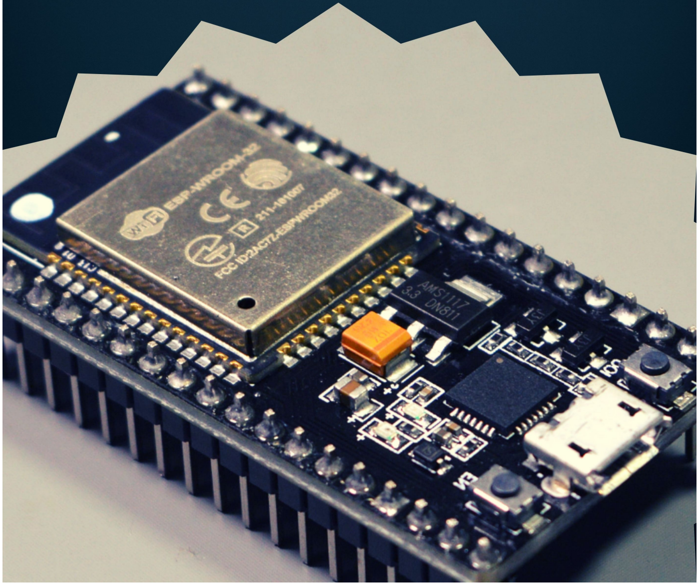
Tech Explorations
www.techexplorations.com

ESP32 NODE-RED

GET THE MOST OUT OF YOUR ESP32 WITH ARTICLES FROM THE TECH EXPLORATIONS BLOG



Peter Dalmaris, PhD

ESP32 and Node-RED

Get the most out of your ESP32 with articles from the Tech Explorations Blog

Welcome to this special collection of articles, meticulously curated from the Tech Explorations blog and guides. As a token of appreciation for joining our email list, we offer these documents for you to download at no cost. Our aim is to provide you with valuable insights and knowledge in a convenient format. You can read these PDFs on your device, or print.

Please note that these PDFs are derived from our blog posts and articles with limited editing. We prioritize updating content and ensuring all links are functional, striving to enhance quality continually. However, the editing level does not match the comprehensive standards applied to our Tech Explorations books and courses.

We regularly update these documents to include the latest content from our website, ensuring you have access to fresh and relevant information.

License statement for the PDF documents on this page

Permitted Use: This document is available for both educational and commercial purposes, subject to the terms and conditions outlined in this license statement.

Author and Ownership: The author of this work is Peter Dalmaris, and the owner of the Intellectual Property is Tech Explorations (https://techexplorations.com). All rights are reserved.

Credit Requirement: Any use of this document, whether in part or in full, for educational or commercial purposes, must include clear and visible credit to Peter Dalmaris as the author and Tech Explorations as the owner of the Intellectual Property. The credit must be displayed in any copies, distributions, or derivative works and must include a link to https://techexplorations.com.

Restrictions: This license does not grant permission to sell the document or any of its parts without explicit written consent from Peter Dalmaris and Tech Explorations. The document must not be modified, altered, or used in a way that suggests endorsement by the author or Tech Explorations without their explicit written consent.

Liability: The document is provided "as is," without warranty of any kind, express or implied. In no event shall the author or Tech Explorations be liable for any claim, damages, or other liability arising from the use of the document.

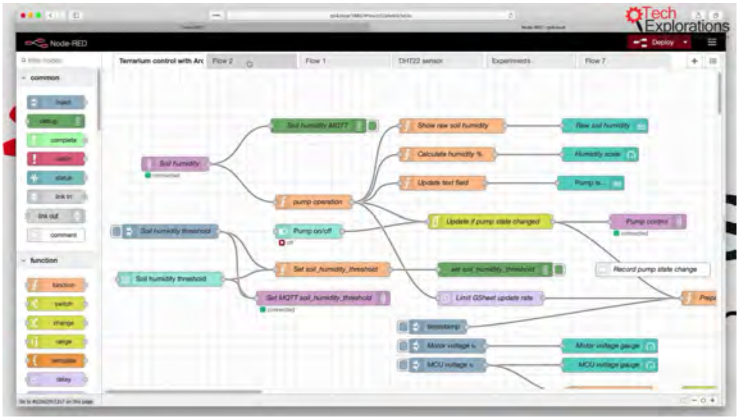
By using this document, you agree to abide by the terms of this license. Failure to comply with these terms may result in legal action and termination of the license granted herein.

1. Introduction to Node-RED, examples and documentation

Node-RED guide series

Introduction to Node-RED: examples and documentation

The Guides in this series are dedicated to Node-RED. Learn about Node-RED, understand what you can do with it with the help of examples, how to install it, and how to use some of the basic nodes that come with it.



Hi and welcome to the Node-RED guide series from Tech Explorations.

In this guide, you will learn about Node-RED through examples. Starting with this article, and continuing with the rest of the articles in this series, I'll show you how to install Node-RED on your Raspberry Pi, explain the use of some of its most important nodes and how they work inside simple flows.

What is Node-RED?

Node-RED is a programming tool.

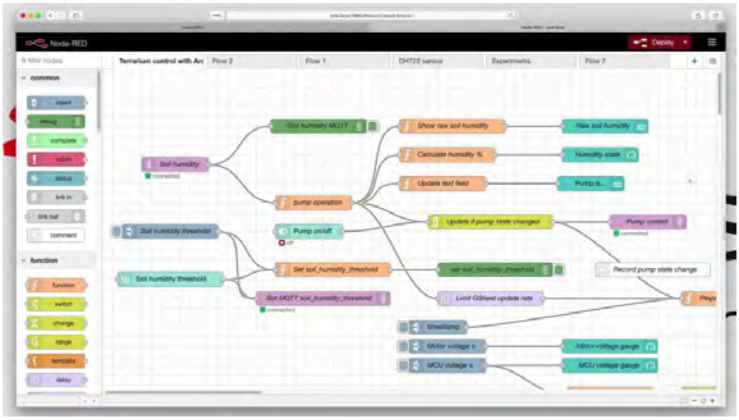
Actually, a better term to describe it is as a “programming environment”.

Node-RED uses graphical flows and nodes, which have individual components in a flow to essentially create a program. What I really like about Node-RED is that it is both graphical, so it gives you the visual capability of creating a program but, also, it allows you a lot of functional control through JavaScript. JavaScript is the programming language that is underlying Node-RED.

Node-RED is a programming environment that operates inside a browser.

You use Node-RED to create graphical programs, which are called flows.

Flows are composed of nodes, which are rectangular objects that you see in the example below.


An example flow from the course.

To construct a flow we use a drag and drop interface. With this interface, it is easy to assemble nodes in configurations that result is a program that does something useful. Each node is pre-programmed to do something specific (similar to a function in a text language line Python or Ruby). You can create custom nodes that contain your choice of JavaScript, similar to custom functions in text languages.Node-RED comes with several built-in nodes, but you can also install third-party nodes. Just like in the Arduino, you can install third-party libraries, and I'm going to show you more about this in a moment.The name Node-RED comes from the underlying technology on which it is built, which is Node.js, which is a JavaScript framework. It's a very lightweight development environment and runtime environment, which makes it excellent for creating applications that are supposed to be very nimble and very fast in their execution, so they can run on low-cost hardware such as the Raspberry Pi.Node-RED is open source. And as a result, there's a lot of people that are contributing to it. It's been around for a long time, and it's really stable and used by hobbyists and large corporations alike.

Node-RED home on the Web

The home of Node-RED on the web is nodered.org.

Take a few minutes to browse through the Node-RED website. You will find information about its features, “flows” (the term that means “Node-RED program”), and “nodes”, which are the functional components of a flow. And just scrolling down this page, you will see its basic features.

Demonstration

In the video that I have included at the top of this page, you can watch a demonstration of the Node-RED. It starts around the third minute of the video.

I have done a fresh installation of Node-RED here installed on my Raspberry Pi 4. I am demonstrating one of the latest versions of the flow that makes up the brains of the terrarium controller from my video course.

In the demonstration I show:

- The Node-RED web-based flow editor.
- Use of tabs to run many flows at once.
- Dive into several nodes, such as the one for the DHT22 sensor, the MQTT-in node, the function node, the HTTP Post node, and more.

This flow implements a full automation control and IoT system, that works in tandem with an ESP32 to control a simple terrarium.

You can use specialized nodes to do things such as:

- inject timestamps if you want to keep track

of time,
- manually trigger another node,
- inject text, numbers, JSON or other data-types in the flow,
- print out debug notifications during the flow execution,
- implement websocket functionality

I encourage you to watch the demonstration in the video for the details.

Documentation

Node-RED has got some amazing documentation.

If you’re just starting now, I recommend you have a look at Getting Started, and then the User Guide.

In Getting Started you will find information on how to install the Node-RED on a variety of computers. This is the source that I also follow in the next article of this series, where I do a fresh install of Node-RED on my Raspberry Pi.

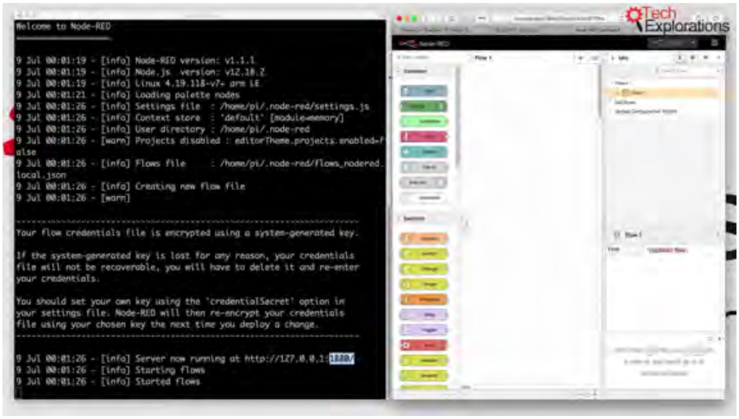
In the documentation you will find tutorials and a cookbook which are also very useful. The cookbook, for example, tells you exactly how to do specific things. So, a fairly comprehensive list here.Let’s move on to the next lecture now, where I’ll show you how to install Node-RED on the Raspberry Pi. Just keep in mind that the Raspbian OS or Raspberry Pi OS does come with Node-RED already installed, but I prefer to just go for a fresh install, so we have total control of what is running on your Raspberry Pi.

Node-RED guide series

Install Node-RED on the Raspberry Pi

Node-RED comes already installed in the Raspberry Pi operating system. Nevertheless, installing a fresh copy is easy, and you get the benefit of working with the latest and greatest version of the tool.

In this article I will show you how.



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

9 Jul 00:01:19 - [info] Node-RED version: v1.1.1
9 Jul 00:01:19 - [info] Node.js  version: v12.18.2
9 Jul 00:01:19 - [info] Linux 4.19.118-v7+ arm LE
9 Jul 00:01:21 - [info] Loading palette nodes
9 Jul 00:01:26 - [info] Settings file  : /home/pi/.node-red/settings.js
9 Jul 00:01:26 - [info] Context store  : 'default' [module=memory]
9 Jul 00:01:26 - [info] User directory : /home/pi/.node-red
9 Jul 00:01:26 - [warn] Projects disabled : editorTheme.projects.enabled=f
alse
9 Jul 00:01:26 - [info] Flows file     : /home/pi/.node-red/flows_nodered.
local.json
9 Jul 00:01:26 - [info] Creating new flow file
9 Jul 00:01:26 - [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.
---------------------------------------------------------------------

9 Jul 00:01:26 - [info] Server now running at http://127.0.0.1:1880/
9 Jul 00:01:26 - [info] Starting flows
9 Jul 00:01:26 - [info] Started flows
```

In this article, I'll show you how to install Node-RED on your Raspberry Pi.

To do that, I'll be following the instructions provided in the Node-RED documentation.

Update npm

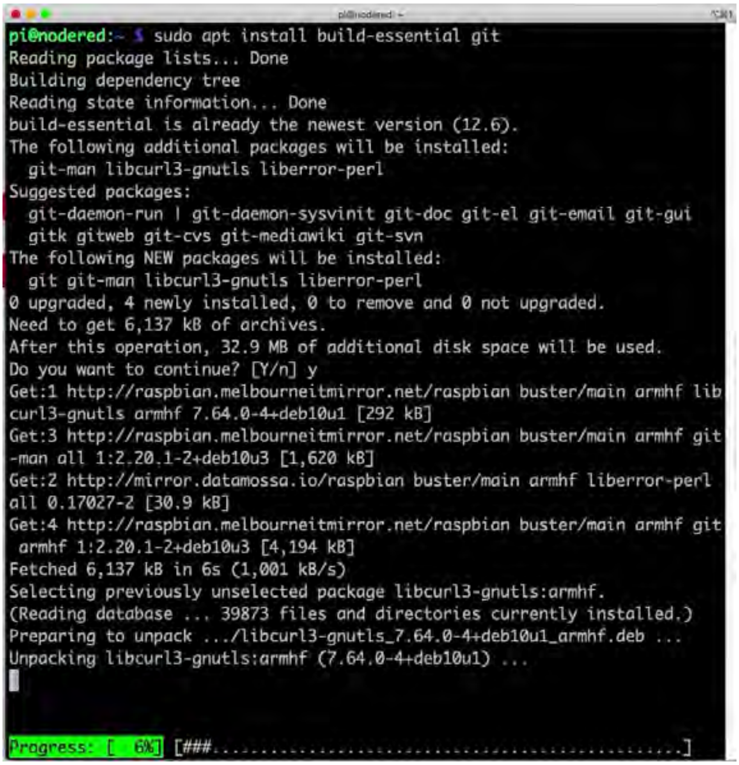
Login to your Raspberry Pi, as “pi”. The default password for the “pi” user is “raspberry”.

Start by installing the “git” repository tools, as well as the essential build tools that will be needed in the next step.

Here is the command:

~$ sudo apt install build-essential git

It looks like this:



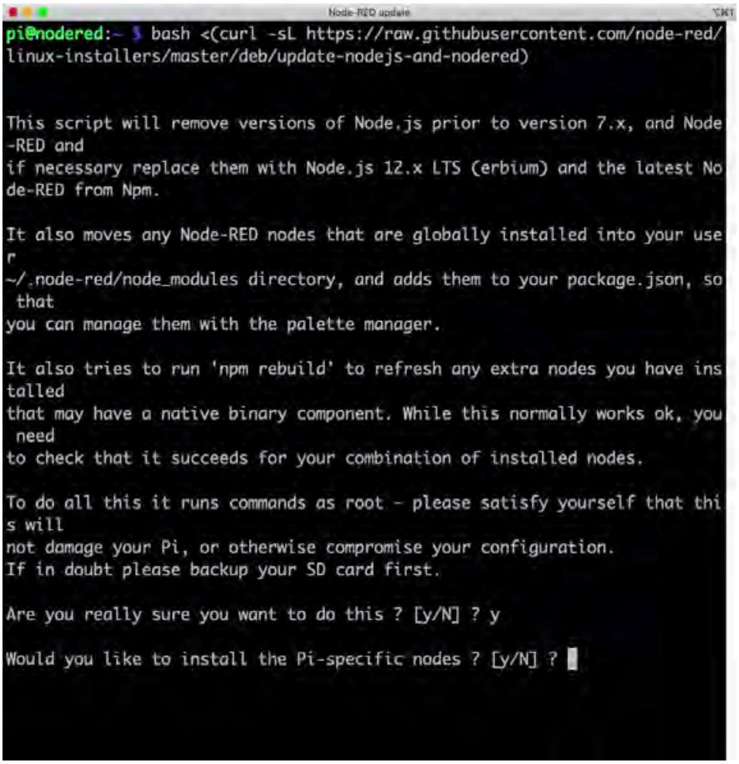
```
pi@nodered:~ $ sudo apt install build-essential git
Reading package lists... Done
Building dependency tree
Reading state information... Done
build-essential is already the newest version (12.6).
The following additional packages will be installed:
  git-man libcurl3-gnutls liberror-perl
Suggested packages:
  git-daemon-run | git-daemon-sysvinit git-doc git-el git-email git-gui
  gitk gitweb git-cvs git-mediawiki git-svn
The following NEW packages will be installed:
  git git-man libcurl3-gnutls liberror-perl
0 upgraded, 4 newly installed, 0 to remove and 0 not upgraded.
Need to get 6,137 kB of archives.
After this operation, 32.9 MB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 http://raspbian.melbourneitmirror.net/raspbian buster/main armhf lib
curl3-gnutls armhf 7.64.0-4+deb10u1 [292 kB]
Get:3 http://raspbian.melbourneitmirror.net/raspbian buster/main armhf git
-man all 1:2.20.1-2+deb10u3 [1,620 kB]
Get:2 http://mirror.datamossa.io/raspbian buster/main armhf liberror-perl
all 0.17027-2 [30.9 kB]
Get:4 http://raspbian.melbourneitmirror.net/raspbian buster/main armhf git
 armhf 1:2.20.1-2+deb10u3 [4,194 kB]
Fetched 6,137 kB in 6s (1,001 kB/s)
Selecting previously unselected package libcurl3-gnutls:armhf.
(Reading database ... 39873 files and directories currently installed.)
Preparing to unpack .../libcurl3-gnutls_7.64.0-4+deb10u1_armhf.deb ...
Unpacking libcurl3-gnutls:armhf (7.64.0-4+deb10u1) ...


Progress: [  6%] [###.............................................]
```

Install the “git” and “build-essential” tools.

Install Node-RED

Next, run this bash command to install a fresh copy of Node-RED:

~$ bash <(curl -sL https://raw.githubusercontent.com/node-red/linux-installers/master/deb/update-nodejs-and-nodered)



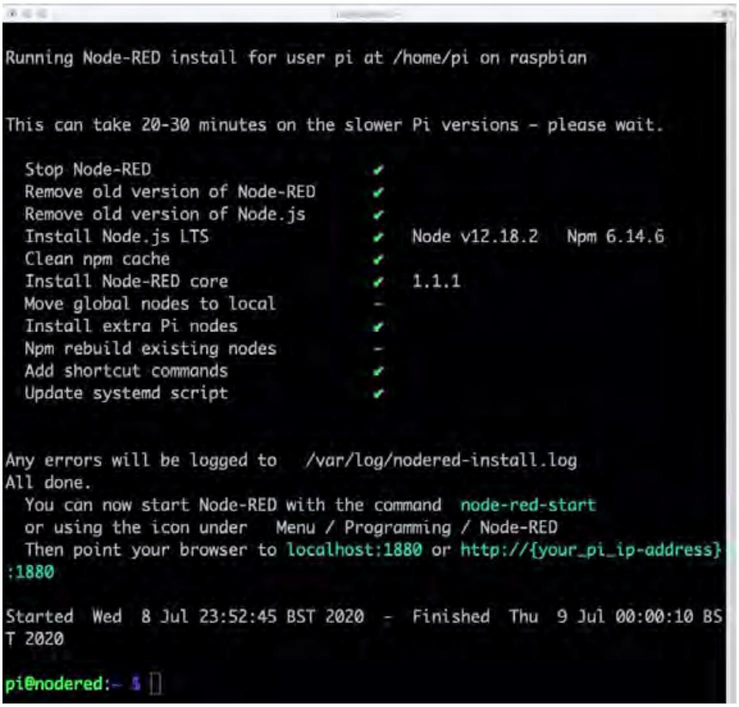
```
pi@nodered:~ $ bash <(curl -sL https://raw.githubusercontent.com/node-red/
linux-installers/master/deb/update-nodejs-and-nodered)


This script will remove versions of Node.js prior to version 7.x, and Node
-RED and
if necessary replace them with Node.js 12.x LTS (erbium) and the latest No
de-RED from Npm.

It also moves any Node-RED nodes that are globally installed into your use
r
~/.node-red/node_modules directory, and adds them to your package.json, so
 that
you can manage them with the palette manager.

It also tries to run 'npm rebuild' to refresh any extra nodes you have ins
talled
that may have a native binary component. While this normally works ok, you
 need
to check that it succeeds for your combination of installed nodes.

To do all this it runs commands as root - please satisfy yourself that thi
s will
not damage your Pi, or otherwise compromise your configuration.
If in doubt please backup your SD card first.

Are you really sure you want to do this ? [y/N] ? y

Would you like to install the Pi-specific nodes ? [y/N] ?
```

Type “y” in the two installer questions.

The installer will ask two questions before it begins the installation, to which you should respond with “Y”.

Around 20 minutes later, the installation will complete, and you will see information about your new Node-RED setup in the console.

Success looks like this:



```
Running Node-RED install for user pi at /home/pi on raspbian


This can take 20-30 minutes on the slower Pi versions - please wait.

  Stop Node-RED                       ✔
  Remove old version of Node-RED      ✔
  Remove old version of Node.js       ✔
  Install Node.js LTS                 ✔   Node v12.18.2   Npm 6.14.6
  Clean npm cache                     ✔
  Install Node-RED core               ✔   1.1.1
  Move global nodes to local          -
  Install extra Pi nodes              ✔
  Npm rebuild existing nodes          -
  Add shortcut commands               ✔
  Update systemd script               ✔


Any errors will be logged to   /var/log/nodered-install.log
All done.
  You can now start Node-RED with the command  node-red-start
  or using the icon under   Menu / Programming / Node-RED
  Then point your browser to localhost:1880 or http://{your_pi_ip-address}
:1880

Started  Wed  8 Jul 23:52:45 BST 2020  -  Finished  Thu  9 Jul 00:00:10 BS
T 2020

pi@nodered:~ $
```

Node-RED is now installed on your Raspberry Pi.

Node-RED is now installed. Notice that the installer suggests that you start the Node-RED service using the “node-red-start” command.

But wait a sec…

How to start Node-RED

There are a few different ways to start the Node-RED service.

The easiest is to follow the prompt of the installed, and just type:

```
~$ node-red-start
```

This will work fine, especially on a Raspberry Pi 4 with more than 2 GB of RAM.

For earlier Raspberry Pi's, or computers with limited RAM, it's better to specify the amount of RAM available like this:

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

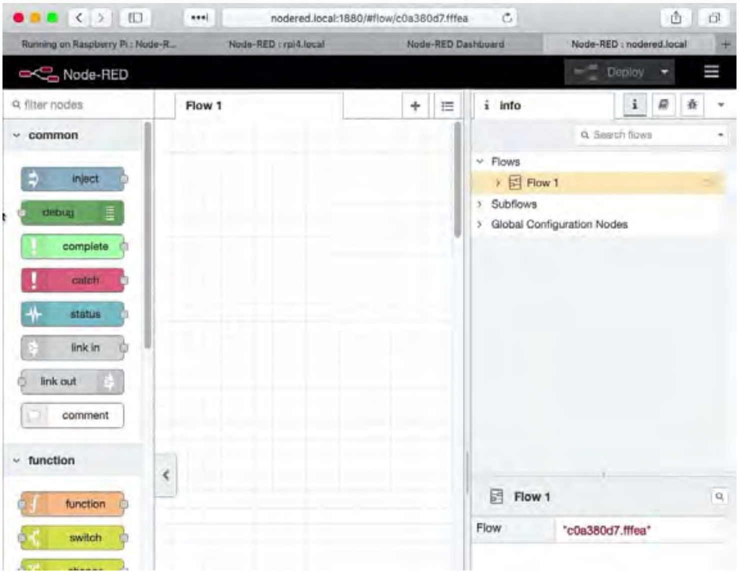
This will instruct the Node-RED service to quickly free up any unused memory so that other services of the operating system can use them.

Regardless of which method you choose, the Node-RED service will begin within a few seconds, and indicate that it is running on localhost, default port 1880.

You can point your browser to this URL:

http://nodered.local:1880

The URL "nodered.local" is the hostname of my Raspberry Pi. You can use your RPi IP address instead of the hostname.



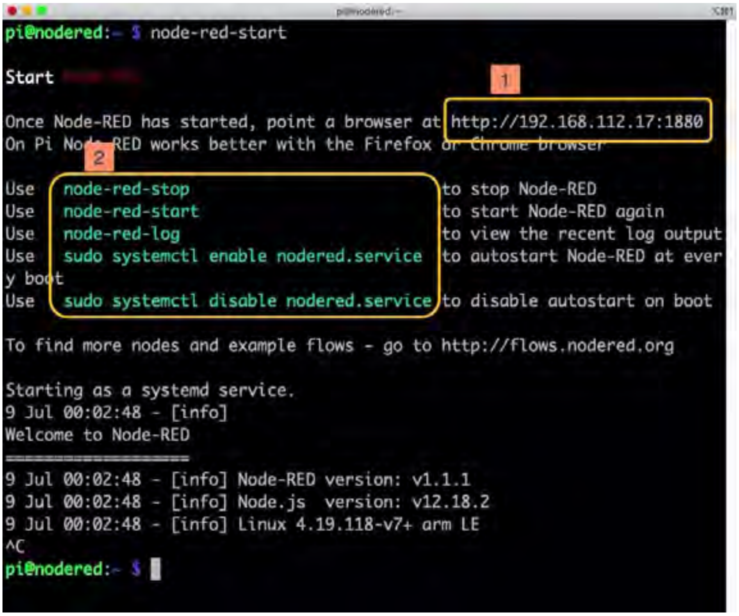
Node-RED running in my browser for the first time.

If you use one of the above methods to start Node-RED, you can stop the service by typing Ctr-C.

Run Node-RED as a service

Node-RED can run as a background service. The benefit of this is that you can set it to autostart when the operating system boots, instead of having to start it manually.

You can manually start the Node-RED service using the “node-red-start” command. This will produce this output:



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

Start [illegible]

Once Node-RED has started, point a browser at http://192.168.112.17:1880
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 ever
y 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.
9 Jul 00:02:48 - [info]
Welcome to Node-RED
===================
9 Jul 00:02:48 - [info] Node-RED version: v1.1.1
9 Jul 00:02:48 - [info] Node.js  version: v12.18.2
9 Jul 00:02:48 - [info] Linux 4.19.118-v7+ arm LE
^C
pi@nodered:~ $
```

Node-RED is installed, and already running on your Raspberry Pi.

