Part 57
-
Node-Red

Version: 2020-03-11

Installing Node-Red

Node-Red is a node application and before you install Node-Red you will need to first install Node.js. Here are the instructions:

Debian contains a version of Node.js in its default repositories. At the time of writing, this version is 10.15.2, which will reach end-of-life on April 1, 2021. At this date it will no longer be supported with security and bug fixes. If you would like to experiment with Node using an easy-to-install, stable, and long-term option, then installing from the Debian repo may make sense.
To get Node.js from the default Debian software repository, you can use the apt package manager. First, refresh your local package index:

```
sudo apt update
```

If there are upgradeable package, run

```
sudo apt -y upgrade
```

Then install the Node.js package, and npm, the Node Package Manager:

```
sudo apt -y install nodejs npm
```

To verify that the install was successful, run the node command with the -v flag to get the version:

```
node -v
v10.15.2
```

If you need a more recent version of Node.js than this, the next two sections will explain other installation options.
Then you can install Node-Red on Windows and Linux (including raspberry pi) using:

```
npm install -g --unsafe-perm Node-Red
```

Using the -g option adds the Node-Red command to your path.

Node-Red on Pi Notes:
Node-Red is already installed on Raspberry Pi but npm (node package manager) isn't.
You may need to upgrade Node-Red which you can do by doing an install using npm but you will need to install npm first. This you can do using:

```
sudo apt update
sudo apt -y install build-essential
sudo apt install npm
```

You can check the versions of npm, node and Node-Red using:

```
npm -v
node -v
node-red –help
```

Installing Node-Red on Raspberry Pi
If you are using Raspbian, then you must have Raspbian Jessie as a minimum version.
A script to install Node.js, npm and Node-Red onto a Raspberry Pi is provided by the Node-Red team. The script can also be used to upgrade an existing install when a new release is available.

Make sure curl is installed

```
apt -y install curl
```

Running the following command will download and run the script.

```
bash <(curl -sL
https://raw.githubusercontent.com/Node-Red/linux-installers/master/deb/
update-nodejs-and-nodered)
```

This script will:

- remove the pre-packaged version of Node-Red and Node.js if they are present
- install the current Node.js LTS release using the NodeSource. If it detects Node.js is already installed from NodeSource, it will ensure it is at least Node 8, but otherwise leave it alone
- install the latest version of Node-Red using npm
- optionally install a collection of useful Pi-specific nodes
- setup Node-Red to run as a service and provide a set of commands to work with the service

Node-Red has also been packaged for the Raspbian repositories and appears in their list of 'Recommended Software'. This allows it to be installed using

```
apt install nodered
```

and includes the Raspbian-packaged version of Node.js, but *does not* include npm.
While using these packages is convenient at first, we strongly recommend using our install script above instead.

Running locally
As with running Node-Red locally, you can use the Node-Red command to run Node-Red in a terminal. It can then be stopped by pressing Ctrl-C or by closing the terminal window.
Due to the limited memory of the Raspberry Pi, you will need to start Node-Red with an additional argument to tell the underlying Node.js process to free up unused memory sooner than it would otherwise.
To do this, you should use the alternative node-red-pi command and pass in the max-old-space-size argument.

```
node-red-pi –max-old-space-size=256
```

you should something like

```
5 Jan 18:49:19 - [info]

Welcome to Node-RED
===================

5 Jan 18:49:19 - [info] Node-RED version: v1.0.3
5 Jan 18:49:19 - [info] Node.js  version: v12.14.0
5 Jan 18:49:19 - [info] Linux 4.19.75-v7+ arm LE
5 Jan 18:49:20 - [info] Loading palette nodes
5 Jan 18:49:22 - [info] Settings file  : /root/.node-red/settings.js
5 Jan 18:49:22 - [info] Context store  : 'default' [module=memory]
5 Jan 18:49:22 - [info] User directory : /root/.node-red
```

```
5 Jan 18:49:22 - [warn] Projects disabled :
editorTheme.projects.enabled=false
5 Jan 18:49:22 - [info] Flows file     : /root/.node-red/flows_MeRasPi3B-
Test.json
5 Jan 18:49:22 - [info] Creating new flow file
5 Jan 18:49:22 - [warn]

---------------------------------------------------------------------
Your flow credentials file is encrypted using a system-generated key.

If the system-generated key is lost for any reason, your credentials
file will not be recoverable, you will have to delete it and re-enter
your credentials.

You should set your own key using the 'credentialSecret' option in
your settings file. Node-RED will then re-encrypt your credentials
file using your chosen key the next time you deploy a change.
---------------------------------------------------------------------

5 Jan 18:49:22 - [info] Server now running at http://127.0.0.1:1880/
5 Jan 18:49:22 - [info] Starting flows
5 Jan 18:49:22 - [info] Started flows
```

At the top of the screen you can see the version of node and Node-Red that you are using and the location of the flows..
At the bottom of the screen it tells you the url to use to access the Node-Red web admin interface which you use to create your flows.

Running as a service
The install script for the Pi also sets it up to run as a service. This means it can run in the background and be enabled to automatically start on boot.
The following commands are provided to work with the service:

- node-red-start - this starts the Node-Red service and displays its log output. Pressing Ctrl-C or closing the window does *not* stop the service; it keeps running in the background
- node-red-stop - this stops the Node-Red service
- node-red-restart - this stops and restarts the Node-Red service
- node-red-log - this displays the log output of the service

You can also start the Node-Red service on the Raspbian Desktop by selecting the Menu -> Programming -> Node-Red menu option.

Autostart on boot
If you want Node-Red to run when the Pi is turned on, or re-booted, you can enable the service to autostart by running the command:

```
sudo systemctl enable nodered.service
```

To disable the service, run the command:

```
sudo systemctl disable nodered.service
```

When running on start up you can stop it using the `node-red-stop` command and restart using `node-red-start` command.
If you want to load Node-Red on startup with a different settings file than the settings.js you will need to edit the file `/lib/systemd/system/nodered.service`.
Un-comment the line `ExecStart` if not already done and change the

```
Environment="NODE_RED_OTPIONS=-v"
```

line to something like:

```
Environment="NODE_RED_OPTIONS=-s /home/pi/.node-red/mysettings.js"
```

then reboot.

Using Node-Red

Opening the editor

Once Node-Red is running you can access the editor in a browser.
If you are using the browser on the Pi desktop, you can open the address:

```
http://localhost:1880
```

We recommend using a browser outside of the PI and pointing it at Node-Red running on the Pi. However you can use the built in browser and if so we recommend Chromium or Firefox-ESR and not Epiphany
When browsing from another machine you should use the hostname or IP-address of the Pi

```
http://<hostname>:1880
```

You can find the IP address by running hostname -I on the Pi.

Node-Red Settings

Node-Red uses a settings file called settings.js. It is located in the `/usr/lib/node-modules/node-red/` folder by default.
The settings.js file is copied from the /usr/lib/node-modules/Node-Red/ folder to your .node-red folder in your home directory when you start Node-Red, and there is no settings.js file present. You can create your own settings file and customise it either by copying the existing file.
To use your own settings file use the -s switch when starting Node-Red e.g.

```
node-red-pi -s mysettingsfile.js
```

Important: If you place a settings.js in the .node-red folder then that will be used by default.

The documentation takes your through the settings file in detail.

The .node-red Folder

User settings and flows are stored in the `.node-red` folder. This folder is located in the users home folder.
E.G For user steve → /home/steve/.node-red

Node-Red Flows Files and Storage

Node-Red flows are stored in a .json file in the .node-red folder.
By default this file is called flows_machine_name.json. For example my Pi is called MeRasPI3B-Test and the default flows file is called flows_MeRasPi3B-Test.json.

Useful Command line Options

Node-Red has a very limited number of command line options. Use:

```
node-red-pi -h

Node-RED v1.0.3
Usage: node-red [-v] [-?] [--settings settings.js] [--userDir DIR]
                [--port PORT] [--title TITLE] [--safe] [flows.json]

Options:
  -p, --port     PORT  port to listen on
  -s, --settings FILE  use specified settings file
      --title    TITLE process window title
  -u, --userDir  DIR   use specified user directory
  -v, --verbose        enable verbose output
      --safe           enable safe mode
  -?, --help           show this help

Documentation can be found at http://nodered.org
```

The -v switch turns on verbose mode.
The -p switch will let you change the port that Node-Red admin interface uses.
You can use this option to quickly and easily run multiple instances of Node-Red.
In version .20 a safe mode has been added to allow you to start Node-Red without deploying any flows.
This is very useful if a flow is stopping Node-Red from running.

```
node-red-pi –safe
```

To use a different flow file than the default you can use

```
node-red-pi myflow.json
```

Because you haven't specified a settings file it uses the default settings file.

Running Multiple Node-Red instances
It is possible to run multiple Node-Red instances on a single machine.
To do that you will need to give each instance a different admin port and also a different user directory or alternatively create and use a new settings file.
You can do this using the -p and -u switches when starting Node-Red e.g

```
node-red-pi -p 1881 -u user1
```

the admin port is 1881 and the user directory is user1 the flow file name uses the default flow name.
Note: If you don't use a different user directory then they will all use the same flows file which could result in conflicts.
If you create a new settings file e,g mysettings.js you can then edit the line starting with flows and remove the two forward slashes.

```
// flowFile: my'flows.json'
```

You could also change the port by editing the line near the top of the file

```
uiPort: process.env.PORT || 1880
```

to

```
uiPort: process.env.PORT || 1888
```

to use port 1888

You then use the command:

```
node-red -s mysettings.js
```

to start the flow.

You could also use the command

```
node-red -s mysettings.js -p 1889
```

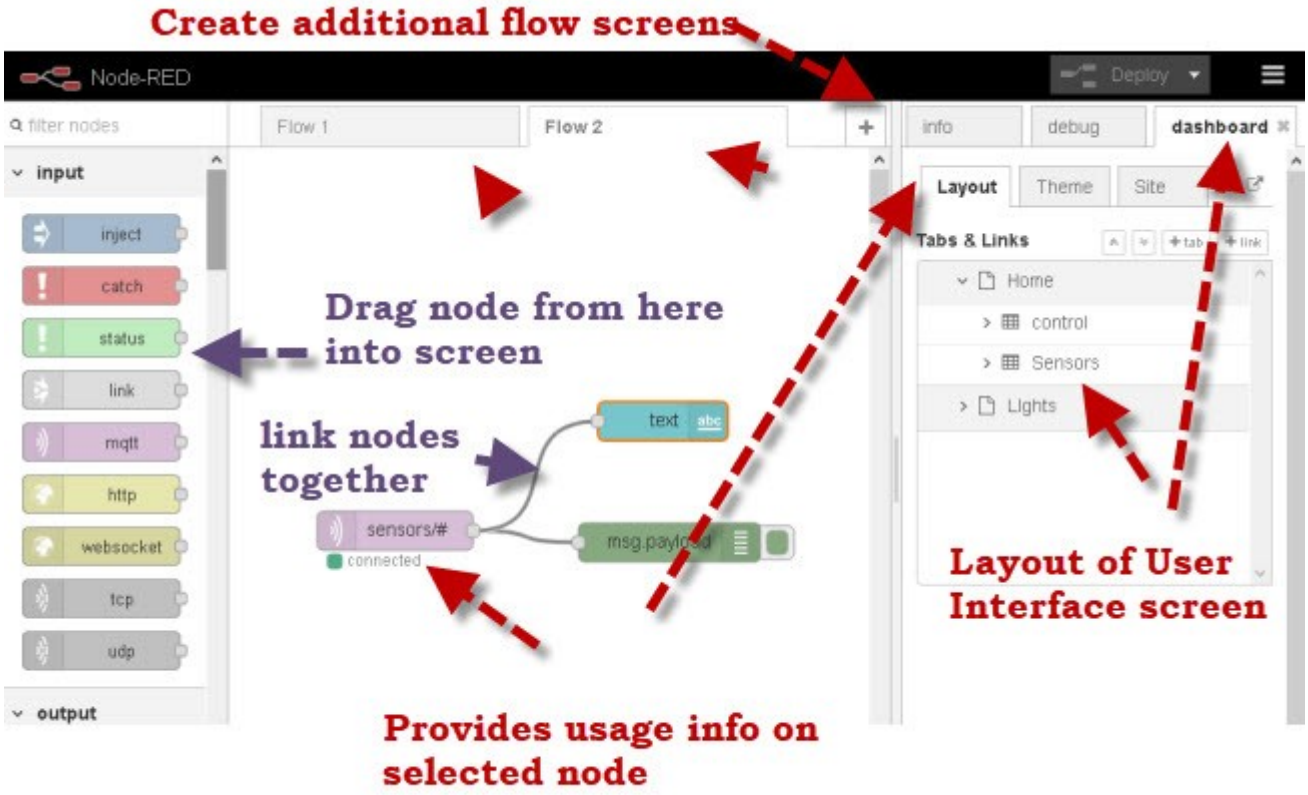
to use a different port.

Creating Flows - Node-Red Admin Basics
To administer Node-Red you will need to go the admin url.
The admin url is the machine name or IP address followed by the port number. e.g

- 127.0.0.1:1880/ if you are running the browser on the same machines as Node-Red.
- localhost:1880/ –
- steve-laptop:1880/ when running Node-Red on a remote machine
- 192.168.1.154:1880/

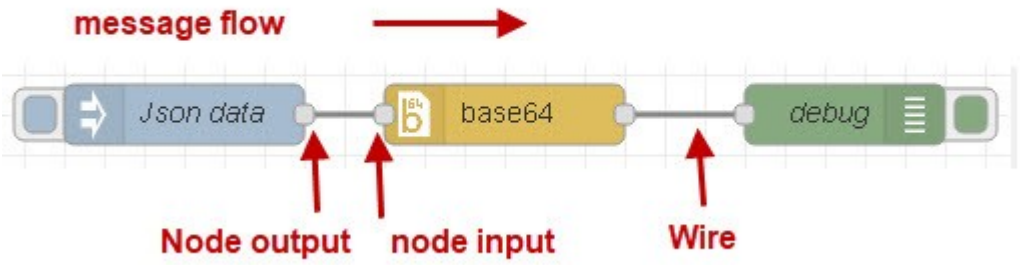
I currently run Node-Red on a Raspberry Pi and use a second machine (Windows 10) to create flows.

Using The Node-Red Admin User Interface (UI)
When you open the Node-Red admin screen for the first time you should start with an empty work space as shown in the earlier screen shot above.
The default view is a three column layout with nodes on the left ,the flows work space in the middle and a third column on the right.
The third column or output pane has two or three tabs – info, debug and dashboard (if installed).



Node-Red Flows
A flow is a collection of nodes wired together and functioning as an application or program. The Node-Red workspace supports multiple flows which all share the same nodejs event loop.

Node-Red Nodes
Nodes are the basic building block of Node-Red.
A node is effectively a software block that processes messages.
A node can have inputs and outputs which enable messages to be passed between nodes.
An input can accept connections from multiple nodes and an output can output to multiple nodes.



The nodes in the left pane are arranged in categories. A Node-Red installation will contain core nodes and the nodes are organized into groups. There is a

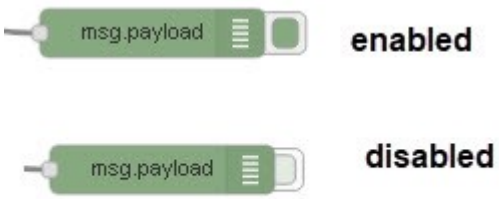
- input group
- output group
- functional nodes
- dashboard or display nodes

Other groups can be created when new nodes are installed. Each node has a well defined function and contains its own data. You can install additional nodes using the Admin Interface.

Frequently Used Nodes
There are two nodes that you will find yourself using quite often. They are the debug node and the inject node.

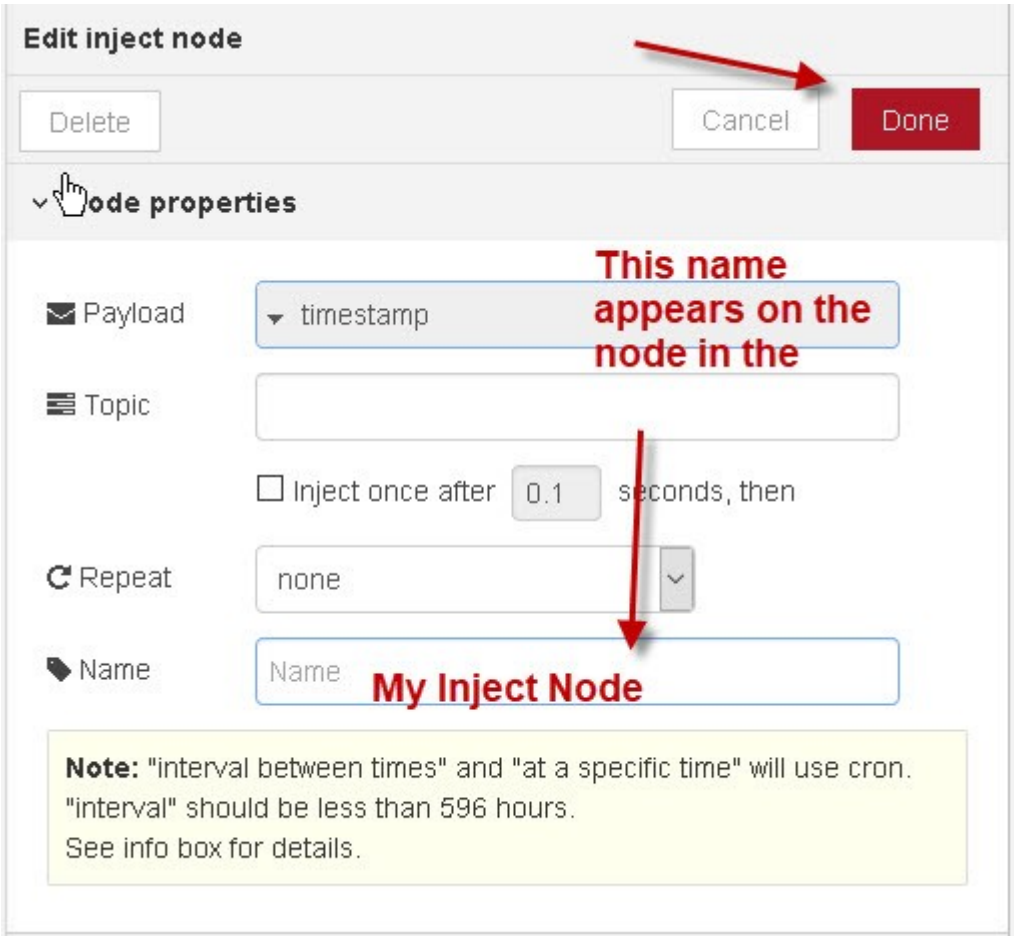
The Inject Node
Node-Red is an event based system and something must happen to create an event. The inject nodes will fire a message into the next node and is used to trigger a flow.

Debug Node
The debug node is used for displaying output for debugging purposes. By default it will display the message payload but can be configured to display the entire message object. You should notice a green box next to the debug node clicking this will toggle to debug node output on/off. Dark green is on and light green is off.

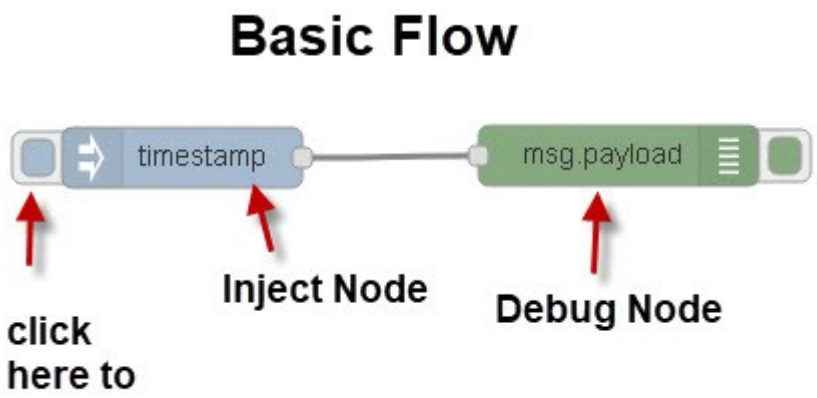


Note: On Node-Red version 1 the default setting is disabled. So if you aren't seeing debug messages check the status of the debug node.

