|  |
| --- |
| Basics of LuvitRED |
|  |



**Franco Arboleda**

**08-Jul-15**

Table of Contents

1 What is LuvitRED? 3

1.1 What is the benefit of using LuvitRED? 3

1.2 Who is intended for? 3

1.3 Where to get LuvitRED from and how to install it? 4

2 LuvitRED editors 5

2.1 Basic interface 5

2.2 Advanced Editor 10

3 What is a node? 14

3.1 Types of nodes 14

3.2 Inject and Debug nodes 15

4 What is a flow? 17

5 What is a message? 18

5.1 Single message structure 18

5.2 Combined message structure 19

# What is LuvitRED?

A visually configurable device agent that is part of the CloudGate solution. The visual editor of LuvitRED is based on the User Interface from IBM's NodeRed. The NodeRed server is rewritten in a Lua server called Luvit.

Why Lua?

Lua is a lightweight, simple to learn programming language that has and easy to use native C interface that makes integration of C libraries relatively simple.

An extensive node set has been developed with M2M use cases in mind. Examples of common nodes include serial, modbus, connection to M2M servers, GPIOs, GPS, etc. The list of nodes is continuously increasing with every new LuvitRED release.

A simple web interface is available for simple configurations such as serial port to TCP local or remote server and GPS to TCP local or TCP/UDP remote server. An advanced editor is available for any other custom configuration that is not cover on the simple web interface. For example, using the two serial ports of the CloudGate's industrial serial card at the same time.

The LuvitRED configuration is stored on the CloudGate´s file system (UCI). This means that the same configuration can be propagated to other CloudGates via CloudGate Universe.

The development of LuvitRED is done in house by Option using **only** the CloudGate´s SDK.

## What is the benefit of using LuvitRED?

LuvitRED is intended to reduce time to solution, sales cycle, solution cost and risk of making a new development from zero every time a new project comes.

## Who is intended for?

* Support staff at system integrators/VARs
* Developers
* Option staff

## Where to get LuvitRED from and how to install it?

LuvitRED can be installed remotely on the CloudGate by using the CloudGate Universe server (http://cloudgate.option.com/), but can also be installed manually on the CloudGate by using the CloudGate´s web interface under the provisioning tab.

The LuvitRED package for manual upgrade can be downloaded from the CloudGate Universe by going into the Library> Applications (View applications)> Option LuvitRED (View details) and clicking on the download button located next to the desired LuvitRED version:



Figure : Option LuvitRED download location on CGU.

The latest version of LuvitRED is (as of July 1, 2015).

Downloading the file will create a bin file on your computer, this bin file can be directly uploaded into the CloudGate using the provisioning tab. A video explaining how to install LuvitRED is available on the following link: <https://option.wistia.com/medias/fr4qiimpu7>

Notes:

* Be aware that LuvitRED has a minimum CloudGate firmware requirement in order to work correctly. The minimum CloudGate firmware version for LuvitRED to work correctly is 1.44.0.
* LuvitRED v 1.x.x is only compatible with CloudGate firmware versions 1.x.x. A new version of LuvitRED is being develop (2.x.x), this version will only be compatible with CloudGate firmware versions 2.x.x.
* If the Plugin page does not show after uploading LuvitRED and rebooting the CloudGate, and/or the Image version under System Information on the Home page is shown in red. It reflects that the combination of CloudGate firmware and LuvitRED is not correct. Please, verify that the two previous notes have been followed.

# LuvitRED editors

As mentioned on the introduction, there are two web interfaces for LuvitRED, one basic interface and one advanced interface. In this section we are going to describe the general features of both interfaces.

Both interfaces are located on the Plugin tab under Serial and GPS settings:



Figure : Plugin tab, Serial and GPS settings.

