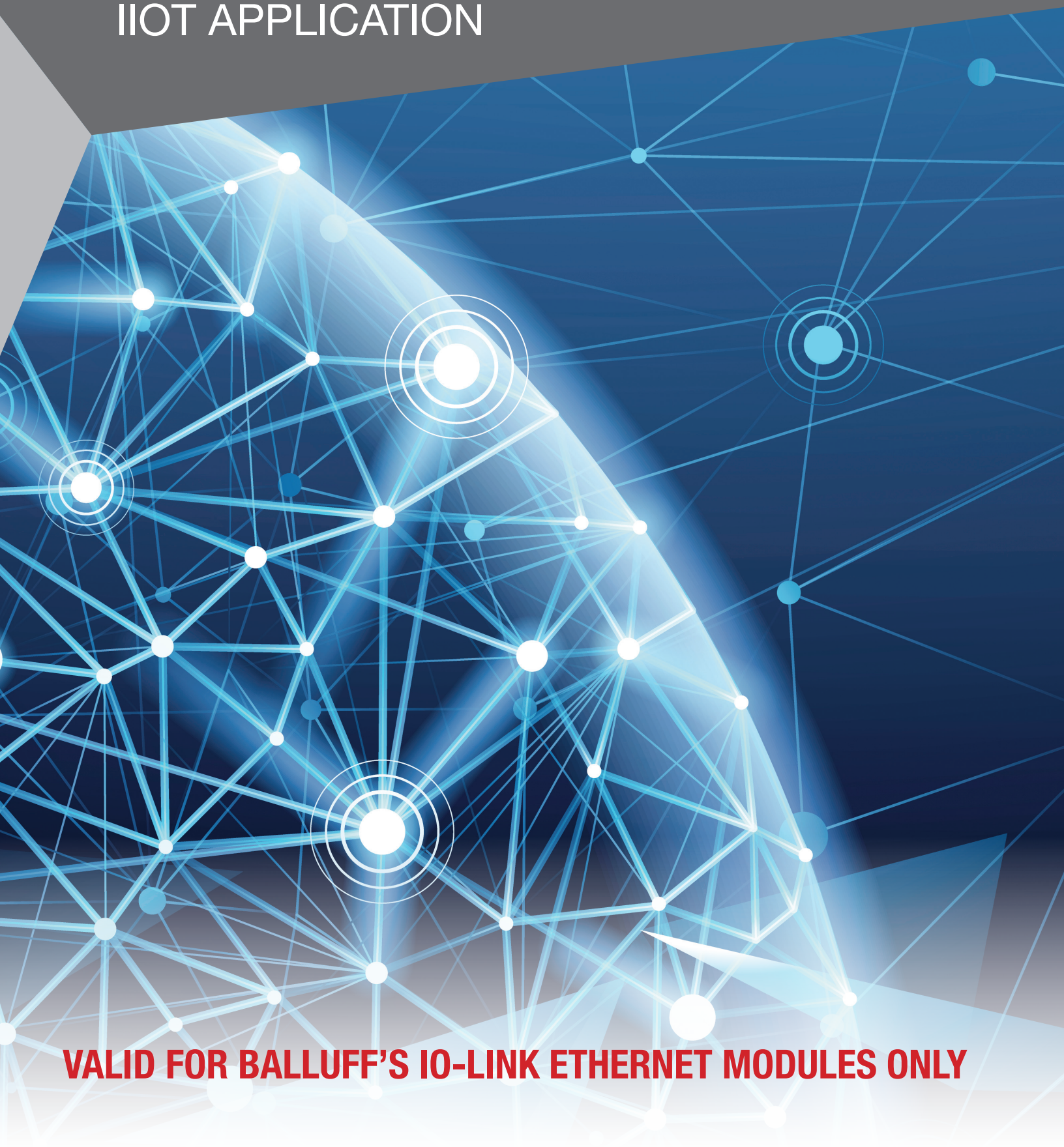
BALLUFF

PROGRAMMERS HANDBOOK FOR IIOT APPLICATION



VALID FOR BALLUFF'S IO-LINK ETHERNET MODULES ONLY

DISCLAIMER

THIS SOFTWARE AND THE DEMO CODES ARE PROVIDED “AS IS” AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

CHAPTER 1

What is BNI006A?

BNI006A is short order code for BNI EIP-508-105-Z015 EtherNet/IP™ IP67 module. It is Eight-Port IO-Link Capable Input/output configurable network block.

The Key Feauteres of this Network Block are :

Principle of operation	Active splitter
Interface	EtherNet/IP
Operating voltage Ub	18...30.2 VDC
Connection (COM 1)	M12x1-Female, 4-pole, D-coded
Connection (COM 2)	M12x1-Female, 4-pole, D-coded
Connection (supply voltage IN)	7/8”-Male, 4-pole
Connection (supply voltage OUT)	7/8”-Female, 4-pole
Connection slots	8x M12x1-Female, 5-pole, A-coded
Digital inputs	16x PNP, Type 3
Digital outputs	16x PNP
Configurable inputs/outputs	yes
Output current max.	2 A
Current sum UA, actuator	9.0 A
Housing material	Zinc, Die casting
Dimension	68 x 37.9 x 224 mm
Ambient temperature	-5...70 °C
IP rating	IP67
Auxiliary interfaces	8x IO-Link
IO-Link version	1.1
Port-class	Type A

Links to Manual and Datasheets

DataSheet :
http://publications.balluff.com/pdfengine/pdf?type=pdb&id=226139&con=en&ws=appoval

Manual:
https://assets.balluff.com/WebBinary1/MAN_BNI_EIP_50x_105_Z015_EN_F17_DOK_933690_01_000.pdf

Fig: BNI006A Ethernet Module

This Guidebook has been written for firmware version 4.4; older version will not work with all the codes shown in the guide book.

This guidebook assumes a working knowledge of how to set up the networking and Basic Computer Operation.
This guidebook uses Node-Red as the mode for running and creating demo functionalities therefore, working knowledge of how to install and code in Node-red environment is expected.
Also, basic knowledge Of javascript or other high-level language is assumed.

CHAPTER 2

Node-Red

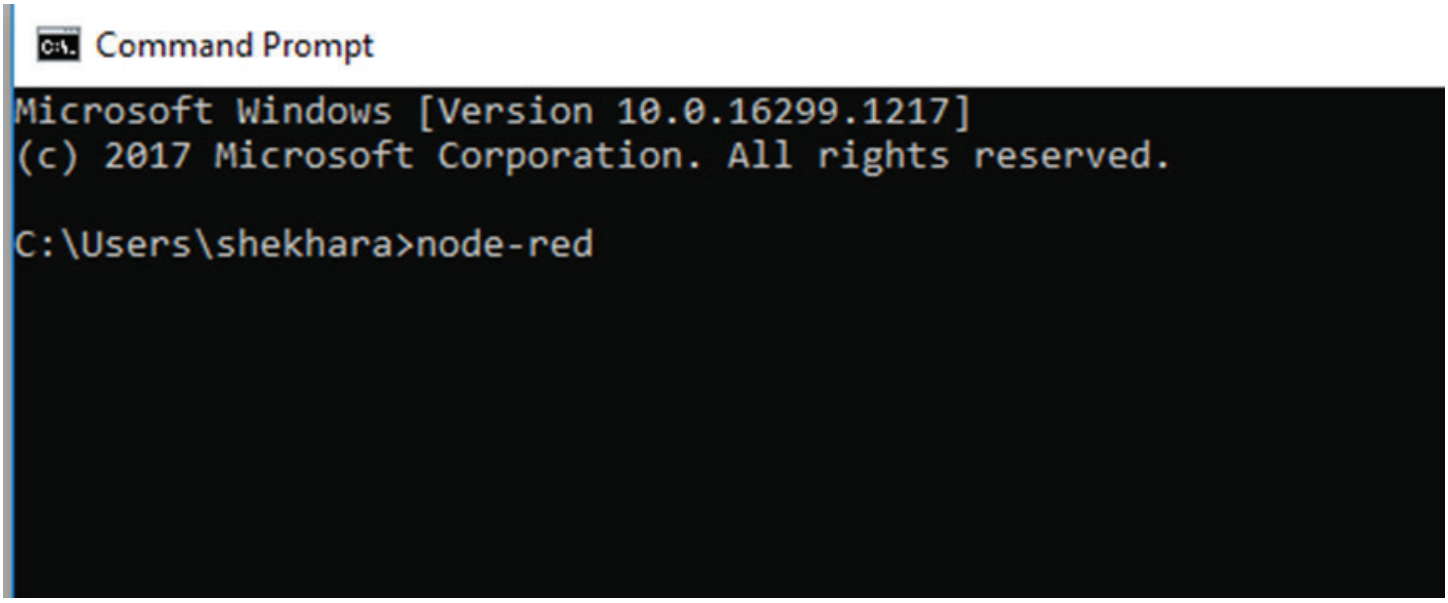
Node-RED is a programming tool for wiring together hardware devices, APIs and online services in new and interesting ways.
It provides a browser-based editor that makes it easy to wire together flows using the wide range of nodes in the palette that can be deployed to its runtime in a single-click.1

Installing Node-Red

Step 1: Go To https://nodejs.org/en/ Step 2: Download and Install Node JS
Step 3: Go To https://nodered.org/docs/getting-started/local Step 2: Follow the directions to install Node-Red

Writing Your First Program

Once you have successfully installed Node-Red Open Command prompt and type node-red and press ENTER



The node-red command starts node-red and you get information about the pallets its loading and the libraries

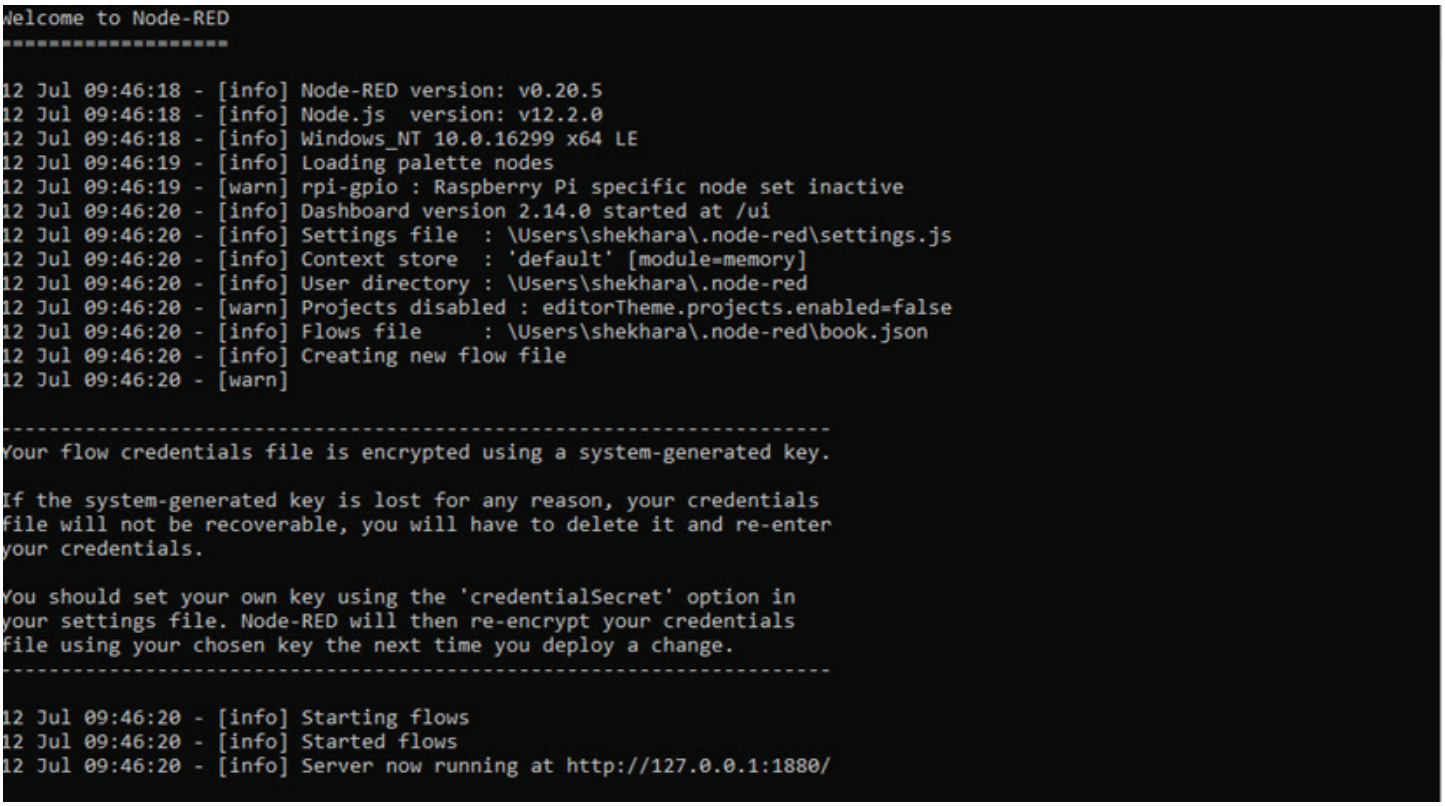


Fig: Node-red command output

The last line informs you where the node-red server has been started. It is by default at http://127.0.0.1:1880/.

