LIGHTWARE
INNOVATIVE. RELIABLE. MADE IN EUROPE.

User's Manual

LARA
LIGHTWARE ADVANCED ROOM AUTOMATION

Lightware Advanced Room Automation

Software Platform

v1.2 06-05-2024

Symbol Legend

The following symbols and markings are used in the document:

WARNING! Safety-related information that is highly recommended to read and keep in every case!

ATTENTION! Useful information for performing a successful procedure; it is recommended to read.

DIFFERENCE: Feature or function that is available with a specific firmware/hardware version or product variant.

INFO: A notice, which may contain additional information. Procedure can be successful without reading it.

DEFINITION: The short description of a feature or a function.

TIPS AND TRICKS: Ideas that you may have not known yet, but can be useful.

Navigation Buttons

Go back to the previous page. If you clicked on a link previously, you can go back to the source page by pressing the button.

Navigate to the Table of Contents.

Step back one page.

Step forward to the next page.

Document Information

All presented functions refer to the indicated products. The descriptions have been made while testing these functions in accordance with the indicated Hardware/Firmware/Software environment:

Item	Version
LARA version	1.2.0b37

Document revision: **v1.2**

Release date: **06-05-2024**

Editor: Laszlo Zsedenyi

About Printing

Lightware Visual Engineering supports green technologies and eco-friendly mentality. Thus, this document is made primarily for digital usage. If you need to print out a few pages for any reason, follow the recommended printing settings:

- Page size: A4
- Output size: Fit to page or Match page size
- Orientation: Landscape

TIPS AND TRICKS: Thanks to the size of the original page, a border around the content (gray on the second picture below) makes it possible to organize the pages better. After punching holes in the printed pages, they can easily be placed into a ring folder.

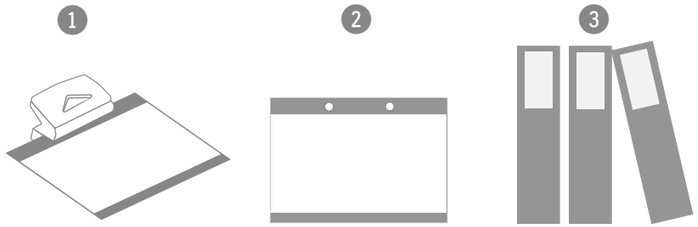


Table of Contents

1

Basics and Building Blocks

Thank You for choosing LARA. In the first chapter we would like to introduce the software, highlighting the most important features in the sections listed below:

1.1. Introduction

What is LARA?

Lightware Advanced Room Automation (LARA) is a future-proof room automation platform. This software can be used for making controlling functions in meeting rooms.

Where Can You Find It?

It runs integrated in Taurus UCX, UCX-TPX and MMX2 series devices, configurable via a browser. Meeting participants can control the devices in the room through a touch panel.

Basics

Room integration

The keyword is: room. LARA is developed for meeting room size environment.

Separated SW tool

LARA is part of the Lightware product, however, it is an independent entity in the device. Since the device is protected with a password, LARA is also safe from unwanted modifications.

Availability

LARA is available in Taurus UCX and UCX-TPX devices, as well as in MMX2 and TPN control box devices.

Key Benefits

Open Integration

Controlling any device that supports open protocols (REST, TCP/IP, LW3, etc). No need for certificated training, as LARA is based on JavaScript.

No External Controller

LARA is in the Lightware products (for now in Taurus and MMX2) and can be used to connect and control third-party devices. No need to purchase an extra controller box.

Up/downloading the Configuration

The full LARA configuration can be downloaded as a ZIP file. The file can be uploaded to the same device or to another device of the same type.

LARA is Expandable

The modular structure, the re-usable Modules and the continuous development mean more and more new features.

Solution for Meeting Rooms

LARA is a software platform that has been designed mainly for controlling meeting rooms. While developing LARA, the experiences of the popular Event Manager - which can be found in numerous Lightware devices - have been applied. The platform is designed and developed by Lightware Visual Engineering.

With LARA, you can automate your meeting rooms, create or re-use software Modules by using the power of JavaScript, control the behavior of the Taurus and connect it to other third-party devices or services, or do virtually everything that is possible.

Supported Devices

LARA is available in the following devices:

TAURUS UCX	TAURUS TPX	MMX2
UCX-2x1-HC30 UCX-2x2-H30 UCX-4x2-HC30 UCX-4x2-HC30D UCX-2x1-HC40 UCX-2x2-H40 UCX-4x2-HC40 UCX-4x2-HC40D UCX-4x3-HC40	UCX-2x1-TPX-TX20 UCX-4x3-TPX-TX20	MMX2-4x3-H20 MMX2-4x1-H20
TPN		
TPN-CTU-X50		

LARA Version

> **ATTENTION!** The LARA versions may contain big differences. This User's Manual is based on a certain version of LARA that is displayed on the bottom of each page. Please check the LARA version of your device. LARA comes with the Taurus/MMX2 firmware package and we always recommend to update the firmware of your device to the latest version.

