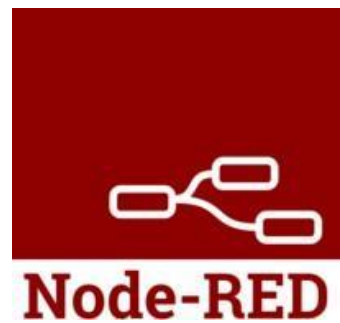


Laboratory 1



Node-RED introduction

On the left side of the screen we are going to have the node palette. We can hover over a node to see what its purpose is.

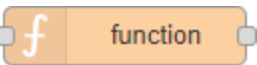
The screenshot shows the Node-RED node palette on the left side of the interface. The palette is titled 'Node-RED' and has a search bar labeled 'filter nodes'. Below the search bar, there is a section labeled 'common' which contains a list of nodes: 'inject', 'debug', 'complete', 'catch', 'status', 'link in', 'link out', and 'comment'. The 'inject' node is highlighted with an orange border. A tooltip for the 'inject' node is shown on the right, describing its function: 'Injects a message into a flow either manually or at regular intervals. The message payload can be a variety of types, including strings, JavaScript objects or the current time.' Below the 'inject' node, the 'debug' node is also highlighted with an orange border. A tooltip for the 'debug' node is shown on the right, describing its function: '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.'

The *inject* and *debug* nodes are the classic input-output nodes.

You can see whether a node is of the type input / output / processing (input-output) by the small squares around the nodes.

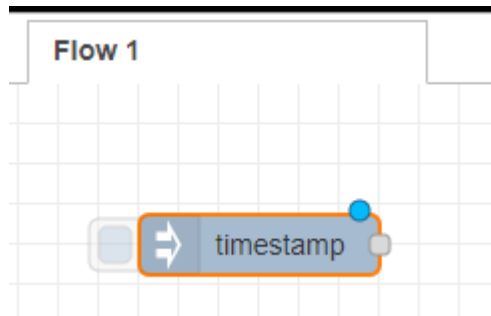
Input node example:

Output node example: 

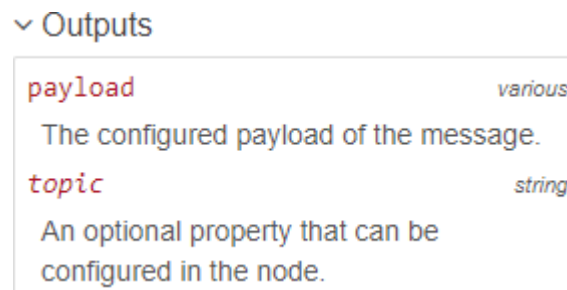
Processing (Input-output) node example: 

Hello Node-RED

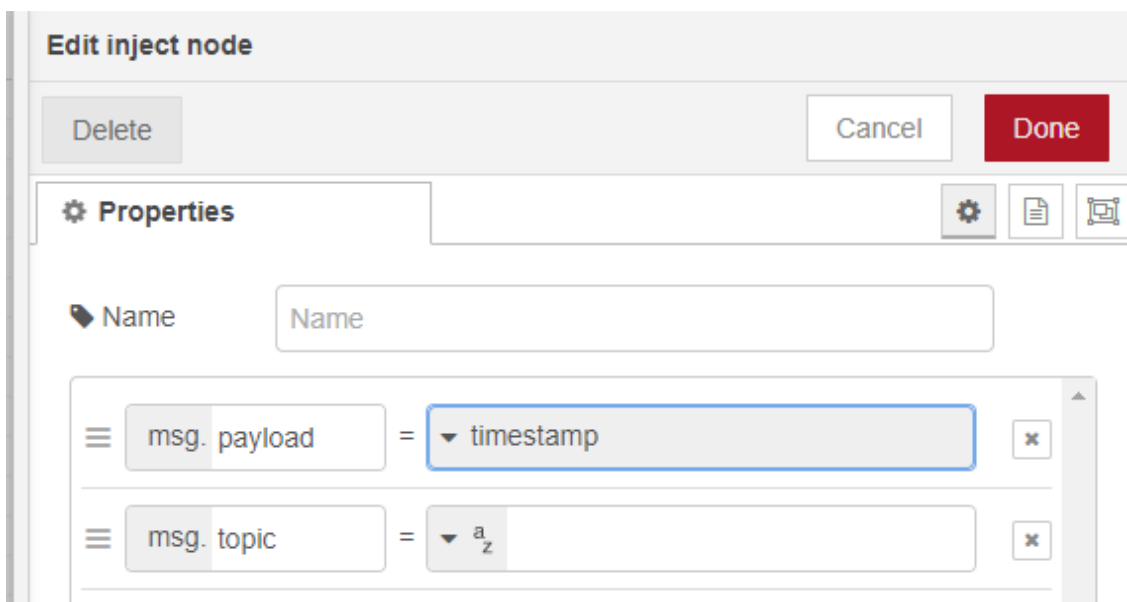
First thing we are going to do is to drag and drop an inject node into the workspace. The **blue dot** indicates that the **node** hasn't been deployed since it was last changed.



