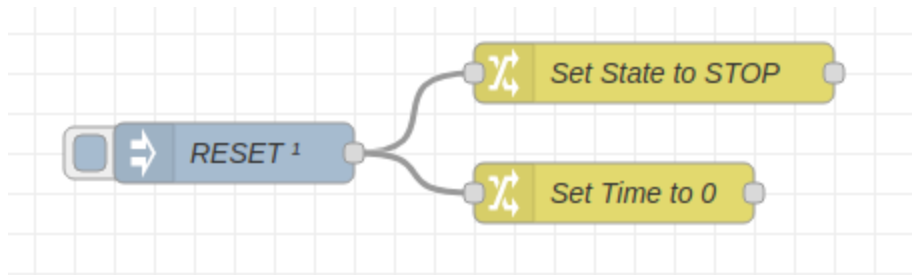


Figure 5-86: Deployment (Take 1)

This is the initial shot at deployment, but you may need to revisit it. Remember: engineering is usually an iterative process. You can plan and plan and plan, but even after you build your flows you will still need to make adjustments for things you have overlooked.

The RESET node simply generates a message to trigger the initialization of the State and Time context variables. You should be able to set this up on your own, but if you have trouble here are the configurations.

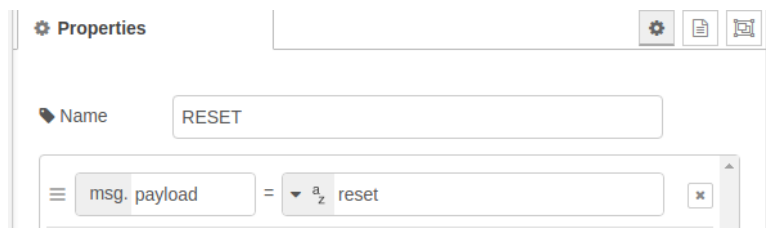


Figure 5-87: Reset Node Configuration

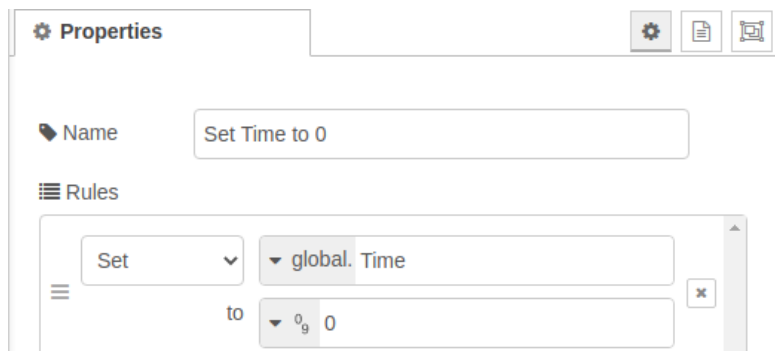


Figure 5-88: Set Time Node Configuration

Step 3 – Test As You Build

**Testing** – now is the time to test this small flow and make sure it does what you think it will do. Testing this flow is easy: **Deploy it!** Once you do this you can check in the Context Data tab under the Global dropdown. Remember that you must refresh the view to see the current value of the Time and State context data variables. If you see the State variable set to the string “STOP” and the Time variable set to numeric 0 then your flow works.



## Learning Kit Workbook (version 1.3)

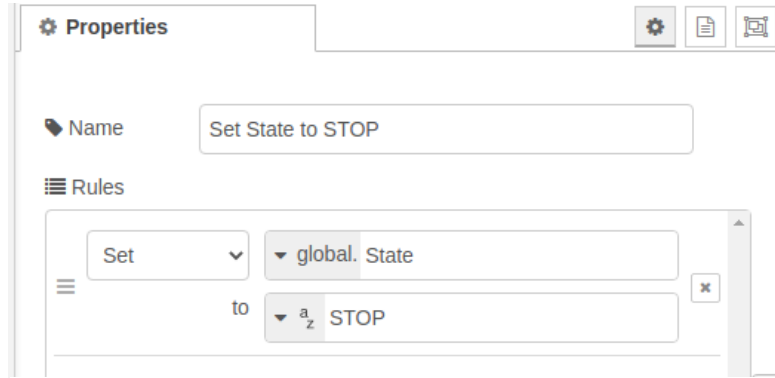


Figure 5-89: Set State Node Configuration

### Step 4 – Build and Test More Flows

Look at the sketch on the back of the envelope (Figure 5-85). Maybe the next flow to develop is the one related to the buttons (but, you might have a better idea). Open up a new flow tab and name it: “Buttons”. Your initial flow is just going to update the State and Time context variables according to the specification. You will find out later that you will need to do more than this, but this is a good place to start with the buttons. Here is initial Button flow for the Start and Stop buttons.

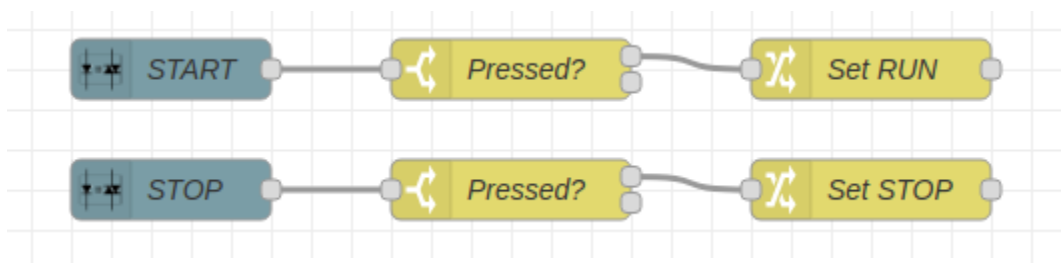


Figure 5-90: Button Flow (Take 1)

This flow should look somewhat familiar because it is the standard way to handle a pushbutton. The START and STOP inputs (IN 2 and IN 1) pick up the button events (pressed and released). The “Pressed?” nodes as Switch nodes that keep the button presses but throw away the releases. Then the “Set Run” and “Set STOP” nodes update the State context data variable. If you get stuck look back at the discussion related Figure 5-75. Below is the configuration of the “Set STOP” and “Set RUN” nodes.

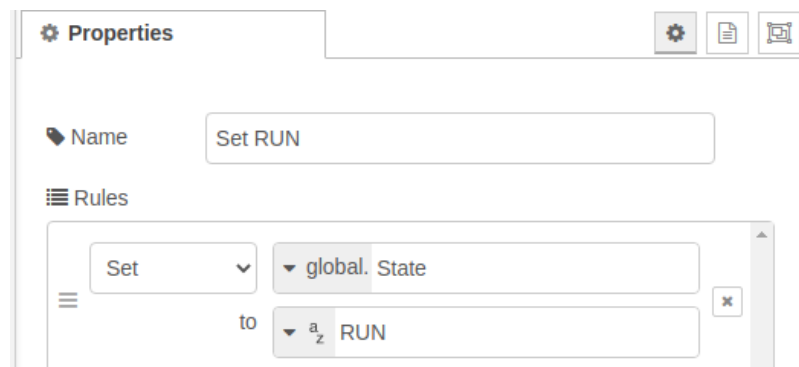


Figure 5-91: Set Run Node Configuration

## Learning Kit Workbook (version 1.3)



Figure 5-92: Set Stop Node Configuration

### DEPLOY IT!!!

**Testing** – All you need to do to verify this flow is to deploy it and then try the IN1 and IN2 buttons. Press one of the buttons and then check the context data sidebar. Don't forget to refresh the context data or you will not see any changes. Try each button several times.

Next up: augment the Button flow with the Add One Second and Subtract One Second flows. These are more complex than the START/STOP button flows because you need to perform minor arithmetic with JSONata. Look back at Figure 5-85. It looks like all you need to do is add or subtract one second from the State variable. However, you must also keep the time from going above 15 or below 0 because that is part of the specification (items 3 and 4) and the limit of the four LED display. Do you think the flow below will do the trick ("Take 1" because you are going to need to enhance it later, but good enough for now.)

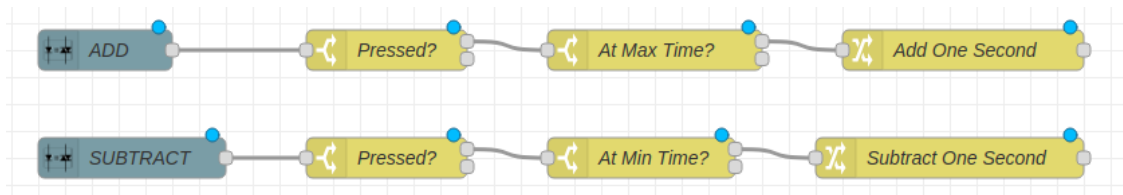


Figure 5-93: Add and Subtract Time Buttons (Take 1)

## Learning Kit Workbook (version 1.3)

You have already used the first two nodes in each flow (LKit OPTO and “Pressed?”) before. The other two nodes are new configurations and a bit tricky. Here are the configurations for the “At Max Time?” and “At Min Time?” nodes.

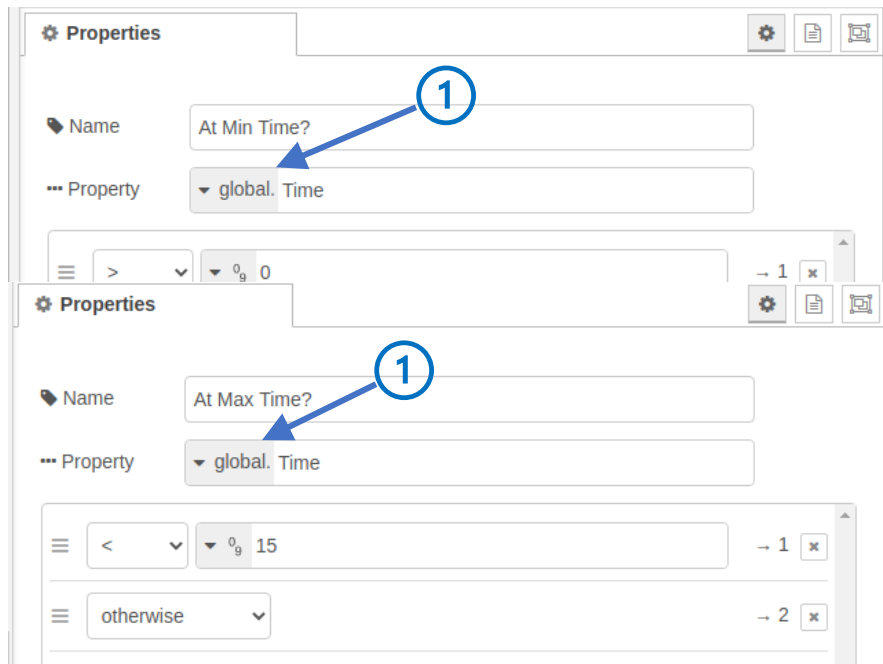


Figure 5-94: At Max Time Node Configuration

The purpose of the “At Max Time” node is to check the current value of the context data variable Time (*global.Time*) and if it is less than 15 seconds then the messages exit at output port 1 and the Time value gets incremented in the next node. However, if the value of Time is already 15 the messages exit output port 2 and vanish into the abyss. This prevents *global.Time* from becoming greater than 15. In a similar manner the “At Min Time” node prevents the *global.Time* value from ever going below zero. Remember: to reference a global context data variable you must specify “global” as the prefix ①.

Here are the configurations for the Add One Second and Subtract One Second nodes.

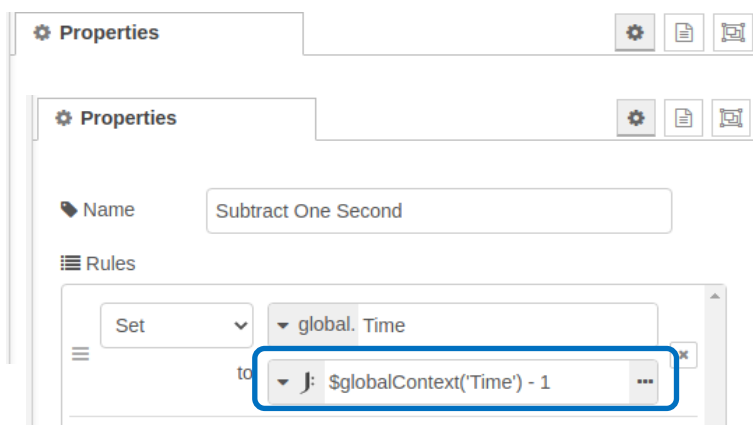


Figure 5-96: Subtract One Second Node Configuration

## Learning Kit Workbook (version 1.3)

Look carefully at the JSONata expressions in the **blue boxes** because they are different in form from what you worked with previously. The objective is to update the *global.Time* variable by plus or minus one second. However, to access a context data variable, *global.Time*, you must use a function rather than just mentioning its name as you can do when the data is contained in a message.

A function in JSONata is a name that starts with a dollar sign (\$) and then has an “argument” in parenthesis. This is just like the functions you learned about in algebra and trig, like *sine* (30), *cos* (90) etc. The name, (sine, cos) tells you what the function is going to do and the value in the parenthesis is the argument, which is what the function is going to use to compute your answer. It’s the same in JSONata except you must start the function name with the dollar sign<sup>53</sup>.

**Important:** there is one more thing you must do to access the *global.Time* variable... you must enclose the name of the variable in quotes. This is an artifact of JSONata because you are really accessing data in a JSON data structure, and when you do this you must use a string, like 'Time'. Working with JSON data structures is an advanced topic, so for the moment just plug in the functions as shown.

Deploy NOW!

Time for, you guessed it, **TESTING**. All you need to do is press the IN4 and IN3 buttons and check that the value of Time changes properly. Don’t forget you must update the Global dropdown in the Context Data sidebar to see the changes. If the value is incrementing and decrementing properly then check that it never increments above 15 or decrements below 0.

### Step 5 – The Main Timer Flow

Time for the central flow: the main timer loop. According to the specification when the timer is in the RUN mode the time must count down once each second to zero (items 5 and 6). If the state changes to STOP the timer must stop counting but hold the current time value (Item 2). Once the timer reaches zero it must flash the LEDs (item 6)). Assume here that the flashing of the LEDs will be done in a separate flow and all you need to do now is to generate a message when the timer reaches zero.

Here is one possible flow with Inject nodes and Debug nodes added for testing purposes.

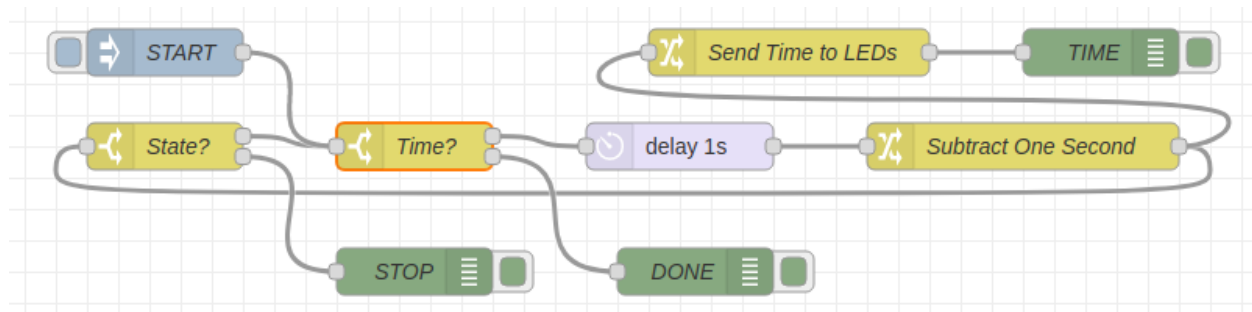


Figure 5-98: Timer Main Loop with Inject and Debug Nodes for Testing

<sup>53</sup> Sorry, it does not look like JSONata includes trig functions.

## Learning Kit Workbook (version 1.3)

This looks more complicated than it is because there are an Inject Node and three Debug nodes attached so that you can test the flow. Note that the Inject and Debug nodes have been given names to make the testing process easier.

See if you can construct this flow based on Figure 5-85. When you click the Inject node labeled “START”, it is going to inject a message to kick off the loop. What should that message be? It might be a payload with *global.Time* in it, or it could be an arbitrary message just to activate the loop. You may have other ideas. In this example it is the latter, simply a Boolean *true*, whose only purpose is to activate each node.

Cogitate for a moment on the overall flow. When the State of the timer is “RUN” each time the message makes a pass around the loop one second is subtracted from Time (“Subtract One Second” node) and this time is sent to the LEDs (“Send Time to LEDs” node). Then the “State?” node checks to see if the State is still “RUN”. Next, the “Time?” node checks to see if the timer has expired. Below are the configurations for the loop nodes, but before you jump ahead see if you can build this flow on your own.

Time passes... See if your node configurations are like those below, or perhaps you have a much better approach that is equivalent.

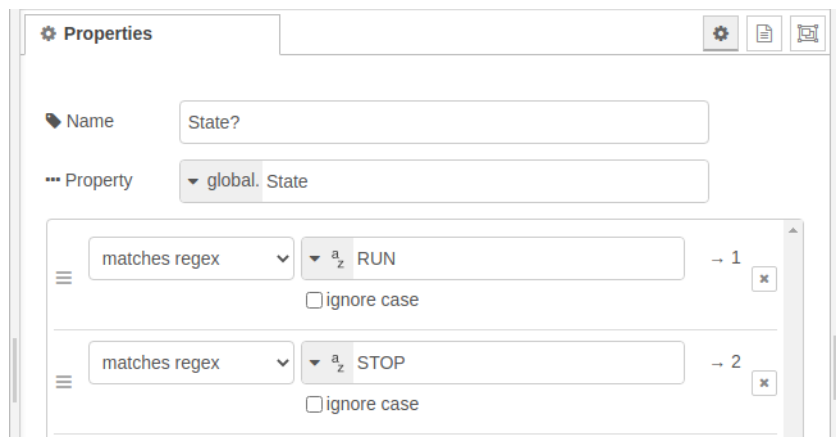


Figure 5-99: "State?" Node Configuration

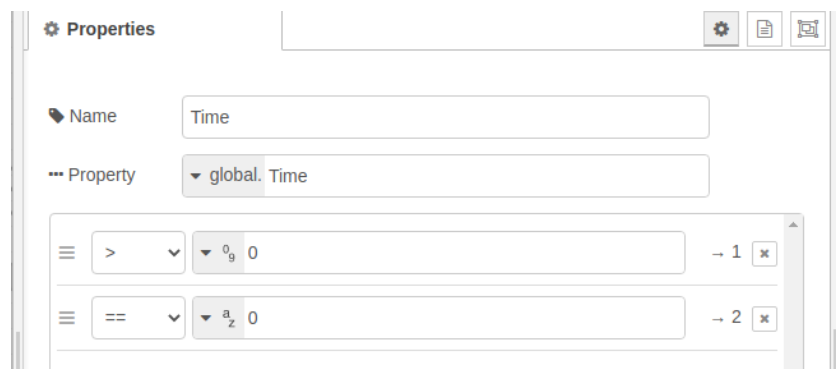


Figure 5-100: "Time?" Node Configuration

## Learning Kit Workbook (version 1.3)

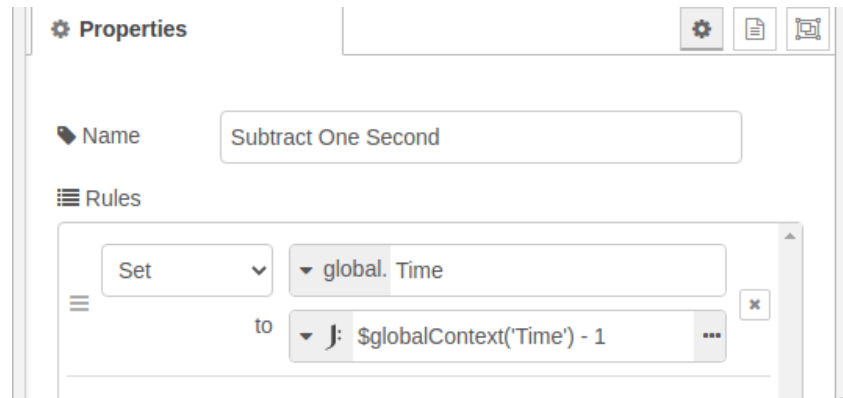


Figure 5-102: "Subtract One Second" Node Configuration

The "Subtract One Second" Node is just like the node in Figure 5-96 use to set up the Time variable.

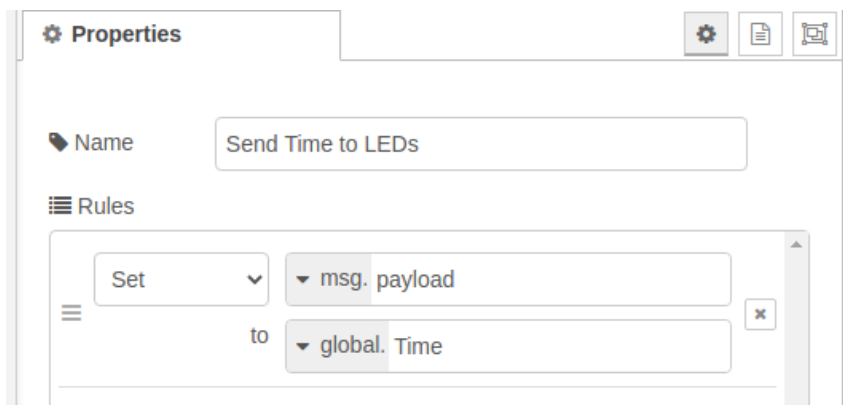


Figure 5-101: "Send Time to LEDs" Node Configuration

In this implementation of the timer loop the circulating message is simply Boolean *true*. Therefore, when *global.Time* is updated it is necessary to place the new Time value in the payload of a message and send it off to the LEDs for display because you want to see the time counting down second by second (item 1 of the specification).

**Deploy** your flow and hope for the best!

Time for **Testing** – This is a complicated flow to test. It might even be necessary to write down a small test plan. Here are some of the things you probably want to check.

- Test 1 - When the timer is running is the context data variable Time being decremented once each second.
- Test 2 - Does the loop stop when Time reaches zero
- Test 3 - Does the loop stop if the State is changed to "STOP"
- Test 4 - Does the loop resume counting if the State changes to "RUN"

You might be able to think of other things to test to make sure the loop meets the specification. If you have built this flow in the sequence of the Tutorial, then you can use your other tested flows to help you

## Learning Kit Workbook (version 1.3)

validate this flow. Also remember to check the data context variable values for Time and State during your testing.

**Test 1** – How about this approach?

- **Step 1** – Press IN 2 to change the State to “RUN”
- **Step 2** – Check context data variable to make sure State is “RUN”. (However, the counter will not count down because it is not yet connected to the updating of the State. Instead you will use the “START” Inject Node as below in step 4)
- **Step 3** – Open and Clear the debug sidebar window.
- **Step 4** – Click the “START” Inject node on the Timer Flow.
- **Step 5** – Verify in the debug sidebar that the output of the “TIME” Debug node counts down to zero.

If your flow passes this first test that is a very good sign because that the main aspect of the loop works. However, you must always test for the unusual, e.g., does the loop stop at zero (Test 2), which would be a good thing. You can figure out the other tests. Be thorough because finding and squashing bugs in a single flow is much easier than trying to find a bug when all your flows are connected up. Careful debugging is a key to successful engineering and programming. If you become skilled at testing and debugging, you will never have to stand in the unemployment line.

**Test 2, Test 3 and Test 4** – You **can** do it! Write out a step-by-step plan for each test. Carry out the plan and find out if your timer flow is working properly.

Step 6 – Completion Flasher Flow

Final Flow – when the timer runs down all the LEDs must flash on and off five times. Will this flow work?

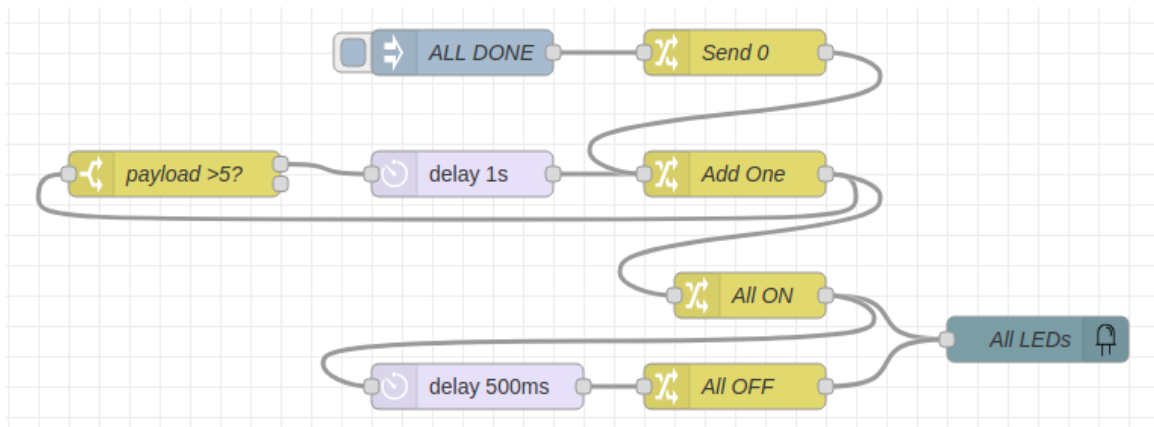


Figure 5-103: Flasher Flow

This flow is almost the same as that shown in Figure 5-79 with the only difference being that the count is five flashes and this flow flashes all the LEDs. No hints here, you can build it! If you get into trouble, go back and look at the discussion above around Figure 5-79.

Deploy to employ!

## Learning Kit Workbook (version 1.3)

**Test it!** This flow is simple. When you inject any message into the “Send 0” node the payload is set to 0 and circulates around the loop five times. Each time around the LEDs flash. Click the “ALL DONE” node to check it out.

### Step 7 – Connecting the Flows

You have four flows and you have tested each one. Now is the time to connect the flows using Link In and Link out nodes and perhaps add a dash of Change nodes here and there to get the messages between flows to work properly. Let’s Go!

You have four flows:

- Deployment
- Buttons
- Timer
- Flasher

**Deployment Flow** – When you deploy your kitchen timer flows the State is set to “STOP” and the Time is set to 0. What about the LED? Opps, better set those to zero to match the state, just in case they are set to some other value from some previous work you did (item 9 of specification). Easy to do... just add a Link Out node to the “Set Time to 0” node of Figure 5-86 and send it to a Link IN wired to the “All OFF” node of the Flasher flow in Figure 5-103. Make life easier and name the Link Out and Link In nodes the same name, like “Clear LEDs”. Don’t forget to open each Link node and connect them to each other.

Now your Deployment flow should look like this (if you click the Link Out node to see its connection to Flasher flow.)

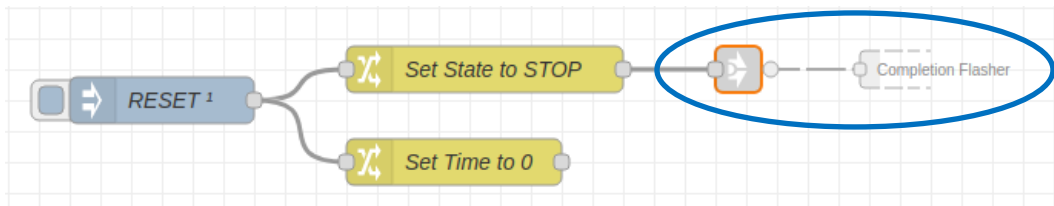


Figure 5-104: Deployment Flow (Take 2)

And on the receiving end, the Flasher flow, you should now have this.