Node Properties
If you double click on the node on the canvas then you can edit the node properties.
All Nodes have a name property. This is the name that is displayed on the canvas.



A Basic Flow
You can wire the inject node into the debug node to create a basic flow as shown below.

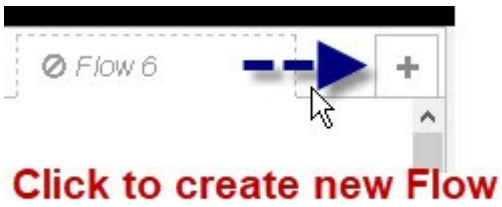


The above flow will inject a unix timestamp into the debug node which can be viewed in the debug tab in the far right pane.

Deploying Flows
When you start Node-Red then all currently enabled flows are automatically started. If you edit or create a new flow you will need to deploy it using the deploy button. The Deploy button on the top right change from grey to maroon when changes have been made to a flow to indicate that in it needs to be deployed.

You can choose to deploy all flows in the workspace, modified flows,modified nodes or restart flows.

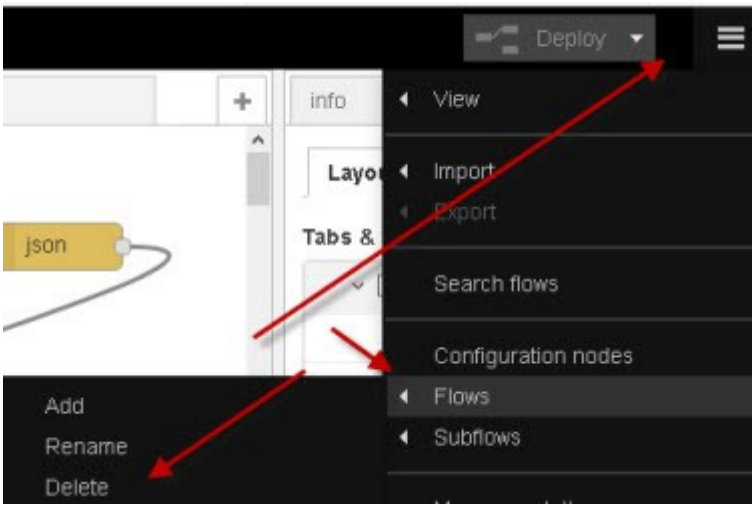
Working With Flows
The workspace consists of all of your flows. Along the top of the workspace pane are tabs that are used to open previously created flows. You can create new Flows by clicking on the Plus button on the top right.



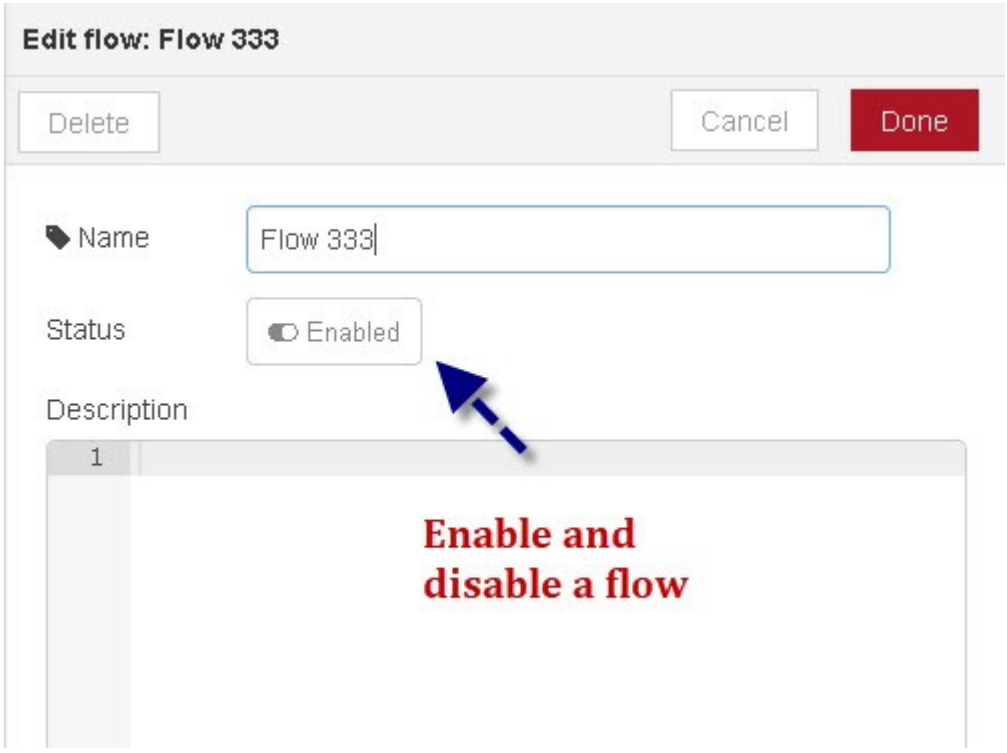
When you create a new empty Flow it is given a default name and is enabled by default. Generally you create a new flow when the flow needs to do a distinct task. To add nodes to a flow drag them from the node palette on the left into the active flow in the middle pane.
You can link the nodes together by dragging a wire between them. A collection of linked nodes is called a flow.

Note: All flows in the workspace share the same node.js event loop.

Renaming, Deleting and Disabling Flows
A flow called Flow1 is pretty meaningless and so you should rename it to something more meaningful. You can access the flow properties from the settings tab. You need to select the flow in the workspace and then. to open the flow go to the menu (top right)  and select flows> rename flow, delete, add.

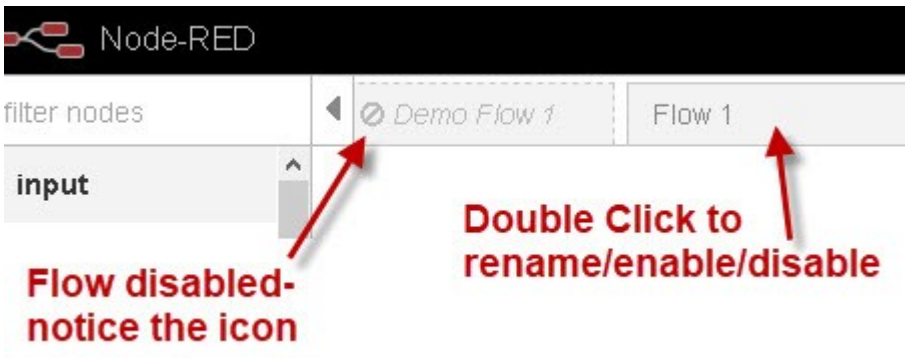


If you select rename then a window opens that displays the flow properties. You can rename the flow, and there is also a toggle to enable/disable it, and also a button to delete it.



There is also a delete button to delete the flow on the top left.

Note: enable button moved to bottom right in version 1.0 You need then to redeploy for the setting to take effect. You can also access this setting by double clicking on the flow name tab:



You should notice the icon in the flow tab that indicates that the flow is disabled.

Flow Files -Storage and Backup
Flows that you create are stored by default in a file in the .node-red directory of your home folder in a file called `flows_servername.json`. The file is created when you run Node-Red for the first time, and is loaded each time that you run Node-Red.
EG. for my Raspberry Pi the file is flows_MeRasPi3B-Test.json.
The file will contain all of the flows in the workspace.
However it is not the only file that is created. You will also see these files

- flows_MeRasPi3B-Test .json.backup – This is a backup of the flows_MeRasPi3B-Test .json file, and functions like a word backup.
  When you do a deploy the old flows_MeRasPi3B-Test .json file is moved to the backup file and the flows_MeRasPi3B-Test .json is overwritten.
  This means that you can recover from a mistake if you notice it after 1 deploy by using the flows_MeRasPi3B-Test .json.backup file.
- flows_MeRasPi3B-Test_cred.json. – This file contains login data used by some nodes. If it is not present the flows will still work, but you will need to add the information back in to any nodes that need it.
- flows_MeRasPi3B-Test_cred.json.backup – Backup of flows_MeRasPi3B-Test_cred.json and functions the same way as the other backup.

Flow Backup
It is a good idea to backup your flows files on a regular basis. The automatic backup should only be considered a temporary one.
To do a complete backup then make a copy of the four files mentioned above:

- flows_MeRasPi3B-Test .json
- flows_MeRasPi3B-Test .json.backup
- flows_MeRasPi3B-Test_cred.json
- flows_MeRasPi3B-Test_cred.json.backup

Note: you will need to change the names to match your flows files.

Moving Your Flows to Another Machine
You can copy all of the flows by copying the flows_newmachinebane.json. file.
To use them on another machine change the file name to flows_newmachinebane.json.
The only file you really need is the flows_MeRasPi3B-Test.json file however you should also copy the flows_MeRasPi3B-Test_cred.json file.

Creating Additional and Multiple Flow Files
You can created additional flow files for storing different workspaces.
To use another flows file you can create a new settings file and edit it.
As an example if I wanted to create a separate flows file to store client flows I would :

- copy the old settings file to a new one called client_settings.js
- Edit the settings file to point to the new flows file and remove the two forward slashes.
- // flowFile: clientflows.json',
- Start Node-Red using

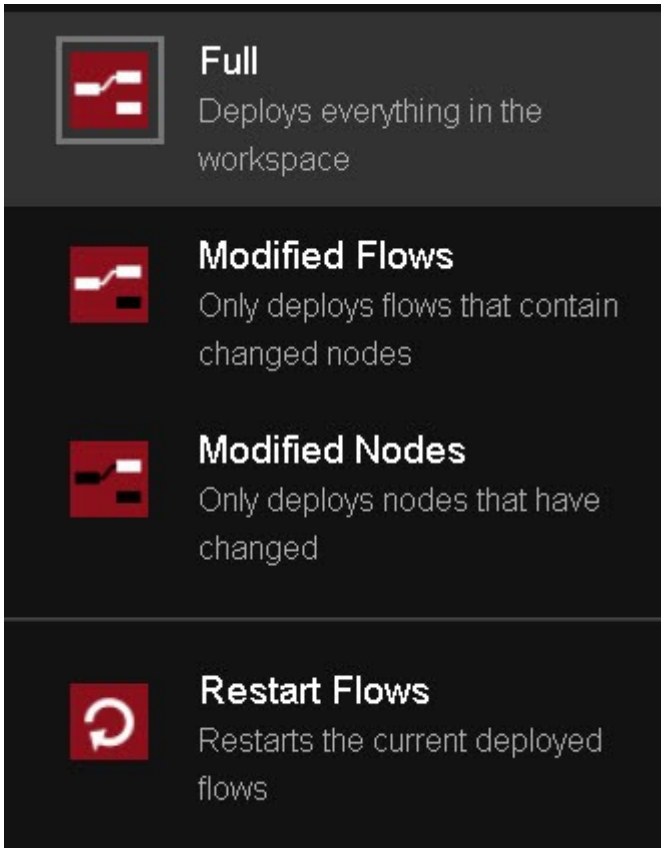
```
node-red-pi -s client_settings.js
```

You could also use the new flows file without creating a new settings file by using the command option which is my preferred option.

```
node-red-pi clientflows.json
```

Notice you don't need a switch.

Deploying Node-Red Flows
When you come to deploying a flow you have four (Node-Red v0.2)options as shown below:



Although the choices appear obvious the effect of the options isn't and can cause some confusion.

- Inject nodes
- Context,Flow and Global Variables
- Other Flows in the Workspace

What is Modified?
Firstly we need to look at how Node-Red decides that a node or flow has been modified?
You should notice that when you edit a node then the deploy button goes red which means you can do a deploy. However you can get the same effect by dragging a node across the screen. In this case the node position has been modified but not the node.

Inject Nodes And Storage Variables
Inject nodes are typically used in a flow to initialise it.
Therefore it is important to understand if the deploy type you are using will do that.
In addition most flows will use Node-Red storage variables (context,flow,global) for storing flow data. What happens to these variables when you do a deploy?

Full Deploy
This will re-deploy all flows in the workspace regardless of whether or not they have been modified.
This means that currently running flows are interrupted and restarted.
Result

- Inject nodes will fire.
- Context variables reset
- Flow and Global Variables are not reset

Modified Flows
Only flows that contain modified nodes will be redeployed.
Flows that haven't been modified will continue to run without interruption.
Result

- Inject nodes in that flow will fire.
- Context variables reset
- Flow and Global Variables are not reset

Modified Nodes
Only nodes that have been modified will be redeployed.
Nodes that haven't been modified will continue to run without interruption.
Result

- Inject nodes in that flow will only fire if they have been modified.
- Context variables not reset
- Flow and Global Variables are not reset

Restart Flows
All deployed flows will be redeployed.
This means that currently running flows are interrupted and restarted.
Result

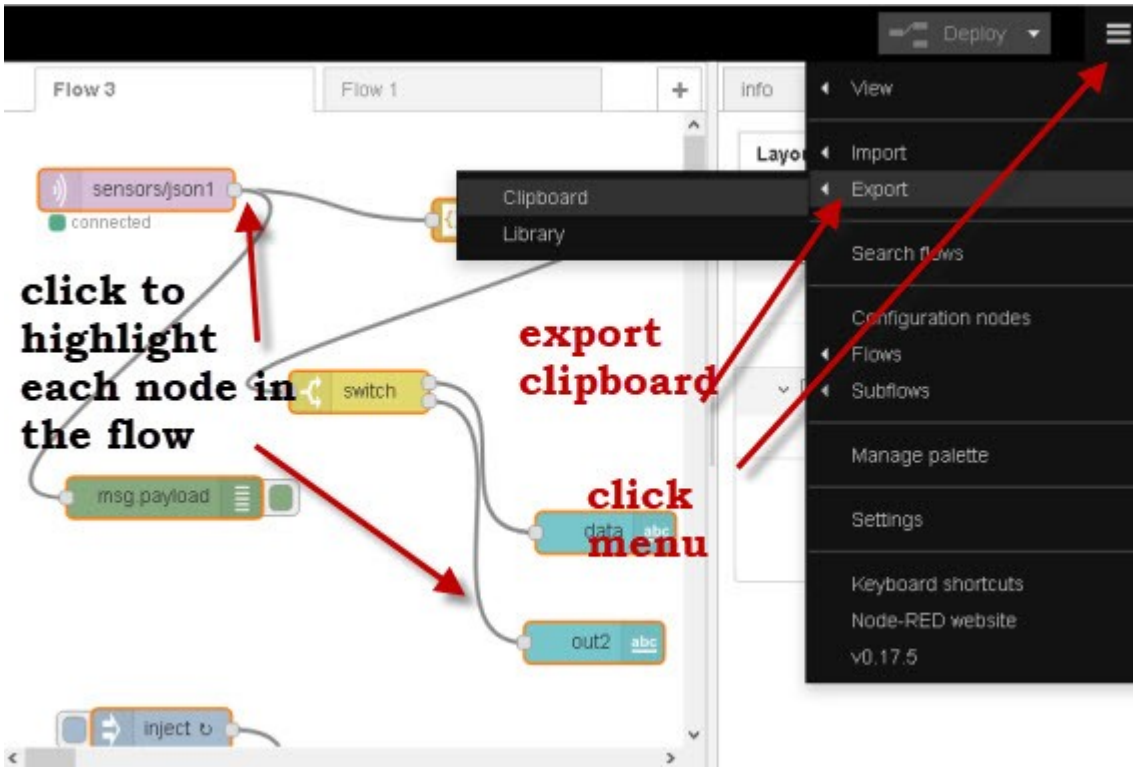
- Inject nodes will fire.
- Context variables reset
- Flow and Global Variables are not reset

Overcoming Flow Problems
If for some reason when you deploy a flow Node-Red stalls or crashes then restarting Node-Red will usually result in a stall or crash as there is something in the deployed flow that is causing it. In this case you can start Node-Red using the safe option which starts Node-Red but doesn't deploy the flows.

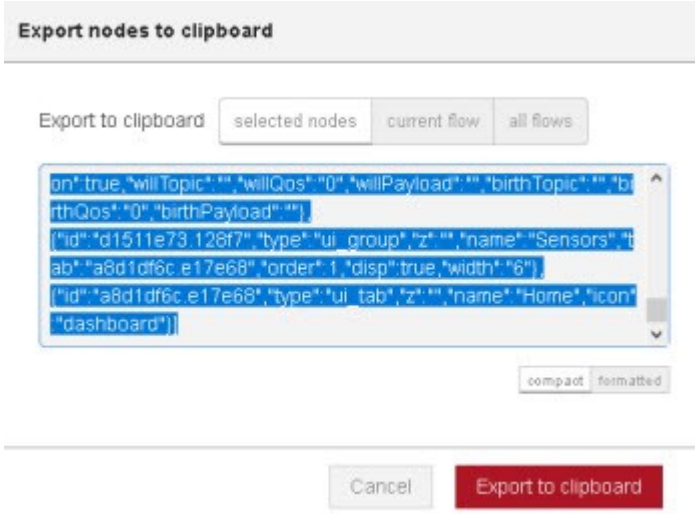
```
node-red-pi --safe
```

How to Export and Import Flows-Node-Red
Node-Red makes it easy to save, share and move flows between computers using the export and import feature. Flows are exported as a JSON file and imported from a JSON file.
You can export parts of a flow or the entire flow or even all flows.

Exporting Flows
Press CTRL and Click each node in the flow that you want to export. The nodes will be highlighted.
Note: Use CTRL+A to select all nodes in the workspace.
Then click on Menu>export>clipboard.

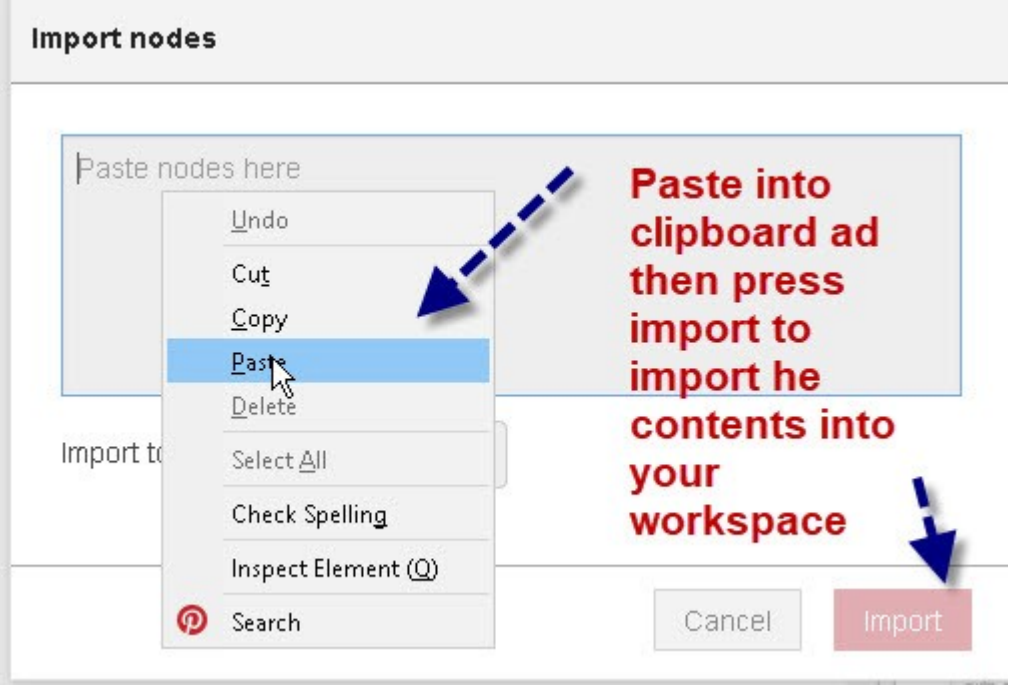


The contents of the export file will be displayed click on export to clipboard.