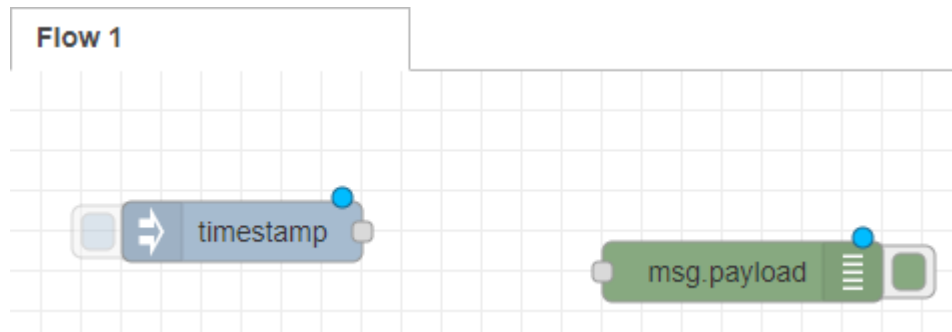
As outputs we are going to have:



To edit a node you double click it, and a new page will appear:

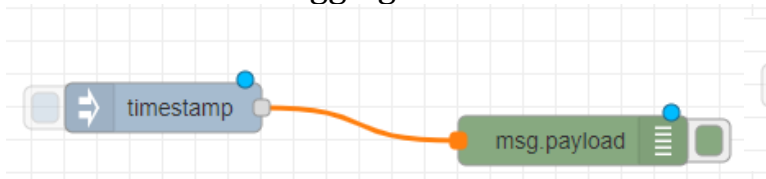


For now, leave the nodes unchanged, and click on done. Next up, add a **debug** node. The flow editor should now look like this:

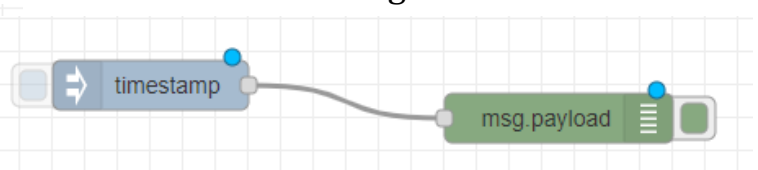


As previously said, judging by the gray nodes we can determine whether a node is of type input or output. To connect two nodes we are going to wire them together. To do this, we click and hold on the gray node and we connect it to the other one. When the two nodes are connected, the wiring between them turns gray.

