SUBSCRIBE AND PUBLISH WITH MQTT NODES, *groov* VIEW, & SNAP PAC

When network connectivity is unreliable or bandwidth is an issue, the MQTT data transfer protocol can be the most effective way to efficiently monitor and update the data in your industrial automation control systems.

This example shows how to:

- Use Node-RED to subscribe and publish to an MQTT message stream
- Parse message strings
- Display data from a message in *groov* View
- Push data from *groov* View and from a SNAP PAC controller to a message stream

NOTE: The tutorial uses a groov Box (part number GROOV-AR1-BASE) as an example. The groov Box is a legacy product not recommended for new applications, but it is still available and supported.

What you need

- Node-RED and the *groov* Nodes (preinstalled on a groov AR1 Box running *groov* R3.3 or higher).
- To complete the SNAP PAC Publishing activity, you'll also need:
  - SNAP PAC R- or S-series controller running SNAP PAC firmware R9.5 or higher.
  - The SNAP PAC Nodes for Node-RED.
  - PAC Project 9.5 or higher. (Both the Professional and Basic versions work with this activity.)
  - The SNAP PAC Learning Center strategy (available for free from the Opto 22 Downloads webpage).

To set up Node-RED and the Opto 22 REST APIs, see developer.opto22.com.

PAC Project Basic is available for free from the Opto 22 website.

The latest version of *groov* Box firmware is available at http://manage.groov.com.

Summary of steps

To create this code sample, you'll perform some of the steps in Node-RED and the rest in *groov* View.

- In Node-RED, you'll create flows that subscribe to a public MQTT broker, and format the received messages.
- In *groov* View, you'll create a Data Store and tags for the data in the MQTT messages, and display the data in *groov* View's Build mode.

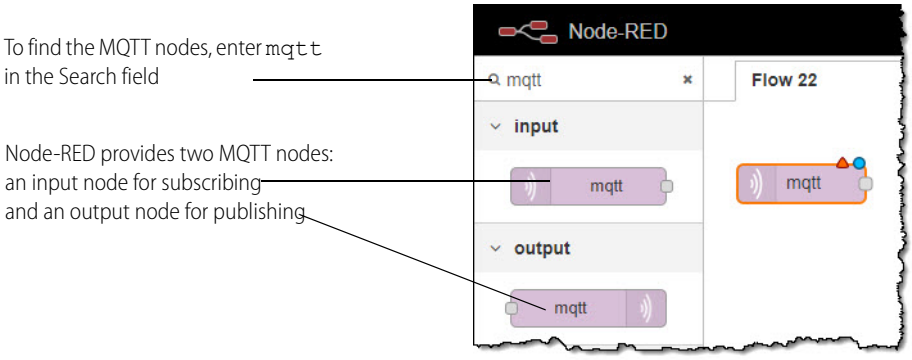
PULL DATA FROM MQTT

Configure an MQTT broker to stream messages

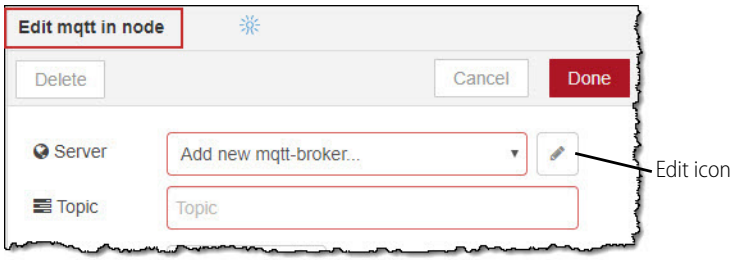
In this step, you'll create a Node-RED flow that subscribes to messages published by a public MQTT broker.

Want the clipboard text to import these flows? Click here.

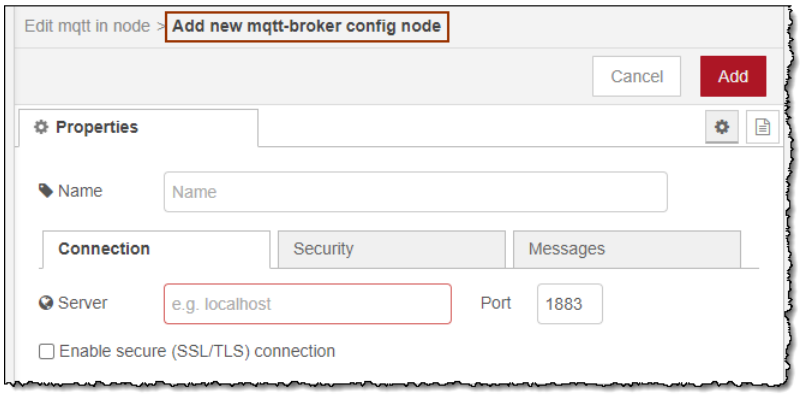
1. Open Node-RED in a browser (The URL is https://*[your* groov-*box-hostname]*:1880).
2. To configure the MQTT broker:
   a. Select an *mqtt input* node and drag it into a new flow.



   b. Double-click the mqtt input node to open the Edit mqtt in node dialog box.



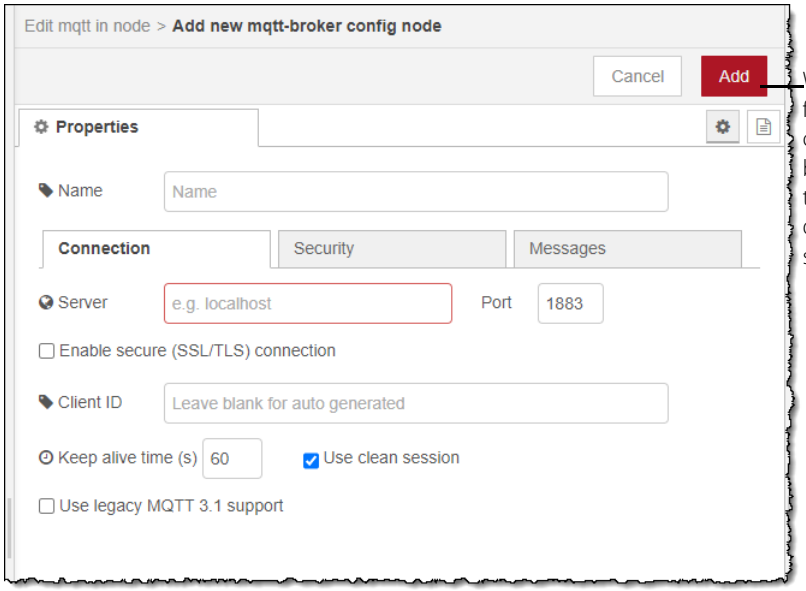
   c. Now, click the edit icon ( ) to open the Add new mqtt-broker config node dialog box.



d. On most systems, you can host an MQTT broker on localhost (in which case you would type `localhost` in the Server field). But for this example, you can use the public *groov* View MQTT broker that's running on a demo server in the Opto 22 headquarters in Temecula, California.

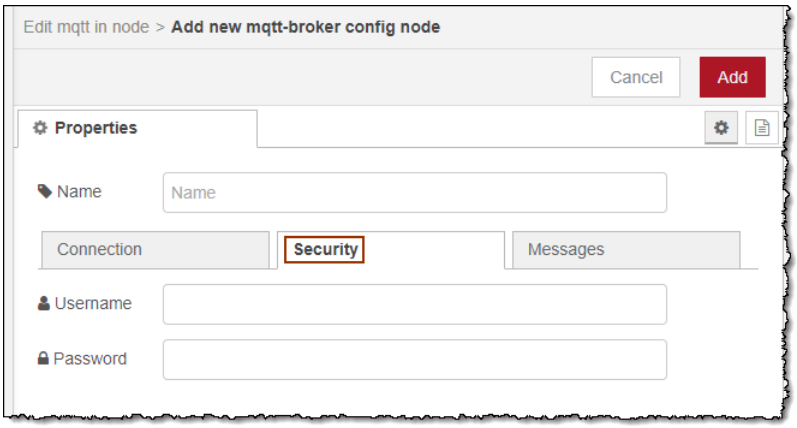
On the Connection tab:

- In the Server field, type: `mqtt.groov.com`
- Keep the default port (1883).
- Leave the Client ID field blank to auto-generate a unique client ID.
  Alternatively, you could use the client's MAC address to ensure that the Client ID is unique.
- Uncheck the box for Use legacy MQTT 3.1 support



When you're finished configuring the broker, click Add to close the dialog box and save the settings

- Fill in the tabs on the Security tab if the broker you're using requires them (for example, if you're connecting to a broker over SSL/TLS).
- If you're using the mqtt.groov.com broker:
  - In the Username field, type: `opto`
  - In the Password field, type: `opto22`



- The fields on the Birth Message and Will Message tabs are optional. For the record:

- When the client connects, the broker will publish a message by using the settings on the Birth Message tab
- When the client disconnects, the broker will publish a message by using the settings on the Will (as in Last Will and Testament) Message tab.

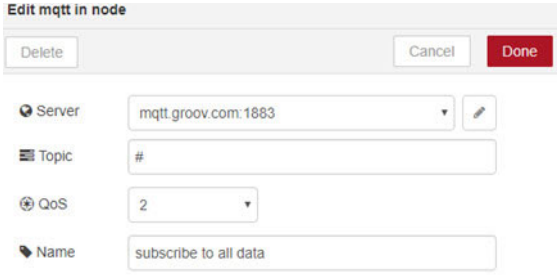
e. Click Add.

The Edit mqtt in node dialog box is displayed again, and this time the Server field shows mqtt.groov.com:1883.

- In the Topic field, type: **#**

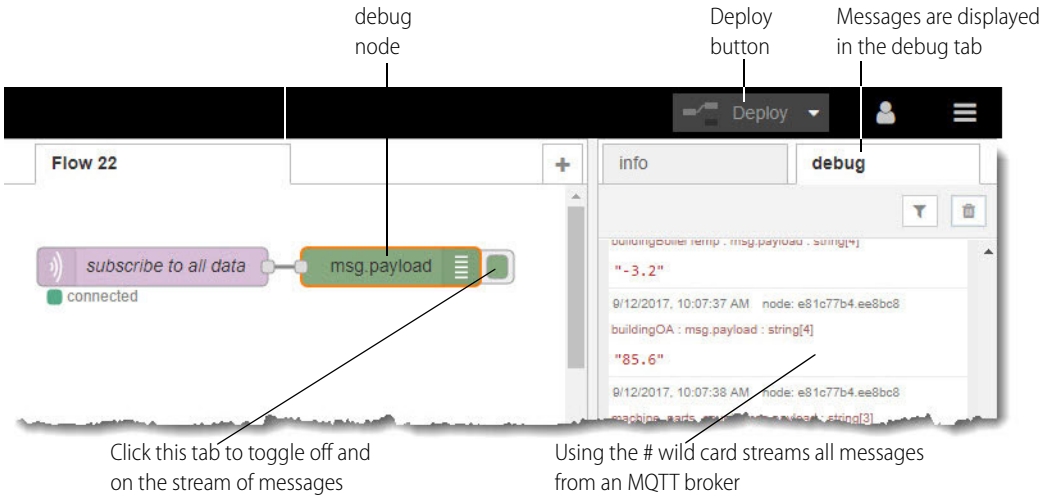
  A topic is a UTF-8 string that the broker uses to filter messages for each connected client. A broker can publish messages from several streams; for example, the broker at mqtt.groov.com:1883 publishes data for a fictional factory and data about recent earthquakes.

  The # character is a multi-level wild card that subscribes to *every* message the broker publishes. To subscribe to a specific topic, replace **#** with the topic's name.
- (Optional.) In the Name field, type a name for the node.



f. Click Done to close the Edit mqtt in node dialog box.

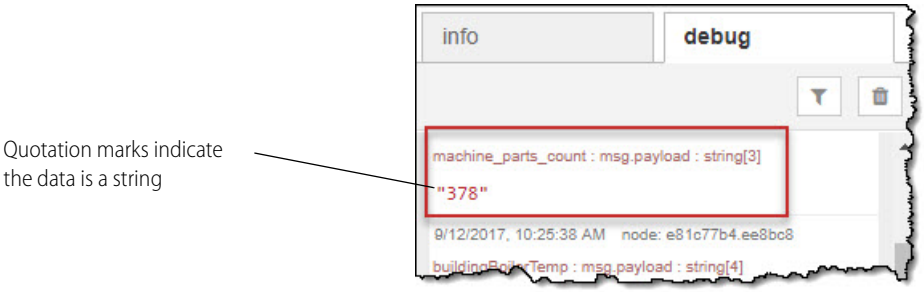
3. Now, add a debug node to the flow, wire it to the mqtt node, and then click Deploy to start streaming messages from mqtt.groov.com.



4. Let the flow run a little while to build up some data. When you're ready, click the tab on the right side of the debug node to stop streaming.