## Basic interface

The basic interface of LuvitRED is the one shown after entering the Serial and GPS settings page. Without any configuration, the basic interface looks as follows:

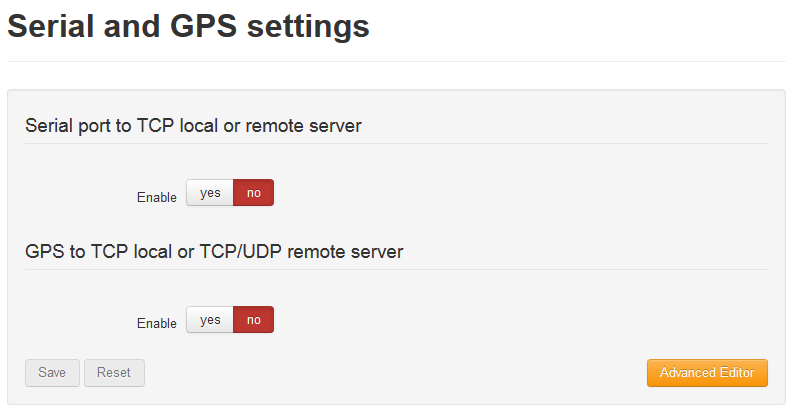


Figure : Basic interface.

Under the basic interface one has two options:

1. Serial port to TCP local or remote server

This section allows the configuration of one single serial port (RS232 - /dev/ttySP0) to be accesible remotely via a local TCP server running on the CloudGate (See Figure 4) or a remote TCP server running on another location (See Figure 5).



Figure : Serial to local TCP server.



Figure : Serial to Remote TCP server.

This configuration is perfect for using with the basic serial expansion card, telematics card (RS232) and the Industrial serial card (RS232).

This configuration is actually replicated on the Advanced editor:



Figure : Same configuration under Advanced editor.

1. GPS to TCP local or TCP/UDP remote server

This section allows the configuration of the GPS to be accesible remotely via a local TCP server running on the CloudGate (See Figure 7) or a remote TCP/UDP server running on another location (See Figure 8).

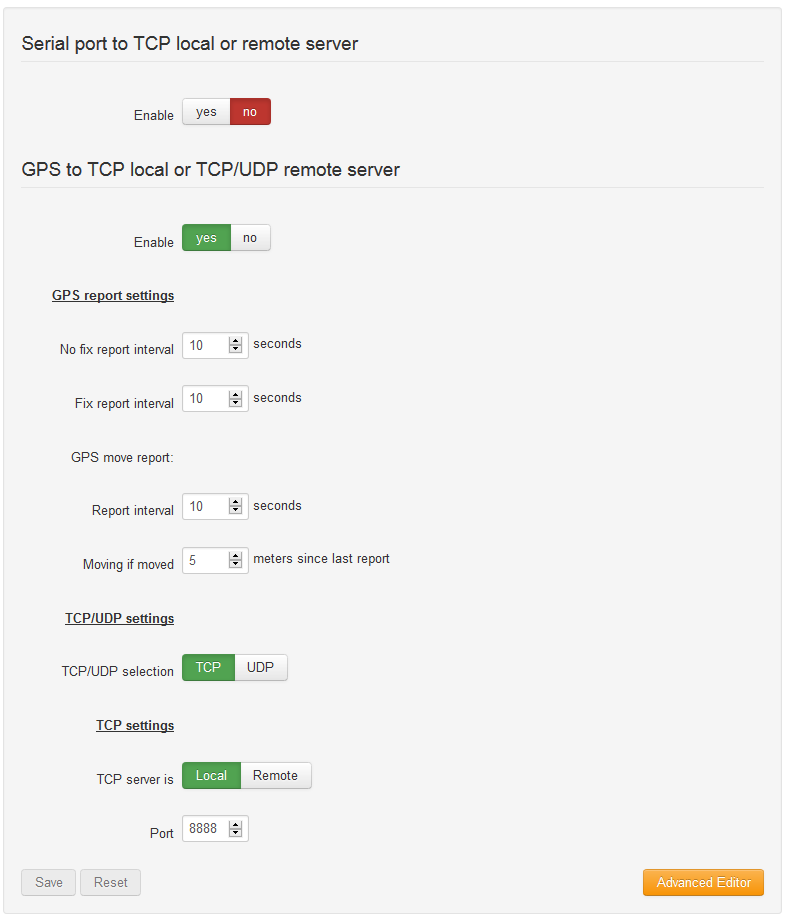


Figure : : GPS to local TCP server.