Open a Browser (Google Chrome is recommended) and type in the address http://127.0.0.1:1880/ and press ENTER.

You would be presented with the following interface (Screenshot on Next Page)

Fig: Node-red start window

If you don't have similar results you may want to recheck the installation and probably google to see what's the issue.

Before proceeding Please Make Everything is Similar. The examples in this guide uses Node-red version 0.20.5 and Node JS version v12.2.0

Node-red is very beginner-friendly but if you find it difficult in following any of the concepts or node functionalities Node-red Documentation is a good place to start.

Hello World!

For this program, we need a way to input our text and a way to display it. We will use Inject node to input the text and a debug node to view the output in debug window.

Drag the Inject node from the input node from the left toolbox window.
The default inject value is a timestamp. We need to change that to string. Click on the inject node you will see a properties window open up for that node.

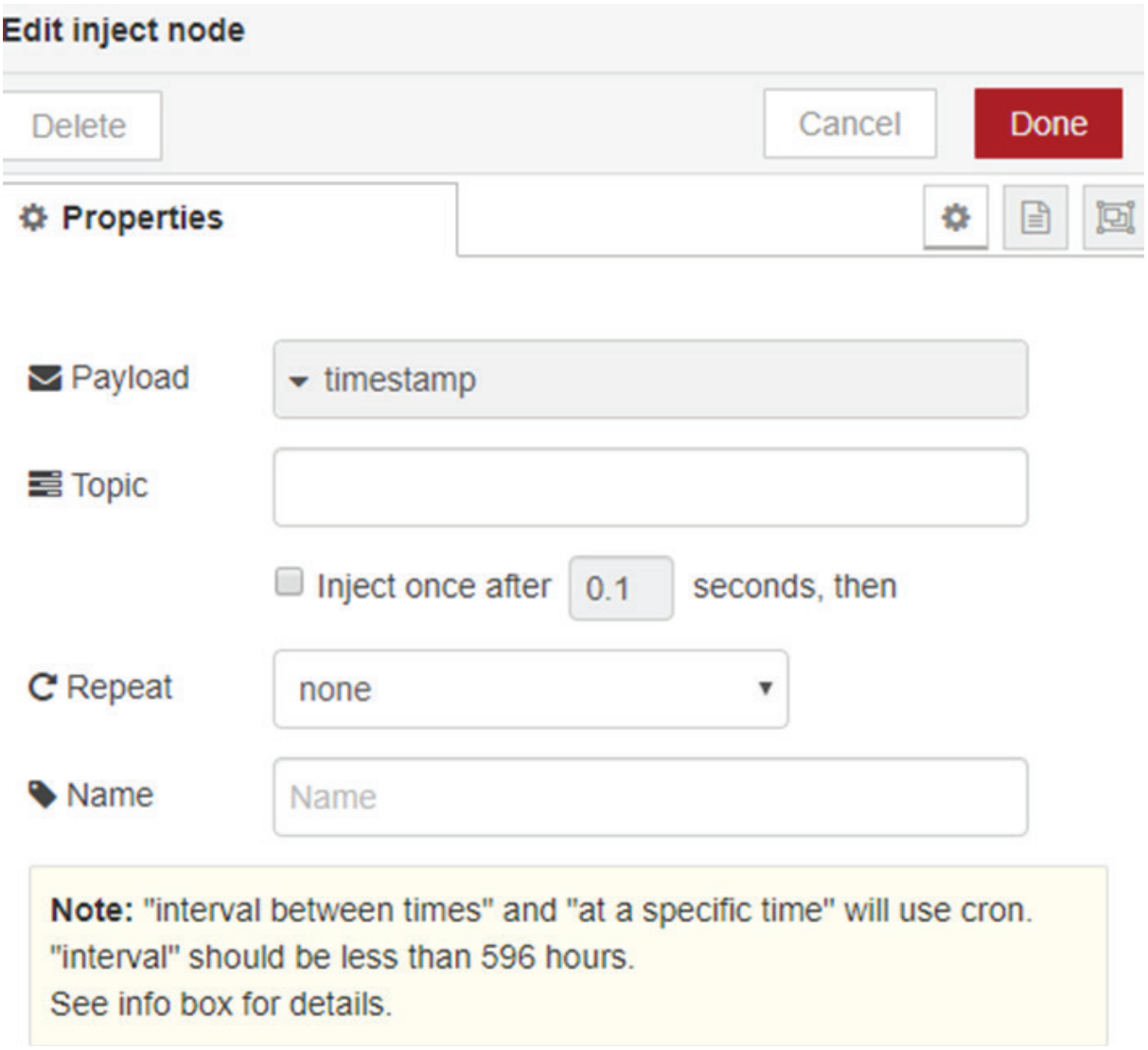
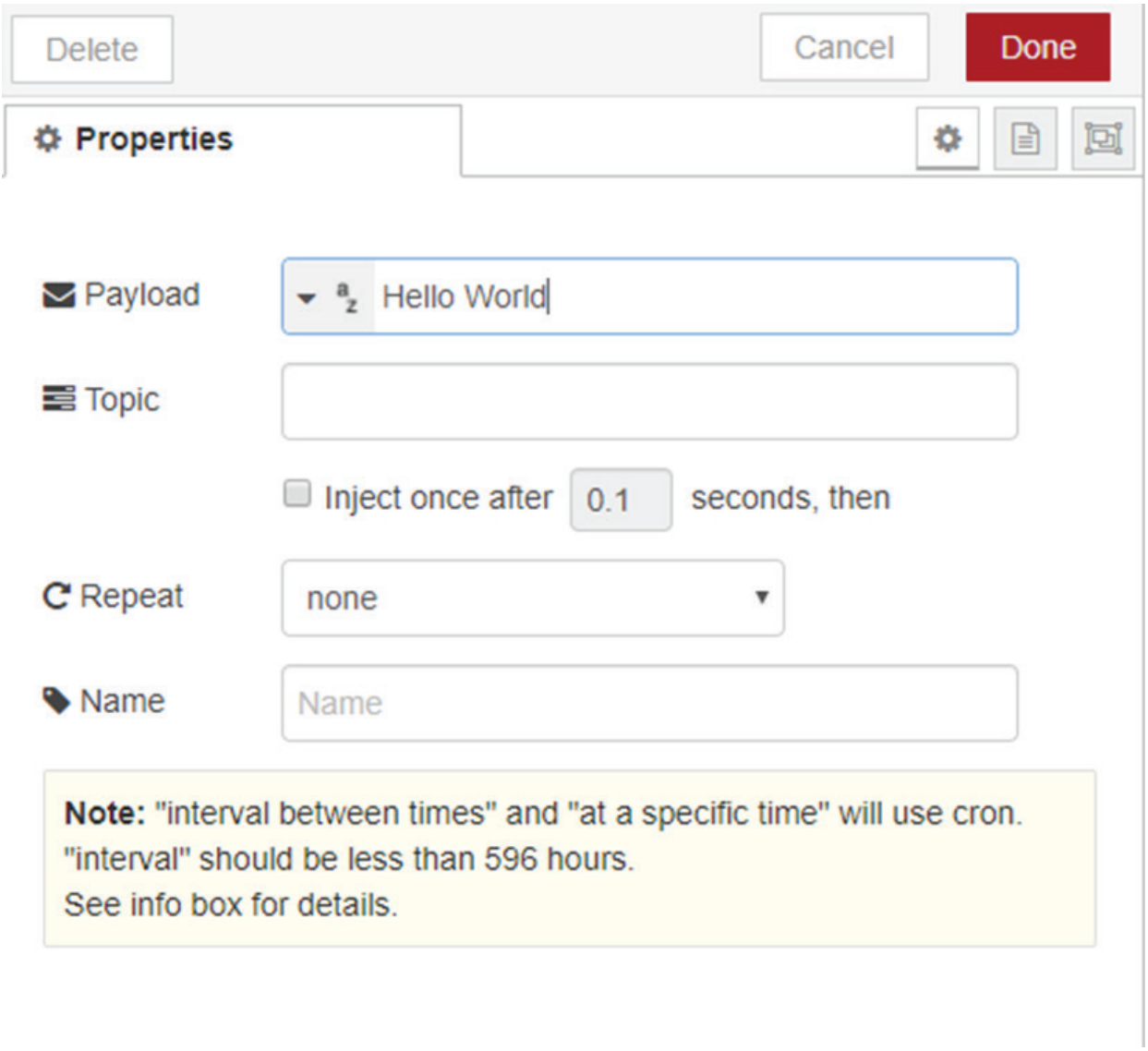
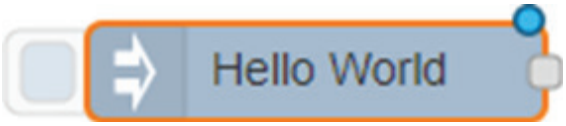


Fig: Default settings for inject node



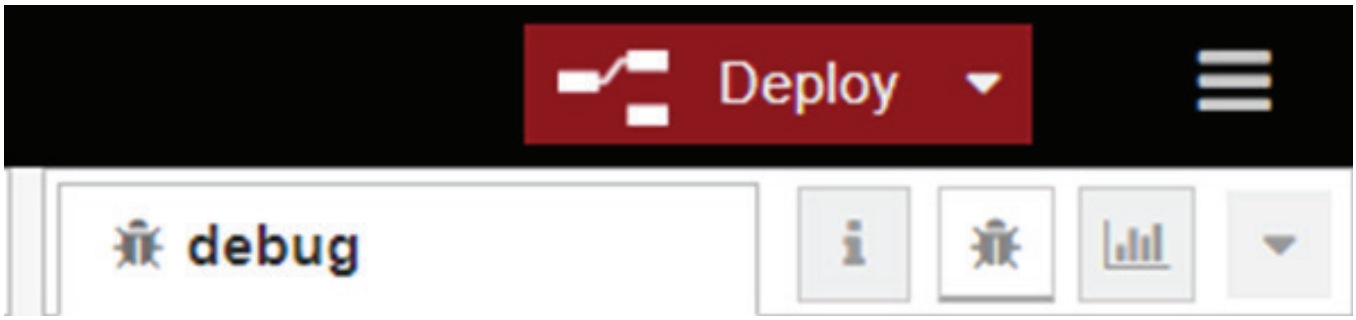
Change the Payload to String type and value to “Hello World”.



The Node should look like as shown above.