This output contains the IP address and port number for the new Node-RED service (1), as well as the systemd commands for starting and stopping the service (2).

Node-RED is now running as a service. You can point your browser to the URL that is indicated in (1).

Autostart the Node-RED service on boot

To automatically start the Node-RED service on boot, use this command:

```
~$ sudo systemctl enable nodered.service
```

Now, when your Raspberry Pi starts, the Node-RED service with begin automatically.

Try this:

1. Type “sudo reboot”, to reboot the Raspberry Pi
2. Wait for a couple of minutes for the reboot to complete
3. Use your browser to access Node-RED

Node-RED should load in your browser, and operate normally.

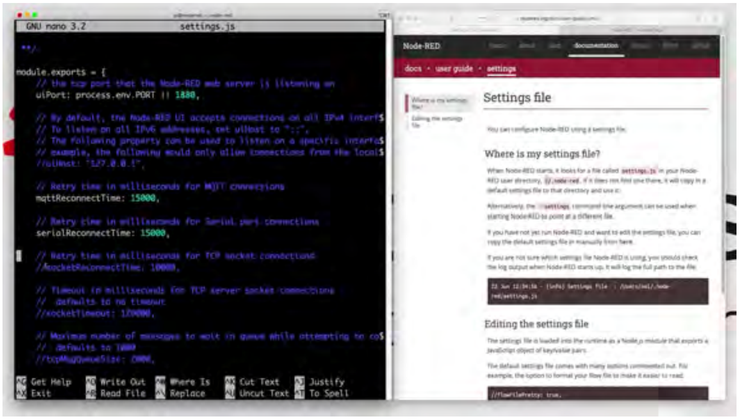
Node-RED offers many configuration options. I will review some of them in the next article.

Node-RED guide series

Node-RED Configuration

The Node-RED service has a configuration file.

In this file, you can set various configuration options and customize your Node-RED installation to better suit your needs.



```
GNU nano 3.2                    settings.js

**/

module.exports = {
    // the tcp port that the Node-RED web server is listening on
    uiPort: process.env.PORT || 1880,

    // By default, the Node-RED UI accepts connections on all IPv4 interf$
    // To listen on all IPv6 addresses, set uiHost to "::",
    // The following property can be used to listen on a specific interfa$
    // example, the following would only allow connections from the local$
    //uiHost: "127.0.0.1",

    // Retry time in milliseconds for MQTT connections
    mqttReconnectTime: 15000,

    // Retry time in milliseconds for Serial port connections
    serialReconnectTime: 15000,

    // Retry time in milliseconds for TCP socket connections
    //socketReconnectTime: 10000,

    // Timeout in milliseconds for TCP server socket connections
    //  defaults to no timeout
    //socketTimeout: 120000,

    // Maximum number of messages to wait in queue while attempting to co$
    //  defaults to 1000
    //tcpMsgQueueSize: 2000,

^G Get Help   ^O Write Out  ^W Where Is   ^K Cut Text   ^J Justify
^X Exit       ^R Read File  ^\ Replace    ^U Uncut Text ^T To Spell
```

Settings file

You can configure Node-RED using a settings file.

Where is my settings file?

When Node-RED starts, it looks for a file called settings.js in your Node-RED user directory, ~/.node-red. If it does not find one there, it will copy in a default settings file to that directory and use it.

Alternatively, the --settings command-line argument can be used when starting Node-RED to point at a different file.

If you have not yet run Node-RED and want to edit the settings file, you can copy the default settings file in manually from here.

If you are not sure which settings file Node-RED is using, you should check the log output when Node-RED starts up. It will log the full path to the file:

```
22 Jun 12:34:56 - [info] Settings file  : /users/nol/.node-red/settings.js
```

Editing the settings file

The settings file is loaded into the runtime as a Node.js module that exports a JavaScript object of key/value pairs.

The default settings file comes with many options commented out. For example, the option to format your flow file to make it easier to read:

```
//flowFilePretty: true,
```

At this point, you’ve got a fully functioning Node-RED installation on your Raspberry Pi. You can actually go ahead and start creating your flows.

But you should be aware that the Node-RED service is configurable via a settings text file. In this file, you can set various configuration options and customize your Node-RED installation to better suit your needs.

The Node-RED configuration file

You can find details about the Node-RED configuration file in the Node-RED documentation.On the Raspberry Pi, the location of the configuration file is:

```
/home/pi/.node-red
```

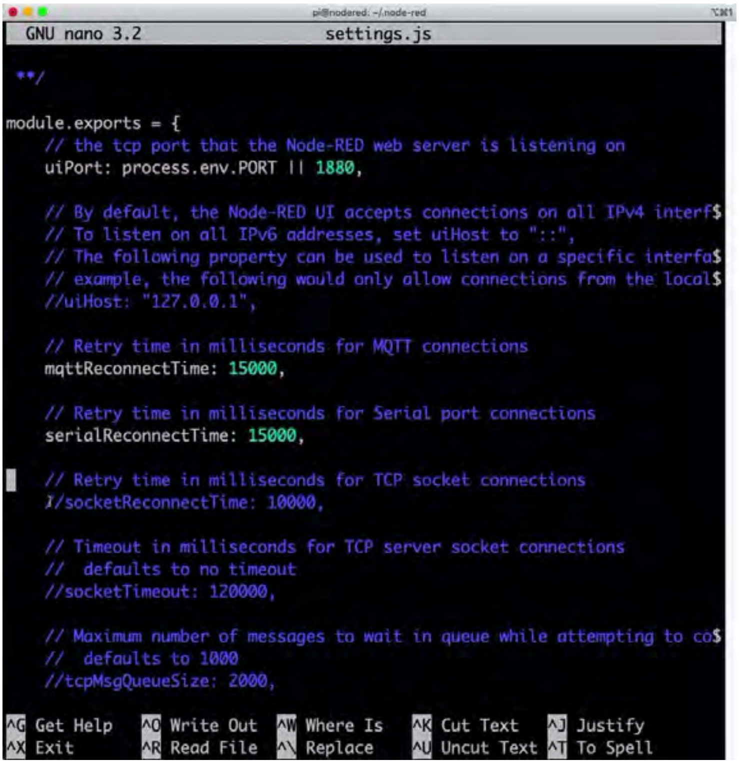
In the hidden ".node-red" directory, Node-RED stores your flows, nodes, and, of course, its configuration file.

The configuration file is named "settings.js".

Go in the node-red directory and open it using the nano editor:

```
pi@rpi4:~ $ cd .node-red/pi@rpi4:~/.node-red $ nano
settings.js
```

Below you can see a section of this file:


```
pi@nodered: ~/.node-red
 GNU nano 3.2                    settings.js

 **/

module.exports = {
    // the tcp port that the Node-RED web server is listening on
    uiPort: process.env.PORT || 1880,

    // By default, the Node-RED UI accepts connections on all IPv4 interf$
    // To listen on all IPv6 addresses, set uiHost to "::",
    // The following property can be used to listen on a specific interfa$
    // example, the following would only allow connections from the local$
    //uiHost: "127.0.0.1",

    // Retry time in milliseconds for MQTT connections
    mqttReconnectTime: 15000,

    // Retry time in milliseconds for Serial port connections
    serialReconnectTime: 15000,

    // Retry time in milliseconds for TCP socket connections
    //socketReconnectTime: 10000,

    // Timeout in milliseconds for TCP server socket connections
    //  defaults to no timeout
    //socketTimeout: 120000,

    // Maximum number of messages to wait in queue while attempting to co$
    //  defaults to 1000
    //tcpMsgQueueSize: 2000,

^G Get Help   ^O Write Out  ^W Where Is   ^K Cut Text   ^J Justify
^X Exit       ^R Read File  ^\ Replace    ^U Uncut Text ^T To Spell
```

Part of the Node-RED settings.js file.

Most of the configuration options are commented out. You can enable them by removing the two forward slashes at the start of the corresponding line.Here's a couple of interesting configuration options:

- "**flowFile**": this is the file that contains your flows. You can customise its location and name with this setting.
- "**uiPort**": this setting controls the port number under which the web user interface is accessible. The default is "1880", but you

can change it to something else.
- “**userDir**“: this setting controls the location of the Node-RED directory, which is where this configuration file, and the user flows and nodes are stored.
- “**nodesDir**“: you can set an additional location that contain nodes (apart from the default userDir). If you enable this setting, Node-RED will scan in nodeDir for additional nodes.
- “**adminAuth**“: you can protect the administrator or editor functions of your Node-RED service by setting up one or more user names and passwords. Each user may have their own set of credentials. Learn more about this in the documentation.
- And many more.

You can find a full list of configuration options in the documentation page.For the purposes of my Node-RED & ESP32 project, I have left my configuration with its default settings.

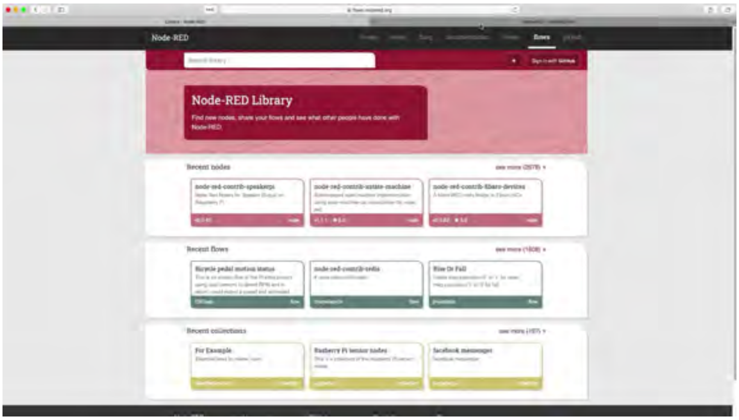
Let’s continue this series with the next article in which I will discuss Node-RED nodes.

4. Node-RED Nodes

Node-RED guide series

Node-RED Nodes

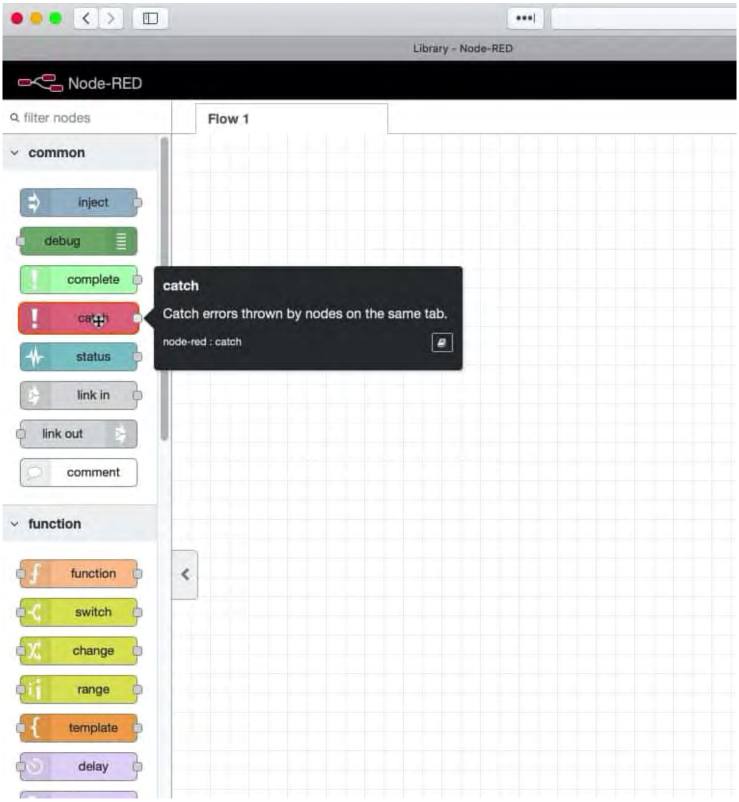
Learn about some of the most important nodes, have a look at their properties, learn how to configure them and learn how to install third-party nodes that are available in the Node-RED library.


Node-RED
Node-RED Library
Find new nodes, share your flows and see what other people have done with Node-RED
Recent nodes
Recent flows
Recent collections

In this article, I will show you some of the most important nodes, have a look at their properties, learn how to configure them, and learn how to install third-party nodes that are available in the Node-RED library.

The nodes toolbar

On the left toolbar, you can see the list of default or built-in nodes that come with a fresh installation of Node-RED. You can add more nodes from the Node-RED library, and, of course, create your own nodes.



catch

Catch errors thrown by nodes on the same tab.

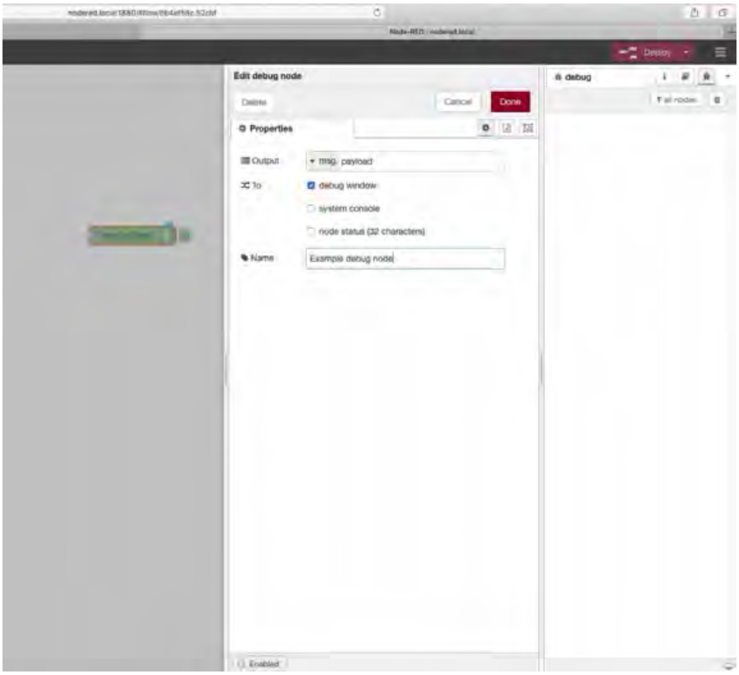
node-red : catch

All available nodes are available from the left toolbar.

Node example: “debug”

A very common node called “debug”. Find it under “common” and drag it on to the flow canvas. The purpose of the “debug” node is to output text to the debug window that can appear on the right toolbar.

In the image below, you can see a debug node on the canvas, with its properties tab and output tab open.



Output: msg. payload

To: [✓ debug window] system console node status (32 characters)

Name: Example debug node

The “debug” node with its properties tab and output tab open.

To reveal the properties of any node, double-click on the node. This will reveal the edit pane. Depending on what kind of node it is, the properties tab will have its own configuration options and set of widgets that you can interact with toset up the node.

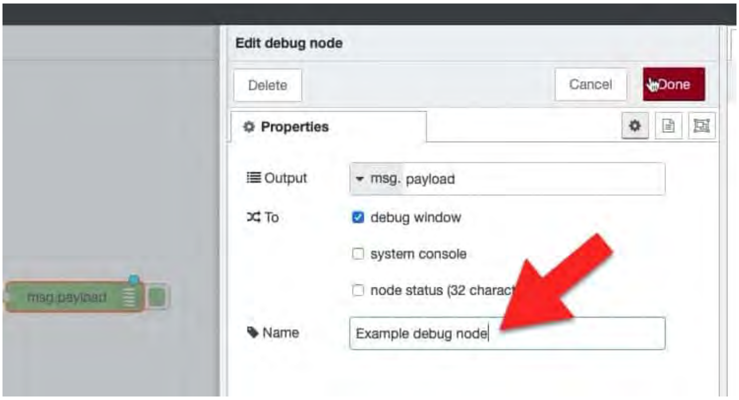
The debug node allow us to send a string of text to the debug pane that you can see in the image above. The string that is shown in the debug pane is passed on to the debug node from the previous node in the same flow. Of course, you can configure exactly what it is that your want the debug node to output.

The debug node is one of the most commonly used nodes in Node-RED flows. It is the equivalent of the “Serial.print()” function of an Arduino sketch.

Keep in mind that a debug node can send a string to the debug pane, but also to the system console. In most cases, you want to send it out to the debug console. You may want to send debug messages the system console, which is particularly useful if you are running Node-RED on the command line.

You can customize the name of a node. In this case, I'll call this debug node "example debug node." Just type the name you want in the "Name" attribute of the node.

You should always give your nodes appropriate names to make it easy to understand what is happening in a flow. This is similar to how we want to give reasonable names to variables and functions in a text programming language.

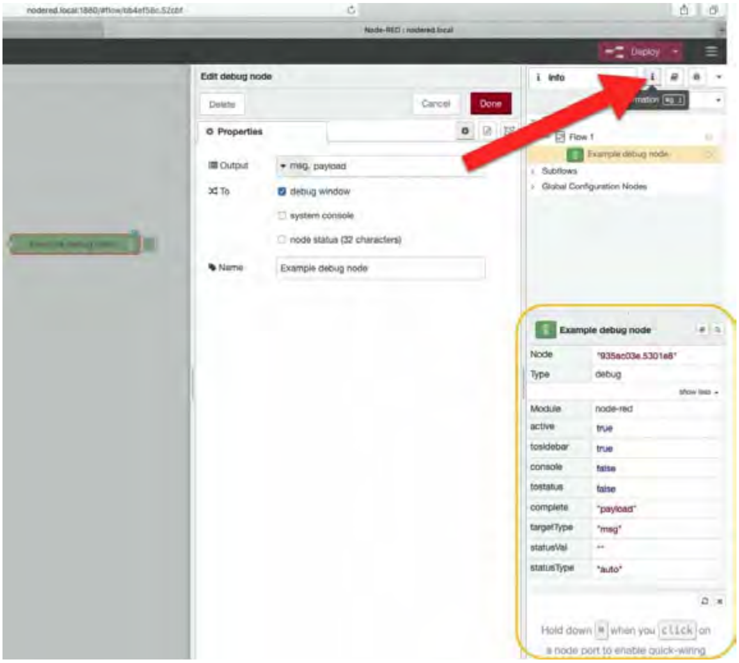


Output: msg. payload
To: [✓ debug window] system console node status (32 charac
Name: Example debug node

Give nodes reasonable names.

Node information

You can get information about the properties of a node by clicking on the "i" button. This will reveal the "info" tab.



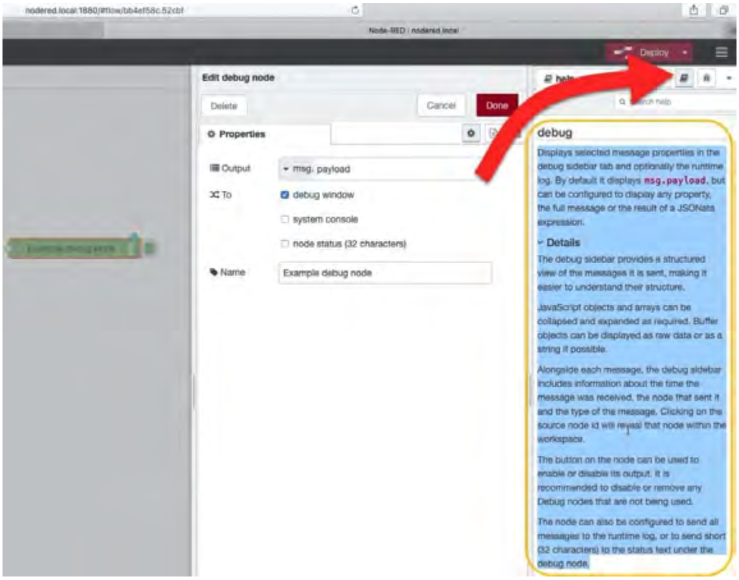
Node	"935ac03e.5301e8"
Type	debug
Module	node-red
active	true
tosidebar	true
console	false
tostatus	false
complete	"payload"
targetType	"msg"
statusVal	""
statusType	"auto"

Use the Info tab to see the node properties.

The information that you will see in the info tab depends on the type of node you are looking at, but all nodes have at least these attributes: node ID, type, and active.

Node documentation

Node-RED has a built-in documentation system. You can see the documentation attached to a node by clicking on the book icon.



Output: msg. payload
To: ☑ debug window ☐ system console ☐ node status (32 characters)
Name: Example debug node

debug

Displays selected message properties in the debug sidebar tab and optionally the runtime log. By default it displays `msg.payload`, but can be configured to display any property, the full message or the result of a JSONata expression.

Details

The debug sidebar provides a structured view of the messages it is sent, making it easier to understand their structure.

JavaScript objects and arrays can be collapsed and expanded as required. Buffer objects can be displayed as raw data or as a string if possible.

Alongside each message, the debug sidebar includes information about the time the message was received, the node that sent it and the type of the message. Clicking on the source node id will reveal that node within the workspace.

The button on the node can be used to enable or disable its output. It is recommended to disable or remove any Debug nodes that are not being used.

The node can also be configured to send all messages to the runtime log, or to send short (32 characters) to the status text under the debug node.

Use the “Help” tab to see the node documentation.