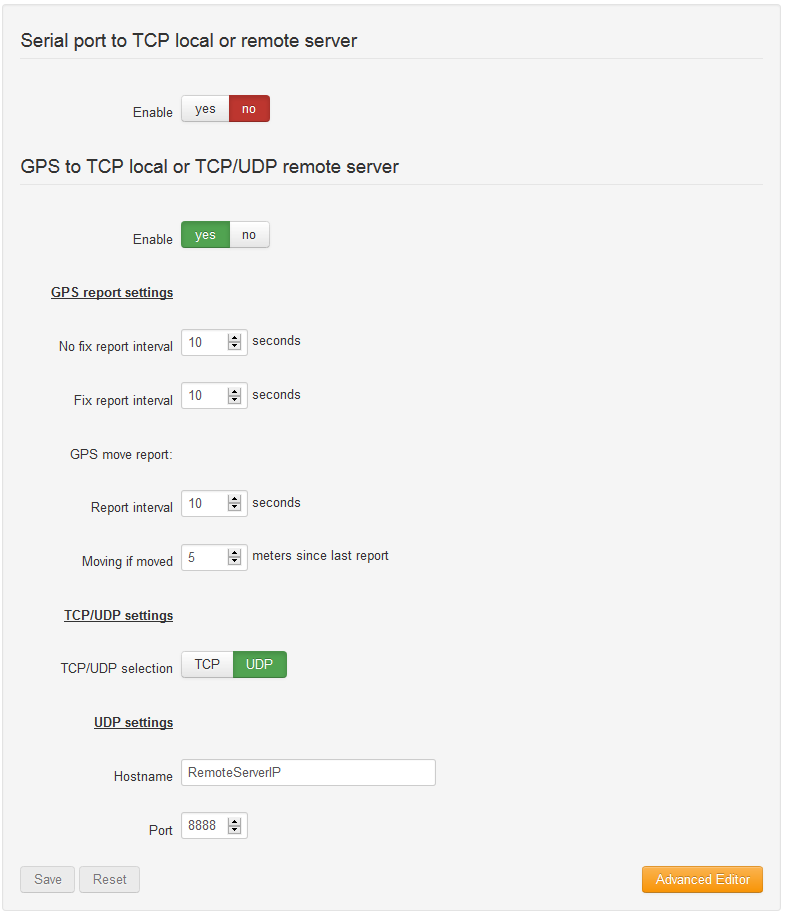


Figure : : GPS to remote UDP server.

This configuration is perfect for using with any CloudGate except for the Ethernet only version which does not have GPS.

This configuration is actually replicated on the Advanced editor:

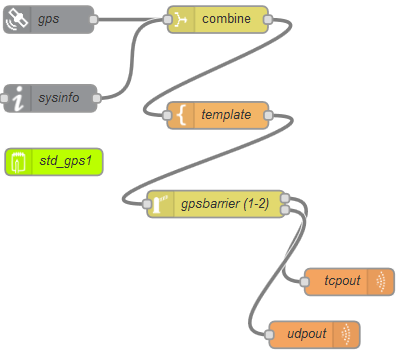


Figure : configuration under Advanced editor.

## Advanced Editor

At the bottom right corner of the Basic configuration interface there is an orange button that says Advanced Editor.



Figure : Advanced Editor.

By clicking on this button, a new web interface tab is going to open. This new tab is the LuvitRED advanced Editor:

