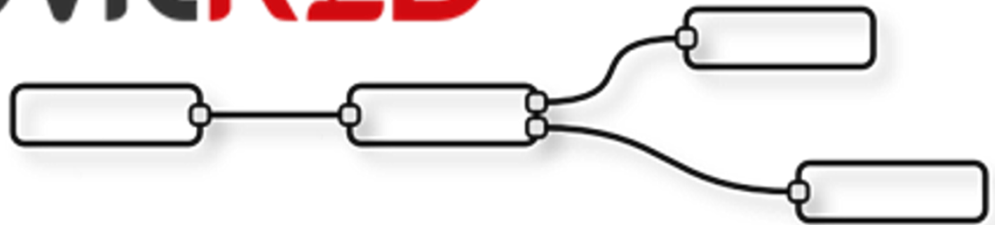


○ ○  
○ ○ P T I O N  
○ ○

# LuvitRED



## Basics of LuvitRED

Franco Arboleda

04-Sep-15

## Table of Contents

1	What is LuvitRED? .....	3
1.1	Why Lua? .....	3
1.2	What is the benefit of using LuvitRED? .....	3
1.3	Who is intended for? .....	3
1.4	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



# 1 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.

## 1.1 Why Lua?

Lua is a lightweight, simple to learn programming language that has an 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 basic 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 covered 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.

## 1.2 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 every time a new project comes.

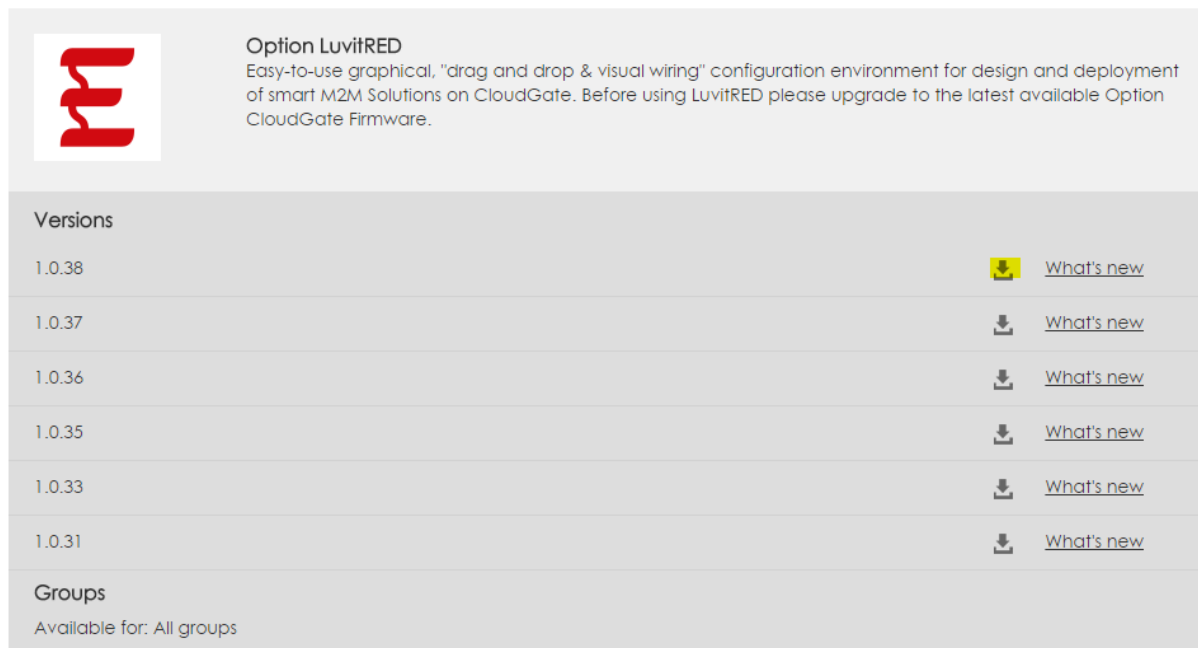
## 1.3 Who is intended for?

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







## 1.4 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:



**Option LuvitRED**  
Easy-to-use graphical, "drag and drop & visual wiring" configuration environment for design and deployment of smart M2M Solutions on CloudGate. Before using LuvitRED please upgrade to the latest available Option CloudGate Firmware.