1.2. Limitations and Capabilities

Please consider the following when using LARA:

- The Taurus/MMX2 configuration contains the whole LARA configuration as well. If the device configuration is handled with Bulk management or with the Backup/Restore feature, LARA settings can be preserved.
- If LARA is running when the Lightware device (that runs LARA) is restarted, LARA will **run automatically again**.
- A connected or an external device can be accessed via an Instance. (See more information about the Instances in the Modules and Instances section.)
- When LARA is running, all the **defined Instances run together**.
- At most 20 Instances can be run parallel if the complexity of the Instances are at 'average' level (see the specification below of a complex sample configuration).
- The storage space is **128 MB** for the Modules, Instances, user Module content and all codes.
- The available RAM is **128 MB**, LARA uses 56 MB in factory default state.
- Restoring the factory default settings in the UCX/MMX device will **delete the LARA configuration**. This also happens during a firmware update if the **firmware version is v2.0 or below**.

The Specifications of a Complex Sample Configuration

- It consists of 17 Instances:
  - 14 from the Generic-tcp-ip-client Module,
  - 1 from the Taurus-ucx-mmx2 Module
  - 1 from the User panel Module, and
  - 1 from the Logic Module.

This configuration needs cca. **92 MB RAM** space when running.

1.3. How to Access?

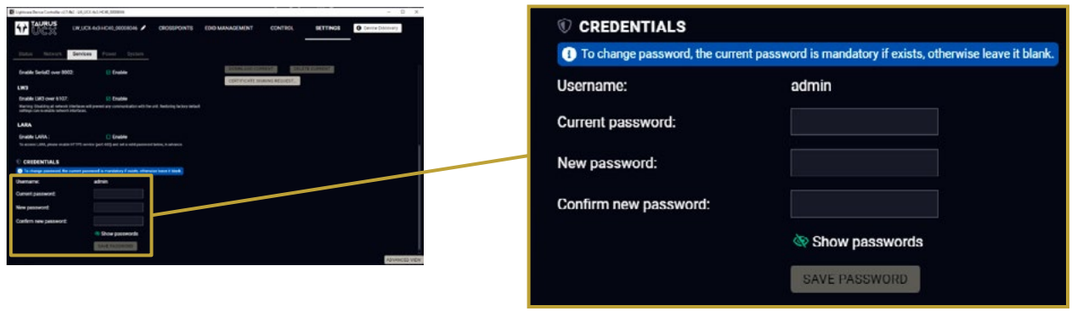
Short Steps

Step 1. Setting the password in the desired UCX/MMX2 device.
Step 2. Enabling port #443.
Step 3. Enabling LARA in the device.
Step 4. Opening LARA in a browser.

Detailed steps

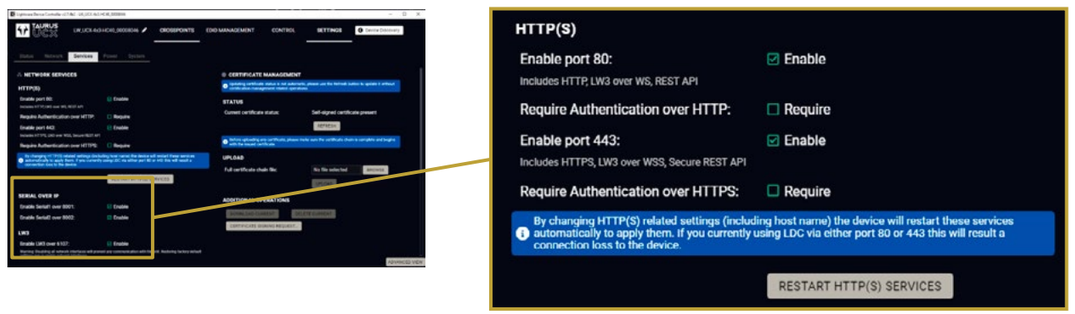
Step 1 – Setting the Credentials in the Desired Taurus Device

- Start the **LDC software** and connect to the device or open the **web LDC** in a browser and type the IP address of the device.
- Navigate to the **Settings/Services** tab.
- **Set a password** for the user 'admin' (if not set previously) and **Save** it.



Step 2 – Enabling Port #443

- Navigate to the **Settings/Services** tab in LDC or in web LDC.
- See the Network services section and mark the **Enable port 443** setting.



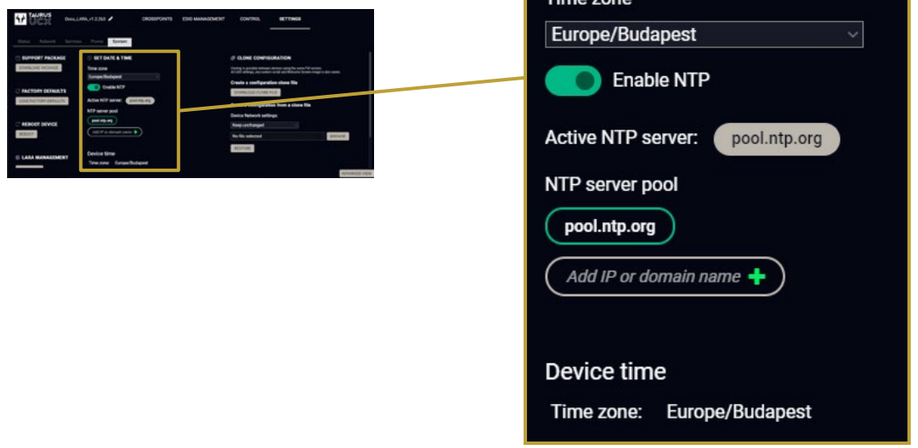
Step 3 – Enabling LARA in the Device

- Navigate to the **Settings/Services** tab in LDC or web LDC.
- See the Network services section and mark the **Enable LARA** (factory default state is **disabled**).