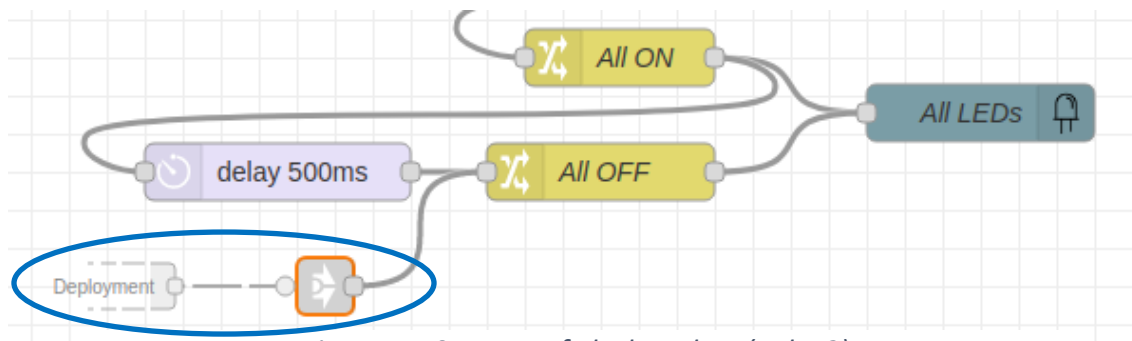


Figure 5-105: Part of Flasher Flow (Take 2)



## Learning Kit Workbook (version 1.3)

*Deploy and Test it!*

**Testing** – To test this addition to your timer you will need to set up another flow with an Inject node and an LKit LED node set to group mode. The Inject node should set two or three of the LEDs. Deploy this flow and click the Inject node to turn on the LEDs. Then shift one of the nodes a bit and deploy again. On the second deployment the LEDs should be extinguished. Sometimes testing gets to be tricky.

**Button Flow** - Now let's look at the Button flow, which includes both the START/STOP buttons from Figure 5-90 and the ADD/SUBTRACT from Figure 5-93 as below.

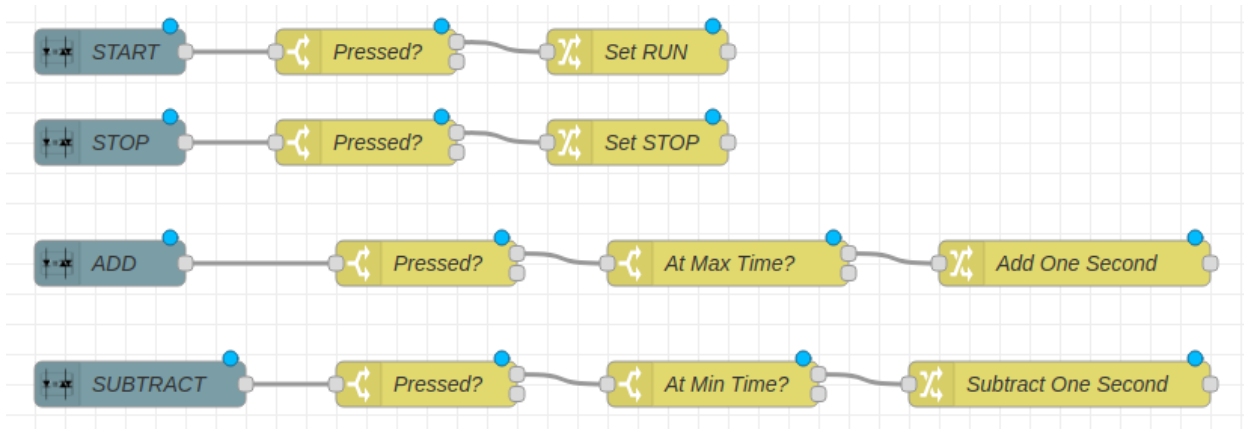


Figure 5-106: Button Flow (Take 1)

When you deploy your flow, the Timer is stopped. Pushing the Start button should start the flow, but is changing the State to “RUN” enough? What is going to kick off the Timer flow? Nothing if you don't do something. How about taking the output of the “Set RUN” node and connecting it to where the START node enters the timer loop on Figure 5-98? That way when you push the START button it will start the timer countdown.

Here is the modification to the START Button sub-flow.

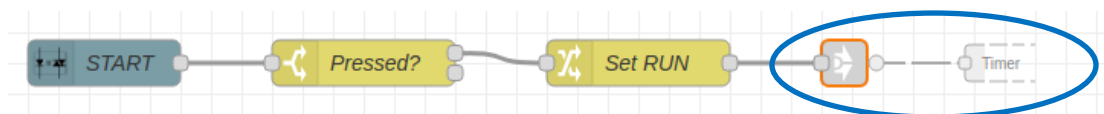


Figure 5-107: START Button Sub-flow (Take 2)

The message from the “Set RUN” node connects to the Timer Flow (partially shown below)

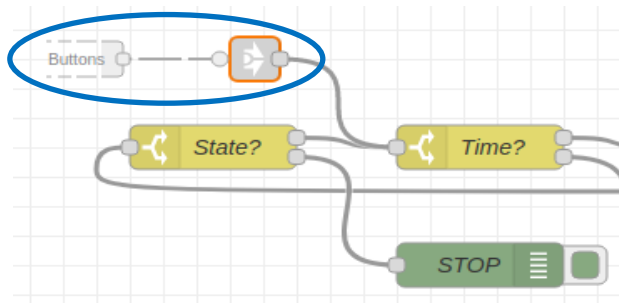


Figure 5-108: Part of Timer Flow (Take 2)

**Deploy and Test.**

## Learning Kit Workbook (version 1.3)

**Testing** – after you deploy your flow push the ADD button a few times and then check that the Time variable has the correct non-zero number in it (5 or 6 will do). Switch to the Debug sidebar, clear it and push the RUN button. You should see the output of the “TIME” Debug node count down to 0.

**ADD/SUBTRACT Button** – Each time you push one of the ADD or SUBTRACT the LEDs should show the new number. This is going to require modification of the Button Flow and a connection to the LEDs in the Flasher flow. Below is the Button flow with modifications to send the value of the context data variable TIME to the LEDs.

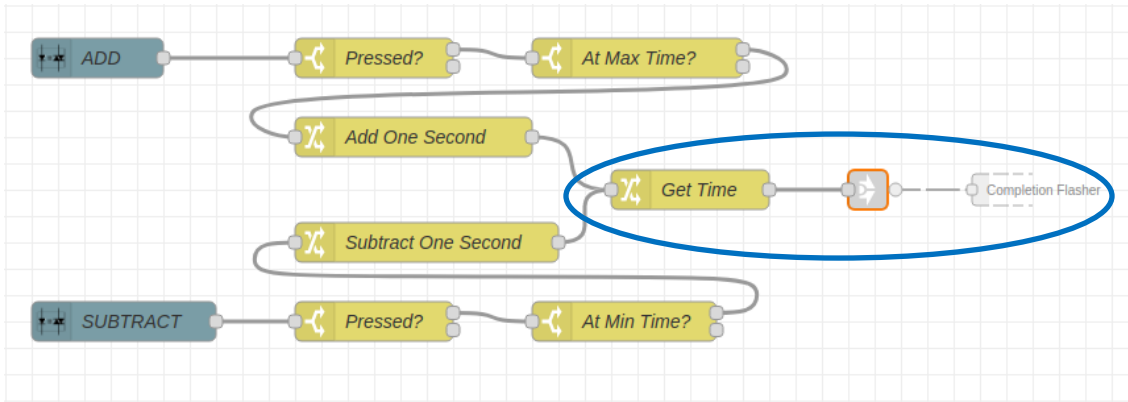


Figure 5-109: Button Flow (Take 3)

Time is held in a context variable and the “Add One Second” node, for example, updates this variable, but does not send the current value in the message that it outputs. Therefore, you will need to add a Change node to copy the value of *global.Time* to the payload before you send it off to the LEDs. Then add a Link Out node and name it “Display Time”. Check out the configuration of the “Get Time” node below.

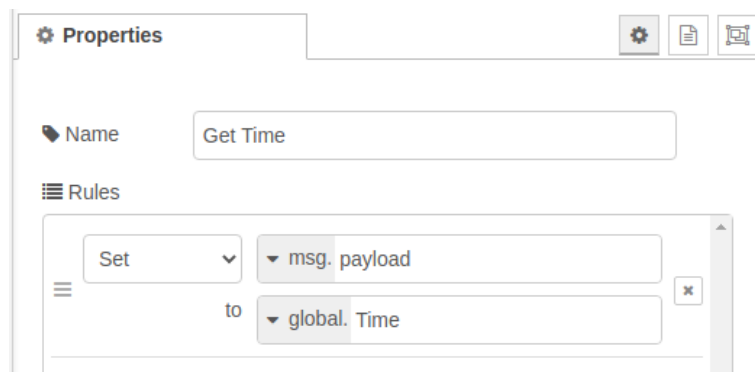


Figure 5-110: Get Time Node Configuration

Go to your Flasher Flow and add a Link In node named “Display Time” and wire it to the “All LEDs” node. Open the Link In node and connect it to the “Display Time” node in the Button flow. Now your Flasher flow will look like the figure below.

## Learning Kit Workbook (version 1.3)

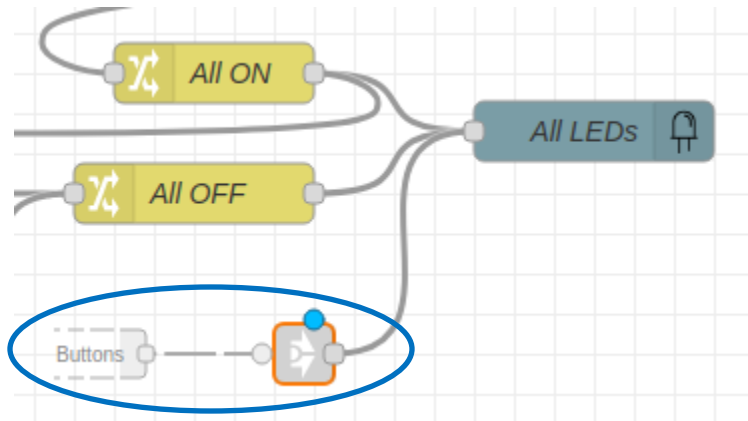


Figure 5-111: Flasher Flow (Take 3)

Time to **Deploy** and Test – This is easy. Press the ADD button a few times. Each time you press the button the LEDs should increment by one. When you press the SUBTRACT button the LEDs should decrement by one. Make sure that the LED display stops at either 0 or 15 when you press SUBTRACT and ADD too many times. Does it work? Wonderful! Next up...

**Timer Flow** – Right now your Timer flow only sends the value of Time to the “TIME” Debug node so that you can test your flow as you did just above. It’s time to connect the output of the Timer flow to the LEDs. Do this by replacing the “TIME” Debug node in the Timer flow with a Link Out node. You might call this “Display Time”. Go to the Flasher flow and add a Link In node with the same name to the “All LED” node. Open each Display Time Link node and connect them together.

A partial view of your new Timer flow should look like the figure below.

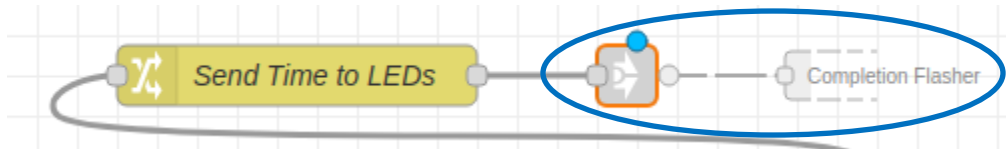


Figure 5-112: Timer Flow (Take 3)

Your updated Flasher flow (partial) should look like this.

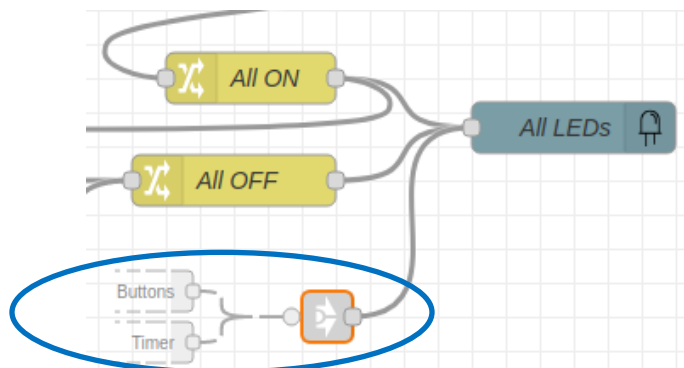


Figure 5-113: Flasher Flow (Take 4)

## Learning Kit Workbook (version 1.3)

Ready, Set, **Deploy!**

**Testing** – Using the ADD button add a few seconds to the timer. Push the RUN button. Your system should count down to zero and then flash the LEDs.

**Flasher** - Your Timer is almost complete. Now it is time to connect the output of the “DONE” node in the Timer flow to the Flasher to indicate when the timer reaches zero. When you were testing your individual flows, you had a Debug node named “DONE” in the Timer flow. Replace that with a Link Out node named “DONE”. On the Flasher flow replace the Inject node named “ALL DONE” with a Link In node named “DONE”. Open each Link node named “DONE” and connect them together.

Does your Timer flow look like the figure below that only shows the addition of the Link Out node?

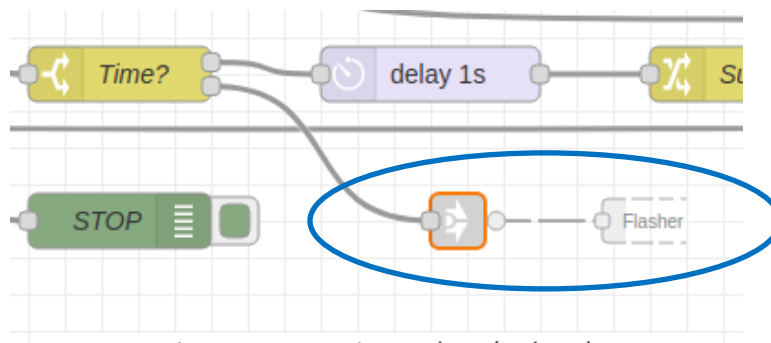


Figure 5-114: Timer Flow (Take 4)

Here is the updated Flasher flow.

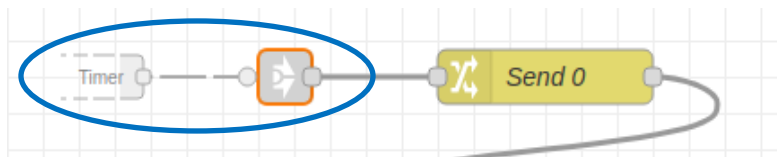


Figure 5-115: Flasher Flow (Take 4)

Deploy for Testing.

**Testing** – Your flow is almost complete. Add some time to the timer with the ADD button. Press the START button. The LEDs should count down to zero and then all the LEDs will flash five times. If it works, there is only one more detail to attend to.

**Timer Flow** – When the timer counts down to zero it would be a good idea to put the timer back into the “STOP” state. Otherwise, next time you go to use the timer you will be trying to add time while the timer is trying to count down. Worse, you will have multiple messages circulating in your timer loop. Ouch!

Go back to your Timer flow and replace the Debug node named “STOP” with a copy of the “Set STOP” node from the button flow. Presto, here is the completed Timer flow.

## Learning Kit Workbook (version 1.3)

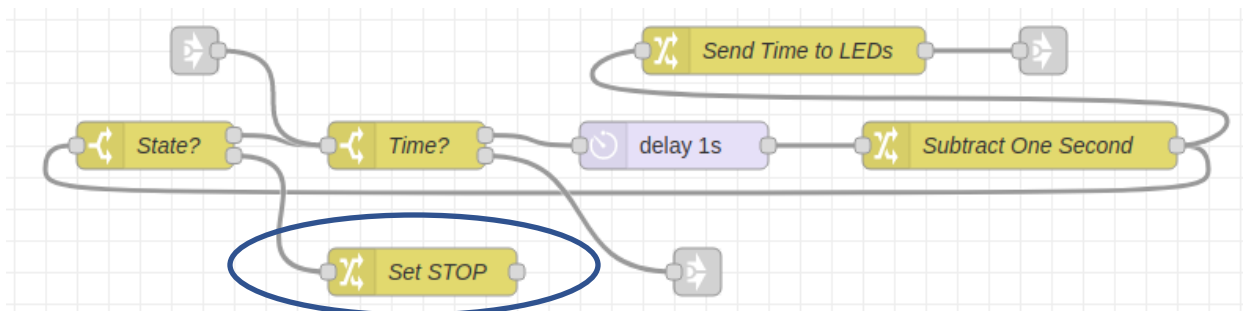


Figure 5-116: Timer Flow (Final!)

Deploy your beautiful flows!

### Step 8 – Final Testing

**Testing** – At this point your Kitchen Timer flow should meet all aspects of the specification. Test each requirement.

**Discussion** – This is not a simple project. If you get it to work even if you needed help you should feel that you have really accomplished something. You needed to draw on all your knowledge of Node-RED. Once you have your project working properly study it carefully because it illustrates many aspects of Node-RED that you will find useful as you go on to bigger projects.

**Puzzle # 6 - Is it Complete?** Go over your Timer project carefully with a giant magnifying glass. Are there any holes in the way it operates? Was the specification you started with really complete? Did it cover all the crazy things someone might do with your timer (aside from dropping it into a pot of hot oil they are frying chicken in)? If you find a shortcoming, see if you can fix it. Here's a hint: what happens if someone presses the RUN button while the timer is operating<sup>54</sup>? Can you think of a way to block the RUN button until the timer runs down, or the stop button is pressed?

If you go on to a stellar career designing products you will spend a great deal of time trying to determine all the strange and wonderful things your customers will try to do when given a button to push. When engineers design a new product testing to the specification is very important. Even if your product passes all these tests most companies will initially send their product out to a select group of tolerant and friendly customers so that these folks can drop it, kick it, punch buttons randomly and generally abuse the product. This will almost certainly reveal more errors. You may have heard this process referred to as “Beta Testing<sup>55</sup>”. Which brings up the usual question of whether or not you would like to be a Beta Tester for a new type of parachute!

**Puzzle # 7 - Egg Timer** – Recipe for cooking eggs:

- 1) Place large egg in a small pan, cover with an inch of water and turn heat to high.

<sup>54</sup> Remember: every time a button is pressed a message is generated. If you get more than one message passing through your flows you might have problems.

<sup>55</sup> Yes, there is an “Alpha Testing” phase that is usually done by poor users inside the company. Back in the dark ages of the 1960 there was a company called HealthKit that had all sorts of complicated electronics kits that you could buy and build on your own, like radios and television. Some kits had hundreds of parts and hundreds of pages of step-by-step instructions. Once a kit was almost ready for release to the public HealthKit would give secretaries, janitors and mail room employees a kit to take home and build. That's Alpha testing

## Learning Kit Workbook (version 1.3)

- 2) When the water boils put the egg in the pan
- 3) Reduce heat slightly
- 4) Cook according to this chart:

Soft Boiled	4 minutes
Medium	5 minutes
Hard Boiled	6 minutes
Really Hard Boiled	8 minutes

Modify the Simple Kitchen Timer so that it can be an egg timer. Use what ever LED display pattern makes good sense to you to represent the cooking time. Do you need a STOP button? Be the Beta tester by cooking an egg with your timer.

**Puzzle # 8 = Ding!** Modify the egg timer or the kitchen timer so that it rings a bell when the time reaches zero.

### Flying with the OPTO Inputs

LEDs the size of sesame seeds on the Learning card are suitable for very limited displays, but wouldn't it be nice to step up to something more modern, like being able to display messages on a screen when your pet hamster pops open the cage door? Well, of course, there is a node for that, or more accurately, there is a pile of nodes for that. Rather than just running with the OPTO inputs now it is time to fly and build a system.

### Background

While Node-RED and the Learning card allow you to connect to the real world through various interfaces, you can also step beyond Inject and Debug nodes for your input and output. You do this by calling up a squad of "dashboard" nodes that will allow you to use your keyboard and monitor as input and output devices.

In node-RED a "dashboard" is a tab on your screen that you can use to interact with a flow. You do this including dashboard nodes that communicate with "widgets<sup>56</sup>" on your screen. A widget can be a gauge, a button, a slider, a chart and many other types of input and output elements. With widget nodes you can create a user interface directly from your flow.

When you create a user interface (UI) you will be defining several objects. At the top of the hierarchy are UI "tabs" that are exactly like the tab that your flow appears in. You may have as many UI tabs as you like, so if you have a small menagerie of pets whose cages you want to monitor you can have one tab for the lizards, one for the hamsters and another for the parakeets. Each UI tab may display several "groups". A group is a container that may hold widgets. Groups help you organize data so that if you change the size of your tab or display (for example, you want to look at your UI on a smartphone) the UI

---

<sup>56</sup> Widget – Actually an old word of unknown origin, but thought to have originated around 1930, possibly as a corruption of the word gadget. Usually "widget" is used to refer to a small device of unknown purpose. In the arena of computer science, it refers to a small displayable information that you can interact with, like a clock, button, dial etc.

## Learning Kit Workbook (version 1.3)

uses groups to resize and reorganize your display. At the bottom of the display hierarchy are the widgets. Typically, you will put widgets related to the same control and display function in a group. This way when you change the size of a screen related widgets stay together. This idea will become clear once you have built a user interface.

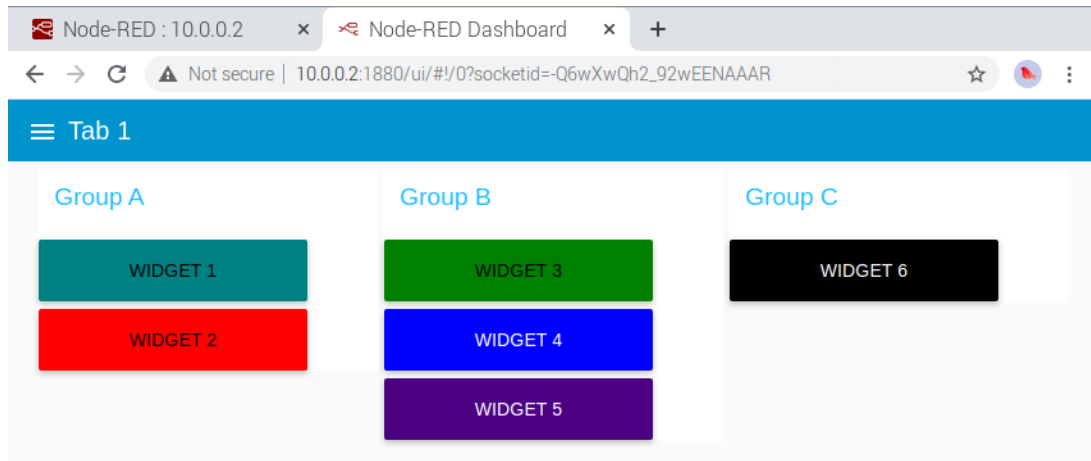


Figure 5-117: Dashboard Structure

Look at the figure above. It shows an example of a dashboard structure. There is a tab called “Tab 1”. Within that tab are three groups: Group A, Group B and Group C. Each group has one or more widgets as shown. Keep this structure in mind as you implement a dashboard. You can also have multiple tabs that you access by clicking on the menu icon to the left of Tab 1 above.

### Installing the Dashboard Nodes

Node-RED usually refers to the UI as the “dashboard” and to make use of the UI you will need to install the Node-RED dashboard nodes. Be careful here, there are a range of dashboard nodes from several sources; follow the instructions below to make sure you have the right nodes. Here is the recipe.

- Open the main menu
- Click “Manage Palette”
- Click “Install”
- Enter dashboard in the “search modules” box (red rectangle below). No need to hit enter.

## Learning Kit Workbook (version 1.3)

- You should see a list of available downloads as shown below.

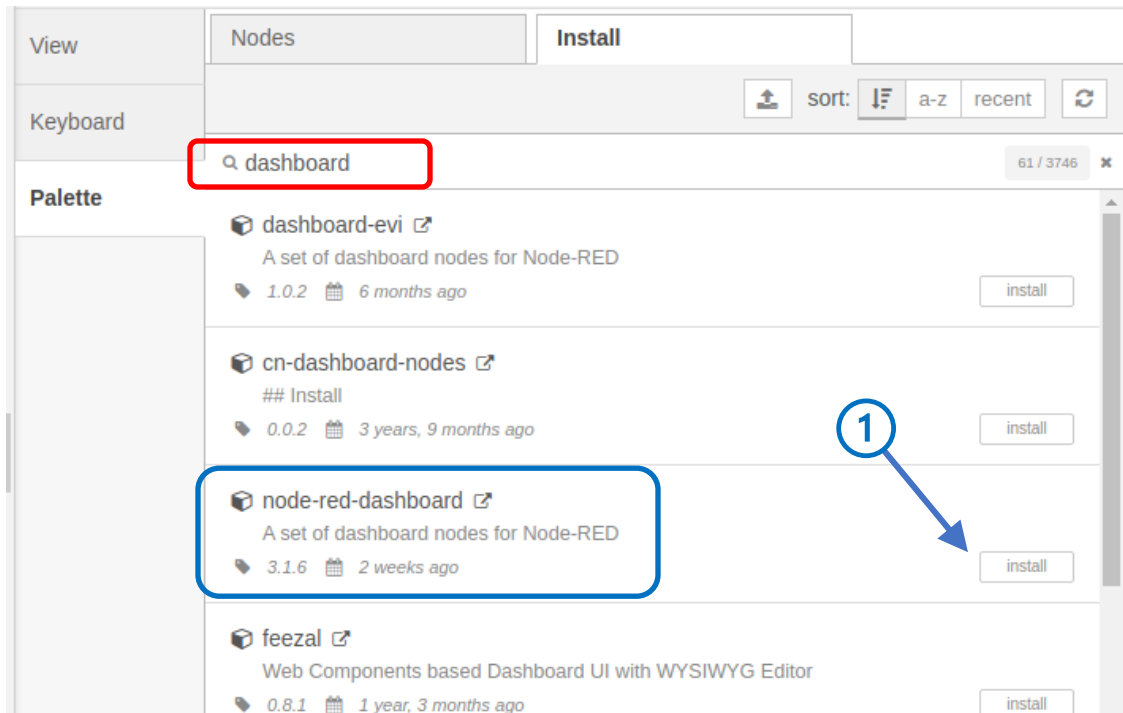


Figure 5-118: Manage Palette Install Window

- Find the entry labeled node-red-dashboard (blue box). Be careful here because there are other entries that include the word “dashboard”.
- Click “Install” ①.

A window will open like the one below with some information about the install.

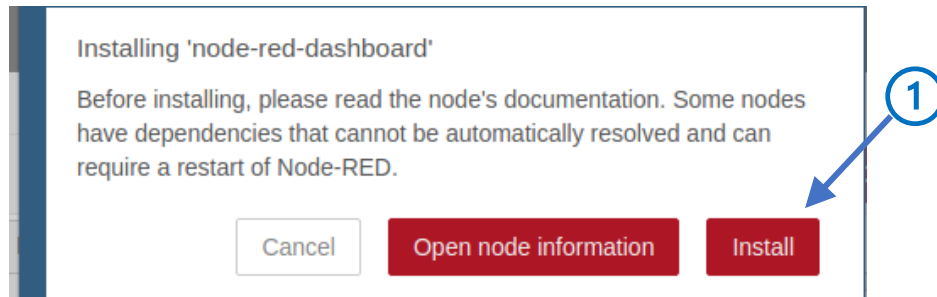


Figure 5-119: Install Dialog Window

- Click “Install” ①.