Versions	
1.0.38	 <a href="#">What's new</a>
1.0.37	 <a href="#">What's new</a>
1.0.36	 <a href="#">What's new</a>
1.0.35	 <a href="#">What's new</a>
1.0.33	 <a href="#">What's new</a>
1.0.31	 <a href="#">What's new</a>

**Groups**  
Available for: All groups

Figure 1: Option LuvitRED download location on CGU.

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/fr4qjimpv7>

### 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 developed (2.x.x); this version will only be compatible with CloudGate firmware versions 2.x.x.
- If the Plugin page does **not show up in the CloudGate GUI** after uploading LuvitRED and rebooting the CloudGate, and/or the Image version under **"System Information"** on the Home page is shown in **red**. This reflects that the combination of CloudGate firmware and LuvitRED is not correct. Please, verify that the two previous notes have been followed.

## 2 LuvitRED editors

As mentioned in 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":

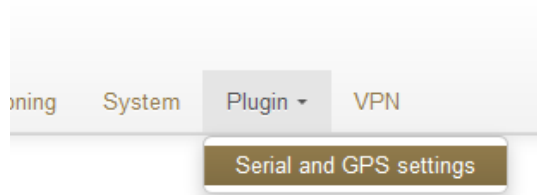


Figure 2: Plugin tab, Serial and GPS settings.

### 2.1 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:

#### Serial and GPS settings

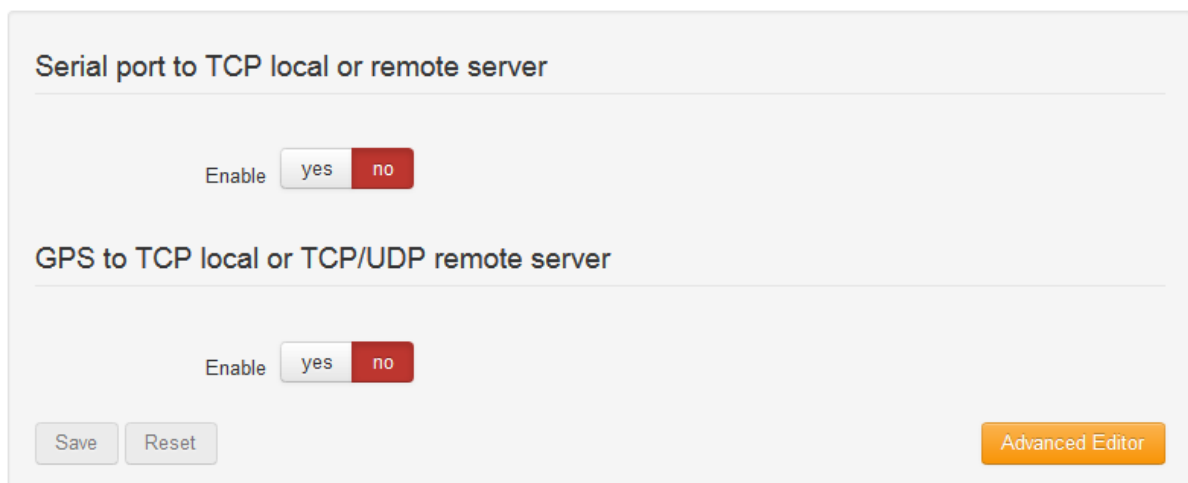
A screenshot of the 'Serial and GPS settings' basic interface. It features two sections: 'Serial port to TCP local or remote server' and 'GPS to TCP local or TCP/UDP remote server'. Each section has an 'Enable' label and a toggle switch with 'yes' and 'no' options. At the bottom, there are 'Save' and 'Reset' buttons on the left, and an 'Advanced Editor' button on the right.

Figure 3: 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 at another location (See Figure 5).

Serial port to TCP local or remote server

Enable
☒ yes
☐ no

Serial port settings

Baud rate

Data bits
☐ 7
☒ 8

Stop bits
☒ 1
☐ 2

Parity
☒ none
☐ even
☐ odd
☐ mark
☐ space

Flow control
☒ none
☐ XON/XOFF
☐ CTS/RTS

TCP settings

TCP server is

Port

GPS to TCP local or TCP/UDP remote server

Enable
☐ yes
☒ no

Save
Reset
Advanced Editor

Figure 4: Serial to local TCP server.