There are three tabs on the screen with the selected nodes tab selected. You can change it to export the entire flow or all flows in the workspace.
Click export to clipboard and then open a file editor and paste the contents into the file using CTRL+V.
Give the file a name and save it.

Importing A Flow
You will need to open the flow in a text editor, and then copy its contents using CRTL+C.
You then go to Menu>import>clipboard and paste the file into the window.
You can import the flow into an existing flow or into a new flow.
Click the import button to complete the import.



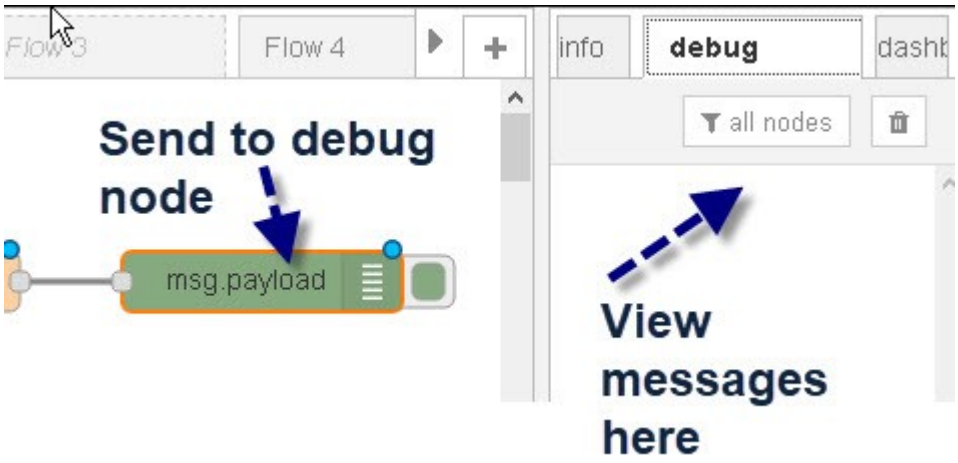
Import Problems
If you find that the import button doesn't change colour and a red line appears around the text box it is probably because what you are trying to import isn't valid JSON.
You can test it by pasting it into this on-line JSON checker https://jsonlint.com/
A confusing issue I found is that my export file verified OK using the JSON checker but failed when doing an import at a remote location.
The problem was due to smart quotes in the export file. I pasted the file into this online smart quotes removal tool https://dan.hersam.com/tools/smart-quotes.html to remove them, and then sent it, and it worked OK.

Using Copy and Paste to Copy Nodes
If you need to copy nodes between flows in the Node-Red Workspace then you can use copy and paste.
This works just like copying text in a word processor. You simply select the nodes you want using the CTRL key to select individual nodes together into a group or CTRL+A to select all nodes in the workspace.
Then use CTRL+C to Copy.
You then move to the place on the workspace or open the workspace where you want to copy the nodes to and use CTRL+V to paste the nodes/flow into the workspace.
Note: This only works for flows in the same workspace.

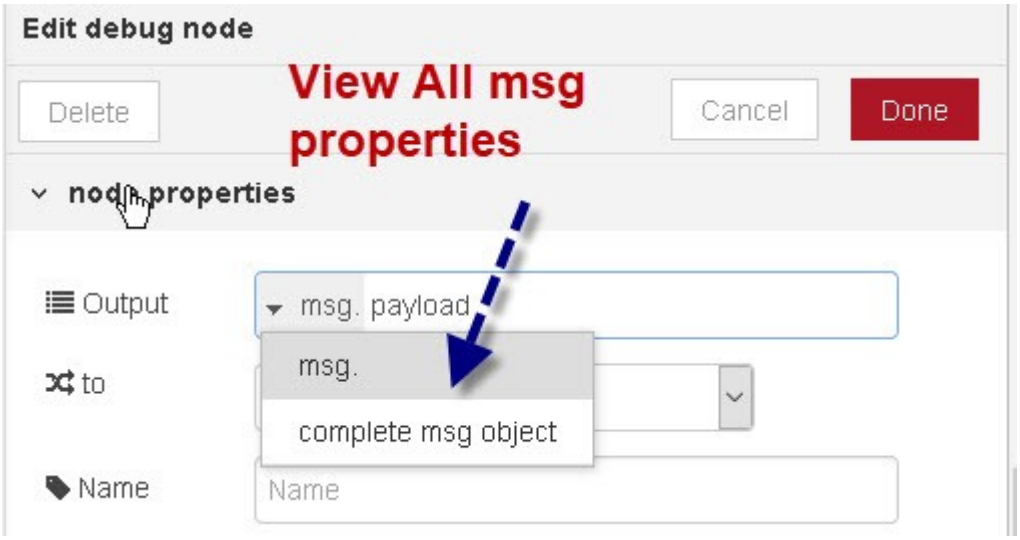
Understanding and Using The Node-Red Message Object
Node-Red flow consists of a series of interconnected nodes.(wired nodes).

All nodes must have an input and can have 0 or multiple outputs.
Nodes exchange data between each other using the msg object**.**
Each node receives the message object from the previous node, and can then pass this message object onto the next node in the flow.

The Msg Object
The Msg Object is a standard JavaScript object and has several existing properties or parts depending on where it originated.
You can see the message properties by sending the msg to the debug node.



By default the debug node will display the msg.payload property, but you can edit the debug node to display all of the message properties (complete message object).



If you Look at the message object properties of the the inject node.

Inject Node Message Object

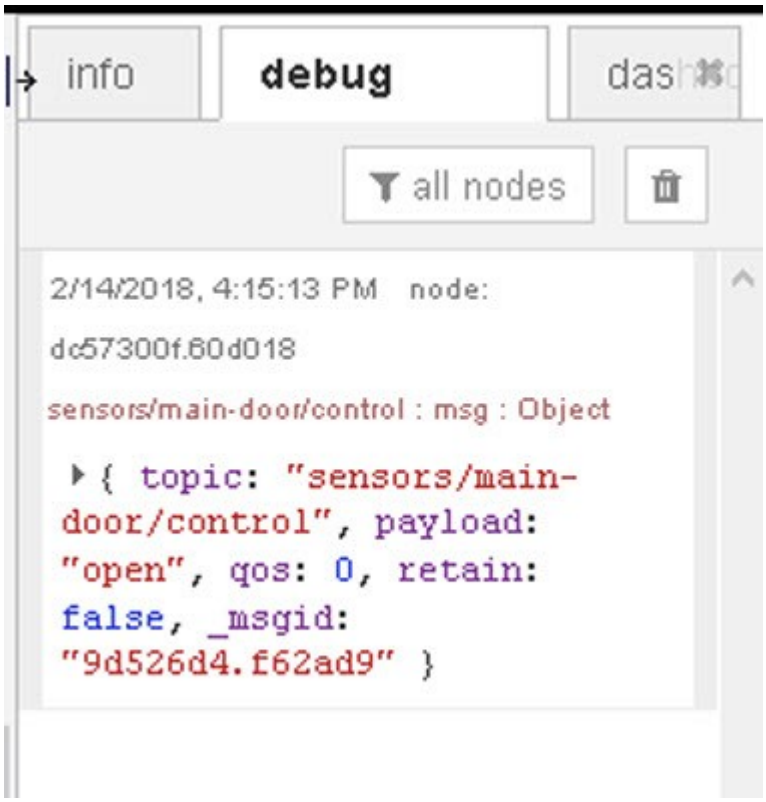
You will see that is has the following properties or parts

- payload
- topic
- _msgid

A msg object that originates from an **MQTT** input node has the following properties:

- payload
- topic
- qos
- _msgid
- retain

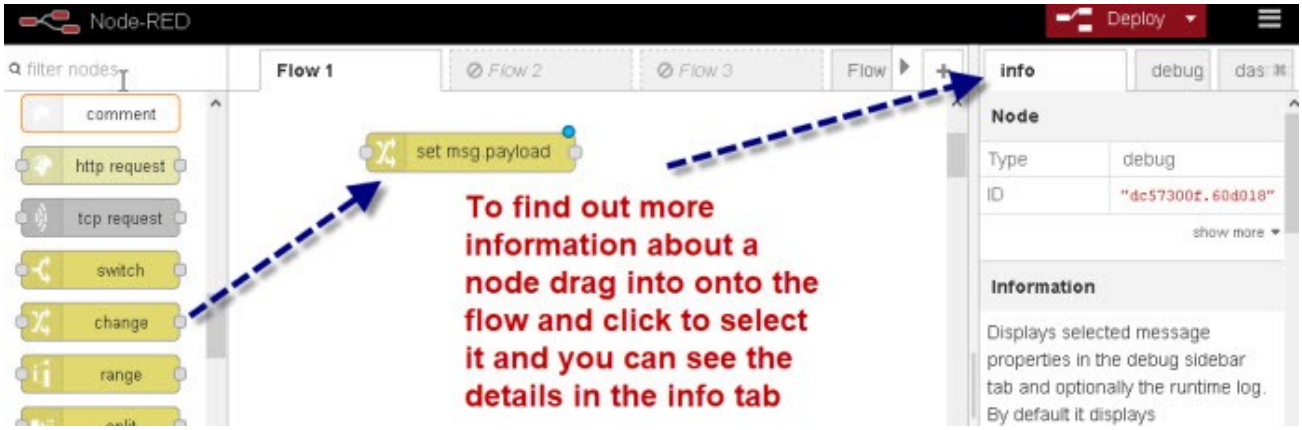
as shown in the screen shot below:



The **_msgid** property is a message identifier added by Node-Red and can be used to track message objects.
Msg object properties can be any valid JavaScript type e.g.

- String
- Integer
- Object
- Boolean

Modifying the msg Object
Node-Red provides various core nodes that can change the messages object without you having to write any JavaScript code.
The main nodes are change, split, join, switch
You can find out more of what they do by dragging them onto the flow and then viewing the info tab associated with them.



There is also the very versatile **function node**, but using it requires writing JavaScript code.

Accessing the msg Properties
You access the msg properties just like you do any JavaScript object. The message payload can be accessed using:

```
var payload=msg.payload;
```

and can be modified using:

```
msg.payload=payload;
```

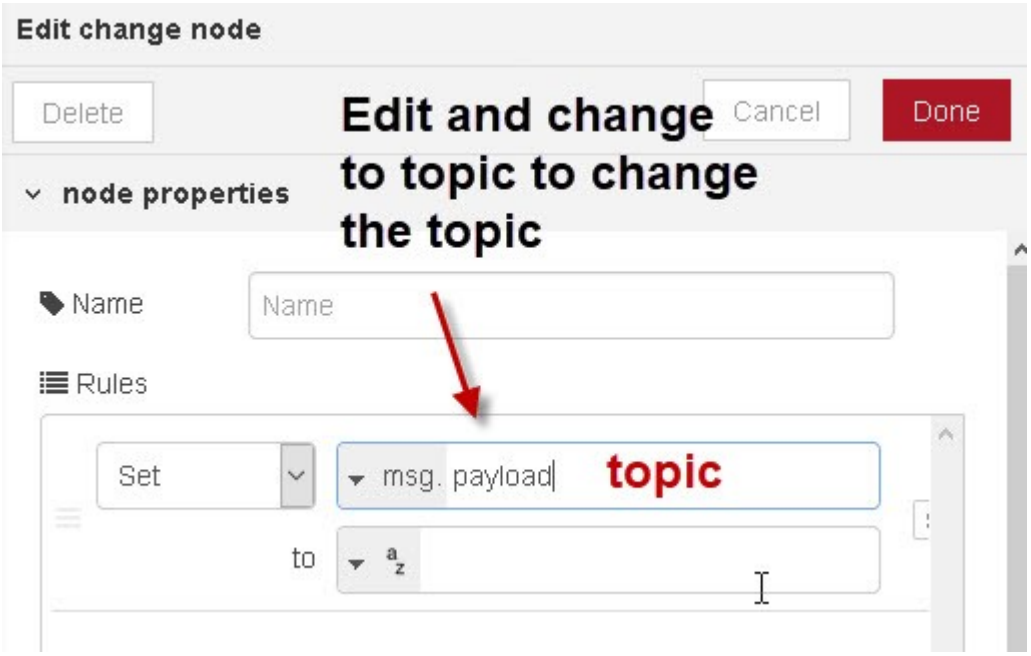
Likewise the message topic can be accessed using:

```
var topic=msg.topic;
```

and can be modified using:

```
msg.topic= topic;
```

If you are using a node like the change node then it defaults to payload but you can change any property by editing the appropriate field as shown below:



Adding Properties to the msg Object
You can add properties to the message object. For example if you had an object containing sensor values that looked like this:

```
sensors={sensor1:20,sensor2:21}
```

You can add it to msg object using

```
msg.sensors=sensors
```

It will be passed on to the next node along with the existing msg properties.

Common Errors
When working with the message object you will probably encounter some of the following errors:

- function tried to send a message of type number Node-Red
- function tried to send a message of type string Node-Red

Both are cause by assigning a string or number to the message object instead of assigning an object e.g

```
msg=1;
msg="This is invalid";
```

Another common mistake is to delete the existing msg properties or not forward them on to the next node.

```
msg={};
return msg;
```

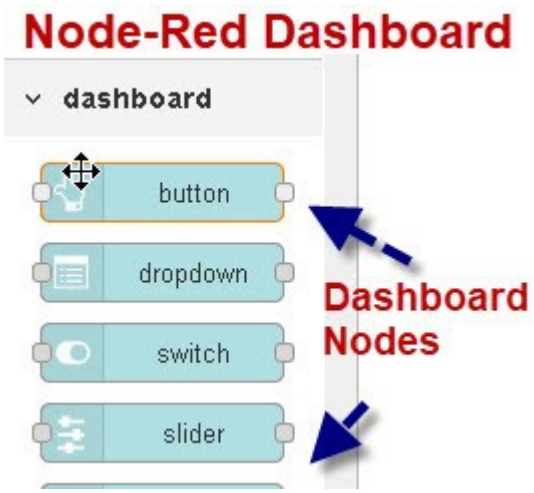
The following is valid but all the properties in the original msg object are lost to the following node.

```
msg1={};
msg1.payload="test";
return msg1;
```

Msg Object and The Node-Red Function Node
Using the function node gives you a better understanding of the Node-Red msg object

How to Create a Basic Node-Red Dashboard
The **Node-Red dashboard** is an add-on module that lets you create live dashboards.
It is an optional module and not installed by default.
To install the stable version use the Menu - Manage palette option and search for node-red-dashboard, or manually

```
cd ~/.node-red
npm install node-red-dashboard
```



They consist of both input and output nodes. To use them just drag them onto the flow canvas.

Note: it might happen that you need to install a special node or due to upgrades or bugs from a specific location.

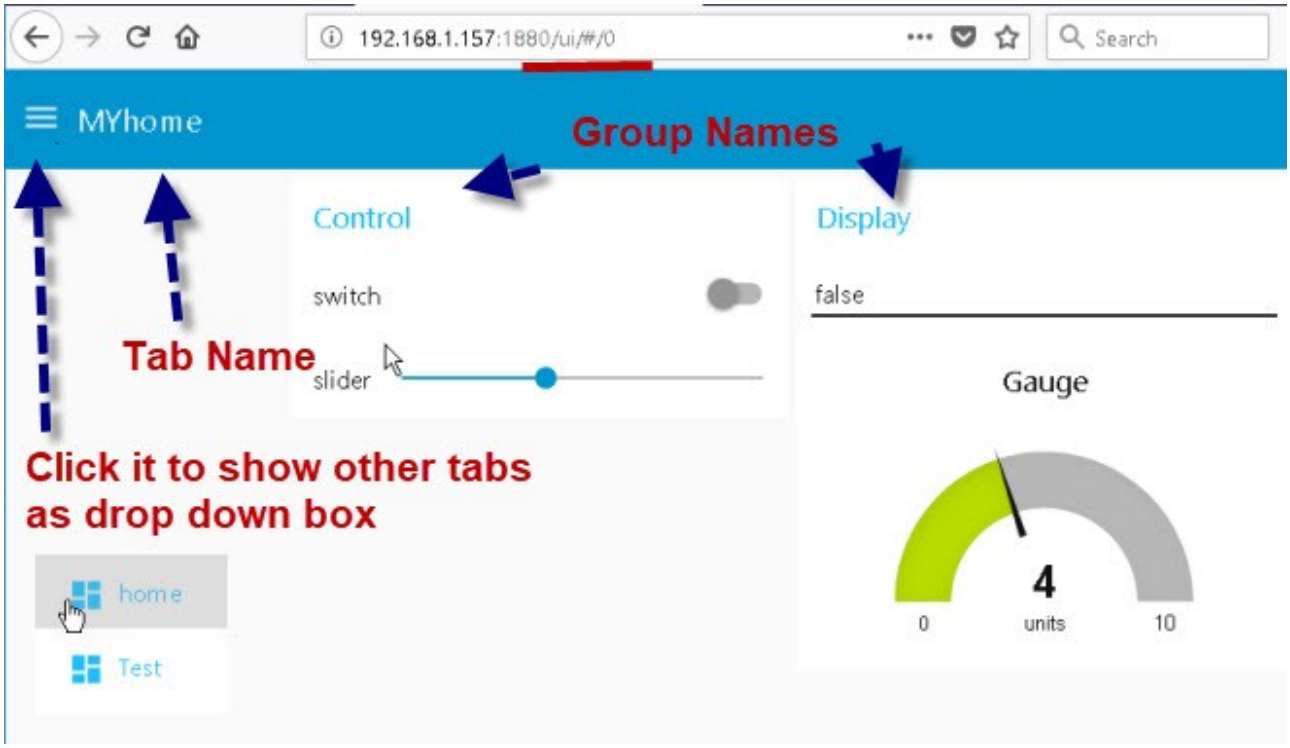
```
cd ~/.node-red
npm install –unsafe-perm node-red-contrib-i2c
```

or

```
cd ~/.node-red
npm install https://github.com/nielsnl68/node-red-contrib-i2c.git
```

Restart node-red or reboot your Pi after a manual install.