Drag a Debug Node into the flow and connect the two nodes. Your result should be as shown above.

Click on Debug on top-right corner of the window to deploy the flow.



Click on debug button to open Debug window. It is the bug icon button. when you inject the time stamp you should see Hello world Output in the debug window.

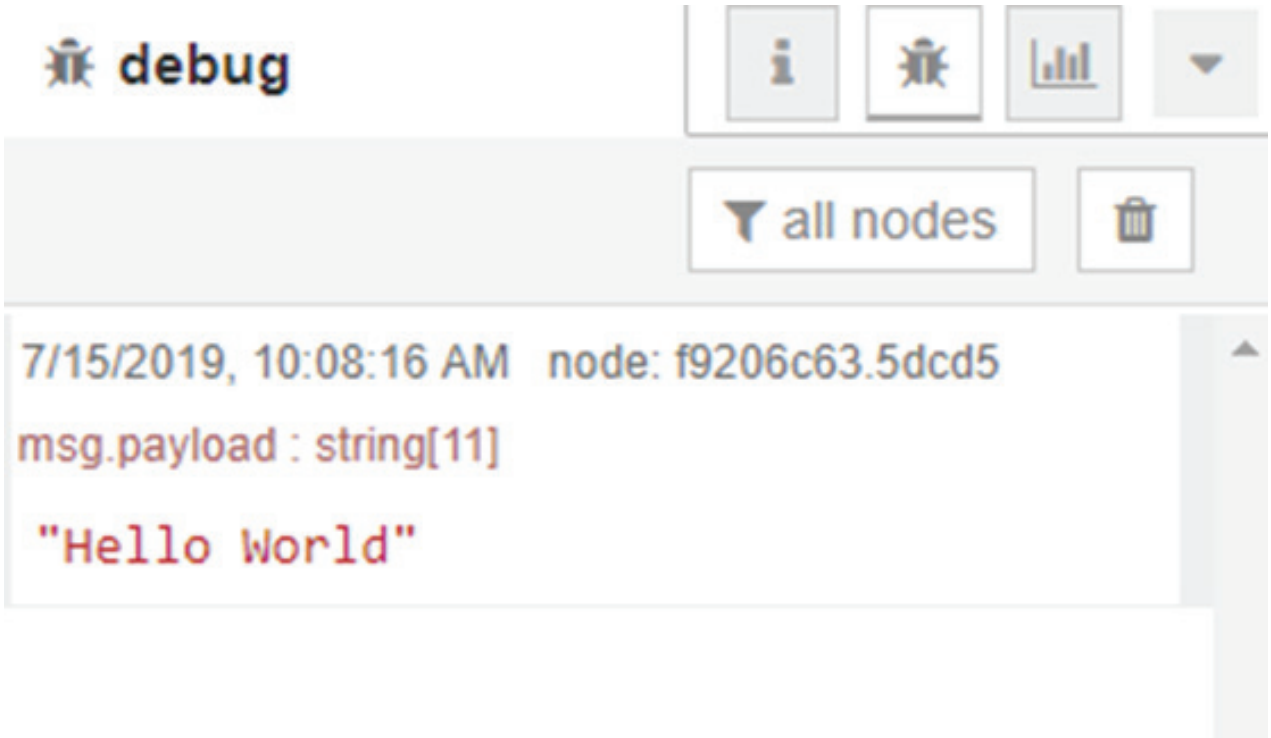


Fig: Output

CHAPTER 3

WebServer

Balluff's Ethernet/IP Fieldbus has an inbuilt web server which can be accessed by going to IP Address of the module using a web browser. Modern web browsers supporting Javascript is needed for functionality.

I am using google chrome for demo throughout the guide. Mozilla Firefox will work too. refrain from using Internet Explorer or Safari.

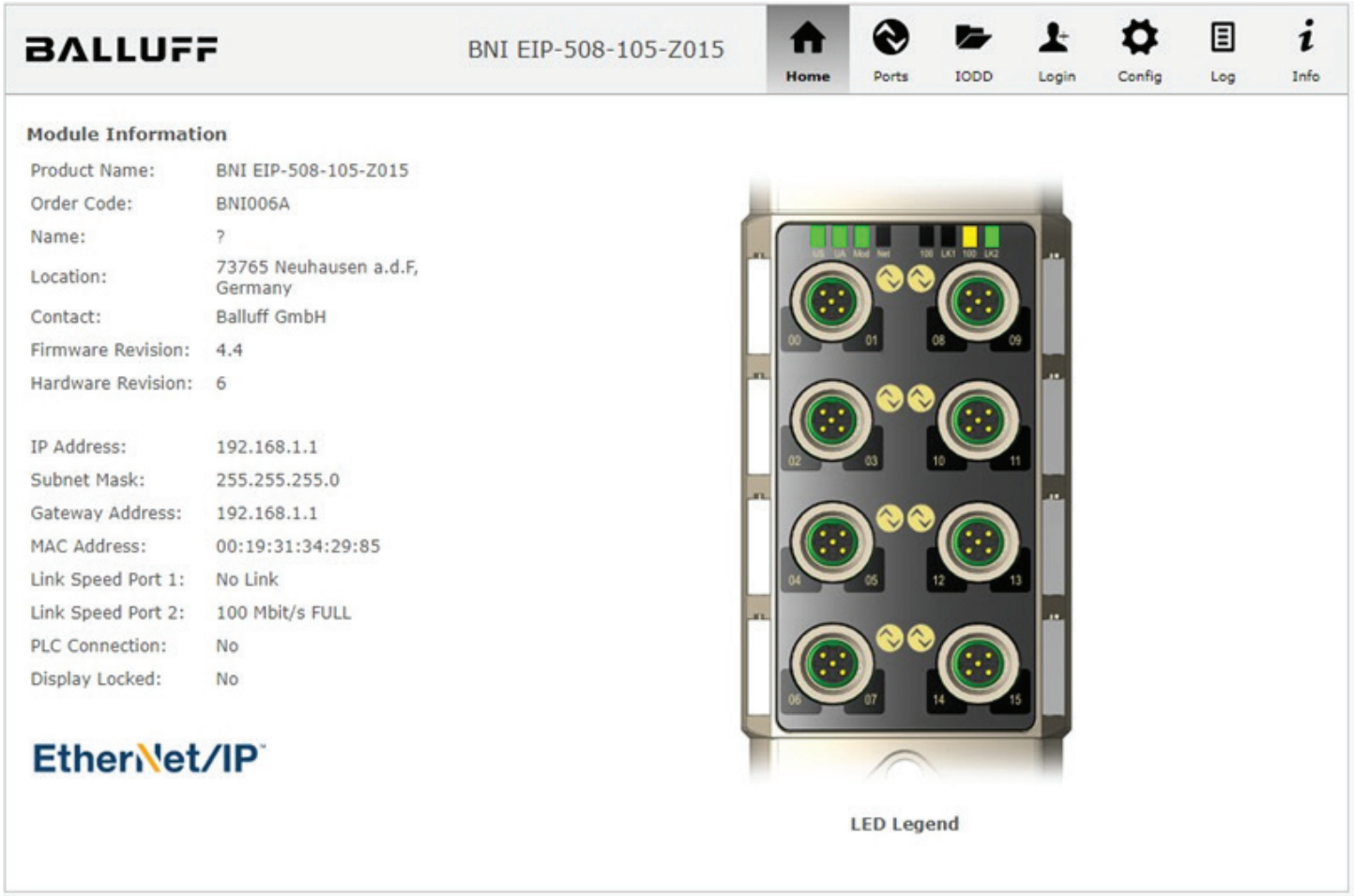


Fig: Webserver on BNI006A

The Manual for the Module explains how to configure and set parameters using the webserver please refer to it for that information.

However, we will be using JSON endpoints used by these pages to get and build our own application for monitoring the blocks.

This method works independently of PLC so can be used in conjunction with PLC's, unlike the UDP which is described in a later section.

JSON

JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. It is based on a subset of the JavaScript Programming Language, Standard ECMA-262 3rd Edition - December 1999. JSON is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others. These properties make JSON an ideal data-interchange language.1

JSON is built on two structures:

A collection of name/value pairs. In various languages, this is realized as an object, record, struct, dictionary, hash table, keyed list, or associative array.
An ordered list of values. In most languages, this is realized as an array, vector, list, or sequence. These are universal data structures. Virtually all modern programming languages support them in one form or another. It makes sense that a data format that is interchangeable with programming languages also be based on these structures.1
More about specification can be learned at http://www.ecma-international.org/publications/ files/ECMA-ST/ECMA-404.pdf

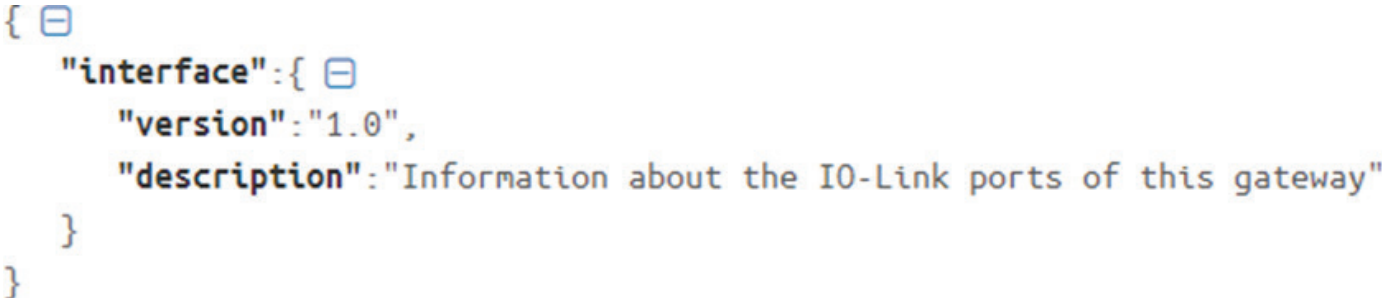


Fig: Output

A JSON object starts with { and ends with}. There are name-value pairs making the objects. The advantage of this notation is values can be accessed by name directly.

CHAPTER 4

Getting Started

There are few prerequisites to follow the steps in the book.

- Make sure the ports are configured to IO-Link.
- The Connected Devices are IO-Link capable.

You can configure the ports using inbuilt webserver and check the device datasheet to confirm if the device is IO-Link capable or not.

You can see my setup after configuring the port. I have 5 devices connected to BNI006A. Your screen will be different depending on what sensors are plugged in.

Make sure you have sensor manual on hand we would be using that to make sense of process data a little bit later.

Getting Process Data

For this tutorial, we will be using BAE00TP an IO-Link switching power supply from Balluff.
The Link To the Product is: https://www.balluff.com/local/us/productfinder/#/ca/A0015/product/ G1504/variant/PV152191

You can use any IO-Link capable sensor you would have to modify the logic accordingly but the process flow remains the same.

The Manual for this can be found at https://assets.balluff.com/WebBinary1/MAN_BAE_PS_XA_1W_24_038_080_60_I_X_G15_DRW_923210_00_000.pdf

Fig: BAE00TP Powersupply from Balluff

We will get the Voltage and Current from the process data and display it in node-red debug console window. You will notice there is a lot of information available apart from process data that can be extracted and displayed following the same procedure. For this guide we will stick to monitoring the Process Data.

The JSON end-point to acess the data is IPaddress/ports.jsn. If you go to this URL you will see a JSON object in the browser. It will vary depending on how the ports are configured and what devices are connected.
It should be similar to the screenshot shown below.

Fig: Screenshot of ports.jsn endpoint

The Logic

To get the JSON data we would need to send out HTTP request and get the JSON reply and Parse it to extract ProcessInputs which contains the data we are interested in.