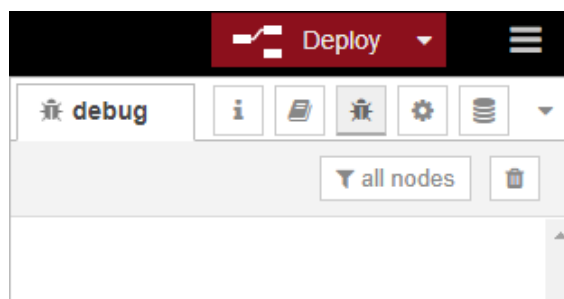
While dragging the wire



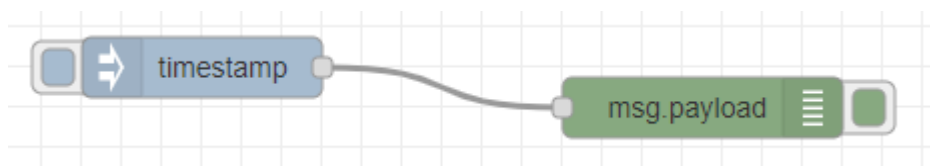
After releasing the wire

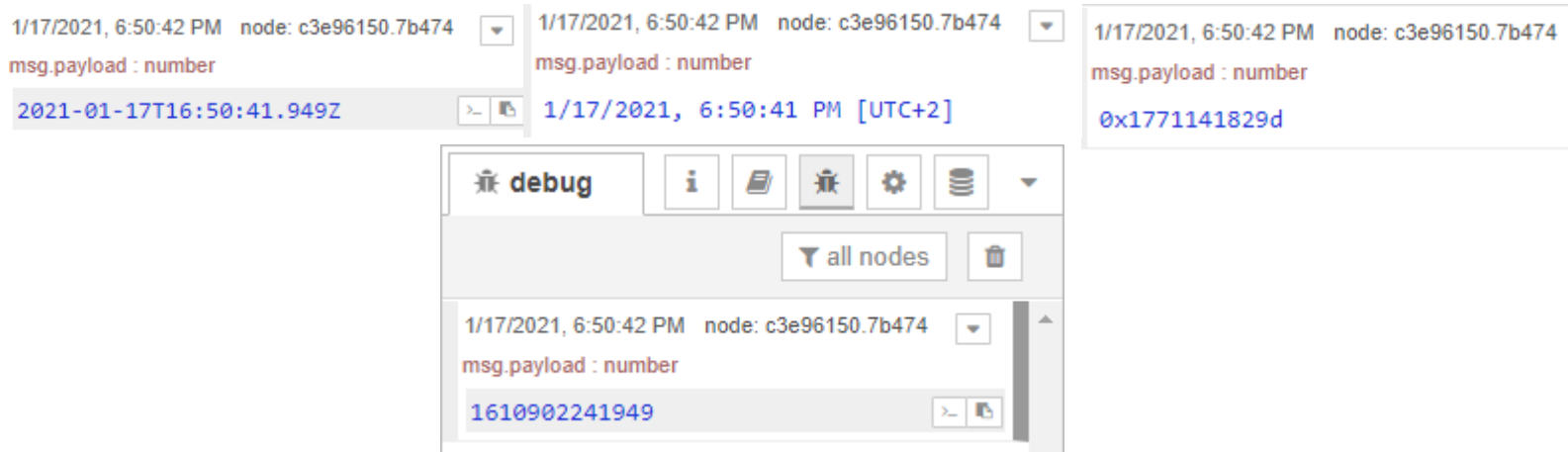


To see the output, we have to go to the **debug mode**. We can find the debug mode on the top-right of the editor, under Deploy.



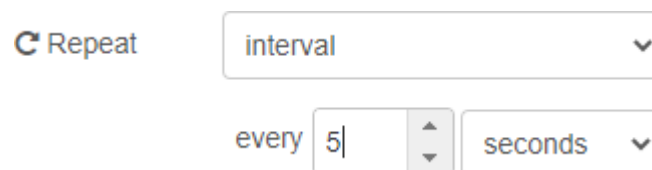
If the Deploy button is red, it means that you have changes that have not yet been deployed. After clicking on it, it turns gray, and the nodes should no longer have blue dots on top of them.





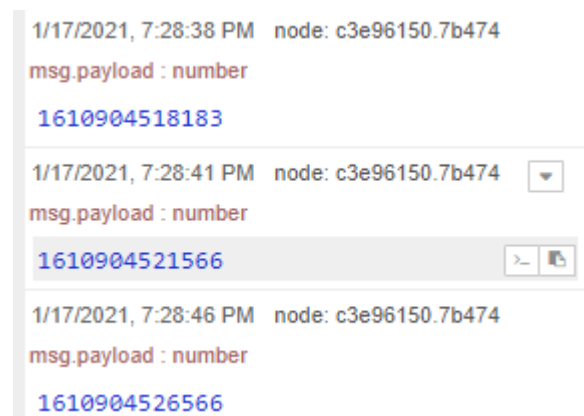
That number is recognized by node-red as a timestamp. This is the UNIX time, which numbers the milliseconds elapsed since the 1st of January 1970. Since node-red recognizes it as a timestamp, you can click it once to display the date in **YearMonthDay** format, twice to see the date in **DateMonthYear** format, three times to see the hexadecimal value, and four times to go back to the initial value.

To make a node repeat its actions, go to the editing menu and select interval. Feel free to play with the numbers and modify the time it repeats its actions. For this example, I have chosen the following setup:



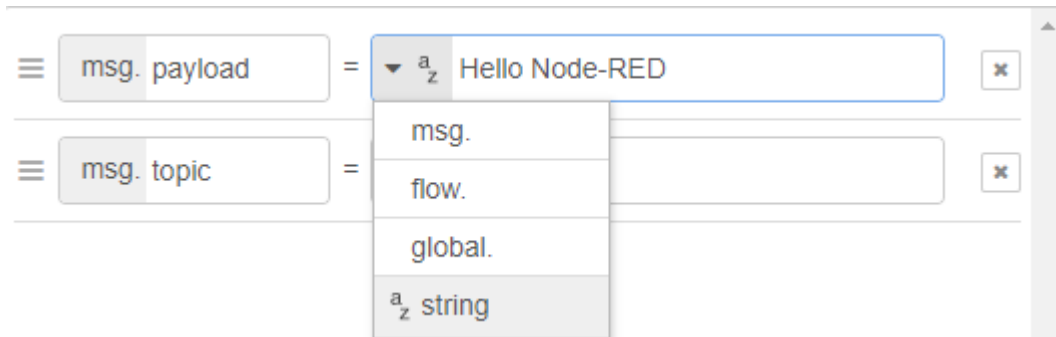
Click done, then deploy to make the changes live, and then run it again.