Node example: “inject”

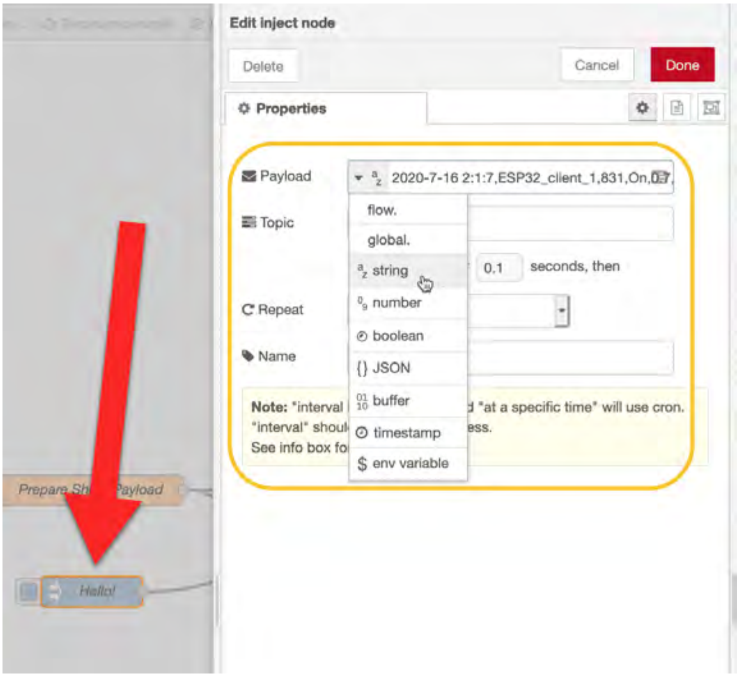
Let’s have a look at another node: inject.

The “inject” node is useful when you want to inject a string into another node.

In the example below, I have exposed the Edit pane of an inject node named “Hello”. In the properties tab, you can see the various types of payloads that the inject node can inject to another node (such as the “debug” node).

The payload can be a fixed string, a number, a boolean, a JSON expression, a flow variable, among other things.

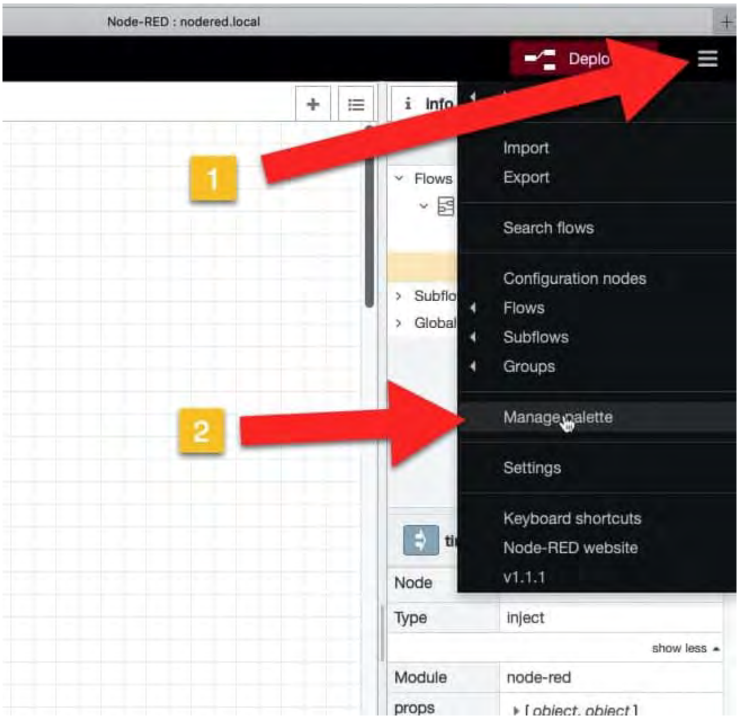
In the example below, I have set the payload to a fixed string as part of my testing along side a Google Sheet node.



The arrow points to the Hello “inject” node.

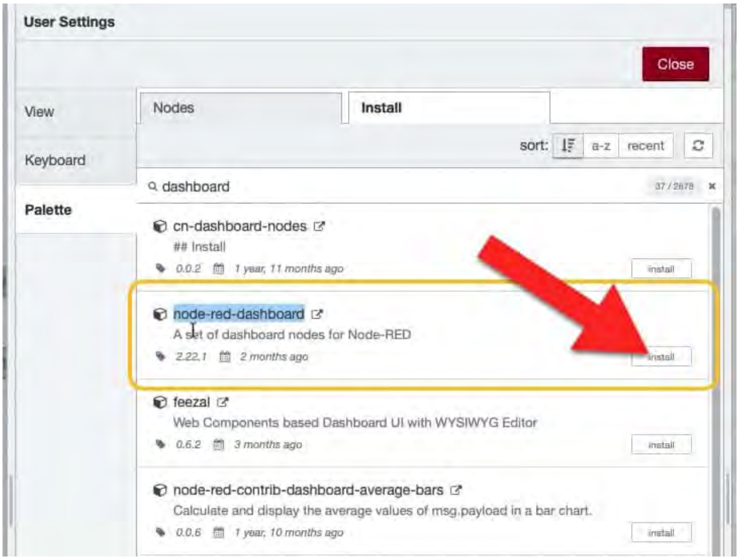
Manage palettes

To install a node or flow from the Node-RED library, you will use the Palettes Manager. You can access the Palettes Manager via the burger menu, at the top right corner of the Node-RED editor window.



Manage palettes from the burger menu.

Via the Manage Palette tool you can install contributed flows, nodes or collection of nodes.A particularly useful collection of nodes that I will be using a lot in this project is called “node-red-dashboard”. To install it, simply search for “dashboard”, find it in the list with the search results, and click on “install”.



dashboard 37 / 2678

cn-dashboard-nodes
Install
0.0.2 1 year, 11 months ago install

node-red-dashboard
A set of dashboard nodes for Node-RED
2.22.1 2 months ago install

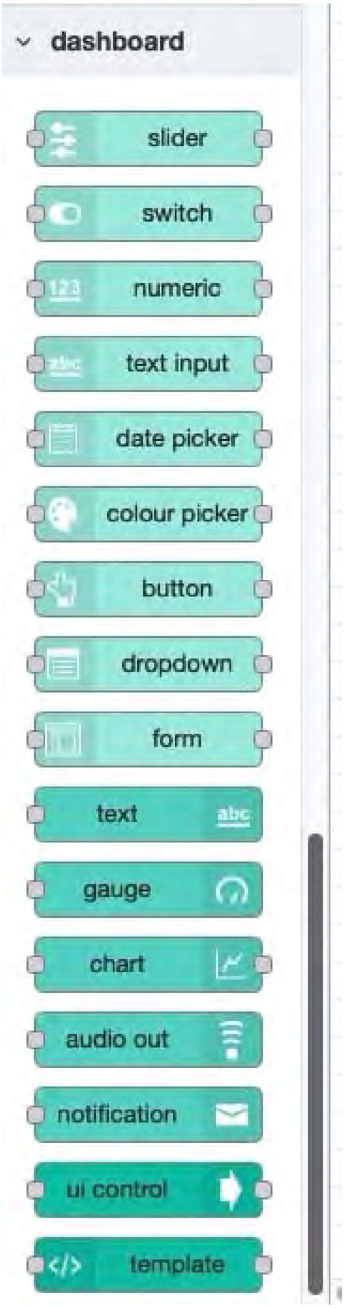
feezal
Web Components based Dashboard UI with WYSIWYG Editor
0.6.2 3 months ago install

node-red-contrib-dashboard-average-bars
Calculate and display the average values of msg.payload in a bar chart.
0.0.6 1 year, 10 months ago install

The “node-red-dashboard” node.

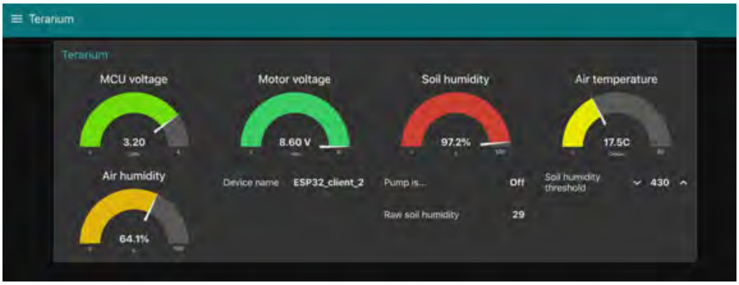
While you are in the Palette Manager, also search for and install the “dht22” node (look for the one named “node-red-contrib-dht-sensor”).

The new nodes will appear in the left tool bar. Here’s what the new nodes for the dashboard look like:


dashboard
slider
switch
numeric
text input
date picker
colour picker
button
dropdown
form
text
gauge
chart
audio out
notification
ui control
template

These nodes below to the dashboard group.

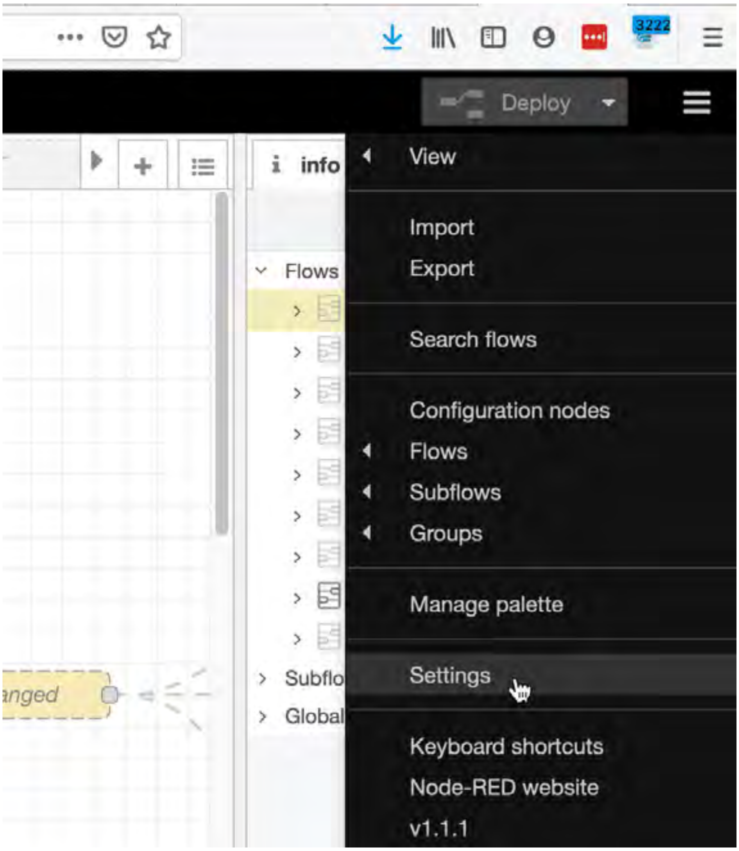
You can use these nodes in your flows to create dashboards like this:



An example Node-RED dashboard.

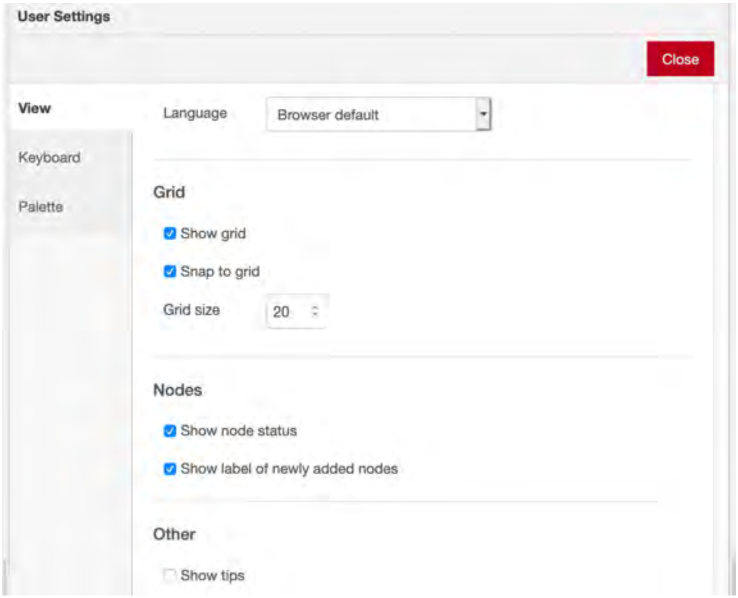
Settings

You can access the Settings pane from the burger menu:



Access the Settings pane from the burger menu.

From the Settings pane, you can configure your working area, the application language, the grid size, etc. You can also set your favorite keyboard shortcutts.


User Settings

Close

View

Keyboard

Palette

Language: Browser default

Grid

☑ Show grid

☑ Snap to grid

Grid size: 20

Nodes

☑ Show node status

☑ Show label of newly added nodes

Other

☐ Show tips

The User Settings pane.

In order to make nodes do something useful, we need to connect them together into flows.

In the next article in this series, I will to show you how to create simple flows using a handful of basic nodes.

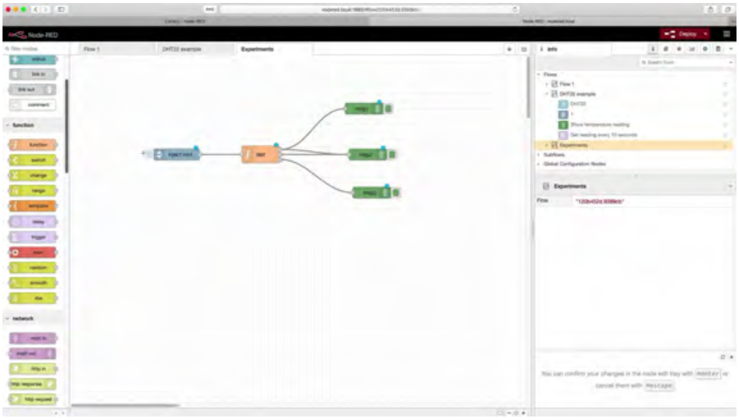
5. Node-RED flows

Node-RED guide series

Node-RED Flows

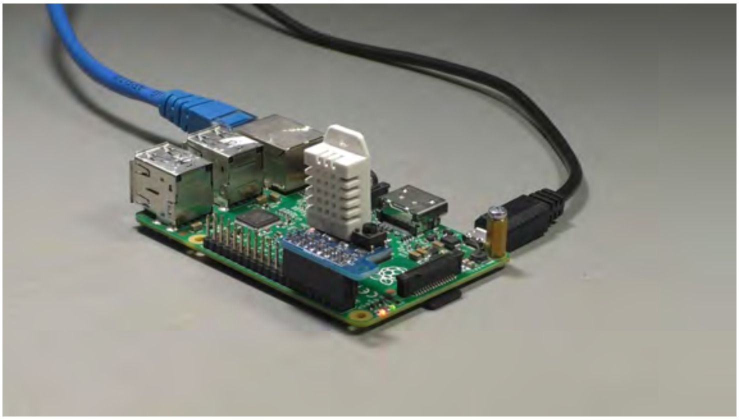
Nodes, on their own, are not very useful. For them to be useful, they must to be connected and configured inside flows.

In this article, I will show you how to create your first Node-RED flows.



In the previous article in this series, I did quick introduction of Node-RED nodes. On their own, nodes are not very useful. For them to be useful, they must to be connected to other nodes and configured inside flows. In this article I'm will show you a few simple examples of nodes configured into flows to help you get started. For the examples that follow, I will be using the DHT22 node. For this purpose, I have connected a DHT22 sensor to my Raspberry Pi.

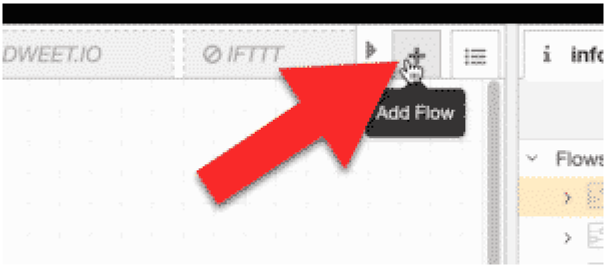
If you don't know how to connect a DHT22 sensor to your Raspberry Pi, please read this article from the Raspberry Pi guides series.



My Raspberry Pi2 “wearing” my DHT22 HAT.

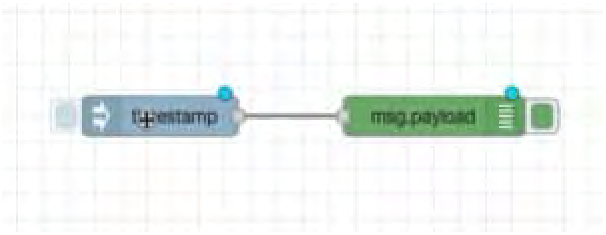
Create a new flow

To create a new flow, click on the “+” button, on the top right corner of the designer canvas.



Create a new flow.

This flow will contain only two nodes: “inject” and “debug”. Find the two nodes unde the “common” group, and arrange them in the canvas like in the screenshot below:



A flow with two nodes.

This flow will perform this operation: when I click on the inject node button, the string “HELLO”, will be sent over to the next node. In this case, the next node is the “debug” node, which will print the message in the debug window.

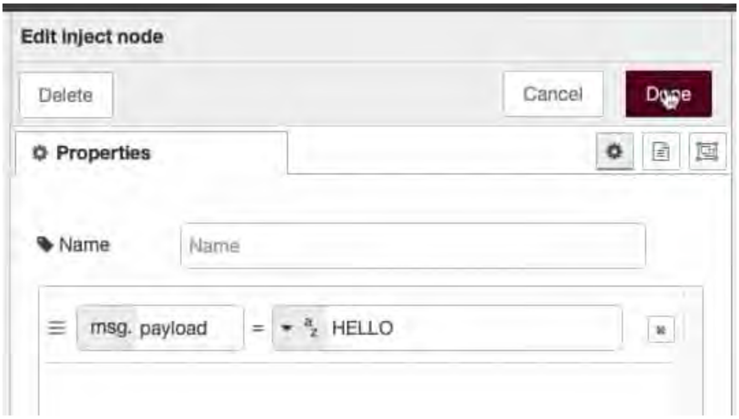
The inject node is on the left, and the debug node on the right. Connect them by clicking and holding on the circle of the inject node, then place the mouse pointer over the circle of the debug note, and release.

Next, configure the inject node. Double-click on it to reveal its properties.

In the “msg” field, select “payload”, and for the payload type select the string option (designated by the “az” icon).

In the payload field, type “HELLO”.

Click on “Done” to exit the inject node pane.


Edit inject node

Delete | Cancel | Done

Properties

Name: Name

msg. payload = HELLO

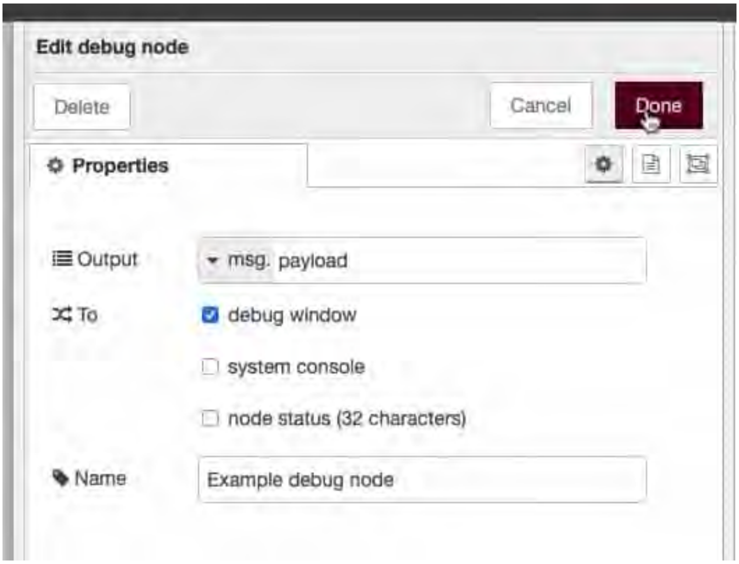
The inject node properties.

Next, double-click on the debug node. There’s only a couple of attributes to set.

The first one is the “output”. In the “inject” node, you stored the string in the “msg.payload” variable. Therefore, in the debug node, you should read the output from the “msg.payload” variable.

Also, in the Name attribute, give this node the name “Example debug node”.

The debug node edit pane will look like this:


Edit debug node

Delete | Cancel | Done

Properties

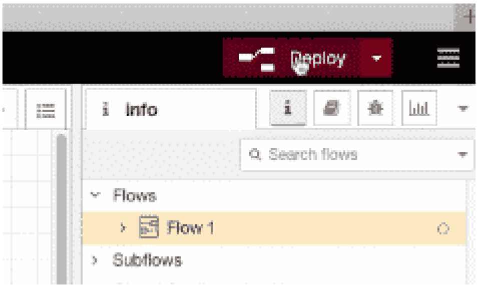
Output: msg. payload

To: [✓ debug window] system console, node status (32 characters)

Name: Example debug node

The “debug” node properties.

Let’s test out the new flow. Click on the red Deploy button:



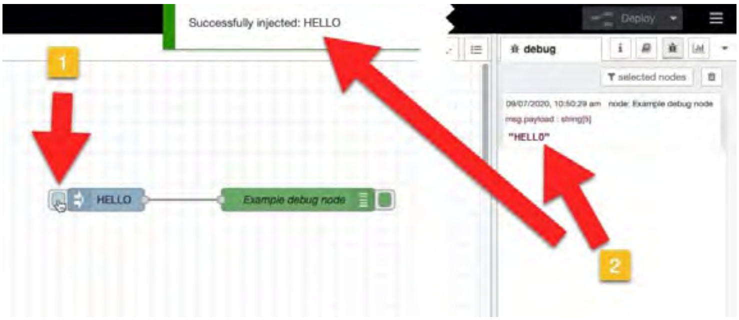
Click on “Deploy” to execute the flow.

Your flow is now running. Click on the button of the inject node, and notice that the string “HELLO” appears in the debug pane.

If the debug pane is not visible, click on the small button with the bug icon.

The debug pane will show a new message each time you click on the inject node button. The message will include a timestamp, the name of the debug node that printed it, and the name of the variable that contained it (“payload”).

Also notice that each time that you click on the inject button, a green notification appears in the Node-RED editor window.



```
09/07/2020, 10:50:29 am  node: Example debug node
msg.payload : string[5]
"HELLO"
```

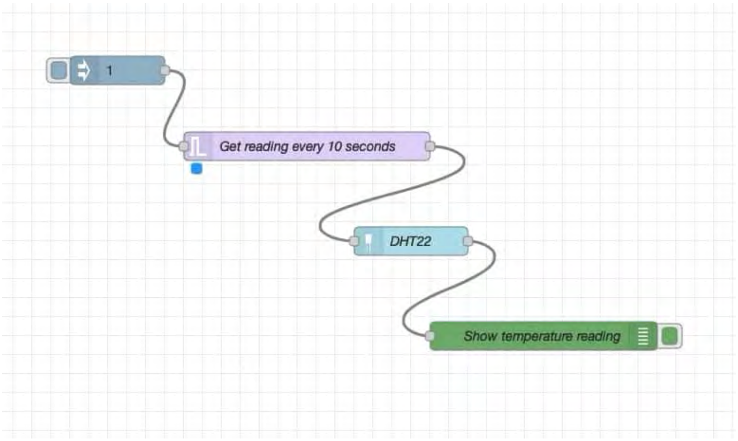
The outcome of this simple flow.