Step 4 – Internal Time Setting

For the proper working in Taurus and LARA, the internal clock must be synchronized with an NTP server. You can create time-sensitive Rules where the exact time and date is essential. The preparation shall be done in the UCX/MMX2 device:

- Navigate to the **Settings/System** tab in LDC or web LDC.
- **Enable NTP** by the switch in the **Set Date & Time** secion.
- Set the NTP server according to your needs.



Step 5 – Opening LARA in a Browser

- Open a **web browser** and type the following in the address line: (make sure to type **https**)

```
https://<IP_address>/lara
```

TIPS AND TRICKS: If you select the **Settings/System** tab, you can open LARA with the **OPEN LARA** button.

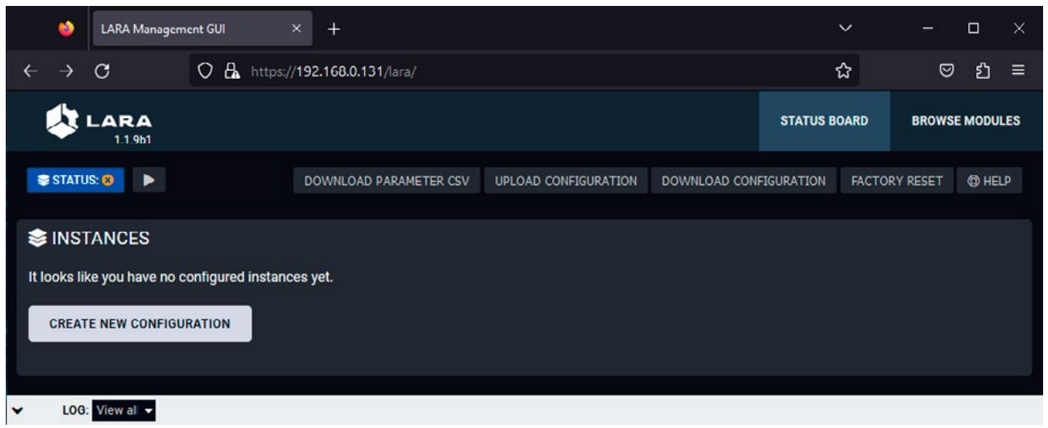
After that you have to enter the user **admin** and the set **password**.

ATTENTION! Due to a known issue a malfunction occurs during login. After pressing the login button you will be directed to the web LDC instead of LARA. Navigate to the **https://<IP_address>/lara** page again to open LARA.

ATTENTION! Session management for security reasons is introduced in UCX and MMX2 devices. When you log into LARA (or in Taurus web LDC) a TCP session is started. After 25 minutes of inactivity you are prompted to confirm to continue the session (in five minutes) or you will be logged out. After two hours you will be logged out automatically regardless of activity.

ATTENTION! Due to a known issue there is a malfunction in Session management: you have to confirm to continue the session every 25 minutes despite the continuous activity.

INFO: When opening LARA for the first time or after factory reset, you must accept the disclaimer statement.



The Status Board of LARA – in Factory Default State

1.4. The Structure of LARA

1.4.1. Modules and Instances

Modules

The **Module** is the basic building block of LARA. The main purpose of the modular structure is to create re-usable units. The Modules have different functions:

	Driver Module	Connecting to a device in the room.
	Logic Module	Creating the connection between other Modules by defining Rules and the working logic.
	User Panel Module	Providing a Graphical User Interface (GUI) for the end-users, e.g. Touch screen control.
	Script Module	Any custom software for a specific purpose, e.g. Cisco Webex Module.

Instances

The Module cannot be used by itself, an Instance has to be created from it. This way of working can be demonstrated with a simple example: if you have an executable file on your computer, that is a Module. When you run it once or multiple times, the windows are the Instances. The software pieces are identical behind the running Instances, however, you can modify their behaviour by giving them different Parameters at start.

Parameter and Value

The modular structure has an effect on the Parameter handling:

The definition of a Parameter: stored in the Module.

The value of the Parameter: stored in the Instance but the default value is stored in the module.

For example: You have two Taurus devices, so you create a Taurus Module. You have to create two Instances and each Instance will contain the IP address of the desired Taurus device.

Driver Module

If you want to make a connection between LARA and an external device, a Driver Module is necessary.

> INFO: Everything that is outside LARA is 'external'. The device that runs LARA is also considered external, too. That's why you have to create a Taurus Driver for UCX/MMX2 devices.

The Driver Module can be:

- **Device-specific Driver**: e.g. Taurus / MMX2 Driver, Generic Lightware LW3 Driver
- **Communication Driver:** e.g. Generic TCP Client Driver, Generic REST API Client Driver

LARA comes with built-in (factory) Modules but you can create your own Module as well, see the Factory Module Descriptions and User Module Descriptions chapters.