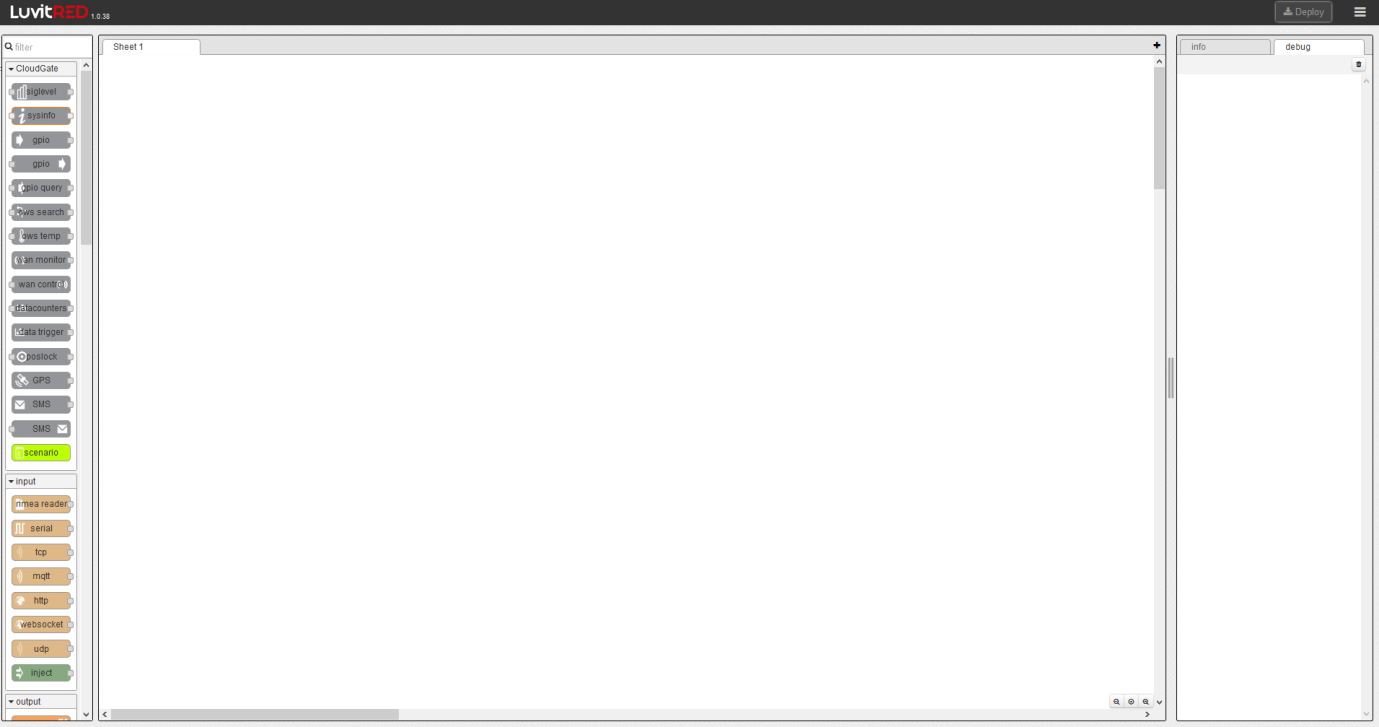


Figure : LuvitRED Advanced Editor.

NOTE: Please, keep an eye on popup blockers that may not allow this new page to be displayed.

The Advanced editor is actually running on a separate port than the standard CloudGate web page:

* Port **8080**: When on a http connection (local connection to the CloudGate)
* Port **8081**: When on a https connection (remote connection to the CloudGate)

NOTE: Please, be aware than when running a remote session to the CloudGate, the right port forwarding rule to port 8081 needs to be in place in order to reach the Advanced Editor.

The Advanced Editor can be divided into four sections:

1. **Deploy and Menu section:**

Located on the top right corner, it is the place where Deploy button and the Menu button is located:



Figure : Deploy and Menu section.

The Deploy button, as the name already says, is the function that deploys any configuration or modification made on the Editor section.

The Menu button contains multiple options of LuvitRED:

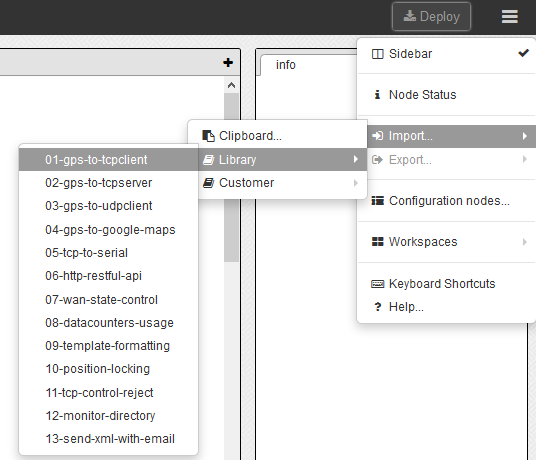


Figure : Menu button.

