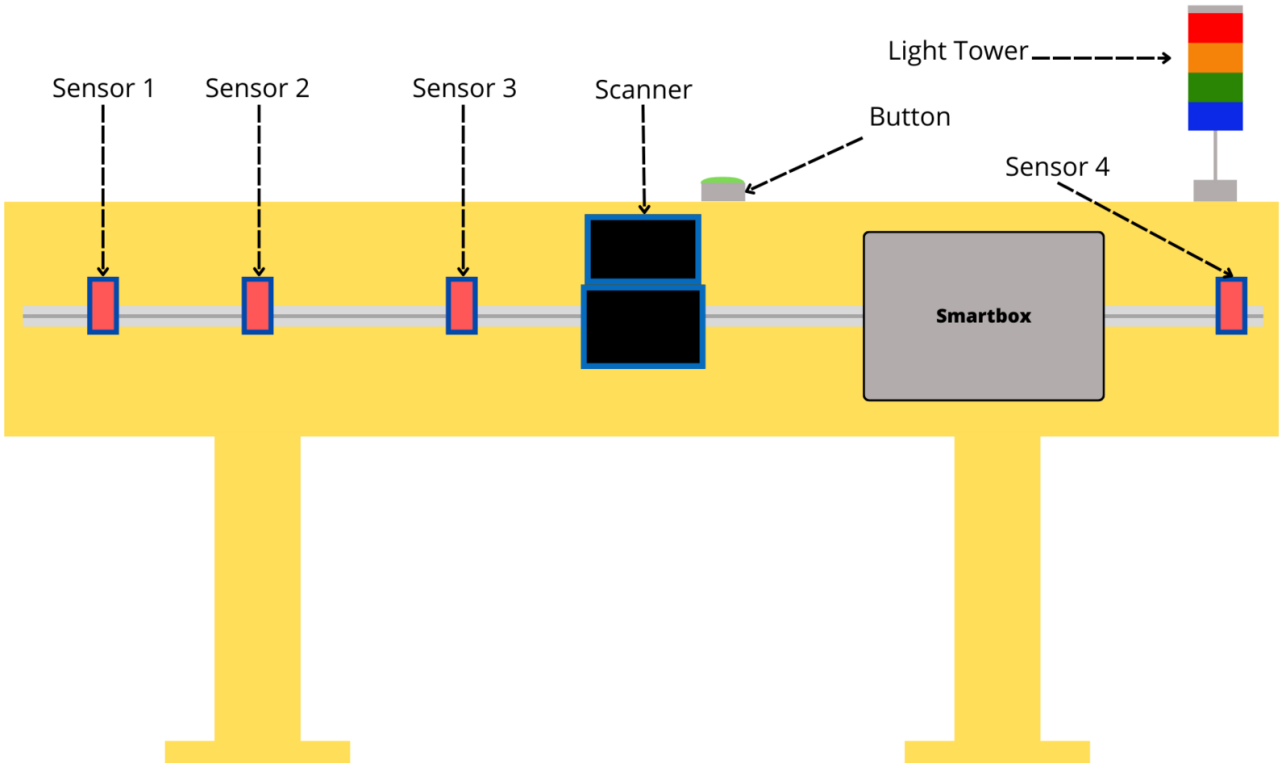
Node-Red flow

General

A loading dock is a system to scan all pallet labels leaving the warehouse via truck. With Node-Red we can add some more intelligence to it so Fenics knows what is going on. It can also check how many pallets that are stacked. The scanner will scan when pallets are entered at the gate. Manual scanning is possible by pressing the button on top off the loading dock.

On this page I will tell you some more about the Node-Red logic that is configured on a loading dock. Our loading dock will be divided in 4 flows. The **Light Tower**, **Sensor's Logic**, **SICK Scanner, Loading screen, CONFIGURATION - Gate** and **KAFKA.**



Flows

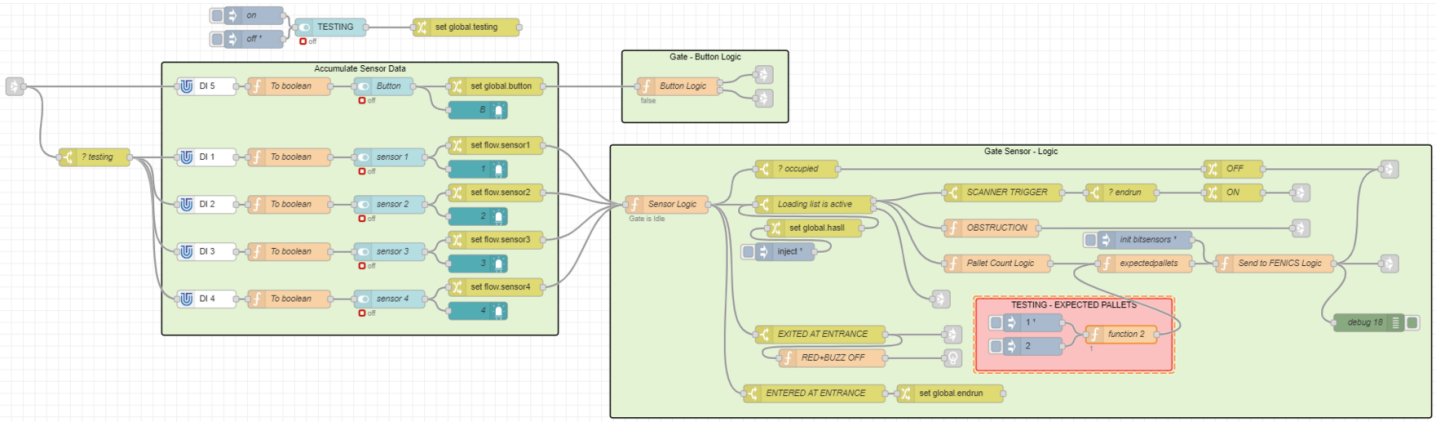
Node-Red flow: https://testwh01gate01.rad.priv.vangenechten.com:1880/#flow/121930c0a19721f7

Node-Red UI: https://testwh01gate01.rad.priv.vangenechten.com:1880/loadingscreen/

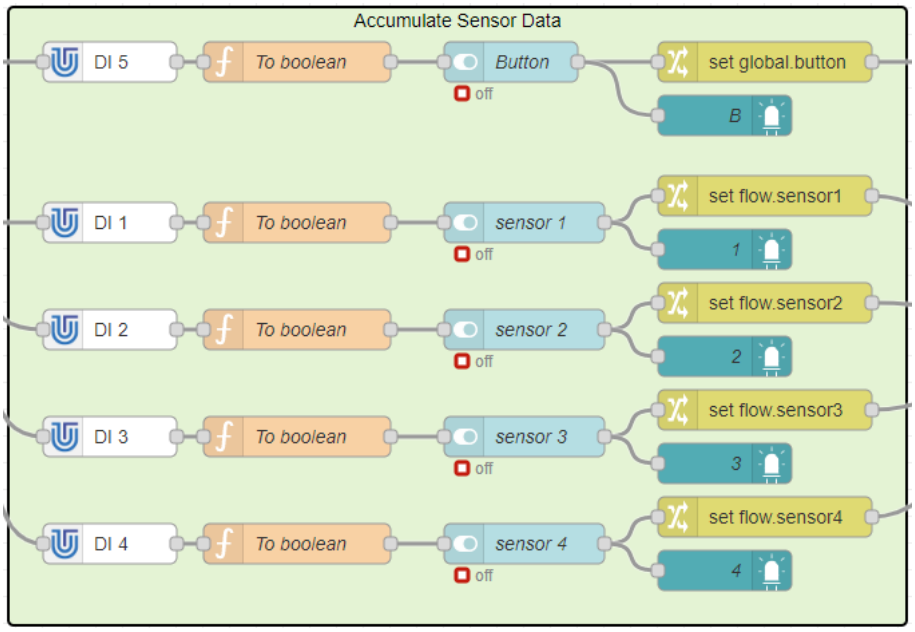
Loading screen IP: 10.1.60.228

Sensor's Logic

This flow can be put in to 3 smaller parts. **Accumulate Sensor Data**, **Gate - Button Logic**, **Gate Sensor - Logic.**



Accumulate Sensor Data



This flow reads the inputs from a **UniPi** extension board for the Raspberry Pi. These inputs are connected to the 4 sensors and the button on top. An input of the sensor is either 1 (no pallet found) or 0 (pallet found). Then a function node is attached that makes a Boolean value of the inputs. It also makes it that **1 is true (on)** and **0 is false (off)** which is more logical. The button sends **true if it is pressed** and **false when not pressed**.