The output should look similar to this:



To stop it from repeating the action, go back to the editing menu, and put the repeat to **none**, then deploy again.

We can also change the message that the **inject** node is inserting into the flow. For this example, we are going to inject the message “Hello Node-RED”. To change it, double click the **inject** node (timestamp), and change the payload from timestamp to string.

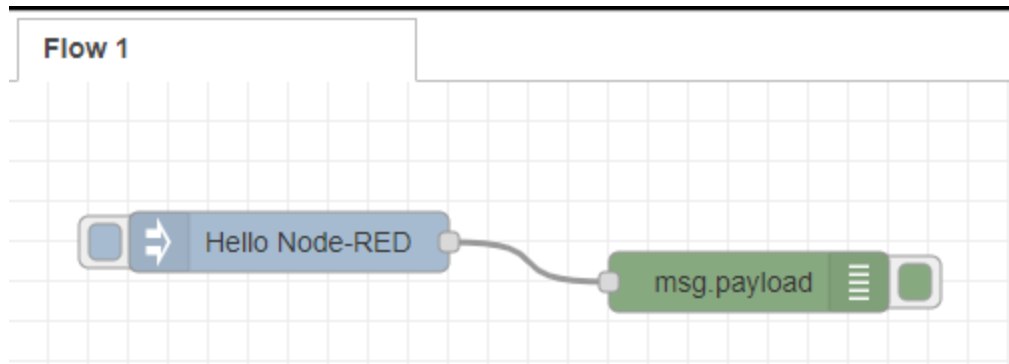


Save and deploy again. After running it again, the message you have previously injected should be outputted.

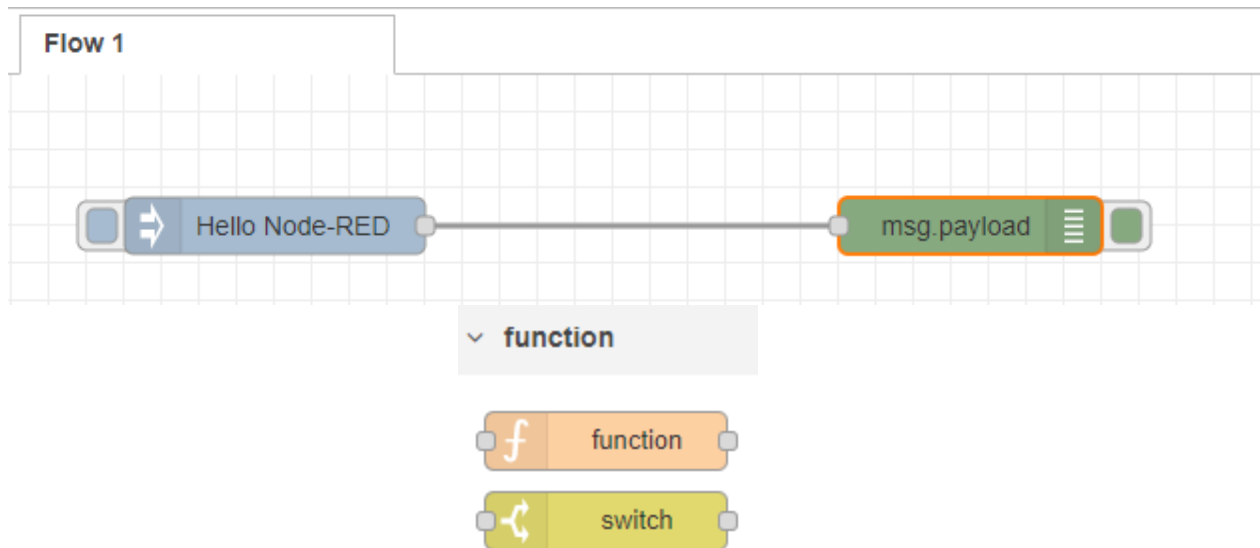
```
1/17/2021, 7:33:20 PM node: c3e96150.7b474
msg.payload : string[14]
"Hello Node-RED"
```

Applying functions

Let's continue the example above. Currently we have a flow that outputs Hello Node-RED.

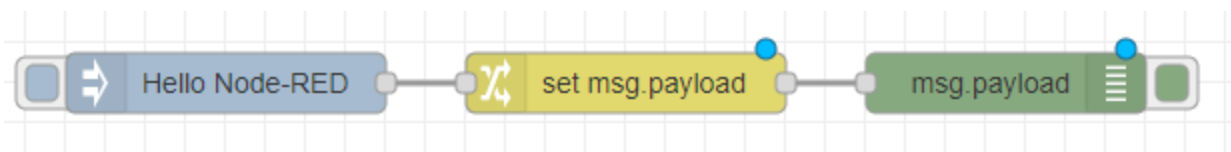


We can move nodes by dragging and dropping them to the new location. Move the **debug** node so that we have room for a **function** node in between the two nodes.