## Learning Kit Workbook (version 1.3)

Time will pass (⌚⌚⌚⌚⌚⌚⌚⌚) and finally your nodes will appear. Look for them in the Palette under “Dashboard” and you should see about a dozen or more new nodes like what is shown below.

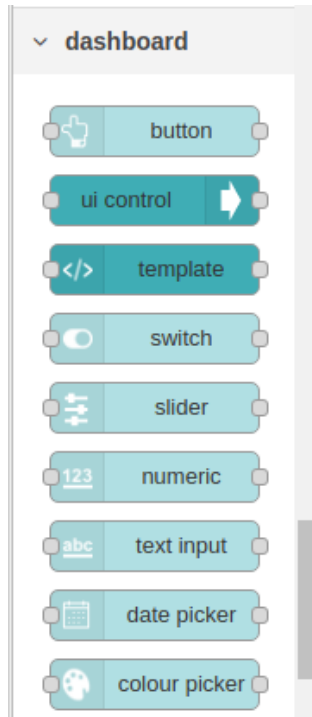


Figure 5-120: Dashboard Nodes in Palette

First: Build Something Simple – “Hello World” (of course)

You know what to do now: **Start Simple!** As always you should sooth the forces of the programming sub-space by issuing a Hello World message after which you are free to do what you want, but don't put humanity at risk by skipping this step.

Clear the decks, save your old work, open a new tab in your workspace and expunge all your old flows.

Your first dashboard flow will use a dashboard Button node and a Text node. Once your dashboard is established you will be able to click the button and print “Hello World!” on the screen. Simple and it is an adequate sacrifice to the gods of computation.

- Find the dashboard Button and Text nodes and drag them into your tab.
- Wire them up in the obvious way as shown below.

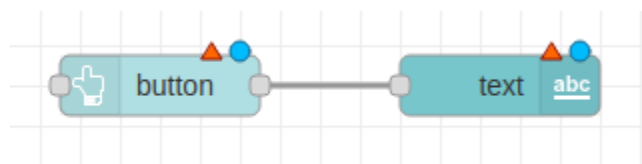


Figure 5-121: Simple Dashboard Flow for Hello

You are familiar with the blue dots that signify that a node has not yet been deployed. Search your memory... “What are those red triangles for?” Ah, yes! They indicate that the node must be configured before it can be deployed. In this case you cannot deploy your nodes until you define a dashboard tab

## Learning Kit Workbook (version 1.3)

and a group to hold the widgets associated with each node. In other words the widgets are all dressed up but they don't know where to go.

Each of the dashboard nodes in Figure 5-121 must be assigned to a group and the group must be assigned to a dashboard tab. To do this proceed as follows:

- Open the Edit window for the Button node (double click on it).

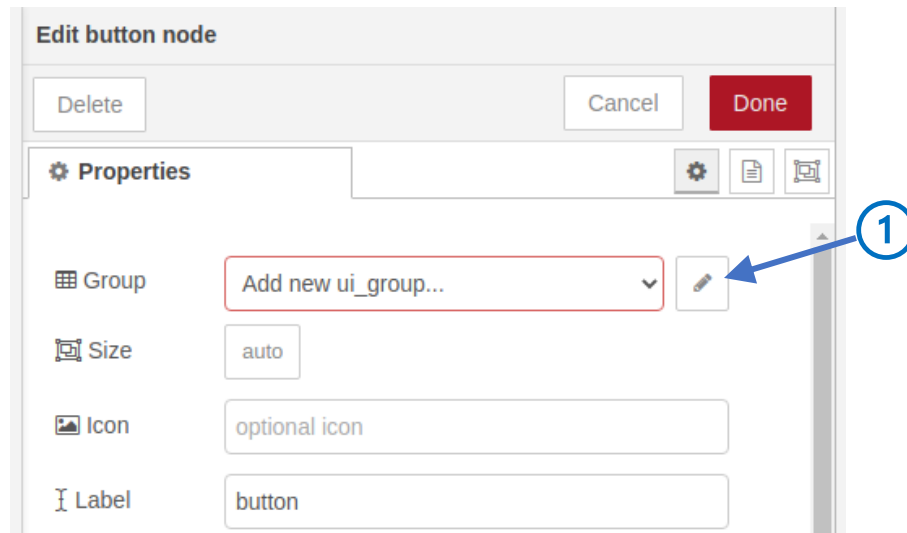


Figure 5-122: Dashboard Button Edit Window

The options for the dashboard Button node are quite extensive, but for now only concern yourself with the Group option.

- Click on the little pencil icon beside the Group dropdown ①.

## Learning Kit Workbook (version 1.3)

This will open a dialog to create a dashboard group node as below. This “node” will not be visible in your flow tab, but it is part of the configuration data associated with the flow, more on this in a moment.

You are not done yet because the group shown above needs to be assigned to a `ui_tab`, in other words

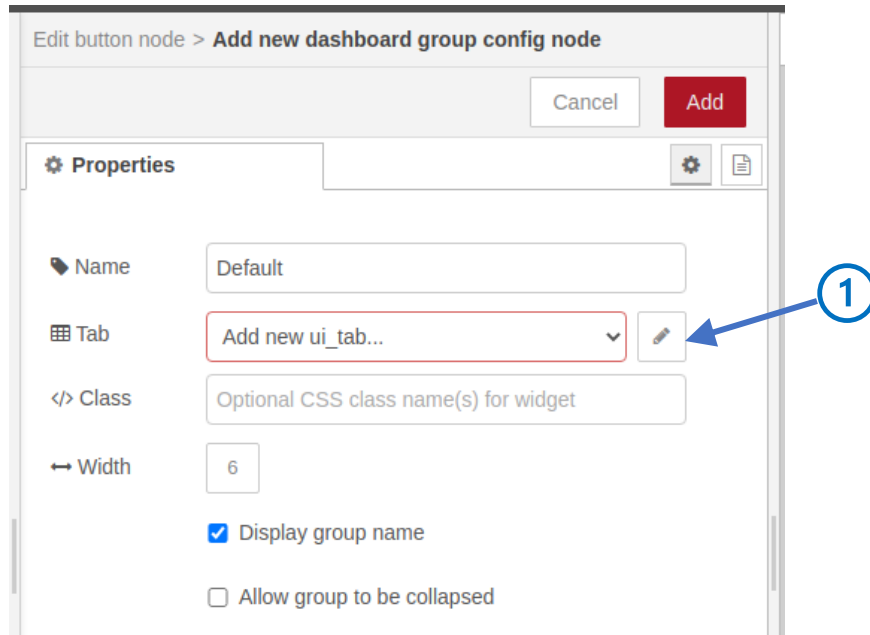


Figure 5-123: Add New Dashboard Group Window

the dashboard tab at the top level that is going to contain the group which contains the button<sup>57</sup>.

- Click the tiny pen next to the Tab dropdown ①. Note; once you define a tab and a group it will appear in the dropdown in the future, making the task of placing widgets easier. In fact, Node-RED try to reduce the work you must do by making a good choice for you.

Now you will see the dialog to create a new dashboard tab.

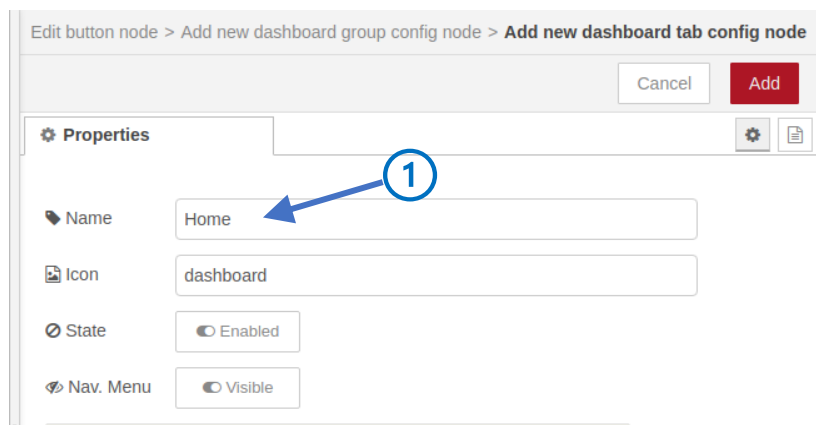


Figure 5-124: Add New Dashboard Tab Window

<sup>57</sup> “As I was going to Saint Ives, I met a man with seven wives...” or “As I was configuring Node-RED, I made a tab with seven groups and each group had seven widgets.”

## Learning Kit Workbook (version 1.3)

- Click in the Name box and give your dashboard tab a name, like “Simple Test” ①.

Your completed dashboard tab window will look like this:

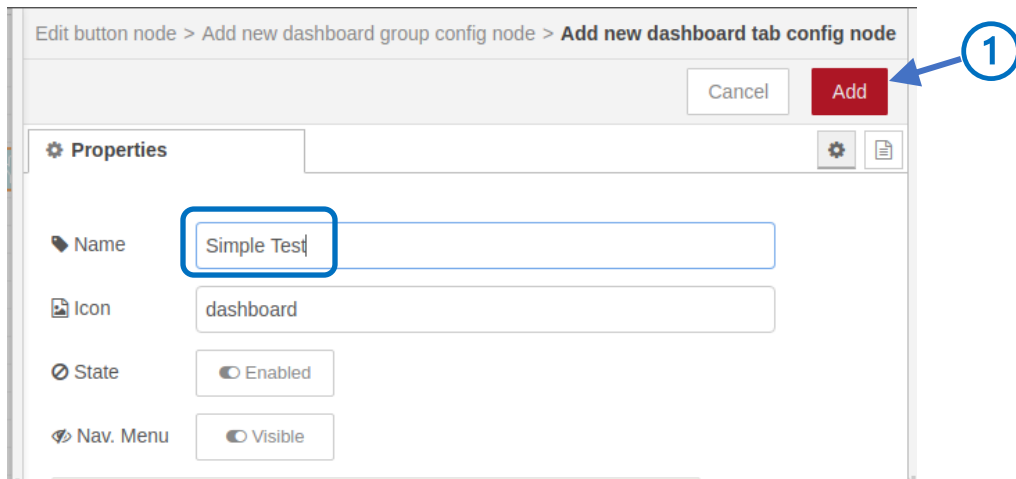


Figure 5-125: Completed Dashboard Window

- Click Add ① and you will be returned to the new dashboard group dialog of Figure 5-123.

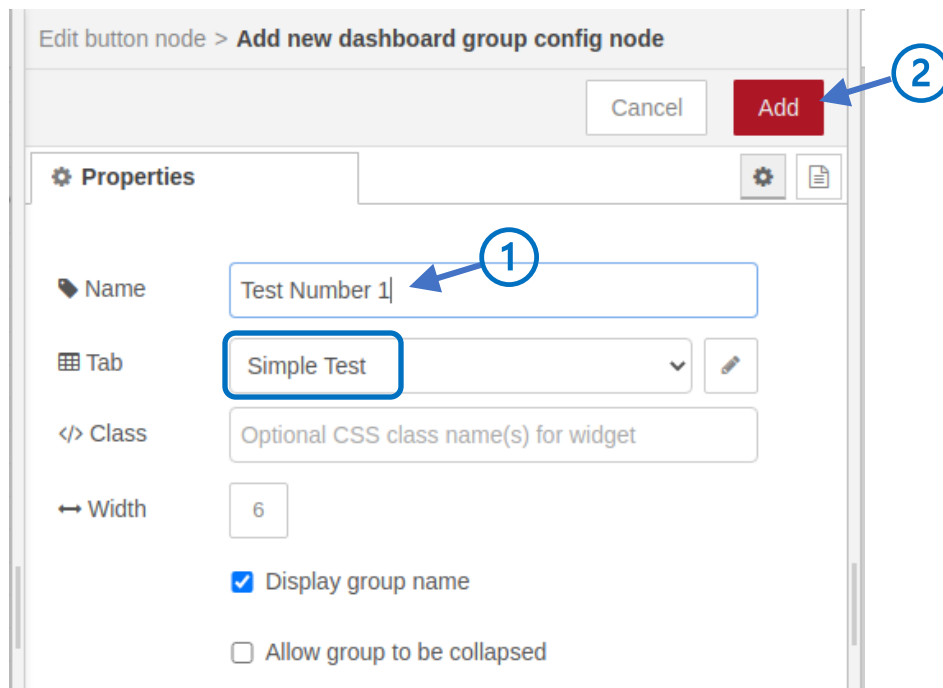


Figure 5-126: Completed Add New Dashboard Group Window

Notice that now the Tab box (blue square) has the name of the dashboard tab where the group holding the button node will be displayed, namely, the tab labeled “Simple Test”.

- Fill in the group Name box with the name of the group, as shown ①.

## Learning Kit Workbook (version 1.3)

- Click Add ②.

You will now see your dashboard Button Edit Window as below.

The screenshot shows the 'Edit button node' configuration window. At the top, there are buttons for 'Delete', 'Cancel', and 'Done'. Below is a 'Properties' section with various fields: 'Group' (a dropdown menu with a blue rectangle around it containing '[Simple Test] Test Number 1'), 'Size' (set to 'auto'), 'Icon' (text field with 'optional icon'), 'Label' (text field with 'button' and a blue arrow pointing to it from a circled '1'), 'Tooltip' (text field with 'optional tooltip'), 'Color' (text field with 'optional text/icon color'), 'Background' (text field with 'optional background color'), and 'When clicked, send:' (checkbox checked). Under 'When clicked, send:', there is a 'Payload' dropdown menu with a blue arrow pointing to it from a circled '2'.

Figure 5-127: Dashboard Button Configuration Second Step

Look in the Group box (blue rectangle) and you will see that the Button is located in the tab labeled “Simple Test” inside the group named “Test Number 1”. Now your Button widget knows where it will be displayed on the screen. Give your button a label to be displayed (like Show It!) on the screen by filling in the Label box.

- Fill in the Label box to your dashboard button a sensible label ①.
- Fill in the Payload box with the string Hello World! ②.

Now your configuration looks like the figure below (see blue rectangles showing the updates).

Figure 5-128: Configured Button Node

## Learning Kit Workbook (version 1.3)

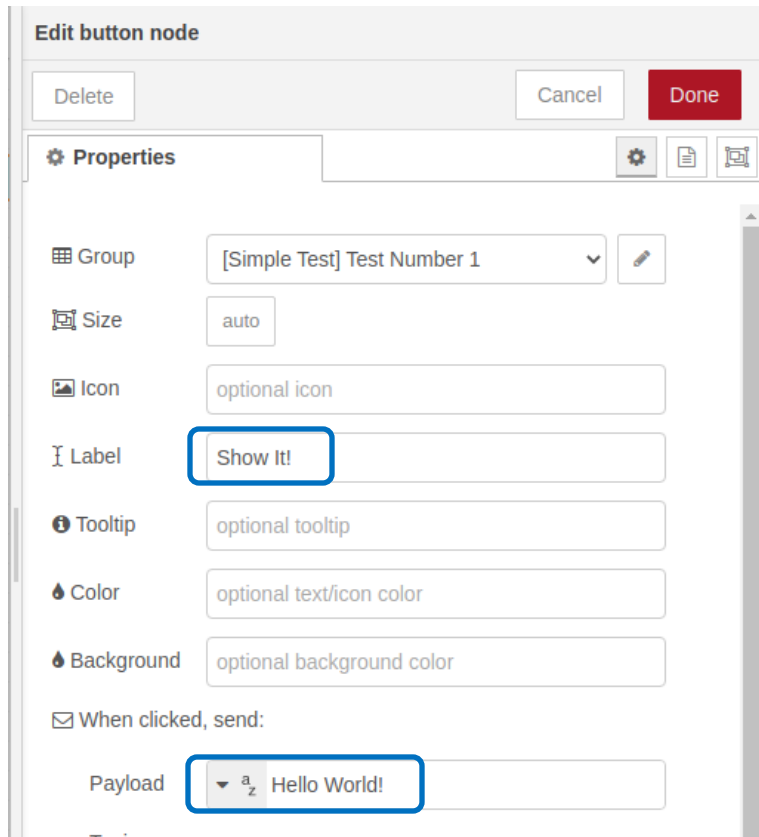


Figure 5-129: Completed Button Node Configuration

Click Done!

Now your flow will look like the figure below.

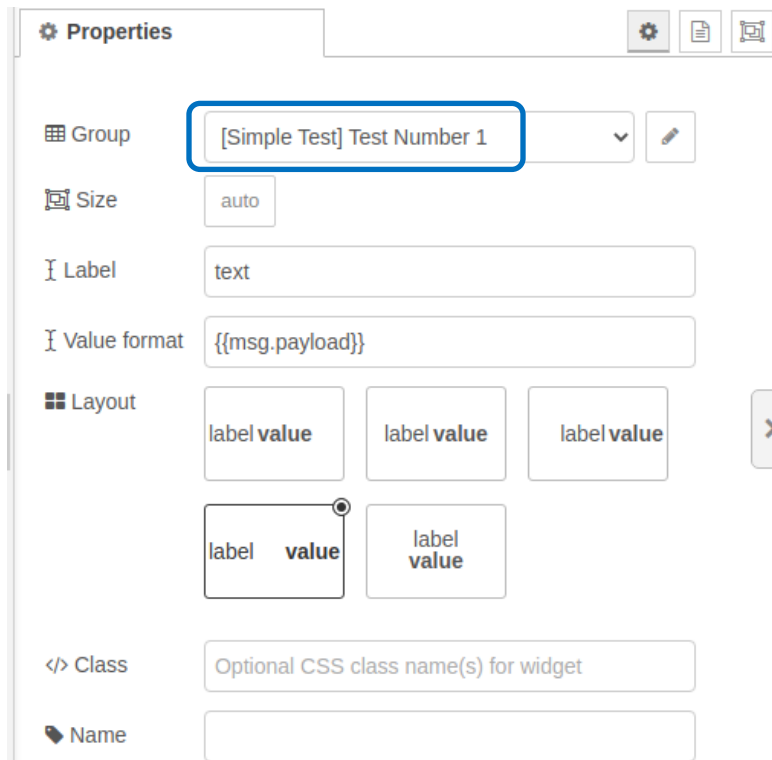


Figure 5-130: Flow After Configuration of Dashboard Button

Much better! The red triangle over the Button node (“Show It!”) is gone indicating that it is now configured for deployment. There are other aspects of the button you may configure, such as the color, the font, and a tool tip to show the function when you hover over the button.

## Learning Kit Workbook (version 1.3)

The remaining task is to configure the Text node, which is done in the same way as you did for the Button node. Open the Text node as in the figure below.



*Figure 5-131: Text Node Before Configuration*

Look at the box next to “Group” (blue rectangle). Node-RED has very helpfully placed your Text node in the same group (“Text Number 1”) and put that group on the same dashboard tab (“Simple Text”) as you specified for the dashboard Button node. That’s okay for now, but if you have other ideas, you can open the dropdown and specify a new group and/or tab just like you did for the Button node.

There are many options for the Text node. For now, just fix up the simple things as below.

## Learning Kit Workbook (version 1.3)

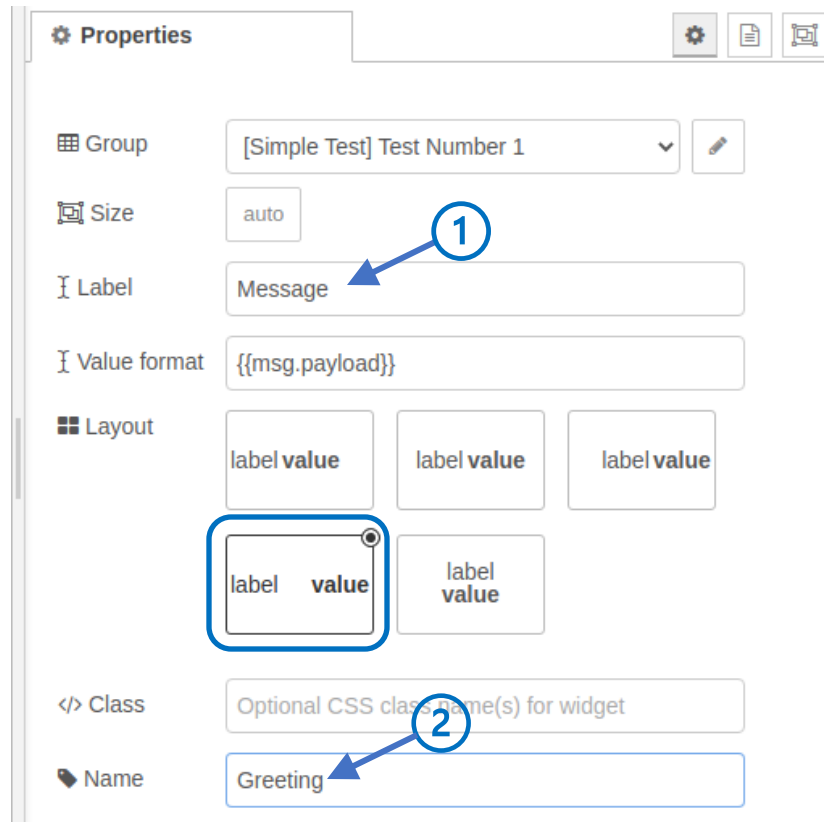


Figure 5-132: Text Node After Configuration

- Change the Label to “Message” ①. The Label is what will appear on your dashboard to identify the value you are going to display.
- Give your Button node a name, like “Greetings” ②, which will appear on the node in your flow.
- The Layout options gives you several ways to show the label and value in the text widget. The selected Layout is the one with the little circled dot in the upper right corner (check out the [blue rectangle](#)) in the figure above. This option places the label at the far left of the widget and the value (e.g., Hello World!) on the far right.
- Click Done!

When you look at your flow you should see this:

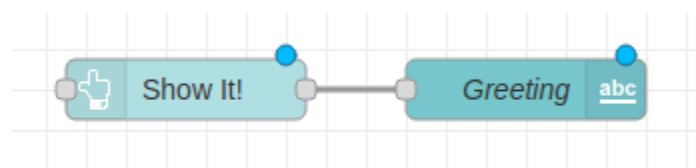


Figure 5-133: Test Flow Before Deployment

The annoying red triangles from Figure 5-121 are gone and your flow is ready for deployment. Before the big reveal take a moment to explore and see what has changed in your Node-RED environment. Dashboard.

First,



## Learning Kit Workbook (version 1.3)

- Click on the dropdown arrow ① to open the sidebar menu as in the figure below.

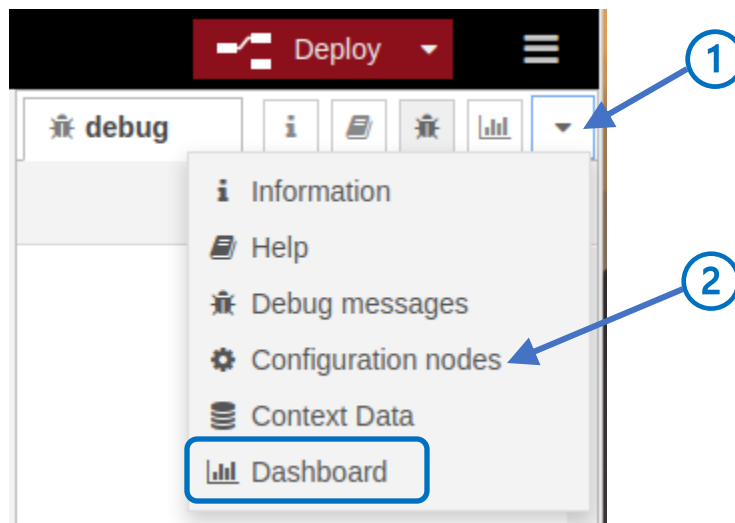


Figure 5-134: Sidebar Dropdown

Notice there is something new here: a selection for the Dashboard (blue rectangle). We will return to this in a moment. Next...

- Click on “Configuration nodes” in the sidebar menu ②, which will open a sidebar window showing the defined configuration nodes as in the figure below.

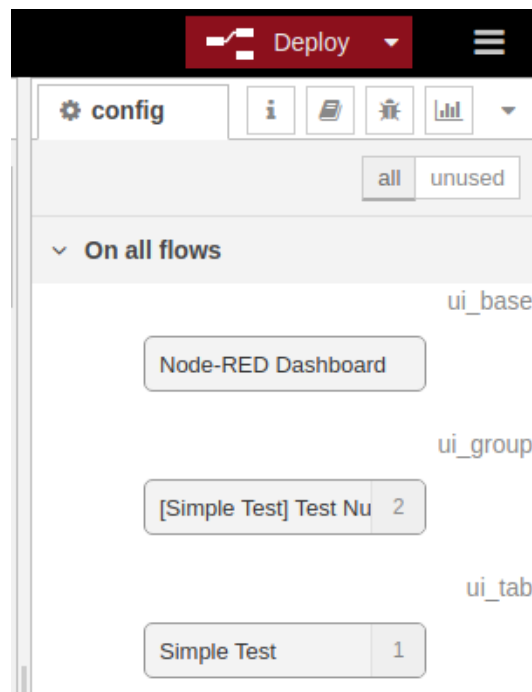
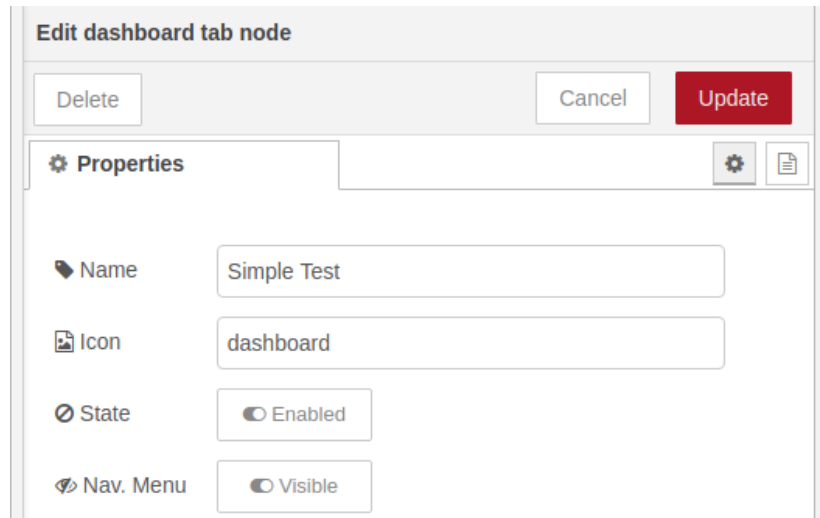


Figure 5-135: Configuration Nodes for Simple Test

Voila! You now see three nodes that were not there before. These nodes relate to the dashboard, the Simple Test dashboard tab and the Test Number 1 group within that tab that is going to contain your

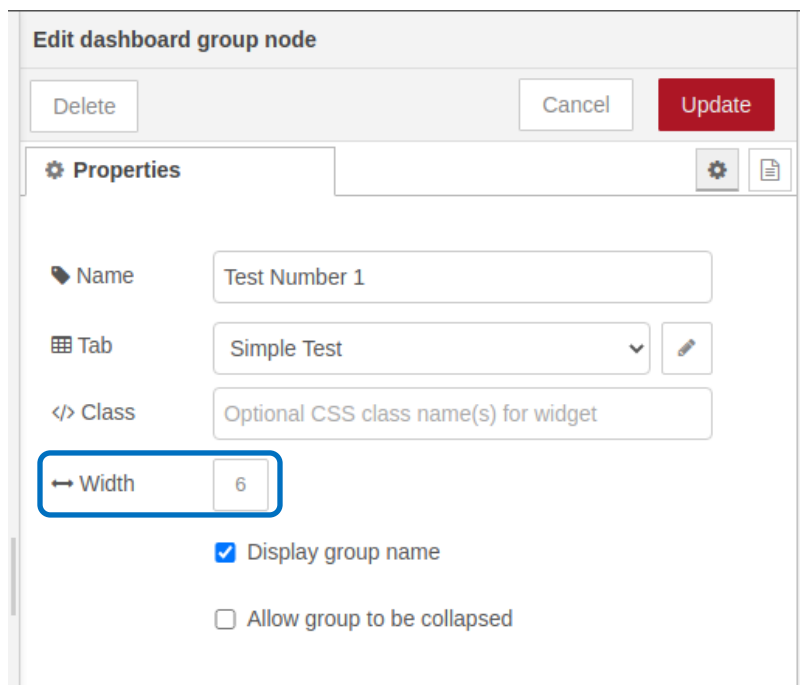
## Learning Kit Workbook (version 1.3)

Button and Text widgets. Double Click on the *ui\_group* or *ui\_tab* and Node-RED will open a configuration window to show you how the group and tab are configured as in the two figures below.