A flow with the DHT22

Let’s try another flow. In this flow, we’ll use the DHT22 node to get a temperature reading from the sensor. We’ll use a debug node to show the temperature.

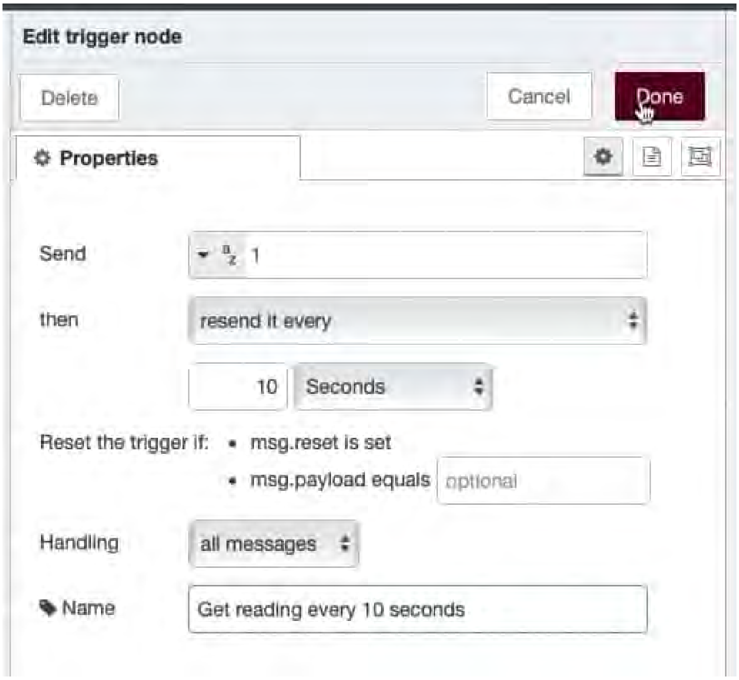
I’d like to get a new reading every 10 seconds, so I will add a trigger node to the flow. Finally, I’d like to trigger the flow manually, so I’ll use an inject node which has a very useful button.

Here’s the flow, assembled:



This flow will display the temperature in the debug window, every 10 seconds.

Below I provide the configuration of each node.



Send: 1

then: resend it every

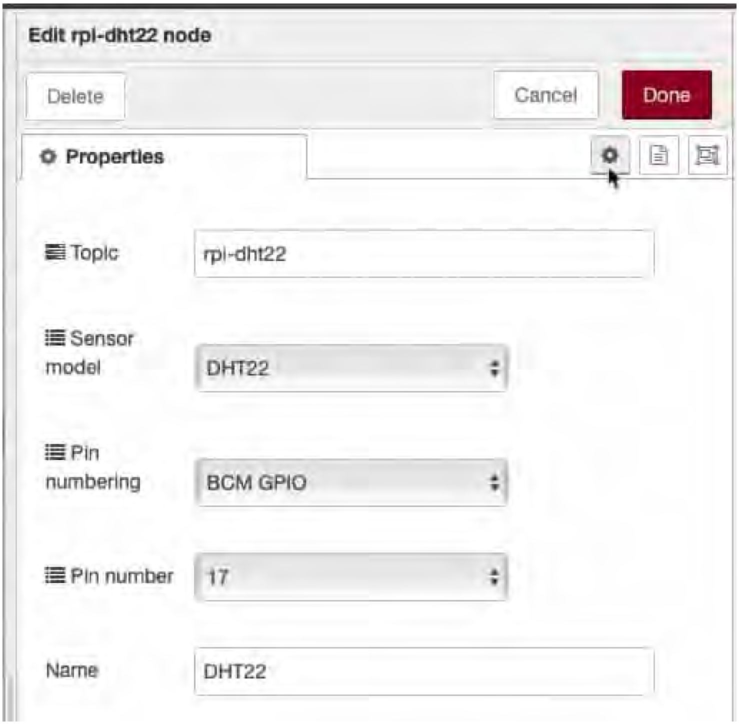
10 Seconds

Reset the trigger if:
- msg.reset is set
- msg.payload equals optional

Handling: all messages

Name: Get reading every 10 seconds

The trigger node.



Topic: rpi-dht22

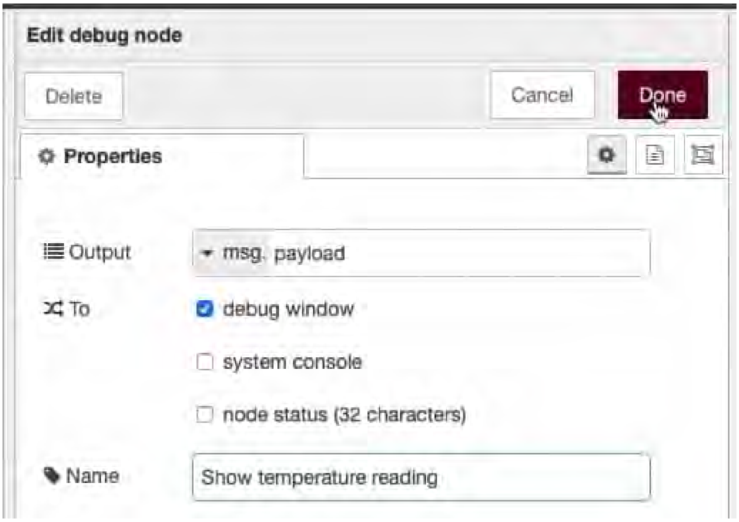
Sensor model: DHT22

Pin numbering: BCM GPIO

Pin number: 17

Name: DHT22

The DHT22 node.

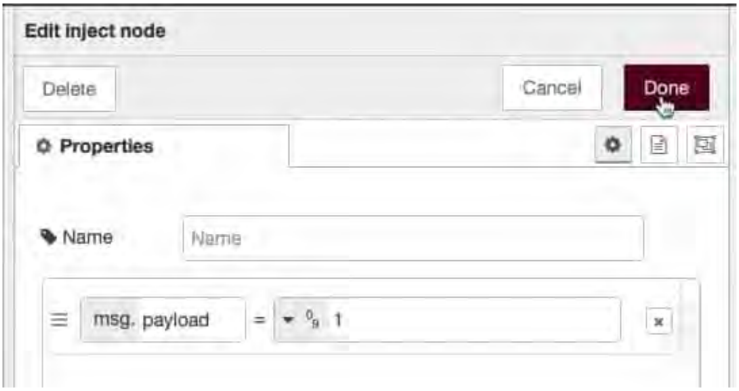

Edit debug node

Output: msg. payload

To: [✓ debug window] system console node status (32 characters)

Name: Show temperature reading

The debug node.

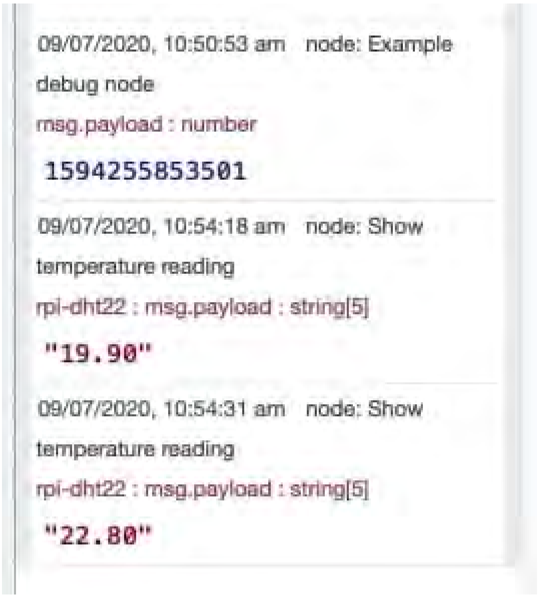

Edit inject node

Name: Name

msg. payload = 1

The inject node.

Once you assemble the flow, click on “Deploy” and then click on the inject button to start the trigger. After this, a new temperature value will appear in the debug window, every ten seconds.



```
09/07/2020, 10:50:53 am   node: Example
debug node
msg.payload : number
 1594255853501

09/07/2020, 10:54:18 am   node: Show
temperature reading
rpi-dht22 : msg.payload : string[5]
 "19.90"

09/07/2020, 10:54:31 am   node: Show
temperature reading
rpi-dht22 : msg.payload : string[5]
 "22.80"
```

The flow outcome.

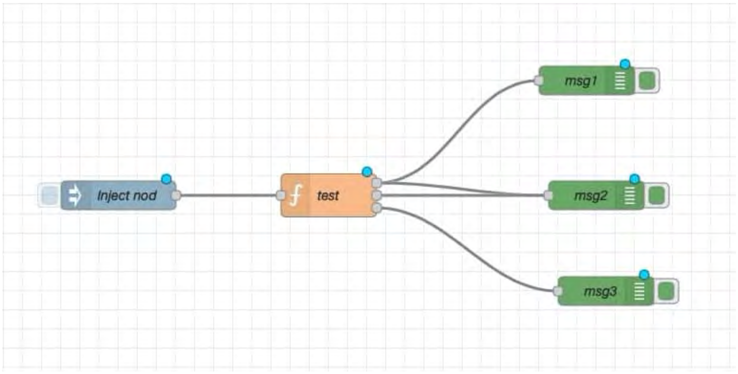
You can see the process of putting the flow together in detail in the video. This experiment begins at approximately 5 mins and 40 seconds in the video.

A flow with a function node

Let’s try an experiment that involves a flow with a function node. Function nodes are very useful and very flexible: you can write custom JavaScript that will run inside the node, and process inputs in any way you want.

The flow in this experiment will contain an inject node, a function node, and three debug nodes. It will demonstrate how you can do simple processing using JavaScript and drive three outputs.

The flow looks like this:



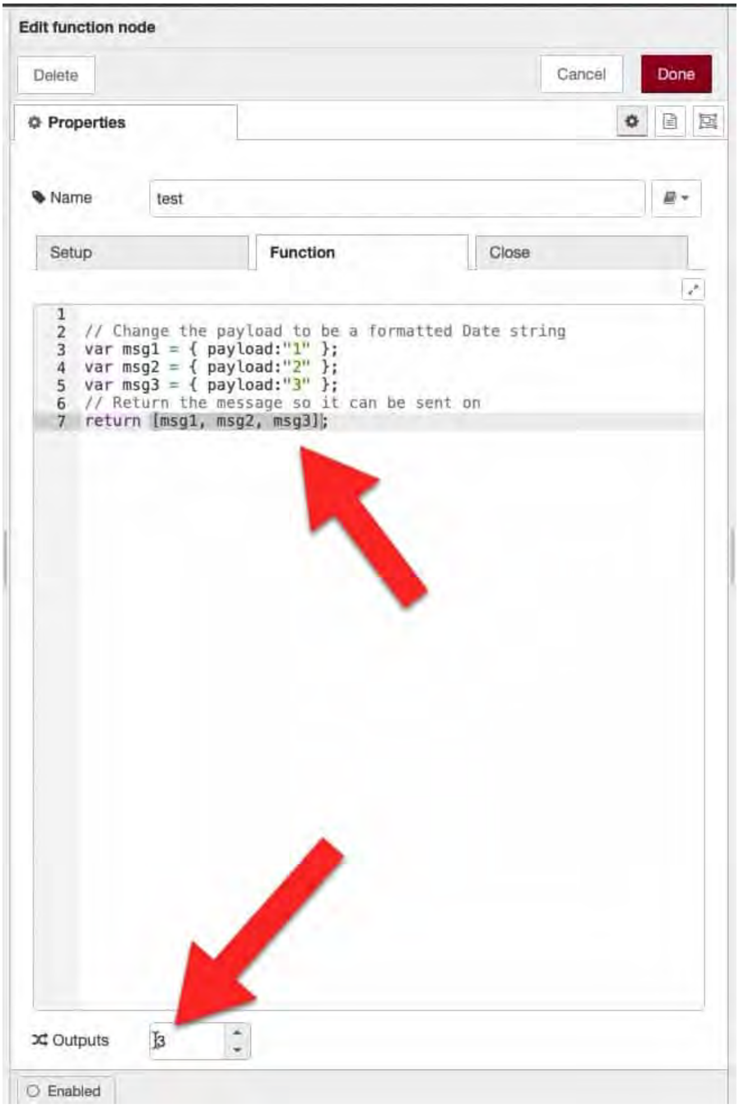
A simple flow that contains a function node.

The function node is in the middle of the flow. Notice that it has three outputs, and those are connected to the debug nodes. One debug node actually receives input from two function node outputs. This design is legitimate.

In the video in this article you can see, in detail, how I assemble this flow. The relevant part starts at around 14 minutes and 20 seconds in.

What is most interesting in this flow, is the JavaScript in the function node, and its configuration.

It looks like this:



Name: test

```
// Change the payload to be a formatted Date string
var msg1 = { payload:"1" };
var msg2 = { payload:"2" };
var msg3 = { payload:"3" };
// Return the message so it can be sent on
return [msg1, msg2, msg3];
```

Outputs: 3

This function node has three outputs.

The JavaScript is very simple. It contains three local variables, msg1, msg2 and msg3. Each contains a small JSON document, with a payload and a one-character string in it.

The return statement is an array with three fields.

At the bottom of the Edit pane, I have set the number of outputs to three, as many as there are fields in the return array.

Each field of the return array becomes an output.

The first field is outputted from the top output in the graphical symbol of the function node.

The second field is outputted from the middle output.

And the third field is outputted from the bottom output.

Isn’t this neat?

Functions, of course, are extremely powerful.

Anything you can do with JavaScript, any kind of program you can imagine, you can go in here to do processing for various inputs and then to create appropriate outputs for further down into the flow.

Let’s move on to the next article where you will learn about Node-RED variables.

Node-RED guide series

Node-RED Messages and Variables

Messages and variables are provide a mechanism by which nodes can pass data to each other and around the Node-RED flow.

In this article, I will explain the basics of messages and variables.

Documentation for messages and variables

If you have read the previous article in this series, you have already seen messages in action. There, I shows you a couple of examples the included the “msg” object.

In this article I will expand on messages and introduce Node-RED variables.

I also recommend that you have a look at the official Node-RED documentation, where messages and variables are covered in detail. You should also read the documentation page on context.

Messages vs variables

A confusion that arises among people that are new to Node-RED has to do with the differences and similarities between messages and variables.

Let’s clear this up right now.

Similarities

Both messages and variables are used to pass data from one node to another.

That’s it.

Differences

A message can only be passed from a node to the very next node to which it is connected.

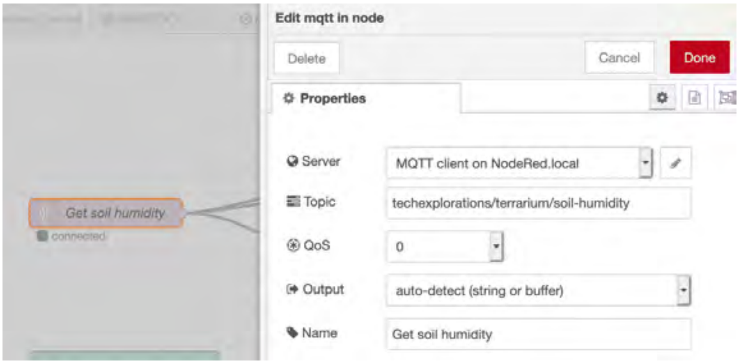
A variable can be passed between a node to any other node. Variables can be passed between nodes that are not directly connected. Variables can even be passed between nodes that belong to different flows.

Messages

The main thing to remember about messages, is that the data that you want to pass to the next node is contained within an object named “msg”. The msg object may contain as many attributes as you want. Some of those attributes are built-in, but you can also create your own.

The most important attribute is called “payload”.

Most nodes will automatically save data in the payload attribute. For example, consider the MQTT-in node:



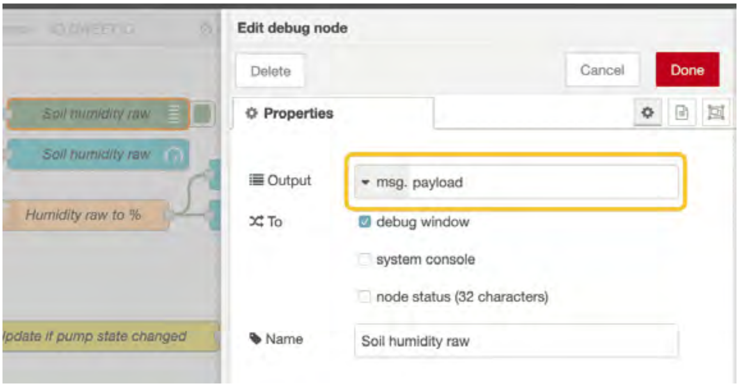
Server: MQTT client on NodeRed.local
Topic: techexplorations/terrarium/soil-humidity
QoS: 0
Output: auto-detect (string or buffer)
Name: Get soil humidity

The MQTT-in node will store data in the msg.payload attribute.

This node (like most other nodes), will store data in the msg.payload attribute. This is done by convention. You don’t have to do anything specific to make this happen.

Because of this, you know that you can get the data you need to process or display in the next node by reading the contents of the payload.

Here is an example of displaying the MQTT-in node data using a debug node:



Edit debug node

Output: msg. payload

To: [✓ debug window] system console, node status (32 characters)

Name: Soil humidity raw

Get the MQTT-in data from the payload attribute of the msg object.

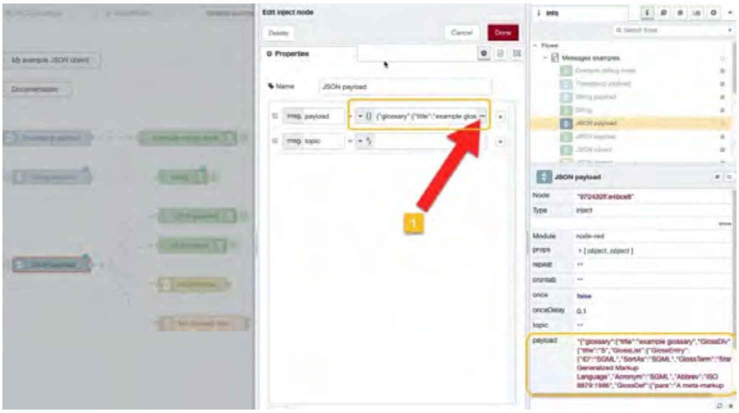
Notice that the output of the debug node comes from the msg object. In the text field, I have typed "payload" so that I can access the payload data.

In the payload, you can store these types of data:

- Boolean. Valid values: true, false
- Number, such as 0, 123.4
- String, such as "hello"
- Array, such as [1,2,3,4]
- JSON Object, such as { "a": 1, "b": 2}
- Null

JSON object payloads are very flexible. As long as the JSON expression is valid, it can be stored in the message payload.

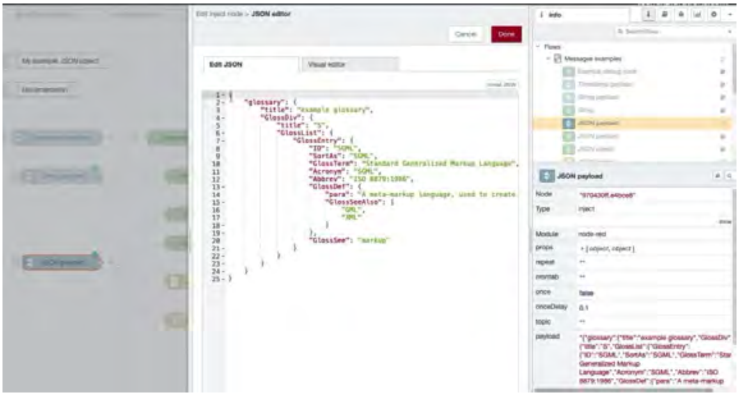
In the example below, you can see an inject node where I have set it's payload to contain a JSON document.



The payload of this inject node is a JSON document.

In the inject node, you can insert the JSON document in the payload by selecting the appropriate JSON data type, and then clicking on the button marked "…" (see arrow "1").

This will show the JSON editor:

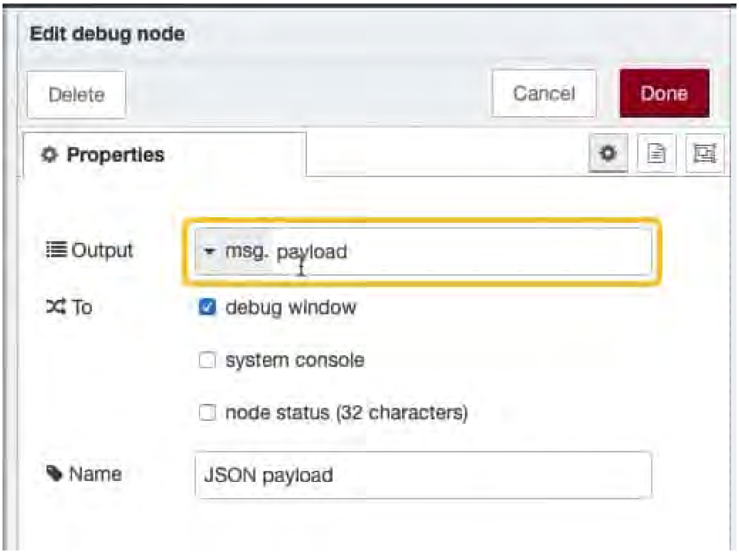


```
{
    "glossary": {
        "title": "example glossary",
        "GlossDiv": {
            "title": "S",
            "GlossList": {
                "GlossEntry": {
                    "ID": "SGML",
                    "SortAs": "SGML",
                    "GlossTerm": "Standard Generalized Markup Language",
                    "Acronym": "SGML",
                    "Abbrev": "ISO 8879:1986",
                    "GlossDef": {
                        "para": "A meta-markup language, used to create
                        "GlossSeeAlso": [
                            "GML",
                            "XML"
                        ]
                    },
                    "GlossSee": "markup"
                }
            }
        }
    }
}
```

The JSON editor.

The JSON editor makes it easy to create a JSON document. There is also a visual editor tab that allows you to browse through a JSON document.

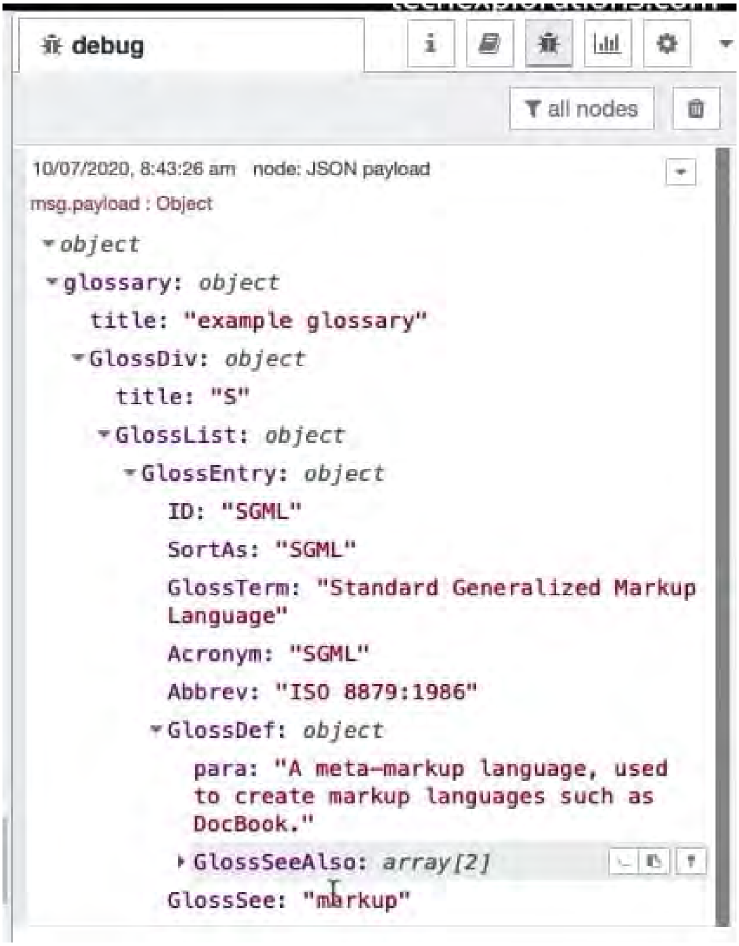
You can pass this JSON (or other datatype) payload to the next node. For example, if the next node is a debug node, it will be configured like this:



Output: msg. payload
To: [✓ debug window] system console; node status (32 characters)
Name: JSON payload

A debug node configured to receive a message payload.