Sensors function

```
return {payload: (msg.payload == 0 ? true : false)}
```

Copy Code

Button function

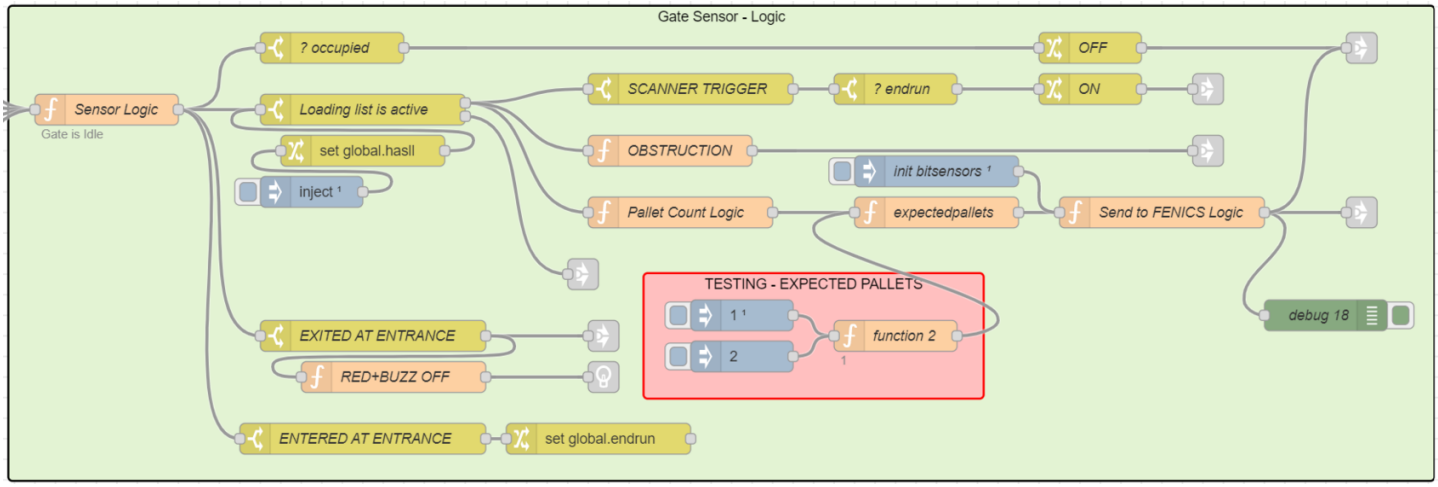
```
return {payload: (msg.payload == 1 ? true : false)}
```

Copy Code

The switches and LED nodes are only used for debugging in the UI. they have no further use.

The set nodes set the sensor values to the payload that is attached to it. All this information is then sent to the sensor logic.

Gate - Sensor Logic



Sensor Logic

- Translating binary to integer
  - 0011 = 3

- Determine which end of the gate was entered/exited, also determine therefor the **occupied state**
  - Ingress = entrance
  - Egress = exit
  - leading sensor = the sensor that was activated first will determine the order of the bits.
  - trailing sensor = the sensor that was activated after the leading sensor was triggered.
- Calculate span
  - amount of sensors that are active.
- Occupied state
  - This state is true when at least 1 sensor is active and false if all of them are inactive. It works like an OR function.
- Breach messages
  - The input is translated into a binary code. There are 4 sensors that can be 1 or 0. So it will have **16 combinations**.
  - The output will not always be the same. It is because the sensor logic will take the **previous state** into account.

Sensor 4	Sensor 3	Sensor 2	Sensor 1	Breach messages
0	0	0	0	Gate is Idle/Exited at exit/ obstruction general
0	0	0	1	Entered at entrance
0	0	1	0	Obstruction General
0	0	1	1	Entered at entrance/obstruction general
0	1	0	0	Obstruction General
0	1	0	1	Obstruction Ingress/General
0	1	1	0	Obstruction General
0	1	1	1	Entered at entrance
1	0	0	0	Entered at exit
1	0	0	1	Obstruction Egress/ Ingress
1	0	1	0	Obstruction Egress/ General
1	0	1	1	Obstruction Egress/ingress/general/Entered at entrance
1	1	0	0	Entered at exit/Obstruction general
1	1	0	1	Entered at exit/obstruction ingress/general
1	1	1	0	Entered at exit/obstruction general/egress
1	1	1	1	Entered at exit/entrance/ Obstruction General/egress/egress

Sensor Logic function

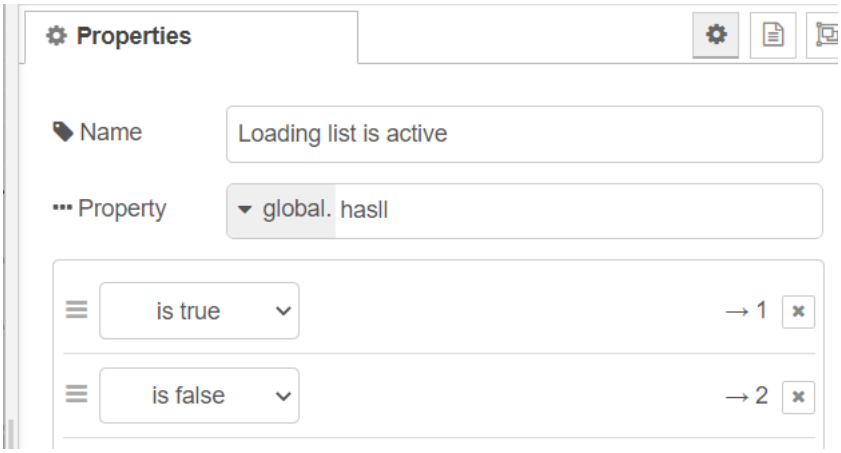
```
// get the previous state                                                    Copy Code
var previousSensors = context.get("previousSensors")

// make a bit addressable sensor integer
```

```
var bitSensors =
    (flow.get("sensor1") ? 1 : 0) +
    (flow.get("sensor2") ? 2 : 0) +
    (flow.get("sensor3") ? 4 : 0) +
    (flow.get("sensor4") ? 8 : 0)

// determine leading sensor
var leading = 0
if (bitSensors & 0b1000) leading = 4
else if (bitSensors & 0b0100) leading = 3
else if (bitSensors & 0b0010) leading = 2
else if (bitSensors & 0b0001) leading = 1

// determine trailing sensor
var trailing = 0
if (bitSensors & 0b0001) trailing = 1
else if (bitSensors & 0b0010) trailing = 2
else if (bitSensors & 0b0100) trailing = 3
else if (bitSensors & 0b1000) trailing = 4

// determine which end of gate was entered/exited
// also determine therefor the occupied state
var ingress=false
var egress=false

var occupied = (bitSensors & 0b1111)==0?false:true

var breachmessage = context.get("previousBreach")

if ((previousSensors == 0b0000) && (bitSensors == 0b0001)) {
    egress=false
    ingress=true
    breachmessage = "Entered at Entrance"
} else if (previousSensors == 0b0000 && bitSensors == 0b1000) {
    egress = true
    ingress = false
    breachmessage = "Entered at Exit"
} else if (previousSensors == 0b0001 && bitSensors == 0b0000) {
    egress = false
    ingress = false
    breachmessage = "Exited at Entrance"
} else if (previousSensors == 0b1000 && bitSensors == 0b0000) {
    egress = false
    ingress = false
    breachmessage = "Exited at Exit"
} else if (previousSensors == 0b0000 && (bitSensors & 0b0110)) {
    egress = false
    ingress = false
    breachmessage = "Obstruction - General"
}

// other obstructions
if (previousSensors == 0b0001 && (bitSensors & 0b1100)) {
    egress = false
    ingress = false
    breachmessage = "Obstruction - Ingress"
} else if (previousSensors == 0b1000 && (bitSensors & 0b0011)) {
    egress = false
    ingress = false
    breachmessage = "Obstruction - Egress"
}

// calculate the span
var span= 0
if (bitSensors == 0) {
    span=0
} else {
    span = leading - trailing + 1
}

// setup leaving criterea
context.set("previousSensors", bitSensors)
context.set("previousBreach", breachmessage)

// be nice structuring your message
msg = {

    topic : "Sensor Logic",

    payload : {
        gatename: global.get("gatename"),
        occupied:   occupied,
        breachmessage: breachmessage,
        ingress:    ingress,
        egress:     egress,

        sensors:{
            bitSensors: bitSensors,
            leading: leading,
            trailing: trailing,
            span: span,
        },

        lights:{
            white: global.get("lightwhite"),
```

```
            blue: global.get("lightblue"),
            green: global.get("lightgreen"),
            orange: global.get("lightorange"),
            red: global.get("lightred"),
        },

        loadinglist: global.get("ll")
    }
}

node.status({ text: breachmessage})

return msg;
```