The Nodes That we will use in this Flow are

1. Inject Node
2. Http Request Node
3. Function Node
4. Debug Node

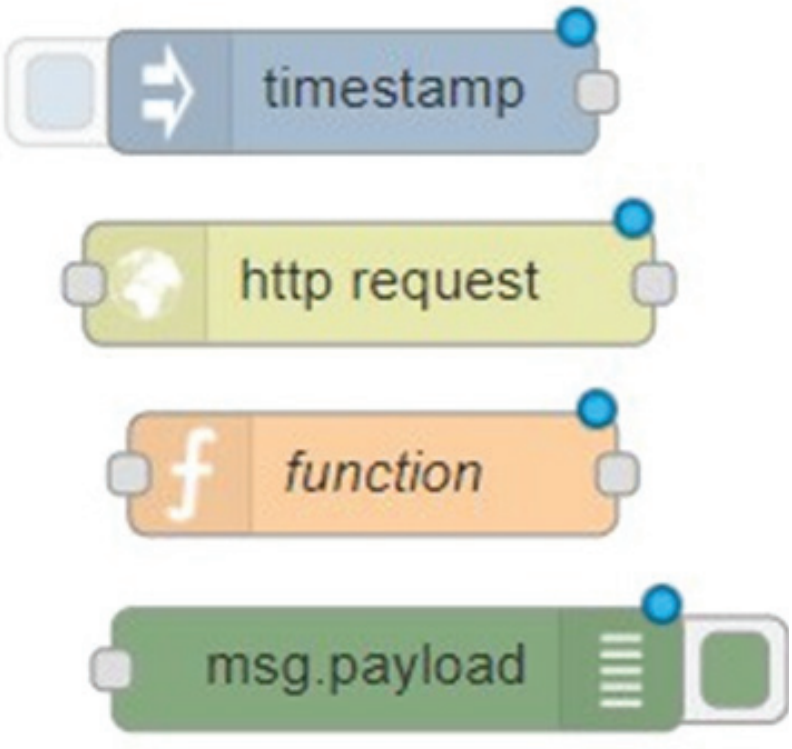


Fig: The nodes needed for this flow

The flow is
inject timestamp --> Send Http request to IPaddress/ports.jsn --> get response data --> extract current and voltage value --> display it as as debug message.

Drag all the nodes to the flow and configure them as follows

Inject Node

We want to trigger get Http request once so we leave this node to default with inject type as a timestamp.

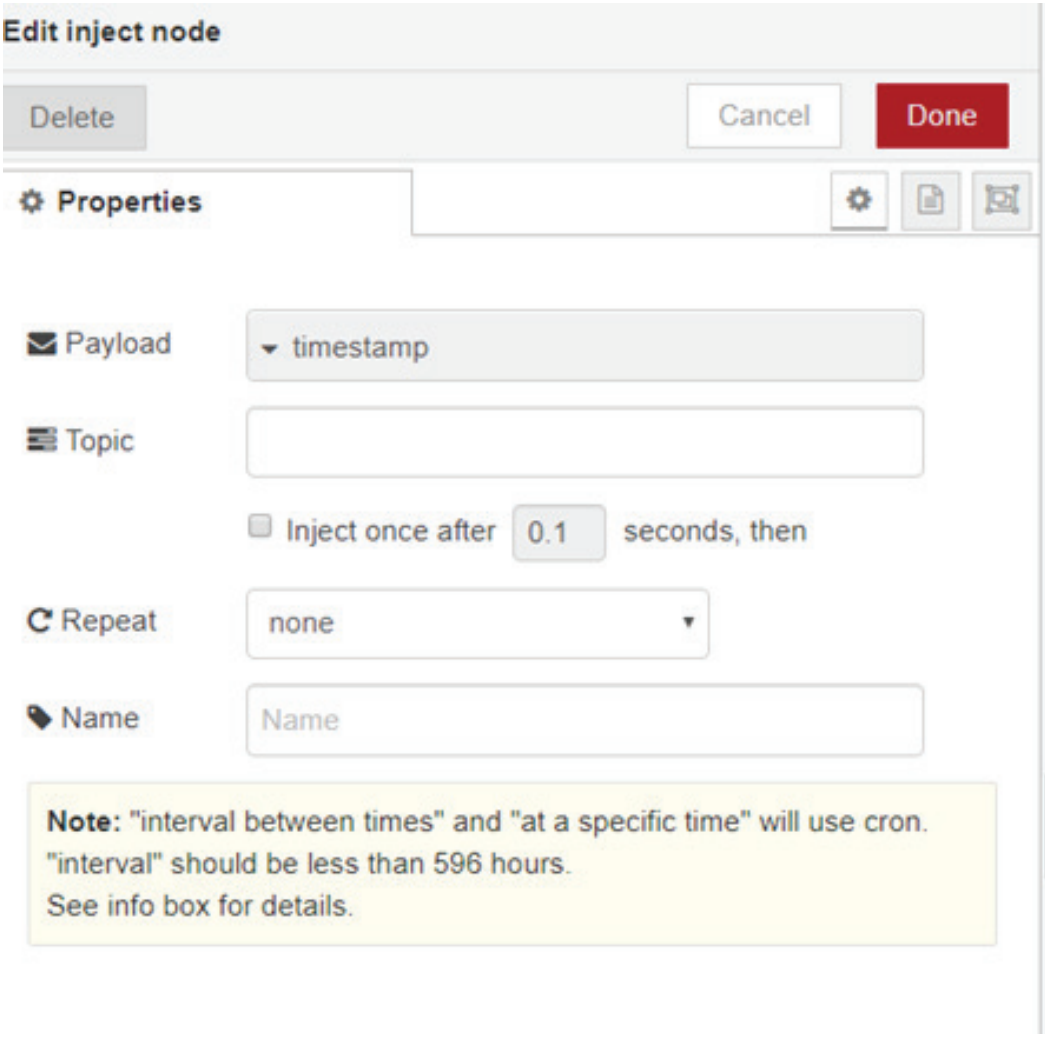


Fig: Inject Node Configuration

Http Request Node

We need to send a GET request to URL IP_ADDRESS OF MASTER/ports.jsn and the return type is a Parsed JSON Object.

Configure the Http request node as shown in the screenshot on the next page. My Ip is 192.168.1.1, yours might be different

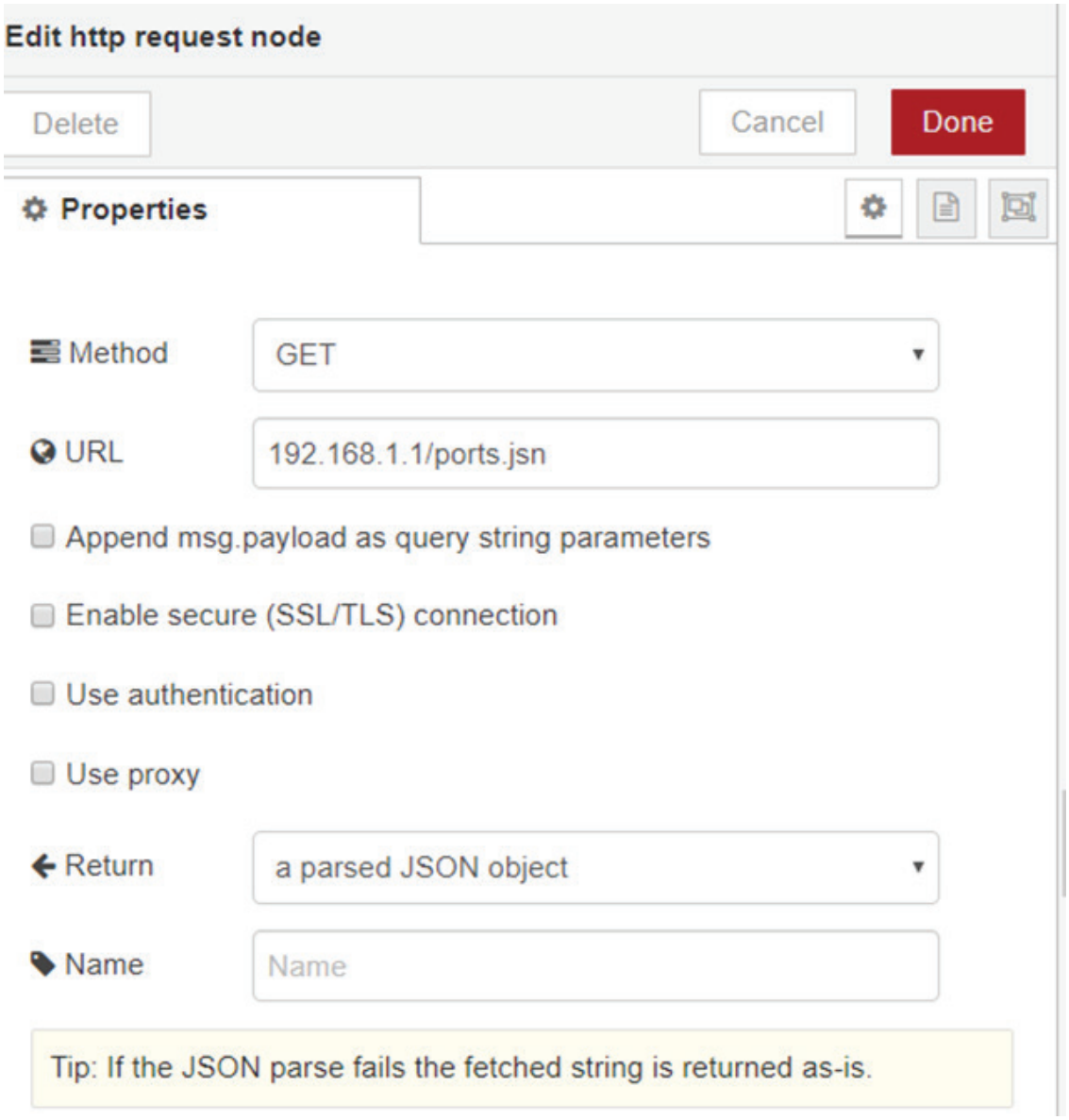


Fig: Http Node Configuration

This node sends out a GET request and returns a parsed JSON object which will be similar to the object that is shown on next page.

{
“interface”: { “version”: “1.0”,
“description”: “Information about the IO-Link ports of this gateway”
},

“ports”: [
{
“vendorName”: “BALLUFF”, “vendorText”: “www.balluff.com”,
“productName”: “BTL6-U110-Mxxxx-PF-S4”, “productId”: “BTL6PFIOL001”,
“productText”: “Balluff Position Transducer”, “serialNumber”: “18112000082843CN”,
“hardwareRevision”: “03”, “firmwareRevision”: “v1.03.002”, “applicationSpecificTag”: “”,
“event”: “0x0”,
“eventFlag”: “0x0”,
“processInputs”: “00 01 88 3F 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00”,
“processOutputs”: “00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00”,
“directParameters”: “98 0D 0A 1B 11 83 00 03 78 03 01 01 00 00 00 00”,
“status”: “87FF”, “dsContentVendorId”: “03 78”,
“dsContentDeviceId”: “03 01 01”,
“dsContentChecksum”: “D8 00 00 00”,
“dsContentBuffer”: “<table><tr><td>Index</td><td>Length</td><td>Data (hex)</td></tr><tr valign=top><td align=center>12.0</td><td align=center>2</td><td>00 00</td></tr><tr valign=top><td align=center>24.0</td><td align=center>32</td><td>00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00</td></tr></table>”
},
{
“vendorName”: “BALLUFF”, “vendorText”: “www.balluff.com”,
“productName”: “BAE PS-XA-1W-24-038-607-I”,
“productId”: “BAE00TP”,
“productText”: “Power Supply, IP67, 1-phase, 24Vdc, 3,8A”,
.....
.....
.....
.....
.....
.....

“dsContentDeviceId”: “(none)”, “dsContentChecksum”: “(none)”, “dsContentBuffer”: “(none)”
}
]
}

The JSON object contains data for all the ports. From BNI006A the data of IO-Link Ports are in the ports object and the port mapping is as shown below.