The screenshot shows the 'Edit dashboard tab node' configuration window. At the top, there are three buttons: 'Delete', 'Cancel', and 'Update'. Below these is a 'Properties' section with a gear icon and a document icon. The properties are: Name (Simple Test), Icon (dashboard), State (Enabled), and Nav. Menu (Visible).

Figure 5-136: Dashboard Tab Edit Window (*ui\_tab*)



The screenshot shows the 'Edit dashboard group node' configuration window. At the top, there are three buttons: 'Delete', 'Cancel', and 'Update'. Below these is a 'Properties' section with a gear icon and a document icon. The properties are: Name (Test Number 1), Tab (Simple Test), Class (Optional CSS class name(s) for widget), Width (6), Display group name (checked), and Allow group to be collapsed (unchecked). The 'Width' field is highlighted with a blue rectangle.

Figure 5-137: Dashboard Group Edit Window (*ui\_group*)

In the group configuration window (Figure 5-137 above) there is an option to adjust the width of the group (blue rectangle). This gives you some control over how your dashboard group is shown.

The big moment is at hand. It is time to **Deploy** your test flow!

## Learning Kit Workbook (version 1.3)

Wait a minute. Where's my dashboard??? I was promised a dashboard and I want to see it now! Relax. Open the side bar menu and click on the "Dashboard" option (see Figure 5-134). When you do this, you will see a sidebar window showing various dashboard options as in the figure below.

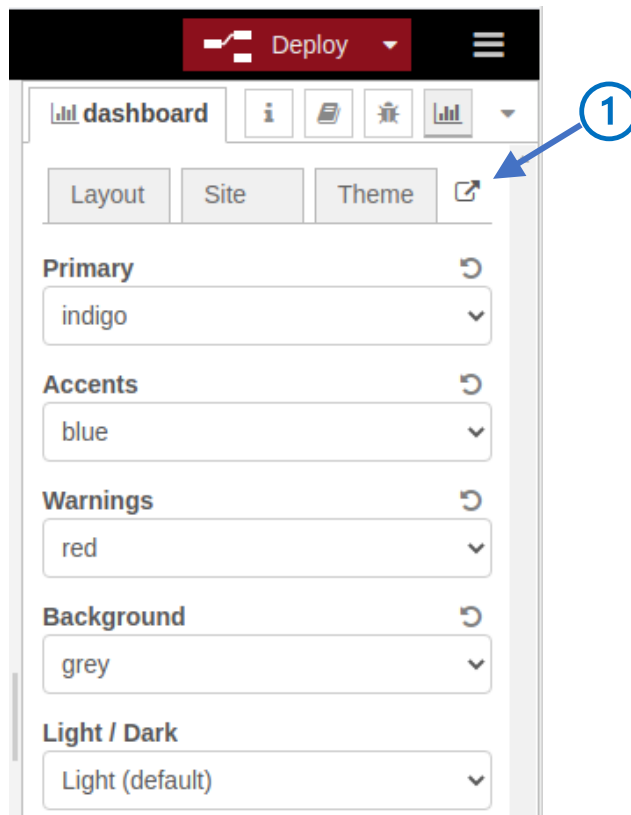


Figure 5-138: Dashboard Sidebar

In the upper right corner is a small icon indicating that will put your dashboard on the screen ①. Note: You may need to make your sidebar window wider to see the icon because in some versions of Node-RED the icon may overlap with the tabs in the sidebar. Also notice that you can adjust the color of various dashboard characteristics.

- Click the expand icon ① in above figure.

Your dashboard will appear as another tab in the Node-RED window as shown below.

## Learning Kit Workbook (version 1.3)

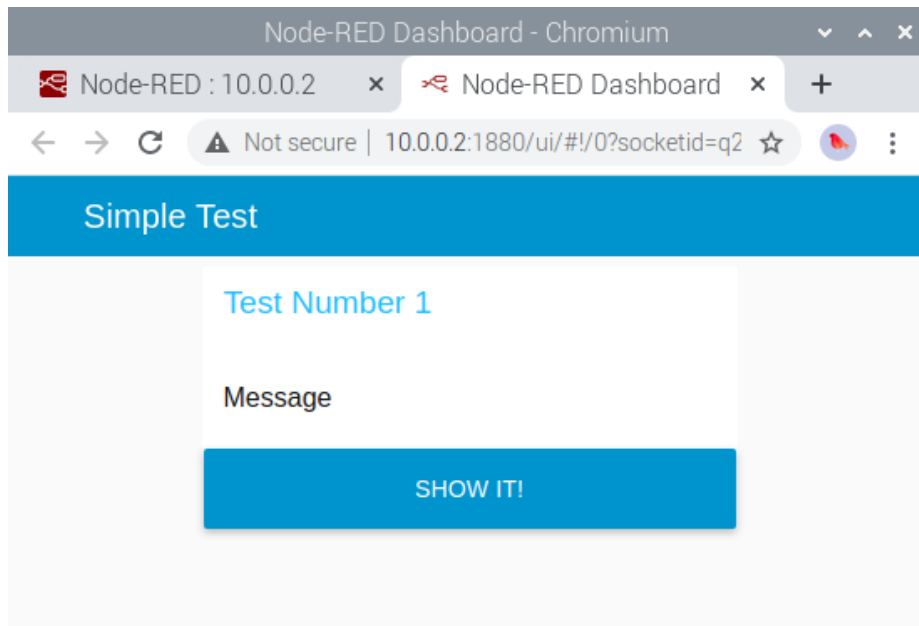


Figure 5-139: Node-RED Dashboard Tab

Within the dashboard tab is the “Simple Test” tab you created above (Figure 5-126) which in turn contains the group labeled “Test Number 1”. Inside that group is the button, “SHOW IT!” and the text output area, “Message”.

- Click the “SHOW IT!” button.

And if you have covered all the bases the message text box will show Hello World! as below.

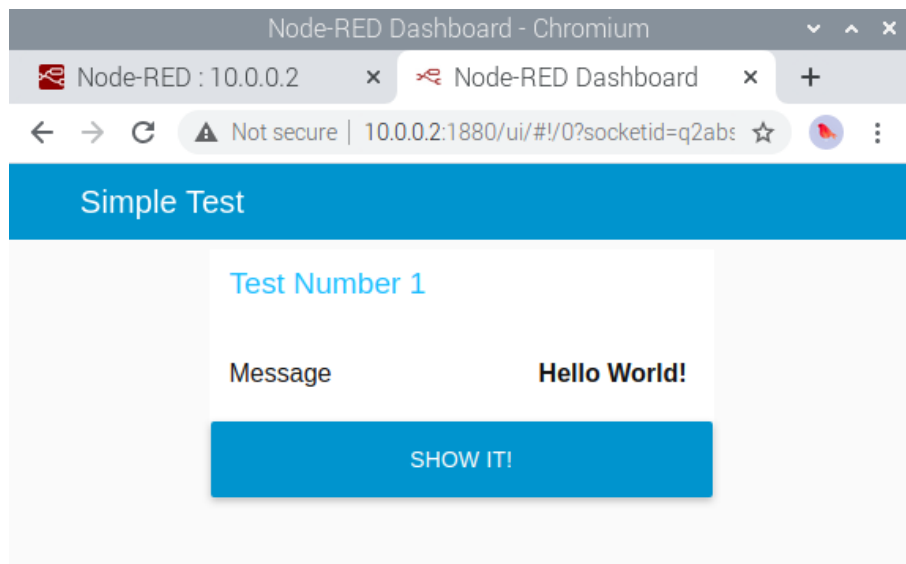


Figure 5-140: Dashboard Test Flow with Hello World!

If you see the message, then you have begun your conquest of the Node-RED Dashboard. If not, then the first step is to check the configuration step by step. Then pull out the Inject and Debug nodes and see if you can locate the problem.

## Learning Kit Workbook (version 1.3)

### Second: Build Something More Complicated

If you look at your Hello World flow it will look very much like other flows you have built with Inject and Debug nodes, and with OPTO and LED nodes. Time to enhance the Hello World a bit at a time.

Start by adding another dashboard button to hide the Hello World message as shown below.

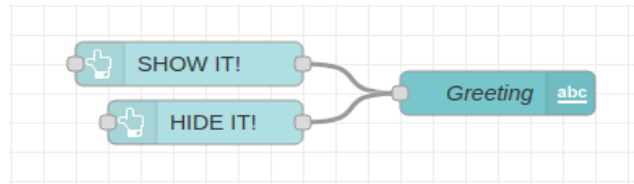


Figure 5-141: Enhanced Dashboard Flow

- Copy the “SHOW IT!” Button and paste a copy of it into the flow.
- Wire the copied button to the Text node.
- Open the copy and change the Label box to “HIDE IT!” ①.
- Change the Payload to be a string with a single space ②. The single space is important because if the string has no characters the button node will send the node ID rather than the empty string<sup>58</sup>. When the string with a single space is sent it will erase the current message.
- The completed configuration of the “HIDE IT!” Node will look like the figure below.

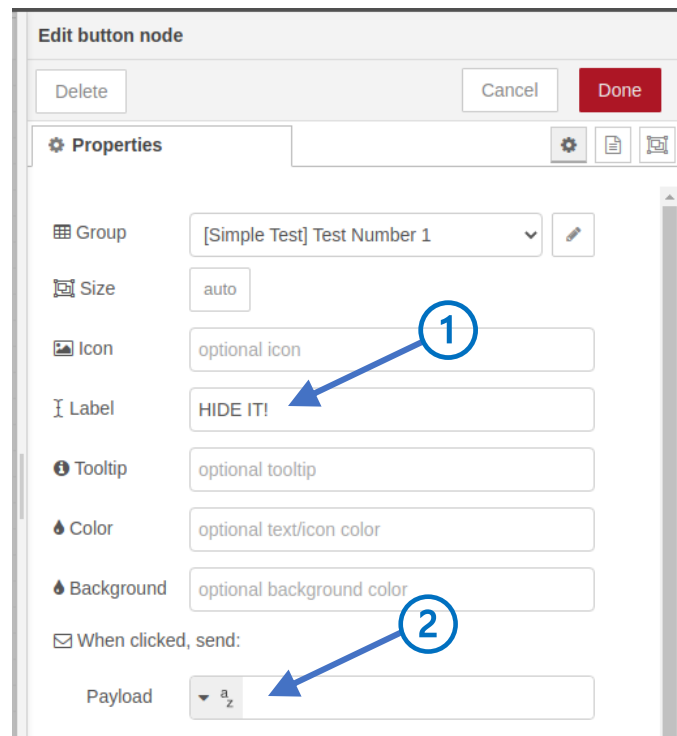


Figure 5-142: HIDE IT! Button Node Configuration

**Deploy** and go to the dashboard. Now your dashboard will have two buttons as shown below.

<sup>58</sup> This is probably an bug in Node-RED.

## Learning Kit Workbook (version 1.3)

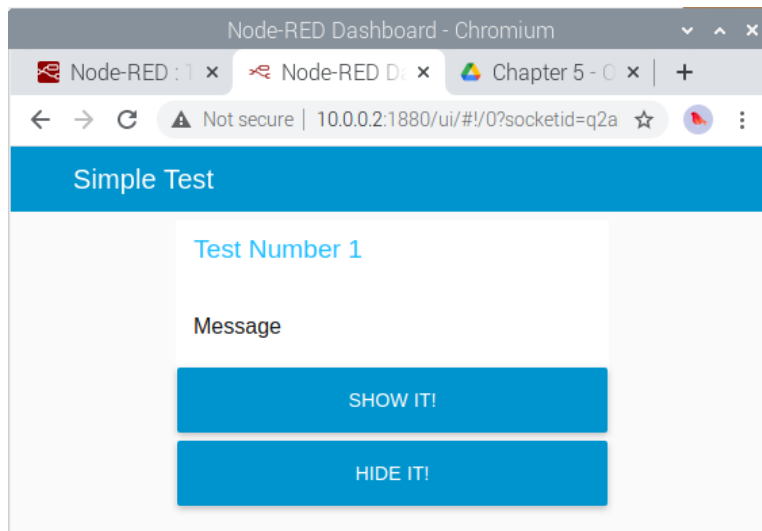


Figure 5-143: Enhanced Dashboard

Click each button and see how your dashboard behaves. When you click “SHOW IT!” the Hello World message should appear and when you click “HIDE IT!” the message should vanish.

**Puzzle # 9 - Explore it!** The dashboard has several options you can use to change the appearance of the button. Read the help file for the dashboard button node and see if you can make changes to the appearance of the buttons.

- **Color** – Go back to Figure 5-128 and Figure 5-142. You can specify the color of your button text using the color box. Type in a common color name<sup>59</sup> and see what the button looks like. Remember: you must DEPLOY before the change will take effect.
- **Background** – Try different colors and see what your button looks like.
- **Size** – The box normally says “Auto”, which means that the button will be one unit high and as long as the width of the group it is within. Change the size, DEPLOY and see what the effect is.
- **Icon** – You may use icons from the [MaterialDesignIcons.com](https://materialdesignicons.com) by simply typing the name of the icon.

### Third: Showing OPTO Status

How about showing the status of an OPTO input on the dashboard? Suppose you wanted to monitor the status of the cage door on your pet’s cage? How about this simple specification?

- Group Name: “Cage Status”.
- Label: “Cage Door”.
- When OPTO IN 1 is open show the text “OPEN”<sup>60</sup>.
- When OPTO IN 1 is closed show the text “CLOSED”.
- On deployment set text depending upon the status of OPTO IN 1.

---

<sup>59</sup> Color Names – There are standardized color names for about 140 colors. See [HTML Color Names](https://www.w3.org/TR/html5/11-color.html)

<sup>60</sup> This way if your pet is exceptionally smart and gets his paws on a pair of wire cutters, cutting the wire will open the circuit and set off the alarm.

## Learning Kit Workbook (version 1.3)

Here is one possible flow:

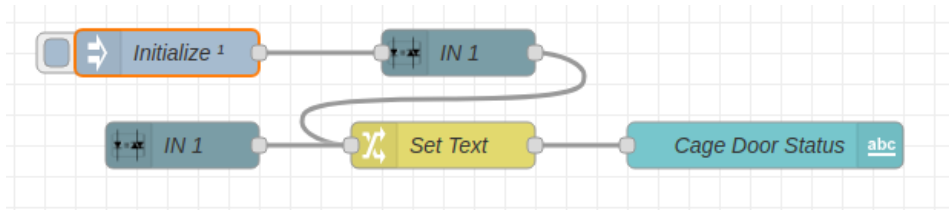


Figure 5-145: Cage Door Status Flow

The top two nodes read the status of IN 1 upon deployment. The “Set Text” Change node converts the Boolean *true* and *false* values from IN 1 into the text message “CLOSED” and “OPEN”, respectively. Here is the configuration of the “Set Text” Change Node.

**Edit change node**

Delete Cancel Done

**Properties**

Name Set Text

**Rules**

Change	Search for	Replace with
msg. payload	true	CLOSED
msg. payload	false	OPEN

Figure 5-144: "Set Text" Change Node Configuration

To make the display a little easier to read you might want to make the width of the Text widget smaller, say 1x4, using the configuration below.

## Learning Kit Workbook (version 1.3)

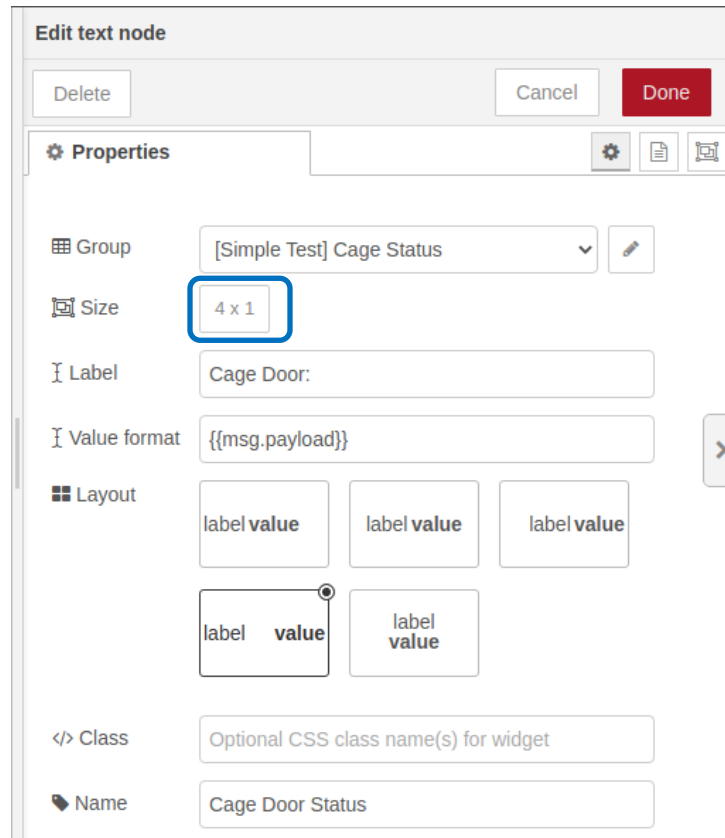


Figure 5-146: Text Widget Configuration

By clicking on “Size” box (blue rectangle) in the text node configuration window you can adjust the size of your text display widget.

### Deploy and Test!!

Fourth: Check Four Cages

Extend the flow above to show the status of the cage door for Alfalfa, Bobo, Chewy and Drowsy. This is easy and your flow will look more or less like the figure below.



## Learning Kit Workbook (version 1.3)

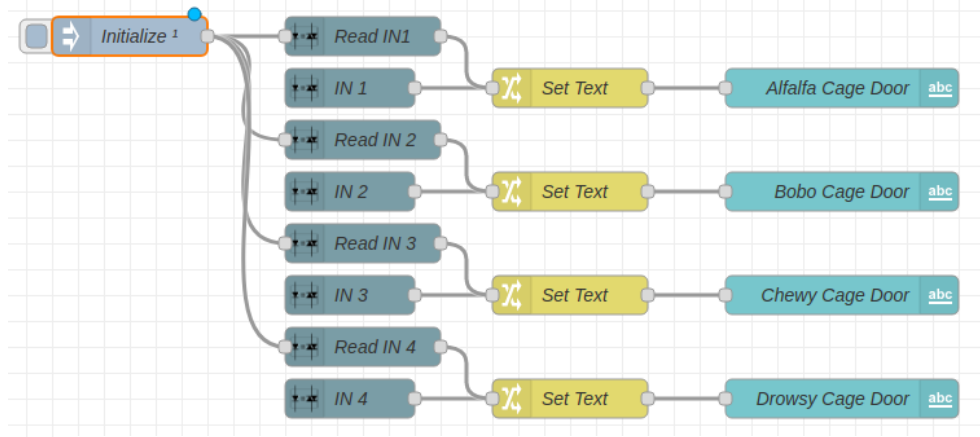


Figure 5-147: Flow for Four Cages

**Deploy** this flow and look at the arrangement of the Text node widgets in the group. You should see something like this.

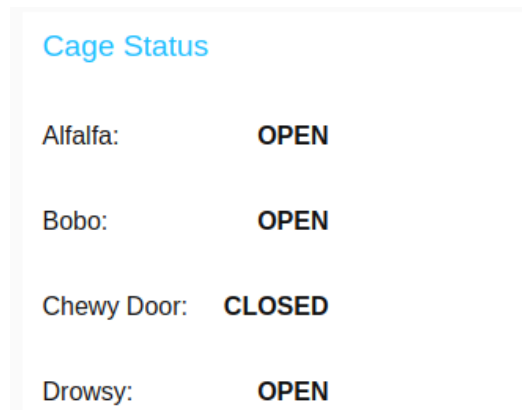


Figure 5-148: Status Display for Four Cages

Node-RED always tries to do right by you. In this case the widgets area arranged from top to bottom based on how they are arranged in the flow. Suppose you want a different arrangement. Easy. Open the Dashboard sidebar, select the Layout tab and then click on the Cage Status group. This will open a dropdown showing the arrangement of the widgets within the Cage Status group as shown below.

## Learning Kit Workbook (version 1.3)

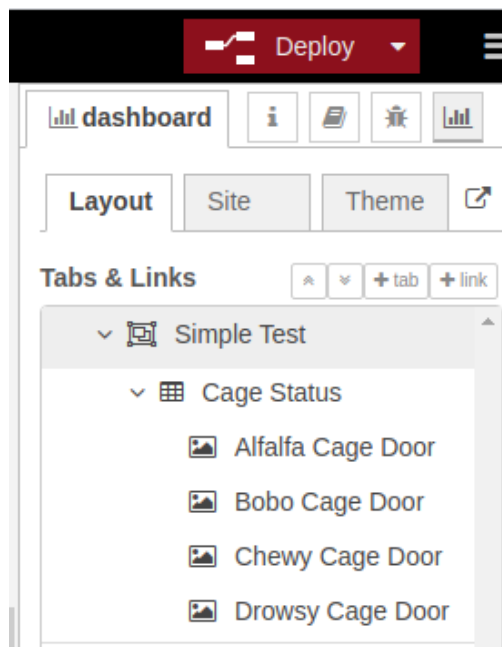


Figure 5-149: Ordering of Text Widgets in Group

If you don't like the ordering you can click on a widget, hold down the mouse key and move the widget to another position on the list. Try rearranging your list. *Deploy!* and check your dashboard. You should now have a new ordering.

**Puzzle # 10 - Color Coding.** Look around on the Internet and see if you can change the color, font or size of the text in the cage door OPEN/CLOSED text. Hint: you can include HTML in your payload text and this will be passed through to the dashboard for display. So, you might adjust the text you are generating in the Set Text Change node to a form like this:

```
<font color = green>CLOSED</font> for the CLOSED text and  
<font color = red>OPEN</font> for the OPEN text.
```

**Puzzle # 11 - Warning!!** Review the code for flashing the LEDs from Chapter 4. Using this and can you build up a flow that cause the dashboard text to flash (half second on half second off) when a cage door is open?

**Puzzle # 12 - Two Hamster Monitoring.** Assume that you have only two hamsters named Scylla and Charybdis<sup>61</sup>. Each in a separate cage. On each cage there are two switches: one to monitor whether the cage door is open or closed, and the other to tell you whether the level in the water bottle is okay or low. Build a dashboard with two groups, one for each cage. Each group will have two widgets, one to show the cage door status and another to show the water level status.

Next Up – Relays: controlling real voltage and current.

---

<sup>61</sup> Look it up! That's why we have an Internet.

# Learning Kit Workbook (version 1.3)

## Chapter 6 – Relays

What you will learn:

- Relay interface and node
- Simple User Interface Widgets

### Introduction

Your Learning Card contains two SPDT<sup>62</sup> relays. Using a relay, you may control external devices requiring higher voltages and currents than you can control directly from the Raspberry Pi. For example, you can control the power to a small motor, a modest sized light or a buzzer.

But first a word about **SAFETY**.

- **Take Responsibility** – You are the Range Safety Officer for your project. Our recommendation is that you do not use the relay outputs to control more than 24 Volts AC/DC. In general, but not always, voltage up to this level are safe to work with. Do not use these relays to control devices that plug into wall voltages. Electricity from wall sockets can easily be lethal.
- **Know What You are Controlling** – If you are going to use the relays to control motors, heaters or other similar equipment be sure you work in a safe manner. Make sure no one is going to get hurt when you energize some piece of equipment, especially if it is where you can see it.
- **Relays Have Limits** – Do not operate relays beyond their rated limits. If you, do you risk damage to the relay. Alternatively, an overstressed relay may stick in the on or off position meaning that you will no longer be able to control it.
- **Protect Your Relay** – If you connect a DC voltage controlled by your relay to an inductive load (like a motor, a buzzer or an electric bell you should add a protective device, like a diode across the contacts. This will prevent high voltage spikes generated when the contacts open from causing arcs across the relay contacts that will significantly shorten the life of the relay.

### Relays – What Are They?

A relay is simple a switch that may be operated by current flowing through a coil. In school you may have experimented with making an electromagnetic by wrapping a few dozen turns of wire around a nail as in the figure below from [Great Basin Observatory](#). With such a contraption you can pick up a few paper clips. If you put hundreds of turns on the nail you would be able to pick up more than a paper clip, you might pick up big nails, screws, or small iron bars. Now if you were an inventive sort, you might think

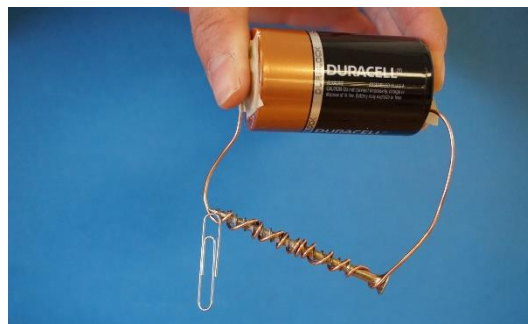


Figure 6-1: Simple Electromagnet

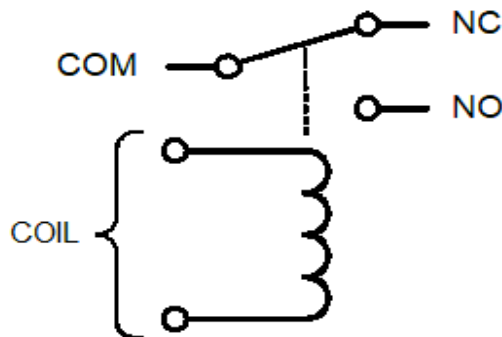
---

<sup>62</sup> SPDT – Single Pole Double Throw – There is a very old taxonomy of switches which is usually applied to relays because they are basically mechanically controlled switches. The form of the designation is “<number> Pole <number> Throw”. The term “pole” refers to the number of independent circuits the relay supports. “Throw” refers to the number of ways the pole can be switched to contact. A simple relay is SPST or Single Pole Single Throw. In other words, it is just a simple off-on switch. The Learning card relays are SPST or Single Pole Double Throw meaning that a single circuit may be connected in two ways.