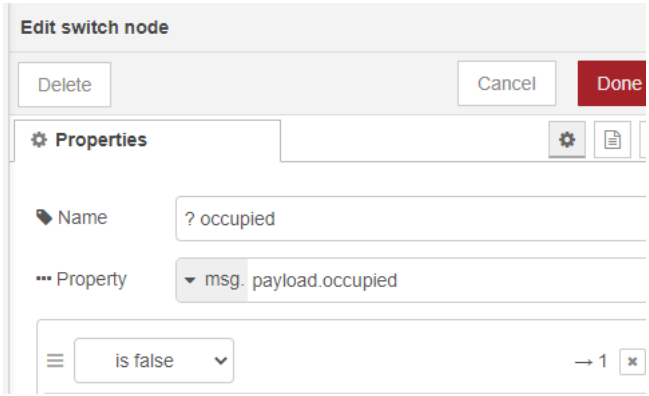
Loading list active

The global **hasll** (has loading list) variable will be checked, if **True** the output will be activated.

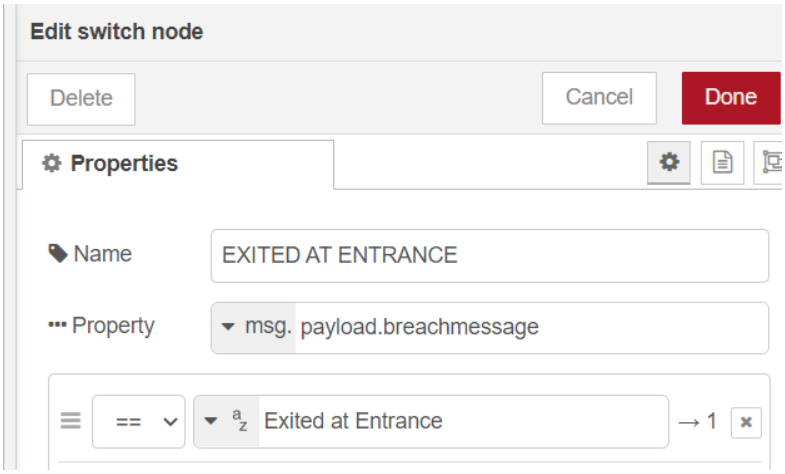


? occupied

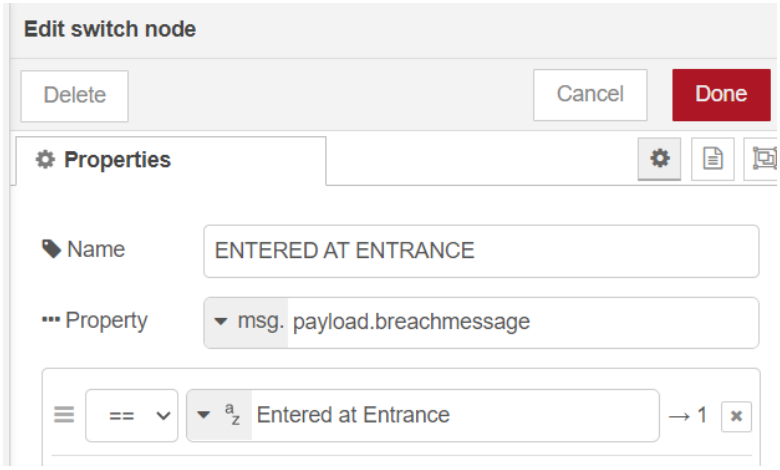
This state is true when at least 1 sensor is active and false if all of them are inactive. It works like an OR function. It will trigger an **alarm** by turning OFF the scanner, activating the red lamp and buzzer.



EXITED AT ENTRANCE

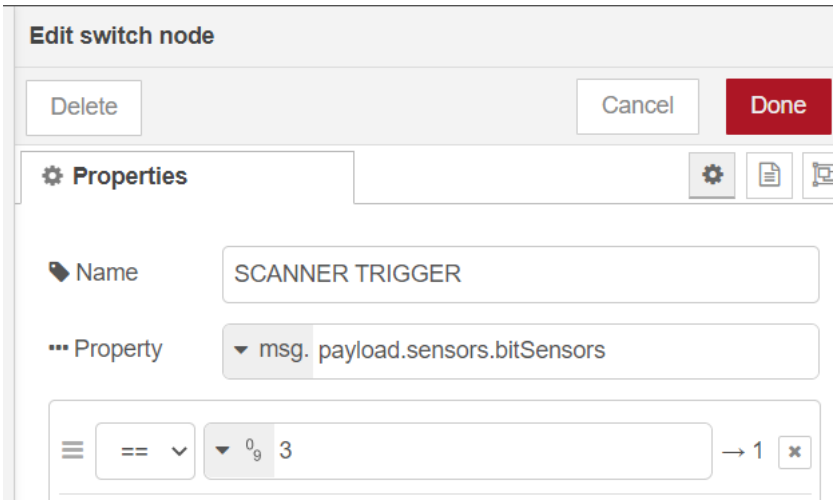


ENTERED AT ENTRANCE



Scanner trigger

As previously mentioned the input is translated into a binary code. This statement will be true if sensor 1 and 2 (entrance) or 4 and 3 (exit) are active.



Obstruction

Obstructions are sent to the screen when the breach message starts with "Obstruction".

```
node.status({ text: msg.payload.breachmessage})

msg.payload = msg.payload.breachmessage.startsWith("Obstruction")


return msg;
```

Copy Code

Pallet count logic

This node calculates how many pallets that are scanned with a switch method.

bitSensors	counterstate	count message	count value
0000	gateActive	/	/
0011	gateActive -> gateEntered	/	/
0011	oneOrTwoPallets	1 pallet (normal)	1
0111	gateEntered -> gateCounting	/	/
0111	oneOrTwoPallets	1 pallet (meyzieu forklift)	1
1100	gateCounting	1 pallet (small)	1
1111	gateCounting	2 pallets	2
1111	oneOrTwoPallets	2 pallets	2
1011	gateCounting	1 pallet (large)	1
0100	gateCounting	1 pallet (normal)	1
0100	oneOrTwoPallets	1 pallet (normal)	1
0110	gateCounting-> oneOrTwoPallets	/	/