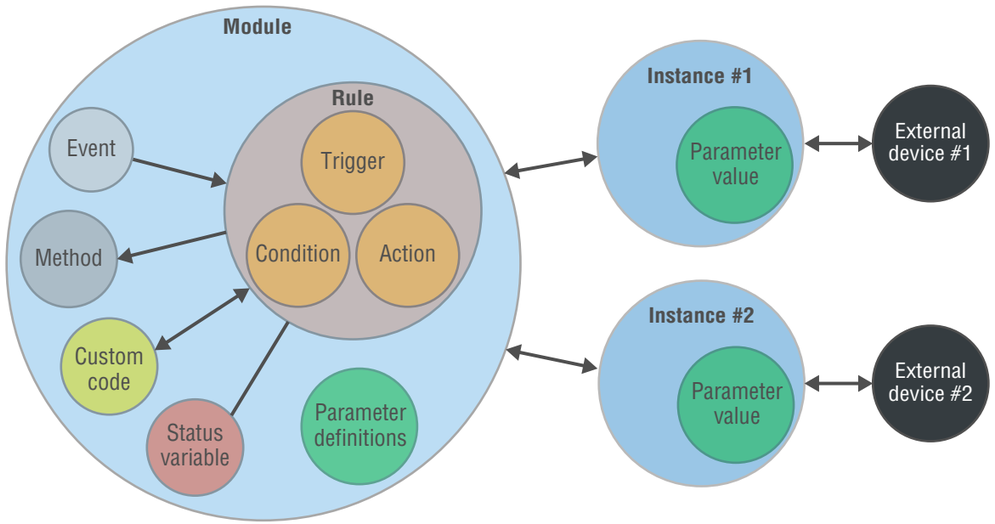
Logic Module

The Logic Module describes the **working logic** of the room and/or it can be used for interactions between other Modules. A logic Module creates the connection between the Modules with **Rules**.

User Panel Module

The User panel Module provides a GUI for the end-users for controlling functions in the room, like a Touch screen controller. HTML content can be uploaded to the Taurus as a pre-formatted HTML file, and it can be edited in the code editor later. The HTML file can be opened in the Touch screen controller. The Module allows modifying the visible elements of the touch screen that makes the GUI interactive. See more information in the User Module Descriptions chapter.

The Inside of the Module



The Connection of a Driver Module and the Instances

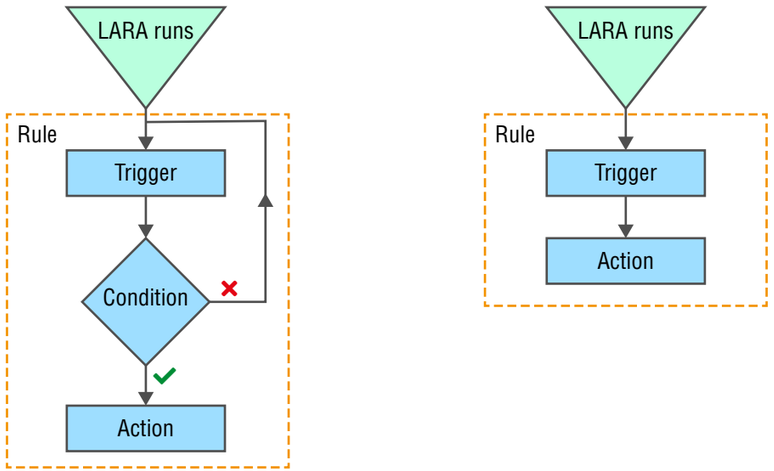
1.5. The Parts of the Automation

RULE = TRIGGER [+CONDITION] & ACTION

Rules

The Rule is the part of the Module where the automation steps are described. You can define the following main parts:

- **Trigger**: a specific change has happened.
- **Condition (optional)**: if a Condition is set, it has to be fulfilled to run the Action.
- **Action**: execute one or more operations.



The Parts of the Automation – With and Without a Condition

Example

- **Trigger**: occupancy sensor sends a signal (GPIO P1 level is high).
- **Condition**: a laptop is connected (signal is present on I1).
- **Action**: switch on the display, show the laptop image (swith I1 to O1 port and send "powerON" command to the display).

The details of the automation can be found in the The Automation Process section.

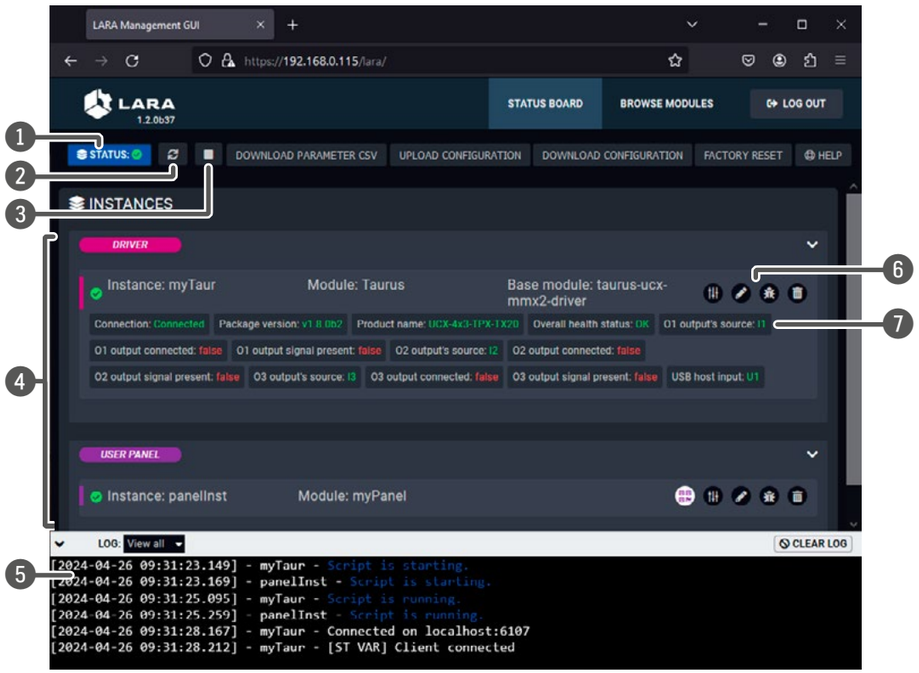
2

Start Using LARA

This chapter is about the structure and the main parts of LARA.