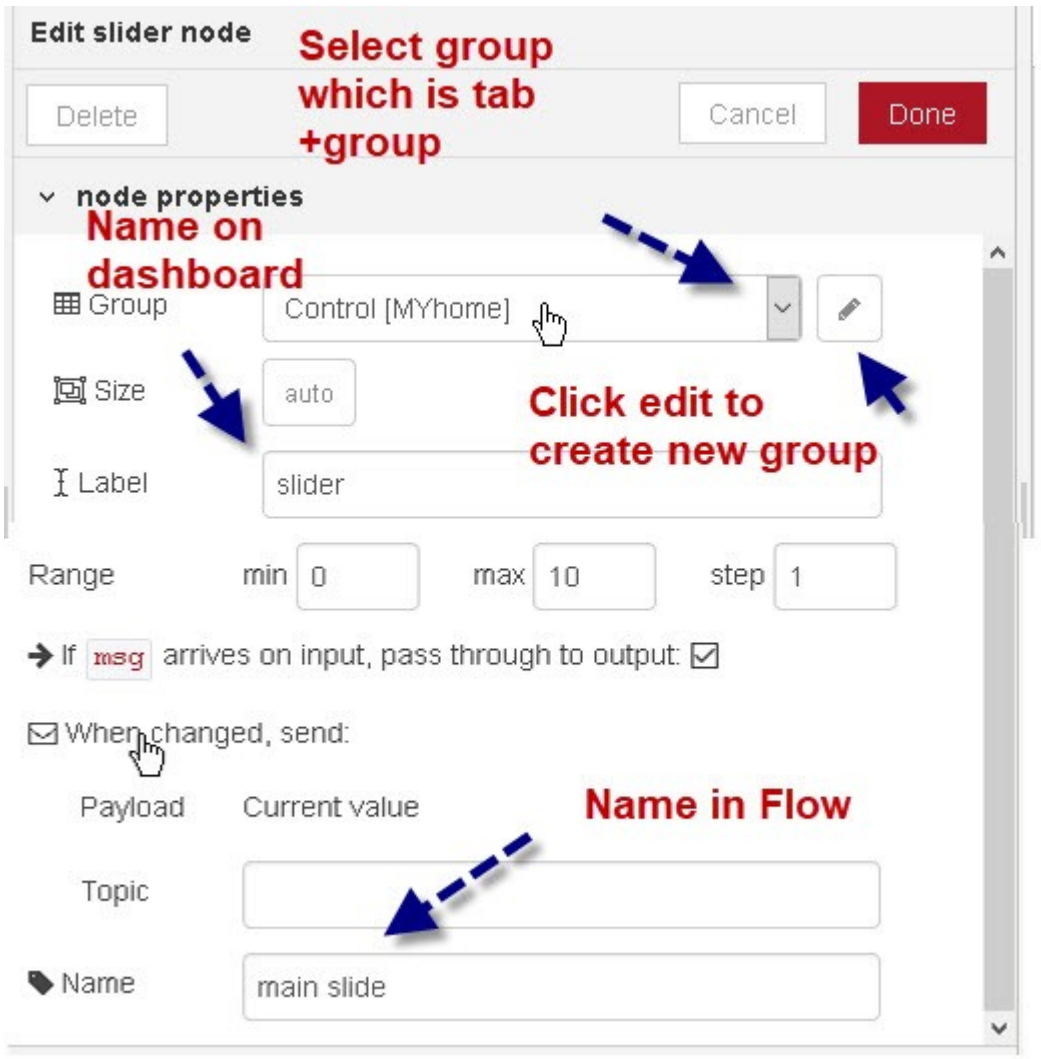
Using The Node-Red Dashboard
The dashboard or display nodes appear on the User Interface (UI) dashboard.
To access this dashboard use the url – host:1800/ui or in my example: 192.168.1.157:1880/ui
An example UI dashboard is shown below:



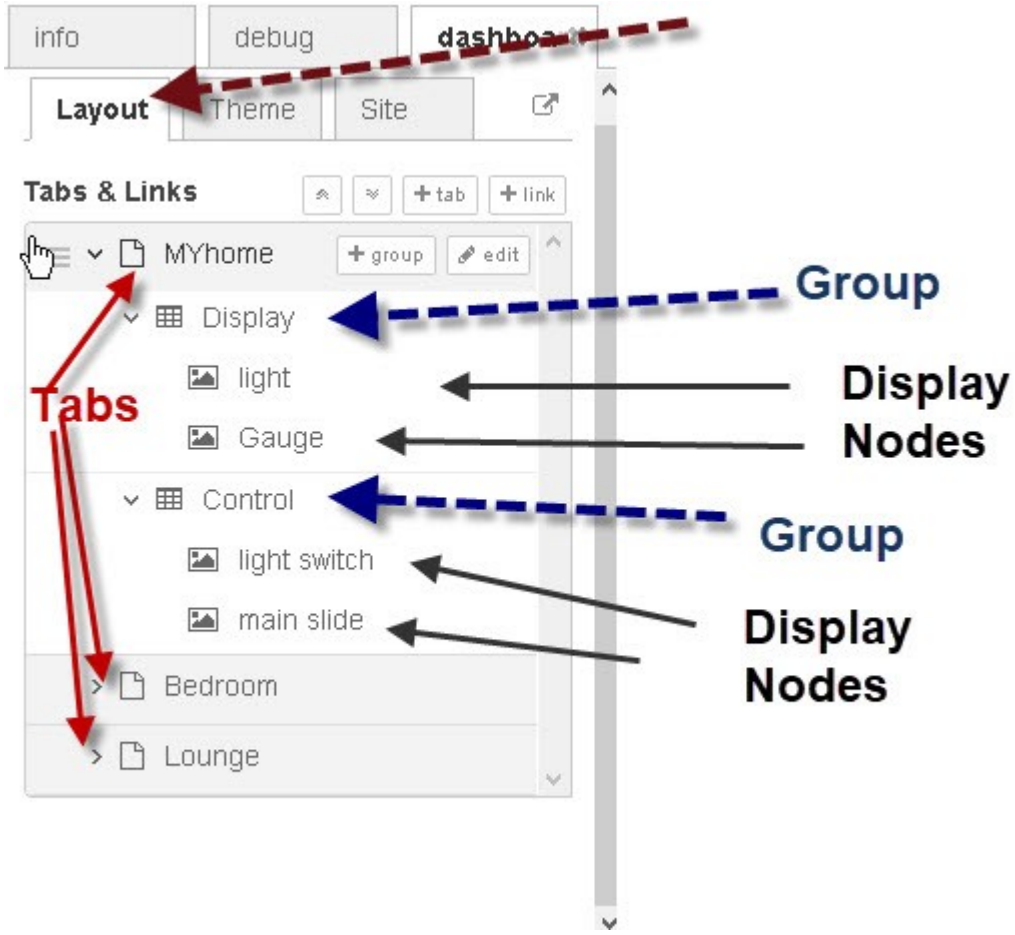
You can have multiple display pages called tabs and each tab has a name.
On a Page the display nodes can be arranged in Groups.
When you drag a node onto the flow canvas you will need to edit it to target a display tab and a display group.

Each dashboard node has three important settings:

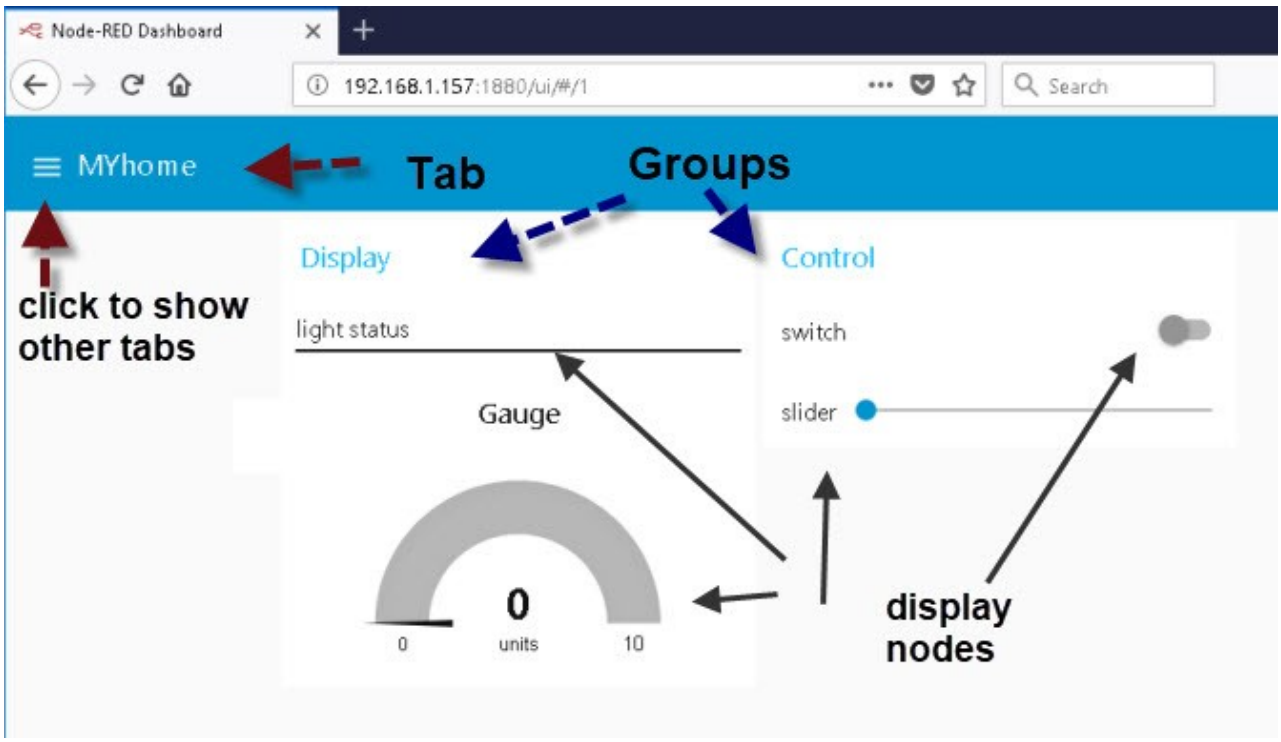
- The Group Name – group and tab example shows a group called control on the MYhome tab.
- Label name – This is the name on the dashboard
- Name – This is the name in the flow workspace.



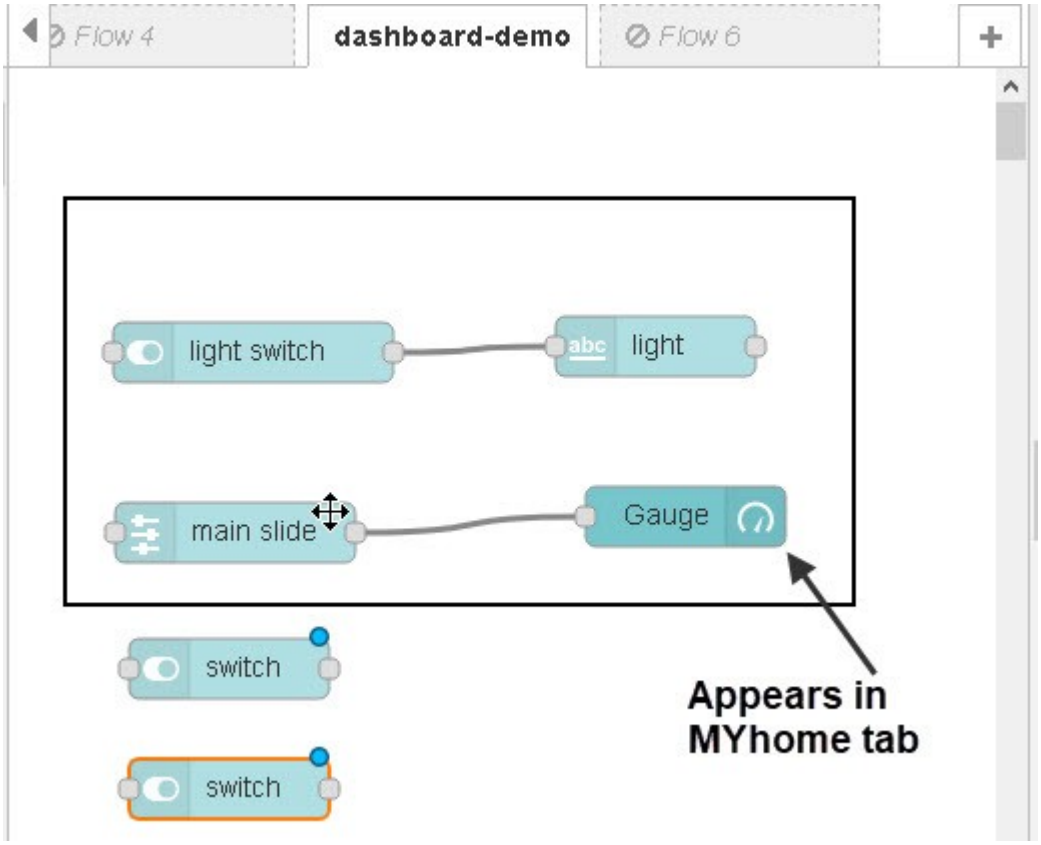
Controlling the UI Dashboard Layout
The layout of the UI screen can be controlled in the dashboard tab (third column main screen) of the admin page and select layout.
You can create new tabs and groups and move display nodes between groups using this screen.
This is usually were you do the initial design.



The screen shot above shows a flow with three display tabs called MYhome and Bedroom and Lounge. The MYhome tab has two display groups call display and control. The display group has two display nodes guage and light and the control group has two display nodes -light switch and main slide. This is how they appear on the User Interface.



This is the Flow I used to create the above.



Working with JSON Data And JavaScript Objects in Node-Red

JSON is popular format for encoding data sent over the Internet, and also stored in files. In computing, JavaScript Object Notation (JSON) is an open-standard file format that uses human-readable text to transmit data objects consisting of attribute–value pairs and array data types (or any other serializable value).

Encoding and Decoding JSON Data
You can encode and decode JSON data using the JavaScript functions `JSON.stringify()` and `JSON.parse()` or the JSON Node.

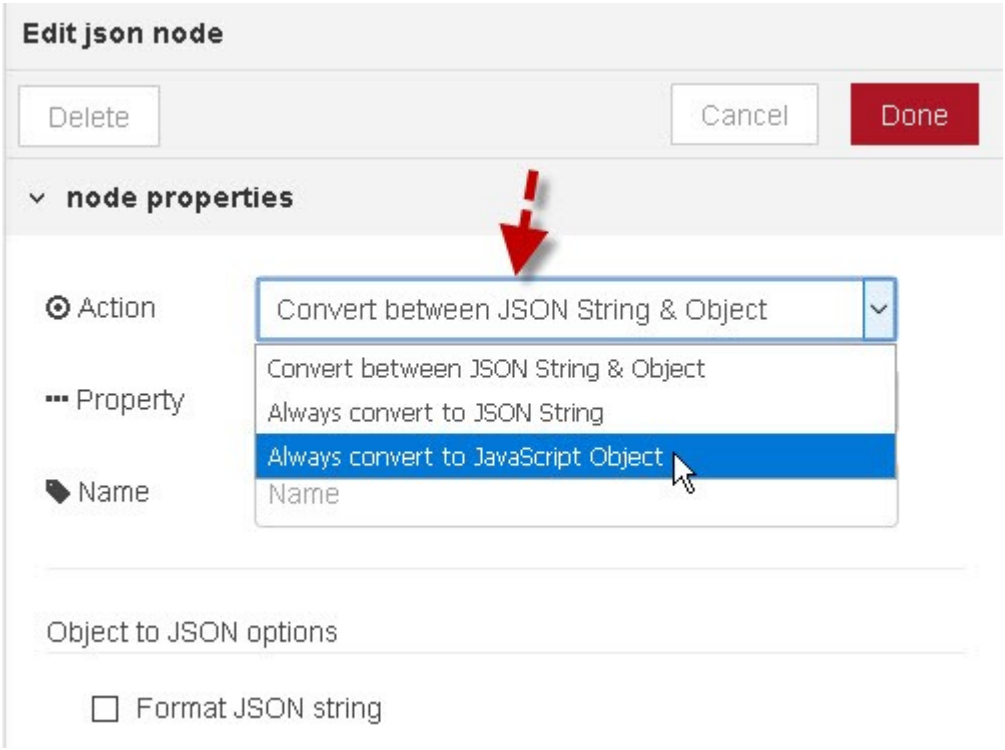
Encoding JSON Data
Creating a JSON string from a JavaScript object

```
var s=JSON.stringify(JavascriptObject);
```

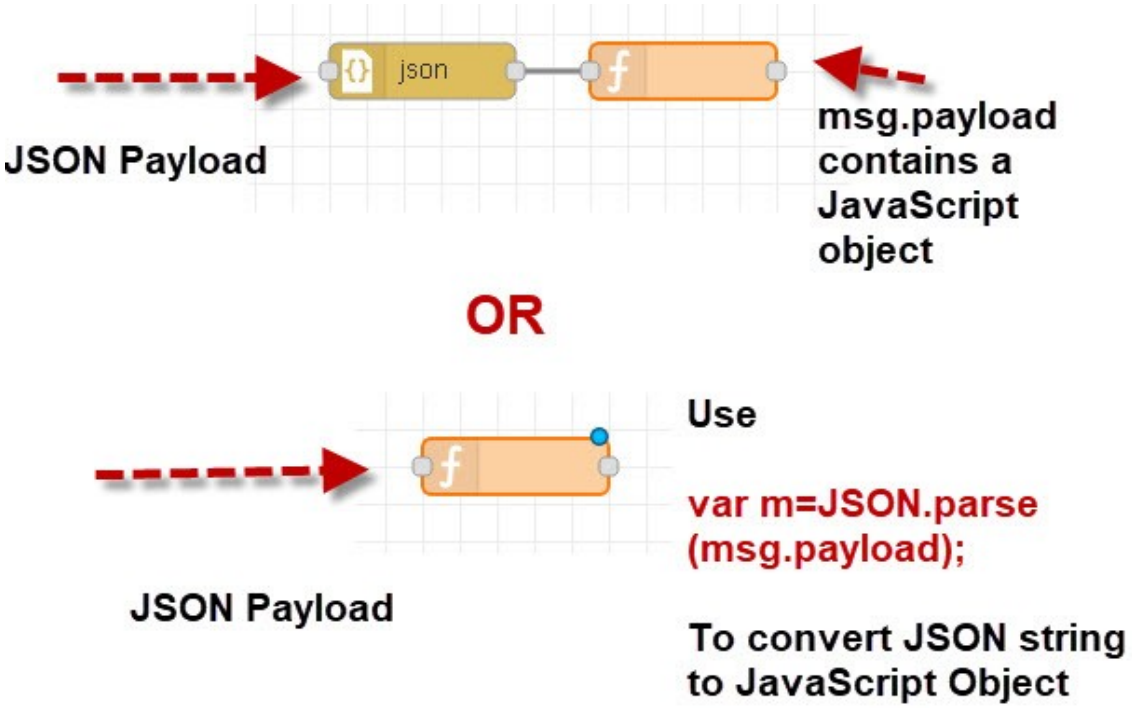
Decoding JSON Data
Creating a JavaScript object from a JSON string

```
var o=JSON.pase(JSONString);
```

The JSON node located under the functions category is capable of converting between a JSON string and JavaScript object and vice versa.



Receiving JSON Data
The first thing you need to do when you receive JSON data is to convert it into a JavaScript object. Once you have a JavaScript object you can work on the data. The two approaches using the JSON node and the `JSON.parse()` command are illustrated below:

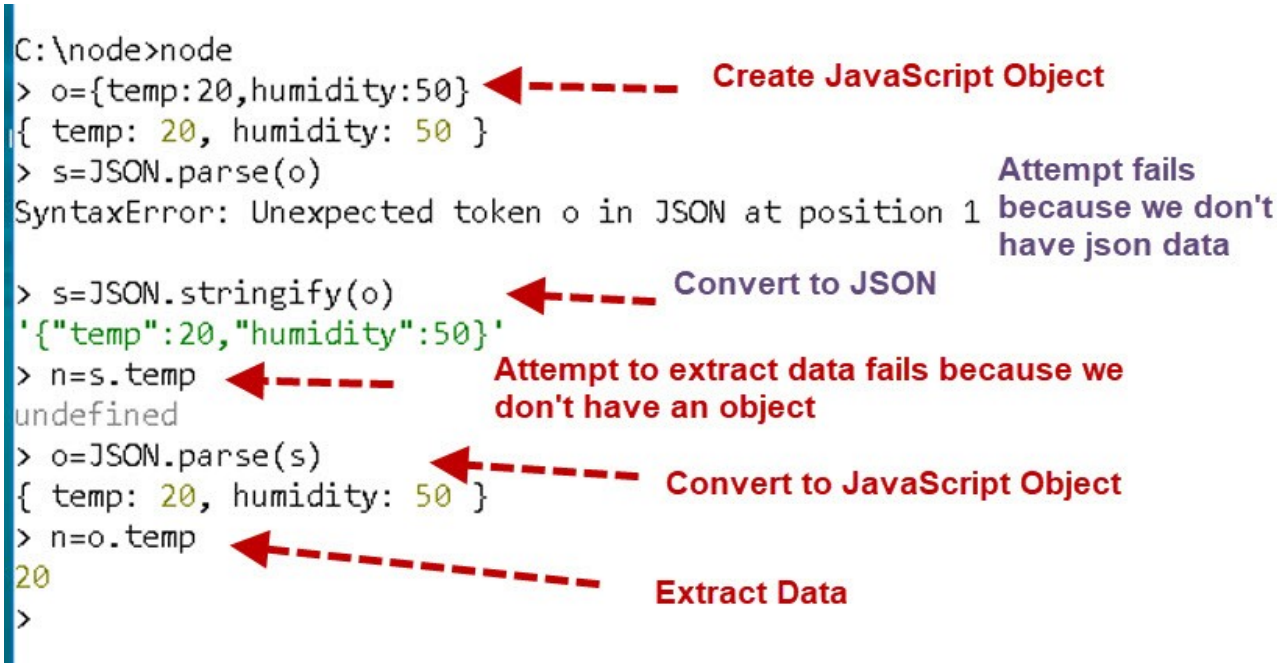


Working With JSON Data In Node-Red

Is it JSON Data or a JavaScript Object?
One of the most common problems that I've seen is trying to extract data from a JSON string thinking that it was a JavaScript object. In Node-Red if you pass the data into the debug node then it will show you the data and tell you the data type. JSON data is a string and is surrounded in quotes as shown below.