Copy Code

```
// Retrieve the current state of the counter
var counterstate = context.get("CounterState")
var countmessage = context.get("CounterMessage")
var countvalue = context.get("CounterValue")

// Process the input from sensors
switch (msg.payload.sensors.bitSensors) {

    case 0b0000 :
        // If all sensors are off, set the counter state to "gateActive"
        context.set("CounterState", "gateActive")
        countmessage = ""
        countvalue = 0
        break

    case 0b0011:
        // If sensors 1 and 2 are on
        switch (counterstate) {
            case "gateActive":
                // If the counter state is "gateActive", set it to "gateEntered"
                context.set("CounterState", "gateEntered")
                break
            case "oneOrTwoPallets" :
            // If the counter state is "oneOrTwoPallets", set the count message to "1 pallet (normal)" and the count value to 1
                countmessage = "1 pallet (normal)"
                countvalue = 1
                break
            default:
                break
        }
        break

    case 0b0111:
        // If sensors 1, 2 and 3 are on
        switch (counterstate) {
            case "gateEntered":
                // If the counter state is "gateEntered", set it to "gateCounting"
                context.set("CounterState", "gateCounting")
                break
            case "oneOrTwoPallets":
                // If the counter state is "oneOrTwoPallets", set the count message to "1 pallet (meyzieu forklift)" and the count value to 1
                countmessage = "1 pallet (meyzieu forklift)"
                countvalue = 1
                break
            default:
                break
        }
        break

    case 0b1100:
        // If sensors 3 and 4 are on
        switch (counterstate) {
            case "gateCounting":
                // If the counter state is "gateCounting", set the count message to "1 pallet (small)" and the count value to 1
                countmessage = "1 pallet (small)"
                countvalue = 1
                break
            default:
                break
        }
        break

    case 0b1111:
        // If all sensors are on
```

```
        switch (counterstate){
            case "gateCounting":
                // If the counter state is "gateCounting", set the count message to "2 pallets" and the count value to 2
                countmessage = "2 pallets"
                countvalue = 2
                break
            case "oneOrTwoPallets":
                // If the counter state is "oneOrTwoPallets", set the count message to "2 pallets" and the count value to 2
                countmessage = "2 pallets"
                countvalue = 2
                break
            default:
                break
        }
        break

    case 0b1011:
        // If sensors 1, 3 and 4 are on
        switch (counterstate) {
            case "gateCounting":
                // If the counter state is "gateCounting", set the count message to "1 pallet (large)" and the count value to 1
                countmessage = "1 pallet (large)"
                countvalue = 1
                break
            default:
                break
        }
        break

    case 0b0100:
        // If sensor 3 is on
        switch (counterstate) {
            case "gateCounting":
                // If the counter state is "gateCounting", set the count message to "1 pallet (normal)" and the count value to 1
                countmessage = "1 pallet (normal)"
                countvalue = 1
                break
            case "oneOrTwoPallets":
                // If the counter state is "oneOrTwoPallets", set the count message to "1 pallet (normal)" and the count value to 1
                countmessage = "1 pallet (normal)"
                countvalue = 1
                break
            default:
                break
        }
        break

    case 0b0110:
        // If sensor 2 and 3 is on
        switch (counterstate) {
            case "gateCounting":
                // If the counter state is "gateCounting", set it to "oneOrTwoPallets"
                context.set("CounterState", "oneOrTwoPallets")
                break
            default:
                break
        }
        break

    default:
        break
}

node.status({ text: context.get("CounterState") + " - " + countmessage})

context.set("CounterMessage", countmessage)
context.set("CounterValue", countvalue)

global.set("excpectedpallets", countvalue )

return msg;
```

expectedpallets

Copy Code

```
if (global.get("testing")) {
    global.set("expectedpallets", flow.get("testexpectedpallets"))
}

node.status({  text: global.get("expectedpallets")})

return msg;
```

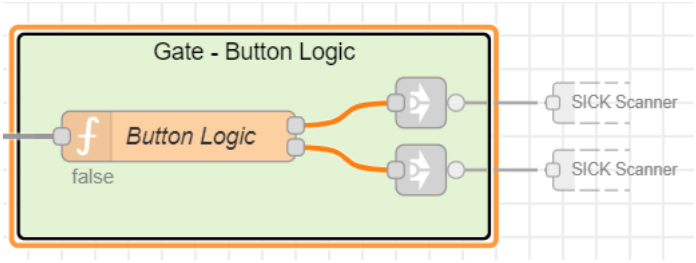
Send to FENICS Logic This node sends the scanned codes to fenics.

Copy Code

```
    switch(msg.payload.sensors.bitSensors) {
        case 0b1000:
            return msg

        default:
            break
```

```
}
```

Gate - Button Logic



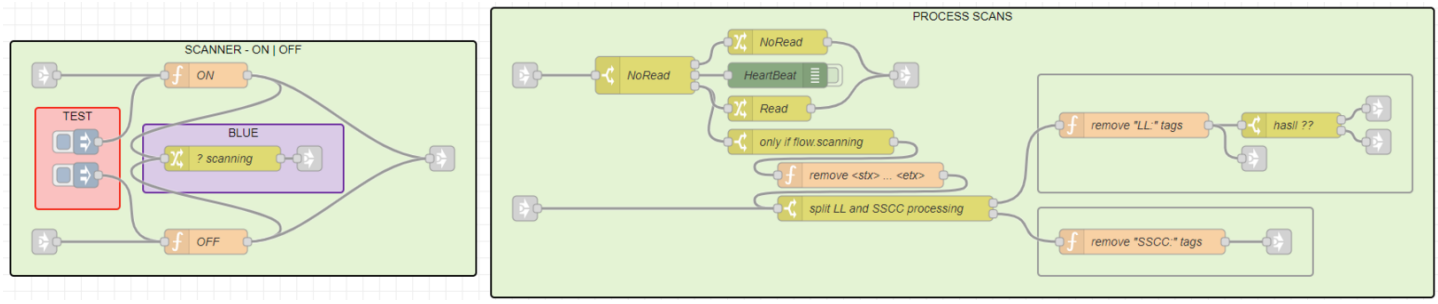
Button Logic

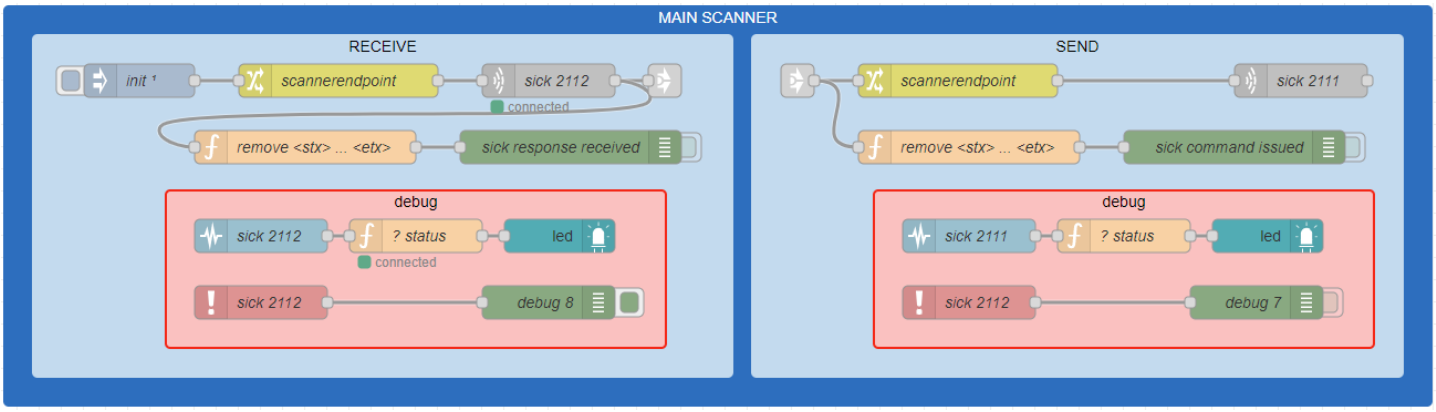
When the button is pressed. The scanner is ON. When not pressed or released, it will be OFF.

```
node.status({ text: global.get("button")})                                    Copy Code


if(global.get("button")) {
    return  [msg, null]
} else {
    return [null,msg]
}
```