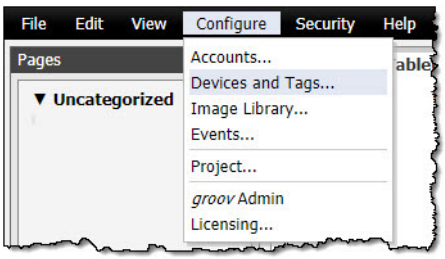
Configure *groov* View to get and display data from the stream

The mqtt.groov.com broker regularly publishes a topic named *machine_parts_count*. Although its values look like integers, they are configured as strings.



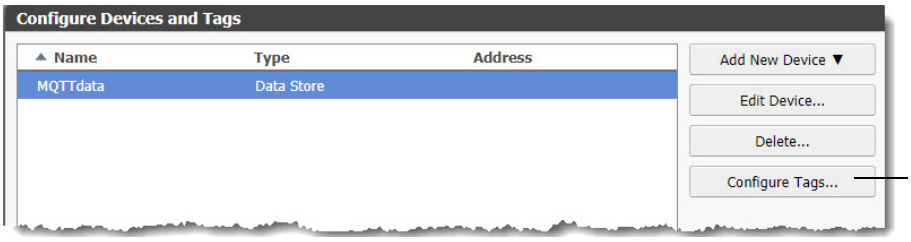
In the next steps, you'll configure *groov* View to store and display the current value of machine_parts_count.

1. In a browser window, open *groov* Build (http://<*your* groov *Box's hostname*>/#build), and then add a new page. (To add a page, click the Add Page link, type a page name, and then click OK.)
2. In the menu bar, click Configure > Devices and Tags.



3. In the Configure Devices and Tags window, click Add New Device, and then click Data Store.

4. In the Add Data Store dialog box, type `MQTTdata` and then click Create. Your new Data Store is displayed in the Configure Devices and Tags window.



5. Now click Configure Tags. The Configure Data Store Tags window is displayed.

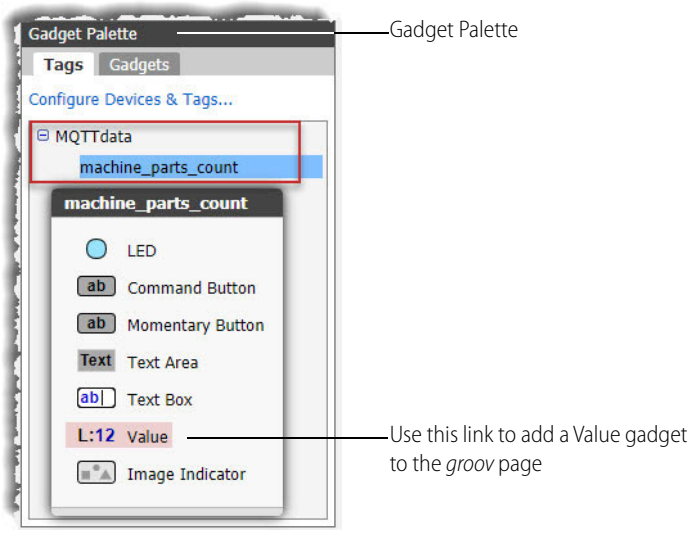
6. Click the Add Tag icon ( ) to start configuring a tag for machine_parts_count.
   a. In the Tag Name field, type: `machine_parts_count`
   b. In the Tag Id field, select: String

   c. Click Save, then click Close, and then click Close again to close the configuration menus and save your changes.

TIP: To quickly add a tag name:

- *In the Identifying Name field, type the tag name.*
- *Press Tab, and then type* **S** *(for string),* **I** *(for integer), or* **D** *for (decimal number).*
- *Press Tab. If the tag is* not *a table, press Tab again, and then press Enter to save the tag.*
  - *If the tag is a table, press the space bar to select the Table check box.*
  - *Press Tab, and then type the table length.*
  - *Press Tab again, and then press Enter to save the table tag.*

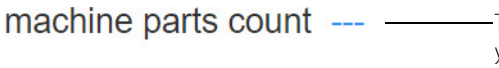
7. In the Gadget Palette's Tags window, expand MQTTdata, and then double-click machine_parts_count.



8. In the pop-up menu, double-click the Value link to create a Value gadget for machine_parts_count.

To adjust the size and look of the gadget, use the fields in the Value Gadget Properties dialog box

9. In the menu bar, click File > Save All Changes and Switch to *groov* View.

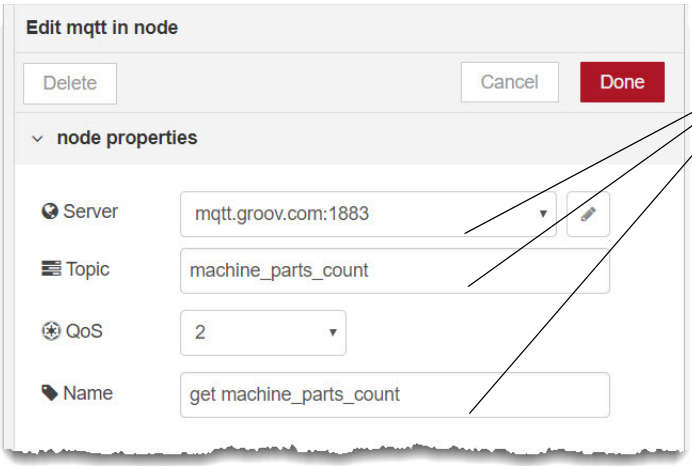


The gadget won't have a value because you haven't written to it yet

Subscribe to a topic and view the data in *groov* View

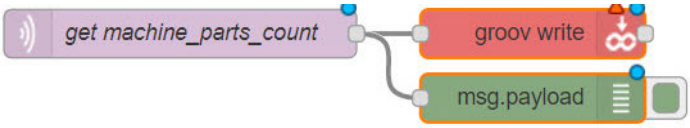
Now you'll create a new flow to subscribe to the machine_parts_count topic and feed the data to *groov* View.

1. In Node-RED, add a new mqtt input node to the flow, and then configure it to get only the machine_parts_count topic from the mqtt.groov.com broker.



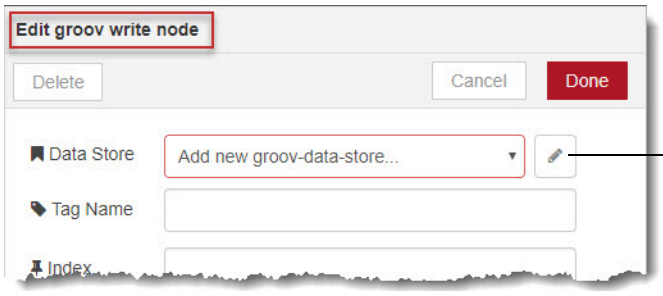
In the new mqtt node, configure the broker, the topic, and (optionally) a name for the node

2. Click Done to close the dialog box and save your changes.
3. Now add a *groov* write node and a debug node to the flow, and wire them to the new mqtt input node.

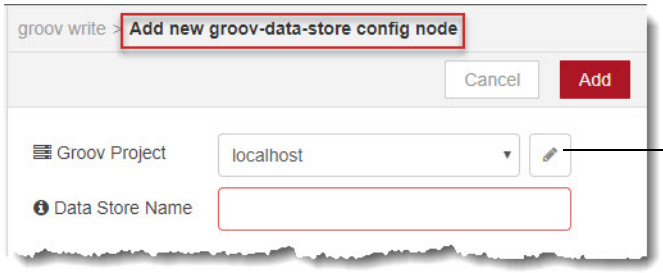


4. Next, you'll need to add your *groov* View API key to the *groov* write node. If you don't have a *groov* View API key, follow the instructions in http://developer.opto22.com/groov/#setting-up-api-users to configure one.

a. Double-click the *groov* write node. The Edit groov write node dialog box is displayed.



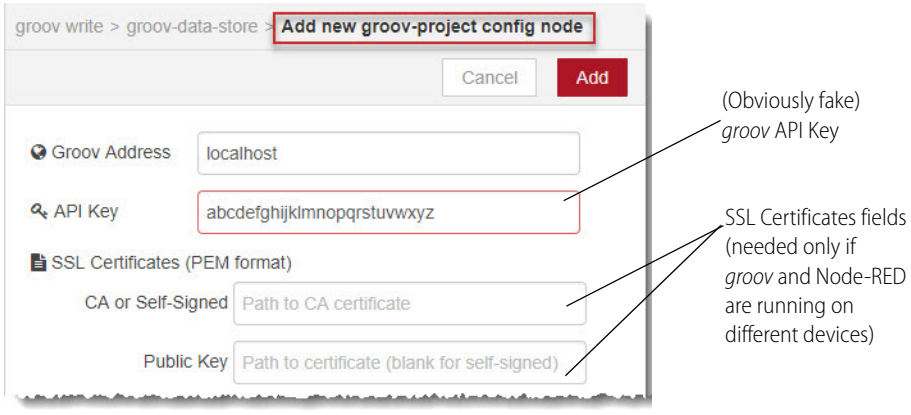
b. Click the Edit icon to open the Add new groov-data-store config node dialog box.



c. Click the Edit icon again to open the Add new groov-project config node dialog box.

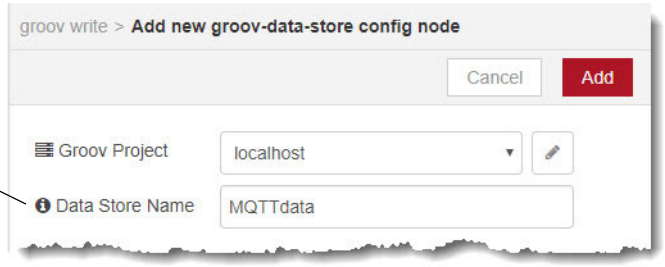
- If you're running Node-RED and *groov* View on the same *groov* Box, the value in the Groov Address field should be "localhost."
- If Node-RED and *groov* View are running on different devices, you'll also need to configure the SSL Certificates fields. (For instructions, see the Opto 22 Developer Portal website.)

In the API key field, enter your *groov* View API key. (It's in *groov* Build, under Configure > Accounts.)



d. Click Add. The Add new groov-data-store config node dialog box is displayed again.
In the Data Store Name field, enter: `MQTTdata`

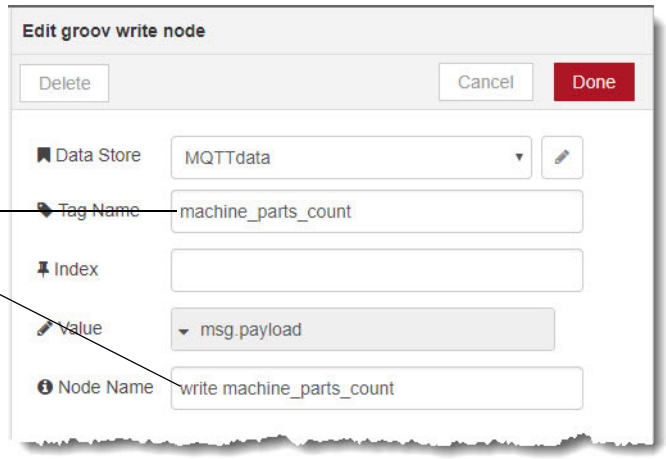
This value needs to exactly match the name you gave the Data Store in *groov*



e. Click Add. The Edit groov write node is displayed again.

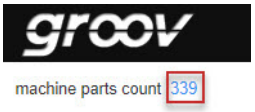
In the Tag Name field, enter: `machine_parts_count`

Configure the tag name and, optionally, give a name to the node



5. Click Done to close the dialog box and save your changes, and then click Deploy.

If the deployment is successful, go back to *groov* View to see the machine parts count being updated!

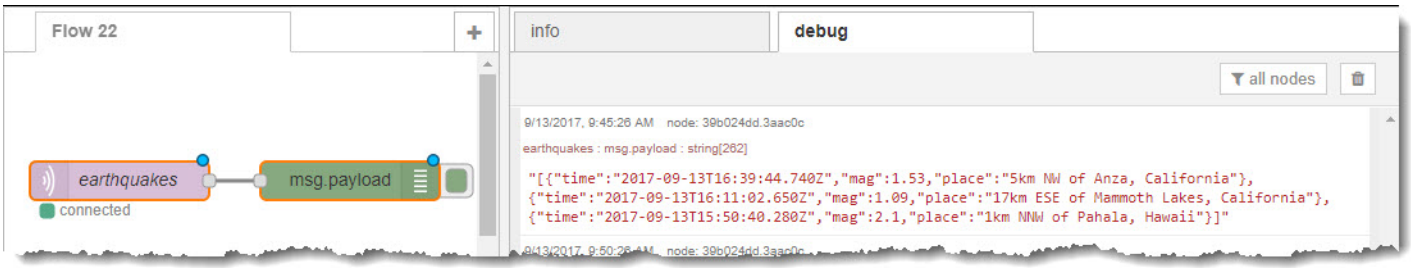


If something goes wrong, check the Node-RED flow window and the debug tab for error messages. (For example, "unknown tag name" indicates the tag name is misspelled or missing in Node-RED.)