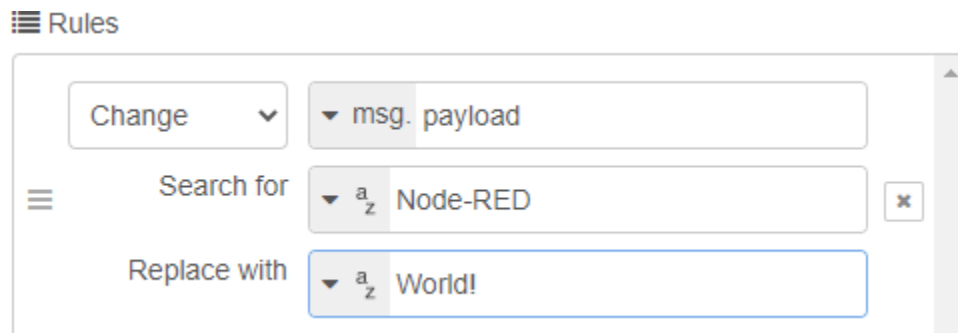


Let's apply the **change** node, to change the output of the flow. Drag and drop the node on the line between the two nodes.

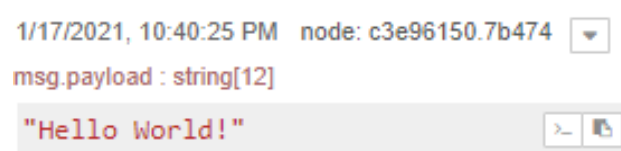
The new flow should look somewhat like this:



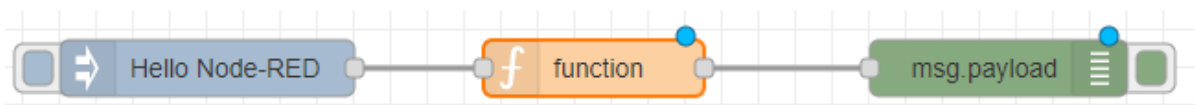
Now we have to edit the change node. Double click the node to open the editor. In the rules section select **Change**. Let's replace **Node-RED** with **World!**



After the changes are made, click on done and deploy. After running the program your output should now show **Hello World!**



Let's understand how the node **function** works. Delete the **change** node. When you are going to delete the node, you can see that the wire between the two remaining nodes has also disappeared. Connect them with a new wire, and then drag the **function** node between them.

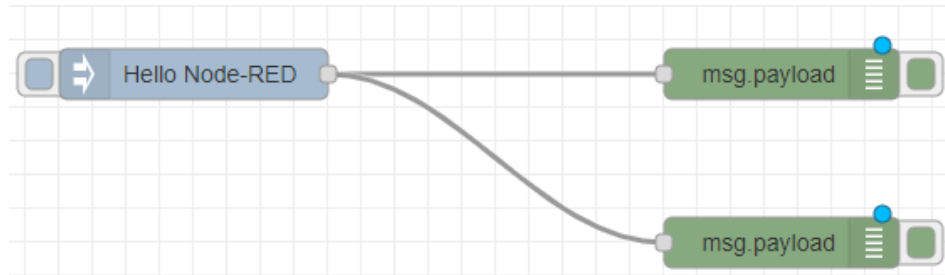


This **function** node is amongst the most powerful ones, since it is made up of a block of **javascript**, which the user can edit in any way they wish.

Let's edit the **function** node so that it changes the payload from **Node-RED** to **World!**, just as the previous example. To do this, we can use the following code:

```
var oldMessage = msg.payload;  
  
var newMessage = oldMessage.replace("Node-RED","World!");  
  
return {payload : newMessage};
```


Let's output more information about the JavaScript object that we are injecting. Delete the **function** node, insert another **debug** node and wire the debug nodes with the inject one. The new flow diagram should look similar to this:



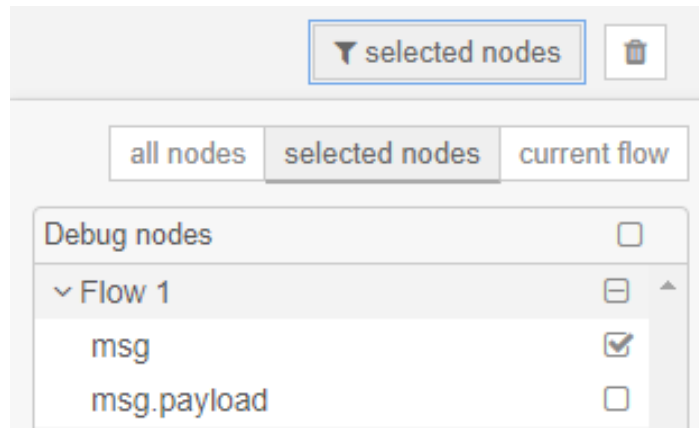
Let's edit one of the two nodes to output the entire message object. For this, change the output to "**complete msg object**".