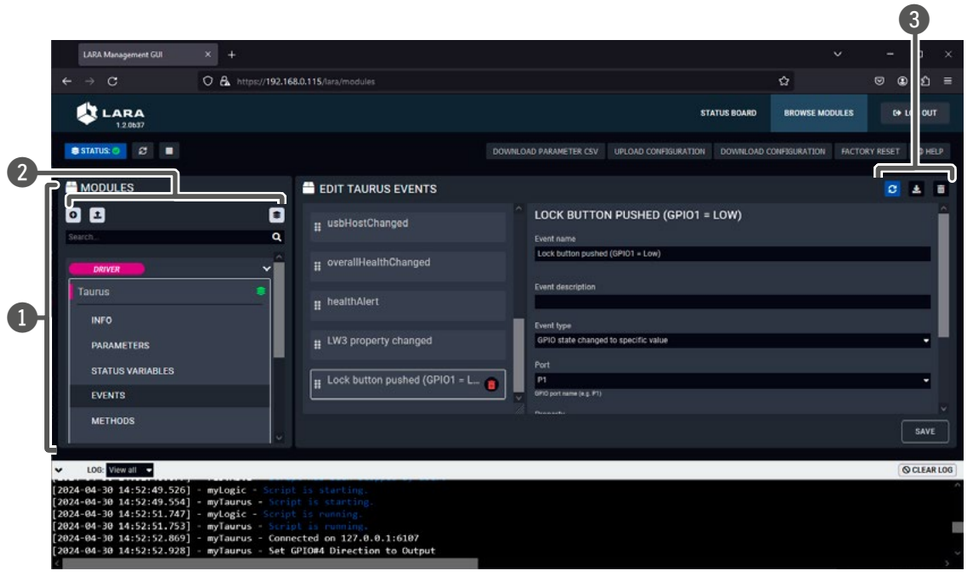
2.1. The Main Screens

2.1.1. The Status Board Tab



1. **LARA status icon** — / LARA configuration is **running** / **not running**
2. **Restarting LARA** — : press to restart LARA configuration
3. **Running LARA** — / : press to start / stop LARA configuration
4. **Instances** — List of the currently defined Instances of any Modules with a status icon.
5. **Log window** — Displaying system messages, but custom messages can be also displayed. The section is resizable or it can be hidden by the down arrow:
6. **Instance buttons**
   - : Editing the **Parameters** of the Instance.
   - : Editing the **parent** Module.
   - : Opening the **Methods and Events** testing page.
   - : **Deleting** the Instance.
   - : QR code and link to the **Web page** of a **User panel** Module.
7. **Info labels** — Only displayed if LARA configuration is running, in order to show valid information. For customizing labels, see the Status Variables section.

2.1.2. The Browse Modules Tab



1. **Modules** — List of the used Modules. By pressing the down arrow ,, the submenu is opened.
   - : The Module has no Instances.
   - : Instance is created from the Module.
2. **Module Handling Buttons**
   - : Creating a new Module from scratch or from a base Module template.
   - : Uploading a new Module from a ZIP file.
   - : Creating a new Instance from the selected Module.
3. **Module Editing Buttons**
   - : Updating the Module. If you upload a configuration/Module, it may contain a newer version of a Module than the existing one of your configuration and this button is displayed. You can update the Module in your LARA with this button. See more information in the Module Update section.
   - : Resetting the Module to the original state.
   - : Downloading the Module (ZIP file).
   - : Deleting the Module (and all of its previously created Instances).

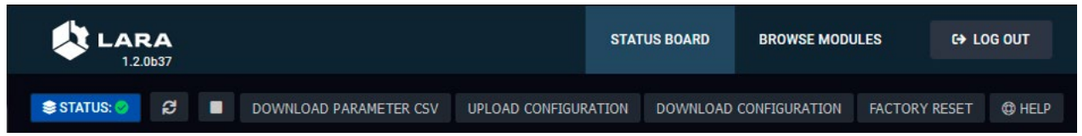
2.2. General Features

When the Firmware is Updated

If the firmware is updated with FW package v2.1 or newer version, the settings of the UCX/MMX2 device can be preserved, as well as the LARA configuration.

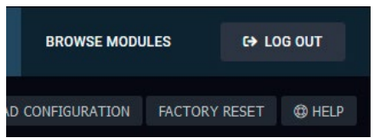
Configuration Download and Upload

The full LARA configuration (including Modules, Instances and Parameters with passwords as well) can be downloaded as a ZIP file. The file can be uploaded to the same device or another device of the same type. Use the **Upload Configuration/Download Configuration** buttons in the upper menu.