Node-Red Debug Node JSON Data

Extracting Values from a JSON String
A Common requirement is to extract a particular value or values from the incoming JSON data. To do that the first thing we must do is to convert the JSON string into a Javascript object. Once we have a JavaScript object we can extract individual values using the key in dot format or quotes. The following screen shot illustrates the process, using the command line ,along with some common problems that you might encounter like:
SyntaxError: Unexpected token o in JSON at position 1



Note: The node command line doesn't require var declaration or ; to terminate the line. You will need them in the script in the function node.

JavaScript Object Notes:
Generally a JavaScript object key doesn't need quotes. In the example above we used.

```
var o={temp:20,humidity:50};
```

and not

```
var o={"temp":20,"humidity":50};
```

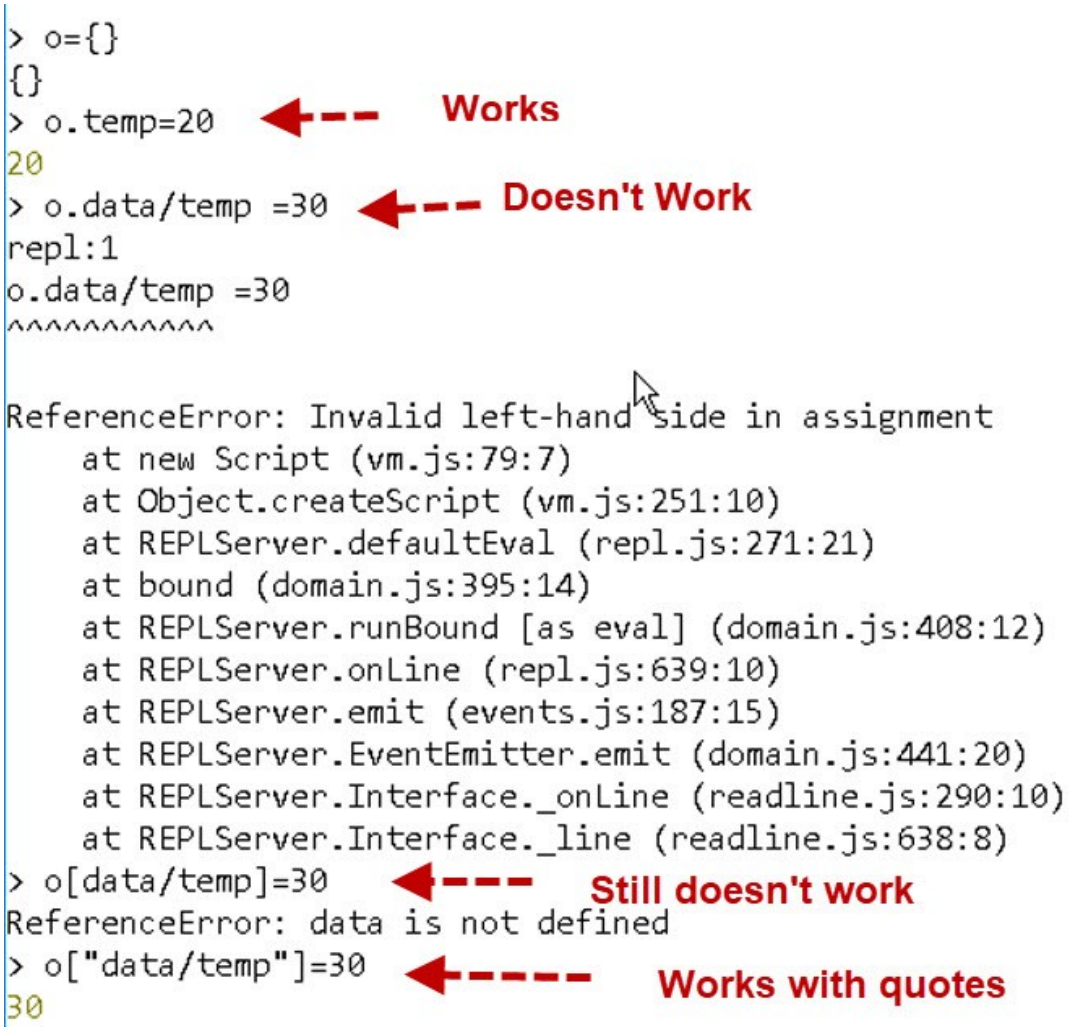
However both are valid. There are various rules on whether or not quotes are needed. However it is usually best to use quotes to avoid confusion. When accessing a value in a JavaScript Object you can use:

```
var value=o.temp;
```

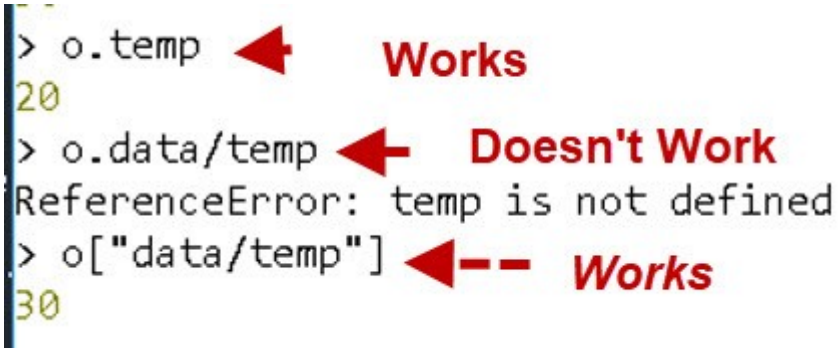
or

```
var value= o["temp"];
```

The bracketed option will always work the dot notation will work depending on the key name. The following screen shot illustrates assigning values to objects using the node command line.



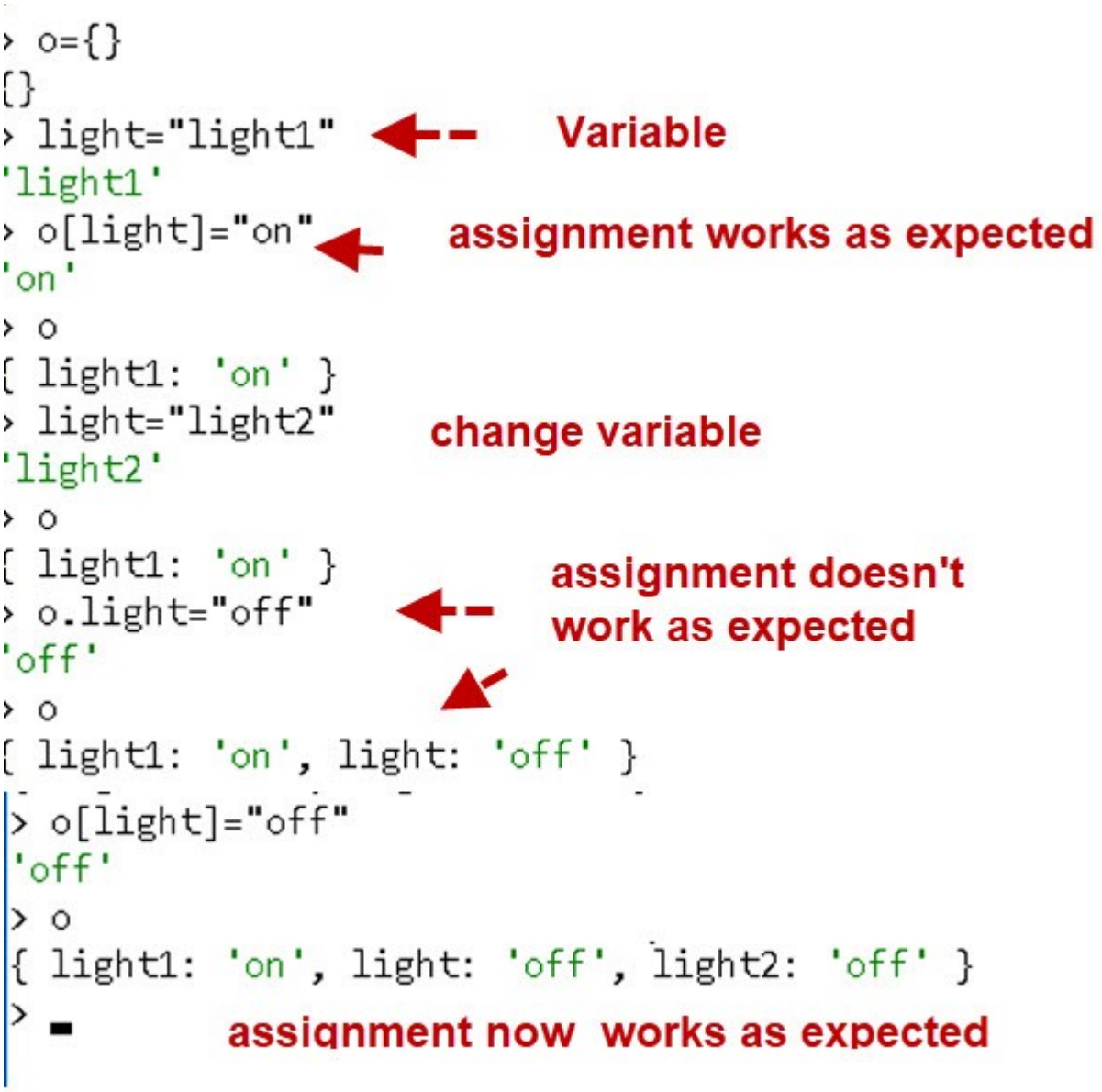
To access the data we encounter the same problems without quotes.



Using Variables as Keys
Using a variables as an object key is a common requirement. Again you find that you are required to use the bracketed option and not the dot notation.

```
var light="light1";
var o={};
o[light]="on";
```

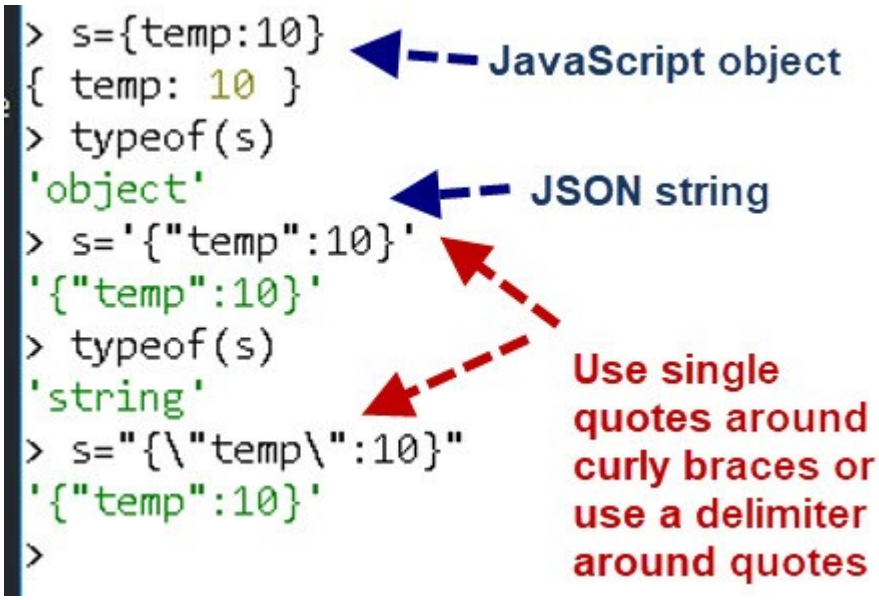
Again We illustrate using the node command line.



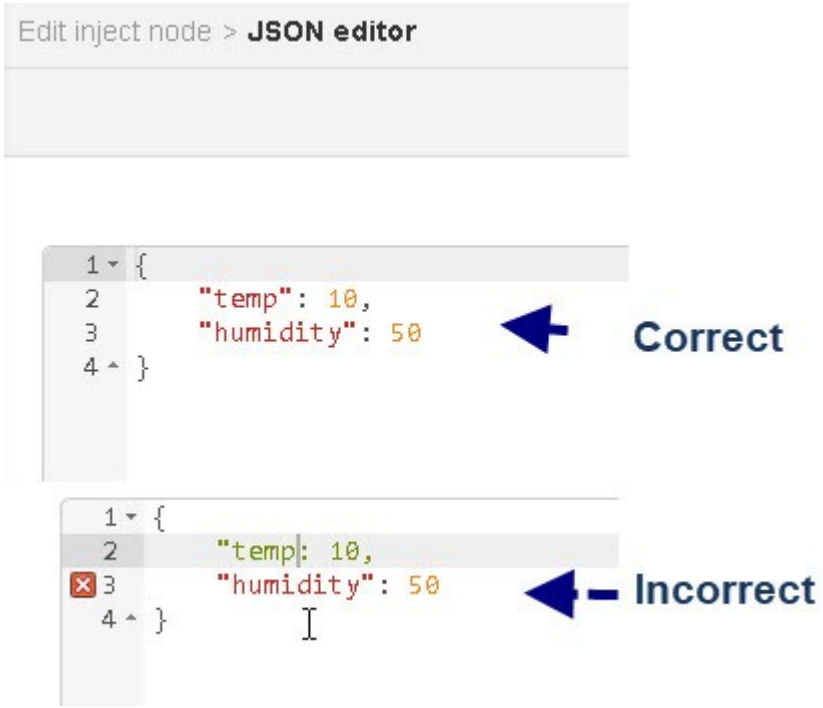
Manually Entering JSON Data
When testing it is often necessary to create test data. It is relatively easy to hand code Simple JSON data, however for more complex data I would recommend using the node command line to create the JSON Data from a JavaScript object. In JSON all string values must be in quotes.

```
var s={temp:10}; //create Javascript object
var  s='{"temp":10}';
var s="{\"temp\":10}";
```

as shown below:



Entering JSON Data into the Node-Red Inject Node
Select the JSON option and enter the data using quotes around strings.
The JSON edit will show you if you have entered an invalid format.



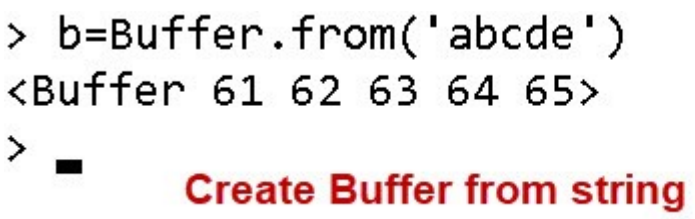
Using the Mosquitto Publish Tool With JSON
With the **mosquitto_pub** tool you will need to escape the quotes and enclose it in quotes.
E.G.

```
mosquitto_pub -h localhost -t test -m "{\"v1\/lights\":1}"
```

Notice also how I needed to delimit the forward slash in the key name.

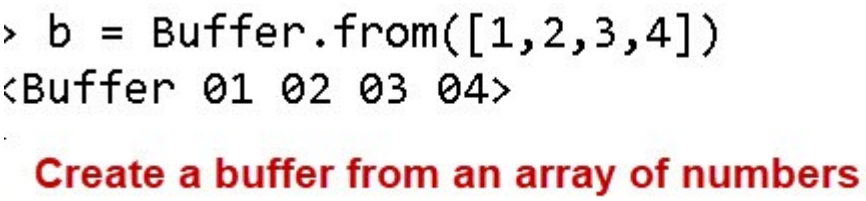
Understanding and Using Buffers In Node-Red
When data is read from a file or network it is read byte by byte into a data buffer.
Data Buffers are temporary storage used for transferring data. To work with binary data we will need access to these buffers. To work with buffers in node and Node-Red we use the buffer object. The following screen shot of the node command line shows how we work with characters using the buffer object. The first thing we need to do is create a buffer. This can be done in a number of ways:
Create a buffer from a string.

```
Buffer.from('abcde')
```

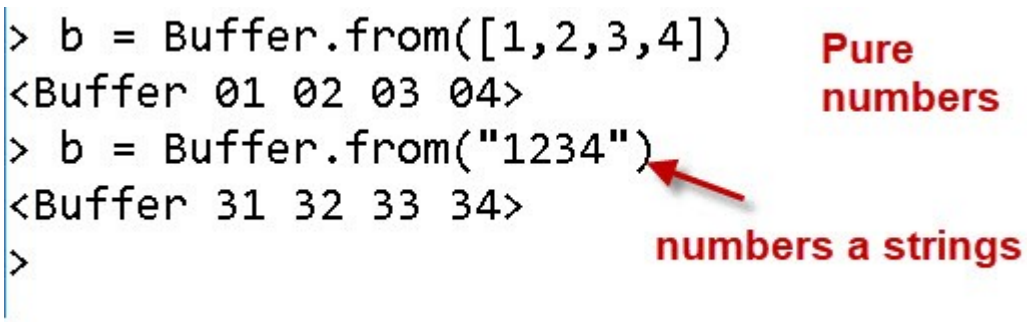


Create a buffer from an array of numbers

```
var b = Buffer.from([1,2,3,4]);
```



Notice the difference between numbers and strings and numbers as numbers.



Create an empty buffer of 10 bytes and Initialise it.

```
> b=Buffer.alloc(10)
<Buffer 00 00 00 00 00 00 00 00 00 00>
>
```

Create and Initialise a buffer of a fixed length

Working With String Buffers
If you read the tutorial on data and character encoding you will see that Characters and numbers e.g A and 1 are represented as bytes by encoding them.
In the early days of computing ASCII was the encoding standard, but due to the lack of support for foreign characters it has been replaced by utf-8.
Below is a screen shot showing a partial ASCII table for the numbers 0-9.

Partial Ascii Table

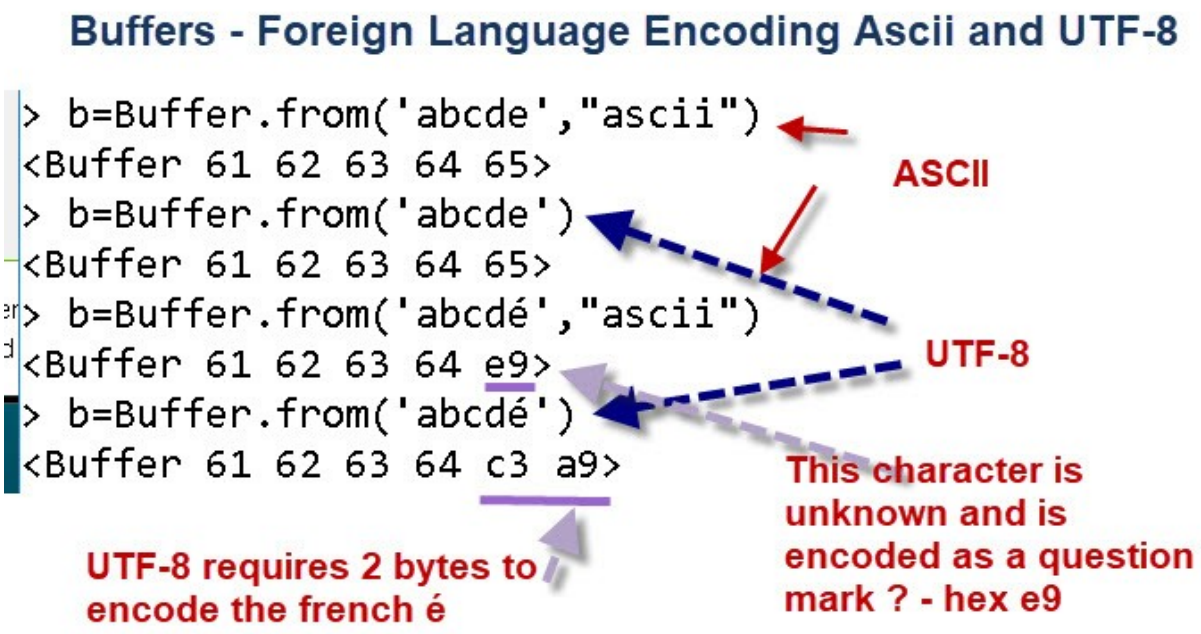
Dec	Oct	Hex	Binary	Character
48	060	30	00110000	0
49	061	31	00110001	1
50	062	32	00110010	2
51	063	33	00110011	3
52	064	34	00110100	4
53	065	35	00110101	5
54	066	36	00110110	6
55	067	37	00110111	7
56	070	38	00111000	8
57	071	39	00111001	9

1 is encoded as hex 31 and binary 00110001
When we use the Buffer.from method to create a buffer from a string then we can also specify an encoding scheme, utf-8 is the default. Now utf-8 uses 1 to 4 bytes to encode an character whereas ASCII only uses 1 byte. However when we encode the characters ‘abcde’ we only have 5 bytes.