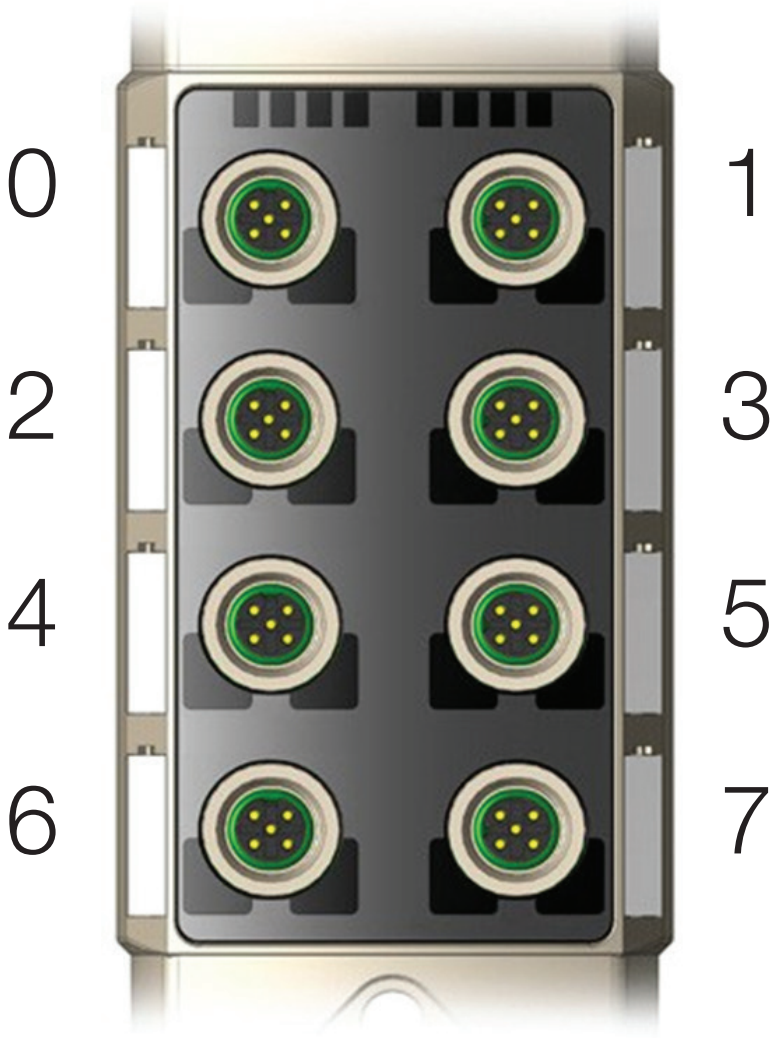


Fig: BNI006A JSON data Port Mapping

My IO-Link PowerSupply is connected to port 1 according to the above notation. This means my IO-link data is in Object ports[1]. since ports are an array of port data.

To retrieve the Input process value we can use the dot operator to reference it so the exact reference is ports[1].processInputs.

You can learn more about how to access JSON objects from https://www.w3schools.com/js/js_ json_ objects.asp

After the get Http request, we have the JSON object. We can extract the input process data using the dot operator.from the port but how to extract the voltage and current values?

To do so we would need to look at the manual. for my use case, it is on page 12. You can see in fig shown below an extract from the manual. it tells that 31 to 20 bits are voltage values and 19 to 8 are current values.

Since each hex character is 4 bits we have to extract 12/4 = 3 characters fro each of these values. So first 6 character in Input process Data is voltage and current values.

Process data

The process data length of the power supply is 32 bits. Included in the process data are the switching states of the 4 switching outputs (BDC1 ... BDC4) as well as the actual measurement values for current and voltage.

31...20 bits	19...8	7	6...4	3	2	1	0
Output Voltage Measure 0...30 V Value 0...300 Multiplier 0.1	Output Current Measure 0...80 A Value 0...800 Multiplier 0.1	PSU On/Off	N/C	BDC3/Output drop	BDC2/Over temperature	BDC1/Over voltage	BDC0/Over load

Fig: Process data Breakup from the Manual of BAE00TP

The values for me are as shown in the screenshot. For easier processing I am removing all the spaces from my code and extracting first 3 chars and parsing it as hex and scaling by 0.1 to get the actual voltage value similarly, for current I am extracting 3 chars from 3rd place and converting it to int and scaling by 0.1 to get current value in Amp.

The code shown in the next page does exactly that and sends out 2 messages so that function node should have 2 outputs.


```
"eventFlag": "0x0",
"processInputs": "0E E0 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00",
"processOutputs": "00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00",
"directParameters": "98 62 62 1B 11 83 00 03 78 01 07 04 00 00 00 00",
"status": "87FF",
"dsContentVendorId": "00 00",
```

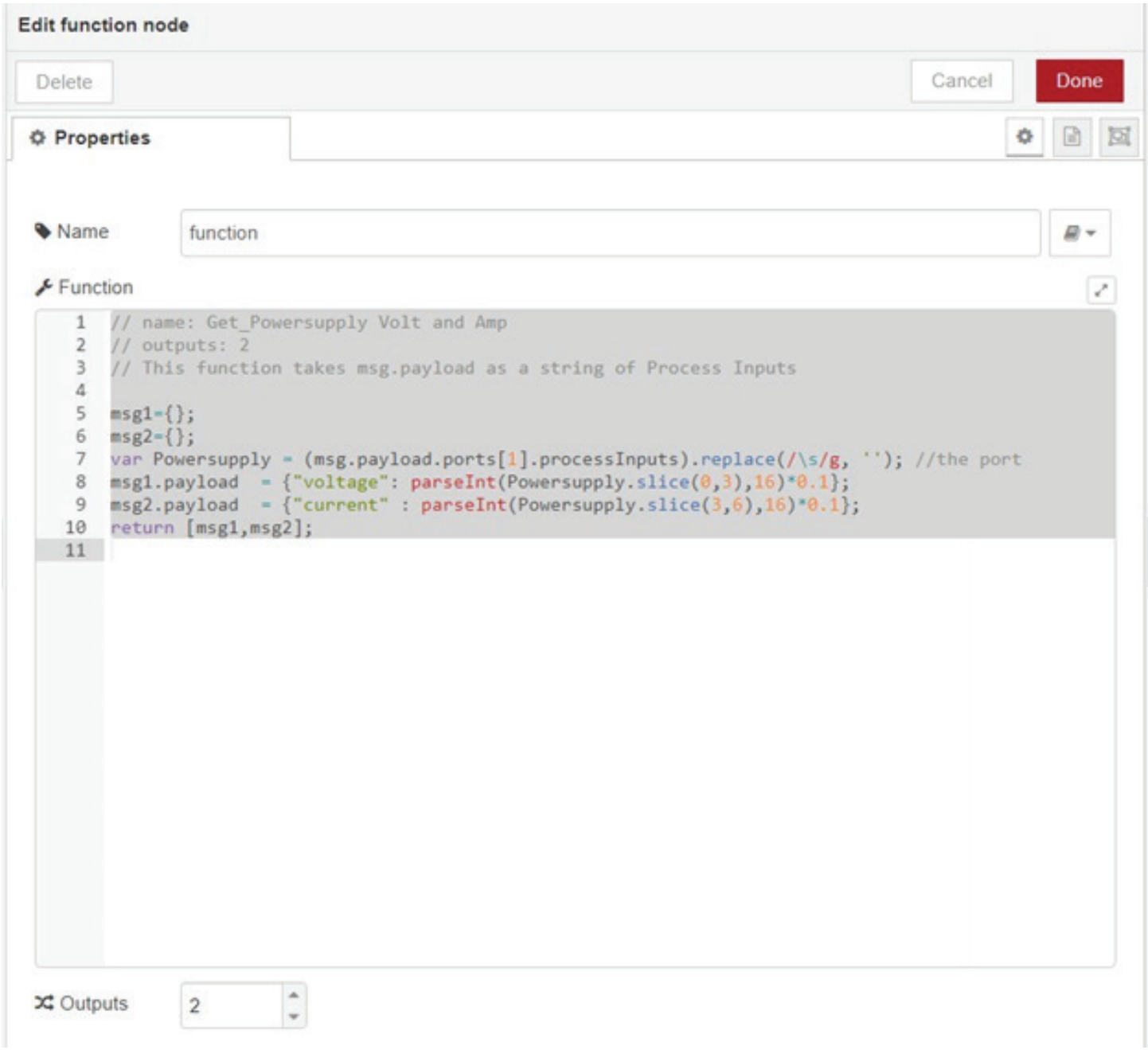
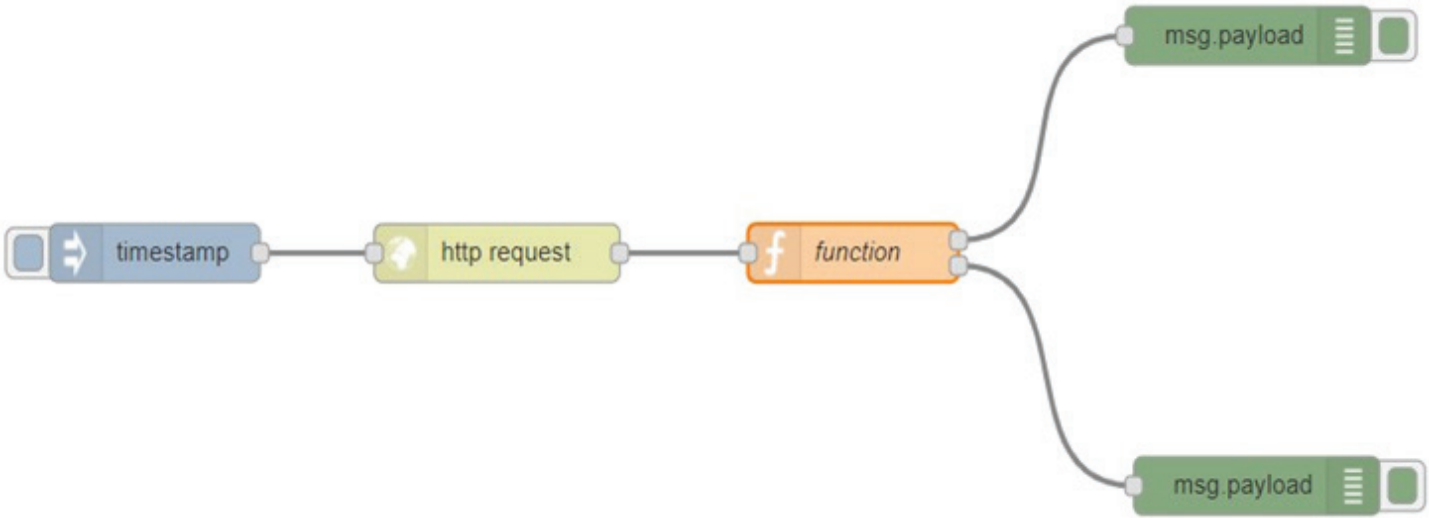