* Node Status: Adds an extra information line bellow some nodes.
* Import: Contains a Library of configurations. It is also a good to to import configurations using the Clipboard (copy and paste from a text file).
* Export: Can export a configuration into Clipboard, Library or Customer.
* Configuration nodes...: Shows the configuration nodes that are currently used and by which nodes on the editor.
* Workspaces: Place to manage the different workspeces on the editor.
* Keyboard Shortcuts
* Help.

1. **Nodes section:**

Located at the left side of the page, it contains a list of the nodes that can be used to create a customer configuration on the editor. It contains a filter to quickly find nodes in the list:

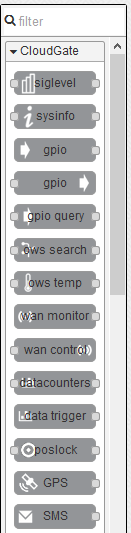


Figure : A reduced view of the Nodes section.

1. **Editor section:**

The editor is the central section of the Advanced Editor, this is the place where nodes are dragged to and where flows are created. Multiple workspaces can be created on the Editor:

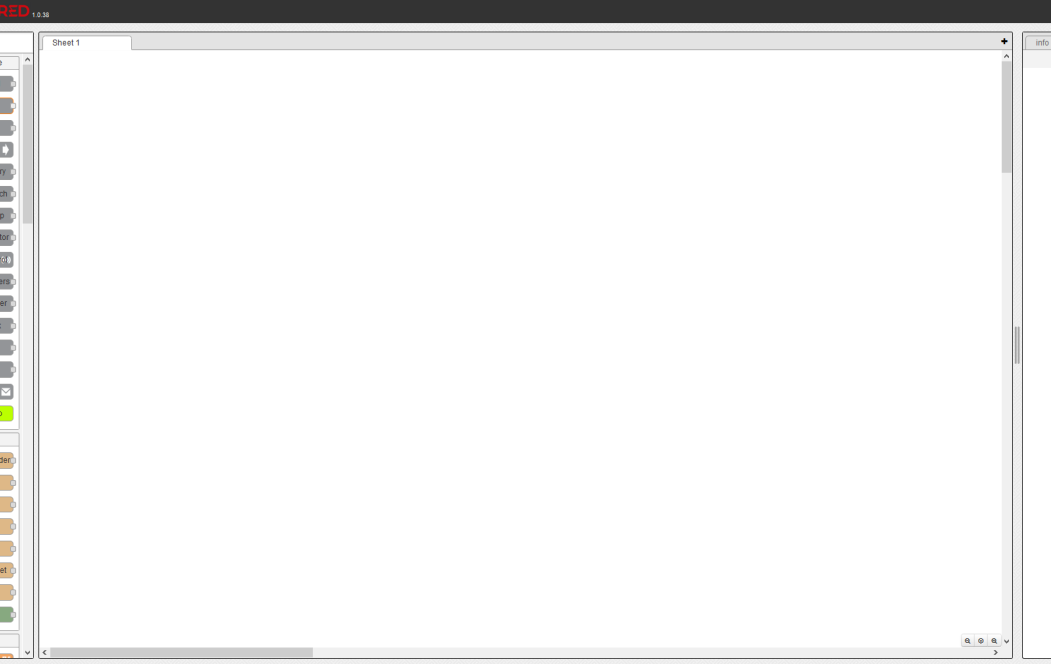


Figure : Editor section.