### Serial port to TCP local or remote server

Enable ☒ yes ☐ no

**Serial port settings**

Baud rate: 9600

Data bits: ☐ 7 ☒ 8

Stop bits: ☒ 1 ☐ 2

Parity: ☒ none ☐ even ☐ odd ☐ mark ☐ space

Flow control: ☒ none ☐ XON/XOFF ☐ CTS/RTS

**TCP settings**

TCP server is:

Hostname: RemoteServerIP

Port: 8889

### GPS to TCP local or TCP/UDP remote server

Enable ☐ yes ☒ no

Figure 5: Serial to Remote TCP server.

This configuration is ideal for setting up the RS232 ports on the following cards:

- Low cost serial expansion card, PN CG1101-11919.
- Industrial serial card (RS232), PN CG1102-11920.
- Telematics card (RS232), PN CG1106-11957, CG5106-11983 and CG5106-11984

This configuration is replicated on the "Advanced editor" explained on section 2.2:

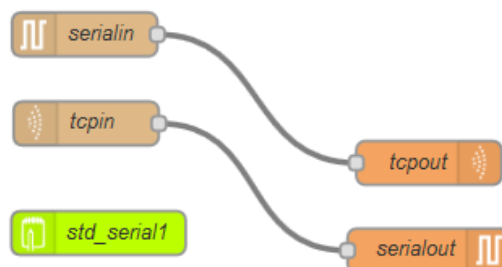


Figure 6: Same configuration under Advanced editor.

## OPTION

### 2. 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).

Serial port to TCP local or remote server

Enable

GPS to TCP local or TCP/UDP remote server

Enable

GPS report settings

No fix report interval  seconds

Fix report interval  seconds

GPS move report:

Report interval  seconds

Moving if moved  meters since last report

TCP/UDP settings

TCP/UDP selection

TCP settings

TCP server is

Port

Figure 7: : GPS to local TCP server.



Serial port to TCP local or remote server

Enable
☐ yes
☒ no

GPS to TCP local or TCP/UDP remote server

Enable
☒ yes
☐ no

GPS report settings

No fix report interval
seconds

Fix report interval
seconds

GPS move report:

Report interval
seconds

Moving if moved
meters since last report

TCP/UDP settings

TCP/UDP selection
☐ TCP
☒ UDP

UDP settings

Hostname

Port

Save
Reset
Advanced Editor

Figure 8: : GPS to remote UDP server.

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

This configuration is replicated on the "Advanced editor" explained on section 2.2:

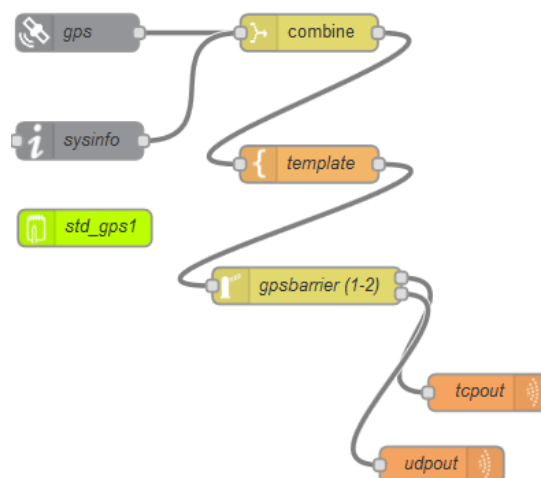


Figure 9: configuration under Advanced editor.

## 2.2 Advanced Editor

The Advanced Editor allows you to configure more complex applications, such as MODBUS or multiple serial ports. An example would be the Industrial serial card, which has one RS232 and one RS485 port.

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



Figure 10: Advanced Editor.