CONVERT A JSON-FORMATTED STRING, AND THEN DISPLAY ITS VALUES

Want the clipboard text to import these flows? Click here.

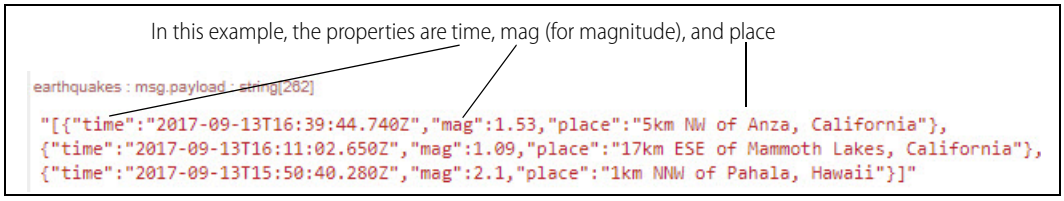
Another topic published on the *groov* View MQTT broker is *earthquakes*. Like everything in the MQTT cloud, the earthquakes topic is a string object. But this string looks a lot like a *JSON* (JavaScript Object Notation) object.



The earthquake topic is published about every five minutes on the public *groov* MQTT broker.

JSON objects consist of *name-value pairs* (sometimes called key-value pairs) separated by commas within objects defined by curly braces. At either end of the string are square brackets, indicating an array of objects.

Each object has the same set of properties. (Properties are the "name" part of a name-value pair.) Of course, the value part of the name-value pair may vary by object and property.

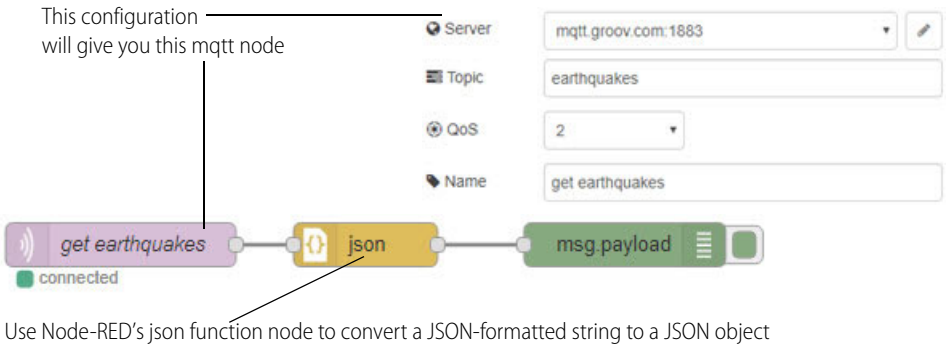


The reason this particular string is formatted this way is that it was a JSON object *before* it was published to the broker. The broker handled the message as a string; consequently, it pushes the data to subscribers as a string.

But many times, you want a message to be a JSON object rather than a string so you can parse it and get the individual pieces of data. Conveniently, Node-RED includes a json function node that converts data from a string to a parse-friendly JSON object.

Ready to try? Here you go!

1. In Node-RED:
   a. Create a new flow consisting of a subscription node to the earthquakes topic (in mqtt.groov.com), a json function node, and a debug node.



b. Deploy the flow, and wait a little while for the earthquake data to be published. Then, examine the reformatted object in the debug tab.

Note the data type for each property:
– time is a string
– mag is a float (decimal number)
– place is a string

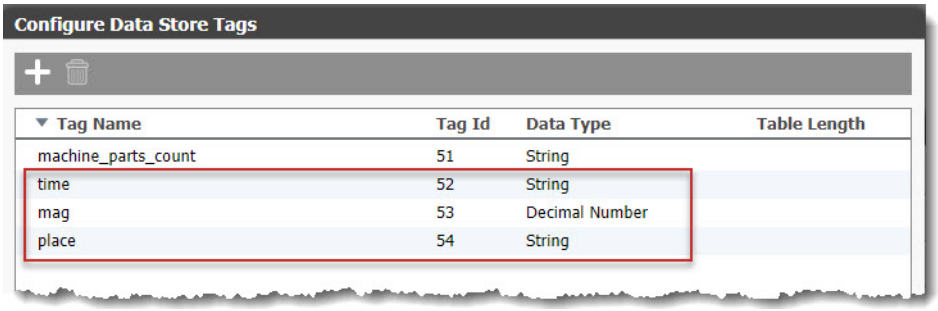
Don't see the messages in the debug tab?
You made need to wait a little longer.
Earthquake data is published about every five minutes or so.

TIP: When you use the wild card (#) in the Topic field, the subscription stream may be easier to read if you run every topic through a json node. However, be aware that some objects may fail to parse *(depending on how the broker handles the message).*

Now that the message is a proper JSON object, *groov* View can easily access each property in the array.

2. In *groov* View, add tags and create gadgets for the properties:

a. In the MQTTdata Data Store, add tags for time (string), mag (decimal number), and place (string).



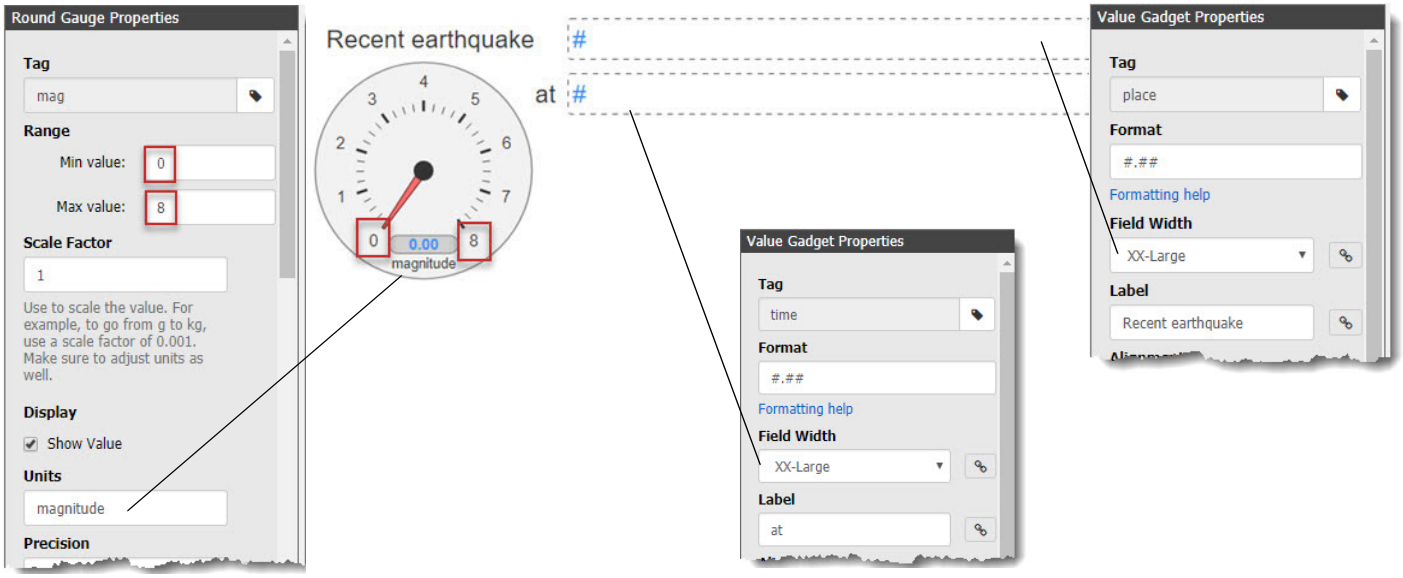
Tag Name	Tag Id	Data Type	Table Length
machine_parts_count	51	String	
time	52	String	
mag	53	Decimal Number	
place	54	String	

b. From the Gadget Palette, add three gadgets:

- Value gadget for place, and then label it to show where the most recent earthquake occurred.
- Value gadget for time to display what time the earthquake occurred.
- Round Gauge gadget to display magnitude.

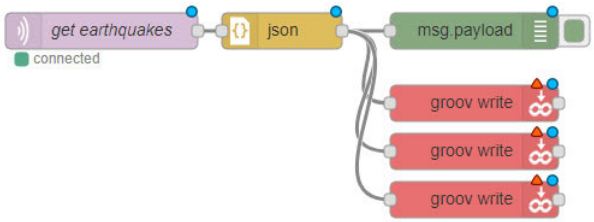
The place and time strings are quite long, so set the Field Width to XX-Large for both of them.

Since earthquakes are measured from 0 to 8 degrees in magnitude on the Richter scale, set the Round Gauge gadget's Range values from 0 to 8.



3. In Node-RED, create nodes that will write to the gadgets in *groov* View:

a. Add three *groov* write nodes to the flow, and then wire them to the output of the json function.

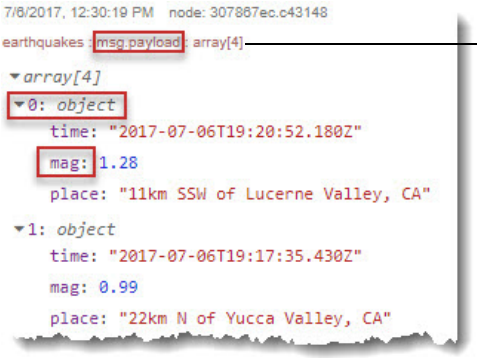


b. Configure a *groov* write node for each of the three properties: time, mag, and place.

Regarding the Value field:

groov View references a specific JSON object by its position in the msg.payload array.

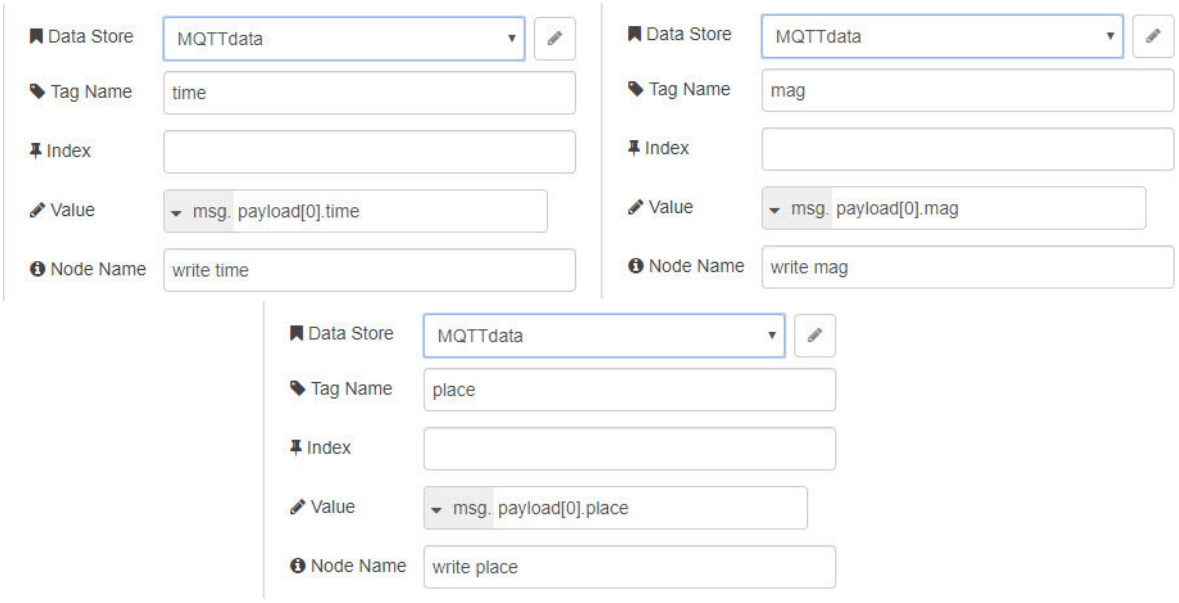
- The position number is enclosed in square brackets **[ ]**.
- The first object in the array is position **[0]**.
- A period (**.**) separates the position and the property's name.



There are four objects in msg.payload
(Only two are shown)

When you configure a *groov* write node to write the value of `msg.payload[0].mag`
groov displays the magnitude of object 0: in this case, **1.28**

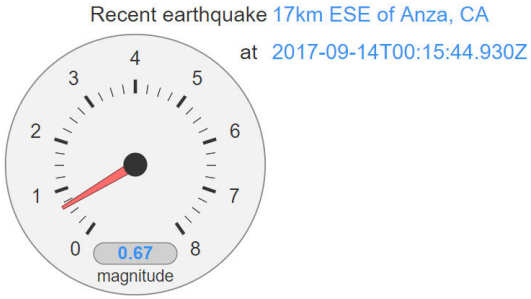
Since you created gadgets for only one instance of time, mag, and place, configure the nodes for the properties of the *first object* in the array (the object at position [0]).