1. **Info and Debug section:**

The Info and Debug panel is where information about each node is presented (by clicking on the node - Info tab), debug from the debug nodes is shown (debug tab) and, if the configuration nodes option is selected under the menu button, configuration nodes are shown (config tab):

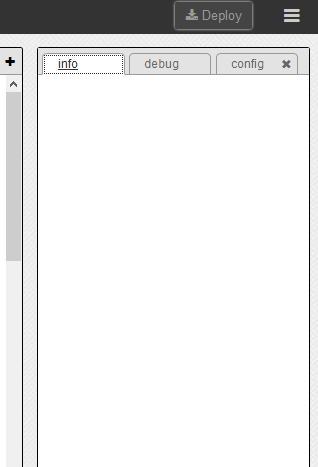


Figure : section.

# What is a node?

Nodes are pieces of code that are represented by a shape of a rectangle with rounded and by other five characteristics:

* An icon
* A background color
* A name
* Inputs
* Outputs

Some other nodes maybe present other characteristics, but in general all nodes follow the above mentioned ones.

Some examples of LuvitRED nodes are:

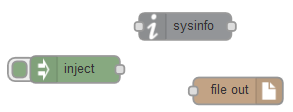
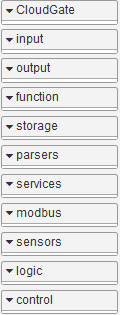


Figure : Examples of LuvitRED nodes.

Each node has its own set of configuration items inside them and serves a different purpose.

## Types of nodes

LuvitRED categorizes nodes into different groups:



* **CloudGate:** CloudGate and Option's expansion cards specific nodes.
* **Input:** General input nodes.
* **output:** General output nodes.
* **function:** Some pre-defined functionalities.
* **storage:** Storage and file handling.
* **parsers:** Parsing functions.
* **services:** Connection to M2M servers.
* **modbus:** Modbus specific nodes.
* **sensors:** Sensors specific nodes.
* **logic:** Logic nodes to route and manipulate the messages between nodes.
* **control:** Control nodes.

## Inject and Debug nodes

There are two nodes that are extremely important on any implementation on LuvitRED, those two nodes are inject and debug.

The inject and debug nodes are the last nodes listed under the input and output nodes respectively:

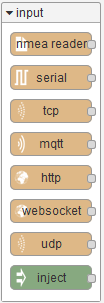


Figure : Inject and Debug nodes.

The inject node can be used to push data to another node. It can even be configured to do so periodically giving a sort of timing to an implementation:

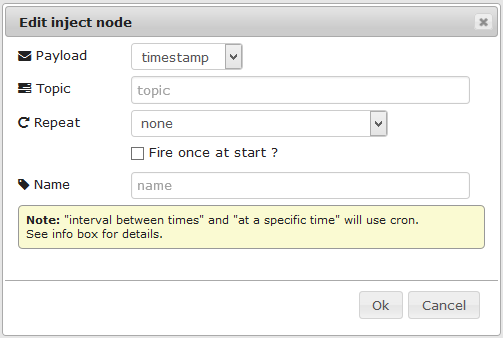


Figure : Inject node configuration.

Inject can be used to push strings, numbers, JSON values, etc.

A Debug node is use for printing information in the debug tab under the Info and Debug section. This node is very useful to find out how a message is being transformed between two nodes. The debug node can also be used to print verbose messages or general error on a configuration:

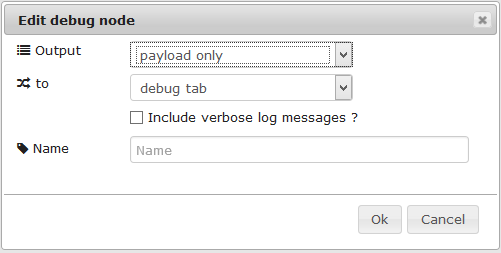


Figure : Debug node configuration.

An example of an inject node directly connected to a debug node printing a number (we are using the default configuration on each node):

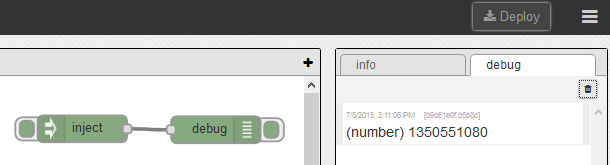


Figure : Inject and Debug nodes example.