Fig: BNI006A JSON data Port Mapping

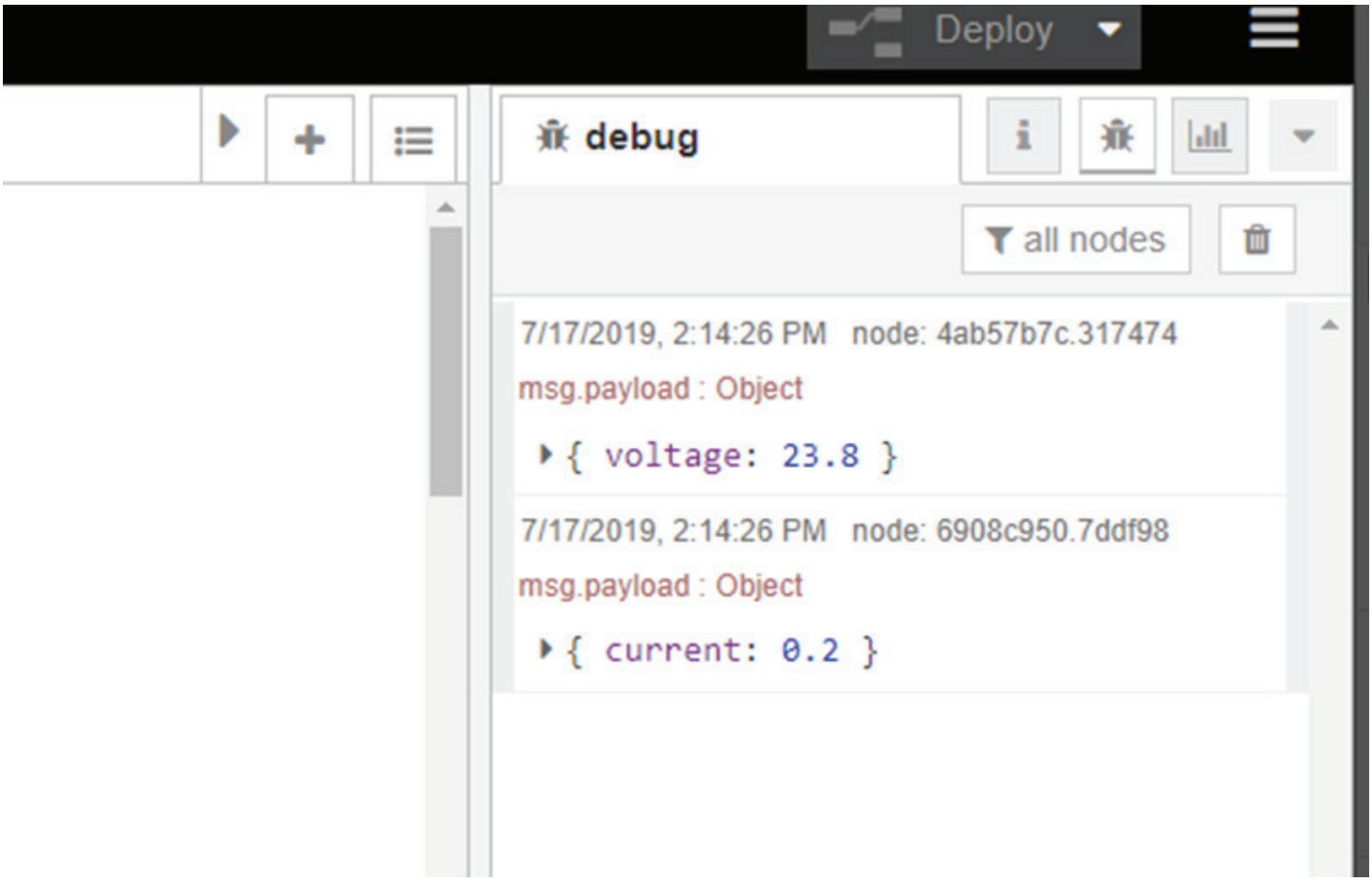
Code For Function Node :

// name: Get_Powersupply Volt and Amp
// outputs: 2
// This function takes msg.payload as a string of Process Inputs
msg1={};
msg2={};
var Powersupply = (msg.payload.ports[1].processInputs).replace(/\s/g, ‘’); //the port msg1.payload = {“voltage”: parseInt(Powersupply.slice(0,3),16)*0.1};
msg2.payload = {“current” : parseInt(Powersupply.slice(3,6),16)*0.1}; return [msg1,msg2];

Finally, attach two debug node. Your flow should look like this :



Click on deploy and inject timestamp. You should get the following output in the debug window.



Click on deploy and inject timestamp. You should get the following output in the debug window.

Getting Server Logs

The procedure to retrieve the logs from the BNI006A blocks remains the same as getting the process data, with the difference being the endpoint and the struture of JSON.
Drag the following Nodes the work area from the side node-panel.

1. Inject node
2. Http Request Node
3. Debug node

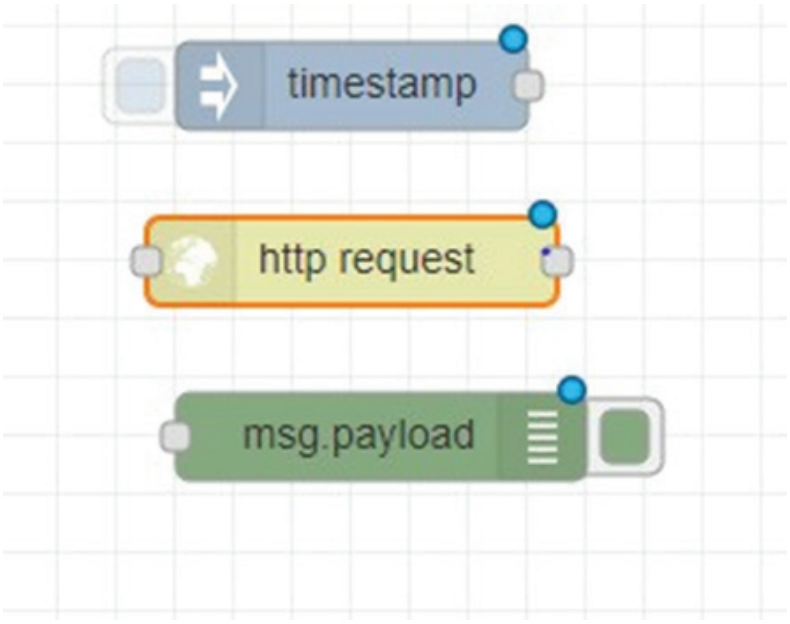


Fig: The nodes needed for this flow

The Endpoint for retrieving the log is IP_ADDRESS_OF_FIELDBUS/log.jsn

Connect the nodes as inject --> HTTP request --> debug

We need to configure the HTTP request node to send a GET request to the endpoint. Both debug and inject node is in default settings.

Refer to screenshot in next page for the configuration on HTTP request node.

My Ip Address is 192.168.1.1 yours might be different.

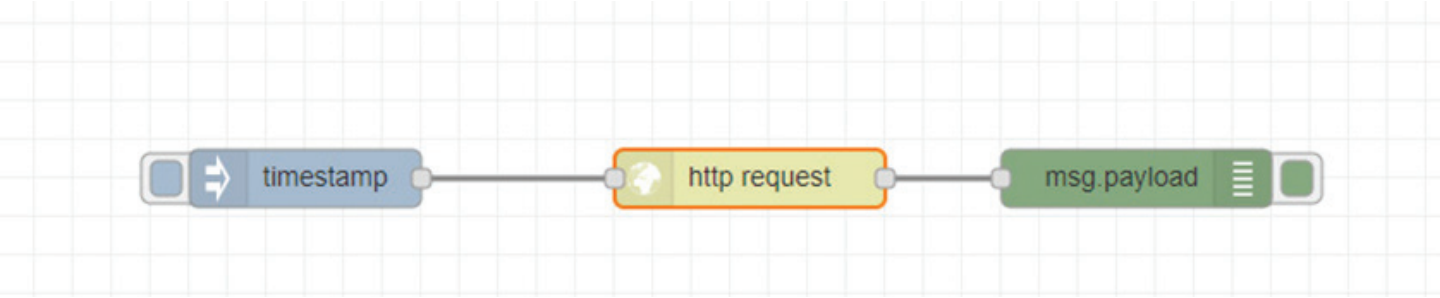


Fig: The wiring for the flow

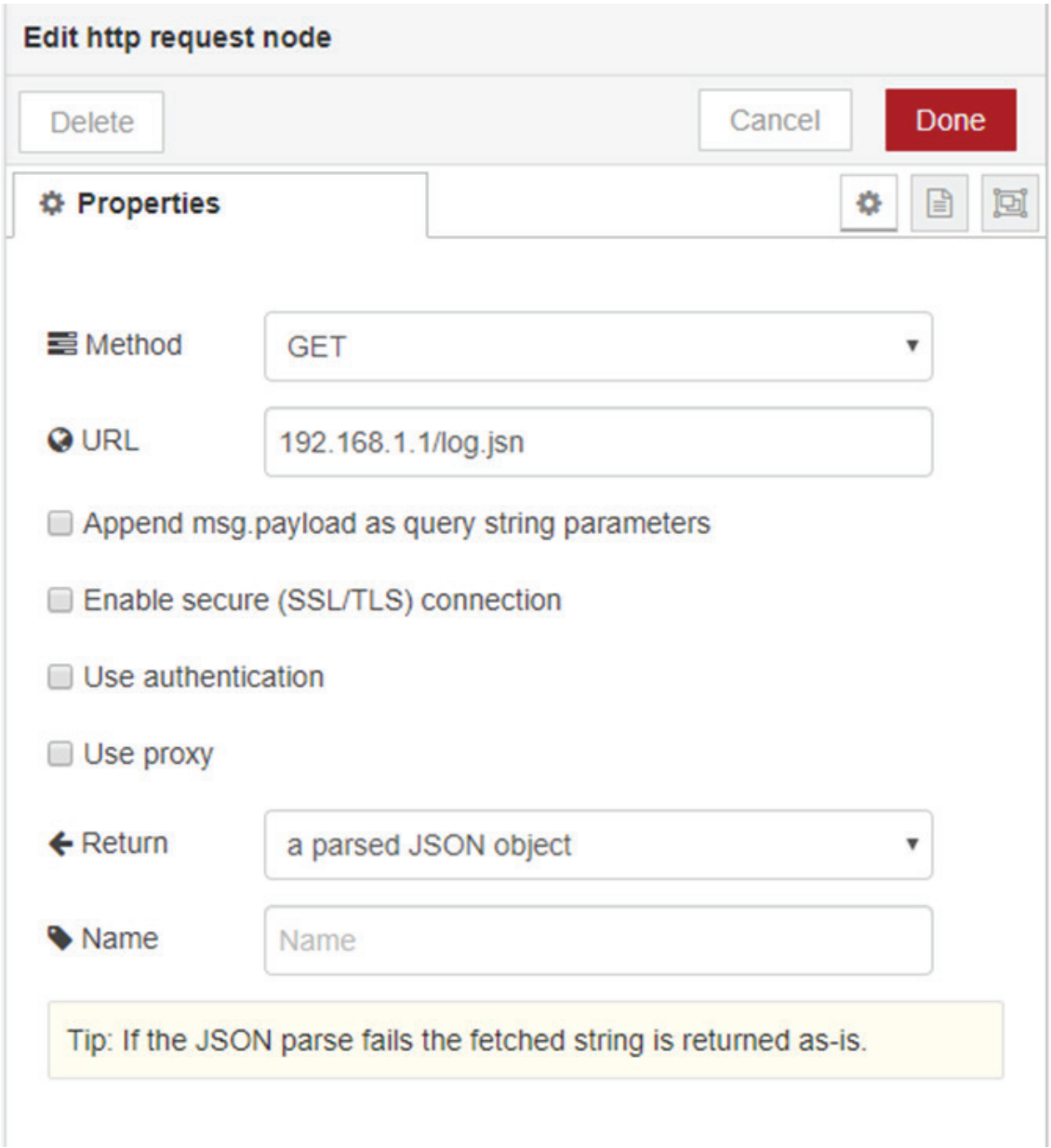


Fig: Configuration of Http request node

After you deploy and execute the node by injecting timestamp you would see the following in the debug window