## Learning Kit Workbook (version 1.3)

that you could connect this to a switch so that you could turn the switch on and off. Congratulations, you have invented the relay.

If you have some exposure to electronics, the schematic symbol for the relay should look obvious.



*Figure 6-2 Schematic for SPDT Relay*

Here you have a coil (i.e., the nail with some turns of wire) and a SPDT switch (i.e., the paper clip). The dashed line indicates that when current flows through the coil the switch will change position. Engineers refer to the switch part of the relay as “contacts” because when parts are in contact current flows. Unless otherwise stated a schematic shows the contact position of a relay in its “deenergized” position, that is the contact position when no current is flowing in the coil.

A set of contacts on a relay is usually labeled with the codes COM, NC and NO, which stand for “Common”, “Normally Closed” and “Normally Open”, respectively. Look carefully at the diagram and you will see that the contacts are electrically separated from the coil. This is important because it means that a small current in the coil can control the switching of much larger currents and voltages.

The contacts of the relay in **Error! Reference source not found.** are a bit like the railroad switch yard from Chapter [redacted]. In a circuit the contacts are going to control the flow of current. When the relay is deenergized current will flow from the COM (common) terminal to the NC (normally closed) terminal. This gives rise to the terminology, when the relay is deenergized it “normal” state<sup>63</sup>, then the NC or “normally closed” contact forms the conductive path from the COM or common terminal. When the relay is “energized” the contact switches to the NO or “normally open” position.

On your Learning card each of the two relays is connected to a three-pin connector. The figure below is a simplified version of the relay circuitry on the Learning card. If you want to know the exact circuitry you can refer to the card schematic.

---

<sup>63</sup> When it is sitting on the shelf!

## Learning Kit Workbook (version 1.3)

In the figure above the relay coil is coupled by a transistor to ground. When the CPU sets the base of the transistor to a high voltage the transistor conducts, current flows through the relay coil and the common contact is switched to the NO position. When the voltage at the base of the transistor is low the transistor does not conduct, and the relay is in the NC position as shown. The LED in parallel with the relay shows you when the relay is energized. The diode across the relay coil is there to protect the circuitry of the Raspberry Pi<sup>64</sup>.

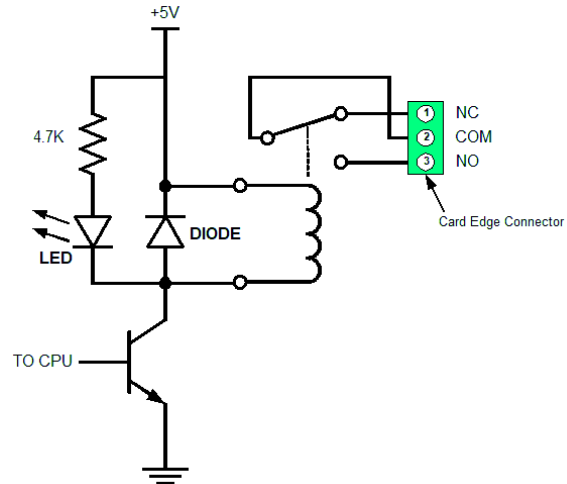


Figure 6-3: Learning Card Schematic for Relay

### Connecting to the Relays

Each of the two relays on the Learning card is connected to a three-pin female plug. Below is a top view of the learning card.

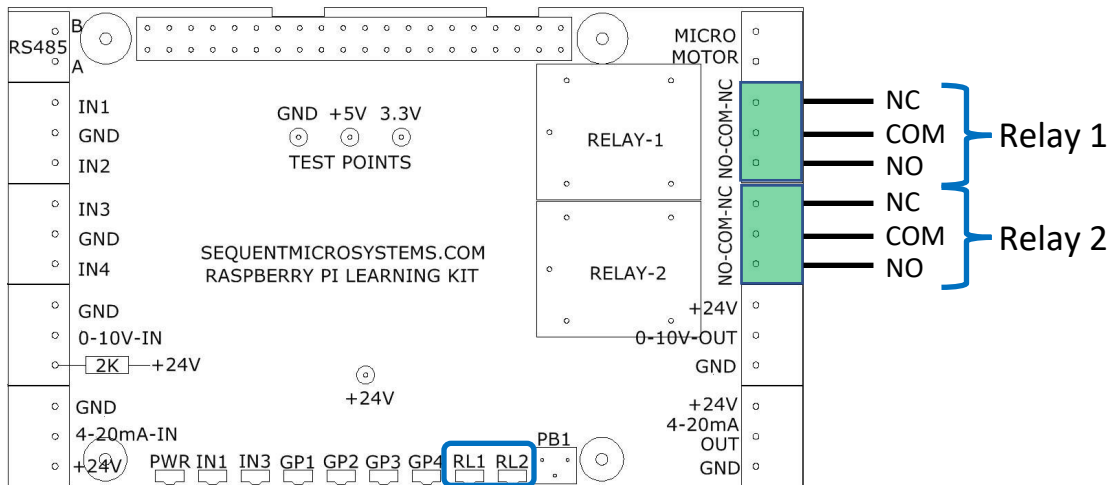


Figure 6-4: Top View of Learning Card

The highlighted plugs are where you connect your circuit to the relay contacts. Look at the bottom edge of the Learning Cards and you will see two LEDs labeled RL1 and RL2 (blue rectangle). These indicate the state of each relay. When the LED is lit it indicates that the relay is energized, and that the COM contact is connected to the NO contact. If the LED is dark, then the relay is deenergized and the COM contact is connected to the NC contact<sup>65</sup>. These LEDs are handy for debugging your program because they show you directly the state of a relay.

<sup>64</sup> The relay coil is an inductor. Whenever current is interrupted suddenly in an inductor a high voltage spike is generated. The diode across the relay coil prevents the high voltage from propagating to the rest of the circuitry and protects the Learning card and Raspberry Pi from damage.

<sup>65</sup> Relays are switches. If you are going to drive a logic circuit (like an OPTO input) with one of the relays you must “debounce” the input just as you would for a switch. See Chapter 5 for details.

## Learning Kit Workbook (version 1.3)

### Crawling with Relays

When it comes to controlling relays, the techniques are very similar to what you used to control LEDs in Chapter 4.

By now you know what is coming next: START SIMPLE! Save your old work, open a new flow tab in the workspace and delete all the other open flows. Bring in a relay node and drop it in your workspace.



Figure 6-5: LKit Relay Node

The first thing to notice is that the LKit Relay node is an output node, it has an input port, but no output port. The *msg.payload* received indicates the state of the relay, e.g., ACTIVATED or DEACTIVATED. Double click on the node and you will open the configuration window as shown below.

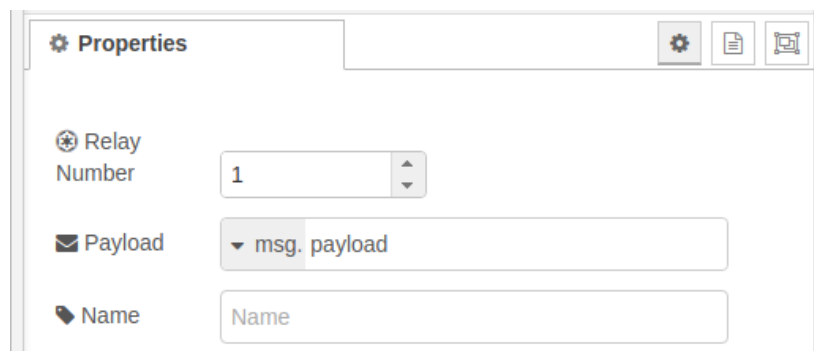


Figure 6-6: Relay Node Edit Window

The relay node has two ways to specify the relay to be controlled: explicitly using the Relay Number dropdown menu, or by the *msg.relay* property value, which you will learn about shortly. Figure 6-6, above, shows the default configuration where Relay 1 is selected.

The *msg.payload* value controls the state of the selected relay as follows

ACTIVATED (COM connects to NO):

- Numeric: 1
- String: "on", "1"
- Boolean: *true*

DEACTIVATED (COM connects to NC):

- Numeric: 0
- String: "off", "0"
- Boolean: *false*

Once the LKit Relay node receives a *msg.payload* it sets the relay according to the value and the relay remains in that state as long as the Raspberry Pi system is powered. Values other than those shown above will result in an error message and no change to the state of the relay.

## Learning Kit Workbook (version 1.3)

### Explicit Selection of Relay

Let's try out Relay 1! Below is a simple test set up because you always want to make your life simple by starting simple.

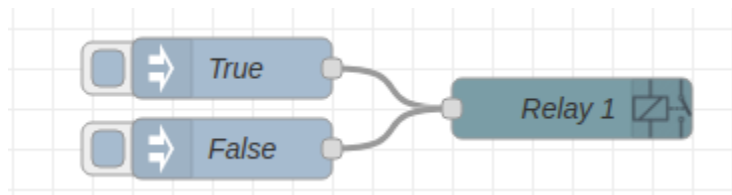


Figure 6-7: Basic Relay Control Test

In this case the msg.payload will be either Boolean *true* or *false*, however, you could just as easily have used one of the other values from above. You know how to program the “True” and “False” Inject nodes and if you are unsure about how to program the Relay node look at the figure below.

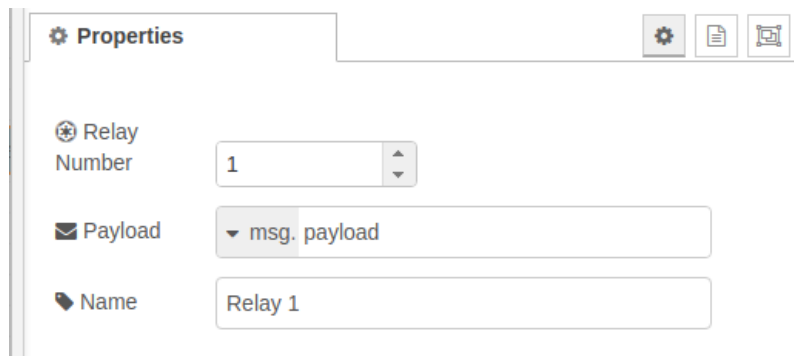


Figure 6-8: Relay Node Configuration for Simple Test

Deploy this lovely flow! Click on the two inject nodes and see if the relay turns on and off, which you can tell by looking at the RL1 LED on the edge of the Learning card. Try changing the Inject nodes to use other values, like numeric 1 and 0, string “on” and “off”. Try some values that should not work.

It is easy to control the relay from either the pushbutton or the OPTO inputs. Just remember that when you push the button you get a message with a numeric 0, but when you close an OPTO input you get a Boolean *true* or *false*. Both messages work fine with the Relay node, but you may wish to invert the message from the button node because you probably want to energize your relay when you push the button.

Here is an example flow for the pushbutton with nodes to invert the payload value.

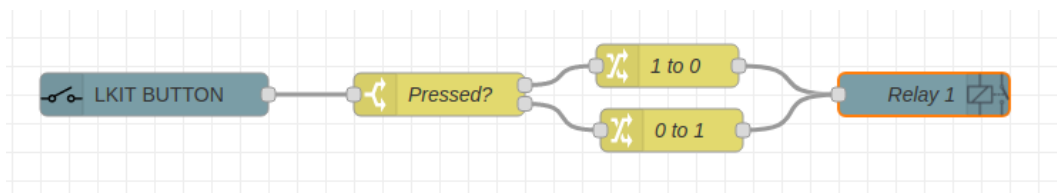


Figure 6-9: Button to Relay Flow

If you are confused about how this flow works, go back to [\[redacted\]](#) and review the pushbutton to LED flow.

## Learning Kit Workbook (version 1.3)

Here is a flow where Relay 1 is controlled by OPTO input 1 and is about as simple as a flow can be.

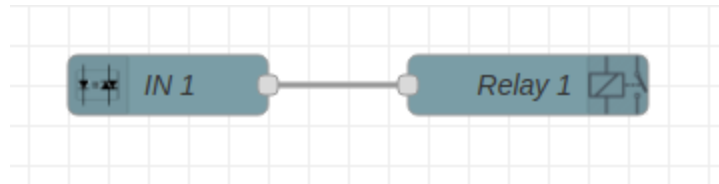


Figure 6-10: OPTO Input to Relay

### Selection of Relay by `msg.relay`

You can select a relay by specifying the relay number in the `msg.relay` property. To do this you must go to the Relay node edit window and delete the number in the “Relay Number” dropdown menu. When you do this “`msg.relay`” will appear next to “Relay Number” in place of the number as shown below.

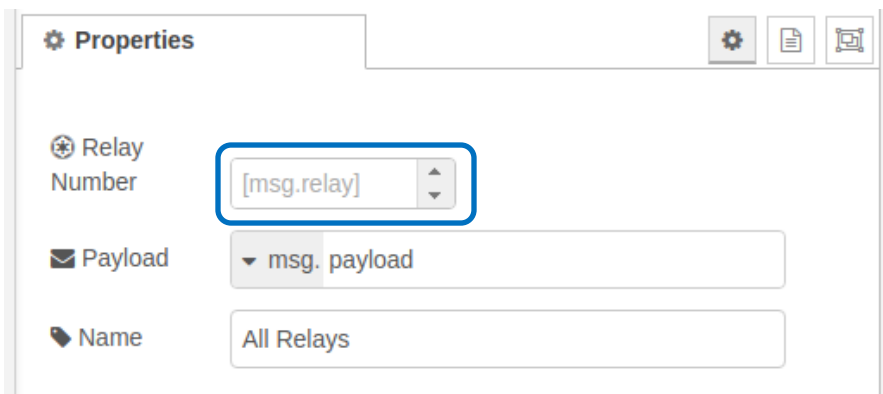


Figure 6-11: Programmed Relay Selection

Here is a simple example of how programmed relay selection might be used. The figure below shows a binary encoder that converts a decimal number (a very small decimal number!) to a binary selection of the two relays according to the following table.

Number	Relay 2	Relay 1
0	OFF	OFF
1	OFF	ON
2	ON	OFF
3	ON	ON

Here is the flow.

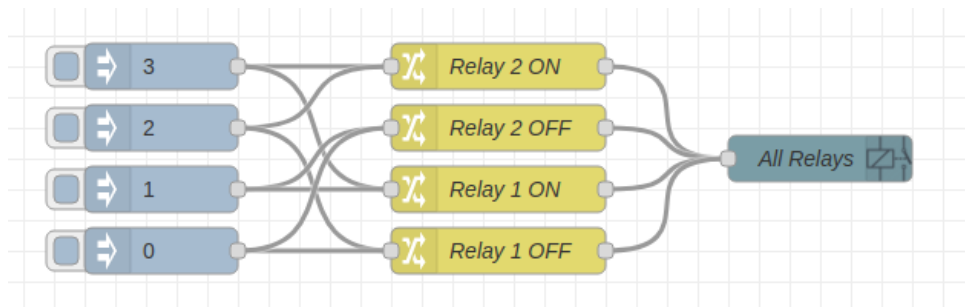


Figure 6-12: Simple Binary Decoder



## Learning Kit Workbook (version 1.3)

The idea is simple. On the left side are four Inject nodes, each of which set the *msg.payload* to a Boolean *true* when triggered. Each of the Change nodes in the middle sets a given relay to a particular state (OFF or ON) according to the name of the node. By sending the output of a given Inject node to the proper subset of change nodes the relay states will be set correctly. Here is an example of the configuration of one of the Change nodes, namely, the “Relay 1 OFF” node.

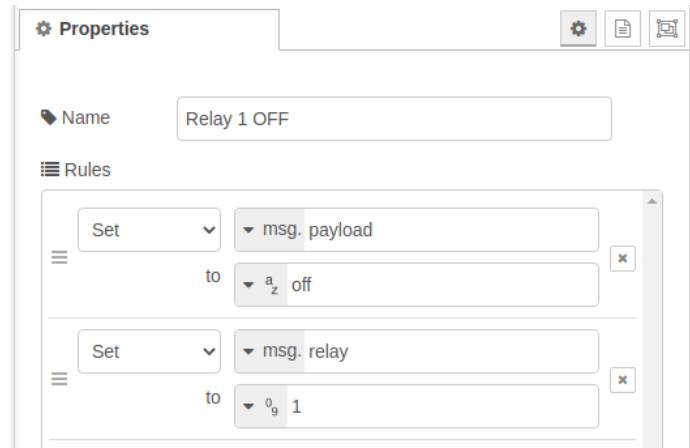


Figure 6-13: Relay 1 OFF Node Edit Window

From this node you should be able to determine how to configure the other Change nodes. There is something interesting about this flow. Consider the “0” inject node. When clicked it sends a message to two nodes, one to control Relay 1 and one to control Relay 2. There is no predetermined order to when the two selected nodes send their commands to the LKit Relay Node. This means that the order in which the LKit Relay node receives the commands is not fixed nor is it necessarily the same each time an Inject node is triggered.

**Delayed OFF** – In some apartment buildings, especially where electricity is expensive, the light switches in stairwells are pushbuttons. When you push the button the stairway lights turn on and stay on for a period of time and then turn off. Assume that Relay 1 is controlling the lights in a stairwell. Design a circuit so that pushing a button (LKit OPTO or Button) turns the light on and after 30 seconds (or five seconds if you are impatient) the light turns off.

Give it some thought and try things out. Do you think a circuit like this will work?

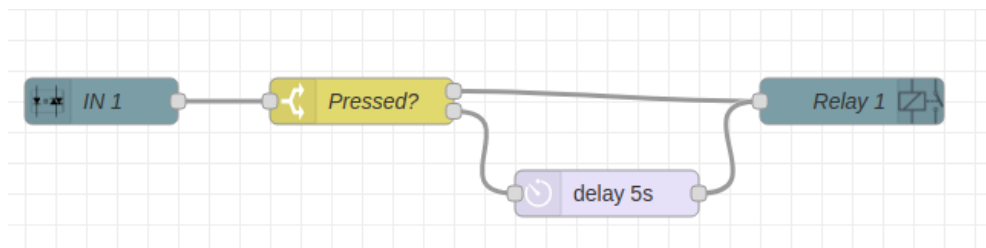


Figure 6-14: Delayed Turn Off

Go further: Go back and look at some of the projects that used the LEDs. Any flow you created for LED you can probably modify to work with the relays. The only limitation is that you have only two relays.

Boot camp is over. You are now a Second Lieutenant of The Relays. Time to begin...

## Learning Kit Workbook (version 1.3)

### Walking with the Relays

Here is a simple motor control project... Turn a motor on and off with Relay 1. You will need a small DC motor and a battery pack (or lab power supply) that is sufficient to power the motor. In this example, assume that a three-volt battery pack composed of 2 AA cells will run the motor. The figure below shows how to wire up your motor and battery to the male plug.

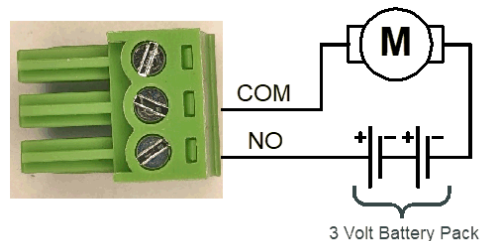


Figure 6-16: Wiring for Simple Motor Circuit

Now it is time to build a flow. Assume that a button connected to OPTO input 1 will control the motor so that when the button is pushed the motor runs. It could not be simpler as the figure below shows.

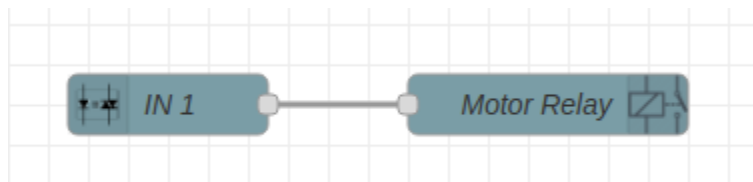


Figure 6-15: Motor Control Flow

This flow is so simple you will not need any help to implement it!

### Deploy and Test!

Does it work? You have now extended your powers of control from the Learning Card to the outside world of circuitry. When you push the button connected to OPTO Input 1 you will hear the relay click and you will see the RL1 LED light up. If this is not happening, then the problem is in your flow or in the button connected to IN 1. You know how to debug this with Inject and Debug nodes.

If the relay is not activating, then look at your circuitry:

- Are the batteries fresh and new or are they old and tired? Try replacing them with batteries that you know work.
- Is your connection complete at the plug? You can try things out by taking a short piece of wire and touching it to the two screws on the plug (COM and CO). If the motor starts, then there is a problem and you should start by rewiring the plug.
- Take everything apart and see if you can connect the motor directly to the battery pack. If the motor runs, then the problem is in your connections.
- Debugging a circuit is very similar to debugging a flow or a program. Work to isolate the problem by testing smaller and smaller parts of the circuit until you find the problem.

## Learning Kit Workbook (version 1.3)

**Reversing Motor Controller** - In the flow above the motor only runs in one direction. Wouldn't it be useful if you could control the direction of the motor? Well, you can because you have two relays. Your system is going to look like the figure below.

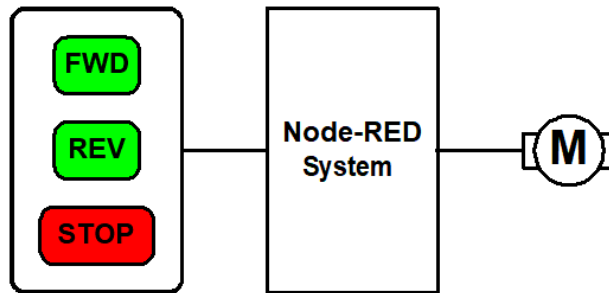


Figure 6-17: System Diagram of Reversing Motor Controller

This project will require three pushbuttons connected to the OPTO inputs, a motor and a battery pack. Here is the specification.

- When the system is deployed the motor will be stopped
- Pressing the FWD button will start the motor in the forward direction
- Pressing the REV button will start the motor in the reverse direction
- Pressing the STOP button will stop the motor.

There is at least one problem with this specification: suppose the motor is running in the forward direction and you press the REV button. Should the motor immediately go into reverse? Probably not, because this is a bit like slamming your car into reverse while you are driving down the road at 60 MPH. However, start by building the simple controller and then once this is working enhance it to put a pause in between switching directions.

The next question is how to use the two relays to both start and stop the motor and reverse the direction. Here is a circuit that should do the trick. Look at it and see if you understand what is happening.

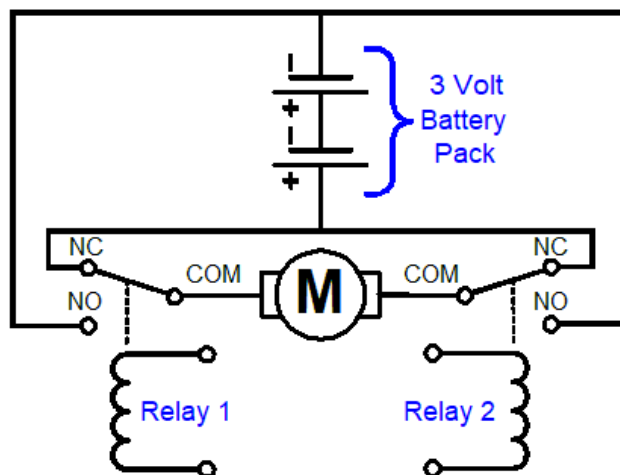


Figure 6-18: Motor Reversing Circuit Schematic

## Learning Kit Workbook (version 1.3)

Figure 6-18 shows the contacts in the initial deployment position where both relays are deactivated. Now suppose that Relay 1 is activated. The situation will be as shown in the figure below.

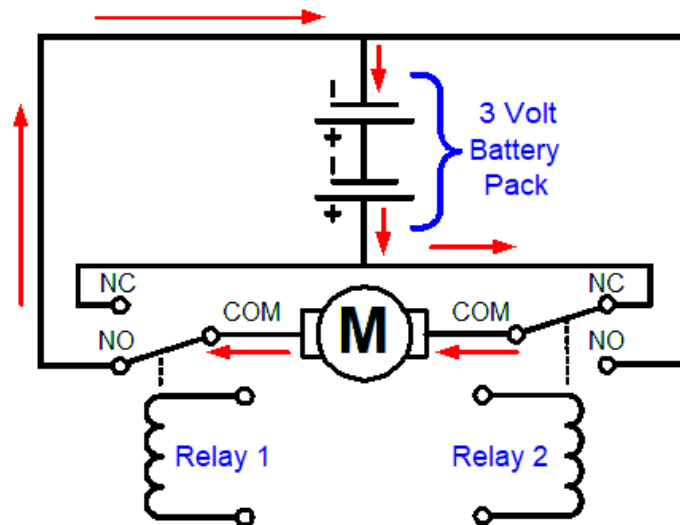


Figure 6-19: Reversing Circuit - Forward

For convenience let's call the situation where Relay 1 is activated the "forward" direction. The red arrows in the diagram show the flow of current<sup>66</sup> from the battery through the NC contact on Relay 2 (because it is deactivated), through the motor, through the NO contact on Relay 1 and back to the battery.

Now consider the situation where Relay 1 is deactivated and Relay 2 is activated, as shown below.

<sup>66</sup> Here we are taking about "conventional" current flow where current flows from the positive terminal of the battery to the negative terminal. The actual physical flow of electrons is from the negative to the positive terminal. Most electrical engineers talk in terms of "conventional" current flow.

## Learning Kit Workbook (version 1.3)

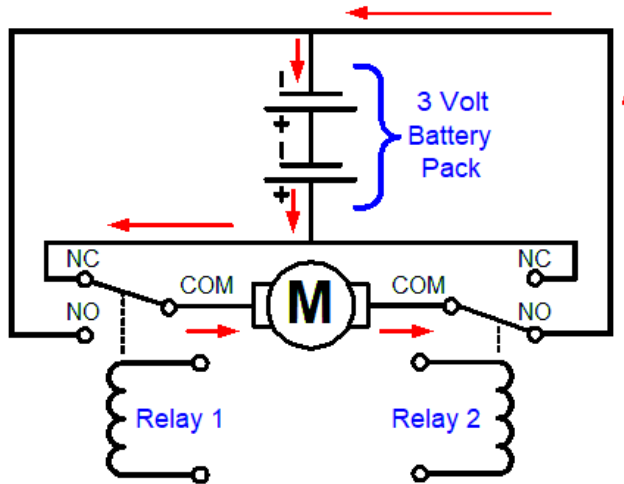


Figure 6-21: Reversing Circuit - Reverse

Compare the current flow in this figure with that in Figure 6-19. You will notice that the current flow through the motor is in the opposite direction. Finally, suppose both relays are energized as in the figure below.

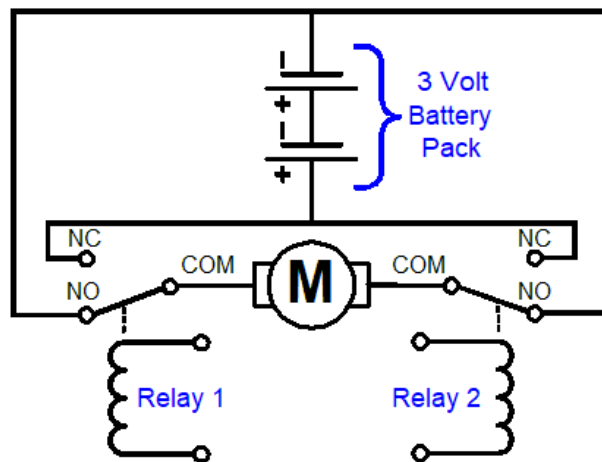


Figure 6-20: Reversing Circuit Both Relays Energized

In the case where both relays are activated there is no closed loop that includes the battery and so no current flows. One fortunate aspect of this circuit is that no matter how the relays are activated there is never a case where the battery is short circuited. However, this is only the case if you wire the circuit properly. Be careful because if you create a short circuit path you will cook the battery, the wires and the relay contacts, but at least there will be a brief moment of smoke to enjoy before the insulation on the wires melt into a small puddle on the table.