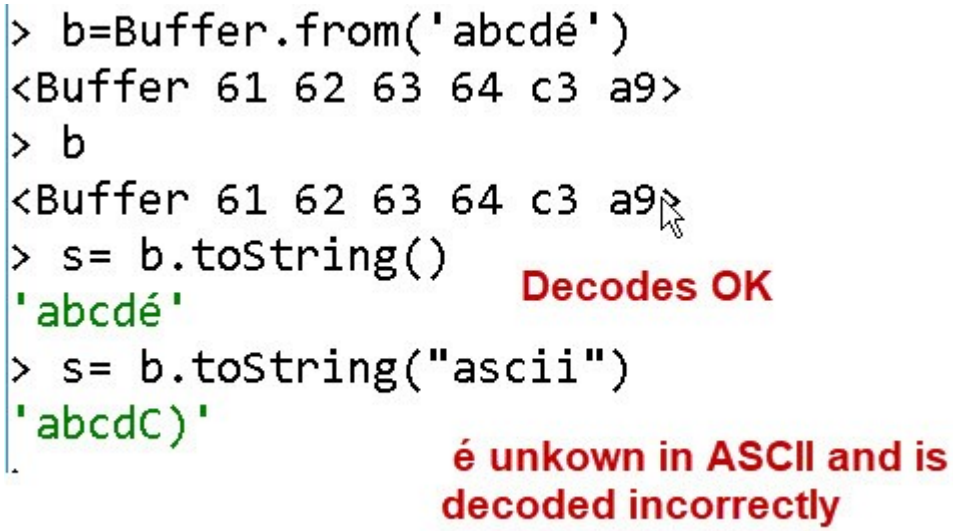
```
> b=Buffer.from('abcde')
<Buffer 61 62 63 64 65>
>
```

Create Buffer from string

This is because the English characters are the same in ASCII as utf-8.
However what happens if we encode foreign language characters line the French é ?
In the above screen below we see for English characters the encoding is the same in ASCII as UTF-8 and uses a single byte.
However when encoding the French é UTF-8 uses 2 bytes and ASCII can't encode it and so substitutes the question mark character.



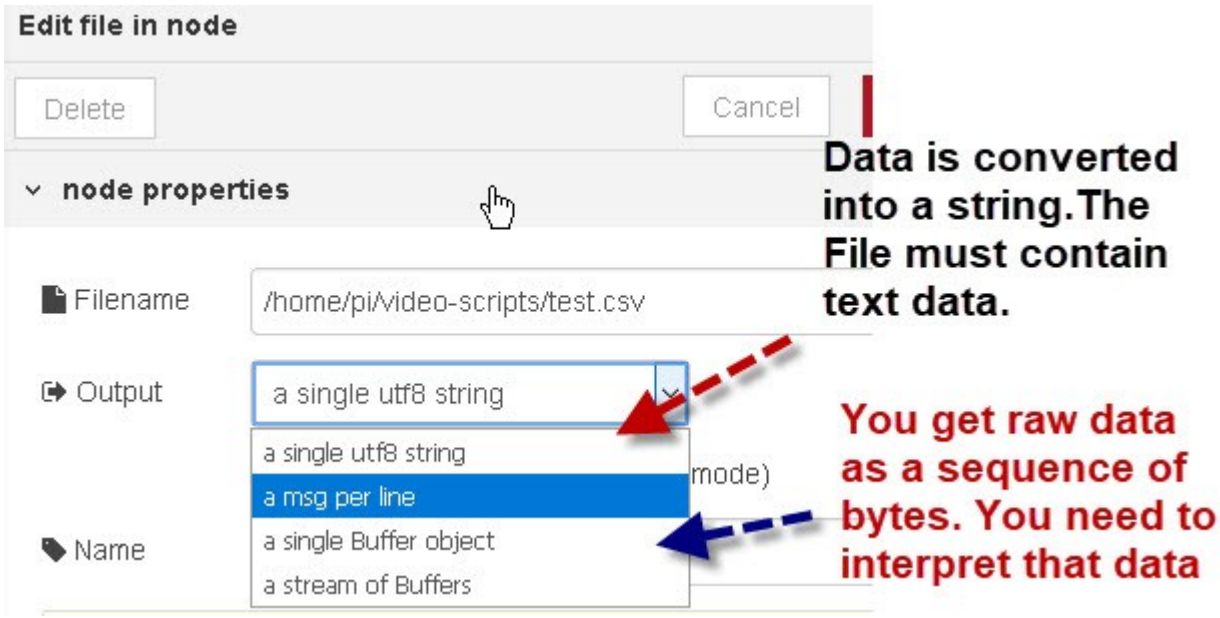
Interpreting Data in Buffers
When reading in data from the network or from a file it is important to know what type of data we are working with.
If it is string data then what is the encoding that was used. If we take our simple example of abcdé encoded as UTF-8.
If we read it using the toString() method, and decode it as ASCII this is what we get:



You will notice that ASCII uses 1 byte per character the French character é is encoded as C3 A9. It is decoded incorrectly in ASCII as C).

Node-Red Example Reading a File
If you look at the file read node you will see that there are 4 options. of how the read presents the data to your flow. They are:

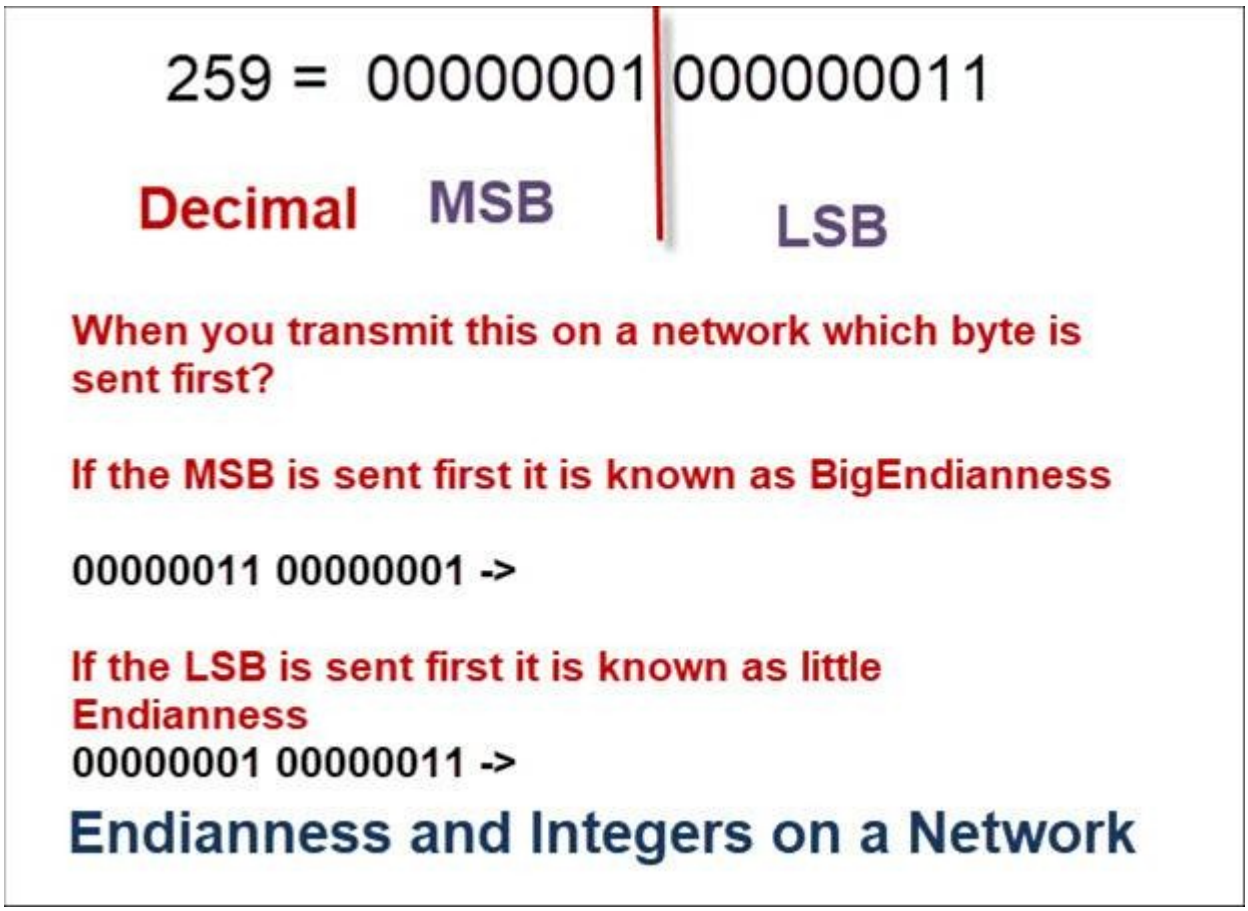
1. A single utf-8 string – In the case the entire file is read and presented to you program as a text string. You need to wait for the read to complete before you can process the data.
2. One msg per line– Presents the data to your program as lines of text. You can process data while still reading the file.
3. A single Buffer object – Same as the first option but now you have a sequence of raw bytes. You need to wait for the read to complete before you can process the data.
4. A stream of buffers -Same as second option but you have a sequence of bytes so you can process data while still reading the file.



The first and second options are the most common as files normally contain text data.
If the data in the file isn't text data or not encoded as utf8 then the single utf8 and 1 msg per line options won't work as with these options the data is read in, and decoded using utf8 into characters.

The third and forth options are for when the file contains binary data or text data encoded in another format other than utf8 or ASCII. These options reads the data into you script/flow without any interpretation i.e as raw data.

Encoding and Decoding Integers and Floats
When working with machines you will often be required to encode and decode integer and float data, modbus is a good example.
Early computers used 16 bits to represent an Integer and 32 bits for a floating point number. Today they can be much larger put the process is the same.
The image below shows a 16 bit Integer and how it is organised into bytes:



It is important to understand that because data is sent across a network in bytes it is important to understand whether the MSB (Most Significant Byte) or the LSB (Least Significant Byte)is transmitted first.
This is know as endianness and is discussed in the video.
Most networks use Big endianness (MSB first).

Creating a 2 byte buffer from an Integer (16 buts)
First we create an 2 byte empty buffer and then write the integer into it

```
var buf=Buffer.alloc(2);
var res=writeInt16BE(17201);
```

Creating a 4 byte buffer from an Integer (16 bits)
First we create an 4 byte empty buffer and then write the integer into it

```
var buf=Buffer.alloc(4);
buf.writeInt16BE(17201);
```

Creating a 16 bit Integer from a 2 byte buffer
We start with our 2 byte buffer containing a 16 bit integer and then we create the integer using the read method.

```
var buf=Buffer.from([0x43,0x6c])
var res =buf.readInt16BE()
```

Creating a 4 byte buffer from an Float (32 bits)
First we create an 4 byte empty buffer and then write the Float into it

```
var buf=Buffer.alloc(4);
buf.writeFloatBE(131.3)
```

Creating a 32 bit Float from a 4 byte buffer
We start with our 4 byte buffer containing a 32 bit Float and then we create the float using the read method.

```
var buf=Buffer.from([0x43,0x03,0x4c,0xcd])
var res =buf.readFloatBE();
```

Using the Node-Red Function Node

The function node is used to run JavaScript code against the msg object. The function node accepts a msg object as input and can return 0 or more message objects as output.
This message object must have a payload property (msg.payload), and usually has other properties depending on the proceeding nodes.

Accessing the msg Properties in The Function Node.
The message payload can be accessed using:

```
var payload=msg.payload;
```

and can be modified using:

```
msg.payload=payload;
```

Likewise the message topic can be accessed using:

```
var topic=msg.topic;
```

and can be modified using:

```
msg.topic= topic;
```

It can be extended using

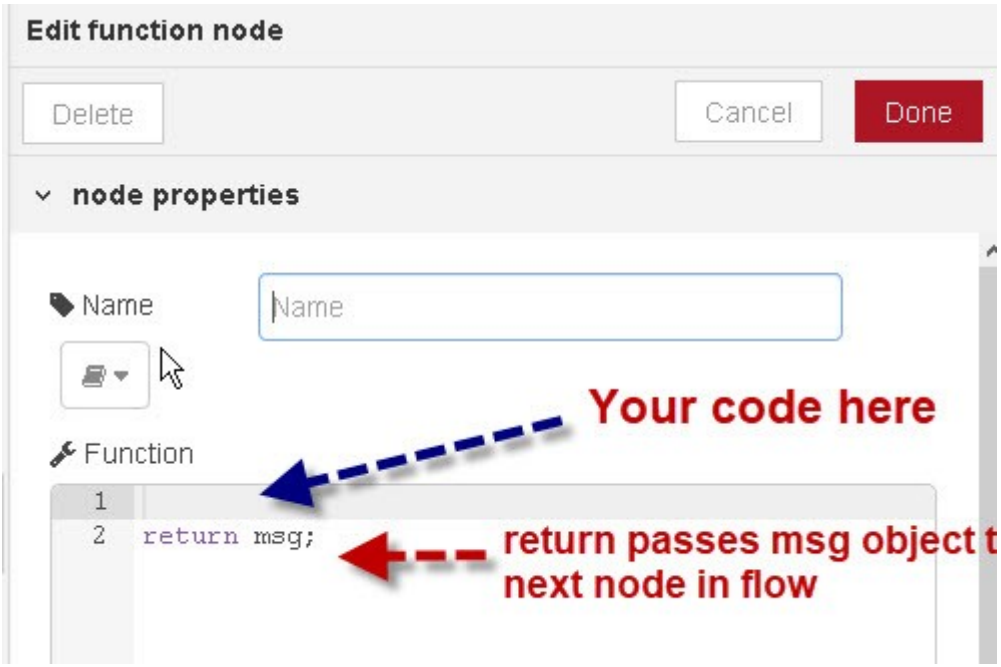
```
var  newvalue;
msg.newproperty=newvalue;
```

Now the msg object has a property called msg.newproperty.

Creating a New Message Object
Inside a function you can create a new **msg object** using:

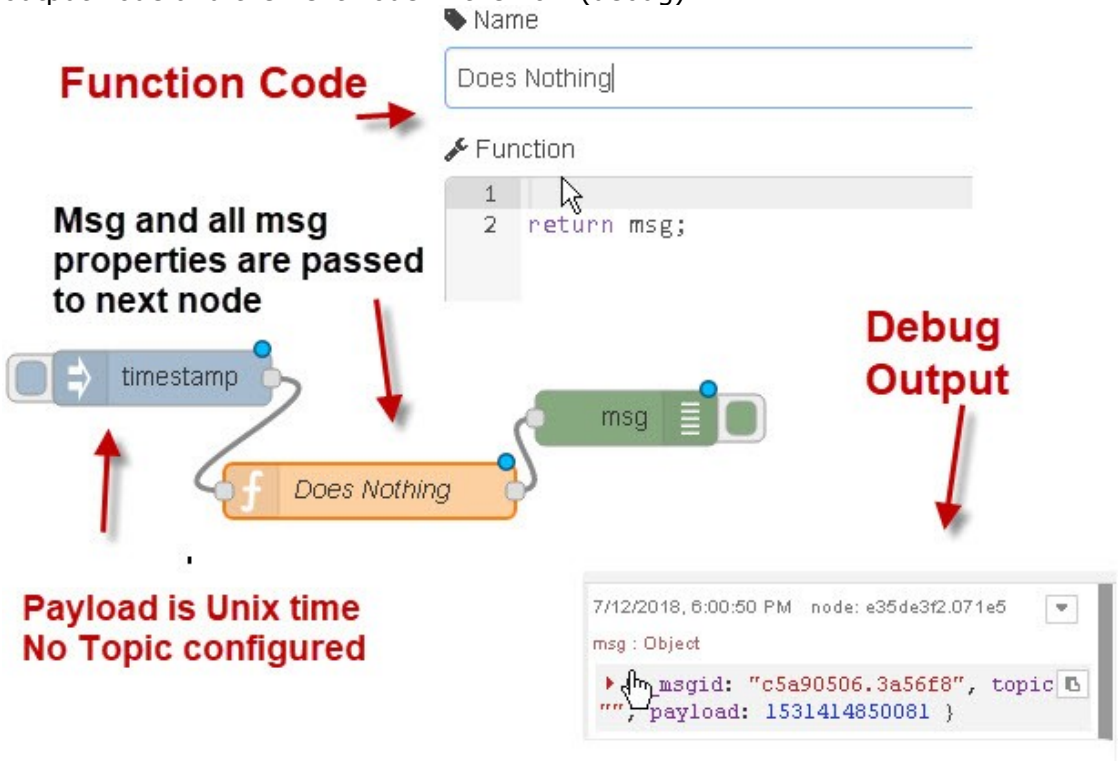
```
var newMsg = { payload: msg.payload,topic:msg.topic };
return newMsg;
```

Using the Function Node
When you drop a function node onto a flow , and go to edit you will see a single line of code that returns the msg object and a blank line above were you can start to enter your own code.



If you don't return a msg object then the flow stops.

Simple Function Example 1
The flow below is uses the function node with the default code which simply returns the msg object.
The effect is simply to pass the msg object and all of it's properties from the input node to the output node and the next node in the flow (debug).



Simple Function Example

The inject node injects a msg object into the flow with the Unix time stamp as the payload and a blank topic. It is passes through the do nothing function node and you can see that this appears on the debug node.

Function Node Example 2
Next we use the inject node to inject a payload with the string "test string" and a topic of test. If we pass this into our do nothing function as before we get the following output.