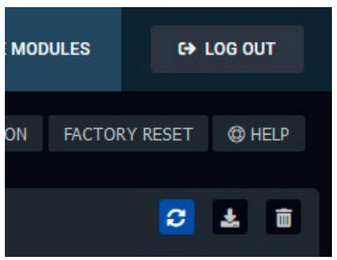
Factory Reset

All Modules and Instances can be deleted (after confirmation) by pressing the button.



Module Update

LARA contains factory Modules that cannot be removed. If you upload a configuration/Module, it may contain a newer version of a Module than the existing one of your configuration. In that case, a blue icon is displayed in the **Browse Modules** page:



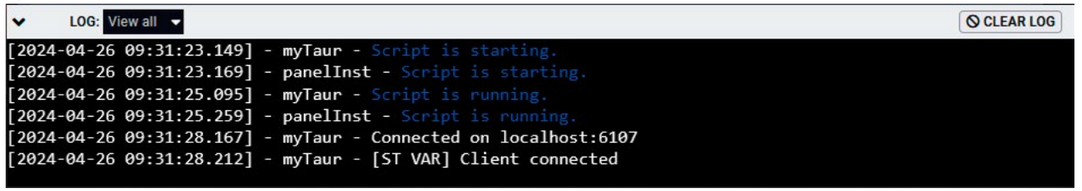
You can update the Module by pressing the button and the new version will be available in your device.

ATTENTION! Please note that the custom code disappears when updating the module.

INFO: Parameters, Events, Methods and Rules defined by the user in a factory Module are preserved when updating to a new version.

Log Window

The bottom part of the window can be toggled to show the log screen. The messages from each running Instance can be viewed here in real time, so you can follow the state of your system. You can view **all Instances** at same time or filter the messages to only **one Instance**. The section is resizable or it can be hidden by the down arrow:



TIPS AND TRICKS: The content of the log window can be copied to the clipboard.

2.2.1. Old/new Version Handling

If you want to update the firmware of your Lightware device that runs LARA, the best way is the follow these steps:

Step 1. Download the LARA configuration (for backup).

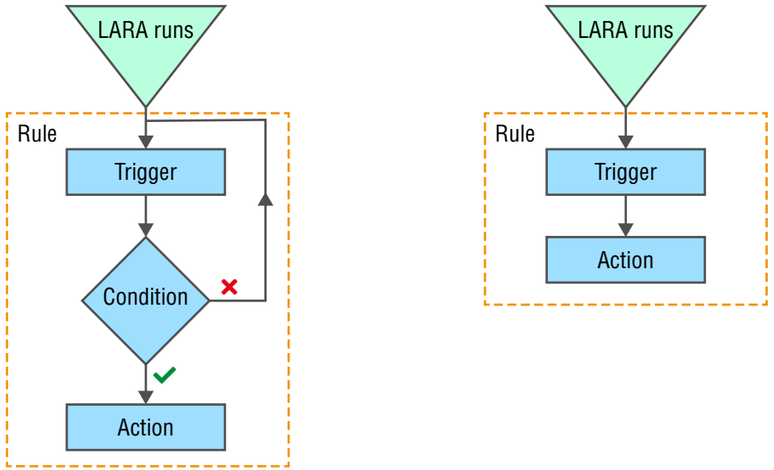
Step 2. Update the firmware of the device.

Step 3. Normally, the LARA configuration is kept as is, but if the factory default settings restored from some reason and the configuration disappeared, upload the backup configuration file.

Step 4. The new firmware may contain modules that are newer than in the running configuration. In that case, if you press the blue icon , the module will also be changed to the new one in the configuration. This step is recommended only if the module does not contain custom mode.

ATTENTION! Please note that the custom code is deleted when updating the module.