NOTE: When configuring the Value field, be sure to use the Value drop-down list to select msg. (instead of msg.payload), and then type the rest of the reference; for example: `payload[0].time`

This ensures that you don't accidentally try to write the entire payload to the tag.

4. Deploy the flow, and then look in *groov* View to see your data. (You may need to wait a few minutes before earthquake data is published to the broker.)



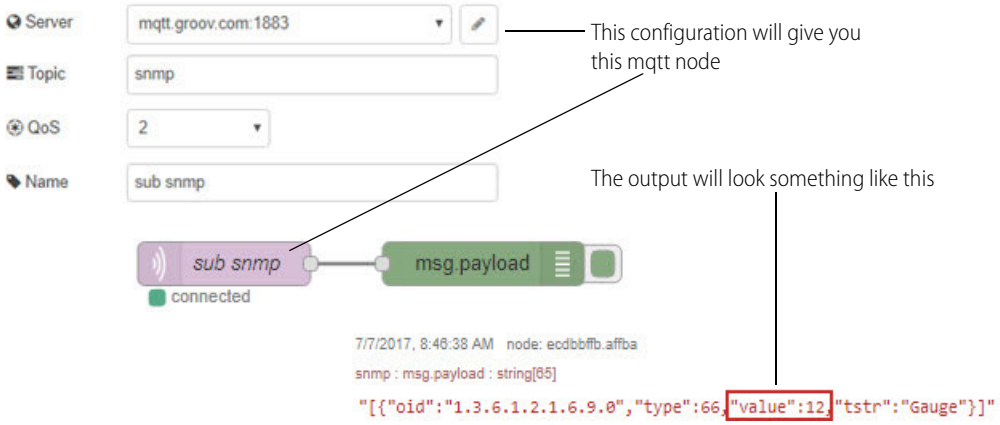
USE A FUNCTION NODE TO CONVERT STRINGS

Want the clipboard text to import these flows? Click here.

Not all data can simply be flowed through a built-in conversion node (like the json node) and distributed from the other side. There may be more complex parsing or conversion involved, in which case the *function* node is a much more flexible (and in some ways, more powerful) tool.

1. Let's say you want to get the value of *session counts*, which mqtt.groov.com publishes in the *snmp* topic.
- **a.** In Node-RED, create a new subscription for the snmp topic in mqtt.groov.com.
- **b.** Wire the output directly to a debug node, and then deploy the flow.

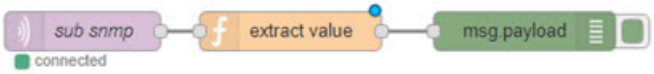
Note that the output is a JSON-formatted string, with the properties *oid*, *type*, *value*, and *tstr*. The value property holds the session count (12, in this example).



But what if the output wasn't a nicely JSON-formatted string? In that case, you couldn't use Node-RED's json function to easily parse the data. That's when the function node can help.

2. Drag a function node between the mqtt and debug nodes. (If you place it just right, Node-RED will wire them together automatically.)

Name the node: `extract value`



You'll be adding code in the function node to parse this string:

```
"[{"oid":"1.3.6.1.2.1.6.9.0","type":66,"value":12,"tstr":"Gauge"}]"
```

The values you get from the live stream may vary from the ones in this example.

When coding in real life, the code you use in a function node will depend on:

- The object you're getting
- How the object is formatted
- How you expect it to change

But when you're parsing a string, the functions you'll need are essentially the same:

For regular expressions, use:	Conversion can be done with:
str.substring(start int , end int)	parseInt(str)
str.split(str)	parseFloat(str)
str.indexOf(str)	Number(str)
str.lastIndexOf(str)	
str.search('str')	
NOTE: parseFloat("13km") => 13, but Number("13km") => NaN. When deciding which function to use, be mindful of the input you expect and the kind of data you want to output.	

Two ways to parse strings

Try any of all of these code samples in your extract value node to see the results.

1. str.split(",")

Using str.split(","), you can break the payload string into an array containing four parts:

```
[{"oid":"1.3.6.1.2.1.6.9.0"
"type":66
"value":12
"tstr":"Gauge"}]
```

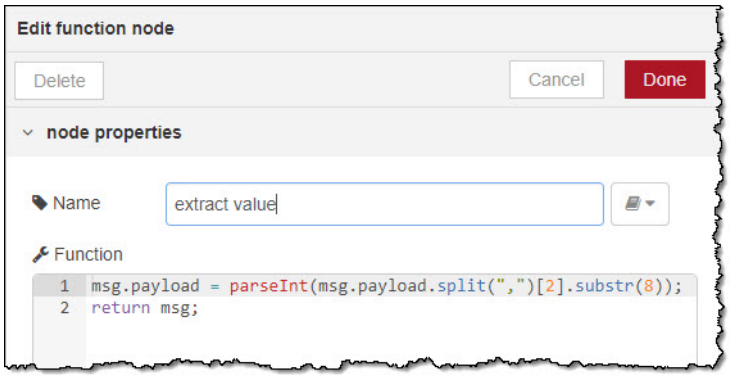
Then you'd return part of array object [2] (namely, `"value":12,`), and cast it as an integer.

Here's the code:

```
var parts = msg.payload.split(",");     // break the string into parts
var num = parts[2].substr(8);           // grab characters after "value":
msg.payload = parseInt(num);            // convert the value to an integer
return msg;                             // return integer value in payload
```

The same code, written more compactly:

```
msg.payload = parseInt(msg.payload.split(",")[2].substr(8));
return msg;
```



The reason for `.substr(8)` is to skip over the first eight characters (`"value":`). The second parameter of `.substr(8)` is omitted, which allows the code to collect the first character after `"value":` and all other characters until the end of the string. The resulting string is cast as an integer and returned through the payload.

The drawback to this method of splitting a string is its inflexibility: if the topic is published with more or fewer commas before `"value":12` (for example, if the topic included an additional property), this code would fall apart and require maintenance.

2. indexOf()

A more flexible way to parse data is to search the string for `"value"`, and then get the number between the next colon and the following comma, no matter how long or short it is.

This code sample locates the first instance of `"value"`, and then gets rid of all preceding data. That way, `indexOf` doesn't find any commas or colons from earlier properties. Then, it uses the index of each character to define a new string, convert it to an integer, and return it in the payload.

```
var mkr = msg.payload.indexOf("value"); // mark where "value" is
var str = msg.payload.substr(mkr);      // get rid of preceding data
var start = str.indexOf(":")+1;         // find where the number starts
```

```
var end = str.indexOf(",");              // find where the number ends
var val = str.substring(start,end);      // get data between : and ,
msg.payload = parseInt(val);             // convert the value to an integer
return msg;                              // return integer value in payload
```

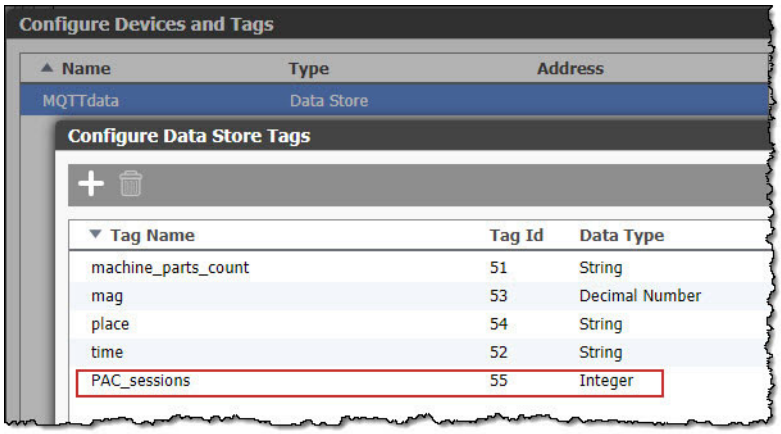
Or compactly:

```
var str = msg.payload.substr(msg.payload.indexOf("value"));
msg.payload = parseInt(str.substring(str.indexOf(":")+1, str.indexOf(",")));
return msg;
```

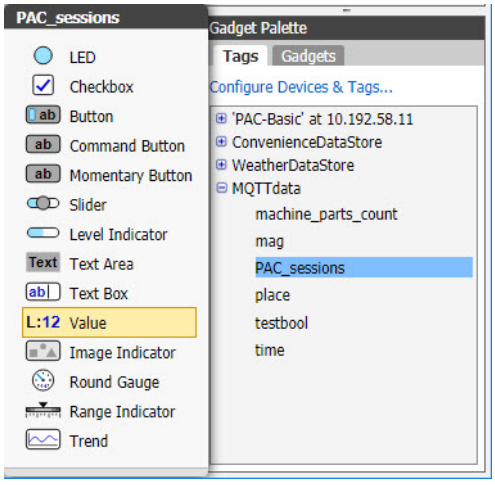
Congratulations! You've parsed the number of sessions running on the PAC controller without needing a JSON-formatted string.

Now you can create a *groov* View Data Store tag to hold the data, and use Node-RED to push the integer to *groov* View.

1. In *groov* Build, add a new integer tag for PAC_sessions:



2. Click Save, and close out of the configuration menus. Then, add a Value gadget for PAC_sessions to your *groov* View page.



3. In the *groov* View menu bar, click File > Save All Changes and Switch to *groov* View. The gadget will read as "PAC sessions 0" until you write to it, so...

4. Go back to Node-RED. Add a *groov* write node to the flow, and configure it to write to PAC_sessions.

5. Click Deploy, and then check out the updated gadget in *groov* View!

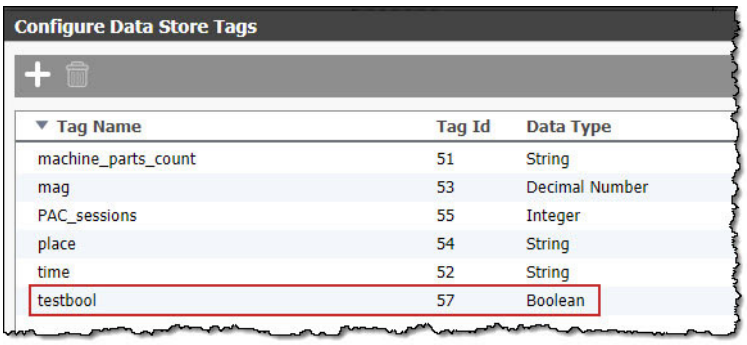
PUBLISH DATA FROM *groov* VIEW TO MQTT

Want the clipboard text to import these flows? Click here.

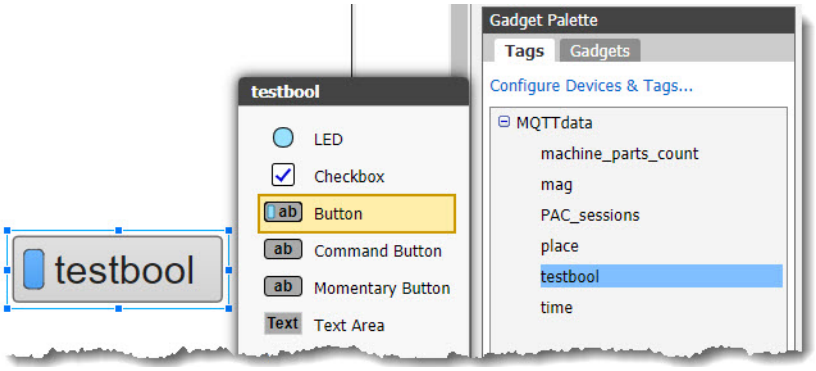
Instead of subscribing to a stream of MQTT messages, you may want to publish data from *groov* View to MQTT. The process is practically the reverse of writing from MQTT to *groov* View.

In this section, you'll create a True/False tag in the *groov* View Data Store and publish its value to MQTT.