# What is a flow?

A Flow is the result of connecting two or more nodes together to achieve a result.

Flows are unidirectional, so if a bidirectional communication is needed, a second flow needs to be added to create the second direction of communication.

The following are examples of a unidirectional flow:



Figure : Unidirectional flow.

And a bidirectional flow:

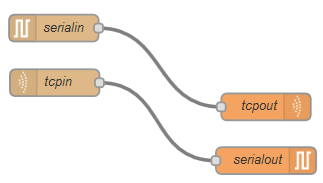


Figure : Bidirectional flow.

Flows can be very simple, as in the two previous examples, but they can also be much more complex depending on the application being developed:

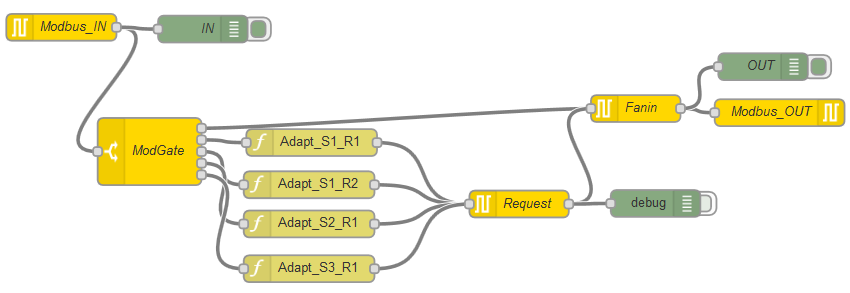


Figure : Example of a more complex flow (Modbus gateway configuration).

# What is a message?

A message is the package of information transferred between two nodes. It is important to understand the message format in order to be able to work with the information contained on the message.

## Single message structure

A message between two nodes is called **msg**. This object contains several sub items or keys (every msg is actually a table on Lua, but it can be seen as a json message). See the below example:



Figure : Simple example.

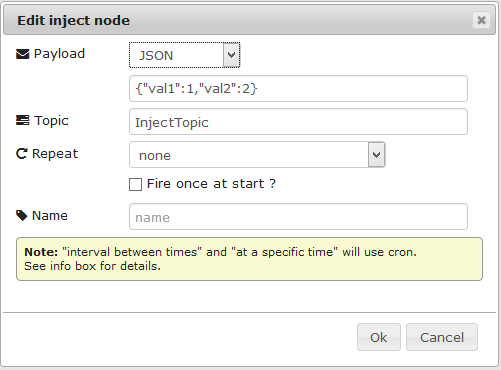


Figure : Inject configuration.

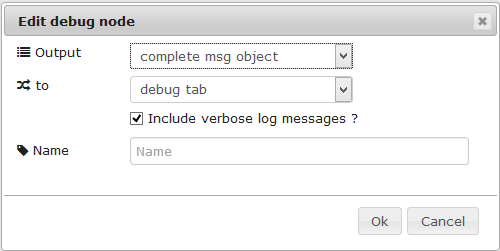


Figure : Debug configuration.

The following message is passed directly from an inject node to a debug node which is displaying the full package and not only the payload:

(Message) created by: f876d25b.7e67f8

Public properties: { payload = { val2 = 2, val1 = 1 }, timestamp = 1435340486, topic = "InjectTopic" } Private properties: { }

For the goal of this document, let's only focus on the "Public properties" of the message. The Public properties contain three items: payload, timestamp and topic

* The **payload** is the actual content of the message
* The **timestamp** is the time of creation of the message (it is in a Linux format)
* The **topic** is just a tag that will help recognize between two packages in the future

For this explanation, the message can be seen as the following structure:

{msg = { payload = { val2 = 2, val1 = 1 }, timestamp = 1435339549, topic = "InjectTopic" }}

In order to access the payload we need to refer to it as **msg.payload**

In order to access the timestamp we need to refer to it as **msg.timestamp**

In order to access the topic we need to refer to it as **msg.topic**

In order to access **val1** of payload we need to refer to it as **msg.payload.val1**

Following this logic, we can access any value contained on the payload of a message by simply looking at the structure of the message using a debug node.