2.3. The Automation Process

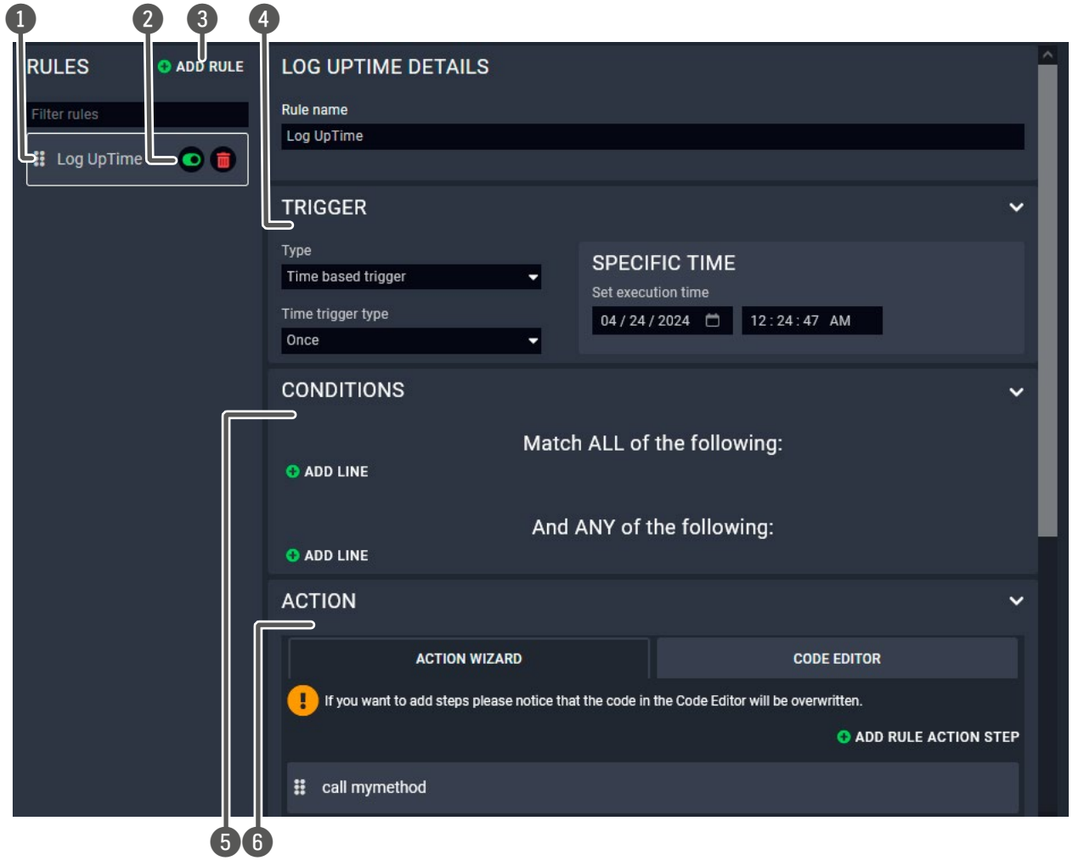


The Main Parts of a Rule – With and Without a Condition

ATTENTION! If your configuration contains more than one Module, it is recommended to create the Rules and the additional components (e.g. unique Methods) in a Logic Module.

2.3.1. The Rule

DEFINITION: The Rule is the part of the Module where the automation steps are described.



1	**List of Rules**	The user-defined Rules are listed here.
2	**Rule-handling buttons**	Delete a Rule Enable/disable a Rule
3	**Add new Rule**	User-defined Rules can be added with this button.
4	**Trigger panel**	Defining the Trigger.
5	**Condition panel**	Defining the Condition(s), optional.
6	**Action panel**	Defining the Action steps.

2.3.2. The Trigger

DEFINITION: The Trigger is the occurence (something has changed) that starts the automation process.

Duration Setting

If set, the Trigger must be detectable throughout the Duration time to launch the Action.

Trigger Types

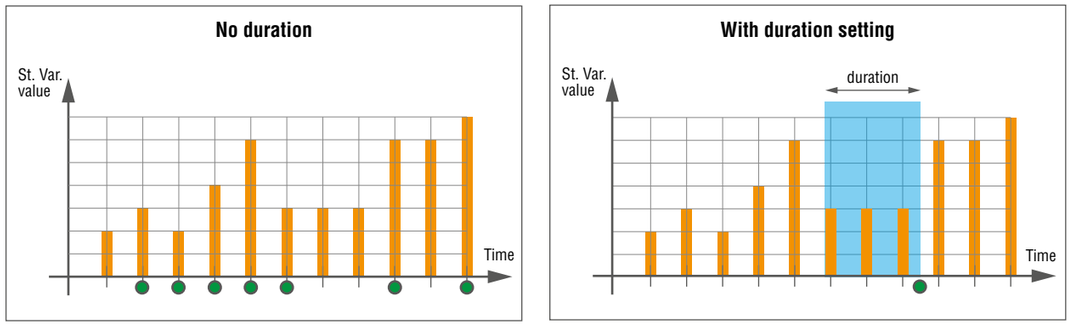
In case of Taurus UCX/MMX2 Module, the following options are available:

Event

The type means: 'something has happened'. This Trigger type is the same EVENT entity that has been used in previous LARA versions.

Variable Changed

A status variable is changed, but the value does not matter, only the fact of the change.



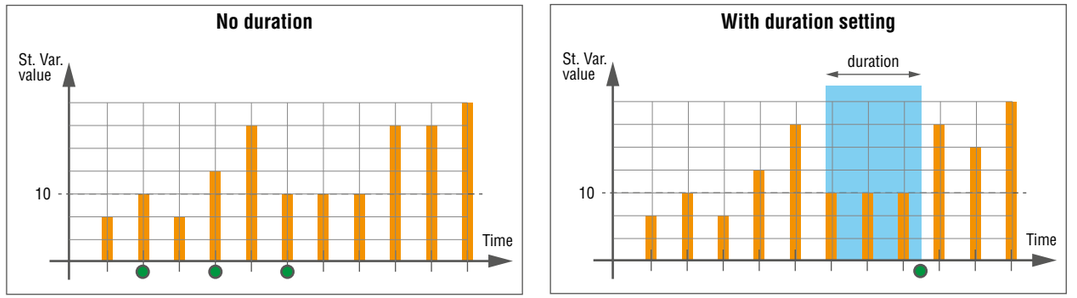
Action is launched

The 'Variable changed' Trigger type and the Actions

Variable Changed to Value

The new value can be evaluated with the following operators:

	number	string / json	boolean
operators	equal does not equal less than less than or equal greater than greater than or equal	equals to not equals to contains matches regexp	equals to not equals to