1. In *groov* View, add a new Boolean tag and name it: `testbool`

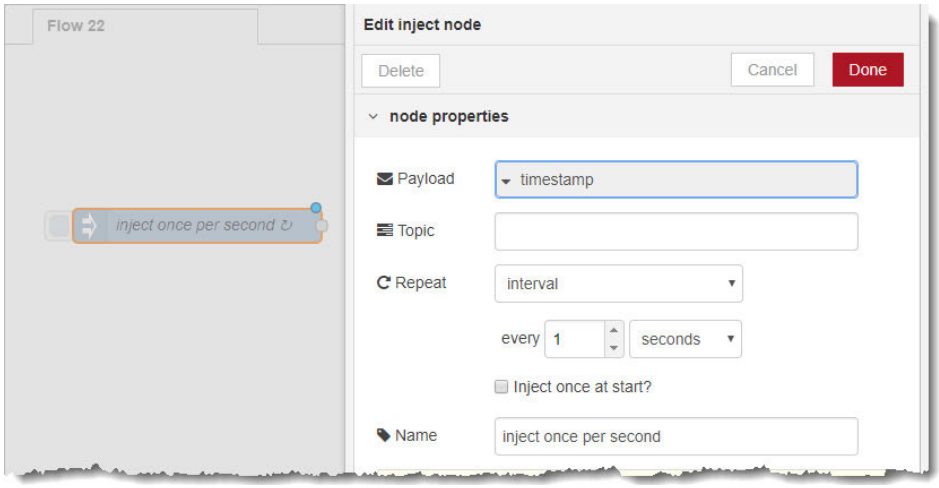


2. Close out of the configuration menus, and then add a Button gadget for testbool to your *groov* View page.

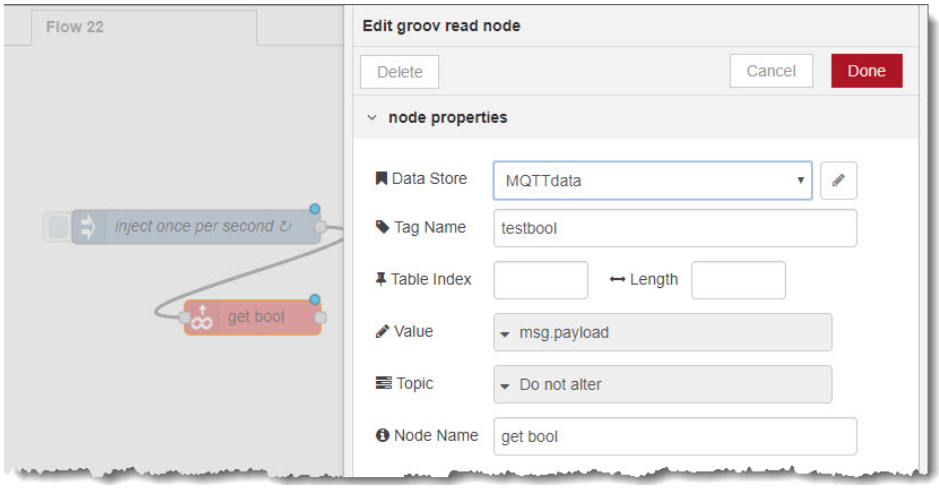


3. Save the changes and switch to *groov* View.

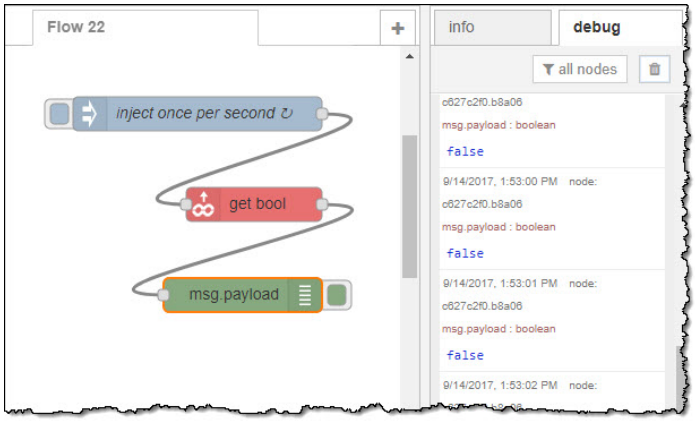
4. In Node-RED:
   a. Add an inject node (located with the input nodes) and configure it to repeat every second.



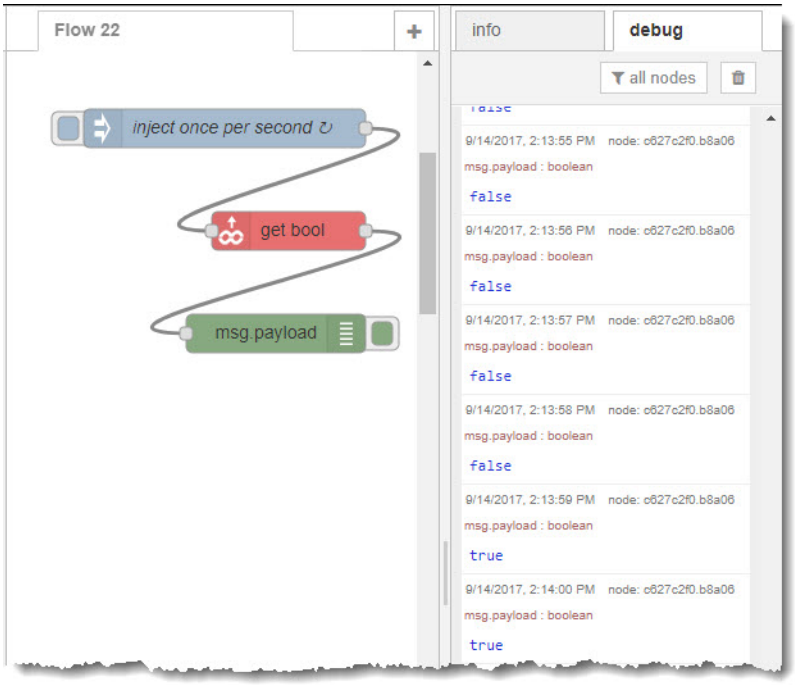
5. Wire the inject node to a *groov* read node that you've configured to read the testbool tag.



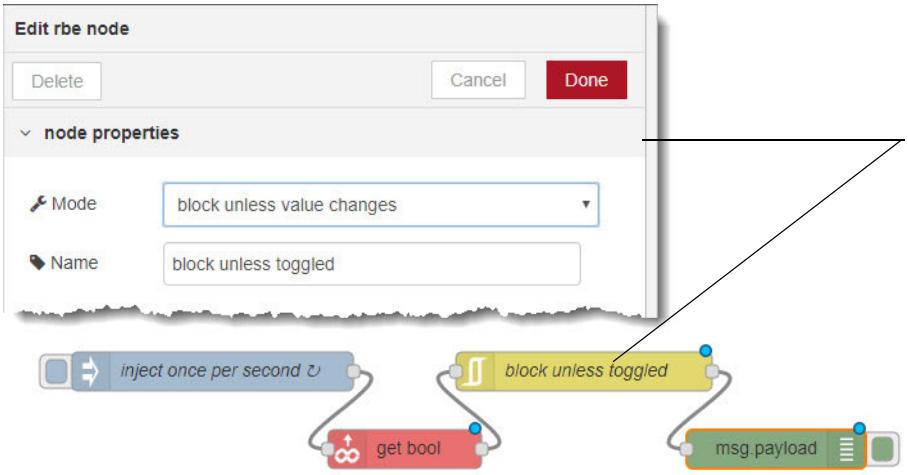
6. Before you publish to the mqtt.groov.com broker, test the flow: add a debug node, deploy the flow, and then watch for data on the debug tab.



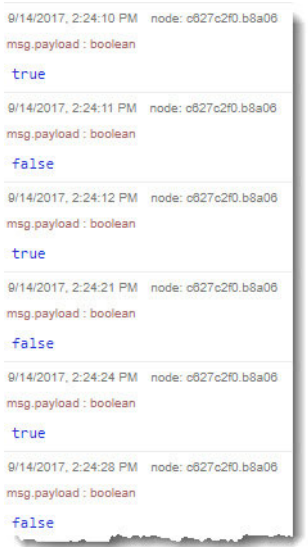
7. Now, open *groov* View and toggle the testbool button off and on.
   In Node-RED's debug tab, watch the value as it changes.
   Clearly, the flow is working. You get "false" when the button is off, and "true" when the button is on.



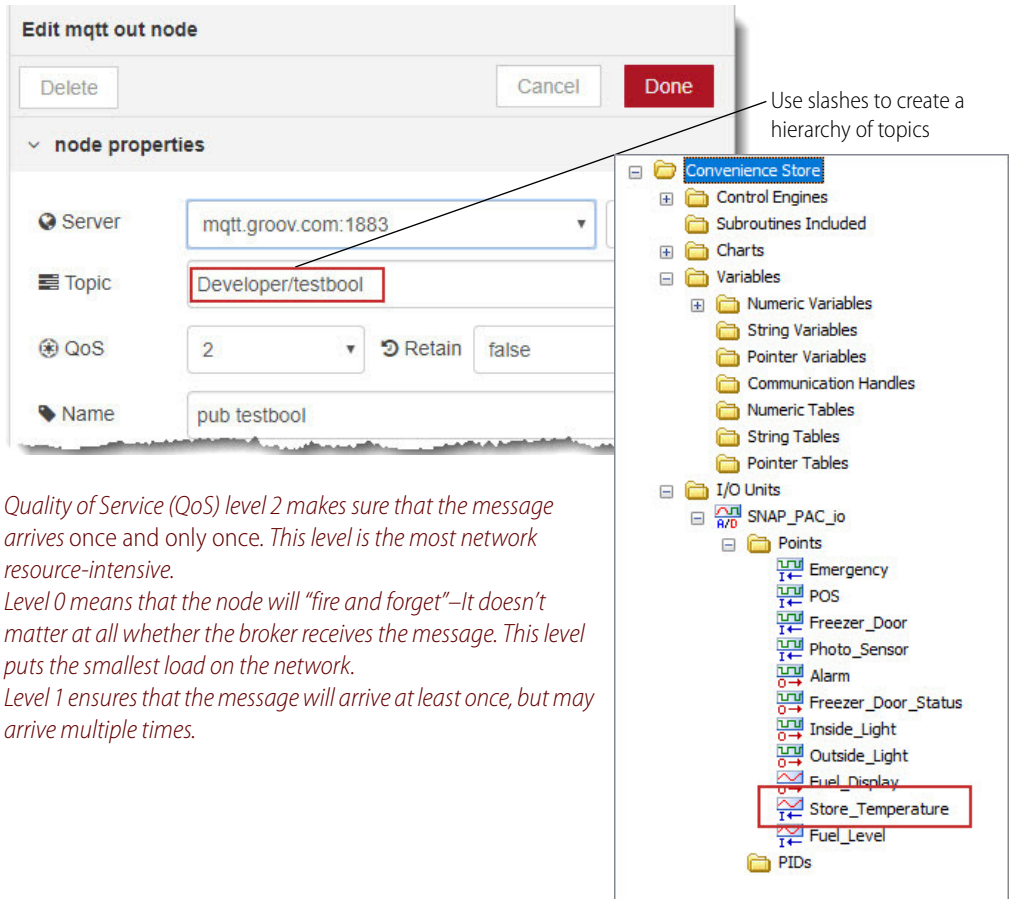
8. However, it would be wasteful to push such a noisy signal to MQTT every second. Instead, let's add a function node, *rbe* (rate by exception), that will publish the value only after it changes.
   a. Drag an rbe function into the flow and configure it to "block unless the value changes."



9. Now deploy, and you'll see that a message is displayed only when the value changes; hence, the messages alternate true, false, true, false.

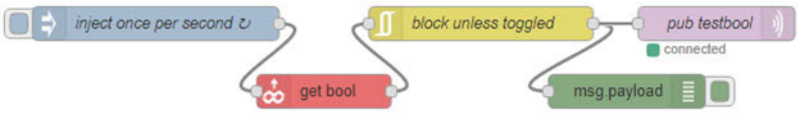


10. Next, add an mqtt output node to the flow, and configure it to publish the data to mqtt.groov.com under the subtopic "Developer."



Quality of Service (QoS) level 2 makes sure that the message arrives once and only once. *This level is the most network resource-intensive.*
Level 0 means that the node will "fire and forget"–It doesn't matter at all whether the broker receives the message. This level puts the smallest load on the network.
Level 1 ensures that the message will arrive at least once, but may arrive multiple times.

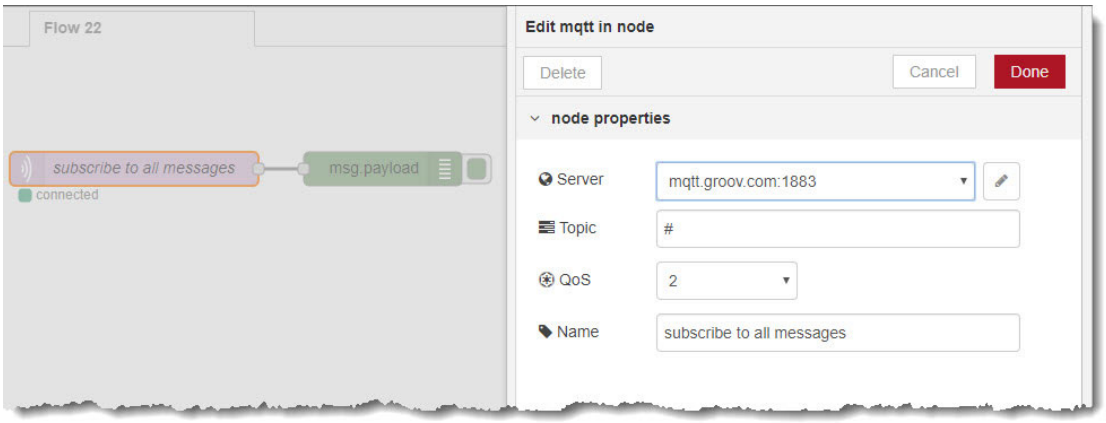
11. Wire the mqtt output node directly to the rbe node, and then deploy the flow.