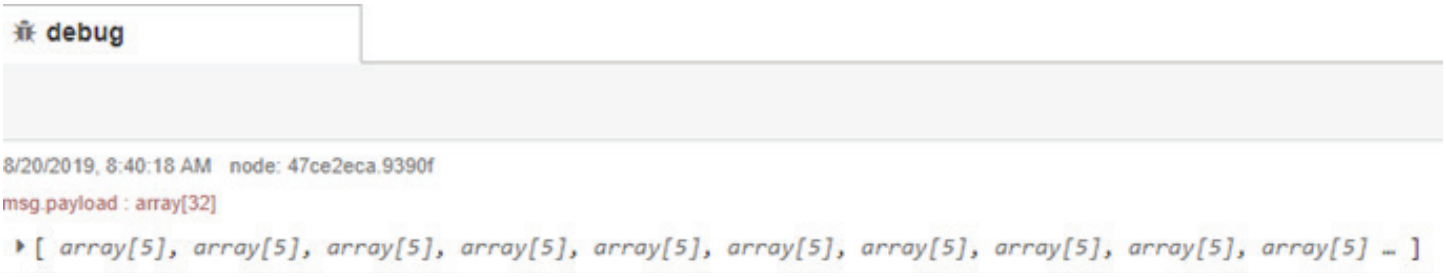
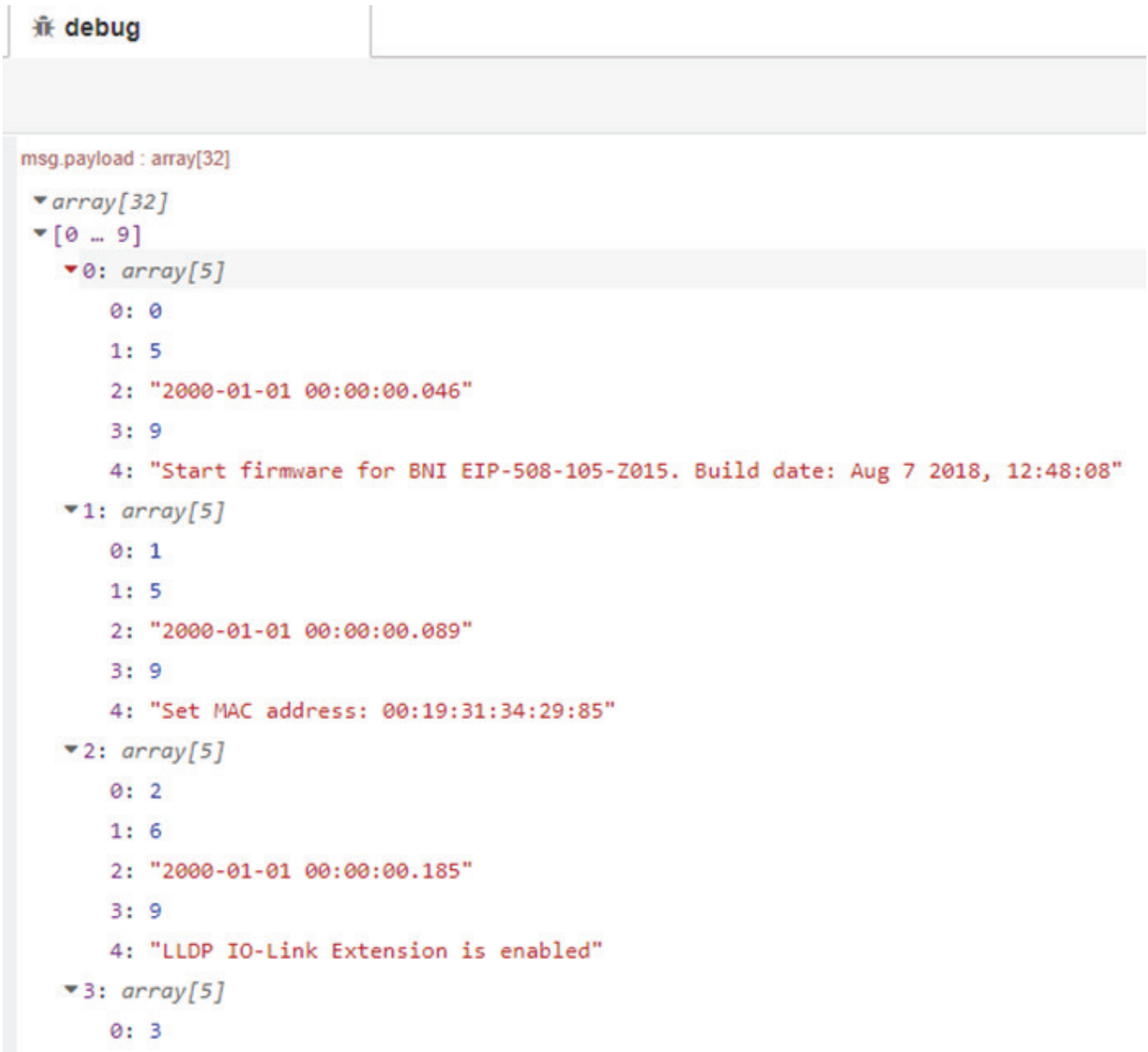


Fig: Reply from the Log endpoint

If you expand the arrays you would see the following structure



Each array has 5 elements. They can be mapped as follows:
0th Element: Number 1st element: Severity 2nd element: Date
3rd Element: Origin of the Event/Error 4th Element: Message

The severity is denoted by numbers from 0 through 7 with 0 being Emergency and 7 being debug
[“Emergency”, “Alert”, “Critical”, “Error”, “Warning”, “Notice”, “Informational”, “Debug”]

The Origin is similarly denoted by numbers from 0 through 11 with 0 being ECT and 11 Web Interface
[“ECT”, “EIP”, “ETH”, “HTTP”, “IOL_DEVICE”, “IOL_MASTER”, “LLDP”, “PNS”, “SNMP”, “SYS”, “UDP”, “WEB_IF”]

For example, the 0th array can be broken into Sl no : 0

Severity: Notice
Date : 2000-01-01 00:00:00.046
Origin: SYS
Message: Start firmware for BNI EIP-508-105-Z015. Build date: Aug 7, 2018, 12:48:08

This can be consumed in any application or dashboard to display the log files by querying this endpoint periodically.

Getting Port Status

The Baluff Network Block has an endpoint index.jsn. This contains information about the led color of the port. This information can be used to determine the port status.

For example, on IO-link Port Green-solid means The port is in IO-Link Mode and device is connected-ed.
And green flashing means the port is in io-link mode but no device connected.

Depending on how the ports are connected you will get similar JSON objects when you navigate to IP_ADDRESS_OF_THE_FIELDBUS/index.jsn

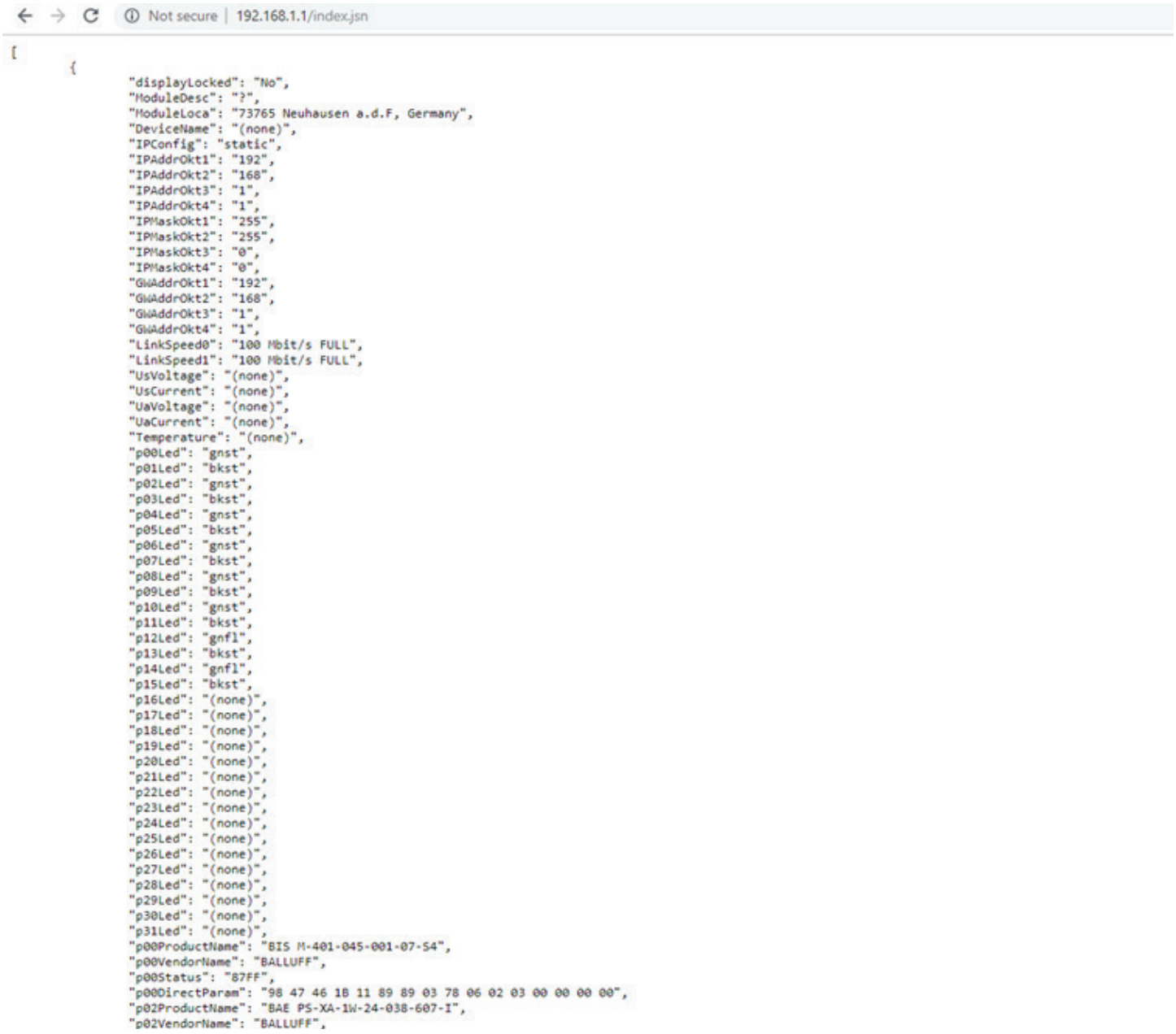


Fig: JSON object for Index.jsn endpoint

If look at the led value it follows the following pattern

gnst means green static
bkst means no color static
rdst means red static
yest means yellow static
gnfl means green flashing
rdfl means red flashing