SICK Scanner

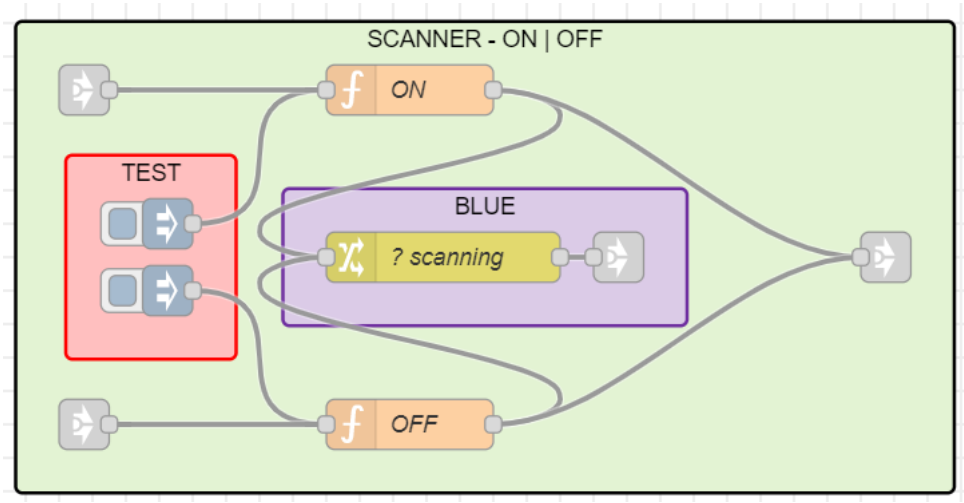
This flow is used to scan codes when they are needed. It splits LL and SSCC codes and sends them to the correct endpoint.





Scanner - ON/OFF

This is a module that is used to turn the scanner on and off. When it is scanning the blue light will be switched on.



ON

This code sends a start scanning command.

```
if( global.get("scanning") == false ){

    global.set("scanning", true);
    global.set("scans", []);

    msg.payload = "\x02sMN mTCgateon\x03"; //Start scanning command
    return msg;

}
```

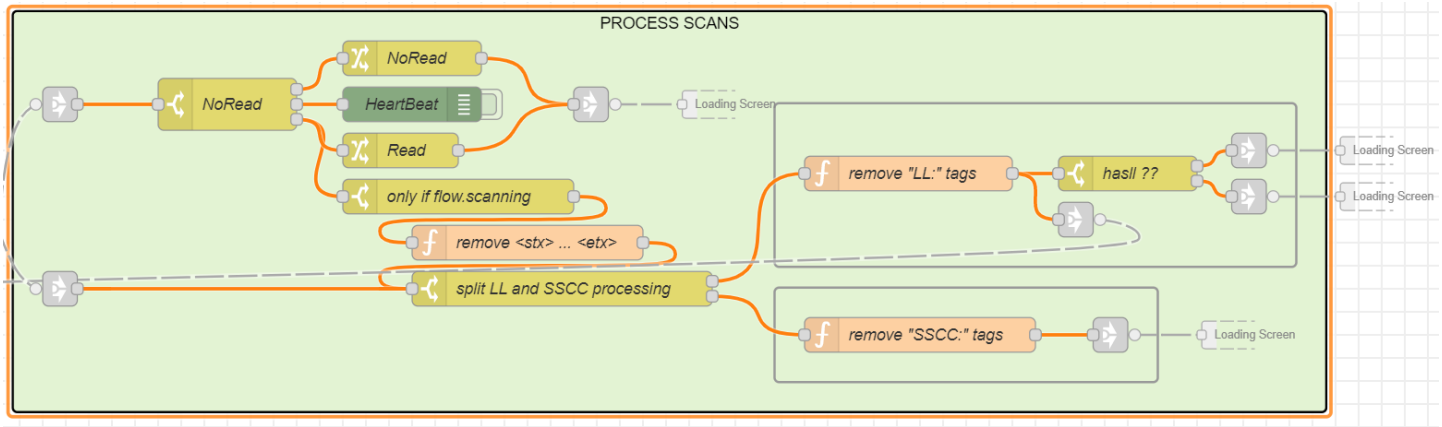
Copy Code

OFF This code sends a stop scanning command.

```
if (global.get("scanning") == true) {

    global.set("scanning", false);

    msg.payload = "\x02sMN mTCgateoff\x03"; //stop scanning command
    return msg;
}
```

Copy Code

Process scans



This flow processes the scans. When the scanner receives a message, a switch node will determine if it has a good read.

A **NoRead** payload is True and will be sent to the loading screen.

If the global scanning variable is True, it will go through.

remove <stx> ... <etx>

A function node sends scanned payload when there is a good read.

```
if (msg.payload != "NoRead" && msg.payload != "HeartBeat") {

    //  remove <stx> ...  <etx>
```

Copy Code

```
    msg.payload = msg.payload.slice(1, -1);
    return msg;
}
```

In the next node scanned codes are split into LL codes (loading list) and SSCC (barcodes) **remove "LL:" tags**

```
msg.payload = msg.payload.slice(6).slice(0,6);
return msg;
```

Copy Code

remove "SSCC:" tags

```
msg.payload = msg.payload.slice(5);
return msg;
```

Copy Code

All this information is then sent to the loading screen.

Main scanner

One flow that receives and one that sends codes.

Receive

IP: 10.1.61.22:2112

The receive flow send a link out to **Process scans** that determines the if scans are good or not.

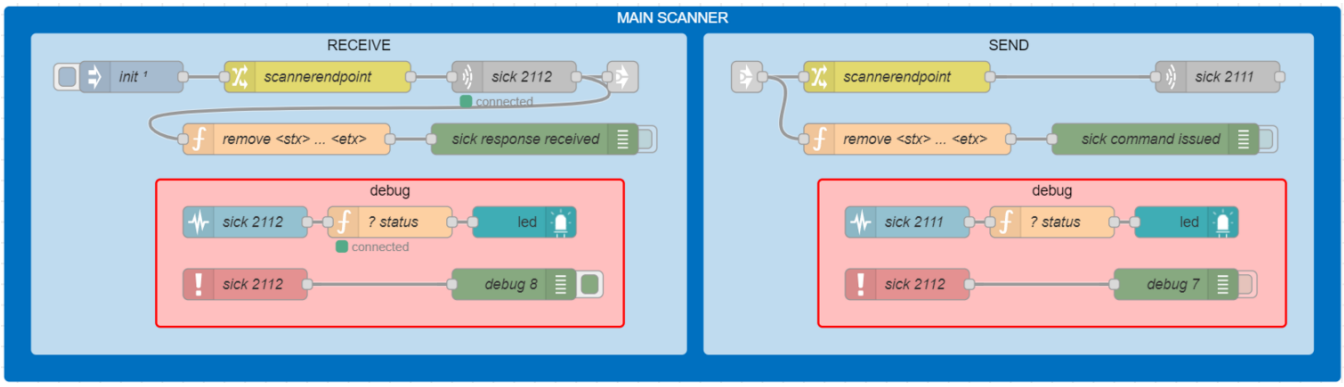
The SICK scanner sensor sends its data to the **receive port 2112**