The output from this flow will look the same in the debug window as it did before. The difference is that now data is also being published to the mqtt.groov.com broker.

So how can you verify that you're truly publishing your data to the broker?

The easiest way is to create a flow that uses the wild card (#) to subscribe to all messages on mqtt.groov.com.



TIP: To subscribe only to the testbool messages, change the topic to Developer/testbool. However, you're going to want to see other messages in the next section, so you should probably leave the wild card topic for now.

Deploy the flow—and then watch for the incoming testbool messages as you toggle the button off and on.

PUBLISH DATA FROM A PAC CONTROLLER TO MQTT

Want the clipboard text to import these flows? Click here.

We've saved the best for last: Hooking into data on an Opto 22 SNAP PAC controller, and publishing it to MQTT.

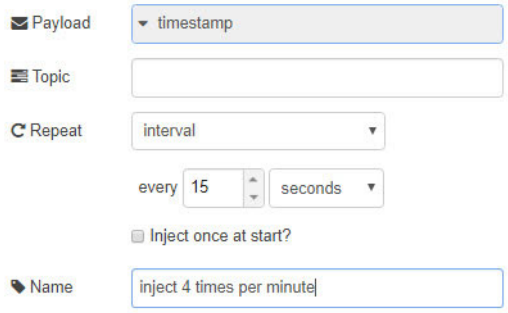
This activity was written using a SNAP PAC Learning Center (part number SNAP-PACLC) with PAC Control running the PAC Control convenience store strategy (Convenience_Store.idb). Both PAC Control Basic and the Learning Center tutorial (including Convenience_Store.idb) are available for free from the Opto 22 Downloads webpage.

You can also perform this activity using your own SNAP PAC R- or S-series controller—you just may need to reconfigure the convenience store strategy to work with your controller and I/O modules.

Let's start by getting the value from the convenience store's temperature sensor every 15 seconds.

1. In PAC Control, open Convenience_Store.idb and run the strategy. *Store_Temperature* is the analog input from the temperature sensor, and that's the tag you'll want to read and publish.

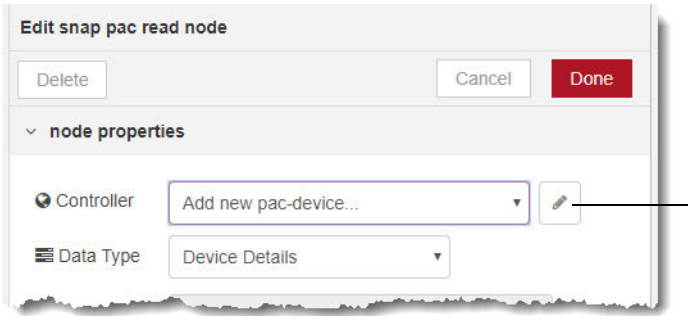
2. In Node-RED, configure an inject node to fire every 15 seconds.



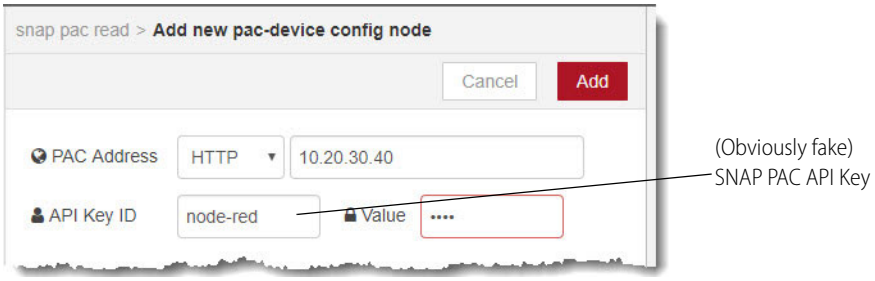
3. Drag in a snap pac read node and wire it to the inject node. (For instructions to install the snap pac nodes in Node-RED, see the Opto 22 Developer Portal website.)

4. Next, you'll need to add your SNAP PAC API key to the snap pac read node. If you don't have an API key, follow the instructions in the Opto 22 Developer Portal (http://developer.opto22.com/nodered/pac/getting-started/controller-configuration/) to configure one.

   a. Double-click the snap pac read node. The Edit snap pac read node dialog box is displayed.

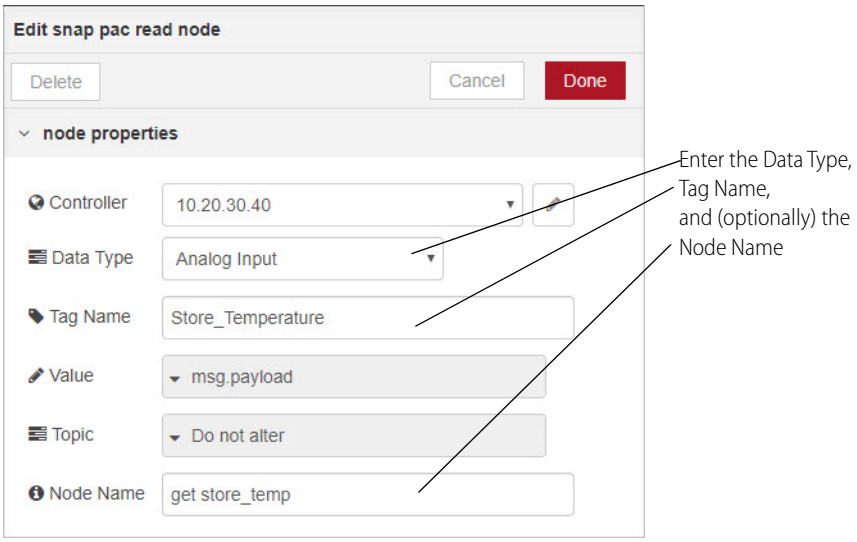


   b. Click the Edit icon to open the Add new pac-device config node dialog box.
   - In the PAC Address field, enter the IP address of your controller.
     Change the address scheme to HTTP if your PAC isn't configured for SSL/TLS.
   - In the API Key ID field, enter your SNAP PAC API key (user name).
   - In the Value field, enter the value (password) for the API key.



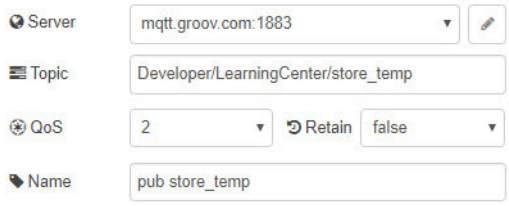
   c. Click Add. The Edit snap pac read node dialog box is displayed again.

Configure the Store_Temperature tag in the Edit snap pac read node dialog box.



5. Now, wire an mqtt output node to the flow.

6. Configure the mqtt output node to publish the data to the Developer/LearningCenter/store_temp topic on mqtt.groov.com. The data will be published as "store_temp."



7. Deploy the flow, and watch the message stream coming off the wild card (#) subscription to see the temperature data appear every 15 seconds.

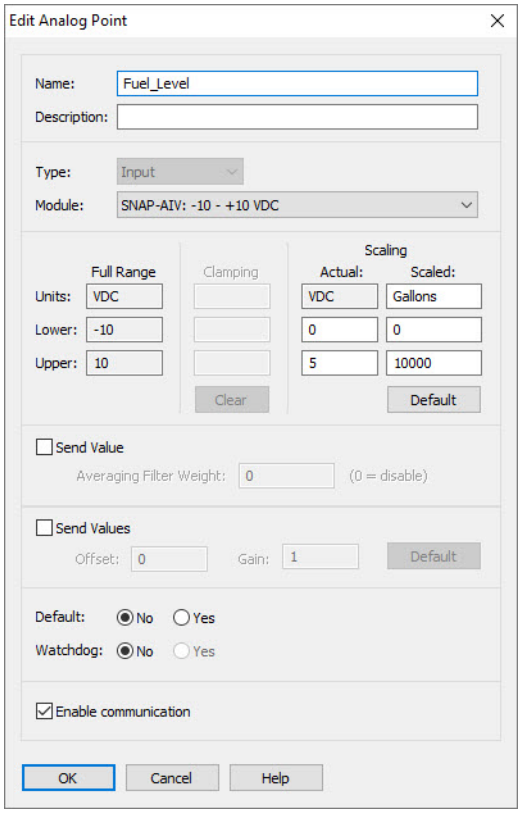
Now, everyone subscribing to the mqtt.groov.com broker can read your data.

PUBLISHING CONDITIONALLY

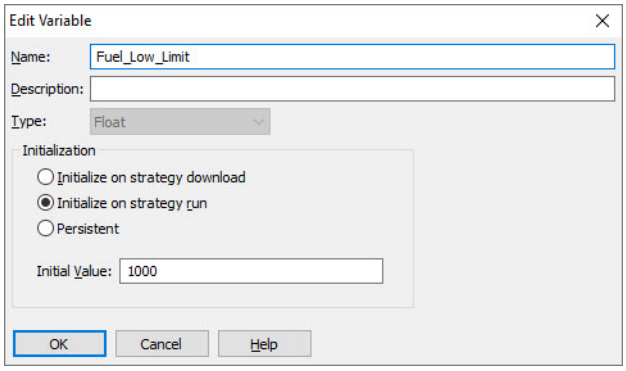
You now have data that is published only when it is updated (testbool) and data that is updated every 15 seconds (store_temp). Next, you'll publish data only when certain conditions are met.

Want the clipboard text to import these flows? Click here.

The Fuel_Level analog input point in the Convenience Store is configured with lower and upper ranges.

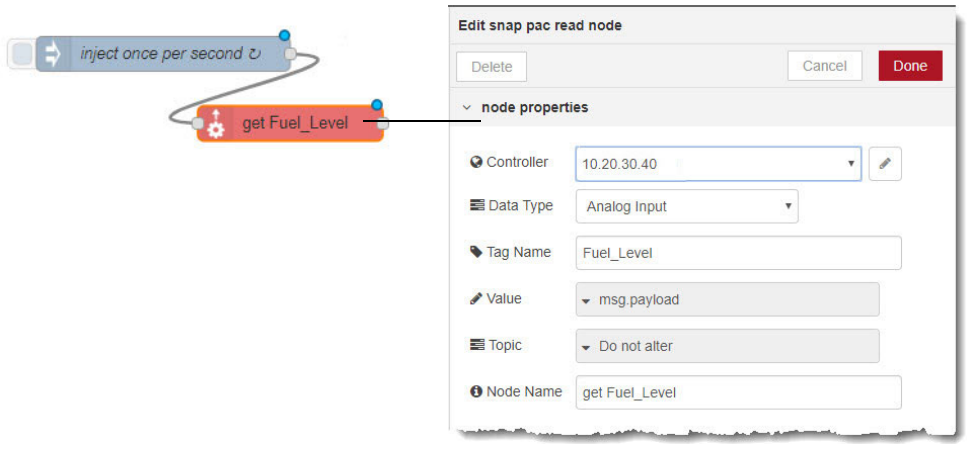


There's also a 32-bit integer tag, Fuel_Low_Limit, that sets the fuel level threshold for an alarm.

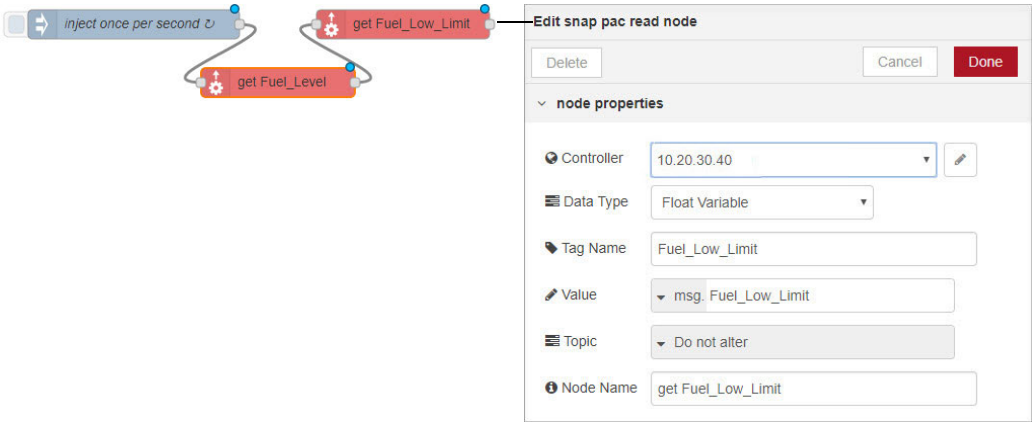


You're going to use both of these tags in Node-RED.