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

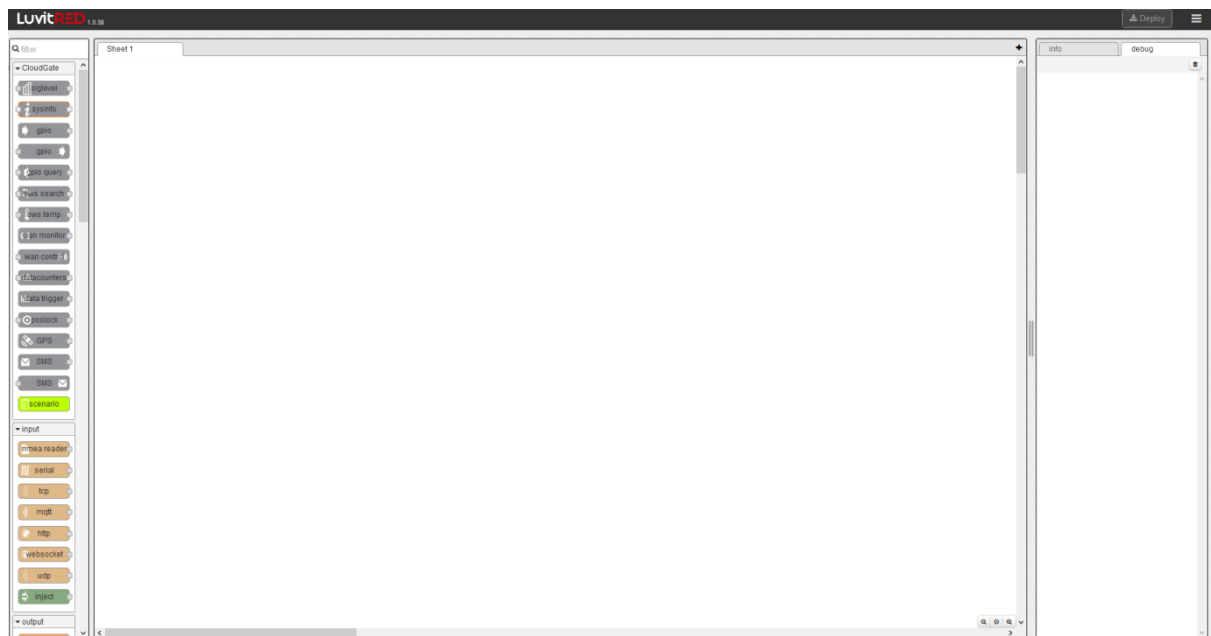


Figure 11: LuvitRED Advanced Editor.

**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:

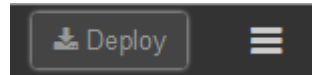


Figure 12: Deploy and Menu section.

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:

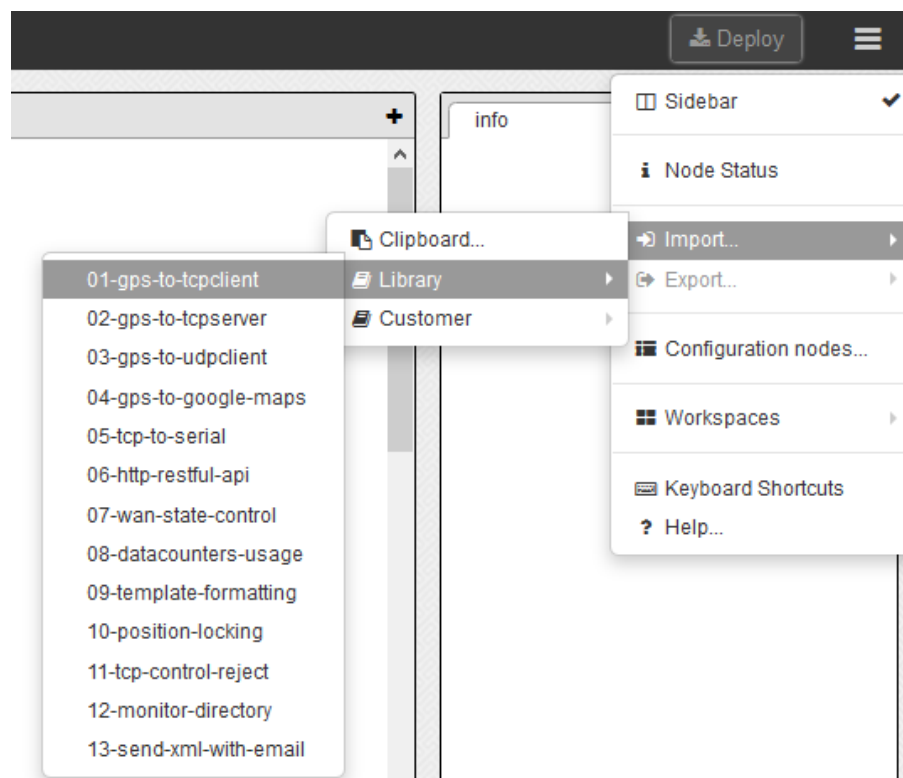


Figure 13: Menu button.