Here is how to wire up your motor controller.

## Learning Kit Workbook (version 1.3)

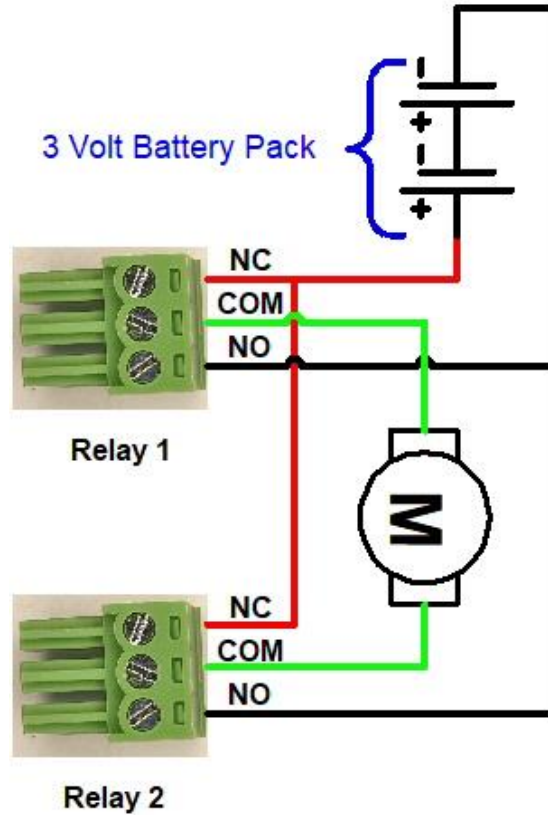


Figure 6-22: Wiring for Reversing Motor Controller

Here is a chart showing how the two relays control the motor.

Relay 1	Relay 2	Motor
OFF	OFF	STOP
OFF	ON	FWD
ON	OFF	REV
ON	ON	STOP

You have everything you need to build the flow. Give it a try! You might start by using a simple flow, like Figure 6-7, to control each relay separately to make sure your circuit is working properly. Then move on to your final flow. Assume that the FWD button is IN3, the REV button is IN2, and the STOP button is IN1.

Here is one possible flow, but there are others that may be simpler or better.

Figure 6-7: Basic Relay Control Test

## Learning Kit Workbook (version 1.3)

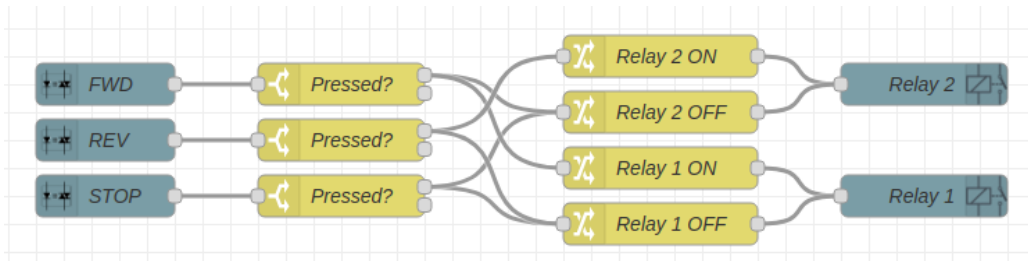


Figure 6-23: Motor Reversing Flow

**Puzzle # 1 - Run while pressed** – modify the flow above so that the motor only runs while a button is pressed. If you do this you will only need two buttons, one Forward and one Reverse.

**Traffic Light Controller** – Aquaville is a small place, but where Hamster Avenue crosses Water Street there is always a traffic jam. The solution of course is a traffic light. Each side of the traffic light will have a red, green and yellow lamp, in this case 12-volt LEDs. Your task is to control the LEDs with the Learning card relays and to develop a flow that will keep everybody happy. Below is a figure showing how the lights are to be sequenced.

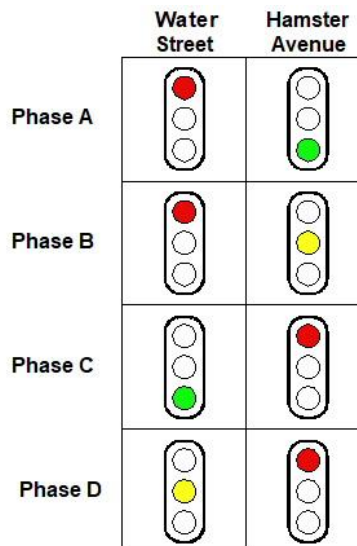


Figure 6-24: Traffic Light Phases

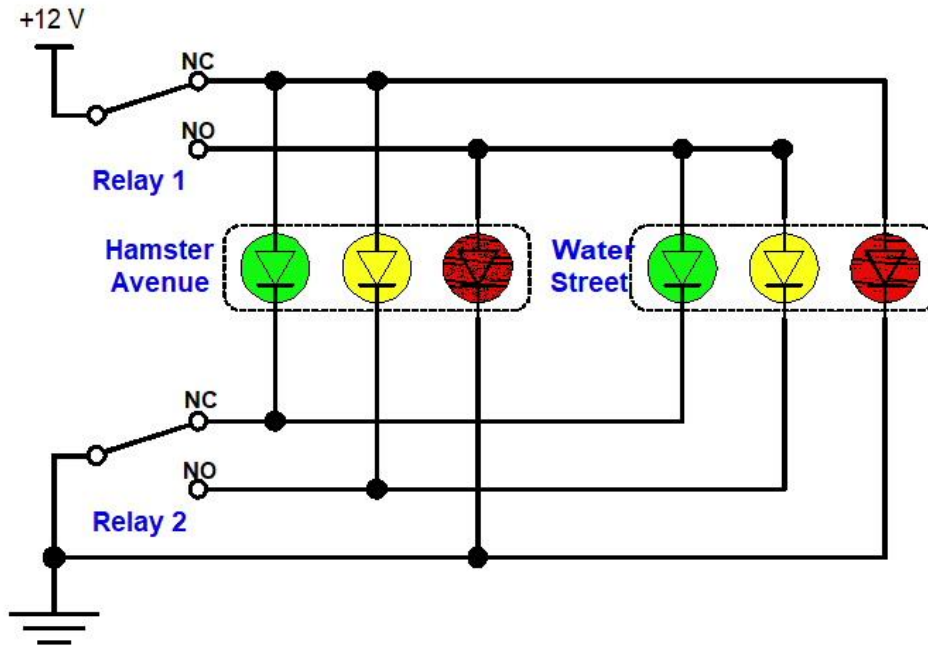
Each column shows what the traffic signal looks like from one of the crossing streets. The rows are the “phases” of the traffic light sequence. “Phase” is traffic engineer talk for one unique configuration of the traffic lights. In a simple traffic control situation, like the one in Aquaville, there are only four phases, A, B, C and D, that repeat in sequence forever.

The first step is to figure out how you are going to control the traffic lights. If your learning card had a dozen relays it would be easy because you could assign one relay to each lamp. However, in Aquaville the town mothers and fathers only have enough money to buy a single Learning card, so you must get by with just two relays. The first thing to notice is that even though you have 12 LEDs to control you can cut the situation in half because the pattern for Hamster Avenue is going to be the same on both sides of the traffic light. The same applies to the light for Water Street.

## Learning Kit Workbook (version 1.3)

Now is the time to stop and think a bit. Your relays can be in one of four states as shown in the table below Figure 6-11. Fortunately, the traffic controller only has four phases. Put on your thinking cap, or some music, or some coffee and see if you can come up with a schematic so that the two relays on the Learning Card can control the traffic lights.

Below is one possibility. If you use LEDs make sure you include a series current limiting resistor (not shown in the figure. Better yet, you can purchase LEDs with bases to fit E10<sup>67</sup> sockets in multiple colors that have a built-in resistor and are much cheaper than buying incandescent lamps.



*Figure 6-25; Traffic Controller Schematic*

Look this schematic over carefully and see if you can understand how the four phases are generated. It might help to make a chart that shows the four possible positions of the relays and from that you should be able to deduce the phases. Note: in the figure above a black dot indicates that two wires are connected. If two wires cross, but there is no dot at the intersection then there is no connection.

Relay 1	Relay 2	Phase	Water Street	Hamster Avenue	Time
OFF	OFF	A	RED	GREEN	12 sec
OFF	ON	B	RED	YELLOW	3 sec
ON	OFF	C	GREEN	RED	9 sec
ON	ON	D	YELLOW	RED	3 sec

Be sure to check the table above against the schematic.

Now for the flow. In normal traffic control the light would be on in each direction for say one minute, the yellow would be 5-7 seconds. However, that is too boring for this project, so the times shown in the

<sup>67</sup> Many lamps have standardized bases. E10 is a common base for older incandescent flashlight bulbs.



## Learning Kit Workbook (version 1.3)

table above have been shortened. Given your vast experience in designing flows you will have no trouble building this flow and configuring the nodes. Use an Inject node to start the flow.

Here is one possible approach.

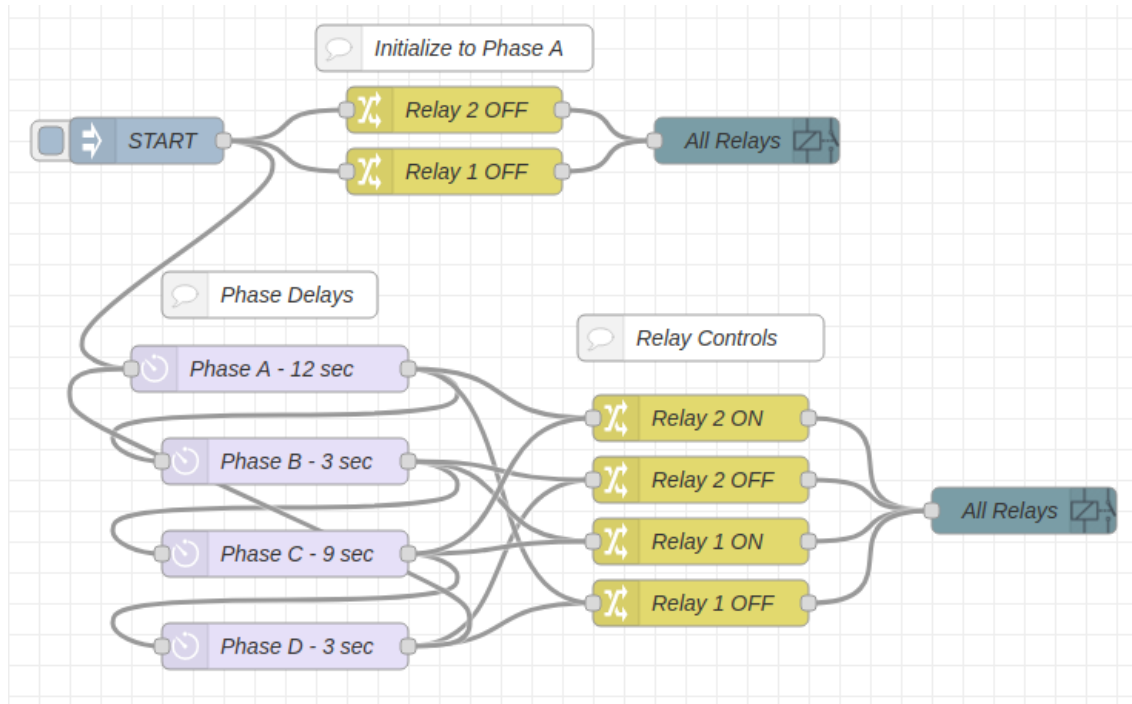


Figure 6-26: Traffic Light Controller Flow

The injected message can have any sort of payload because it is only used to trigger other actions. Each delay node is associated with a different phase. Just remember that when the message leaves a node it should set the relays to the configuration for the following node. For example, when the message leaves the “Phase A – 12 sec” Delay node it must set relay 1 off and relay 2 on to indicate the start of Phase B. Every node in this flow is a node you have configured before. For example, Figure 6-12 shows how to decode a phase into the relay controls.

**Puzzle # 2 - I Don't Wanna Wait!** After the new traffic light was installed the citizens of Aquaville complained to the village council that Hamster Avenue was a very busy street, but Water Street had very little traffic. Cars on Hamster Avenue just hated to wait while the light cycled through phases C and D when there was no traffic. The council told Ma Water (Chapter 1) to fix it up. Her solution was to install a switch on Water Street at the intersection on each side of Hamster Avenue that is triggered by the weight of a car<sup>68</sup>. Your task is to modify the flow so that normally it cycles only through phase A and phase B. If a car stops on the switch, then at the end of the next Phase B the controller should cycle through Phase C and D. If you do this everybody in Aquaville will think you are a hero.

**Puzzle # 3 - I Don't Wanna Wait Either!** Well, the folks in Aquaville are never happy. Now they are complaining the during the rush hour the people getting out of the Purrfect Pet Pellet Plant on Water

<sup>68</sup> Using switches activated by the weight of a car is old technology. Rain, snow and the constant pounding of cars would make the switches unreliable. Modern systems use loops in the pavement to detect the cars by the change in loop inductance. Some other modern systems use cameras to detect the presence of cars.

## Learning Kit Workbook (version 1.3)

Street do not get enough time to go through the light went the day shift ends. Your task is to design a flow that changes the Phase C delay so that it is 15 seconds only between 4:45 PM and 5:15 PM. Hint: a few Inject nodes can do this if you configure them properly

**Puzzle # 4 - Gray Codes** – There is a small engineering shortcoming in the design shown in Figure 6-25. Look at the table below the figure and you will see that when the relays transition from phase B to phase C (and from phase D to phase A) that both relay contact change simultaneously. Think back to the discussion about how electrical components live in your world, the real world. The contacts do not switch instantly, and one relay may be faster than another relay. This means that when the contacts change from say phase D to phase A they might hit phases B or C before they finally settle down. You might even see this as a small flicker between phases.

Engineers have a solution for this called “[Gray Codes](#)”. In a Gray coded sequence only one relay contact changes each time the phase changes. This keeps false contact configurations from occurring. One Gray code that would work is this:

Relay 1	Relay 2	Phase
OFF	OFF	A
OFF	ON	B
ON	ON	C
ON	OFF	D

Look carefully and you will see that between each phase transition only one relay contact changes. Perfect, now you will need to fix the schematic in Figure 6-25 and make a few small changes to the flow in Figure 6-26. Are you up to the challenge?

### Combination Lock

**Combination Lock** – Assume for a moment that you need to protect your precious supply of Healthy Hamster Treats™ from predation by your other pets. You decide to build a combination lock for your fireproof (and rodent proof) pet food safe. Here are the specifications:

- This project will use four pushbuttons on the OPTO inputs, two relays to control a motor<sup>69</sup>, one on-board pushbutton and four LEDs! In other words everything you have learned about.
- The safe will have four pushbuttons numbered 1 through 4 corresponding to OPTO inputs IN1 to IN4, respectively.
- The combination will be 2-1-1-4.
- If a wrong number is entered at any point in the sequence the sequence must be entered again from the beginning.
- The on-board pushbutton (LKit Button node) will clear any entry sequence in progress so that the sequence must be entered from the beginning.
- When the correct sequence is entered the motor will run in the unlock direction for 3 seconds.
- While the safe is unlocked all the LEDs will be lit.

---

<sup>69</sup> A typical small electronic safe, like a hotel room safe, has a small motor inside that unlocks the safe by pulling in a small peg that holds the door shut.

## Learning Kit Workbook (version 1.3)

- When the safe is open operating the on-board pushbutton (PB) will extinguish the LEDs and run the motor in the lock direction for 3 seconds.
- Upon Deployment the safe will be locked

The combination problem is a good example of a [finite state machine](#) implementation. A finite state machine is an important logic and programming abstraction used to implement certain types of action sequences. The notion behind a finite state machine (FSM) is that it has a “state” that remembers what actions have taken place up to the present point in time. Think about a mechanical combination lock, like the one on a typical school locker. It has a mechanical memory inside, because if you enter the first two numbers and set the lock aside, you can come back any time later and enter the last number to open the lock<sup>70</sup>. This is because the lock was in a “state” that represented the fact that the first two numbers had been entered correctly. Who knew? The combination lock is a mechanical finite state machine! Computer science abstraction in the palm of your hand.

### Warm Up with Something Simple – Bottle Production Line

Before plunging into the combination lock problem let’s start with something simpler. Yes, the problem is a bit contrived, but at least it is easy to understand and illustrates how to build a Finite State Machine.

Here is the situation. On the production line red and blue bottles move along a conveyor belt. There is a sensor at the end of the conveyor belt that closes OPTO input IN 1 if the bottle is red and IN 2 if the bottle is blue. For whatever crazy reason your boss wants you to build a flow that will use LED 1 and LED 2 to show whether the count of red bottles that has passed the sensor is even or odd. If the current count is even, then turn on LED 1 if it is odd turn on LED 2. However, you should not count the blue bottles. Here’s what the physical situation looks like.

Many times, problems like the bottle counter are represented as finite state machines using diagrams like the one below.

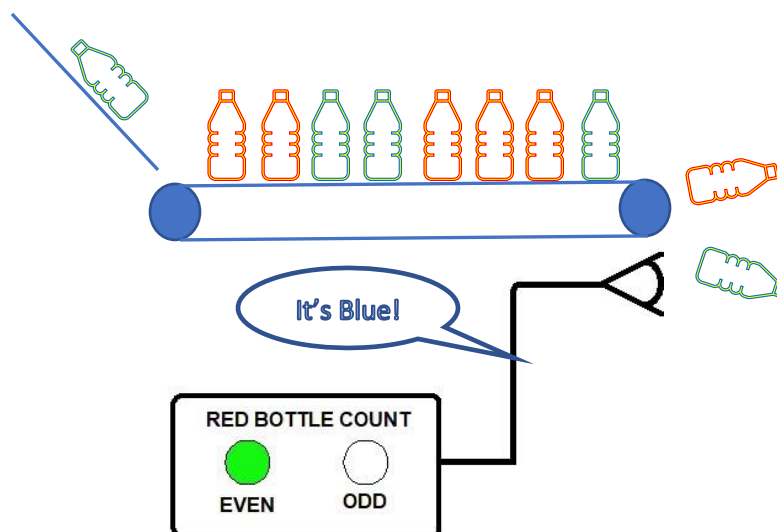


Figure 6-27: Bottle Production Line

<sup>70</sup> That’s why the kid in the locker next to yours in high school could get into their locker so quickly and was first in the lunch line. When they closed it in the morning, they entered the first two numbers of the combination before they left for class.

## Learning Kit Workbook (version 1.3)

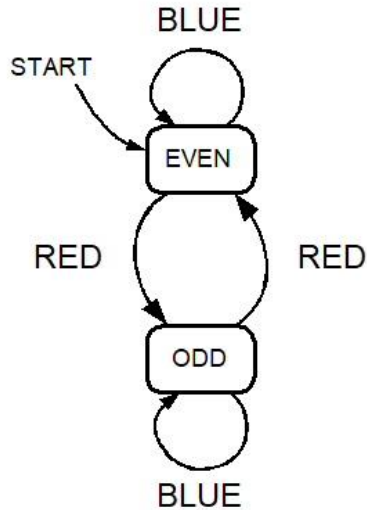


Figure 6-28: Finite State Machine Diagram for Bottle Counter

In this diagram bubbles represent the “states”, in this case whether the current count of red bottle is even or odd. So, only one thing to remember, which is perfect use of a context variable. When the production line starts up the count is even, because zero is an even number. When a bottle passes the sensor in Figure 6-27 either OPTO input IN 1 or IN 2 will be closed depending upon the color of the bottle.

If you were to draw Figure 6-28 out in chalk on the sidewalk you could be the key element in the finite state machine, namely the keeper of the state. Start out by jumping on to the block labeled EVEN because when the production line starts moving no bottles have been detected and zero is, after all, an even number. Suppose someone standing by the production line yells “Red” or “Blue” each time a bottle goes by? If you hear “Red” you look down at your feet and jump to where the red arrow points. If they yell blue, you just jump up in the air because the arrow points to the same state bubble you are standing on. When someone asks you whether the current count is even or odd all you need to do is look down at your feet and report the name of the state you are standing in. Simple enough?

To make things easy start by thinking about the parts of your flow:

- A flow to detect the state of the OPTO inputs IN 1 and IN 2.
- A finite state machine flow to implement Figure 6-28.
- A flow to take the proper action, such as turning on LED 1 when the count is even.

Here is the flow to detect and encode the events. To make things easy events are encoded as strings (i.e., “BLUE” and “RED”).

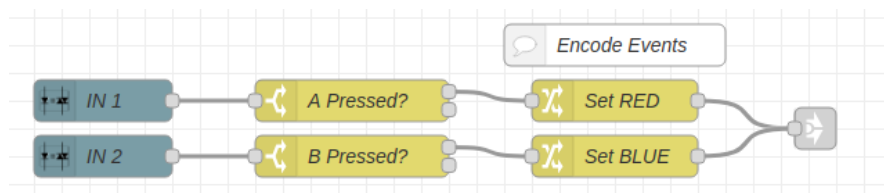


Figure 6-29: Bottle Counter - Encode Events

## Learning Kit Workbook (version 1.3)

The messages with events “RED” and “BLUE” are passed to the Finite State Machine flow below.

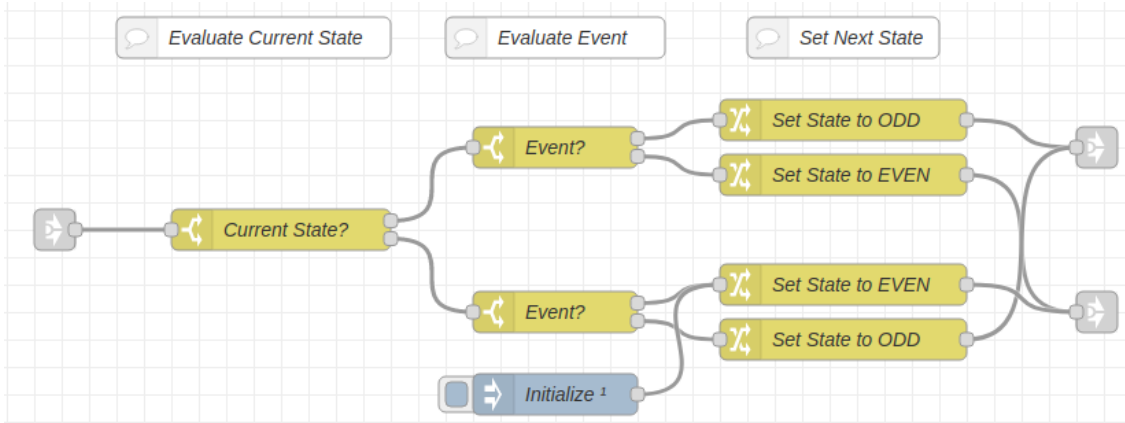


Figure 6-30: Bottle Counter - Finite State Machine

Event messages from the output of the event encoding flow (Figure 6-29) are passed to the Link In node on the left side of the diagram above. The first Switch block (“Current State”) determines if the current state is “EVEN” or “ODD” as designated by the State context data variable. Here is the configuration of the “Current State?” node.

The “Initialize” node makes sure that the initial state of “EVEN” is set when the flow is deployed.



Figure 6-31: Finite State Machine - Current State Evaluation

The purpose of the “Current State?” block is to send messages that occur when the current state is EVEN (Port 1 on the top of the node) one way and messages that occur when the current state is ODD the other way (i.e., to port 2).

## Learning Kit Workbook (version 1.3)

Next, the “Event?” Switch node determines whether the message payload is “RED” or “BLUE”. The configuration is below.

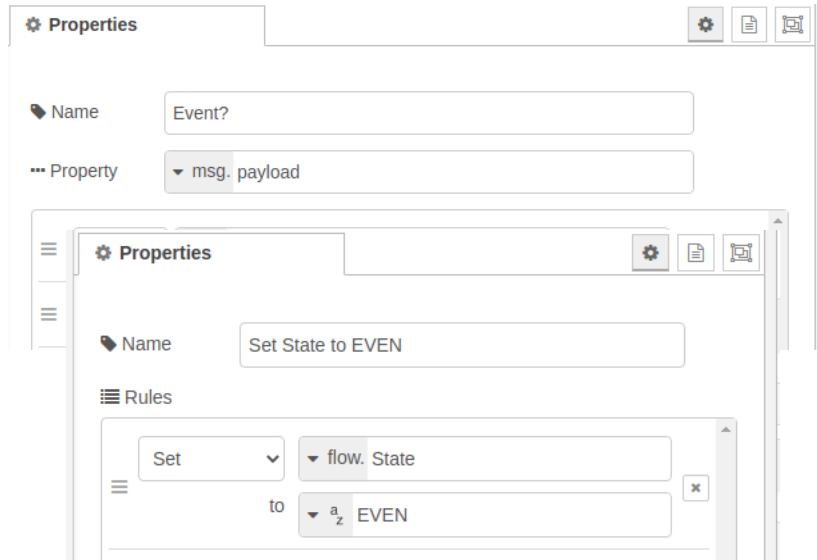


Figure 6-33: Finite State Machine - Setting Next State to Even

Once the message has been routed through the “Current State?” and “Event?” nodes it activates one of the Set Next State nodes at the righthand side. In this flow even if the state is unchanged (e.g., the current state is odd and a blue bottle is detected) the next state is still set, it is just set to the same value. This arrangement is for completeness. Later you will see how to simplify the slow by deleting these unneeded state setting nodes. Here is an example of setting the next state to EVEN.

Here is the configuration for setting the next state to ODD.

The final flow is to take the message that set the next state and use it to generate an output action to set the LEDs

This flow uses the “group” mode for the LKit LED node because it allows on LED to be turned off and the other to be turned on with just one node.

Look back at Figure 6-28. Notice that whenever a blue bottle goes by the state of does not change. And in Figure 6-30 two of the nodes that set up the next state based on the event being “BLUE” are redundant because they simply set the new state to the save value as the old state. Therefore, these nodes can be removed from the finite state machine flow as shown below. Note: it is necessary to move

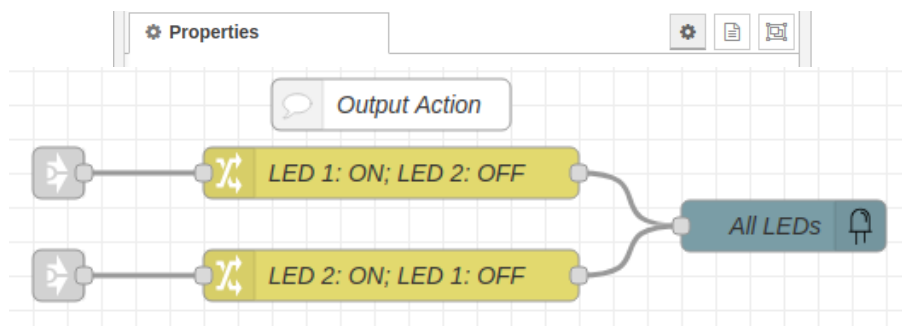


Figure 6-35: Bottle Counter - Output Action

## Learning Kit Workbook (version 1.3)

the initialization to set the EVEN state because the node that it was originally connected to has been eliminated.