Send

IP: 10.1.61.22:2111

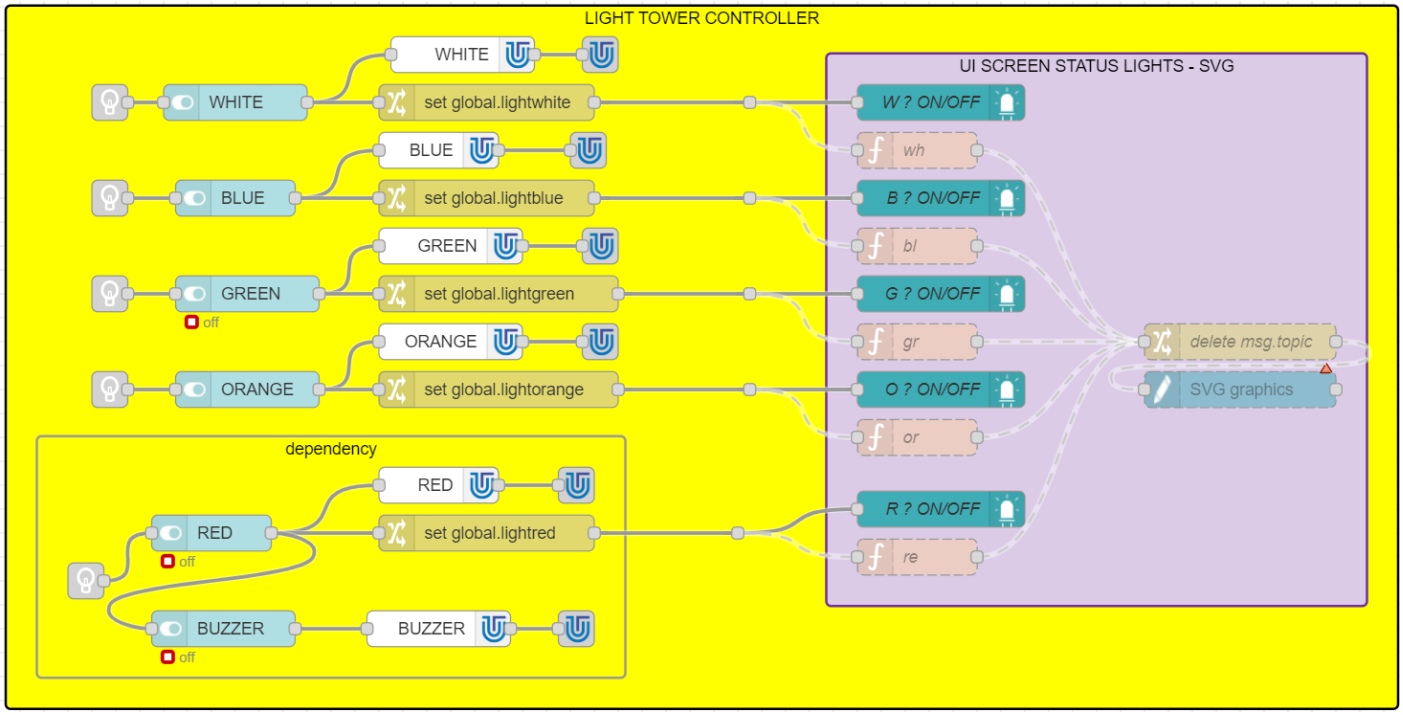
The send link in is the command that was sent by the scanner. It turns on or off, according to the command that was sent.

The SICK scanner sensor sends its data to the **send port 2111**



Light Tower

The lights are can be controlled by the switches or outputs of the other flows. As you can see on the bottom. When the red lamp is triggered, the buzzer automatically turns on to be even more noticeable.

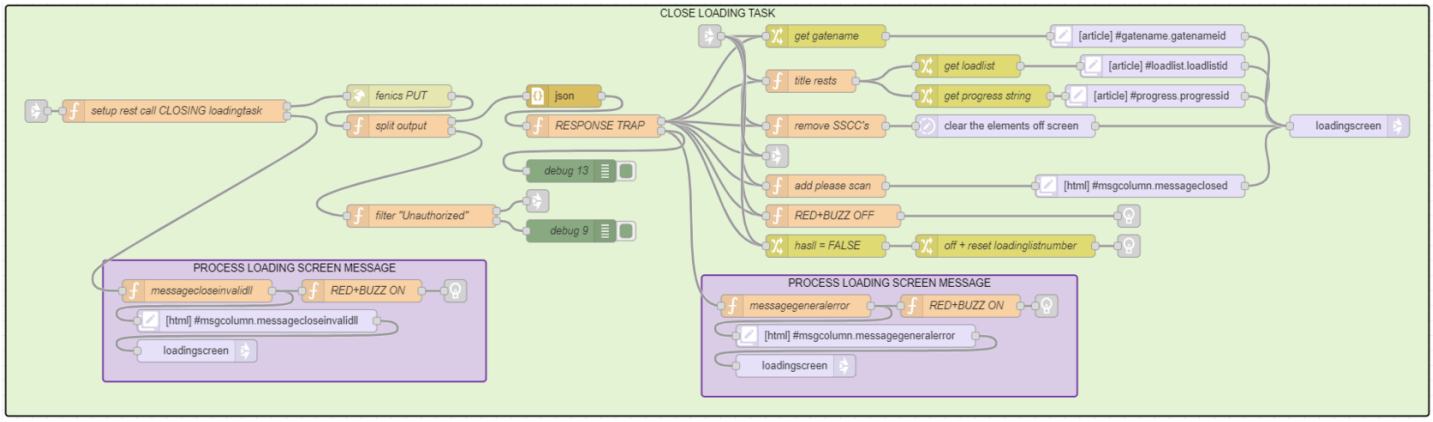

LIGHT TOWER CONTROLLER
WHITE
WHITE
set global.lightwhite
BLUE
BLUE
set global.lightblue
GREEN
GREEN
set global.lightgreen
off
ORANGE
ORANGE
set global.lightorange
dependency
RED
RED
set global.lightred
off
BUZZER
BUZZER
off
UI SCREEN STATUS LIGHTS - SVG
W ? ON/OFF
wh
B ? ON/OFF
bl
G ? ON/OFF
gr
O ? ON/OFF
or
R ? ON/OFF
re
delete msg.topic
SVG graphics

LIGHTS	Actions
White (in the future)	Power on
Blue	Scanning
Green	Loading List is closed/ no loading list
Orange	Obstruction
Red	Alarm/error

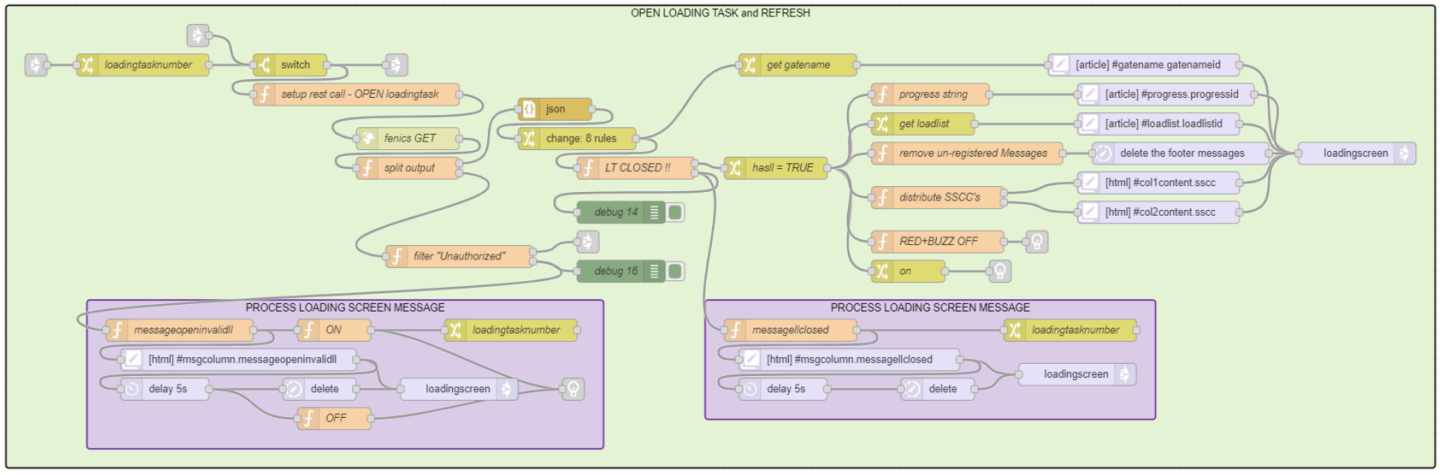
Loading screen

The loading screen shows all the useful information about the loading list. It has a lot of functionalities.