- Node Status: Adds an extra information line below 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 workspaces on the editor.
- Keyboard Shortcuts
- Help.

## 2. 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:

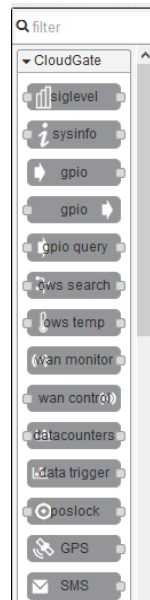


Figure 14: A reduced view of the Nodes section.

## 3. 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:

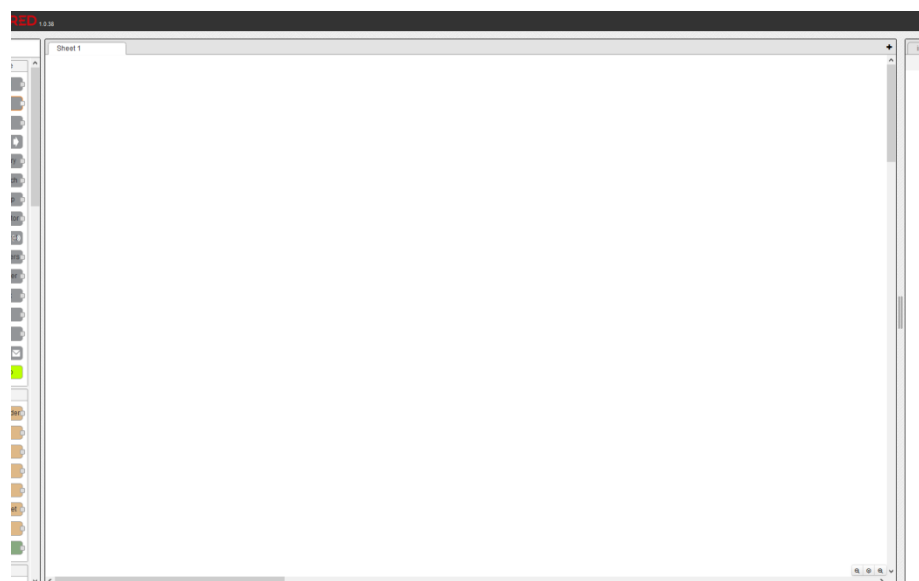


Figure 15: Editor section.

#### 4. 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):

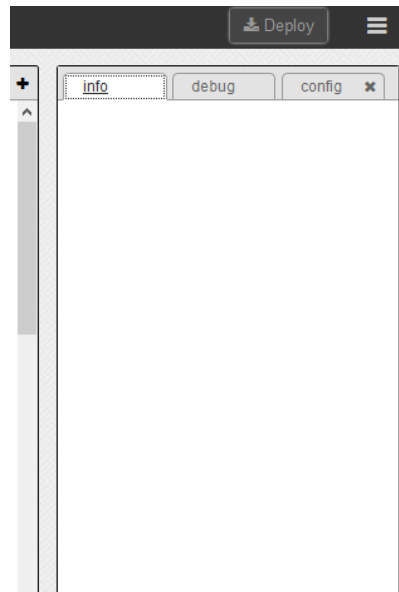


Figure 16: section.

### 3 What is a node?

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

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

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

Some examples of LuvitRED nodes are:

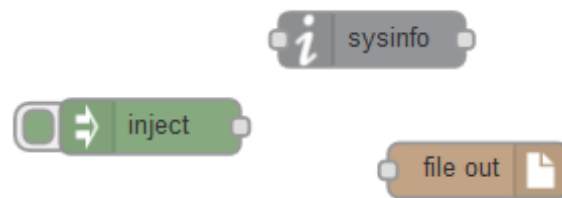


Figure 17: Examples of LuvitRED nodes.