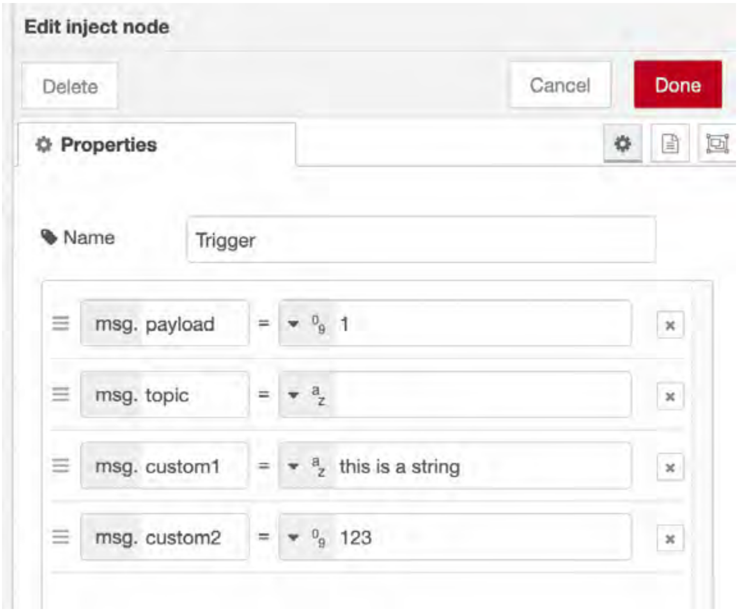
In the debug window, the JSON document will look like this:



```
10/07/2020, 8:43:26 am   node: JSON payload
msg.payload : Object
▾object
 ▾glossary: object
    title: "example glossary"
   ▾GlossDiv: object
      title: "S"
     ▾GlossList: object
       ▾GlossEntry: object
          ID: "SGML"
          SortAs: "SGML"
          GlossTerm: "Standard Generalized Markup
          Language"
          Acronym: "SGML"
          Abbrev: "ISO 8879:1986"
         ▾GlossDef: object
            para: "A meta-markup language, used
            to create markup languages such as
            DocBook."
           ▸GlossSeeAlso: array[2]
          GlossSee: "markup"
```

JSON output from a debug node.

Some nodes, such as the inject node, can also work with custom attributes attached to the msg object. Here is an example:

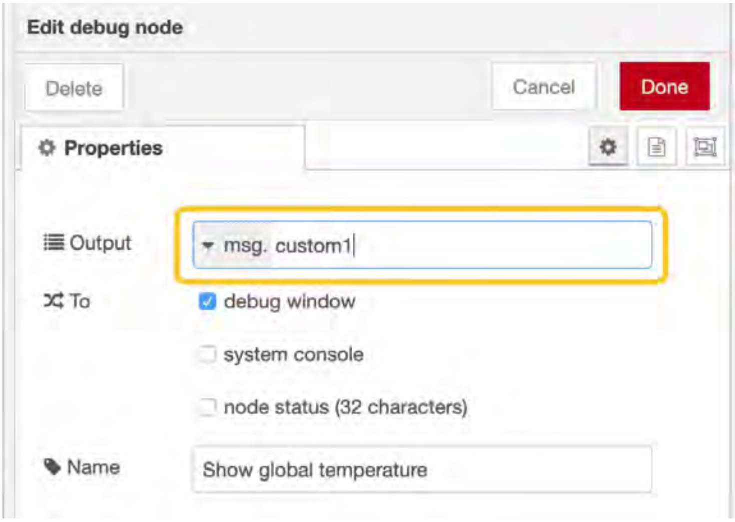


Name: Trigger

msg. payload = 1
msg. topic =
msg. custom1 = this is a string
msg. custom2 = 123

This inject node msg object contains custom attributes.

In this example, I have created two custom attributes, a number and a string. I can use these attributes in the next node by referencing them:


Edit debug node

Output: msg. custom1

To: [✓ debug window] system console, node status (32 characters)

Name: Show global temperature

I can access the data in the custom attributes by referencing the attribute name.

Variables

Next up: variables.

With variables you can share data across any nodes, and even flows.

Think of variables in Node-RED as variables in a programming language like Ruby and Python. In such languages we have global and local variables. Access to the variable depends on the variable's scope.

Similarly in Node-RED, we have the concept of scope, and as a result there are variables that allow access to their values depending on their scope.

You can learn more about context in Node-RED by reading this documentation page.

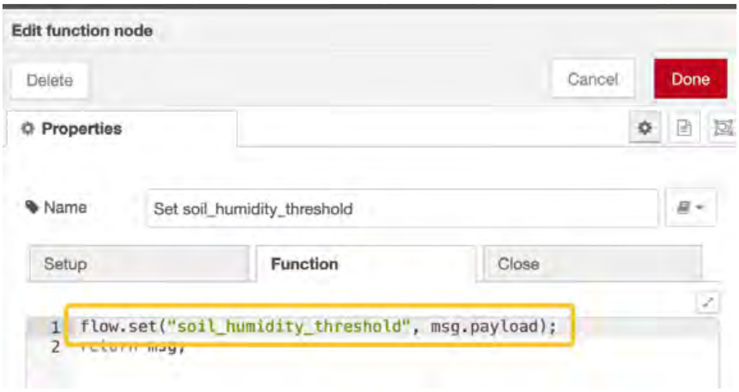
In Node-RED there are three scopes:

- Node (equivalent to a local variable in Python). It is only visible to the node that set the value.
- Flow (equivalent to a global variable in Python). It is only visible to all nodes on the same flow (or tab in the editor)
- Global (equivalent to a system variable in Python) – visible to all nodes

A global variable is can be set by a node in one flow, and be read by another node in any other flow. In a programming language like Python, this is roughly equivalent to the concept of cross-process (or inter-process) communication.

Most often you will be working within a flow context. In such case, you can set a flow variable using the "flow.set()" function.

Here is an example from the Terrarium project:



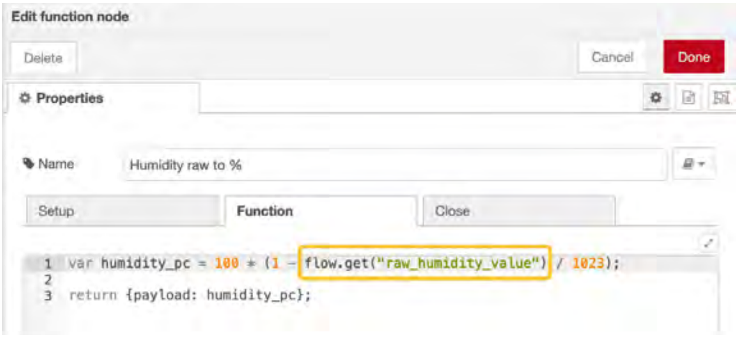
Name: Set soil_humidity_threshold

```
flow.set("soil_humidity_threshold", msg.payload);
return msg;
```

Set a flow variable.

In this example, I set a flow variable called “soil_humidity_threshold” to the value stored in msg.payload.

To read a flow variable, you can use the “flow.get()” function. Here is an example from another function node:



Name: Humidity raw to %

```
var humidity_pc = 100 * (1 - flow.get("raw_humidity_value") / 1023);

return {payload: humidity_pc};
```

Edit your caption text here

Here, I read the value stored in a flow variable called “raw_humidity_value”, and apply it to a calculation.

The video at the top of this page contains detailed examples of messages and variables.

At this point, you have a good basis for getting into hands-on examples of flows that contains the most common Node-RED nodes.

In the next article, you will dive into the debug node.

7. The "complete" node

Node-RED guide series

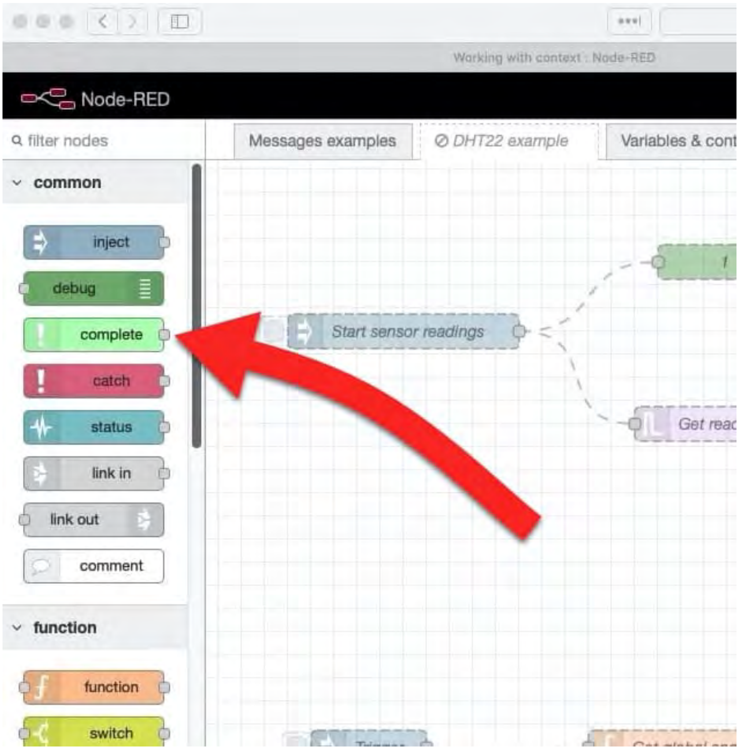
Node-RED, the “complete” node

Learn about the “complete” node and how to use in your flow to trigger an action

With the “complete” node, your can trigger a node in your flow when any other node (that supports this functionality) completes it’s operation.

In this article, I will demonstrate the use of the “complete” node through an example.

You can find the “complete” node under “common” in the left toolbar.

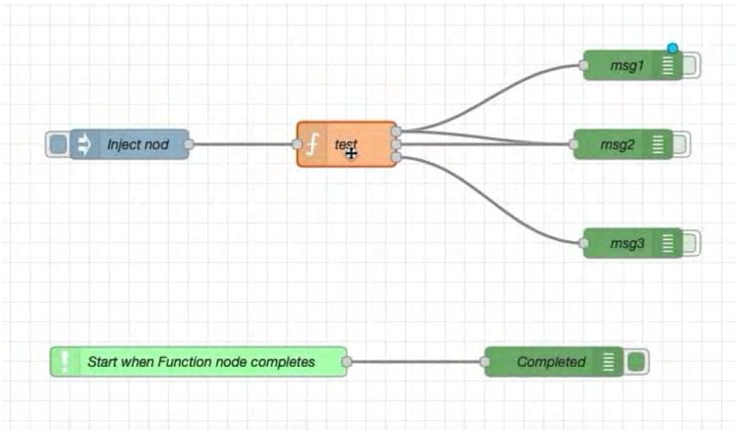


The “complete” node.

Setup the “complete” node

Let’s experiment with the “complete” node.

We’ll use the test flow from the previous article on “flows”.



The “Completed” debug node is triggered when “test” finishes.

In the original flow, I have added two nodes: a “complete” node that monitors the “test” node for completion, and a “debug” node.

The “complete” node is titled “Start when Function node completes”.

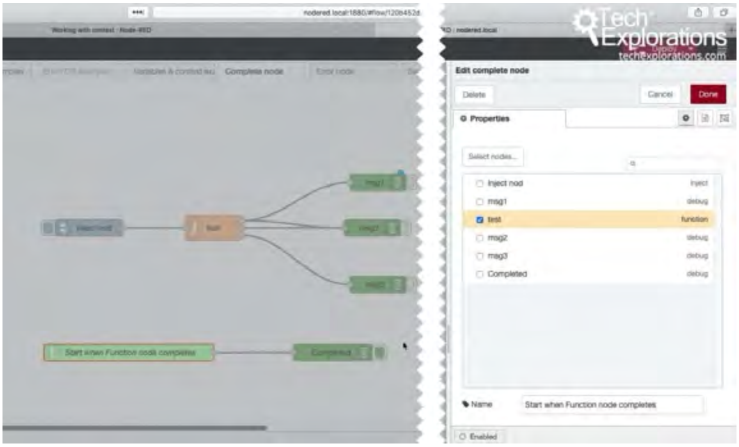
The “debug” node is titled “Completed”.

I have not made any changes to the original 5 nodes of the top row of the flow.

When the “test” function node completes its operation, the “Start when Function node completes” node is notified, and that triggers the “Completed” node.

In addition to the notification, the data from the node that has completed (“test” in this this case), is also passed along. So, the “Completed” node has access to the msg object of the “test” node.

In the screenshot below, you can see how I have configured the “complete” node titled “Start when Function node completes”:



The configuration of the “complete” node.

Double-click on the “complete” node to reveal its edit pane. In the Properties tab you will see a list of other nodes in the flow that are able to provide completion notifications.

Just click on one or more nodes that you want this “complete” node to receive completion notifications.

That’s it.

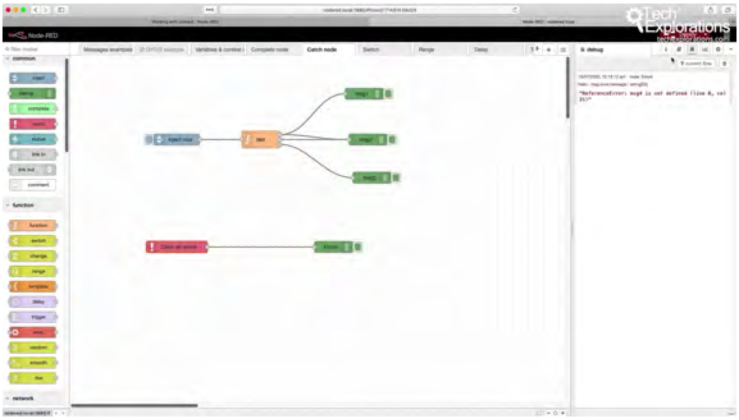
After you deploy the flow, every time that any of the selected nodes completes its operation, your “complete” node will be notified, and whichever node is wired at its output will be triggered. The msg object of the completed node is also passed to the triggered node.

8. The "catch" node

Node-RED guide series

Node-RED, the “catch” node

With the “catch” node, your flow can catch errors thrown by any node that belongs to the same flow.

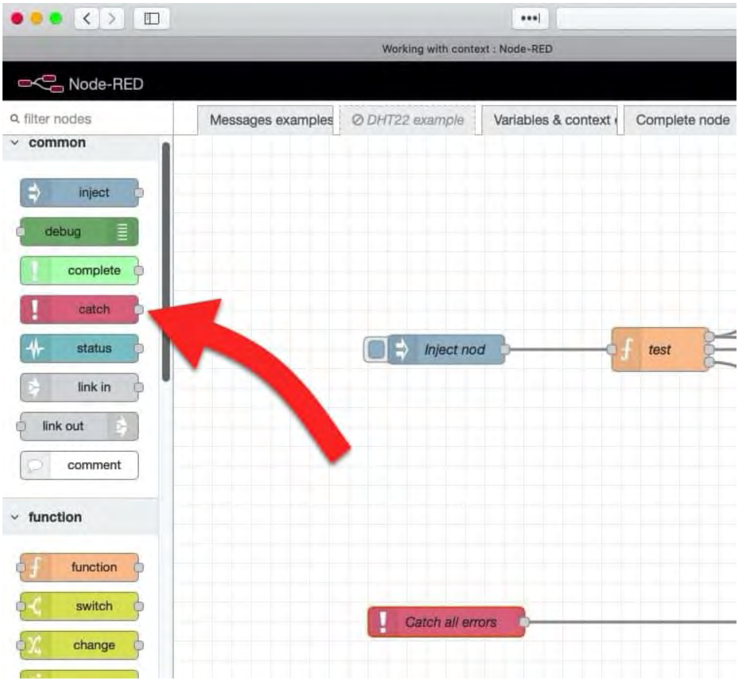


The “catch” node is used for catching exceptions in your flow. Once the “catch” node catches an exception, it can pass the relevant information in the msg object to another node, such as a “debug” node, which can handle it.

The “catch” node is similar to Python’s “try” statement or Ruby’s “throw” statements.

All these, “catch” in Node-RED, “try” in Python, and “throw” in Ruby have the same purpose: to make it possible for your program to detect an exception condition and handle it gracefully instead of simply “crashing”.

You can find the “catch” node under “common” in the left tool bar.

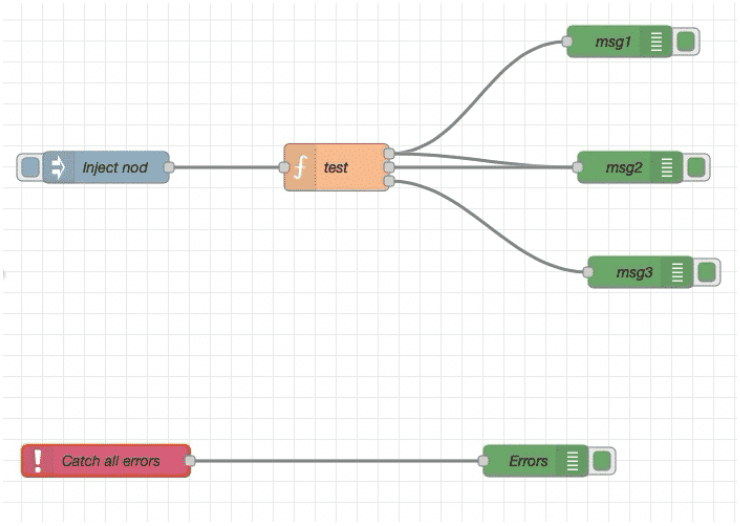


The “catch” node.

Setup the “catch” node

Let’s experiment with the “catch” node.

We’ll use the test flow from the previous article on “flows”.



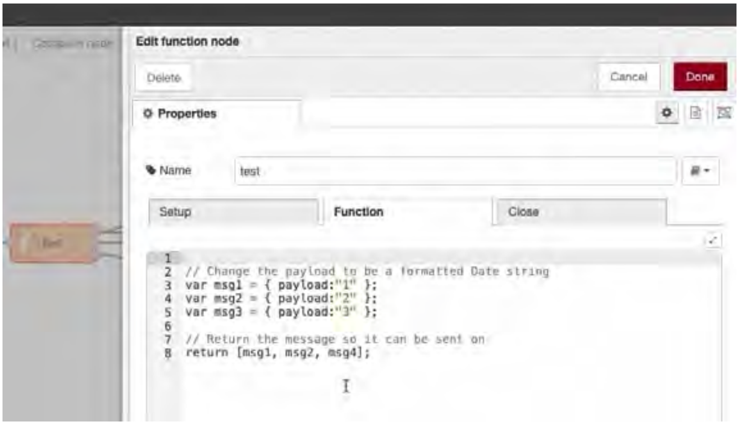
The “Catch” node can detect exceptions anywhere in the flow.

In the original flow, I have added two nodes: a “catch” node that monitors the flow for exceptions, and a “debug” node that will display information about an exception after it is caught.

The “catch” node is titled “Catch all errors”.

The “debug” node is titled “Errors”.

In the original flow “test” function, I have introduced an error. You should be able to see this error below:



Name: test

```

// Change the payload to be a formatted Date string
var msg1 = { payload:"1" };
var msg2 = { payload:"2" };
var msg3 = { payload:"3" };

// Return the message so it can be sent on
return [msg1, msg2, msg4];
```

Ooops… I've made a typo. Can you find it?

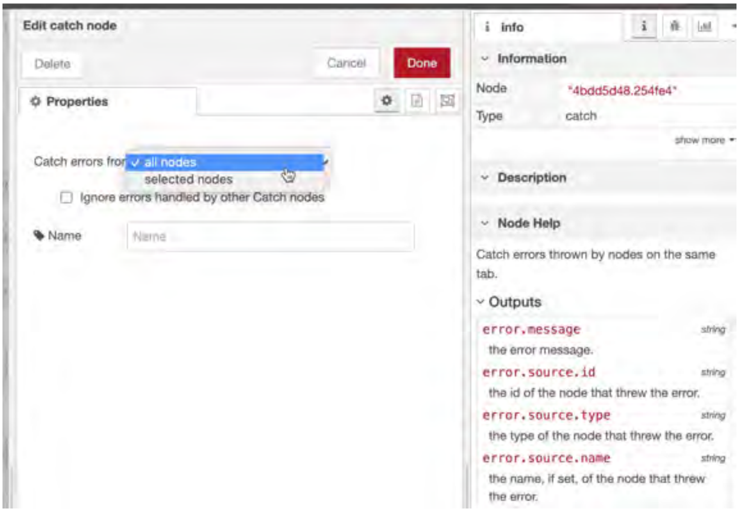
The "catch" node only has two configuration options:

- catch errors from all nodes
- catch errors from selected nodes

When an error is caught, the "catch" node will store relevant information inside the msg object in the form of attached attributes:

- error.message
- error.source.id
- error.source.type
- error.source.name

You can see these options in the node's edit pane, and in the node information (documentation). I have included both of these in the screenshot below:


Edit catch node

Delete | Cancel | Done

Properties

Catch errors from: ✓ all nodes / selected nodes

☐ Ignore errors handled by other Catch nodes

Name: Name

Information

Node "4bdd5d48.254fe4"

Type catch

Description

Node Help

Catch errors thrown by nodes on the same tab.

Outputs

`error.message` string — the error message.

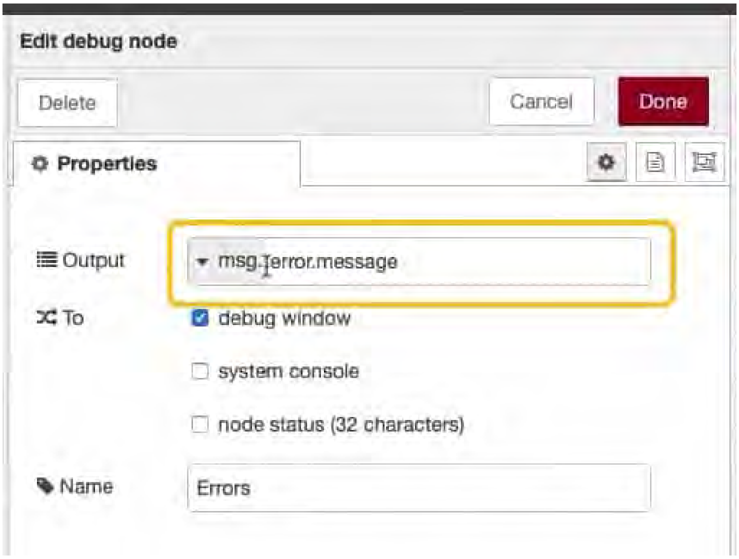
`error.source.id` string — the id of the node that threw the error.