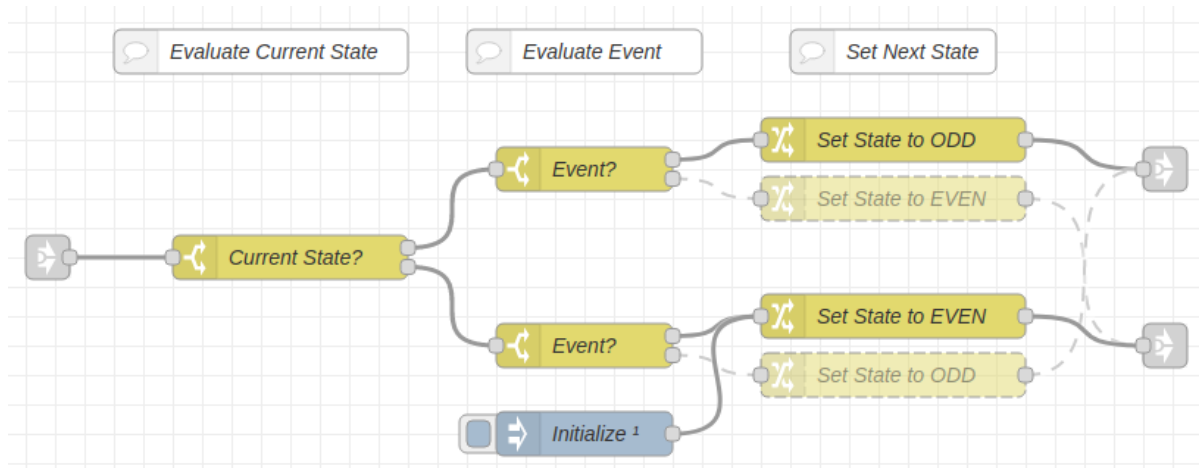


Figure 6-36: Bottle Counter - Simplified Finite State Machine

In the flow above the deleted nodes are shown in faded colors. These are not needed because they do not change the current state, so in that sense they are redundant.

See if you can build and test this flow. Once you get it working study it carefully because the general approach can be used in other designs.

**Puzzle # 5 - Divisible by Three.** Modify the bottle counting above to turn on LED 1 if the number of red bottles that have gone past the counter is exactly divisible by three.

**Puzzle # 6 - Count Red and Blue.** Modify the bottle counting flow above so that it keeps track of the even and odd counts for both red and blue bottles. Let LED 1 turn on when the red count is even and let LED 2 turn on when the blue count is even. The LEDs will each be off when the associated count is odd.

### Combination Lock Implementation

Now you are ready to build the full combination lock flow according to the specification given earlier.

## Learning Kit Workbook (version 1.3)

For the electronic combination lock of the specification the figure below shows the function as a finite state machine.

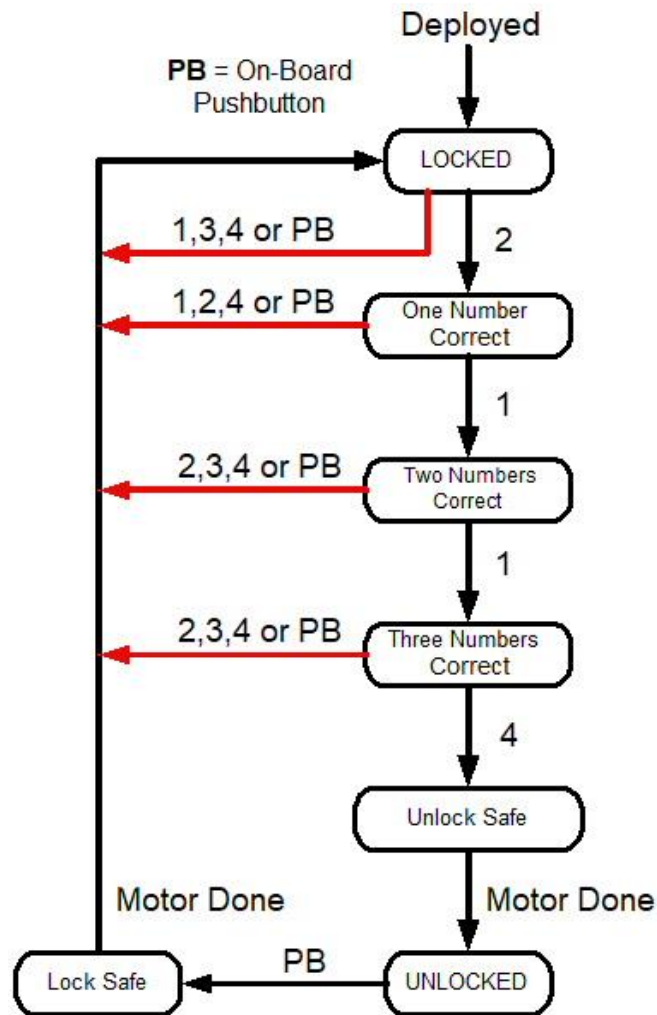


Figure 6-37: Finite State Machine for Combination Lock

**Error! Reference source not found.** shows a finite state machine for the combination lock puzzle. Each of the rectangles represents a state and the arrows represent transitions between states. Associated with each arrow are the conditions for transitioning to the next state. Black arrows are the normal path to open the safe with the combination, 2-1-1-4, and to relock the safe. Red arrows show situations where the wrong number was entered. PB is the on-board pushbutton used to reset the entry sequence.

The “state” of a finite state machine is a value that must be remembered over the time. This certainly sounds like a job for a context data variable, so keep in the back of your mind that you will need such a variable. A good name for this variable would be “State” because it is clear, concise and to the point. This variable must store a value that indicates the current state of the combination lock flow. You might



## Learning Kit Workbook (version 1.3)

as well store strings, like “LOCKED”, “One Number Correct” and so forth. This will be a little bit clumsy<sup>71</sup>, but it will make debugging easier.

Next step: sketch out your flow on the back of an envelope, napkin or whatever scrap of paper is handy. Below is one possible organization of flows. As always, you may have a much better idea!

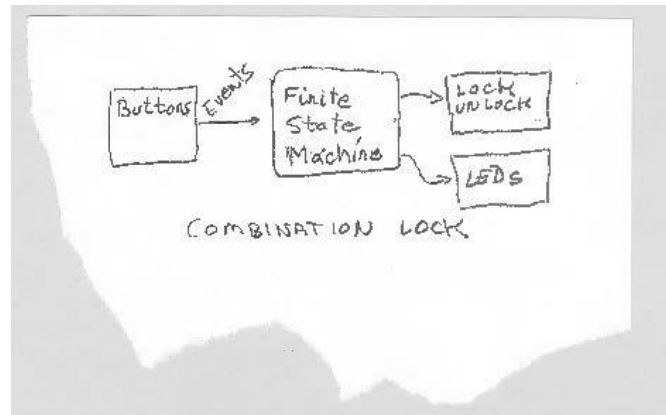


Figure 6-38: Combination Lock Flow

As you think about your overall structure also think about how you are going to test each part. It’s much better to exterminate bugs one room at a time rather than having to tent your whole house later.

First up... create a flow that will convert the four OPTO inputs and the pushbutton input into events, like 1, 2, 3, 4 and “PB”, respectively as in **Error! Reference source not found.** This is easy, the only thing you need to keep in mind is that you only want events to be generated when a button is pressed. Also, you will need to set up initialization to be done when you deploy the system. You have done this before ( ) and your flow might look like the figure below.

---

<sup>71</sup> In other programming languages you might assign an enumerated constant to each state, but there is no feature like this in Node-RED.

## Learning Kit Workbook (version 1.3)

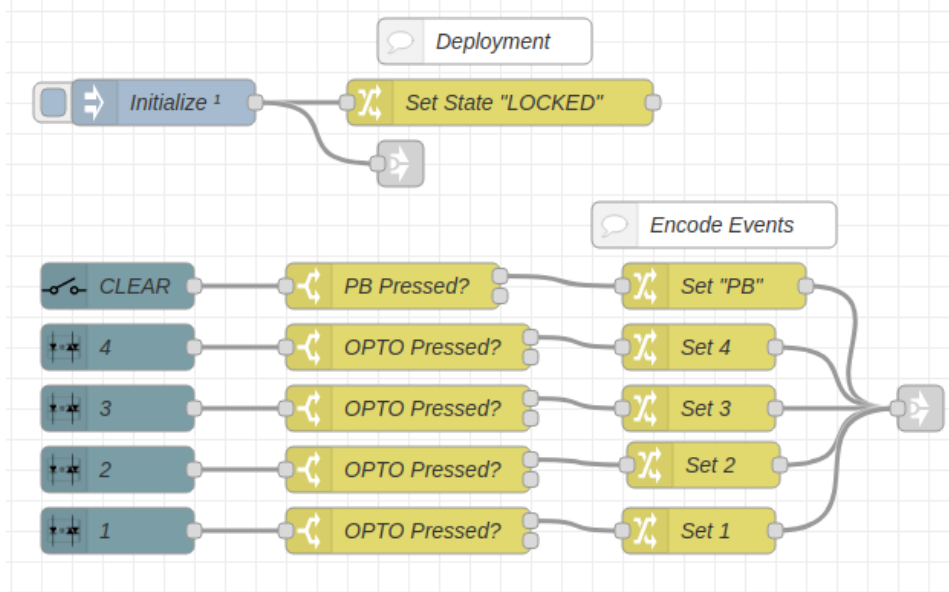


Figure 6-40: Deployment and Button Encoding

When this flow is deployed it will set the state of the safe to LOCKED and also reset the LEDs. The rest of the flow just encodes the push buttons and the four OPTO inputs into events that are sent out on the Link Out node at the righthand side.

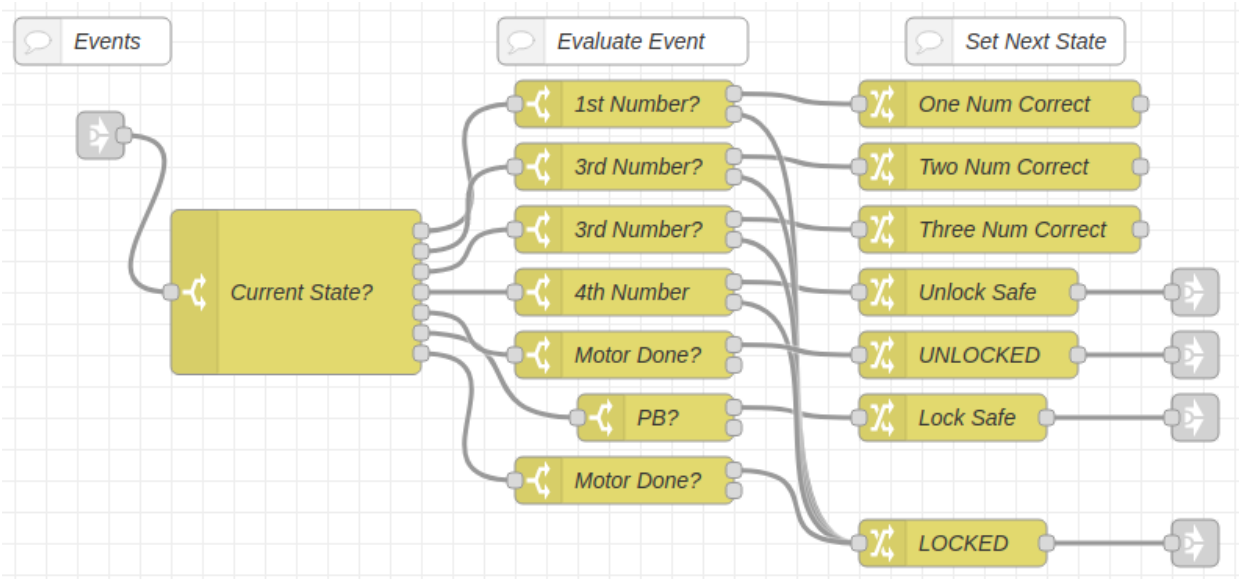


Figure 6-39: Finite State Machine for Combination Lock

If the flow above looks complicated that is because it is. A finite state machine receives events and based on the current state evaluates the events it has received in order to select a new state. In some case the new state may perform an action, like running the lock/unlock motor or setting the LEDs. Sometimes these actions may generate new events (see next flow below).

The Link In node at the lefthand side brings events into this flow. Based on the Switch node, "Current State?", the event is sent to the event evaluation nodes in the middle column. Here the event message

## Learning Kit Workbook (version 1.3)

is sent to one of the next state nodes in the rightmost column where the new state is set and if necessary, the message is passed on to perform an action.

Let's look at one simple example. Suppose the current state is "Three Numbers Correct" meaning that the first three numbers have been received with the correct value and order. There are two possibilities for what happens next (1) the number 4 is received correctly completing the sequence or (2) the numbers 1, 2, 3, or the pushbutton (PB) is received canceling the sequence. In the first case, where the number 4 is received, the "4th Number" Switch node will send the message to the "Unlock Safe" node where the state "Unlock Safe" is set and a message is sent to the unlock motor flow. In the other case, where the wrong number is received, the message is sent to the "LOCKED" Change node to return the system to the LOCKED state.

Check the rest of Figure 6-39 against the finite state machine in Figure 6-37 and make sure you understand how this flow works. Try building the Flow on your own. If you get stuck, follow this [link](#) to see the configuration of key nodes

In Figure 6-39 if a set next state node on the righthand side has an attached Link Out Node it means that that entry into that state is going to trigger an action. Trace the four Link Out nodes shown and see if you can understand what the action is.

The last flow is the one that controls the lock/unlock motor and the LEDs as below.

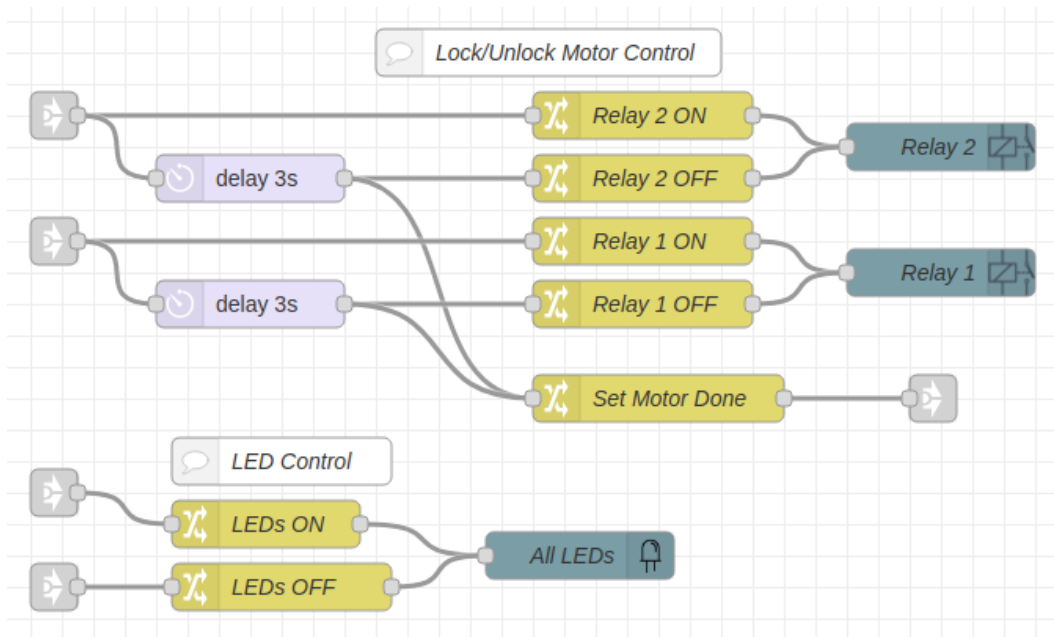


Figure 6-41: Motor and LED Controls for Combination Lock

Note the relay control for the lock/unlock motor is similar to that shown in Figure 6-23, but slightly simplified. When this flow receives a message to lock or unlock the safe it runs the motor for three seconds forward or backwards and when it is done generates a "Motor Done" event that advances the finite state machine. You should be able to construct this flow on your own, as you are now a Master Wrangler of nodes! Remember, test each section before you put everything together.

Frustrated? Look here for the configuration details [link](#).

## Learning Kit Workbook (version 1.3)

Be sure to test you design against the specification.

**Puzzle # 7 - Timeout.** In the combination safe example above if you enter the first three number and then get distracted by a commercial hamster grooming products on TV the safe sits there only one number away for opening. This does not seem very secure. After all, if one of your hamsters accidentally steps on the number 4 button the safe will give up its treats to the poacher. Can you modify the flow so that if no button is pressed for five seconds the safe automatically relocks? Of course, if the safe is unlocked normally you do not want to relock it until the PC button is pushed. Hint: you might be able to use a Trigger node to accomplish this task.

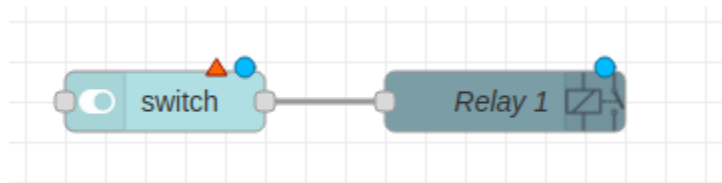
### Flying with Relays: User Interface

The time is now... the time to put the Node-RED dashboard User Interface (UI) to work with the relays. If you have completed the User Interface section in Chapter 5 you will be ready to fly with the relays. The Node-RED dashboard contains several nodes that you can use to control output devices, like on-board LEDs and the relays. What you already know about using real buttons and switches will serve you well here.

### Dashboard Switch Widget

As per the usual, start with something simple. How about this? Set up a single switch on your dashboard to control one relay, which is about as simple as it gets.

- Go to the Dashboard section of the Palette and drag a dashboard Button node to the workspace.
- Pull in a Learning card Relay node.
- Wire them up as shown below



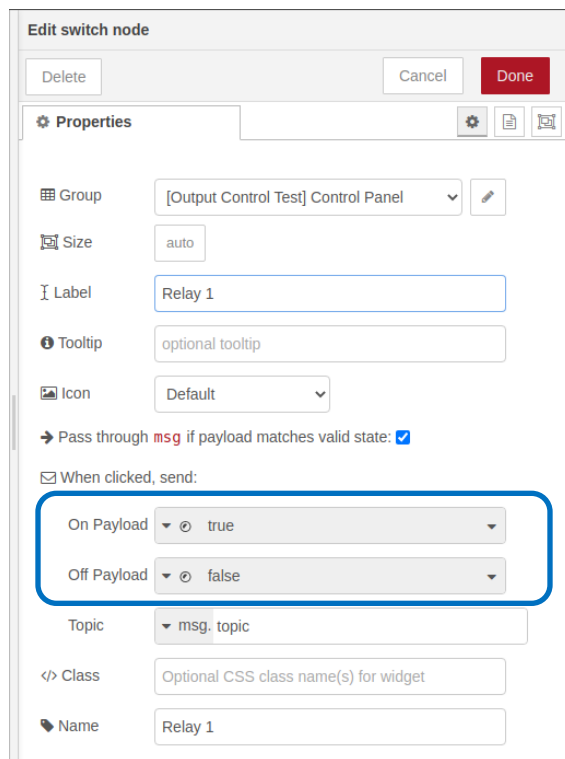
*Figure 6-42: Simple Dashboard Switch to Relay Flow*

The little red triangle perched on the Switch node tells you that this node needs to be configured before you can deploy it.

*Figure 6-43: Dashboard Switch Node Configuration*

## Learning Kit Workbook (version 1.3)

- Open the Switch node and configure it as shown below.



- Look at the [blue rectangle](#) above. Here Node-RED is trying to help you out by configuring your Switch node for you.

On the dashboard the switch will change from on to off or off to on whenever you click on it. When you do the node will send a message with a payload that depends upon whether you turned the switch on or off. Because the Learning card Relay node understands Boolean *true* and *false*, you can wire the switch directly to the Relay node.

- Set up the Relay node to control Relay 1. You know how to do this already because you are now a Captain of Relays.

*Deploy it!*

Go to your dashboard and you should see this.

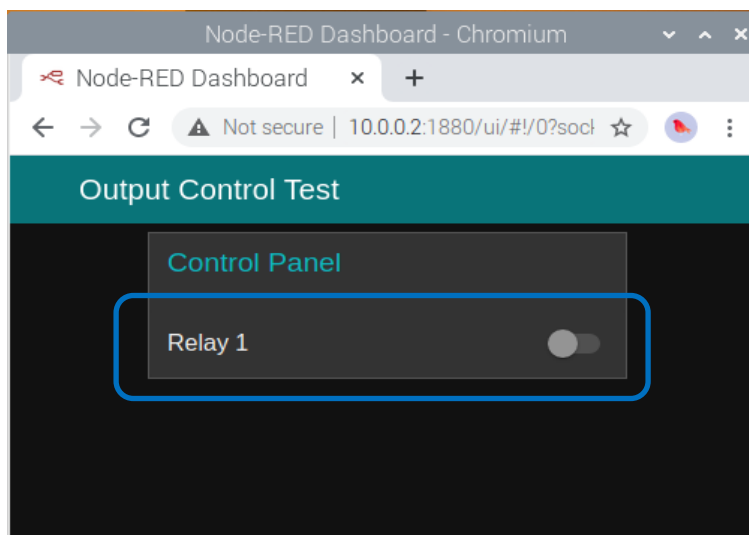


Figure 6-44: Dashboard for Switch to Relay Flow

## Learning Kit Workbook (version 1.3)

The area within the blue rectangle is the Switch widget. If you click anyplace in that area Relay 1 will turn on (or turn off). Take note: the dashboard Switch widget has a toggle action. When you click in the active area the switch changes state as shown by the little icon on the right side. Furthermore, the state only changes when you release the mouse key.

Why is the dashboard display black and dark teal in this example? The answer: you can change and even customize your dashboard in the dashboard sidebar. The theme above is the so called “dark” theme. Some folks think it looks better than the bland “light” theme that is the default.

Go to your sidebar open the dashboard tab and select the theme tab as below.

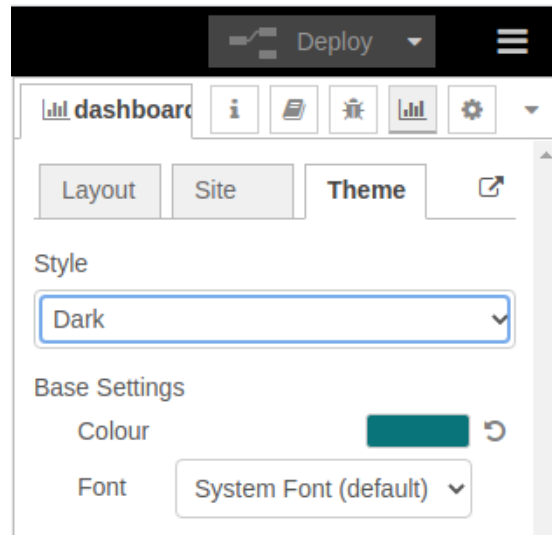


Figure 6-45: Dashboard Sidebar Theme Tab

The Style dropdown will let you select a theme. If you select “Custom” you will be able to adjust many color aspects of your dashboard.

Try your Switch to Relay flow. Remember the relay has a light to indicate whether it is on or off, so you can easily check the function of your flow. Connect something to the relay, like a motor or a light and control it from the dashboard.

## Learning Kit Workbook (version 1.3)

### Initializing Your Relay Flow

There is one small problem with the Switch to Relay flow in Figure 6-42. When you deploy your flow what state will the relay and switch be in? If you are unlucky the relay and the widget will be out of sync and your dashboard will show the relay in the wrong state. The way to fix this is to make sure the relay is set to a known state when you deploy your flow. The dashboard Switch node supports this by letting you send a message to the node to set it to a known state. The only provision is that the message you send must be one of the valid payloads specified by the “On Payload” or “Off Payload” dropdowns. Below is a flow that will do the trick.

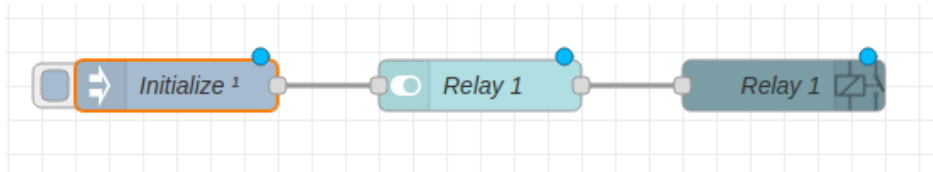


Figure 6-46: Flow to Initialize Switch and Relay Nodes

The “Initialize” node is an Inject node configured to fire a message once upon deployment. You have done this before in [Figure 6-45](#), but if you have forgotten the secret formula, here is the configuration.

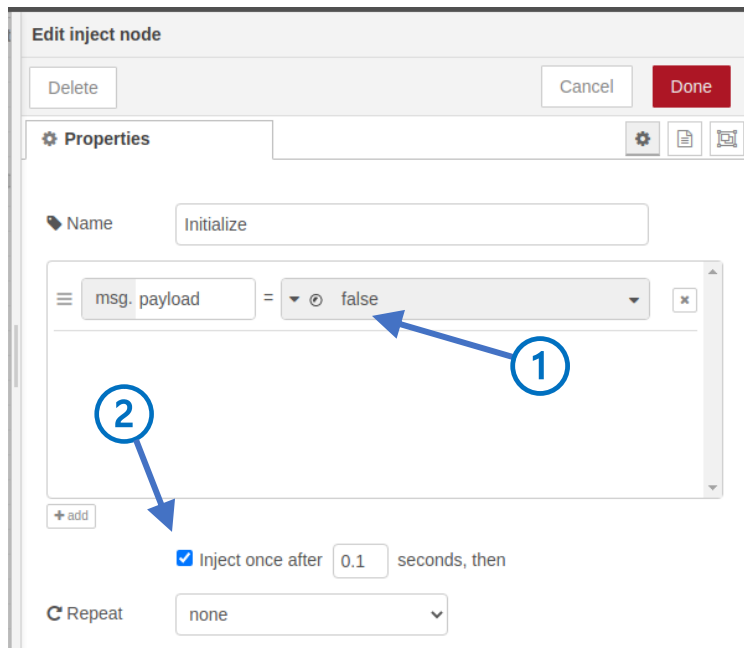


Figure 6-47: Inject Node Configuration

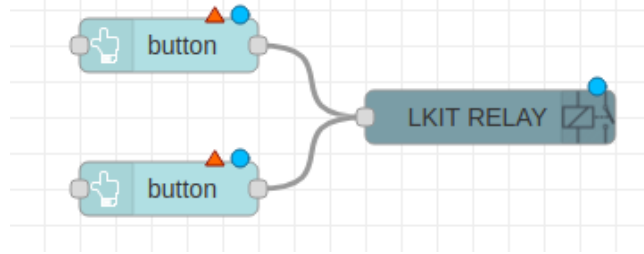
- Set the payload to Boolean *false* because you want to initialize the relay to OFF ①.
- Check the box to “Inject one after....” This will send one message with the payload *false* upon deployment ②.

**Deploy** this flow and then cook up a way to test it. You might add an normal inject node to set Relay 1 to on within your flow and then redeploy the flow to see if the relay and the switch widget are in agreement.