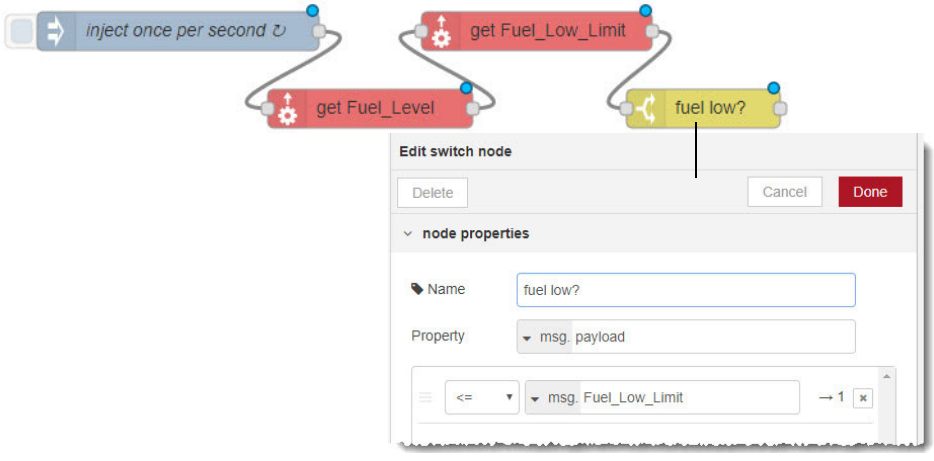
1. In Node-RED, create a flow that reads the status of Fuel_Level once a second. (In a real application, once a second is probably too often—but it'll make testing your results easier.)



2. Add to the flow another snap pac read node to get the value of Fuel_Low_Limit, and store it in a new property: msg.Fuel_Low_Limit. This prevents Fuel_Level payload from being overwritten (since you also want to publish Fuel_Level).



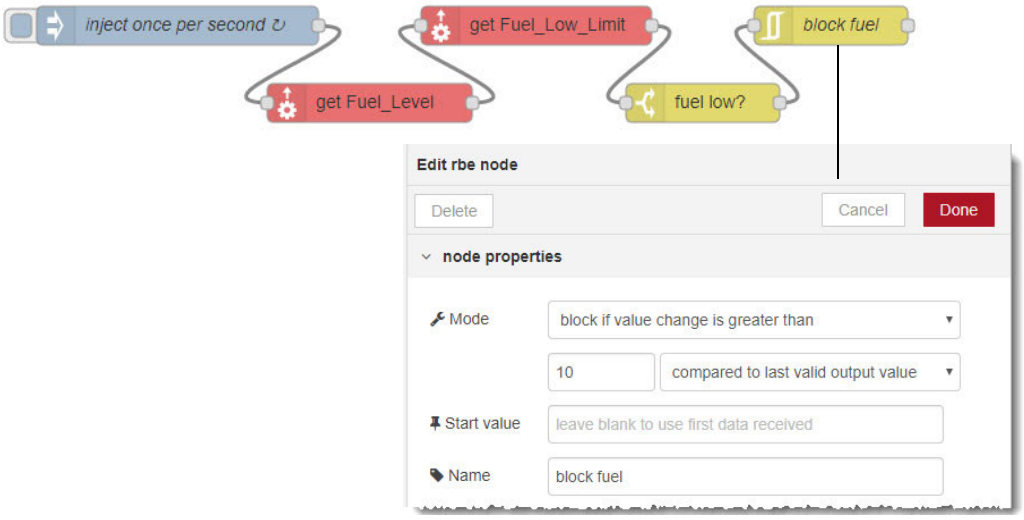
3. Here's where you use Fuel_Low_Limit to determine whether or not Fuel_Level should be published: add a switch node to the flow, and set up the switch node to let the signal through if property msg.payload is ≤ msg.Fuel_Low_Limit. (The switch node is listed with the function nodes.)



TIP: You're free to add as many conditions as you like; for example, publishing the fuel level under one topic when it's full, under a different topic when it's low, and under yet another topic when it's critically low.

However, in this activity, there's only one flow path, so there's only one output for publishing messages when the fuel is below the limit set in PAC Control.

4. To prevent a message from being published every second when fuel is low, add an rbe node to let the signal through every time the amount of gallons changes by 10 or more.



5. Finally, wire an mqtt output node to the end of the flow, and publish the message payload to the Developer/LearningCenter/LOWFuelLevel topic.

Then, deploy the flow.

Well done! Your flow injects every second to get the fuel level and lower limit, compares the two values, and publishes the fuel level to mqtt.groov.com when it dips below the lower limit. After that, it publishes again every 10 gallon change.

For more tips, help, and code samples, visit the Opto 22 Developer Portal at http://developer.opto22.com.

SOLUTIONS

This section contains text for the Node-RED flows in this activity.[1] Here are the changes you'll need to make when using these solutions:

- In Node-RED, you'll need to add your own *groov* View API key (or SNAP PAC API key, in the Publishing activity), and make any other changes required for your environment (for example, configuring SSL certificates if you're using SSL/TLS or running *groov* View and Node-RED on different devices.)
- For the *groov* View gadgets to work, you'll also need to add the MQTTdata Data Store (click here for instructions) and the tags used in the activity.
- The SNAP PAC Publishing activity uses the SNAP PAC Learning Center and convenience store strategy. You can perform this activity using your own SNAP PAC R- or S-series controller—you just may need to reconfigure the strategy to work with your controller and I/O modules.

"Pull data from MQTT" on page 2

```
[{"id":"1dff4582.4c83ea","type":"mqtt
in","z":"fd9a904f.7bc27","name":"subscribe to all
data","topic":"#","qos":"2","broker":"5b328761.fb9ba8","x":90,"y":20,"wir
es":[["7b8d94d1.ac322c"]]},{"id":"7b8d94d1.ac322c","type":"debug","z":"fd
9a904f.7bc27","name":"","active":false,"console":"false","complete":"fals
e","x":290,"y":20,"wires":[]},{"id":"7efa21d3.32614","type":"mqtt
in","z":"fd9a904f.7bc27","name":"get
machine_parts_count","topic":"machine_parts_count","qos":"2","broker":"5b
328761.fb9ba8","x":110,"y":100,"wires":[["f0ce202f.61528","977d3e44.00558
"]]},{"id":"f0ce202f.61528","type":"groov-write-
ds","z":"fd9a904f.7bc27","dataStore":"62470a5e.901ad4","tagName":"machine
_parts_count","tableStartIndex":"","value":"","valueType":"msg.payload","
name":"","x":370,"y":100,"wires":[[]]},{"id":"977d3e44.00558","type":"deb
ug","z":"fd9a904f.7bc27","name":"","active":false,"console":"false","comp
lete":"false","x":340,"y":140,"wires":[]},{"id":"5b328761.fb9ba8","type":
"mqtt-
broker","z":"","broker":"mqtt.groov.com","port":"1883","clientid":"","use
tls":false,"compatmode":true,"keepalive":"60","cleansession":true,"willTo
pic":"","willQos":"0","willPayload":"","birthTopic":"","birthQos":"0","bi
rthPayload":""},{"id":"62470a5e.901ad4","type":"groov-data-
store","z":"","project":"a8a1846b.9a93c8","dsName":"MQTTdata"},{"id":"a8a
1846b.9a93c8","type":"groov-project","z":"","address":"localhost"}]
```

"Convert a JSON-formatted string, and then display its values" on page 10

```
[{"id":"7d4da2bc.35cfcc","type":"mqtt
in","z":"d20e748a.ecc998","name":"get
earthquakes","topic":"earthquakes","qos":"2","broker":"e2ba5103.02bcf","x
":140,"y":60,"wires":[["d37db406.5de698"]]},{"id":"56818d07.a17864","type
":"debug","z":"d20e748a.ecc998","name":"","active":true,"console":"false"
,"complete":"payload","x":530,"y":60,"wires":[]},{"id":"d37db406.5de698",
"type":"json","z":"d20e748a.ecc998","name":"","x":330,"y":60,"wires":[["5
6818d07.a17864","29b701aa.b461ae","e62714ee.fec2b8","b405eceb.c7966"]]},{
"id":"e62714ee.fec2b8","type":"groov-write-
ds","z":"d20e748a.ecc998","dataStore":"91fb4f0a.47c79","tagName":"mag","t
```

1. To import a Node-RED flow:
   a. Copy the text.
   b. In Node-RED, click the menu icon (☰), and then click Import > Clipboard.
   c. Paste the text in the text field, and then click Import to create the flow.

```
ableStartIndex":"","value":"payload[0].mag","valueType":"msg","name":"wri
te
mag","x":520,"y":140,"wires":[[]]},{"id":"29b701aa.b461ae","type":"groov-
write-
ds","z":"d20e748a.ecc998","dataStore":"91fb4f0a.47c79","tagName":"time","
tableStartIndex":"","value":"payload[0].time","valueType":"msg","name":"w
rite
time","x":520,"y":100,"wires":[[]]},{"id":"b405eceb.c7966","type":"groov-
write-
ds","z":"d20e748a.ecc998","dataStore":"91fb4f0a.47c79","tagName":"place",
"tableStartIndex":"","value":"payload[0].place","valueType":"msg","name":
"write
place","x":530,"y":180,"wires":[[]]},{"id":"e2ba5103.02bcf","type":"mqtt-
broker","z":"","broker":"mqtt.groov.com","port":"1883","clientid":"","use
tls":false,"compatmode":true,"keepalive":"60","cleansession":true,"willTo
pic":"","willQos":"0","willPayload":"","birthTopic":"","birthQos":"0","bi
rthPayload":""},{"id":"91fb4f0a.47c79","type":"groov-data-
store","z":"","project":"aebb57fc.2701f8","dsName":"MQTTdata"},{"id":"aeb
b57fc.2701f8","type":"groov-project","z":"","address":"localhost"}]
```


"Use a function node to convert strings" on page 13

Flow 1


```
[{"id":"d128ff90.31f52","type":"mqtt
in","z":"9d74f6c8.3809f8","name":"sub
snmp","topic":"snmp","qos":"2","broker":"5e45a2a0.a6ca7c","x":140,"y":340
,"wires":[["13eac35b.51a4bd"]]},{"id":"13eac35b.51a4bd","type":"function"
,"z":"9d74f6c8.3809f8","name":"extract value","func":"var parts =
msg.payload.split(\",\");     // break the string into parts\nvar num =
parts[2].substr(8);           // grab characters after
\"value\":\nmsg.payload = parseInt(num);           // convert the value
to an integer\nreturn msg;                          // return integer
value in
payload","outputs":1,"noerr":0,"x":310,"y":340,"wires":[["e13a3693.0d72e8
","ef0239ba.28f1e8"]]},{"id":"e13a3693.0d72e8","type":"groov-write-
ds","z":"9d74f6c8.3809f8","dataStore":"8c38dd82.3f9f6","tagName":"PAC_ses
sions","tableStartIndex":"","value":"","valueType":"msg.payload","name":"
write
PAC_sessions","x":530,"y":340,"wires":[[]]},{"id":"ef0239ba.28f1e8","type
":"debug","z":"9d74f6c8.3809f8","name":"","active":false,"console":"false
","complete":"false","x":510,"y":300,"wires":[]},{"id":"5e45a2a0.a6ca7c",
"type":"mqtt-
broker","z":"","broker":"mqtt.groov.com","port":"1883","clientid":"","use
tls":false,"compatmode":true,"keepalive":"60","cleansession":true,"willTo
pic":"","willQos":"0","willPayload":"","birthTopic":"","birthQos":"0","bi
rthPayload":""},{"id":"8c38dd82.3f9f6","type":"groov-data-
store","z":0,"project":"cb2b7005.0a753","dsName":"MQTTdata"},{"id":"cb2b7
005.0a753","type":"groov-project","z":0,"address":"localhost"}]
```