## Combined message structure

The above explanation is key to understanding the data when one or more messages are combined into a single one, here is where topics are important. For example, let's say that one has two injects with different topics that go to a combine (no topic and also not collapse topics) node and then to a debug, like this:

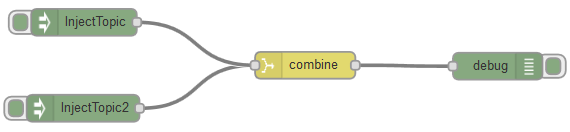


Figure : Example of combined messages.

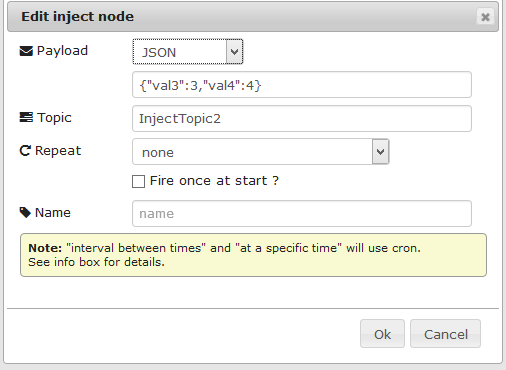


Figure : Second inject configuration.

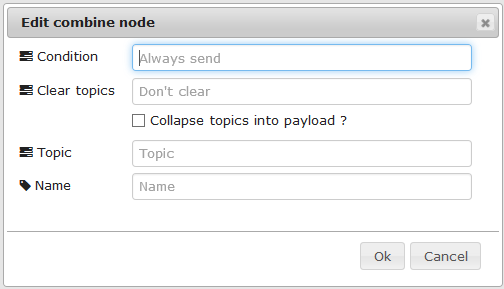


Figure : Combined node configuration.

Combine merges both messages in order to create a new one in which the new message topic is going to be the same as of the last message received on Combine, unless Combine is set with its own topic!:

 (Message) created by: 72e88578.9bc20c

Public properties: { payload = { InjectTopic = { val1 = 1, val2 = 2 }, InjectTopic2 = { val3 = 3, val4 = 4 } }, timestamp = 1435340622, topic = "InjectTopic2" } Private properties: { }

NOTE: The inject nodes where pressed in order, this is why the topic of the new message is InjectTopic2.

We can observe that the new message has a slightly different structure that the original one:

{msg = { payload = { InjectTopic = { val1 = 1, val2 = 2 }, InjectTopic2 = { val3 = 3, val4 = 4 } }, timestamp = 1435340622, topic = "InjectTopic2" } } }

Now, in order to get to val1, we need to add an extra step/level (InjectTopic):

**msg.payload.InjectTopic.val1**

The timestamp and topic keys are still on a level right under **msg**, so one can still access them in the same way as before:

**msg.timestamp**

**msg.topic**

If we now modify the Combine node to collapse the topics:

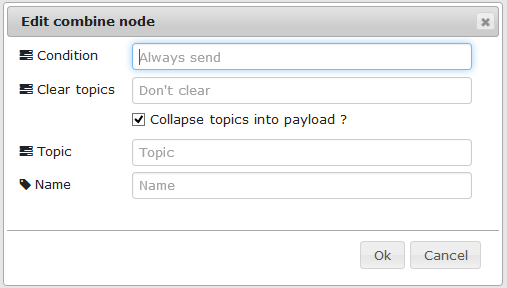


Figure : Collapse topics selected on combine node.

this is what happens:

(Message) created by: 72e88578.9bc20c

Public properties: { payload = { val3 = 3, val1 = 1, val4 = 4, val2 = 2 }, timestamp = 1435340830, topic = "InjectTopic2" } Private properties: { }

We can see now, that there is no more longer an extra step in order to access val1 (The order of appearance of the values is not really relevant as long as all the data is contained on the payload). To access val1 in this case, we only need to use:

**msg.payload.val1**

Now we have a complete new message. This new message can be then combined again with another message and one will have the same exercise over and over again.