## Learning Kit Workbook (version 1.3)

### Dashboard Button Widget

Among the dashboard control widgets is the Button node. Although this is called a “button” it behaves a bit differently from the pushbutton on the Learning Card and the OPTO inputs. When you click on a dashboard button it only sends a payload when you release the mouse key. In this sense it works like the



*Figure 6-48: Button to Relay Initial Flow*

Inject nodes shown in Figure 6-7, which you should compare to the figure below.

Go to your workspace and build the flow shown above.

Time to get fancy. Look at the button configuration below. Notice that you can set the color of the button body, the color of the text and the label for the button. Let’s assume that turning the relay on is going to start a conveyor belt moving boxes of live hamsters for shipping to anxious pet owners. A green



## Learning Kit Workbook (version 1.3)

button to start the conveyor is good and better is a red button to stop the conveyor belt. Check out the configuration of the “Conveyor Run” node shown below.

Figure 6-49: ON Button Configuration

- Set the Label field to “Conveyor RUN” ①.
- Set the Color box to the color you would like for the text that will appear on the button ②. Valid colors come from the standard [HTML color names](#).
- Set the Background box to the color you would like for the background ③.
- Click Done.

Do something similar for the STOP button. You already know how to set up the Learning Kit relay node.

### Deploy!

Below is what your control panel should look like.

## Learning Kit Workbook (version 1.3)

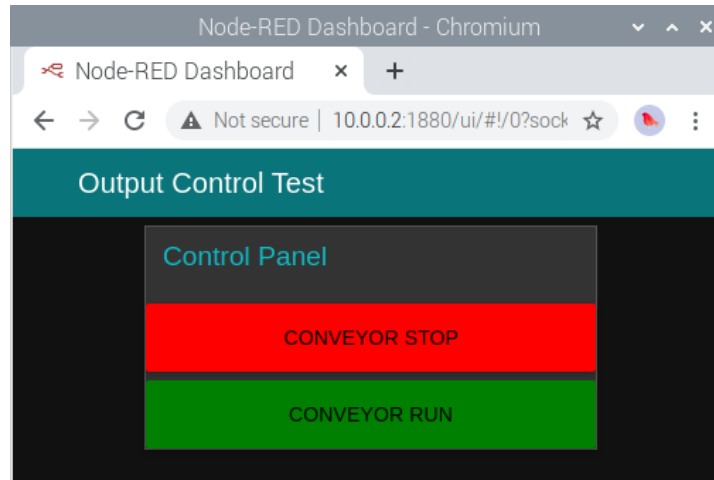


Figure 6-50: Dashboard for Conveyor Control

Give it a try and see if the state of Relay 1 changes when you click the different buttons.

### User Interface Icons

The Node-RED dashboard widgets can also display icons along side of or in place of the text. Using Icons, you can design nice looking control panels. Where are you going to find your icons? They are already part of Node-RED. [Font Awesome](#) is a group that supports a massive collection of icons. No need to visit their website or take any special action because you can reference many of the icons directly in Node-RED. Let's enhance the flow of Figure 6-48 by adding an icon to the buttons

- Open the edit window on the "Relay OFF" node
- Then fill in the Icon box as shown in the figure below ①.

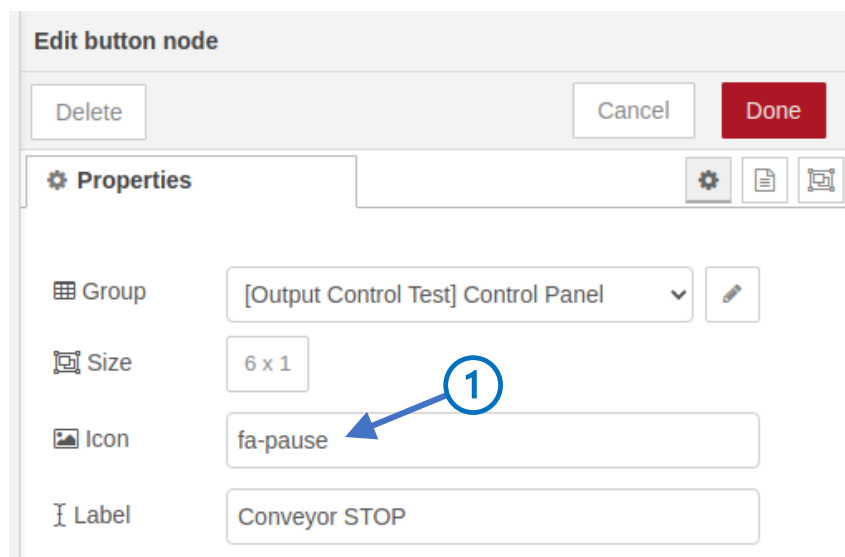


Figure 6-51: Adding an Icon to the Relay OFF Node

The name of the icon is "fa-pause" meaning that it is the "pause" icon from the Font Awesome library, hence the "fa-" preceding the name. You can find the name of the icon at Font Awesome's website. If it is available within Node-RED appending the name to "fa-" will load your icon. Not all Font Awesome icons are available for free.

## Learning Kit Workbook (version 1.3)

- Configure the “Relay ON” node the same way as shown below but using the fa-arrow-right icon ①.

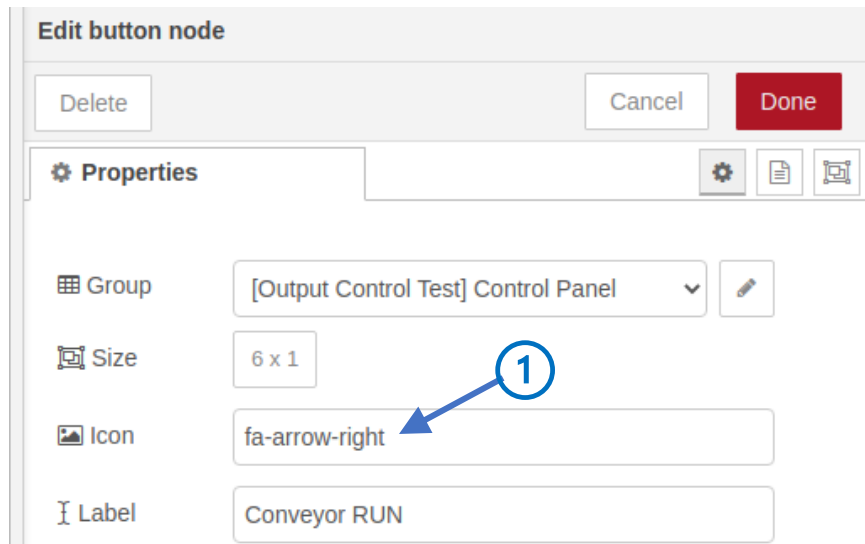


Figure 6-52: Adding an Icon to the Relay ON Node

- Click Done.

Ready, Set, DEPLOY!

You should see a dashboard that is very much like the one shown below.

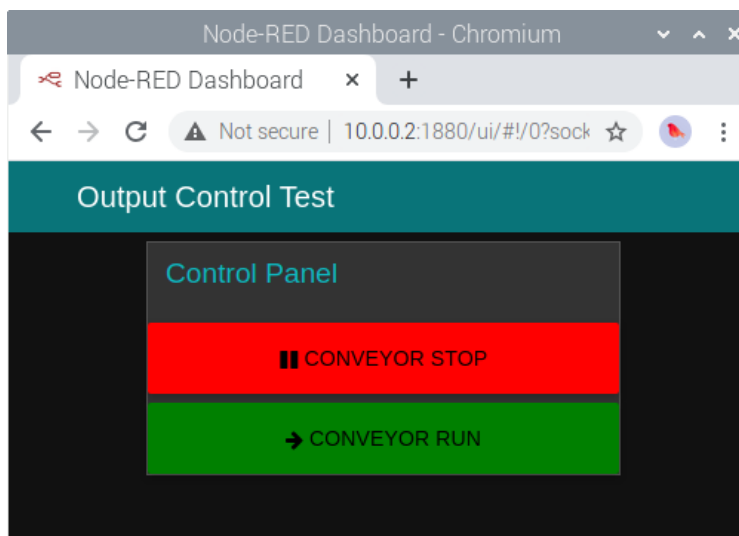


Figure 6-53: Dashboard with Icons

Looking more professional by the second. You can also increase the size of the icon, eliminate the label and resize the button so that it looks more like a front panel button as in the figure below.

## Learning Kit Workbook (version 1.3)

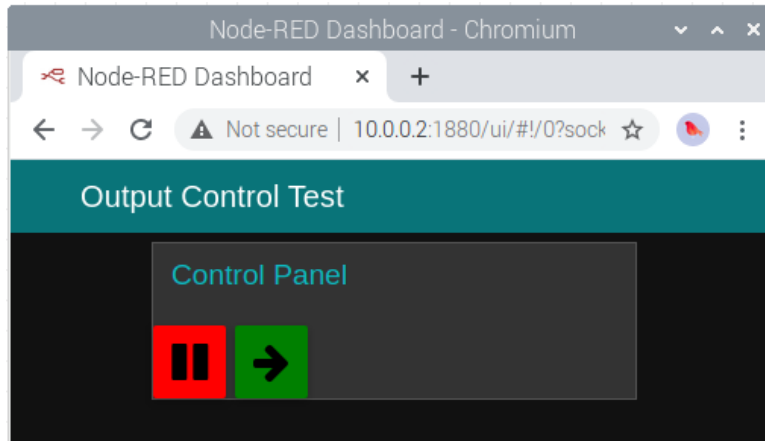


Figure 6-55: Relay Control with Large Icons

If you use Font Awesome icons, it is easy to control the size as shown in the configuration of the “Relay OFF” node below.

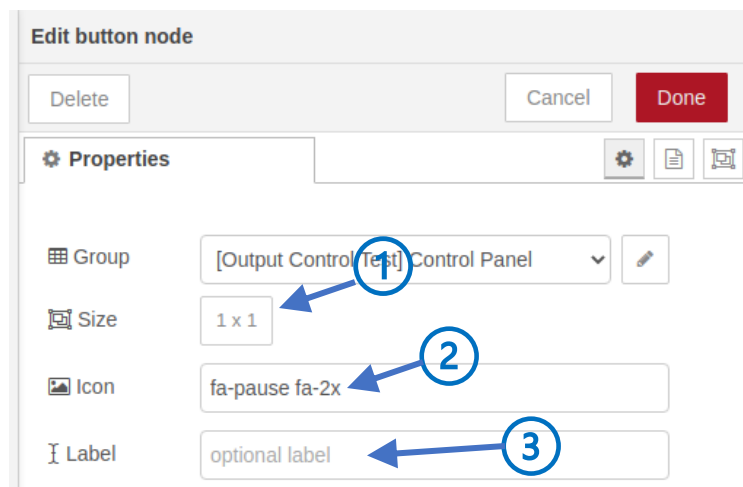


Figure 6-54: Relay OFF Node Configuration for Large Icons

- Reduce the size of the button widget to 1x1 ①.
- Then at the string “fa-2x” to the Icon box ②. This is an indication, unique to Font Awesome, that displays the selected icon twice as large.
- Finally, delete the name of the label, so that only the icon is visible on the button widget ③.
- Click Done!
- Rinse and repeat for the “Relay ON” node as shown below.

## Learning Kit Workbook (version 1.3)

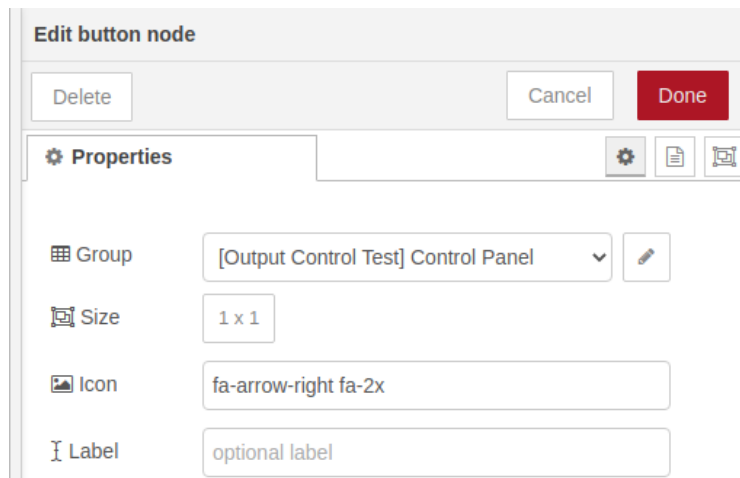


Figure 6-56: Relay ON Node Configuration for Large Icons

### Deploy and Test!

If your new control panel looks like Figure 6-55 good work. You are now a Commander of the Relays  
With this knowledge it is now PUZZLE TIME!

**Puzzle # 8 - Buttons for the Motor Controller.** Go back to the motor controller of Figure 6-17. Instead of using OPTO buttons revise your flow to control the motor using dashboard Button widgets. Because Button widgets only send a message when the mouse key is released you will not be able to control the motor in the same way. Instead, when you click the “Forward” button the motor will run until you click either “Reverse” or “Stop”.

**Puzzle # 9 - Buttons for the Combination Lock** – Go back to the combination lock example and set up four icon buttons for the numbers and a CLEAR button in place of the on-board pushbutton. Set up a dashboard Text widget to show the state of the safe, “LOCKED” or “UNLOCKED”.

- Make your safe more secure by adding buttons for 1 to 9 and extend the combination to 6 digits.
- Do some Internet research and see if you can figure out how to change a button icon dynamically. Add a button that has no function except to show the locked or unlock status of the safe using lock and unlock icons. Change the button color when the safe is unlocked.
- See if you can arrange the icons in a nice pattern. You might need to use one *ui\_group* for the number buttons and another for the control and status buttons (i.e., Clear, status).

**Puzzle # 10 Water the Lawn.** The relays on the Learning Card can control a valve to lawn sprinklers. The valves cost about \$10 and the 24VAC transformer about \$15. An inject node can be programmed to send a message at a specific time (in the Repeat dropdown select “at a specific time”). See if you can set up a flow to water your lawn or garden at 10:00 AM for 10 minutes on Tuesday, Thursday and Saturday.

**Puzzle # 11 But Not When It Rains!** Modify your sprinkler controller so that it does not water the lawn when it has rained within the last two days. Look at the flow from Chapter 4 related to determining the temperature at your location for some ideas about how to figure out when it rained last.

## Learning Kit Workbook (version 1.3)

Go Forward Commander of the Relays!

Time to crawl into the digital world and learn about sensing and controlling voltage levels.

### Chapter 7 – Reading Voltage Levels – COMING SOON!

Hints – If You Need Them

Combination Lock Configurations – Finite State Machine Flow

Below are some node configurations related to the flow shown in Figure 6-39, the Finite State Machine for the combination lock.

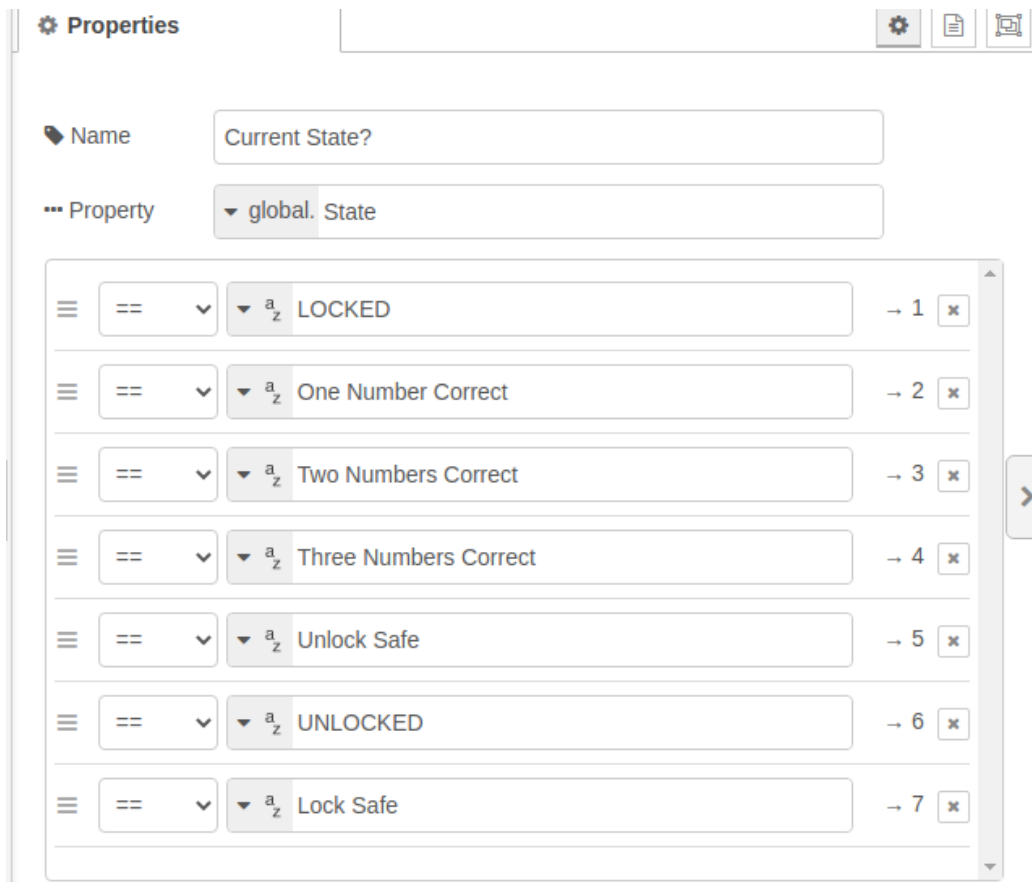


Figure 6-57: Current State Evaluation Node Configuration

In the flow above the current state held in global.State is used to select an output port for the message. There is one output port for each of the possible states. States are encoded as strings because this makes debugging easier. For example, if you look at the state variable you will see immediately what it is and what it means. Also if you put the mouse cursor on the output port of this node you can see which state the port is associated with.

## Learning Kit Workbook (version 1.3)

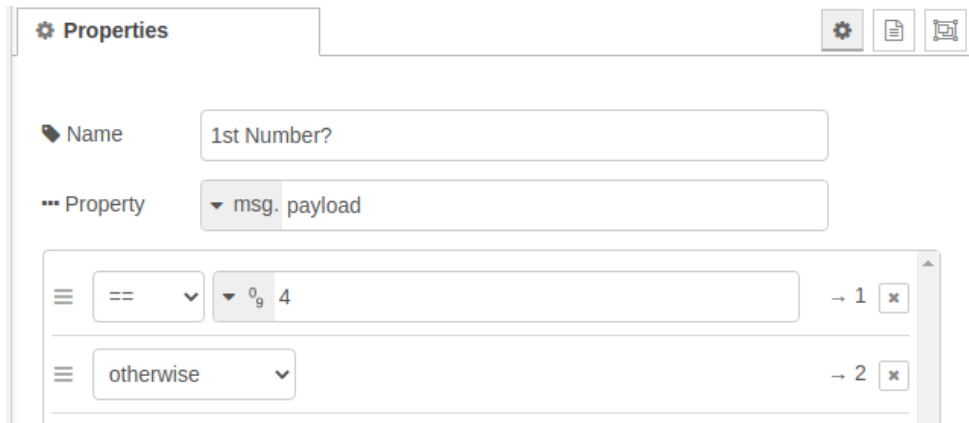


Figure 6-59: "1st Number?" Node Configuration

The switch node above evaluates the current event, which is in the payload, and determines how to advance the finite state machine. The other nodes are similar.

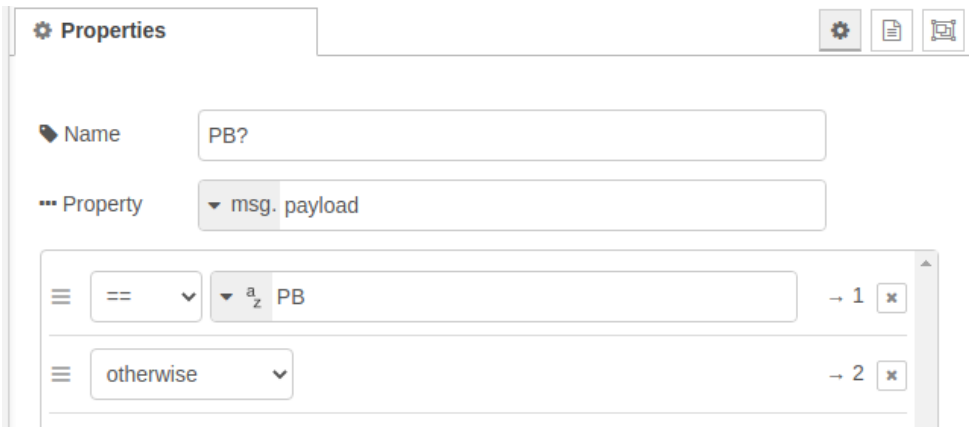


Figure 6-58: "PB?" Node Configuration

The Switch node named "PB?" looks for the event caused by pushing the on-board pushbutton, which is encoded as the string "PB".

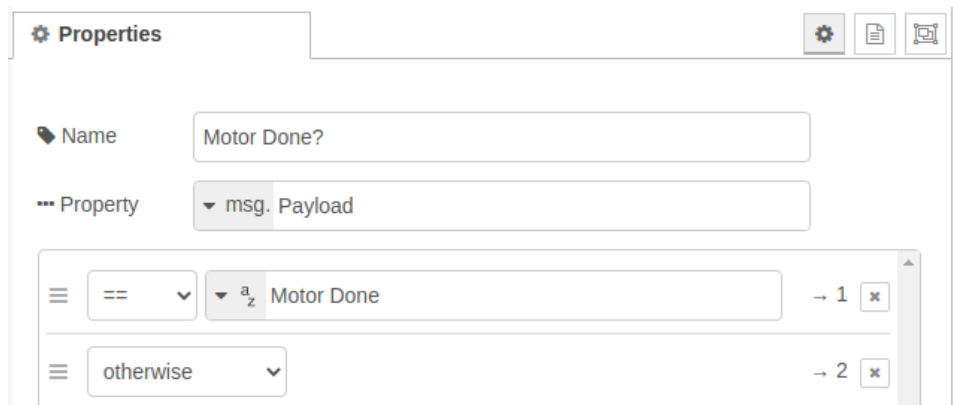


Figure 6-60: "Motor Done?" Node Configuration

When the motor completes its three second cycle it sets the event "Motor Done" in the payload and sends it to the input of the "Current State?" node (see below). In the node above this event is recognized to advance the finite state machine to the next state.

## Learning Kit Workbook (version 1.3)

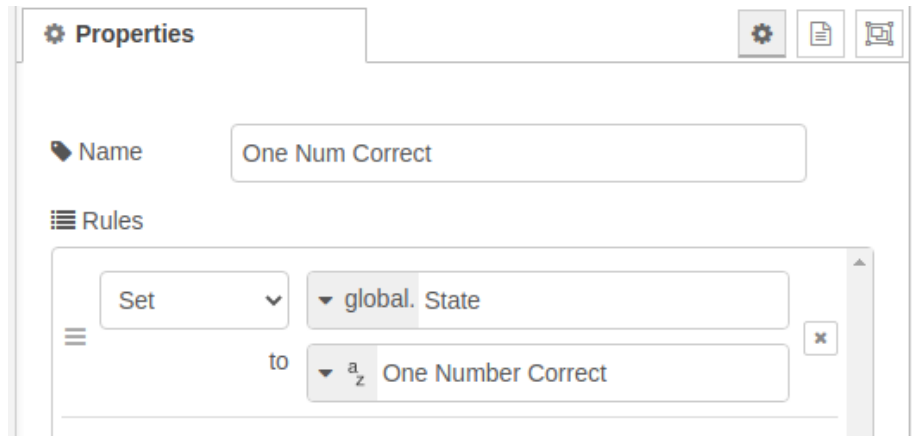


Figure 6-61: "One Num Correct" Node Configuration

All of the nodes used to set the next state in Figure 6-39 are similar to the node shown above. This node simply sets *global.State* to the string representing the next state. In this case once the first number entered correctly the state is set to "One Number Correct".

### Combination Lock Configurations – Action Nodes

Action nodes perform actions based on the transition to a new state. Most of the action nodes in the combination flow should be self-evident. The node configuration shown below may not be.

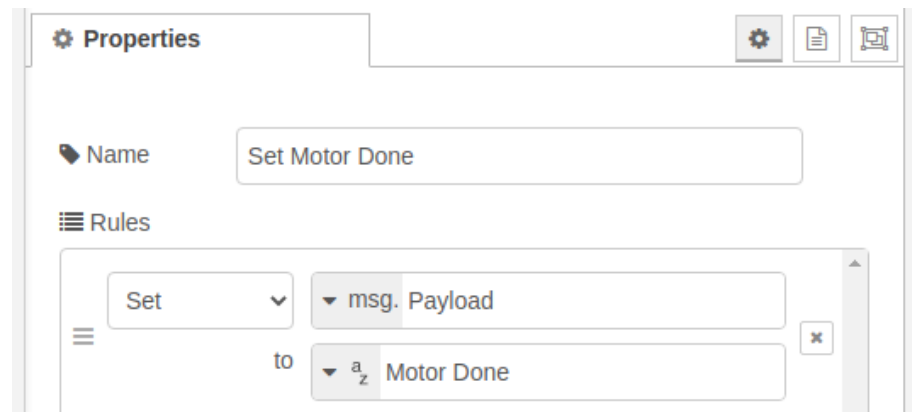


Figure 6-62: "Set Motor Done" Configuration

In the combination lock flow most of the state transitions take place based on external events, such as pushing the OPTO inputs. However, the operation of the motor is an example of a type of an internally generated event. When the motor starts it runs for three seconds. Once the time expires the node above generates a new event to advance the finite state machine from either "Unlock Safe" to "UNLOCKED" or from "Lock Safe" to "LOCKED".



## Coming Soon

### Chapter 7 – It's an Analog World – Voltage In

- Analog User Interface - Gauges and Graphs
- Sensing Temperature and Light
- Temperature controller
- 0-10 Volt Control

### Chapter 8 - Controlling the Analog World – Voltage Out

- Controlling Analog Outputs from a User Interface

### Chapter 9 Chapter 9 – Distant Learning - Current Sensors

### Chapter 10 Chapter 10 – Spooky Action at Distance - Controlling Current Actuators

### Chapter 11 Chapter 11 - Talking to Others – RS-485 Interfaces

### Chapter 12 Chapter 12 – Proportional Control – Motors

### Chapter 13 Chapter 13 – Servo Motor Control

- Stock Market Display

## Learning Kit Workbook (version 1.3)

### Appendices

Appendix 1 – Interface Electronics – TBD

Appendix 2 – I/O Learning Card Python and Command Line Interfaces - TBD

Appendix 3 – Testing the I/O Learning Card – TBD

Appendix 4 – Node-RED Messages - TBD

Appendix 5 – Simple Node-RED Types – TBD

### Updates History

<b>Version</b>	<b>Date</b>	<b>Reason</b>
Preliminary 0.0	26 Oct 2021	Original
Preliminary 1.0	05 Nov 2021	Add LKit Key node access, correct typos
Preliminary 1.1	09 Feb 2022	Add Chapter 4 - LEDs
Preliminary 1.2	22 Feb 2022	Add Chapter 5 – Opto-Isolated Inputs
Preliminary 1.3	20 March 2-22	Update Chapter 5 with User Interface Add Chapter 6 including User Interface