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

#### 3.1 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.

## 3.2 Inject and Debug nodes

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

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

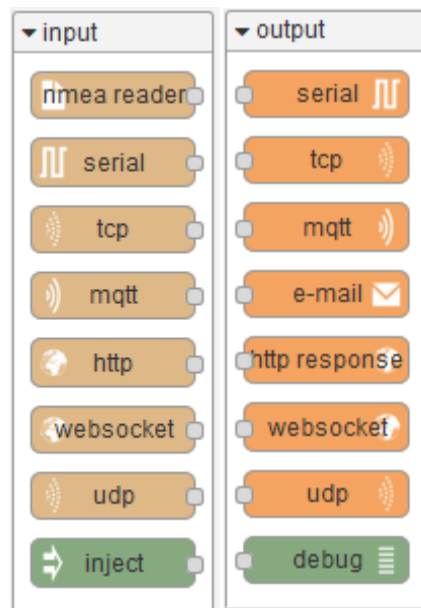


Figure 18: Inject and Debug nodes.

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:

**Edit inject node**

✉ Payload: timestamp

📄 Topic: topic

🔄 Repeat: none

☐ Fire once at start ?

🔑 Name: name

**Note:** "interval between times" and "at a specific time" will use cron. See info box for details.

Ok Cancel

Figure 19: Inject node configuration.

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

## OPTION

A Debug node is used 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:

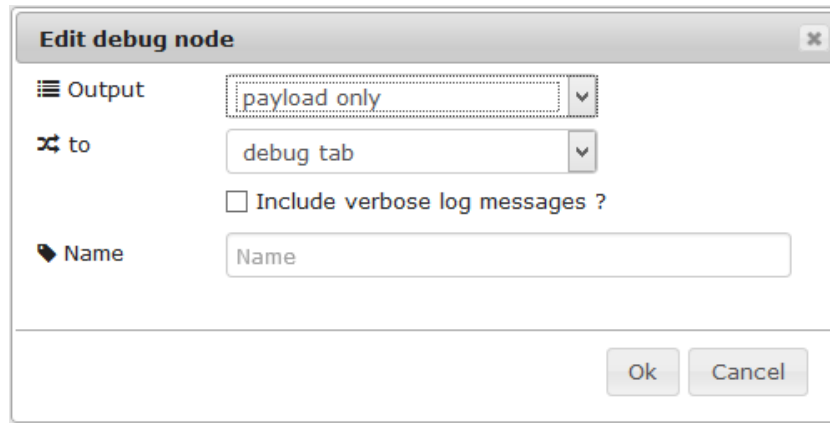


Figure 20: Debug node configuration.

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

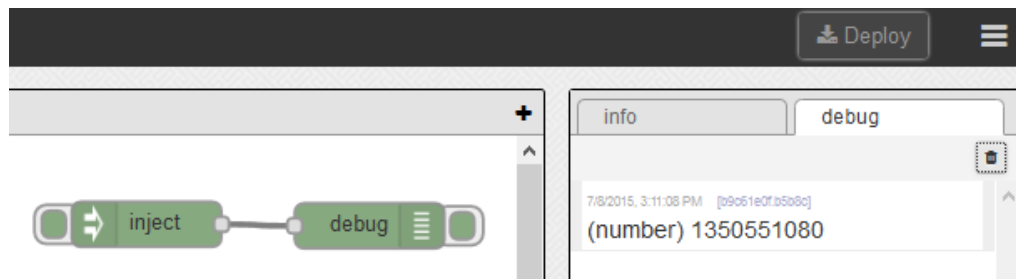


Figure 21: Inject and Debug nodes example.



## 4 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 22: Unidirectional flow.

And a bidirectional flow:

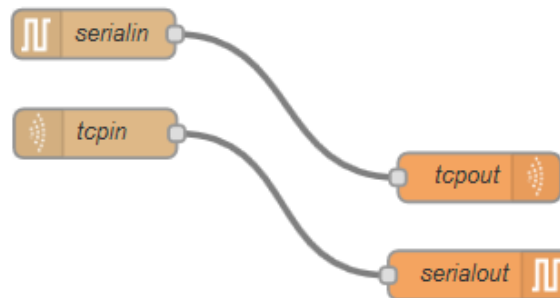


Figure 23: Bidirectional flow.