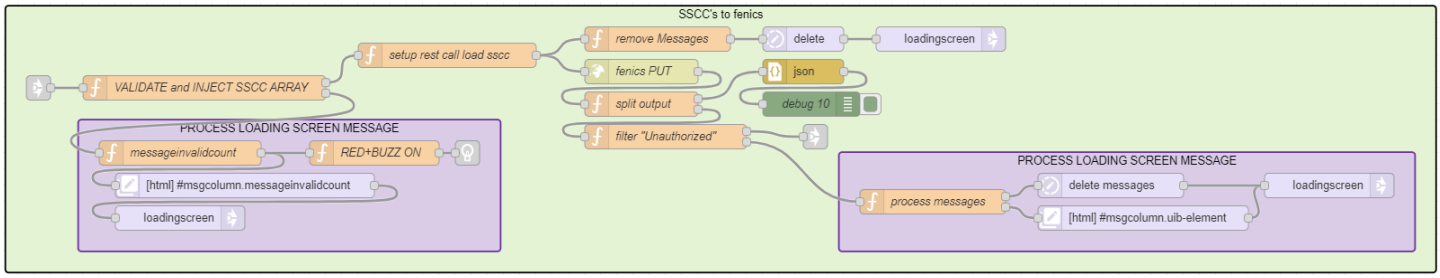
Close loading task



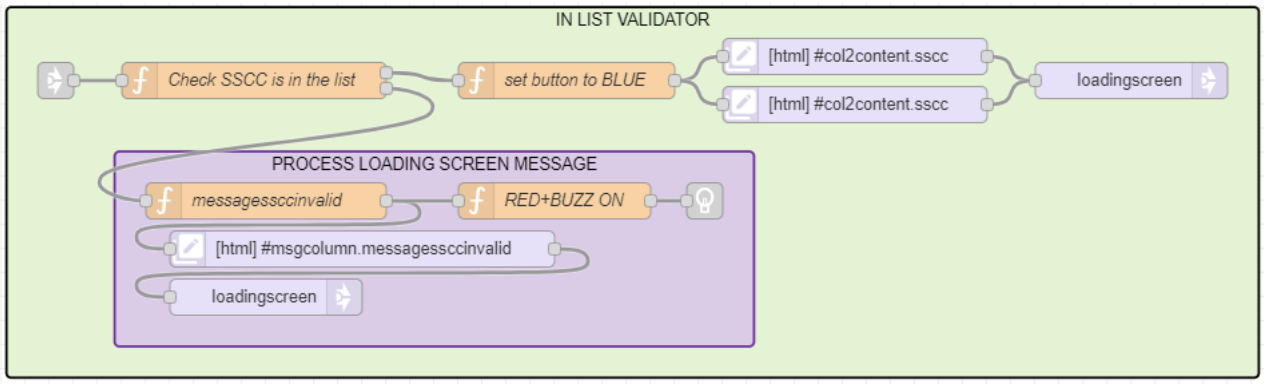
Open loading task and refresh



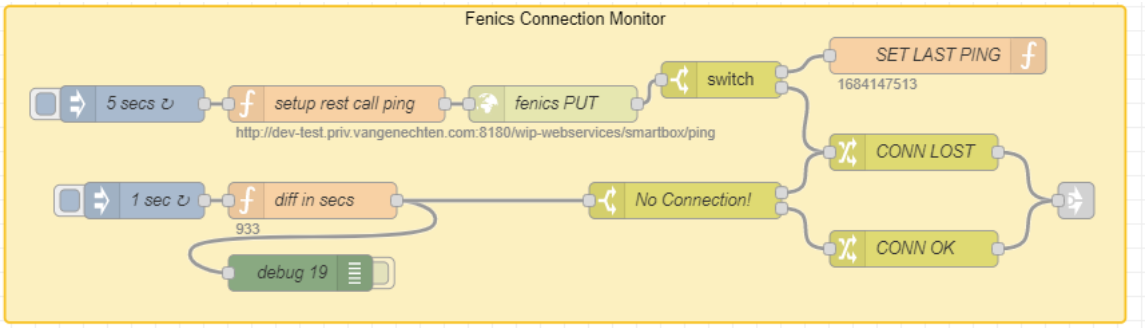
SSCC's to FENICS



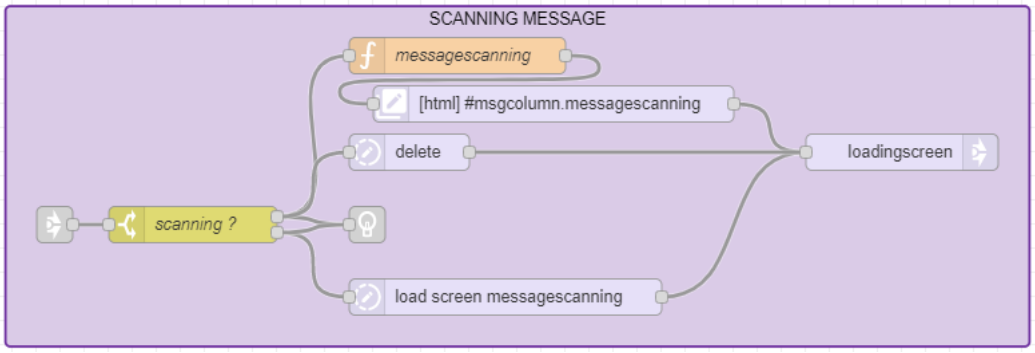
In list validator



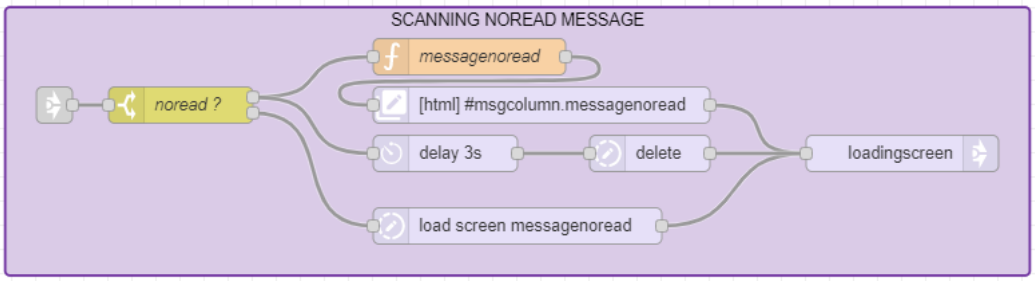
Fenics connection monitor



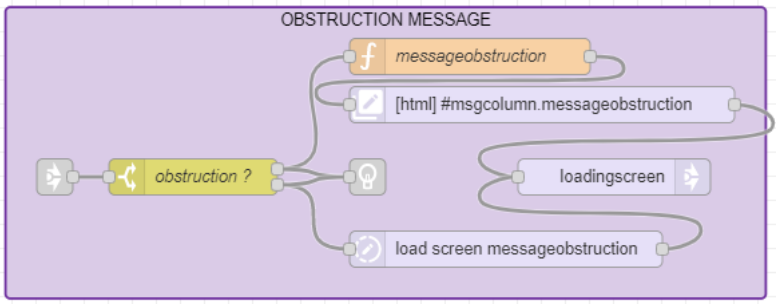
Scanning message



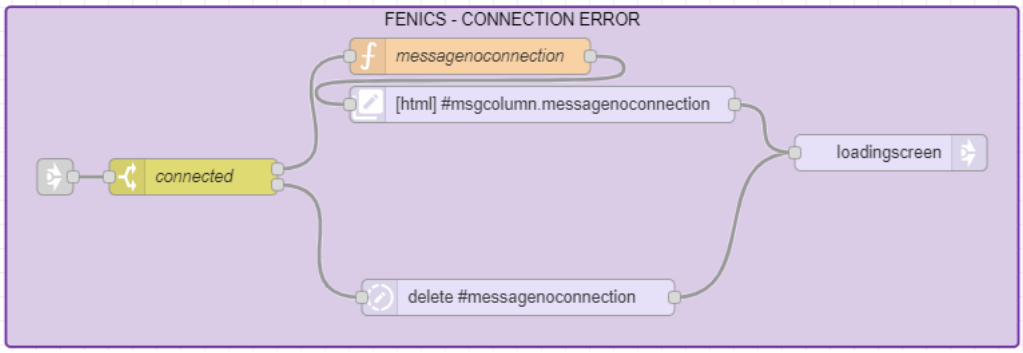
Scanning noread message



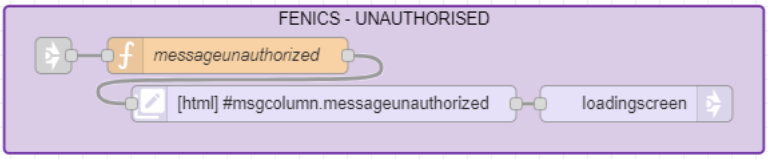
Obstruction message



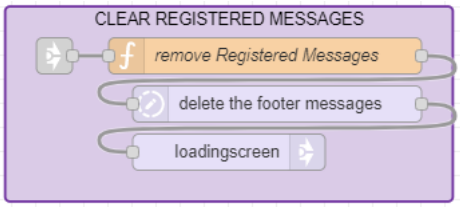
Fenics connection error



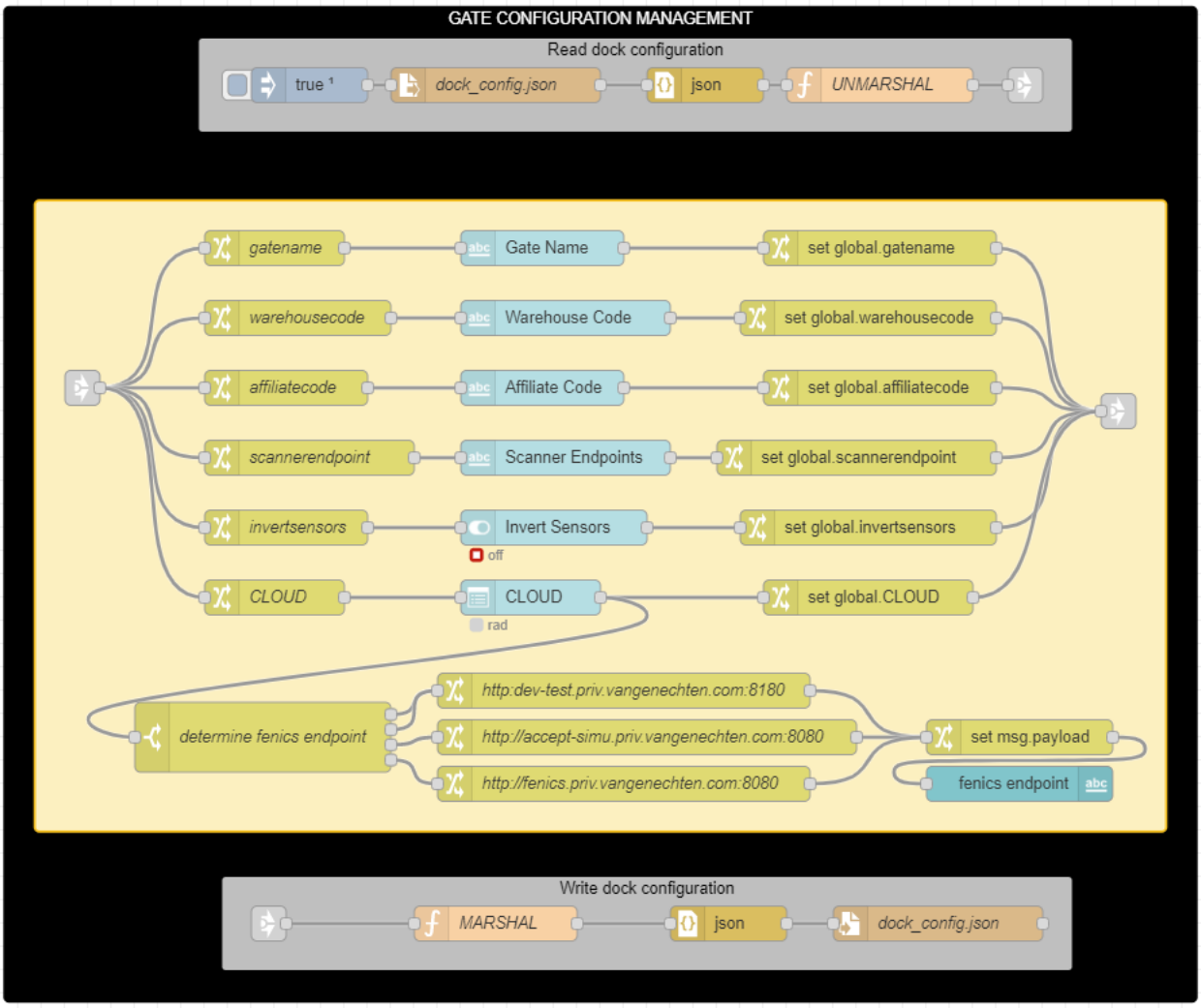
Fenics unauthorised



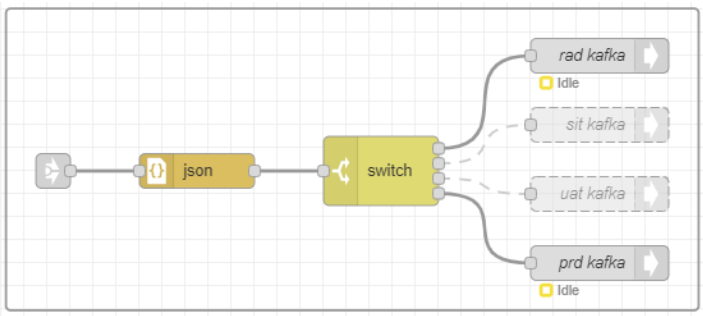
