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



**Franco Arboleda**

**01-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 What is a node? 5

2.1 Types of nodes 5

3 What is a flow? 6

4 What is a message? 7

4.1 Single message structure 7

4.2 Combined message structure 8

# 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. Nodes such as 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 the most 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.

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.

Notes:

* Be aware that LuvitRED has always a minimum firmware requirement in order to work correctly. The minimum firmware version for LuvitRED to work correctly is 1.44.0.
* LuvitRED v 1.x.x is only compatible with firmware versions 1.x.x. A new version of LuvitRED is being develop (2.x.x), this version will only be compatible with 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, this means that the combination of firmware and LuvitRED is not correct. Please, verify the two previous notes.

# 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 example of LuvitRED nodes are:

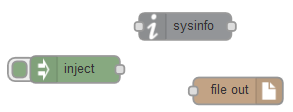


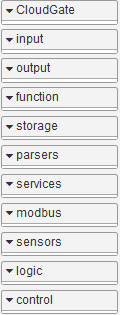
Figure : Examples of LuvitRED nodes.

Each node has its own different configuration items inside them and server for a different purpose.

## Types of nodes

LuvitRED supports different kind of nodes that are placed into 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.



# 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:

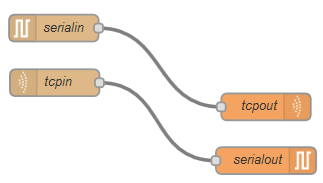


Figure : Bidirectional flow.

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

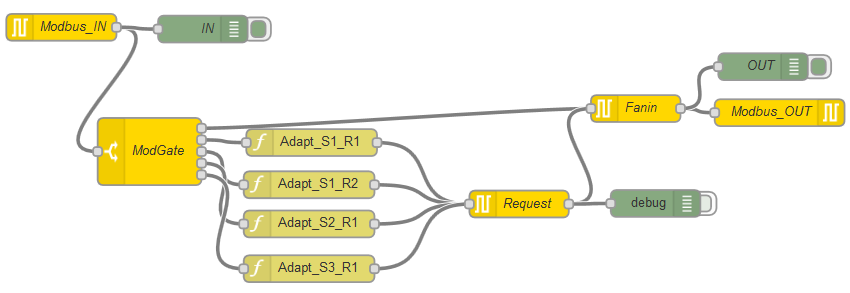


Figure : Example of a more complex flow (Modbus gateway configuration).

# 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.

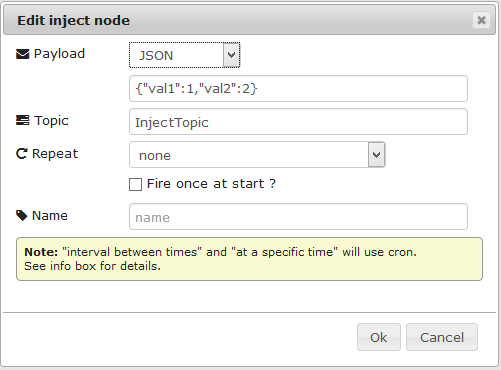


Figure : Inject configuration.

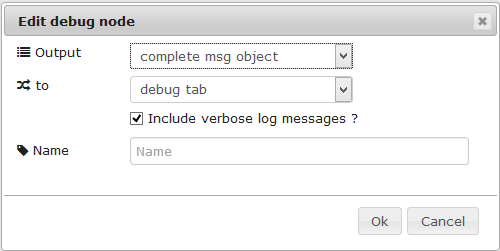


Figure : Debug configuration.

The following message is passed directly from an inject node to a debug node which is printing 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, lets only focus only 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**

Is like every { bracket separates different levels of data that can be access by replacing the { bracket with a . (dot).

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 understand the data when one or more messages are combined into a single one, here is where topics are important. Let say that I have 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:

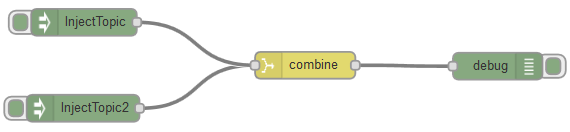


Figure : Example of combined messages.

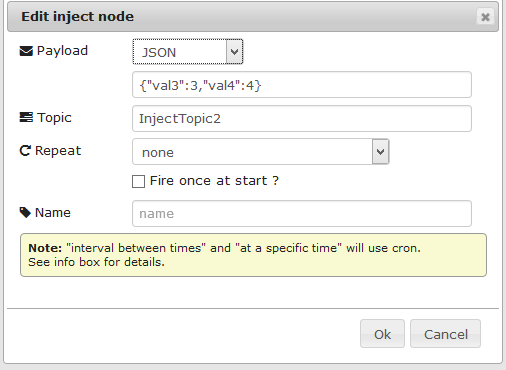


Figure : Second inject configuration.

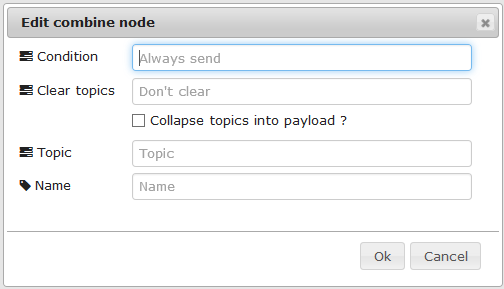


Figure : Combined node configuration.

Combine merges both messages to create a new one on 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**

while timestamp and topic can still be access in the same way:

**msg.timestamp**

**msg.topic**

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

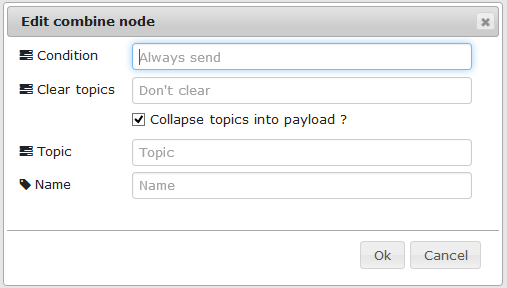


Figure : Collapse topics selected on combine node.

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, 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 you will have the same exercise over and over again.