`error.source.type` string — the type of the node that threw the error.

`error.source.name` string — the name, if set, of the node that threw the error.

“catch” node configuration and output.

To display information about the error that I planted in the function node, we can use a debug node, configured like this:


Edit debug node

Delete | Cancel | Done

Properties

Output: msg.error.message

To: ☑ debug window

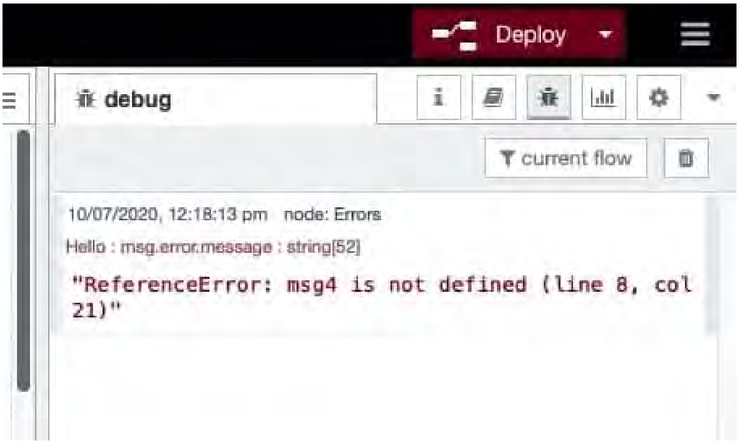
☐ system console

☐ node status (32 characters)

Name: Errors

Display the contents of msg.error.message

When I deploy the flow and click on the inject node button, the function node will through an exception which is caught by the “catch” node. The “catch” node will pass information about the exception to the “debug” node which will show the error in the console, like this:



```
10/07/2020, 12:18:13 pm   node: Errors
Hello : msg.error.message : string[52]
"ReferenceError: msg4 is not defined (line 8, col
21)"
```

Oops, there’s the bug.

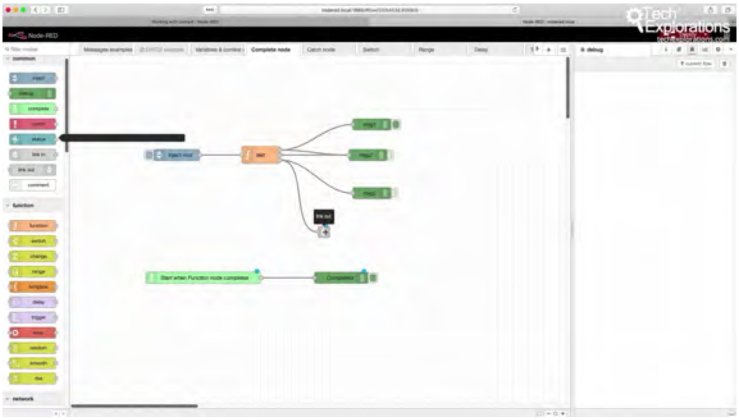
Now that I know there there is a bug, I can fix it and re-deploy the flow.

All good!

It’s good practice to keep a “catch” node in your flows just in case the JavaScript in the function nodes contain typos or errors that, otherwise, will not be reported.

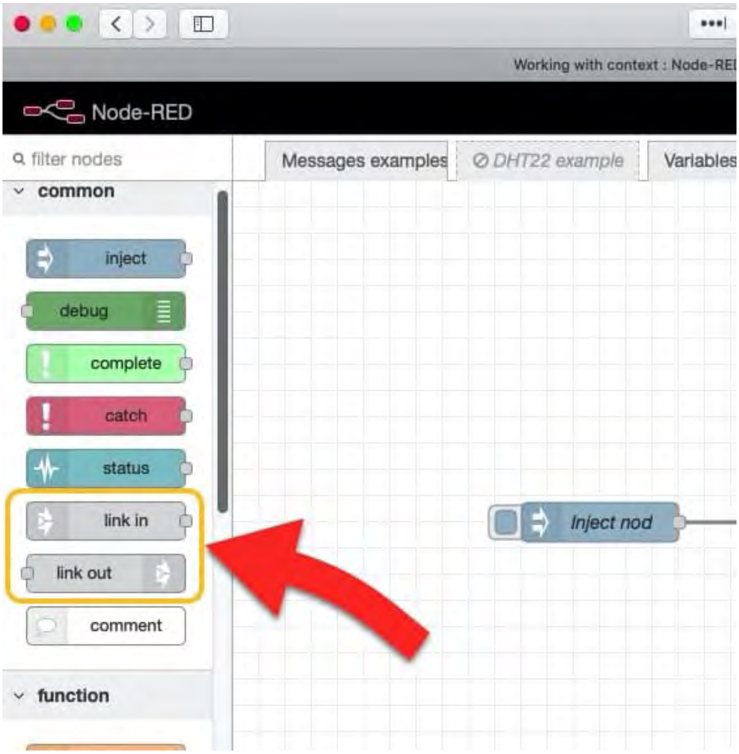
Node-RED, the “link out” and “link in” nodes

With the “link in” and “link out” nodes, you can connect and exchange data between nodes that belong to different flows.



You already know how to exchange data between nodes that belong to different flows using global variables. The “link out” and “link in” nodes provide you with an alternative method to do the same. These nodes are easy to set up, and especially useful if you simply want to share data from one node directly to another.

You will find the “link in” and “link out” nodes in the common group of the left toolbar.

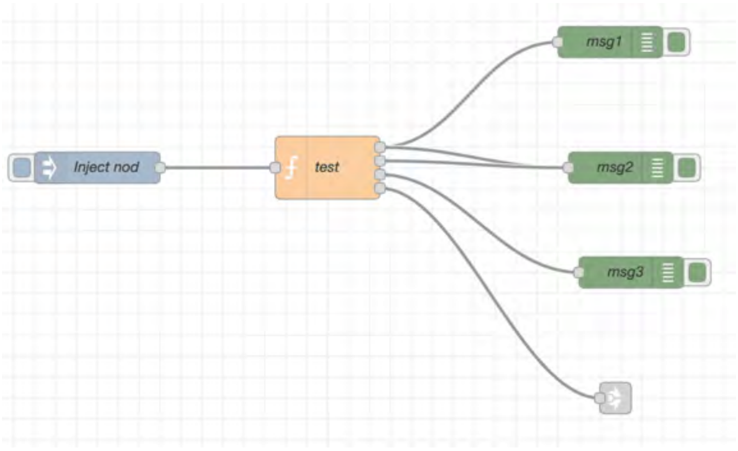


The “link out” and “link in” nodes.

Setup the “link out” and “link in” nodes

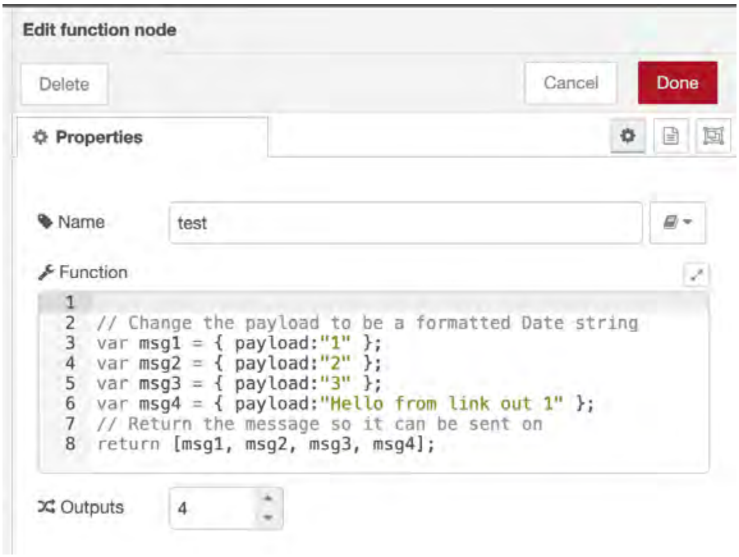
To demonstrate how “link out” and “link in” works, we’ll use the familiar flow from the previous few articles as our basis.

Here’s a slightly modified version of the flow. Notice that I have added a new output in the function node, which I have connected to a link “out node”.



This flow contains a “link out” node.

I have made a small change to the “function” node: I have added a fourth output and adjusted the JavaScript so that a string of text is send to the new output.



Name: test

Function

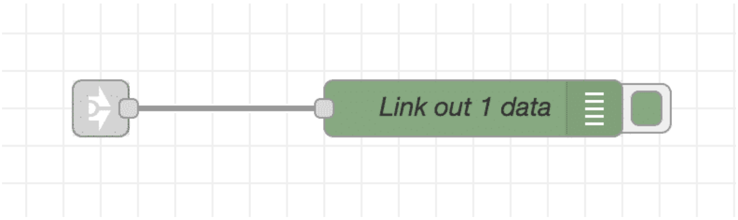
```

// Change the payload to be a formatted Date string
var msg1 = { payload:"1" };
var msg2 = { payload:"2" };
var msg3 = { payload:"3" };
var msg4 = { payload:"Hello from link out 1" };
// Return the message so it can be sent on
return [msg1, msg2, msg3, msg4];
```

Outputs: 4

The fourth output is connected to a “link out” node.

Don't worry about configuring the "link out" node yet; it's best to do this after you have added the "link in" node.

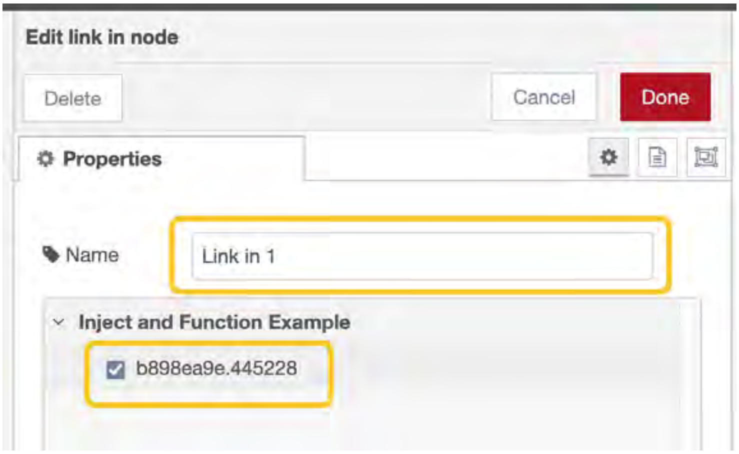
Create a new flow, and create a simple flow with a "link in" and "debug" nodes, like this:



This flow contains a "link in" node.

Double-click on the "link in" node to see it's properties.

In the "Name" field, type in "Link in 1" or something reasonable. This name will appear in the "link out" properties so that you can easily select the target of the "link out" node. The properties for the "link in" node look like this:



Name: Link in 1

Inject and Function Example

☑ b898ea9e.445228

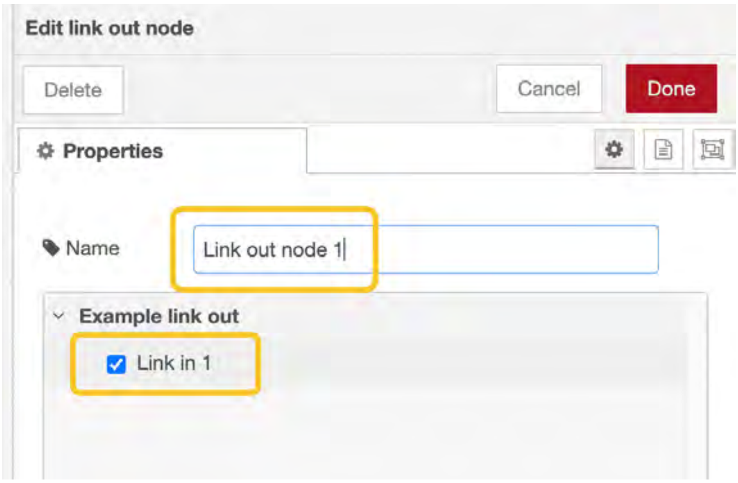
The "link in" node properties.

Notice that the "link out" node is listed under the name field.

Because our “link out” node does not have a name yet, Node-RED is showing the node ID. When you assign a name to the “link out” node, you will see the name in this list, instead of the node ID.

Continue with the flow that contains the “link out” node.

Double-click on the “link out” node to see the properties pane, and set its name to “Link out node 1” or anything else you want. The “link out” properties will look like this:



Name: Link out node 1

Example link out

☑ Link in 1

The “link out” node properties.

You can see that the “link in 1” node is listed as a candidate node to which you can link this node. Any other “link in” nodes that you may have created across all flows will be listed here.

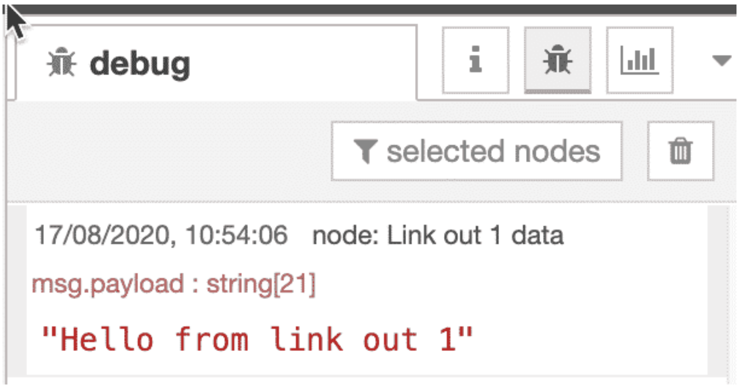
Select the “link in” node, and click on Done.

Before you deploy the changes, check on the name of the “link out” node as it appears in the “link in” node. Instead of the node ID, you now see the actual node name.

You can link “link out” and “link in” node from wither one.

Deploy the changes and then click on the "inject" node's button.

You will see this message in the debug tab:



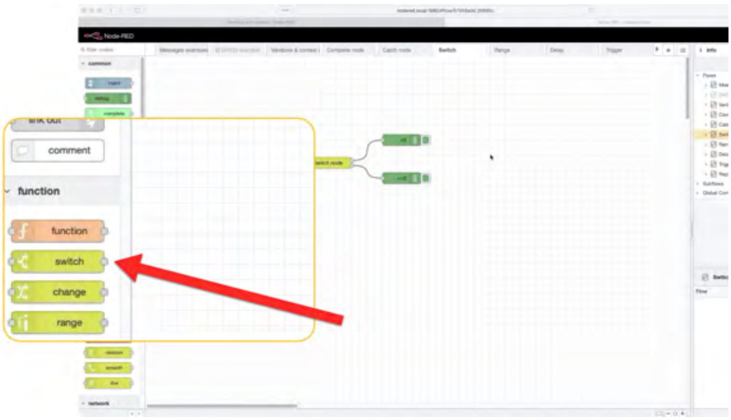
```
17/08/2020, 10:54:06   node: Link out 1 data
msg.payload : string[21]
"Hello from link out 1"
```

The output from "link in".

Link-in and link-out provide an alternative way for data to be shared among nodes that belong to different flows.

Node-RED, the “switch” node

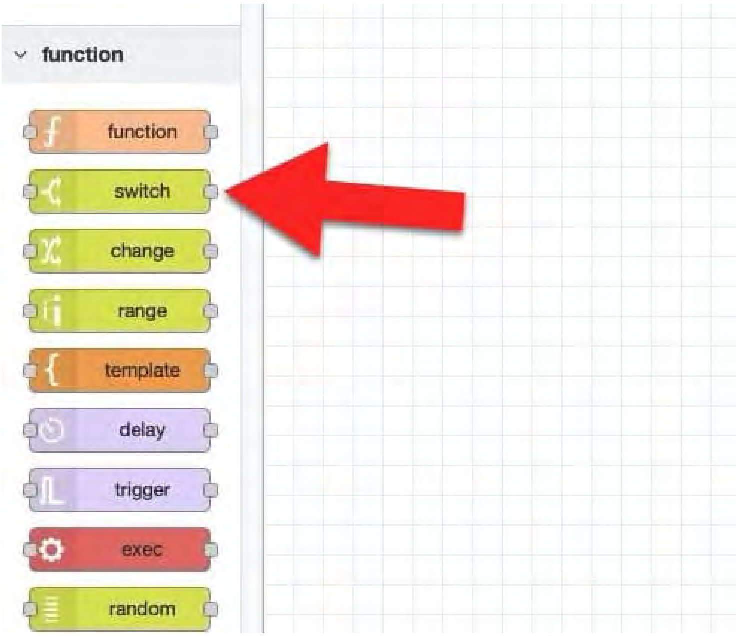
The “switch” node allows your flow to execute one of several possible paths depending on the rules you have specified.



The “switch” node is like the “if” statement in a text programming language.

Like the “if” statement, you can use the “switch” node to define one or more logical rules. According to the truthfulness (or falseness”) of those rules, the “switch” node can be configured to trigger one or more paths of execution.

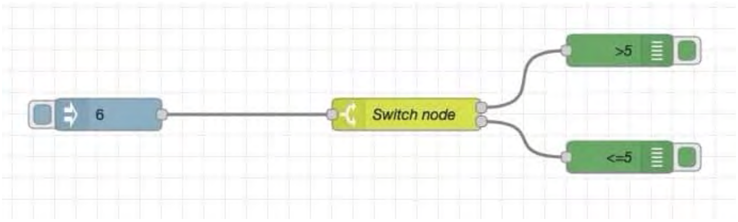
You can find the “switch” node in the “function” group of the left tool bar.



The “switch” node.

Setup the “switch” node

To explain how to use the “switch” node, I have create this simple flow:

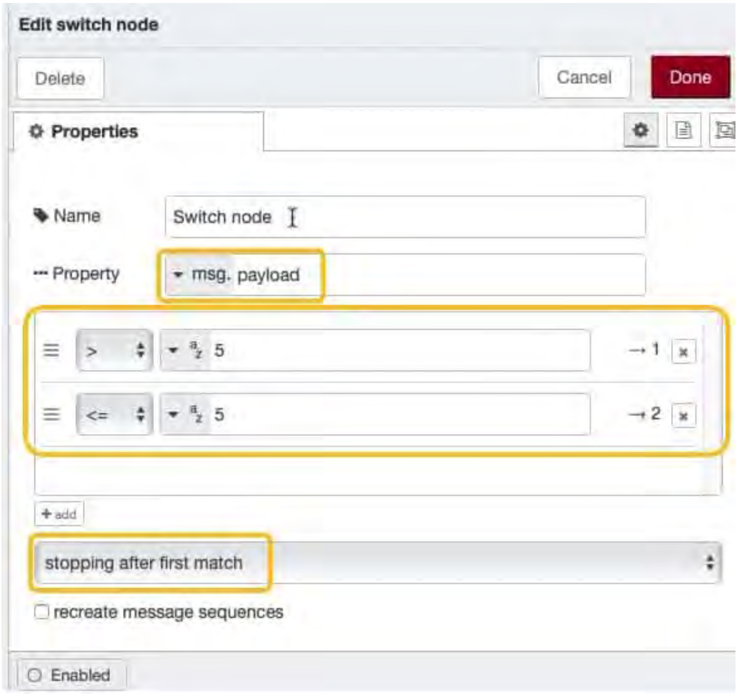


This flow contains a “switch” node.

The flow begins with an “inject” node which will emit a numerical “6” when you click on the button.

The payload will go into the switch node, which will evaluate it. If the payload is larger than 5, then the top output will be triggered. If the payload is equal or less than 5, then the bottom output will be triggered.

Take a look inside the “switch” node:


Edit switch node

Delete | Cancel | Done

Properties

Name: Switch node

Property: msg. payload

```
> 5   → 1
<= 5  → 2
```

stopping after first match

recreate message sequences

Enabled

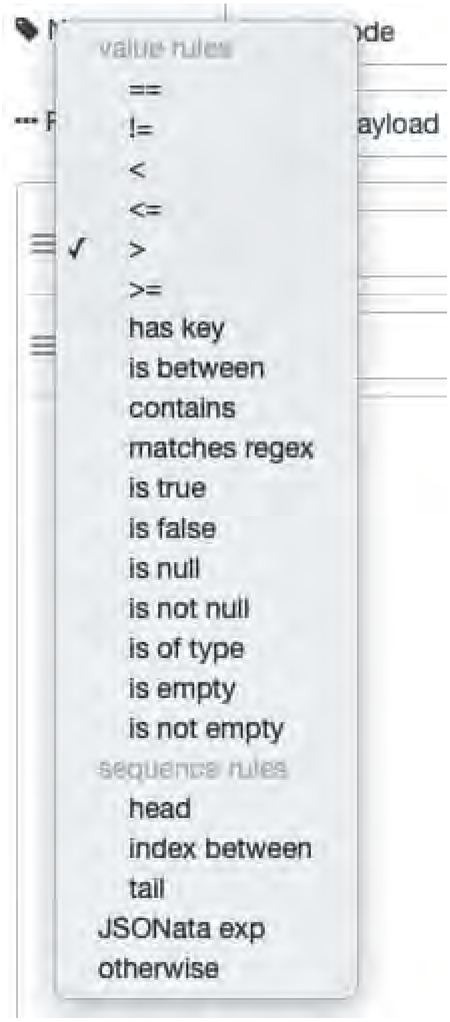
The properties of the “switch” node.

I have marked the important fields of the edit pane with a box.

First, you must set the property to evaluate. In this case, I have configure the node to get the value from the message payload.

Next, you must define at least one rule to be evaluated. The rule is defined by a boolean comparator, a value, and the output that will be triggered if the rule is evaluated to be true.

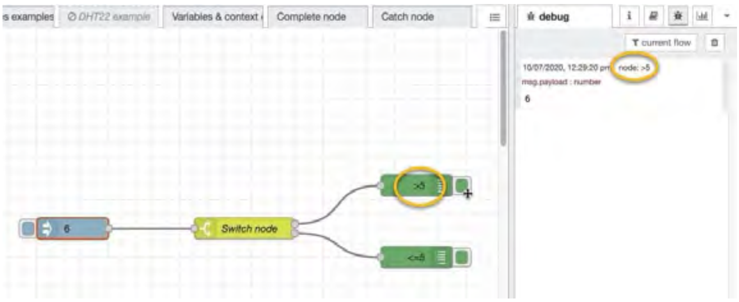
The available comparators are comprehensive:



The comparators available in the “switch” node.

The last field in the edit menu allows you to set the evaluation mode. You can set the node to stop when it finds a true rule, or to evaluate all rules.

Go ahead and try out this example. Deploy the flow, and then click on the “inject” node’s button. The output will be “6” from node “>5”, as expected:



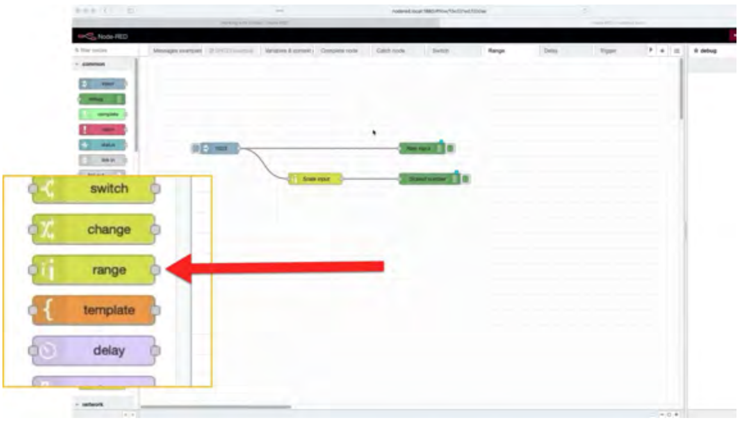
```
10/07/2020, 12:29:20 pm  node: >5
msg.payload : number
6
```

The output shows that the first rule of the “switch” node was true.