The output is as expected. This time we show the topic as test and the payload as test string. Now if we modify the function to change the payload t Upper case and the topic to upper case using the following code:

```
var payload=msg.payload; //get payload
msg.payload=payload.toUpperCase(); //convert to uppercase
var topic=msg.payload; //get topic
msg.topic=topic.toUpperCase();//convert to uppercase
return msg;
```

The first line of the code retrieves the msg payload.

```
var payload=msg.payload; //get payload
```

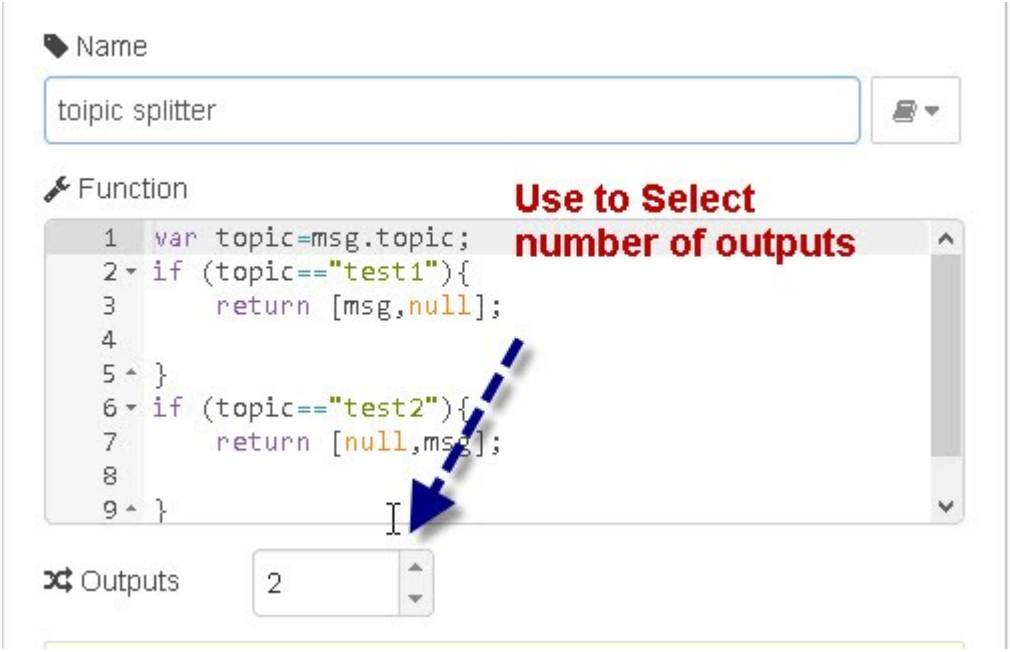
The second line converts it to upper case and re-assign it back to the msg object.

```
msg.payload=payload.toUpperCase();
```

We then do exactly the same with the topic property before returning the complete msg object. The output on the debug screen is shown below:

Notice the topic and payload have been converted to upper case.

Multiple Outputs
The function node can be configured with multiple outputs.
This is useful when the flow splits into separate paths depending on a message property.
To configure multiple outputs open the function node and use the up/down arrows to adjust the outputs. Outputs are numbered starting with 1 at the top.



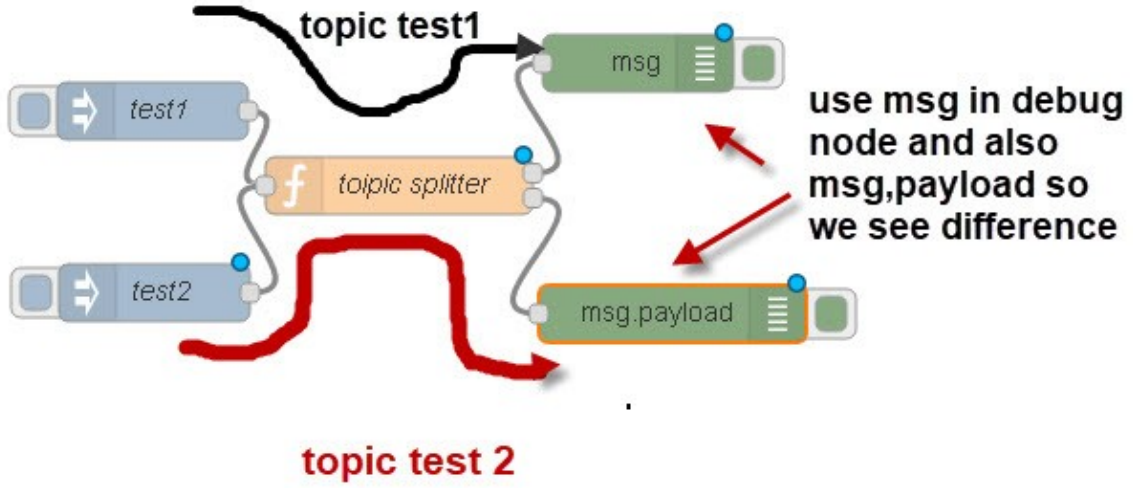
To return messages to multiple outputs you need to return an array. So the return looks like this:

```
return [msg1,msg2];
```

Msg1 will appear on output1 and msg2 on output2. To stop a flow you return null.
So to return the msg objext on output1 and nothing on output2 use:

```
return [msg1,null];
```

Example
In the example flow we use two inject nodes to inject a message on two different topics and send the message to the output based on the topic name.
Topic test1 goes to output1 and test2 goes to output2.



Node-Red Function Example 3

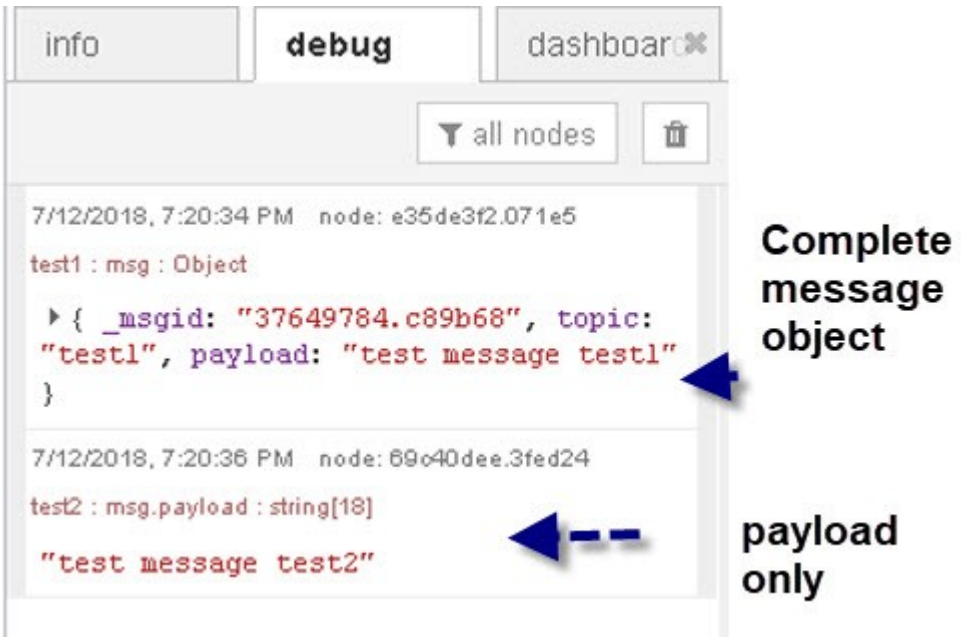
The following code is used in the function node to spit the message path based on the topic name. Notice the return statement.

```
var topic=msg.topic;
if (topic=="test1"){
    return [msg,null];

}
if (topic=="test2"){
    return [null,msg];

}
```

The output on the debug screen is show below:



Multiple Messages on a Single Output
You can use an array to return multiple msg objects on a single output.
So the return would look like this

```
return [msg_object_array];
```

Important: you are returning an array of objects in an array.
This is best seen with an example

Example Multiple Messages on Single Output:
In this example we use an inject node to inject a test string into the function node.
The function node takes the string and uses it for the message payload, but instead of sending 1 message it has a for loop which creates 3 messages and puts them in an array . The function returns the array as an array! Here is the flow:

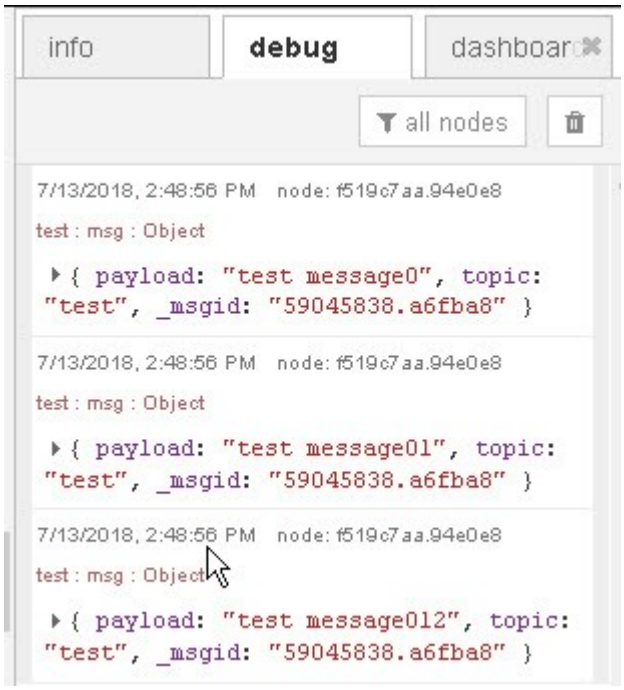
Here is the code of the function node:

```
var m_out=[]; //rray for message objects
var message=msg.payload;
for (i=0;i<3;i++){

    message=message+i; //add count to message
    var newmsg={payload:message,topic:msg.topic}
    m_out.push(newmsg);
}
return[m_out];
```

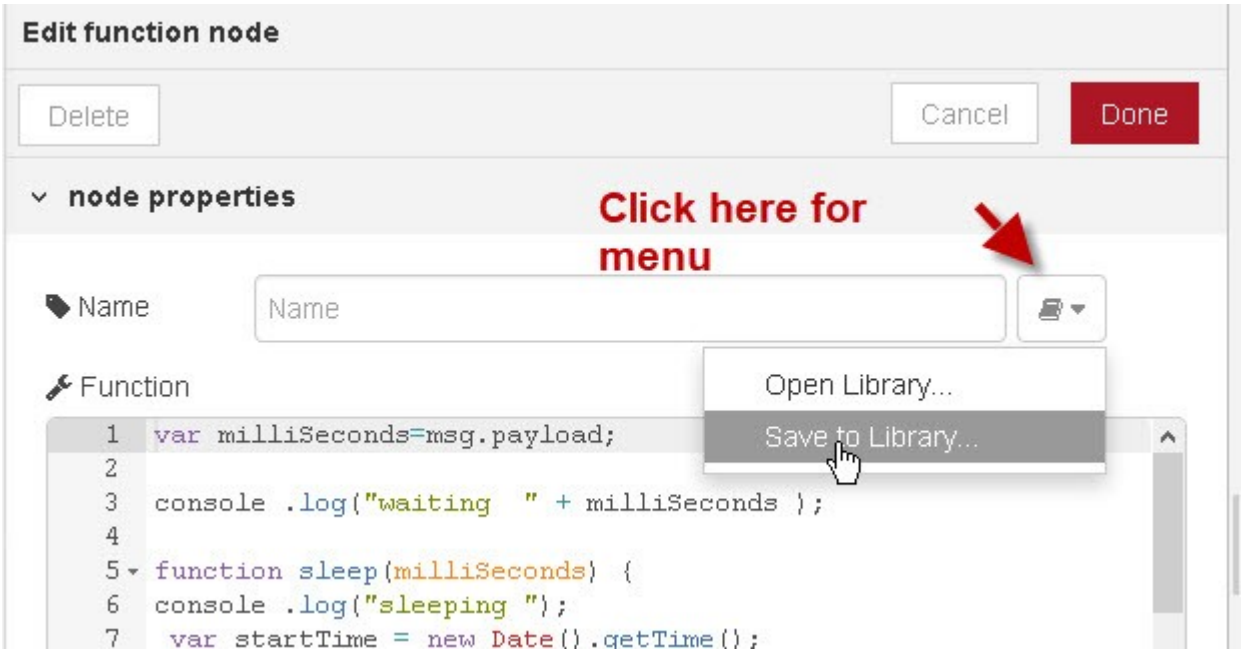
Important: notice the return statement returns an array.

Here is the debug screen output when run notice the debug screen shows 3 messages:



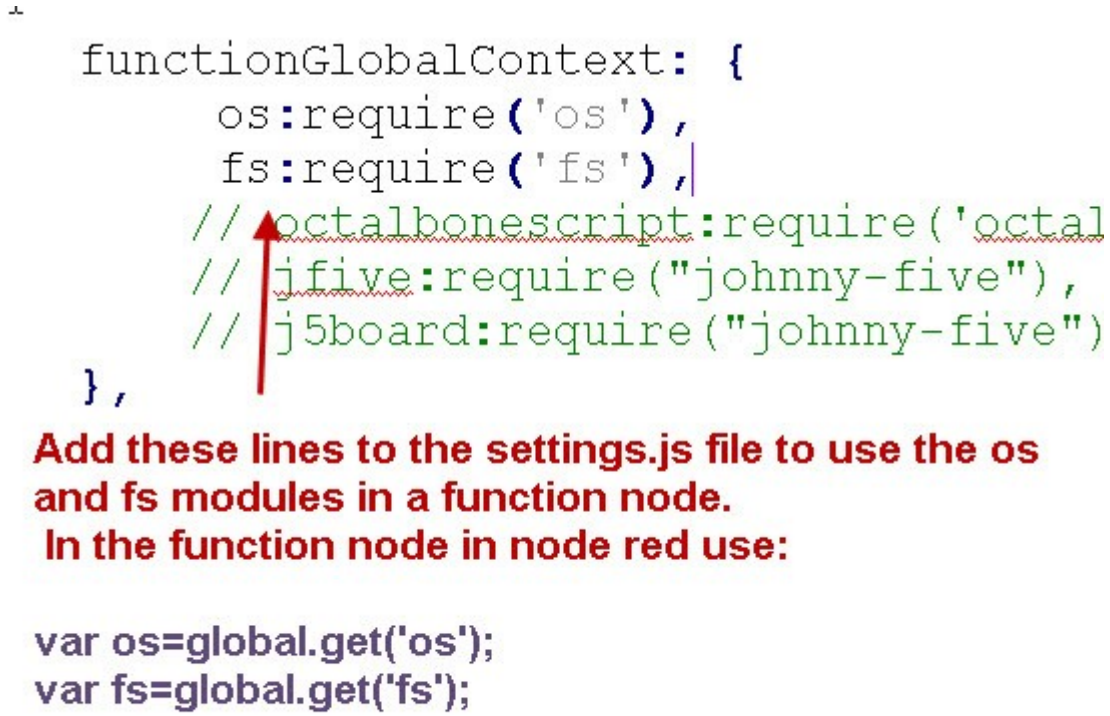
Storing Data
Data can be stored in the function node using the context object.

Reusing Function Nodes
You can save your function nodes in the library and reuse them in other flows by importing them
To save a function to the library double click on the function to edit it and click on the bookmark icon next to the function name.
A drop down menu appears to either import or save the function to the library.



You can also use a sub flow to store your functions. Using a sub flow makes them available as nodes which you can select from the node palette on the left.

Using Additional Node Modules
If you need to use an node module e.g the os or fs module then you need to enable them in the settings.js file under the functionGlobalContext: object as shown in the screen shot below:



To use them in the function node they are part of the global object so use:

```
var os=global.get('os');
var fs=global.get('fs');
```

Function Example:
This function will list all files of the type **.js** on the console

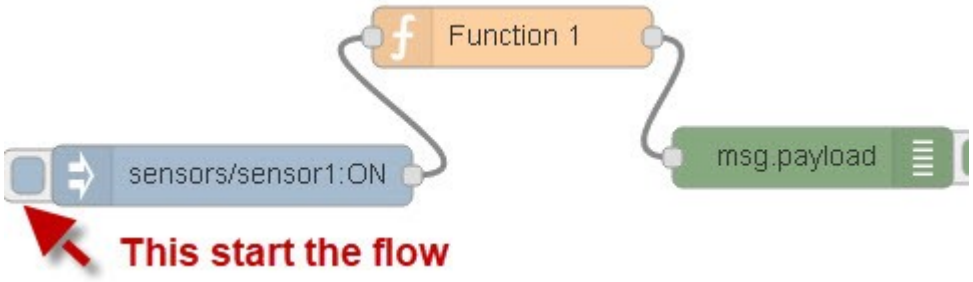
```
var fs=global.get('fs');
var path="/home/steve/.node-red";
try{
fs.readdir(path, function(err, items) {
    for (var i=0; i<items.length; i++) {
 var r=items[i].search(/\.js/);
if (r!= -1)
        node.log(items[i]);
else
node.log(items[i]+"is not a js file");
    } //for loop
});
}
catch(err){
node.log("error");
}
return msg;
```

Note: path attention to the file path name

Storing Data in Node-Red Variables
Node-Red nodes pass the msg object between nodes.
However this object is replaced by the next msg object. So how do you store data between node calls?
Node-Red provides three mechanisms:

- The context object - stores data for a node
- The Flow object - stores data for a flow
- The global object - stores data for the canvas

I will illustrate these methods using a simple flow like the one below.

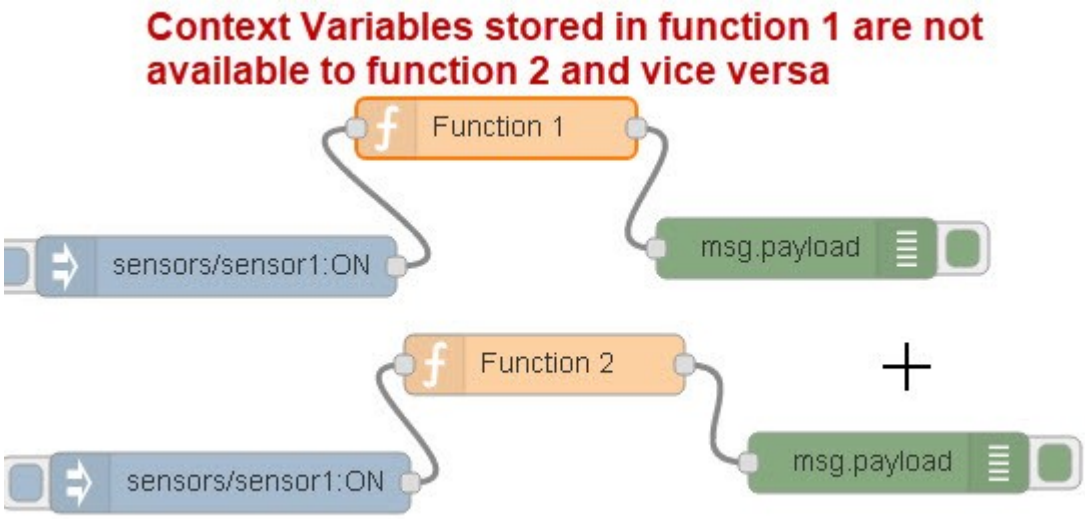


The inject node is used to start the flow , then function node implements the counter and the debug node displays the result. The idea is to count the number of times the message was injected. The actual message that gets injected isn't important.

Using the Context Object
This is used for storing function variables.
The process for retrieving and storing is to use the get method of the object for retrieving value and the set method to store values.:

```
x =context.get(name); //to retrieve a variable
context.set(name)=x; // to store a variable
```

A variable stored for function 1 in the context object is not available to function 2 and vice versa.



Initialising the Variable
The standard way is to include this code at the top of the script:

```
var count=context.get('count') || 0;
```

Which means- If count doesn't exist in the context object then make our local variable count zero; otherwise assign the stored value to our local variable count. You can use multiple variables e.g.

```
var count=context.get('count') || 0;
var count2=context.get('count2') || 0;
You can also use an object e.g
var local=context.get('data') || {};
if (local.count===undefined) //test exists
{
  local.count=0;
}
```

In the code above data is an object stored in the context object and local is our local object. Here is an example script that uses a single variable as a counter in function 1:

```
var count=context.get('count') || 0;
count +=1;
msg.payload="F1 "+msg.payload+" "+count;
context.set('count',count);
return msg;
```

Here is an example script that uses a object variable as a counter in function 2::

```
var local=context.get('data') || {};
if (local.count===undefined)//test exists
{
  local.count=0;
}
local.count +=1;
msg.payload="F2 "+msg.payload+" "+local.count;
context.set('data',local);
return msg;
```

If look at the code for function 1 and function 2 you will see that they use the same counter variable. However when you click the inject node for function 1 you see the counter value is 1 and then the inject node for function 2 then counter is also 1. This shows that the counters are local to each function and not shared between functions.
Note: Currently if you restart the flow the variables are reset but this is likely to change

Using the Flow Object
You use the flow object in the same way as the context object. To retrieve values stored in the flow object use:

```
var count=flow.get('count') || 0
;
```

and to store values use:

```
low.set('count',count);
```

This time you should notice that the functions can share variables stored in flow objects.

Using the Global Object
You use the flow object in the same way as the context object.
To retrieve values stored in the flow object use:

```
var count=global.get('count') || 0;
```

and to store values use:

```
global.set('count',gcount);
```

This time you should notice that the functions can share variables stored in the global object even across flows.