The screenshot shows the Node-RED interface. On the left, the 'Output' tab is active, displaying a dropdown menu with the following options: 'complete msg object' (selected), 'msg.', 'complete msg object', and 'J: expression'. The main workspace shows a flow diagram with a 'Hello Node-RED' node connected to two 'msg.payload' nodes. The bottom right shows the debug console output for the selected node:

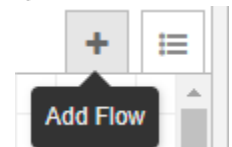
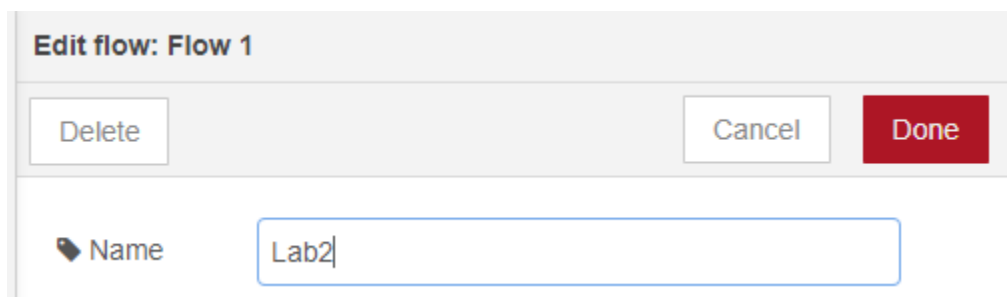
```
1/18/2021, 12:09:25 PM node: c3e96150.7b474
msg.payload: string[14]
"Hello Node-RED"

1/18/2021, 12:09:25 PM node: 6af3109e.a0038
msg: Object
  {
    _msgid: "61976f59.cf22e",
    payload: "Hello Node-RED",
    topic: ""
  }
```

We can also select the outputs of which nodes we want to see in the debug window. To do this, click on **all nodes** in the debug window, and on the selected nodes, deselect the ones you do not want to appear in the debug window. For this example, let's only output only the entire object.



Now we should only see the output from the node that we have previously selected. Let's give this flow a name and create a new one for the next application. Double click on **Flow1** and change the name to something suggestive for you. In my case, I chose Lab2. Click done and deploy, and your first flow has been saved.



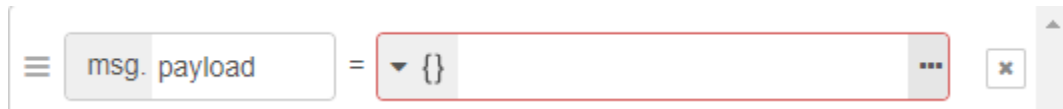
To create a new flow, simply click on the “+” on the top right.

Let's add an **inject** node. We can rename this node when editing it. Let's name this “Inject”. This time, let's send a JavaScript object. Let's learn how to write a JavaScript object. <https://jsonlint.com/>. This is a link towards a validator, you can write your object, and once it's done, you can check if it's valid. For this example, let's use the following object:

```
{
  "payload": "Akita Inu",
  "fur": "White",
  "origin": "Japan"
}
```

When we are going to write objects to use in Node-RED, we **must** have a **payload** parameter.

Going back to our **inject** node, change the **msg.payload** to JSON. Afterwards, open the editor by clicking on the "...".



We can paste our JSON object here, and also use the visual editor to check if everything appears as expected.

Edit JSON

Visual editor

Edit JSON

Visual editor

1 {
2 "payload": "Akita Inu",
3 "fur": "White",
4 "origin": "Japan"
5 }

1/18/2021, 12:36:10 PM node: 2b65c.39b21ab4
msg.payload : Object
▶ { payload: "Akita Inu", fur: "White", origin: "Japan" }

1/18/2021, 12:36:20 PM node: 2b654c.39b21ab4
msg.payload : Object
▶ { payload: "Akita Inu", fur: "White", origin: "Japan" }

1/18/2021, 12:36:30 PM node: 2b654c.39b21ab4
msg.payload : Object
▶ { payload: "Akita Inu", fur: "White", origin: "Japan" }