The “switch” node will evaluate numerical “6” and continue flow execution from the top output.

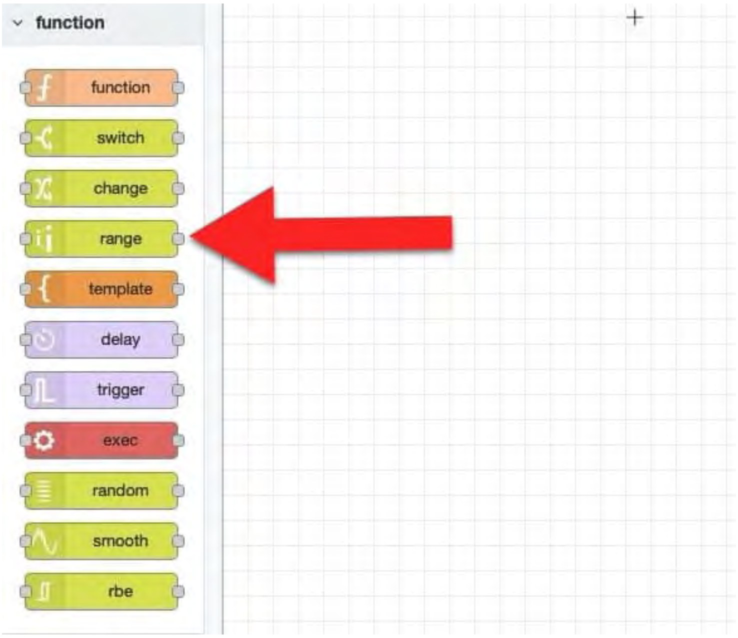
Node-RED, the “range” node

The “range” node takes in a number, which belongs to a particular scale range, and converts it to a new number that belongs to a different range.



The range node works like the map function in the Arduino programming language. It takes a number, which belongs to a particular range, and it will re-map it into a new number in a new range.

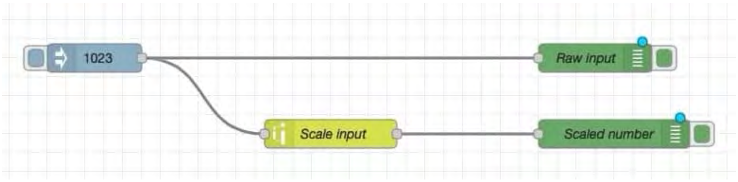
You will find the “range” node in the “function” group of the left toolbar.



The “range” node.

Setup the “range” node

To explain how to use the “range” node, I have create this simple flow:



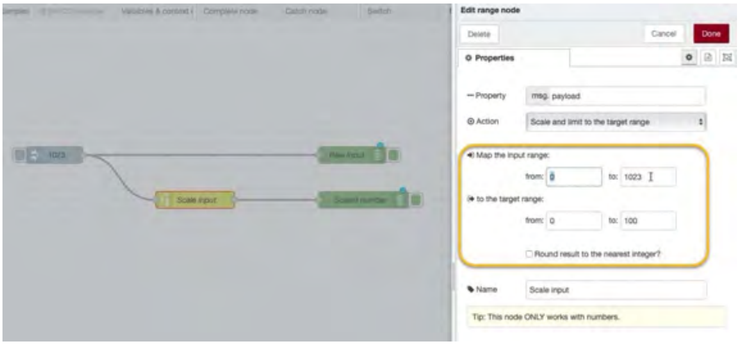
This flow contains a “range” node.

I am using an “inject” node to send a number to the “range” node titled “scale input”.

The “range” node takes the number from its input and re-maps it.

Below you can see how I have configured the “range” node.

The input range is 0 to 1023, and the output range is 0 to 100:

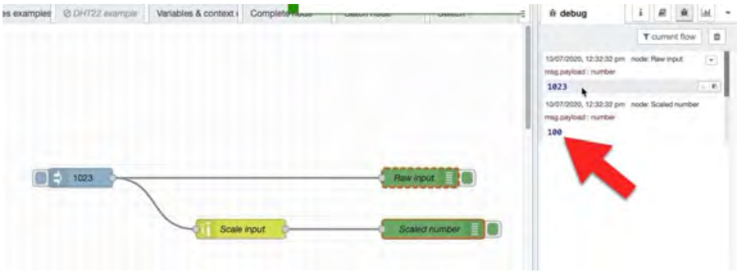


Property: msg. payload
Action: Scale and limit to the target range
Map the input range: from: 0 to: 1023
to the target range: from: 0 to: 100
☐ Round result to the nearest integer?
Name: Scale input
Tip: This node ONLY works with numbers.

The edit properties of the “range” node.

Let’s try out a couple of input values. There’s already “1023” set in the “inject” node. Deploy the flow, and click on the “inject” node button.

Look at the output in the debug pane. It should look like this:



```
10/07/2020, 12:32:32 pm  node: Raw input
msg.payload : number
1023
10/07/2020, 12:32:32 pm  node: Scaled number
msg.payload : number
100
```

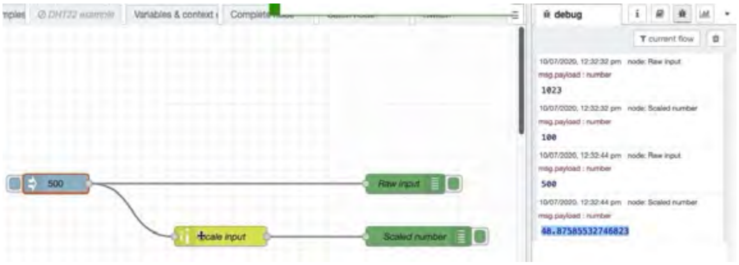
The arrow shows the scaled value of the input number. 1023 becomes 100.

The input “1023” belongs to the range 0 to 1023. Remapping this to the range of 0 to 100 yields “100”.

Let’s try another input number: “500”.

Edit the “inject” node, redeploy and click on the button to start the flow.

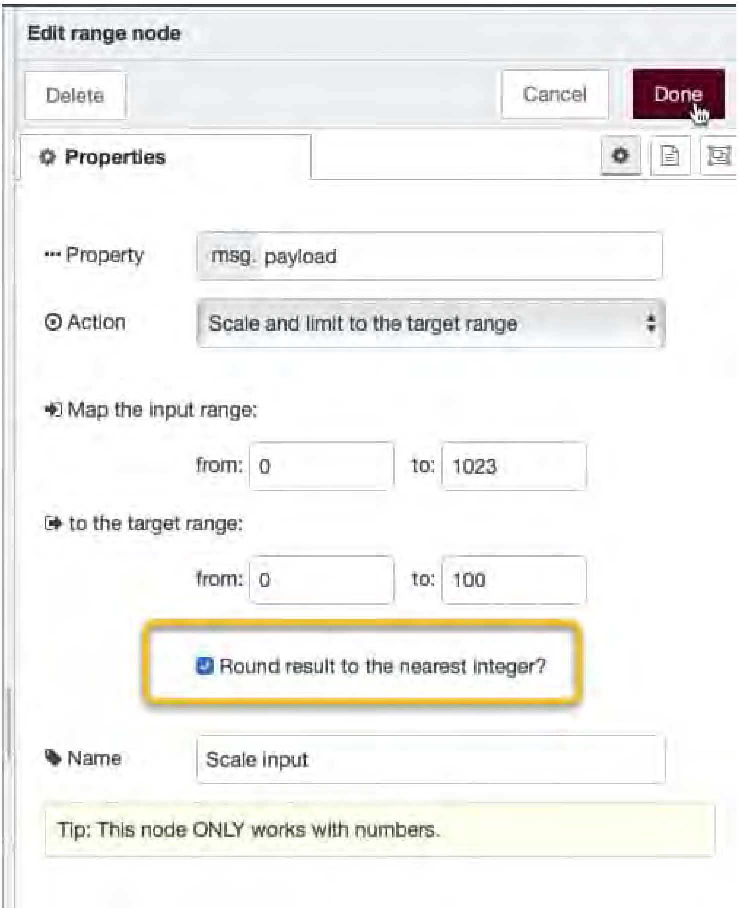
The result is below:



```
10/07/2020, 12:32:32 pm   node: Raw input
msg.payload : number
1023
10/07/2020, 12:32:32 pm   node: Scaled number
msg.payload : number
100
10/07/2020, 12:32:44 pm   node: Raw input
msg.payload : number
500
10/07/2020, 12:32:44 pm   node: Scaled number
msg.payload : number
48.87585532746823
```

500 out of 1023 is remapped to 48.87 out of 100.

The remapped value of 500 out of 1023 is 48.875855… out of 100. If you don’t need all these decimals, you can round the result to the nearest integer by selecting the appropriate checkbox in the “range” node’s properties:


Edit range node

Property: msg. payload

Action: Scale and limit to the target range

Map the input range:

from: 0 to: 1023

to the target range:

from: 0 to: 100

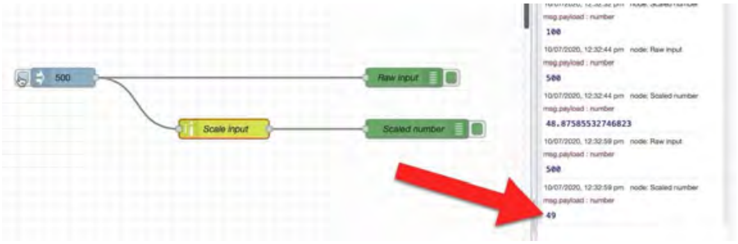
☑ Round result to the nearest integer?

Name: Scale input

Tip: This node ONLY works with numbers.

Scale the mapped value to the nearest integer.

Re-deploy and click on the inject button. The result is easier to read:



```
msg.payload : number
100
10/07/2020, 12:32:44 pm   node: Raw input
msg.payload : number
500
10/07/2020, 12:32:44 pm   node: Scaled number
msg.payload : number
48.87585532746823
10/07/2020, 12:32:59 pm   node: Raw input
msg.payload : number
500
10/07/2020, 12:32:59 pm   node: Scaled number
msg.payload : number
49
```

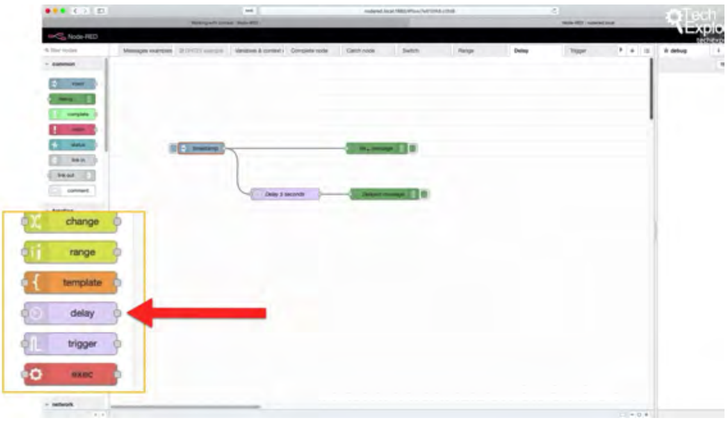
The mapped number is rounded to the nearest integer.

The “range” node is very useful, especially when you work with numbers exist within a specific range. For example, I’ll be using this node in the terrarium project to scale the analog input that comes from the ESP32 for the humidity of the soil into a number from zero to 100 to represent humidity as a percentage.

Node-RED guide series

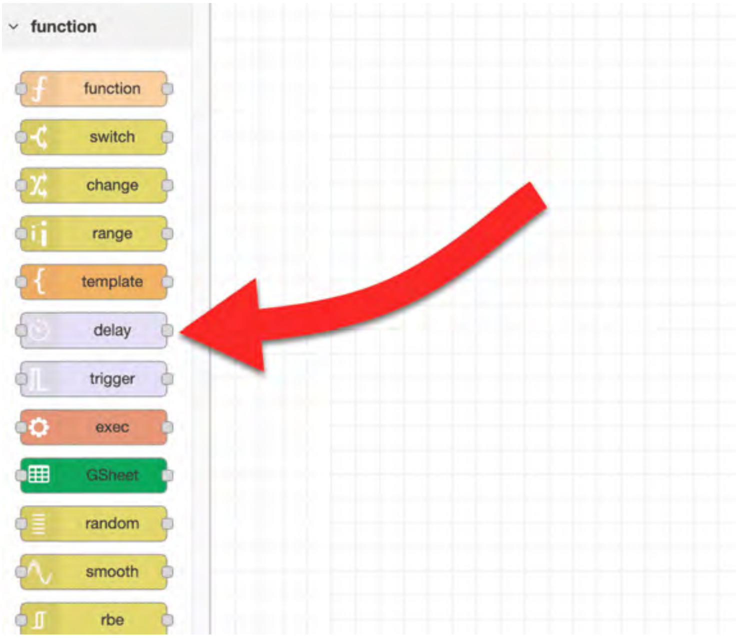
Node-RED, the “delay” node

The “delay” node does two things: (1) It allows you to delay a message by an arbitrary amount of time and (2) to limit the rate of messages that are passing through it.



In this article I will show you how to use the “delay” node to introduce a delay in the propagation of a message through a flow, and to limit the rate of messages coming through.

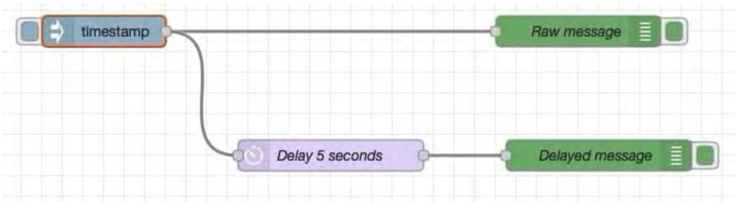
The rate-limiting function is very useful, for example, when you want to connect your flow to an external resource on the cloud. Typically, IoT resources impose a limit to how often you can “hit” them. If your application exceeds this limit, your account can be suspended, or at least made in-operable for an amount of time. With the “delay” node, you can ensure that your flow does not “hit” the IoT resource beyond a specific rate.



The “delay” node.

Setup the “delay” node

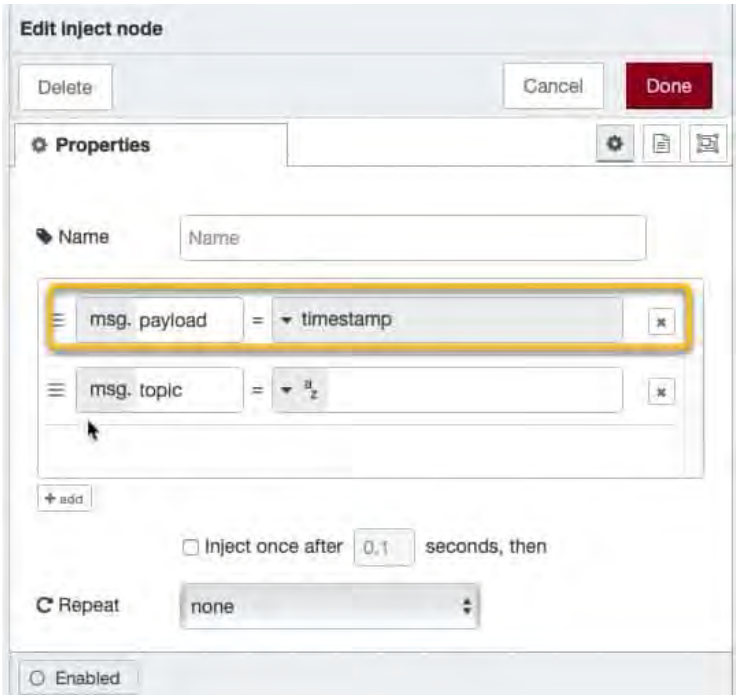
To explain how to use the “delay” node, I have create this simple flow:



This flow contains a “delay” node.

I am using an “inject” node to send a timestamp to two “debug” nodes.

I’ll be doing two experiments with the delay node. For both, the configuration of the inject node is the same, and it looks like this:


Edit inject node

Delete | Cancel | Done

Properties

Name: Name

msg. payload = timestamp

msg. topic = ᵃz

+ add

☐ Inject once after 0.1 seconds, then

Repeat: none

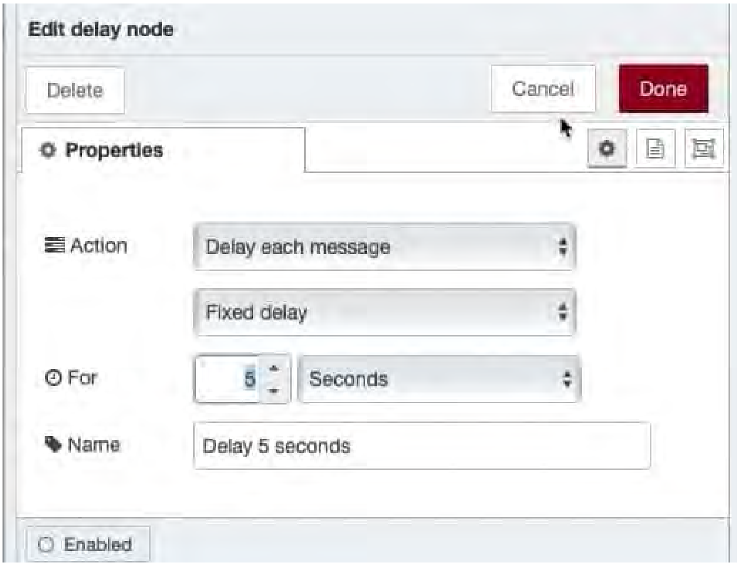
Enabled

The configuration of the “inject” node.

Experiment one: simple delay

In the first experiment, I want to use the “delay” node to hold the incoming message for 5 seconds, and then pass it on to the “debug” node.

To achieve that, I have set the “delay” node like this:



Action: Delay each message
Fixed delay
For: 5 Seconds
Name: Delay 5 seconds

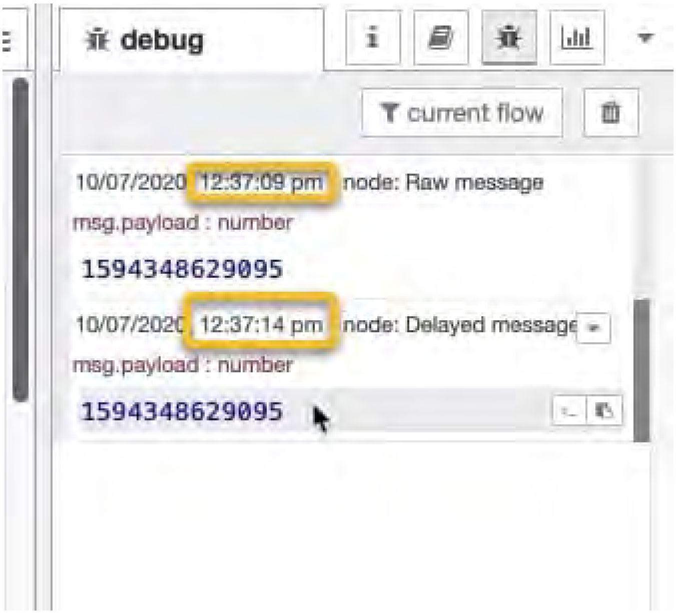
The configuration of the “delay” node.

Deploy the flow, and click on the inject node button.

The first “debug” node will display the timestamp immediately.

However the second “debug” node is fed by the delay node, which holds the message for 5 seconds before it lets it through.

The second “debug” node will display the message exactly 5 seconds after the first one, as you can see from the message timestamps, marked by the yellow boxes:



```
10/07/2020 12:37:09 pm node: Raw message
msg.payload : number
1594348629095
10/07/2020 12:37:14 pm node: Delayed message
msg.payload : number
1594348629095
```

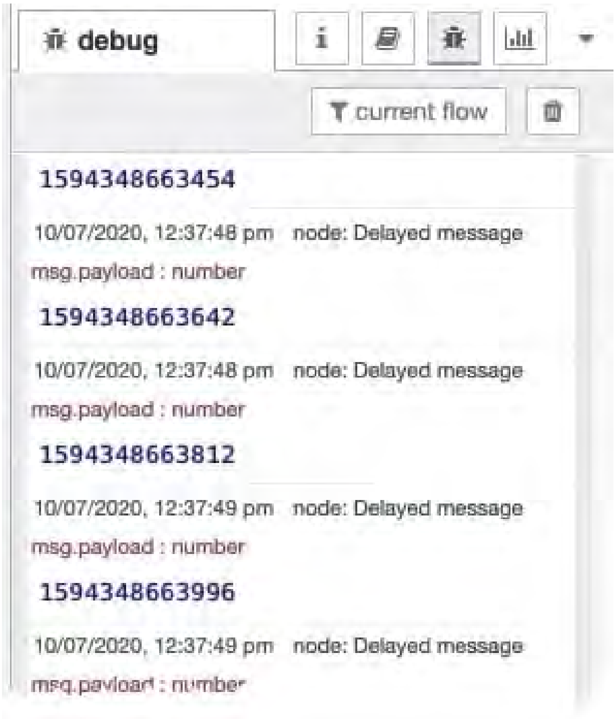
The two messages are exactly 5 seconds apart, even though they were created at the same time

Experiment two: rate limiter

In the second experiment, I want to use the “delay” node as a rate limiter.

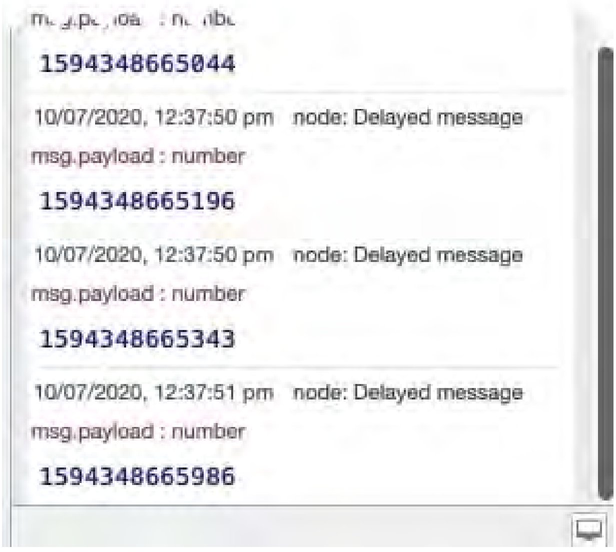
The way that the flow works now, if you click on the “inject” button repeatedly and as fast as you can, you will see a timestamp for each click.