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:

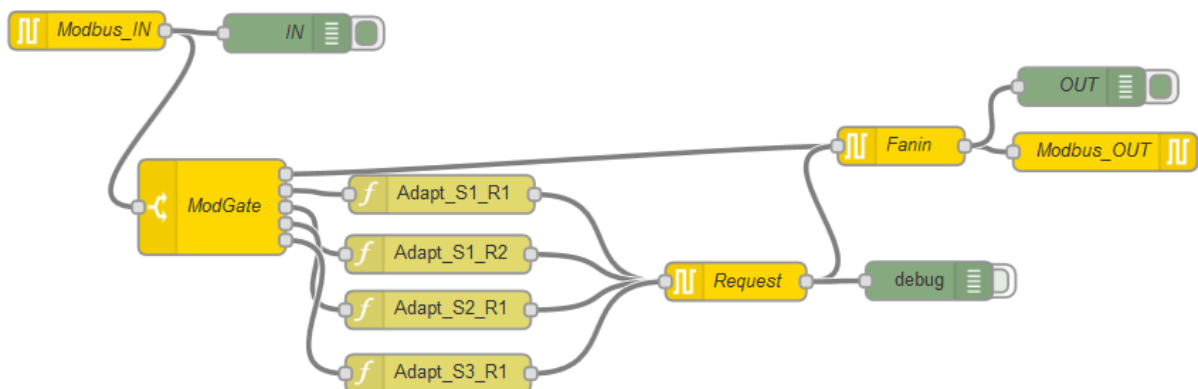


Figure 24: Example of a more complex flow (Modbus gateway configuration).

## 5 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.

### 5.1 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 25: Simple example.

**Edit inject node**

✉ Payload: JSON

{ "val1": 1, "val2": 2 }

📄 Topic: InjectTopic

🔄 Repeat: none

☐ Fire once at start ?

📌 Name: name

**Note:** "interval between times" and "at a specific time" will use cron. See info box for details.

Ok Cancel

Figure 26: Inject configuration.

**Edit debug node**

📄 Output: complete msg object

🔗 to: debug tab

☒ Include verbose log messages ?

📌 Name: Name

Ok Cancel

Figure 27: Debug configuration.

## OPTION

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 objective 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.

## 5.2 Combined message structure

The above explanation is key to understanding the data when one or more messages are combined into a single message; however, 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 in payload") node and then to a debug, like this:

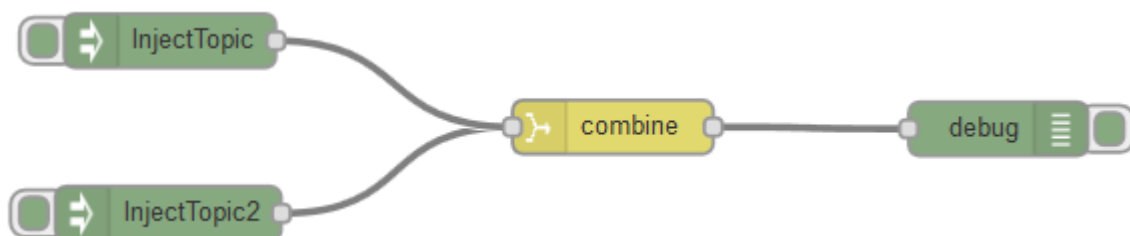


Figure 28: Example of combined messages.

Figure 29: Second inject configuration.

Figure 30: 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 were 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 than 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:

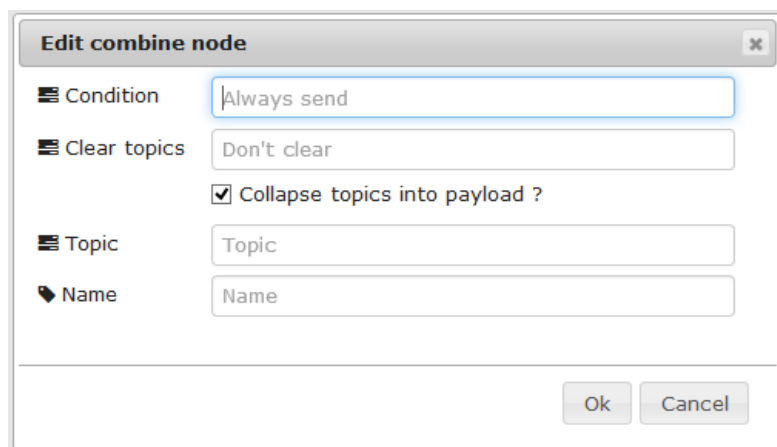


Figure 31: Collapse topics selected on combine node.

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 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.

○ ○  
○ ○ P T I O N  
○ ○