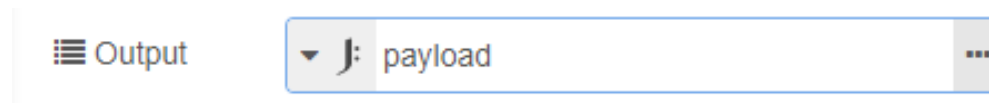
object

"payload" : "Akita Inu"
"fur" : "White"
"origin" : "Japan"

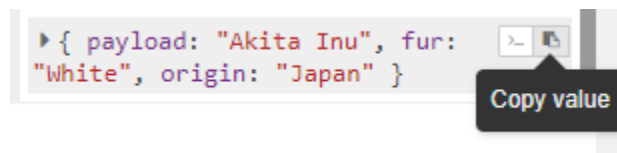
A screenshot showing the Node-RED editor with two side-by-side panels. The left panel is the 'Edit JSON' view, showing a JSON object with three properties: 'payload' (value: 'Akita Inu'), 'fur' (value: 'White'), and 'origin' (value: 'Japan'). The right panel is the 'Visual editor' view, showing the same JSON object in a tree structure. Below the JSON editors is a console log showing three entries, each representing a message object with the same payload. The console log shows the timestamp, node ID, and the message object. The visual editor also shows a tree view of the object with the same three properties.

Once you can see it is working, make it not repeat itself anymore, and deploy the changes to make them live.

For the next step, let's output only one property, let's say, the origin. For this, edit the **debug** node, and at **output**, select **expression**.



Click on the dots to open the expression editor. We can use this to test the results we get for different expressions, using different messages. Let's copy the object we have outputted so far. To do this, go to the debug window and select "Copy value".



Going back to the Expression editor, change the payload to our desired property, which is **payload.origin**. In the bottom side of the editor, select "Test", and paste the object in the example message section, **inside the payload object**. If you did everything correctly, the output should show Japan. (To sort the message, click on format JSON)



Now deploy the changes and test it by injecting the message. If you got an error saying "undefined", you did not put the object inside the payload. The

reason for which we have to do this is that we are sending a **payload** towards the debug node, and, for example, to access the payload from the JSON object, we have to output **payload.payload**.

More functionalities

Let's create a new flow in which we test some other functionalities of the function node.

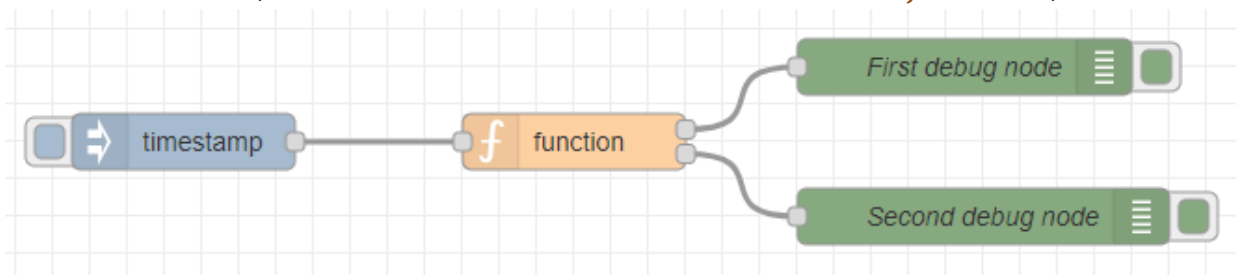
Multiple messages

A function can return multiple messages on an output by returning an array of messages within the returned array. We can edit this from the edit window of the **function** node:



When multiple messages are returned, subsequent nodes will receive the messages one at a time in the order they were returned.

For this, let's create a flow with four nodes: an **inject** node, a **function**



Edit the **function** node to contain the following:

```
var msg1 = { payload:"first from output 1" };  
var msg2 = { payload:"second from output 1" };  
var msg3 = { payload:"only one from output 2" };  
return [ [ msg1, msg2 ], msg3 ];
```