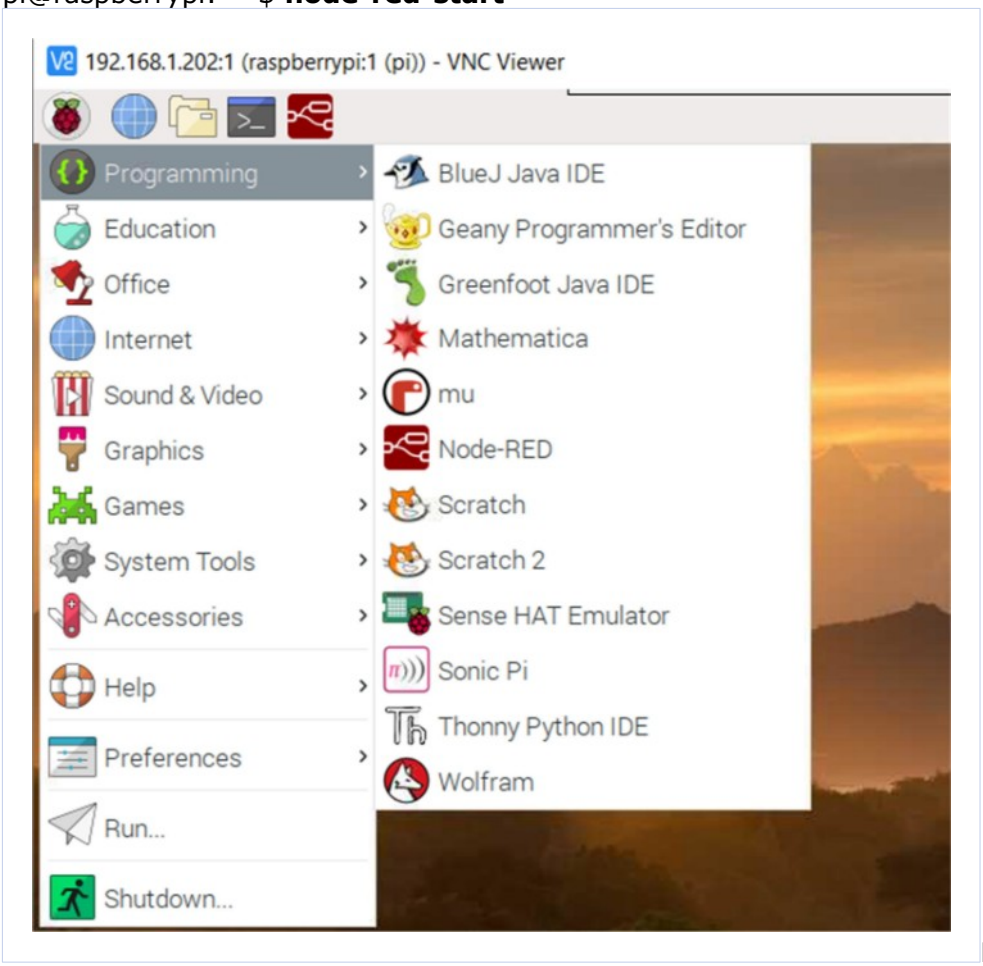
Exchange Data Between Nodes Without Wires
With Node-Red messages are usually passed between nodes using wires.
However it is possible to pass data between nodes with them being wired together using flow and global objects.

Selektor: Beginnen met Node-RED

Dogan Ibrahim (Groot-Brittannië)

De afgelopen jaren zijn blokgebaseerde visuele programmeertools steeds verder ontwikkeld en neemt het gebruik ervan hand over hand toe. Het idee hier is dat de programmeur een set visuele 'blokken' krijgt plus alles wat nodig is, is om deze bouwstenen op een logische manier te koppelen om het gewenste applicatieprogramma te creëren. Node-RED is een open source visuele blokgebaseerde programmeertool die nodes bevat voor het uitvoeren van uiterst complexe taken, waaronder internettoegang, Twitter, e-mail, HTTP, Bluetooth, MQTT en het aansturen van de GPIO-poorten van populaire computers zoals de Raspberry Pi , Arduino, ESP32 enzovoort. Het leuke van Node-RED is dat de programmeur niet hoeft te leren hoe hij of zij complexe programma's moet schrijven. Een e-mail kan bijvoorbeeld worden verzonden door een paar knooppunten samen te voegen en een paar regels eenvoudige code te schrijven.
Node-RED installeren
Node-RED is al geïnstalleerd op Raspberry Pi en kan vanaf het bureaublad worden gestart (**figuur 1**) of door de volgende opdracht in te voeren vanaf de opdrachtregel (**figuur 2**):

pi@raspberrypi: ~ $ **node-red-start**



Figuur 1. Node-RED starten op de Raspberry Pi.

```
pi@raspberrypi:~ $ node-red-start

Start Node-RED

Once Node-RED has started, point a browser at http://192.168.1.202:1880
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.
```

Figuur 2. Node-RED starten vanaf de opdrachtregel.

Node-RED wordt gestart als een service; na de start verschijnt een lijst met geldige opdrachten om het te starten en te stoppen, zoals te zien in figuur 2. Merk hier op dat 192.1681.202 het IP-adres is van de Raspberry Pi van de auteur. U kunt het IP-adres van uw eigen Raspberry Pi gemakkelijk vinden door de opdracht ifconfig in te voeren vanaf de opdrachtregel; zoek dan de regel die begint met wlan0:

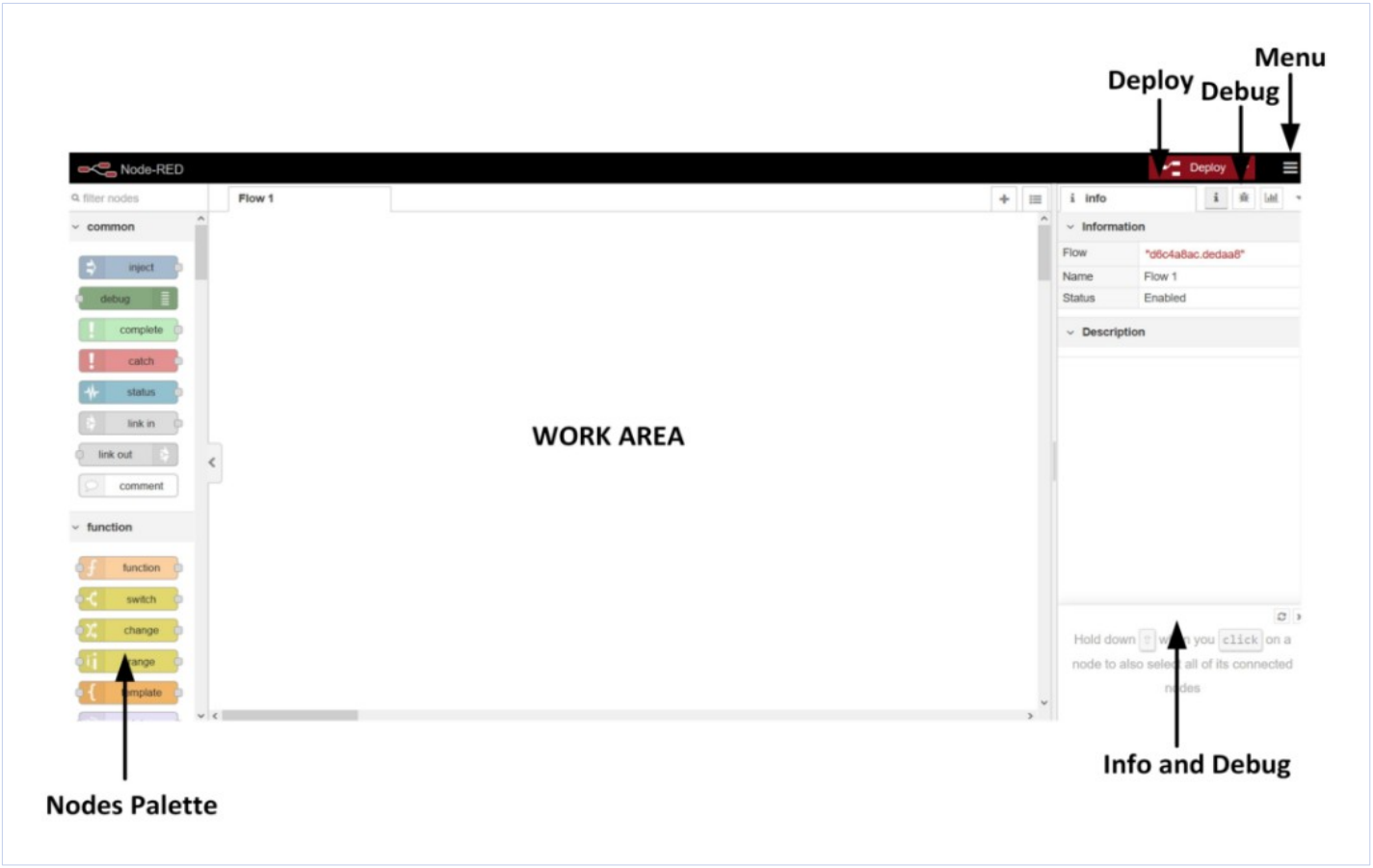
Nu moet u uw browser starten en de volgende link op uw PC invoeren om de Node-RED GUI in uw browser te starten:

https://<uw IP-adres>:1880

Voor Raspberry Pi van de auteur is dit bijvoorbeeld:

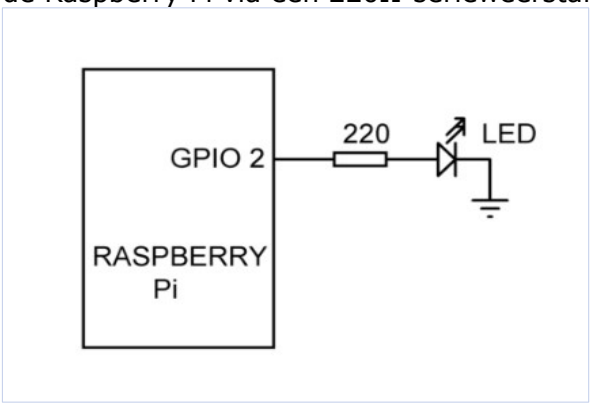
https://192.168.1.202:1880

Figuur 3 toont het Node-RED-opstartscherm. Dit bestaat in principe uit 3 secties: links ziet u een lijst met nodes die standaard beschikbaar zijn wanneer het programma is geïnstalleerd. Gebruikers kunnen echter extra knooppunten uit verschillende internet-bronnen toevoegen. Het middelste gedeelte is het werkgebied waar de te gebruiken knooppunten naar toe worden gesleept en neergezet om te worden gekoppeld tot een flow-programma. Rechts ziet u het info- en debuggebied, dat erg handig kan zijn tijdens het ontwikkelen van programma’s.



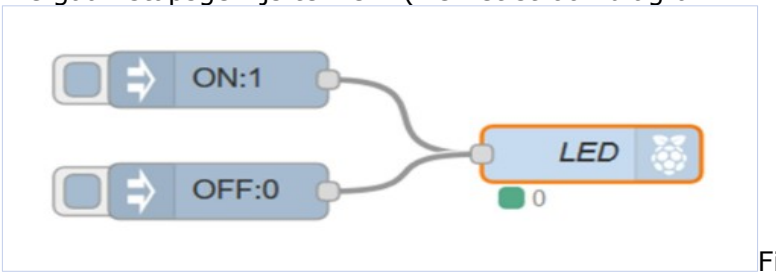
Figuur 3. Het Node-RED startmenu.

De eenvoudigste manier om te begrijpen hoe u Node-RED moet gebruiken, is natuurlijk een eenvoudig voorbeeld uit te proberen. In dit voorbeeld is een LED aangesloten op GPIO 2 van de Raspberry Pi via een 220Ω-serieweerstand, zoals getekend in **figuur 4**.



Figuur 4. Schema van het voorbeeld.

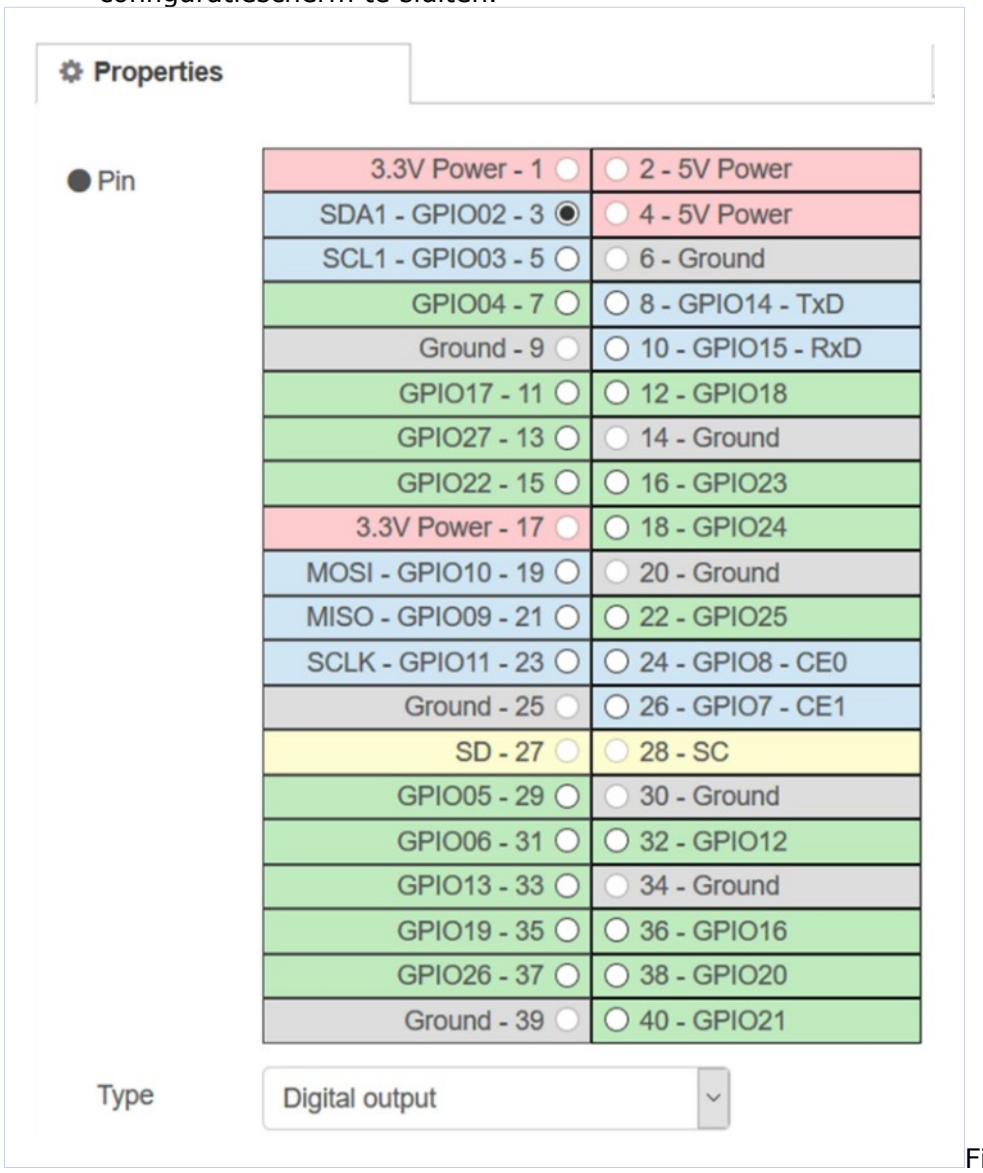
In dit eenvoudige project gaan we de LED aansturen vanuit een Node-RED flow-programma. We gaan stapsgewijs te werk (zie het stroomdiagram in **figuur 5**):



Figuur 5. Flow-diagram van het voorbeeld.

- Klik, sleep en plaats een inject node naar het werkgebied. Deze nodes worden gebruikt om berichten (een string, een getal enzovoort) in een flow te 'injecteren'. In dit voorbeeld injecteren we 1 of 0 om de LED respectievelijk AAN of UIT te zetten.

- Dubbelklik op de inject node om die te configureren. Stel de Payload in op ‘number’ en voer 1 in. Als u op het vierkante vakje links van deze node klikt, verschijnt ‘1’ weergegeven om de LED in te schakelen.
- Stel het Topic van de inject node in op ON zodat we de functie van deze node kunnen zien. Klik op Done om het configuratiescherm te sluiten.
- Klik, sleep en plaats een andere inject node zoals in figuur 5, stel de Payload in op 0 en het Topic op OFF. Als u op deze node klikt, wordt een ‘0’ uitgevoerd om de LED uit te schakelen.
- Klik, sleep en plaats een rpi gpio out node in het werkgebied. Deze node wordt gebruikt om digitale of PWM-data naar een GPIO-poort van de Raspberry Pi te sturen. Verbind de node zoals getoond in figuur 5.
- Dubbelklik op de rpi gpio out node om deze te configureren zoals getoond in **figuur 6**. Stel Pin in op GPIO2, Type op Digital output klik op Initialise pin state en stel de beginstatus in op 0. Geef de node de naam LED. Klik op Done om het configuratiescherm te sluiten.



Figuur 6. Configureren van de RPi gpio-out node.

Hiermee is ons flow-programma afgerond. Klik op Deploy om het programma te compileren en de uitvoerbare code te genereren. Zorg ervoor dat er in deze fase van het ontwerp geen foutmeldingen optreden. Klik op de vierkante knop aan de linkerkant van de ON inject node; de LED moet nu oplichten. Als u op de knop van de andere inject node klikt, wordt de LED uitgeschakeld.

Maar er is meer!

In deze inleiding hebben we aan de hand van een uiterst eenvoudig voorbeeld gezien wat Node-RED is en hoe het kan worden gebruikt. Een van de sterke punten van Node-RED is dat het wordt ondersteund door grote communities en dat er veel nodes zijn ontwikkeld door derden, die dan gratis beschikbaar zijn en in complexe projecten kunnen worden ingezet. De openweathermap node presenteert bijvoorbeeld het weer in uw omgeving (temperatuur, vochtigheid, luchtdruk, windsnelheid, neerslaggegevens enzovoort). En dat overal ter wereld. Een weerbericht-project kan in minder dan een uur worden gebouwd met behulp van deze en een paar extra nodes. Het uitvoeren van zo'n project met behulp van externe sensoren is meestal kostbaar, en ook kan de ontwikkeling veel tijd in beslag nemen. Node-RED ondersteunt op UDP en TCP gebaseerde WiFi-communicatie met een enkele node. Met deze functie kunnen gebruikers draadloze gegevens verzenden en ontvangen van andere WiFi-apparaten zoals mobiele telefoons, tablets, PC's enzovoort.
Enkele andere knooppunten die voor u interessant kunnen zijn en die gratis vanuit het internet kunnen worden geïnstalleerd, zijn:

- Dashboard
- Bluetooth
- Amazon Alexa
- LCD (I2C en parallel)
- Wereldkaart
- A/D-converter
- Ultrasone sensor

Node-RED is niet alleen voor de Raspberry Pi. Het kan ook worden gebruikt (hoewel beperkt) met de Arduino- en ESP32-processoren, of met een combinatie van Raspberry Pi en Arduino, of Raspberry Pi en ESP32.
Een van de belangrijkste Node-RED nodes is de function node. Deze node kan worden geconfigureerd met meerdere uitgangen; gebruikers kunnen programmacode schrijven tijdens de configuratie van dit knooppunt. Nadat we bijvoorbeeld de lokale weergegevens hebben verkregen met behulp van de openweathermap node, willen we daar misschien de lokale temperatuur- en vochtigheidsgegevens uit ophalen. Dat kan eenvoudig worden gerealiseerd door een paar instructies in de function node te schrijven. Geïnteresseerde lezers vinden meer informatie via de weblinks [1] [2] [3] [4] [5] of in het Node-RED-boek van de auteur.

(200164-04)