Very quickly, the debug pane will fill up with timestamps:


debug

current flow

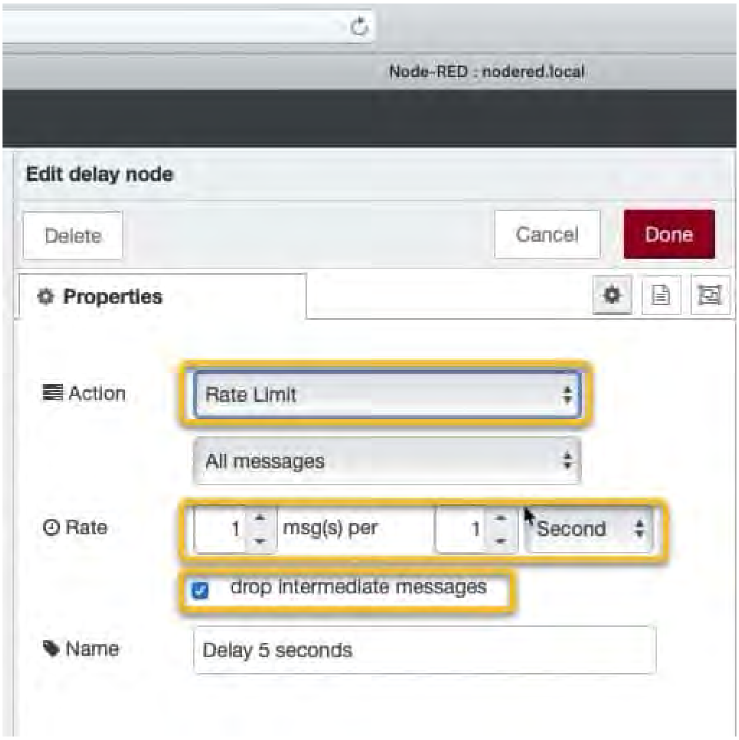
```
1594348663454

10/07/2020, 12:37:48 pm   node: Delayed message
msg.payload : number
1594348663642

10/07/2020, 12:37:48 pm   node: Delayed message
msg.payload : number
1594348663812

10/07/2020, 12:37:49 pm   node: Delayed message
msg.payload : number
1594348663996

10/07/2020, 12:37:49 pm   node: Delayed message
msg.payload : number
```



```
msg.payload : number
1594348665044

10/07/2020, 12:37:50 pm   node: Delayed message
msg.payload : number
1594348665196

10/07/2020, 12:37:50 pm   node: Delayed message
msg.payload : number
1594348665343

10/07/2020, 12:37:51 pm   node: Delayed message
msg.payload : number
1594348665986
```

The debug pane is filled with timestamps.

To reduce the number of messages that get through to the “debug” node to a small number, say one message per second, I can configure the delay node like this:


Node-RED : nodered.local

Edit delay node

Action: Rate Limit
All messages
Rate: 1 msg(s) per 1 Second
☑ drop intermediate messages
Name: Delay 5 seconds

The rate is limited to one message per 1 second.

Now, no matter how fast I can click on the “inject” node button, only one message per second will appear in the debug pane. All intermediate timestamps will be dropped.

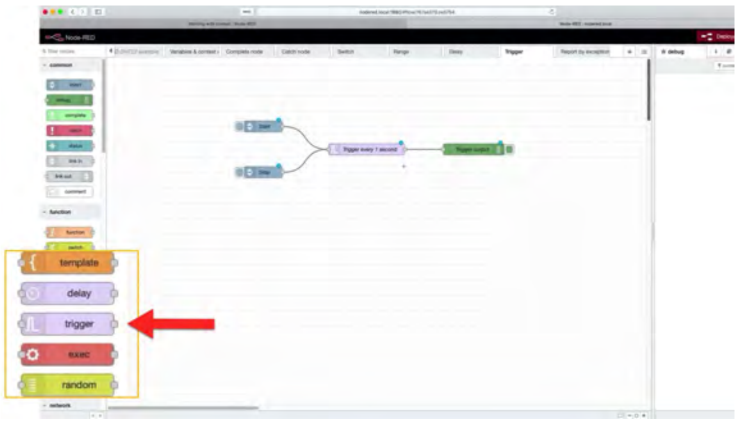
The rate limiting capability of the “delay” node is something use in the terrarium controller project. For example, I have implemented a feature so that the gadget can notify me when the MCU or pump voltage drops below a threshold. I don’t want to flood IFTTT (or my inbox) with such messages, so I have used the rate limiter to only send me one message every ten minutes.

13. The trigger node

Node-RED guide series

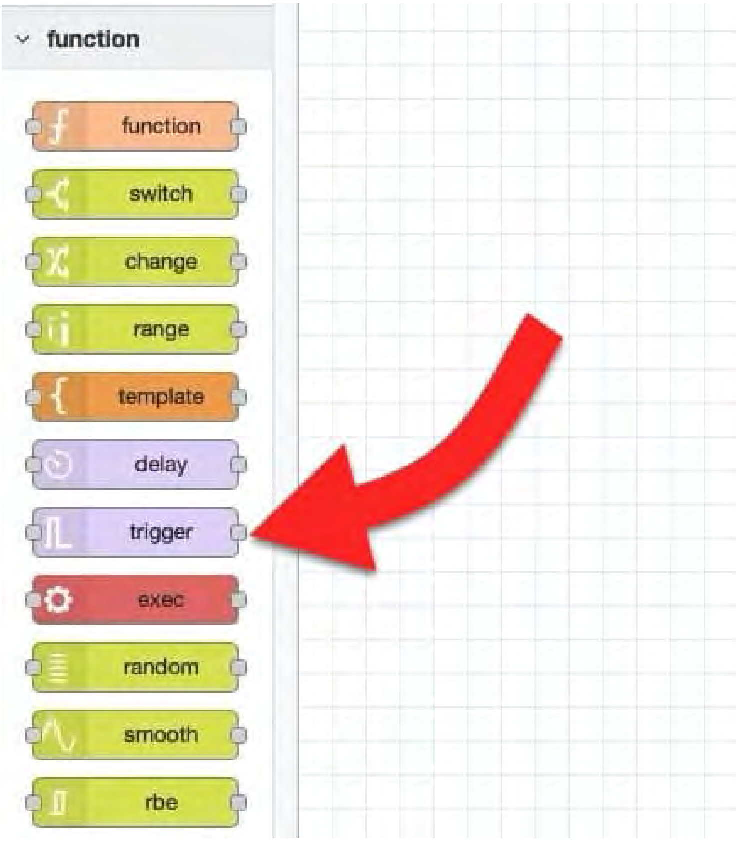
Node-RED, the “trigger” node

With the “trigger” node, you can repeat a message at an arbitrary period.



With the “trigger” node, you can repeat a message at an arbitrary period.

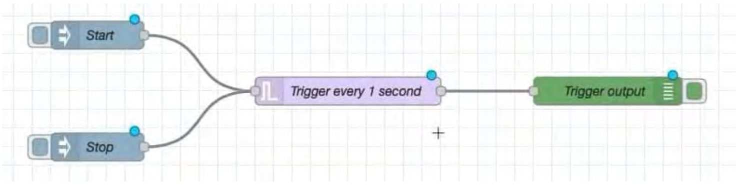
You will find the “trigger” node in the functions group of the left toolbar.



The “trigger” node.

Setup the “trigger” node

To explain how to use the “trigger” node, I have created this simple flow:

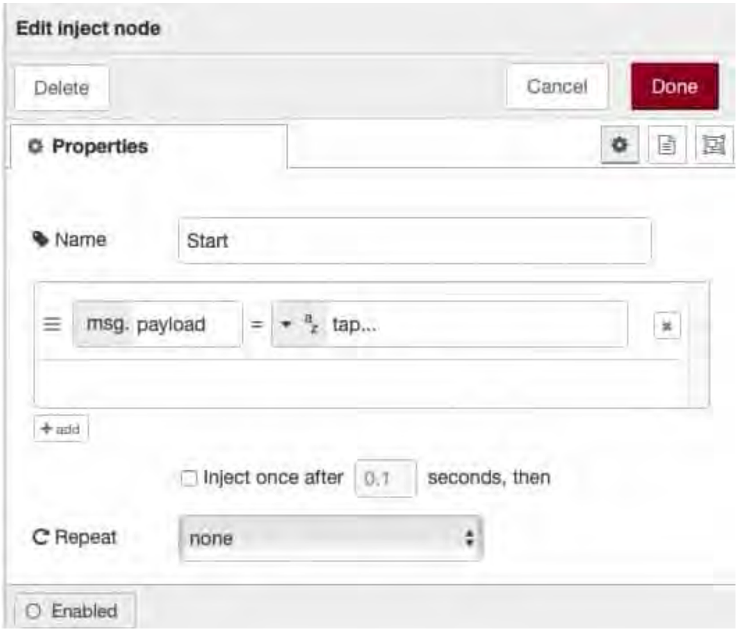


This flow contains a “trigger” node.

When I click on the button of the “inject” node named “Start”, the “trigger” node will send the string “tap…” to the “debug” node, and will continue to do so every 1 second.

When I click on the button of the “inject” node named “Stop”, the trigger node will reset and wait for the next message from “Start”.

Here is the configuration of the Start node:



Name: Start

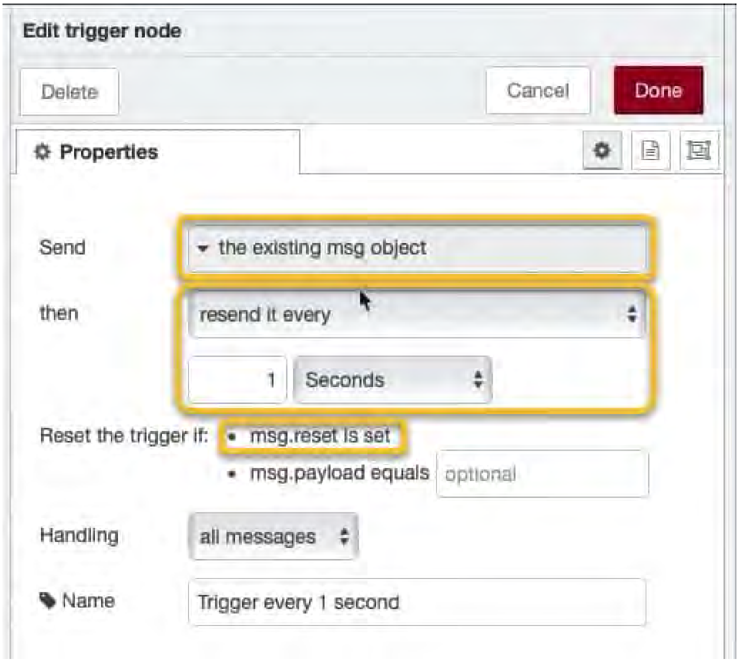
msg. payload = tap...

☐ Inject once after 0.1 seconds, then

Repeat: none

The Start “inject” node. Just sends out a text message.

This is the configuration of the “trigger” node:


Edit trigger node

Send: the existing msg object

then: resend it every

1 Seconds

Reset the trigger if: • msg.reset is set

• msg.payload equals optional

Handling: all messages

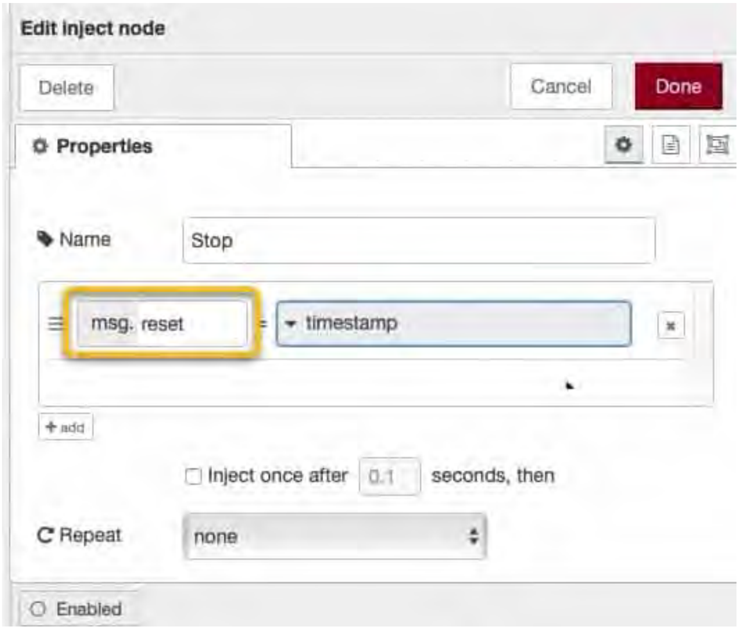
Name: Trigger every 1 second

The configuration of the “trigger” node.

There’s a few options available. I have set this “trigger” node to propagate whichever message object it receives in its input, and to resend it every one second.

Notice that there is an option to reset the trigger by sending a msg.reset (the content of the reset attribute is not important), or by setting a specific value in the msg.payload attribute.

In this example flow, I use the msg.reset method. Here is the configuration of the Stop node:



Name: Stop

msg. reset = timestamp

+ add

☐ Inject once after 0.1 seconds, then

Repeat: none

The configuration of the Stop node.

In the Stop node configuration, notice that I have created the “rest” attribute of the msg object, and set it to contain a timestamp. It does not matter what you store in the reset attribute. It can be a number, a string, a JSON object etc. The actual content is ignored by the trigger node; what matter is that the reset attribute exists.

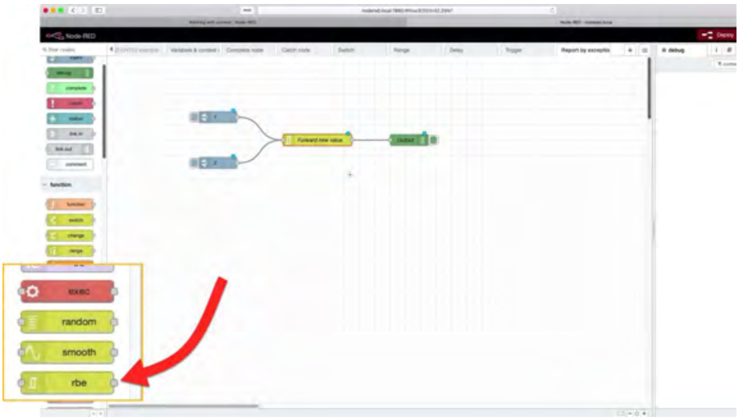
In the terrarium project, among other things, I use a trigger node to get a reading from the DHT22 sensor every 10 seconds.

14. The RBE (Report by Exception) node

Node-RED guide series

Node-RED, the”rbe” node

“RBE” stands for “Report by Exception”. This node will will only pass changes to its output,.



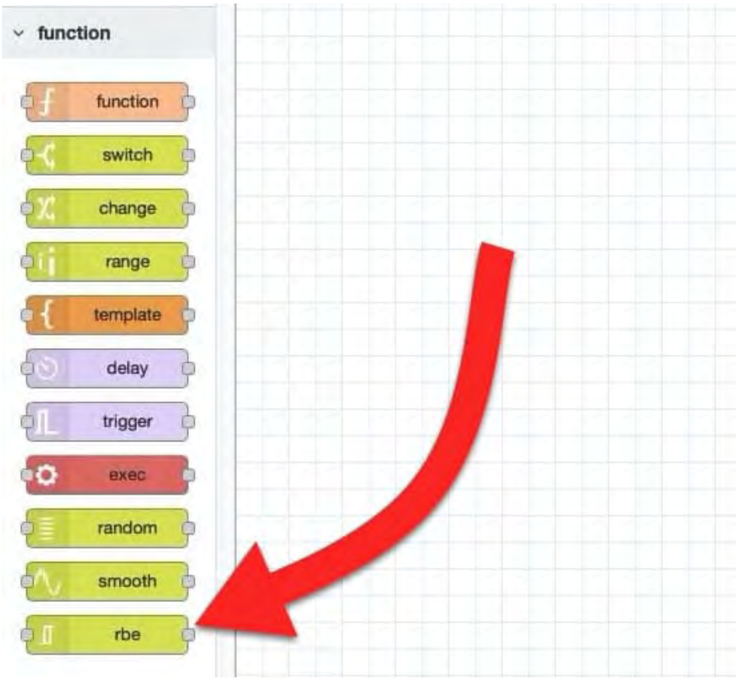
Suppose that you are sending on/off instructions to a motor or other actuator. The motor will turn on after it receives an “on” instruction, and any subsequent “on” instructions will have no effect.

In a Node-RED flow, this is case where you can use an “rbe” node to only propagate changes in the content of a message. With the “rbe” node, a message that is equal to the previous message will not be propagated, while a different message will.

With the motor example (which actually comes from the

Terrarium controller project), the “rbe” flow will block new “on” messages if the last message was also “on”, but will propagate an “off” message.

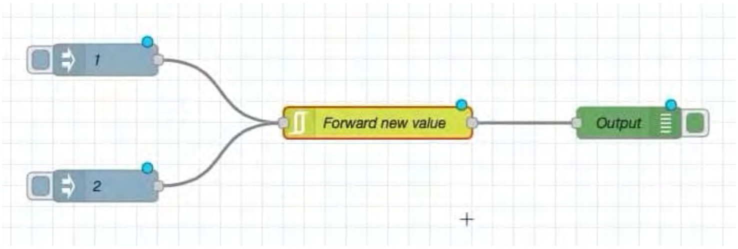
You will find the “rbe” node in the functions group of the left toolbar.



The “rbe” node.

Setup the “rbe” node

To explain how to use the “rbe” node, I have created this simple flow:



This flow contains an “rbe” node.

This flow is triggered by the two “inject” nodes on the left. The “debug” node on the right will simply print out values “1” or “2”, depending on which button was clicked.

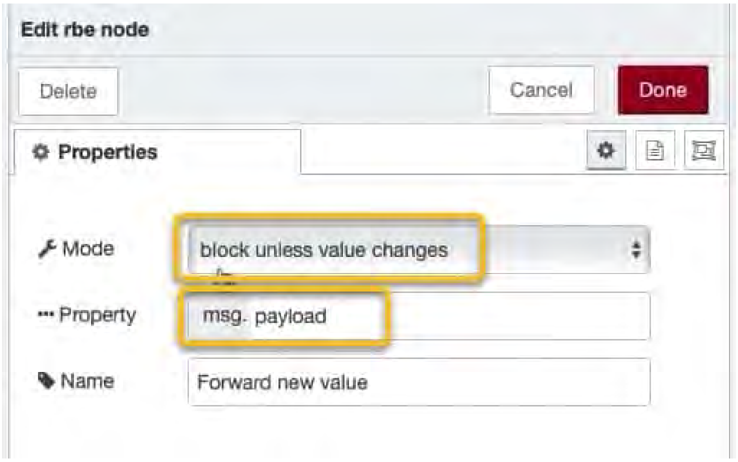
Here’s the interesting part: The message that arrives to the “debug” node is controlled by an “rbe” node. Only changes in the message payload will go through.

So, If you click the inject node named “1” multiple times, the output will show “1” only once.

You will get the same behavior by pressing inject node “2” multiple times.

However, if you alternate your clicks between “1” and “2”, all of the messages will be printed in the debug pane.

Here is how I have setup the “rbe” node



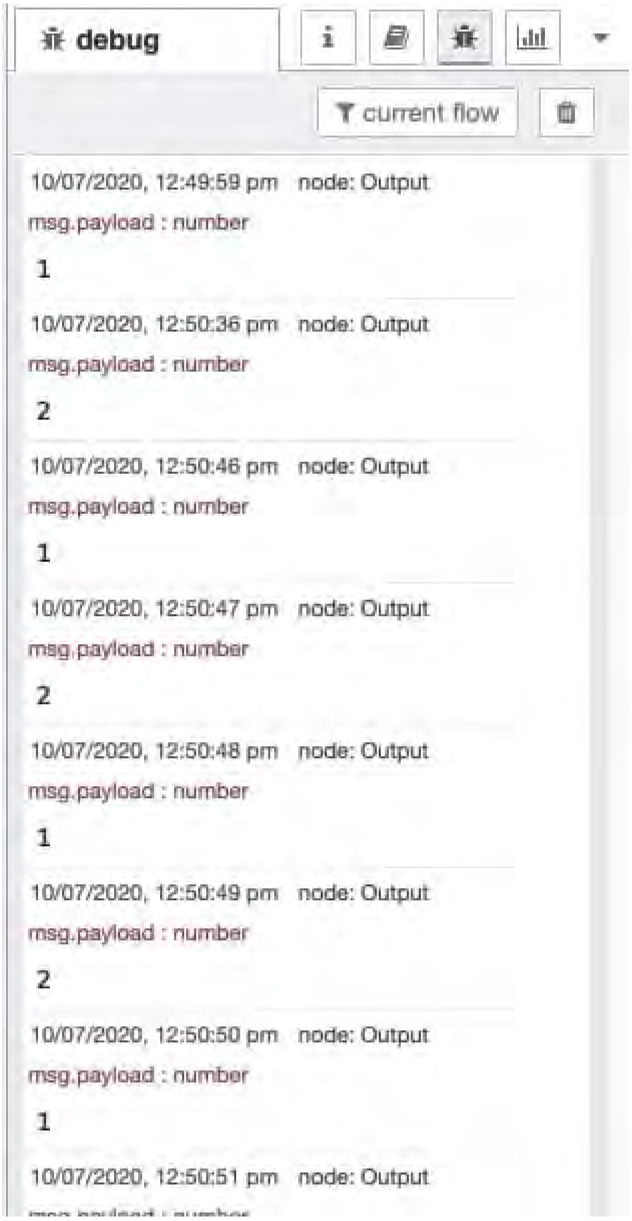
Mode: block unless value changes
Property: msg. payload
Name: Forward new value

The “rbe” node setup.

I have set the mode to “block unless value changes”, and the value that is evaluated to “msg.payload”.

That’s all there is to it.

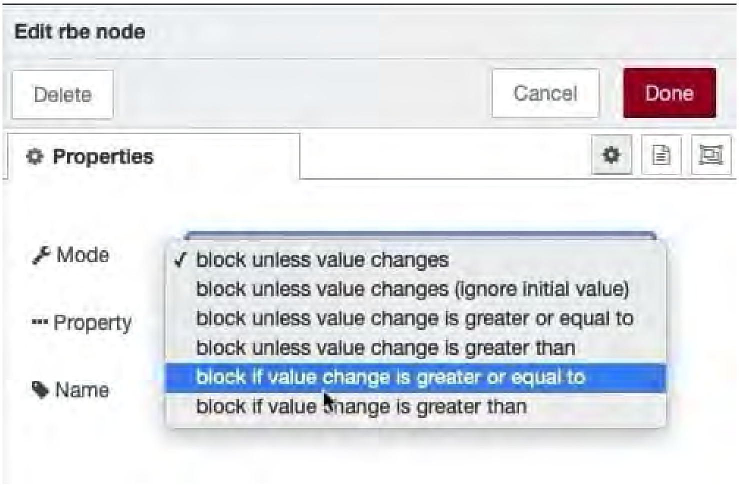
To test it, deploy the flow and click on the two “inject” node buttons. Here is an example output:



```
10/07/2020, 12:49:59 pm   node: Output
msg.payload : number
 1
10/07/2020, 12:50:36 pm   node: Output
msg.payload : number
 2
10/07/2020, 12:50:46 pm   node: Output
msg.payload : number
 1
10/07/2020, 12:50:47 pm   node: Output
msg.payload : number
 2
10/07/2020, 12:50:48 pm   node: Output
msg.payload : number
 1
10/07/2020, 12:50:49 pm   node: Output
msg.payload : number
 2
10/07/2020, 12:50:50 pm   node: Output
msg.payload : number
 1
10/07/2020, 12:50:51 pm   node: Output
```

Notice that the messages alternate between “1” and “2” even though I was frantically clicking on the same button multiple times.

The “rbe” node offers several mode options as you can see in the screenshot below. You can qualify for the kind of change that want to block or allow message propagation.



Mode options:
- ✓ block unless value changes
- block unless value changes (ignore initial value)
- block unless value change is greater or equal to
- block unless value change is greater than
- block if value change is greater or equal to
- block if value change is greater than

“rbe” node mode options.