Flow 2


```
[{"id":"d128ff90.31f52","type":"mqtt
in","z":"9d74f6c8.3809f8","name":"sub
snmp","topic":"snmp","qos":"2","broker":"5e45a2a0.a6ca7c","x":140,"y":340
,"wires":[["13eac35b.51a4bd"]]},{"id":"13eac35b.51a4bd","type":"function"
,"z":"9d74f6c8.3809f8","name":"extract value","func":"var str =
msg.payload.substr(msg.payload.indexOf(\"value\"));\nmsg.payload =
```

```
Number(str.substring(str.indexOf(\":\")+1,str.indexOf(\",\")));\nreturn
msg;","outputs":1,"noerr":0,"x":310,"y":340,"wires":[["e13a3693.0d72e8","
ef0239ba.28f1e8"]]},{"id":"e13a3693.0d72e8","type":"groov-write-
ds","z":"9d74f6c8.3809f8","dataStore":"8c38dd82.3f9f6","tagName":"PAC_ses
sions","tableStartIndex":"","value":"","valueType":"msg.payload","name":"
write
PAC_sessions","x":530,"y":340,"wires":[[]]},{"id":"ef0239ba.28f1e8","type
":"debug","z":"9d74f6c8.3809f8","name":"","active":false,"console":"false
","complete":"false","x":510,"y":300,"wires":[]},{"id":"5e45a2a0.a6ca7c",
"type":"mqtt-
broker","z":"","broker":"mqtt.groov.com","port":"1883","clientid":"","use
tls":false,"compatmode":true,"keepalive":"60","cleansession":true,"willTo
pic":"","willQos":"0","willPayload":"","birthTopic":"","birthQos":"0","bi
rthPayload":""},{"id":"8c38dd82.3f9f6","type":"groov-data-
store","z":0,"project":"cb2b7005.0a753","dsName":"MQTTdata"},{"id":"cb2b7
005.0a753","type":"groov-project","z":0,"address":"localhost"}]
```


Flow 3


```
[{"id":"ce3b7e36.3f426","type":"mqtt
in","z":"9d74f6c8.3809f8","name":"","topic":"machine_parts_count","qos":"
2","broker":"5e45a2a0.a6ca7c","x":180,"y":120,"wires":[["a10049e2.7d8b38"
,"2c5c6c12.58d044"]]},{"id":"a10049e2.7d8b38","type":"debug","z":"9d74f6c
8.3809f8","name":"","active":false,"console":"false","complete":"false","
x":410,"y":160,"wires":[]},{"id":"2c5c6c12.58d044","type":"groov-write-
ds","z":"9d74f6c8.3809f8","dataStore":"8c38dd82.3f9f6","tagName":"machine
_parts_count","tableStartIndex":"","value":"","valueType":"msg.payload","
name":"","x":440,"y":120,"wires":[[]]},{"id":"e9e0bf60.26eee","type":"mqt
t in","z":"9d74f6c8.3809f8","name":"get
earthquakes","topic":"earthquakes","qos":"2","broker":"5e45a2a0.a6ca7c","
x":160,"y":260,"wires":[["4391b159.d00c5"]]},{"id":"9e76eff4.b9c51","type
":"debug","z":"9d74f6c8.3809f8","name":"","active":true,"console":"false"
,"complete":"payload","x":650,"y":260,"wires":[]},{"id":"4391b159.d00c5",
"type":"json","z":"9d74f6c8.3809f8","name":"","x":350,"y":260,"wires":[["
9e76eff4.b9c51","ec9310f3.bae97","4a324f23.fd11e","bca77e47.b80fe"]]},{"i
d":"4a324f23.fd11e","type":"groov-write-
ds","z":"9d74f6c8.3809f8","dataStore":"8c38dd82.3f9f6","tagName":"mag","t
ableStartIndex":"","value":"payload[0].mag","valueType":"msg","name":"wri
te
mag","x":640,"y":180,"wires":[[]]},{"id":"ec9310f3.bae97","type":"groov-
write-
ds","z":"9d74f6c8.3809f8","dataStore":"8c38dd82.3f9f6","tagName":"time","
tableStartIndex":"","value":"payload[0].time","valueType":"msg","name":"w
rite
time","x":640,"y":140,"wires":[[]]},{"id":"bca77e47.b80fe","type":"groov-
write-
ds","z":"9d74f6c8.3809f8","dataStore":"8c38dd82.3f9f6","tagName":"place",
"tableStartIndex":"","value":"payload[0].place","valueType":"msg","name":
"write
place","x":650,"y":220,"wires":[[]]},{"id":"78a9b317.448ddc","type":"mqtt
in","z":"9d74f6c8.3809f8","name":"subscribe to all
data","topic":"#","qos":"2","broker":"5e45a2a0.a6ca7c","x":170,"y":60,"wi
res":[["e81b8e62.e6b56"]]},{"id":"318261dd.3eb11e","type":"debug","z":"9d
74f6c8.3809f8","name":"","active":false,"console":"false","complete":"pay
load","x":510,"y":60,"wires":[]},{"id":"e81b8e62.e6b56","type":"json","z"
:"9d74f6c8.3809f8","name":"","x":350,"y":60,"wires":[["318261dd.3eb11e"]]
},{"id":"2bc9c45b.204dec","type":"mqtt
in","z":"9d74f6c8.3809f8","name":"sub
```

snmp","topic":"snmp","qos":"2","broker":"5e45a2a0.a6ca7c","x":140,"y":360
,"wires":[["1e87e7a6.e2f9a8"]]},{"id":"1e87e7a6.e2f9a8","type":"function"
,"z":"9d74f6c8.3809f8","name":"extract value","func":"var str =
msg.payload.substr(msg.payload.indexOf(\"value\"));\nmsg.payload =
Number(str.substring(str.indexOf(\":\")+1,str.indexOf(\",\")));\nreturn
msg;","outputs":1,"noerr":0,"x":310,"y":360,"wires":[["c91f335d.aaf65","2
958ff08.6f70f"]]},{"id":"c91f335d.aaf65","type":"groov-write-
ds","z":"9d74f6c8.3809f8","dataStore":"8c38dd82.3f9f6","tagName":"PAC_ses
sions","tableStartIndex":"","value":"","valueType":"msg.payload","name":"
write
PAC_sessions","x":530,"y":360,"wires":[[]]},{"id":"2958ff08.6f70f","type"
:"debug","z":"9d74f6c8.3809f8","name":"","active":false,"console":"false"
,"complete":"payload","x":510,"y":320,"wires":[]},{"id":"5e45a2a0.a6ca7c"
,"type":"mqtt-
broker","z":"","broker":"mqtt.groov.com","port":"1883","clientid":"","use
tls":false,"compatmode":true,"keepalive":"60","cleansession":true,"willTo
pic":"","willQos":"0","willPayload":"","birthTopic":"","birthQos":"0","bi
rthPayload":""},{"id":"8c38dd82.3f9f6","type":"groov-data-
store","z":0,"project":"cb2b7005.0a753","dsName":"MQTTdata"},{"id":"cb2b7
005.0a753","type":"groov-project","z":0,"address":"localhost"}]

“Publish data from groov View to MQTT” on page 17

[{"id":"deb39116.8940c","type":"groov-read-
ds","z":"9d74f6c8.3809f8","dataStore":"8c38dd82.3f9f6","tagName":"testboo
l","tableStartIndex":"","tableLength":"","value":"","valueType":"msg.payl
oad","topic":"","topicType":"none","name":"get
bool","x":320,"y":960,"wires":[["7fd9617e.aebef"]]},{"id":"7fd9617e.aebef
","type":"rbe","z":"9d74f6c8.3809f8","name":"block unless
toggled","func":"rbe","gap":"","start":"","inout":"out","x":420,"y":1020,
"wires":[["4aa83874.dc9c48","f2274560.f2c368"]]},{"id":"f2274560.f2c368",
"type":"mqtt out","z":"9d74f6c8.3809f8","name":"pub
testbool","topic":"Developer/
testbool","qos":"2","retain":"false","broker":"5e45a2a0.a6ca7c","x":650,"
y":1020,"wires":[]},{"id":"4aa83874.dc9c48","type":"debug","z":"9d74f6c8.
3809f8","name":"","active":false,"console":"false","complete":"payload","
x":650,"y":960,"wires":[]},{"id":"75f921af.cc82c","type":"inject","z":"9d
74f6c8.3809f8","name":"inject once per
second","topic":"","payload":"","payloadType":"date","repeat":"1","cronta
b":"","once":true,"x":210,"y":900,"wires":[["deb39116.8940c"]]},{"id":"8c
38dd82.3f9f6","type":"groov-data-
store","z":0,"project":"cb2b7005.0a753","dsName":"MQTTdata"},{"id":"5e45a
2a0.a6ca7c","type":"mqtt-
broker","z":"","broker":"mqtt.groov.com","port":"1883","clientid":"","use
tls":false,"compatmode":true,"keepalive":"60","cleansession":true,"willTo
pic":"","willQos":"0","willPayload":"","birthTopic":"","birthQos":"0","bi
rthPayload":""},{"id":"cb2b7005.0a753","type":"groov-
project","z":0,"address":"localhost"}]

“Publish data from a PAC controller to MQTT” on page 21

[{"id":"152dfef9.0e5b51","type":"inject","z":"9d74f6c8.3809f8","name":"in
ject 4 times per
minute","topic":"","payload":"","payloadType":"date","repeat":"15","cront
ab":"","once":false,"x":190,"y":540,"wires":[["138070f3.dfd2df"]]},{"id":
"138070f3.dfd2df","type":"pac-
read","z":"9d74f6c8.3809f8","device":"ec4a8d40.93fa3","dataType":"ana-

input","tagName":"Store_Temperature","tableStartIndex":"","tableLength":"","value":"","valueType":"msg.payload","topic":"","topicType":"none","name":"get
store_temp","x":420,"y":540,"wires":[["5b0864dc.17a7fc"]]},{"id":"5b0864dc.17a7fc","type":"mqtt out","z":"9d74f6c8.3809f8","name":"pub
store_temp","topic":"Developer/LearningCenter/
store_temp","qos":"2","retain":"false","broker":"5e45a2a0.a6ca7c","x":640,"y":540,"wires":[]},{"id":"ec4a8d40.93fa3","type":"pac-
device","z":"","address":"10.192.58.11","protocol":"http"},{"id":"5e45a2a0.a6ca7c","type":"mqtt-
broker","z":"","broker":"mqtt.groov.com","port":"1883","clientid":"","usetls":false,"compatmode":true,"keepalive":"60","cleansession":true,"willTopic":"","willQos":"0","willPayload":"","birthTopic":"","birthQos":"0","birthPayload":""}]

"Publishing conditionally" on page 24

[{"id":"cd71685f.302198","type":"inject","z":"9d74f6c8.3809f8","name":"inject once per
second","topic":"","payload":"","payloadType":"date","repeat":"1","cronta
b":"","once":true,"x":190,"y":660,"wires":[["72c3b489.06f06c"]]},{"id":"72c3b489.06f06c","type":"pac-
read","z":"9d74f6c8.3809f8","device":"ec4a8d40.93fa3","dataType":"ana-
input","tagName":"Fuel_Level","tableStartIndex":"","tableLength":"","value":"","valueType":"msg.payload","topic":"","topicType":"none","name":"get
Fuel_Level","x":220,"y":720,"wires":[["1437ed6e.fedf33"]]},{"id":"5faa4208.1db3ec","type":"mqtt out","z":"9d74f6c8.3809f8","name":"send
FuelLevel","topic":"Developer/LearningCenter/
LOWFuelLevel","qos":"2","retain":"false","broker":"5e45a2a0.a6ca7c","x":520,"y":780,"wires":[]},{"id":"d1f3a0e0.3d901","type":"rbe","z":"9d74f6c8.3809f8","name":"block
fuel","func":"deadband","gap":"10","start":"","inout":"out","x":460,"y":720,"wires":[["5faa4208.1db3ec"]]},{"id":"74695f15.f81f3","type":"switch","z":"9d74f6c8.3809f8","name":"fuel
low?","property":"payload","propertyType":"msg","rules":[{"t":"lte","v":"Fuel_Low_Limit","vt":"msg"}],"checkall":"true","outputs":1,"x":420,"y":660,"wires":[["d1f3a0e0.3d901"]]},{"id":"1437ed6e.fedf33","type":"pac-
read","z":"9d74f6c8.3809f8","device":"ec4a8d40.93fa3","dataType":"int32-
variable","tagName":"Fuel_Low_Limit","tableStartIndex":"","tableLength":"","value":"Fuel_Low_Limit","valueType":"msg","topic":"","topicType":"none","name":"get
Fuel_Low_Limit","x":230,"y":780,"wires":[["74695f15.f81f3"]]},{"id":"ec4a8d40.93fa3","type":"pac-
device","z":"","address":"10.192.58.11","protocol":"http"},{"id":"5e45a2a0.a6ca7c","type":"mqtt-
broker","z":"","broker":"mqtt.groov.com","port":"1883","clientid":"","usetls":false,"compatmode":true,"keepalive":"60","cleansession":true,"willTopic":"","willQos":"0","willPayload":"","birthTopic":"","birthQos":"0","birthPayload":""}]