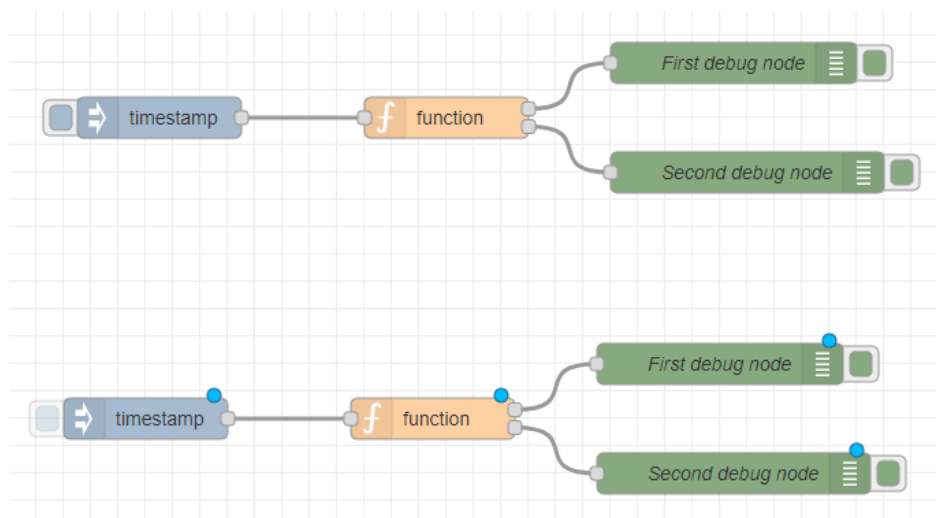
In this example, the first two messages will be sent to the first **debug** node, while the third message will be sent to the second **debug** node.

```
1/21/2021, 3:41:42 PM node: First debug node
msg.payload : string[19]
"first from output 1"

1/21/2021, 3:41:42 PM node: First debug node
msg.payload : string[20]
"second from output 1"

1/21/2021, 3:41:42 PM node: Second debug node
msg.payload : string[22]
"only one from output 2"
```

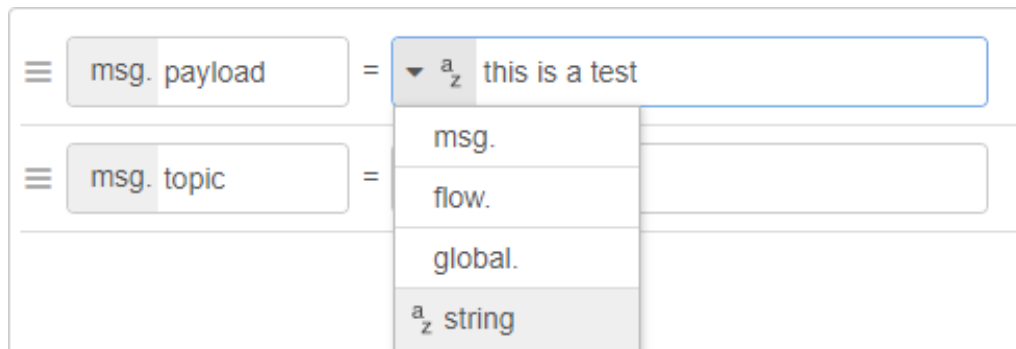
Copy the nodes we already have by selecting all and using the “Ctrl-C + Ctrl-V” that we are all so used with. Paste them below the current flow:



Now, let's change the function to send one word to each **debug** node.

```
var output = [];
var words = msg.payload.split(" ");
for (var w in words)
{
    output.push({payload:words[w]});
}
return [output];
```

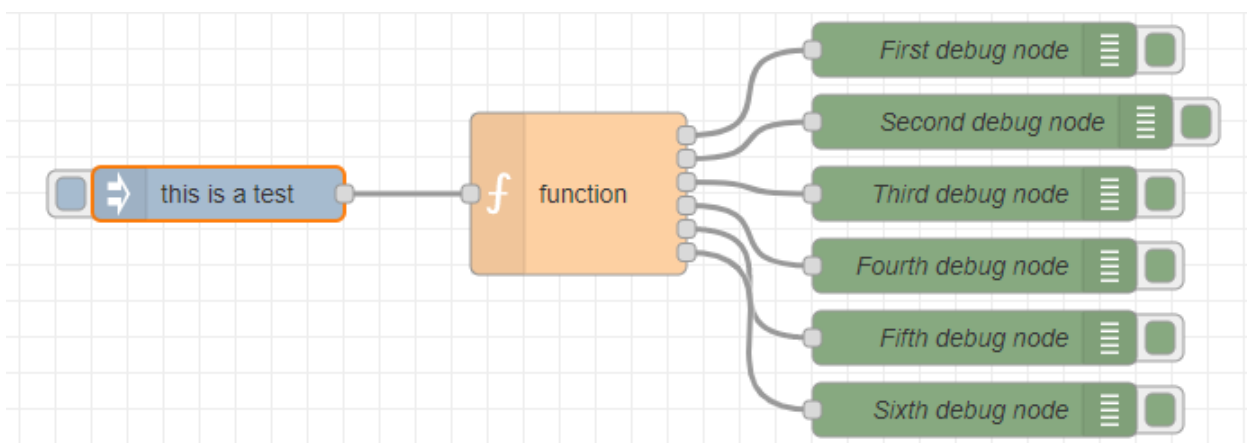
This will split the message from the payload. For this, let's change the input type of the **inject** node to String, and type a random message.



Next, we have to configure as many outputs as we want. For this, you can copy the two already existing **debug** nodes a couple of times, and rename them.



Deploy the changes and run the program with different inputs.



You've made it this far. Good job, here's an appreciation doggo