Clear registered messages

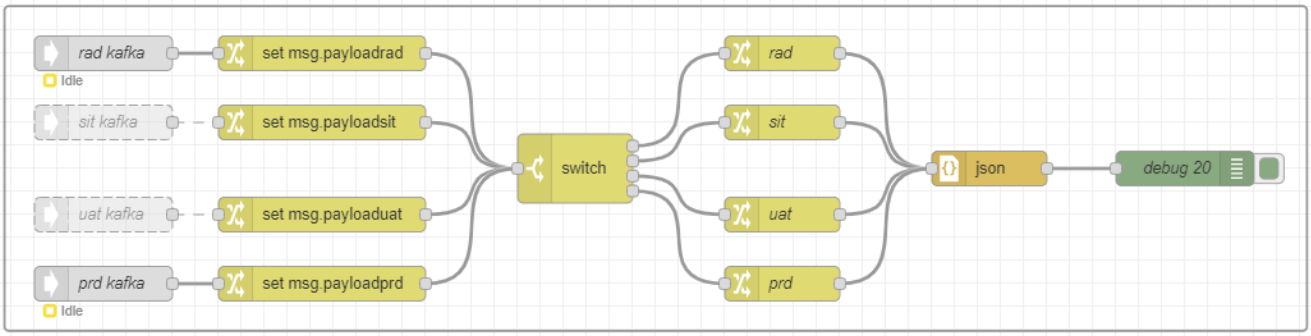


CONFIGURATION - Gate



KAFKA





FENICS

Loading Task Control:
http://git0301.priv.vangenechten.com/development/fenics/blob/dev/systems/finishedgoods/web/src/main/java/com/vangenechten/system/finishedgoods/ecb/contrc

Loading Dock Control:
http://git0301.priv.vangenechten.com/development/fenics/blob/dev/systems/skumanagement/web/src/main/java/com/vangenechten/system/skumanagement/ecb/c

Smart Box Resource:
http://git0301.priv.vangenechten.com/development/fenics/blob/dev/systems/wip/web/src/main/java/com/vangenechten/system/wip/ecb/control/SmartBoxResource

---

Back to: Loading Docks for VGPIoT (CDP)
Back to: Main Page

LoadingDock VGPIoT