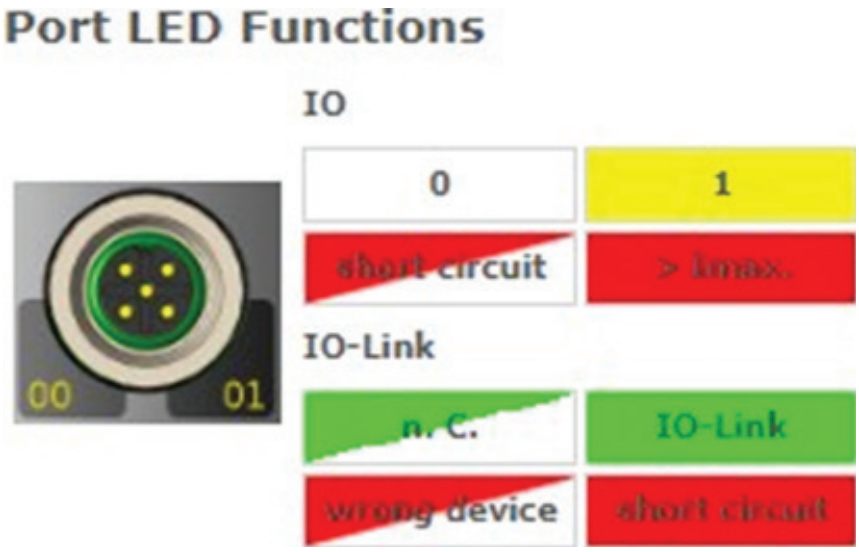
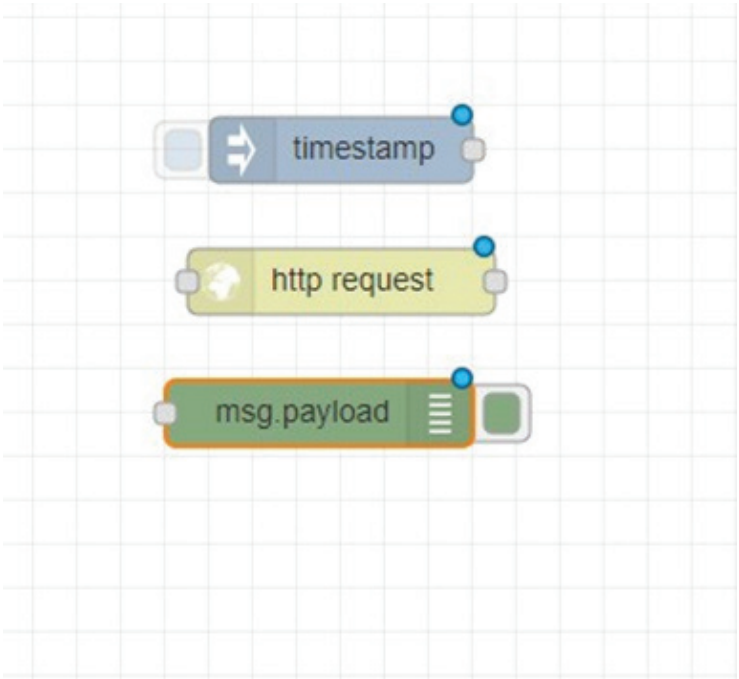


Fig: Port LED color Meaning full means color and half represents flashing

To Get the Status of led 00 to follow the Node-red flow

Drag the Inject HTTP and debug node in the work area



Configure the HTTP node as shown below. Ip address is IP_ADDRESS_OF_FIELDBUS/index.jsn

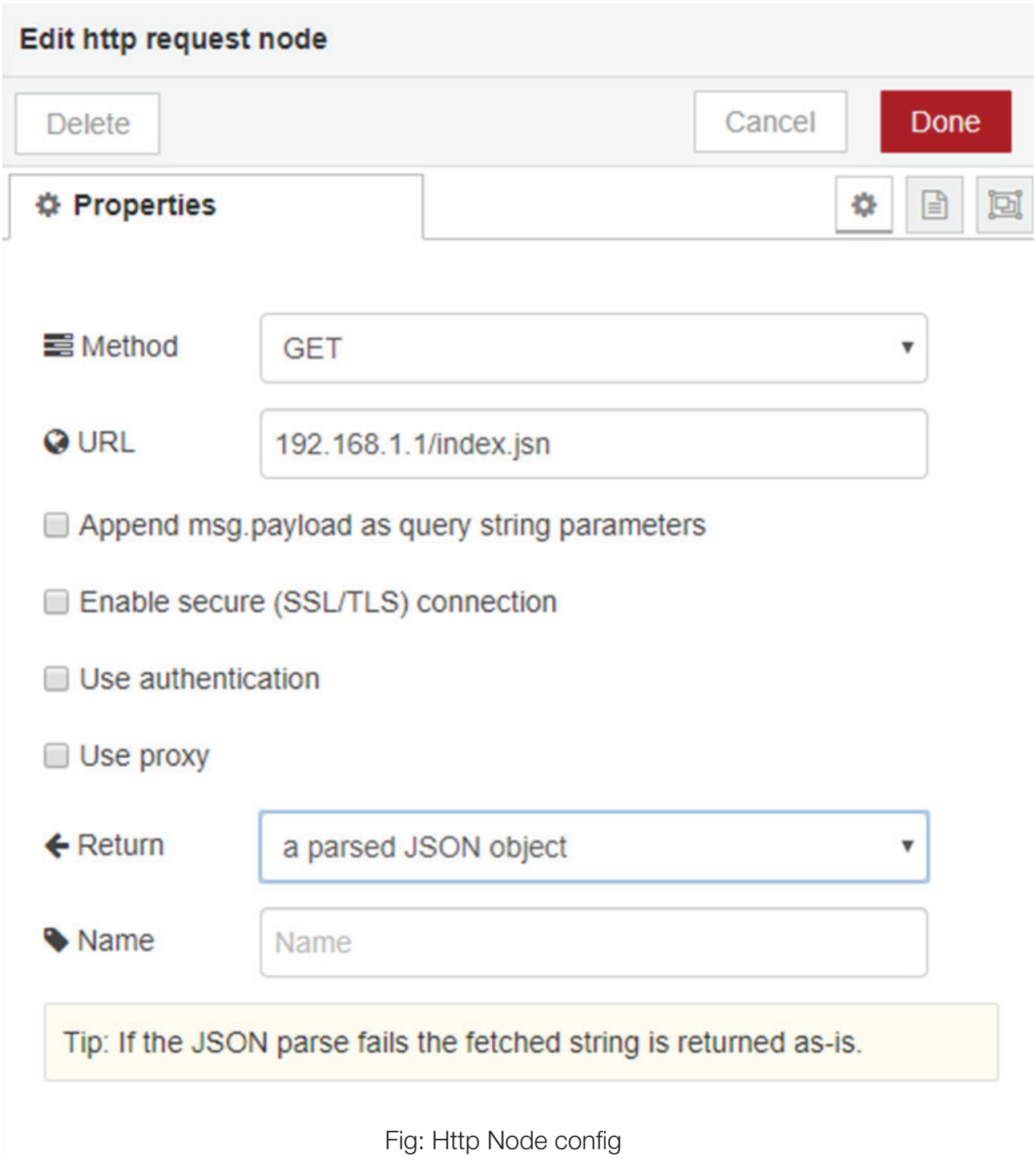


Fig: Http Node config

Connect the node as shown below and configure the debug node to extract the led (note it is JSON object so follows the dot operator).

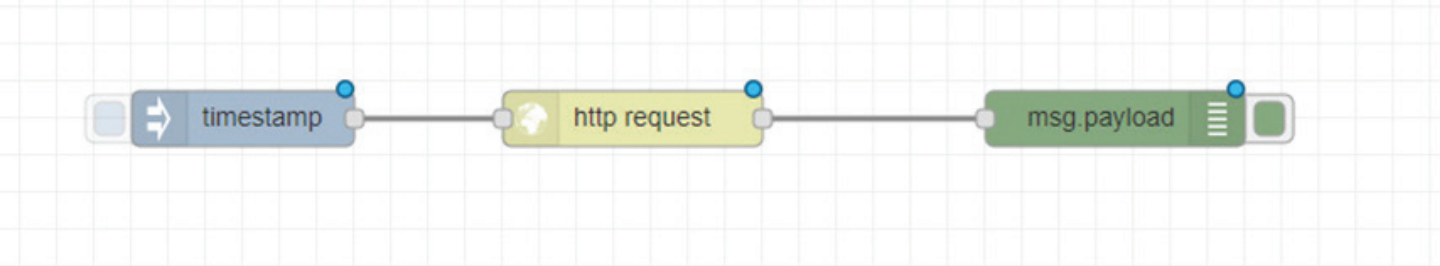


Fig: The wiring for the flow

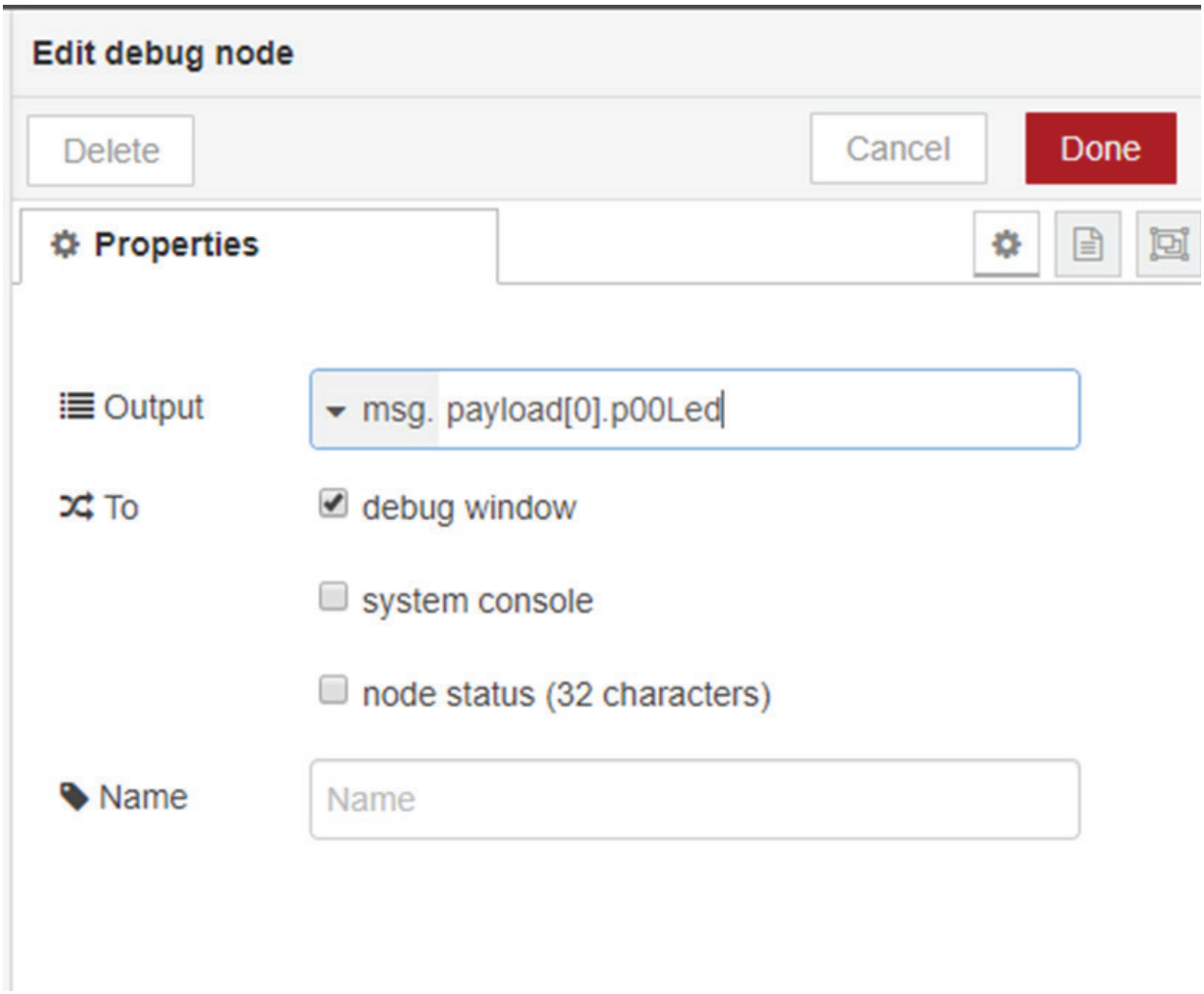


Fig: Debug node setting to extract P00 Led

After you deploy it and inject the timestamp you should see the following output in debug window

Fig: Debug window

Mine says “gnst” which means green static.
It means Its configured in IO-LInk MOde and A device is connected to it. Yours might be different depending on your configuration.

There are other data available on this endpoint like vendor ID and Device Id which can be extracted using dot operator and fed to other applications.

NOTES

USA
Balluff Inc.
8125 Holton Drive
Florence, KY 41042
Phone: (859) 727-2200
Toll-free: 1-800-543-8390
Fax: (859) 727-4823
balluff@balluff.com

www.balluff.com

HOW
TO REACH
US