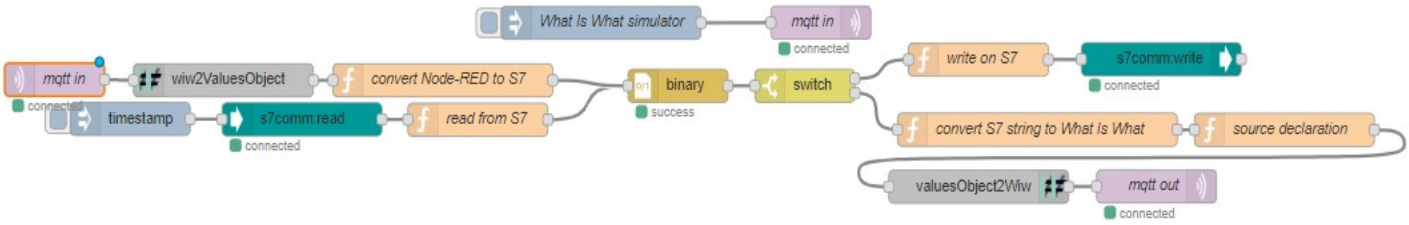
# Introduction

This document explains how to get *What Is What* communicate with binary-based devices (modbus, S7, etc).

# The flow




## Third party dependencies

A *Mqtt broke*r must be running. See *mqttBroker* file for more information.

In this example we do communication with binary-based *Siemens S7 PLC*. This is just an example, and it would work as well with any binary-based devices (for example a modbus-enabled PLC). Instead of using *node-red-contrib-s7*, we use here *node-red-contrib-s7-comm* because it's using PLC's raw buffers.

To test this example, we recommend to use a *S7 PLC*, but a simulator may be good enough. You can find one in *Snap7* library, available here https://freefr.dl.sourceforge.net/project/snap7. Snap7 v1.4.1 has been successfully tested.

## Explanation

This flow demonstrates how to use binary nodes to read/write buffers from/to *Siemens S7 PLC*. The same kind of solution could be used with *modbus* or any other communication device based on binary data (with adaptation to specific data format like string, date,...).

Our goal is to handle the following PLC's data structure.

| Name | Offset | Size (byte) | Type |
|---|---|---|---|
| qtyMachine | 1 | 4 | Double integer |
| pauseMachine | 5 | 1 | Byte |
| errorMachine | 6 | 2 | Integer |
| speed | 8 | 2 | Integer |
| Reserved | 10 | 1 | |
| id | 11 | 52 (2 bytes header + 50 characters) | String |
| desc | 63 | 92 (2 bytes header + 90 characters) | String |
| prodId | 155 | 52(2 bytes header + 50 characters) | String |
| qty | 207 | 4 | Double integer |
| qtyProduced | 211 | 4 | Double integer |
| start | 215 | 18 (2 bytes header + 16 characters) | String |
| end | 233 | 18 (2 bytes header + 16 characters) | String |

**Binary node**

This node builds a buffer from an object or vice-versa.

While creating the template, keep in mind that variables are following each others, that means there is no offset.

Numbers are easy to declare: “b” for big-endian, “l” for little-endian and the size in bit(s) (see *javascript packet* documentation for more information: http://bigeasy.github.io/packet).

On *Siemens S7 PLC*, data are :

- stored using big-endian,
- strings are made of a 2 bytes header (1st byte: max string’s length, 2nd byte: current string’s length) followed by the string’s characters in ASCII. This header will have to be extracted when reading and created when writing. Further more, when strings don’t fill their whole space, they may be filled with space characters (32 ASCII code).

To create a string buffer, there are 2 different possibilities:

- what we call “normal mode” : do not fill the unused characters with “space”,
- what we call the “fill space mode” : fill unused characters with “space”.

Here is the binary’s setting when s7 node is set to read from byte 1, for both modes :

| Name | Normal mode | Fill space mode |
|---|---|---|
| qtyMachine | b32 | |
| pauseMachine | b8 | |
| errorMachine | b16 | |
| speed | b16 | |
| reserved | x8 | |
| id | b8[52] | b8[52]{32}z |
| desc | b8[92] | b8[92]{32}z |
| prodId | b8[52] | b8[52]{32}z |
| qty | b32 | |
| qtyProduced | b32 | |
| start | b8[18] | b8[18]{32}z |
| end | b8[18] | b8[18]{32}z |

## *First line of the flow (see above)*

The first line simulates a message from *What Is What.* It is needed only in this example, and is not relevant in production. The inject node sends a *JSON* object that could have been sent by *What Is What*.

## Second line *of the flow (see above)*

The second line shows the reception of a message from a *Mqtt broker*. The message is sent to a *Wiw2ValuesObject* node which will cast it into a new message (see *wiw2ValuesObject* documentation). Then *convert Node-RED to S7* casts it into a message compatible with *binary* node, which creates a buffer compatible with S7 PLC.

The *inject* node is used to read current data from *PLC*. The *binary* node then casts it into a JSON object compatible with *valuesObject2Wiw* (see *valuesObject2Wiw* documentation).

### *Note*

Please make sure to use the right *Mqtt* broker settings in the *Mqtt* client setup.