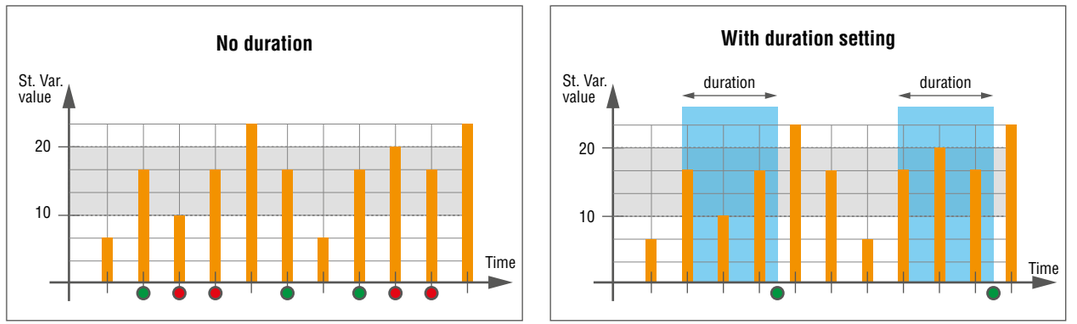
Action is launched

The 'Variable changed to value' Trigger type and the Actions

INFO: Above working method is valid in case of the **Less than/Greater than** operators, too.

Variable Changed Into Range

This type can be used only for number/any types. The options are similar as above, but in this case you can define a range instead of a specific value. The limit values are included in the range.



Action is launched

Action is not launched

The 'Variable changed into Range' Trigger type and the Actions

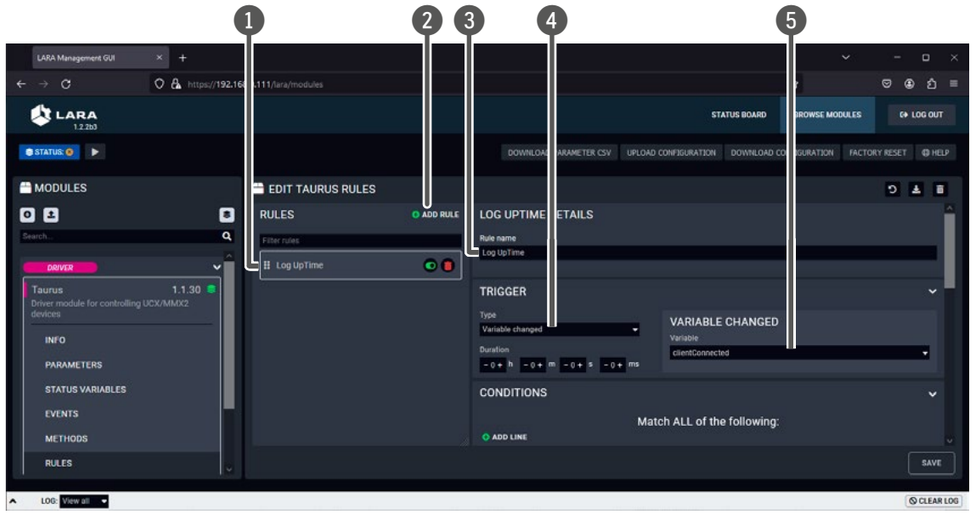
ATTENTION! Due to a known issue a malfunction occurs at this Trigger type as the Action should also be run at the red dot marked moments ● by design. The error will be corrected in a later LARA release.

Time-based Trigger

The Trigger is fulfilled if the current date/time meets the set value.

ATTENTION! For a proper working the Network Time Protocol (NTP) settings must be set precisely. See the Step 4 – Internal Time Setting section.

The Trigger Panel



The Trigger Panel in the Rule Editor

1. **List of Rules** — The user-defined Rules are listed here.
2. **Add new Rule** — Custom Rules can be added with this button.
3. **Name of the Rule**
4. **Type of the Trigger** — As follows:
   - Event
   - Variable changed
   - Variable changed to value
   - Variable changed into range
   - Time-based Trigger
5. **Variable selector** — In case of Status variable-based Trigger you can select the variable here.

ATTENTION! When LARA is started, the Status variables are empty in the first moment, **if default value is not defined**. Then the Status variables are updated immediately, but this 'value change' in the background will launch the 'Variable changed' Trigger types. This may cause Actions to execute. If default value is defined for the variable, this phenomenon can be avoided.

DIFFERENCE: If you have a configuration created with a previous version of LARA, the Events in the Rules are converted to Event-type Triggers.

See more information in the Status Variables section.

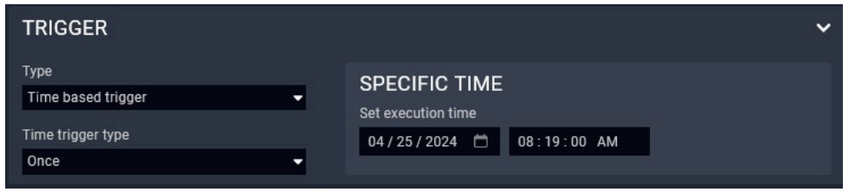
'Event' vs. 'Status Variable'

In many cases, you can set the **Trigger type** as an **Event** or use a **Status variable** for the same purpose. Actually, the Event is kept in LARA for compatibility reasons mainly. But there is also another reason to use Status variable instead. If you have a configuration with many Instances and you run the LARA configuration, the Instances are not started at exactly the same time but with a minimal delay – after each other. If a Rule refers to an Instance that has not been started yet, it may happen that the Event would not occur in that moment but later. If your configuration is built with only Status variables as Triggers, this would not happen due to the different working pecularity.

Time-based Trigger

ATTENTION! For a proper working the Network Time Protocol (NTP) settings must be set precisely. See the Step 4 – Internal Time Setting section.

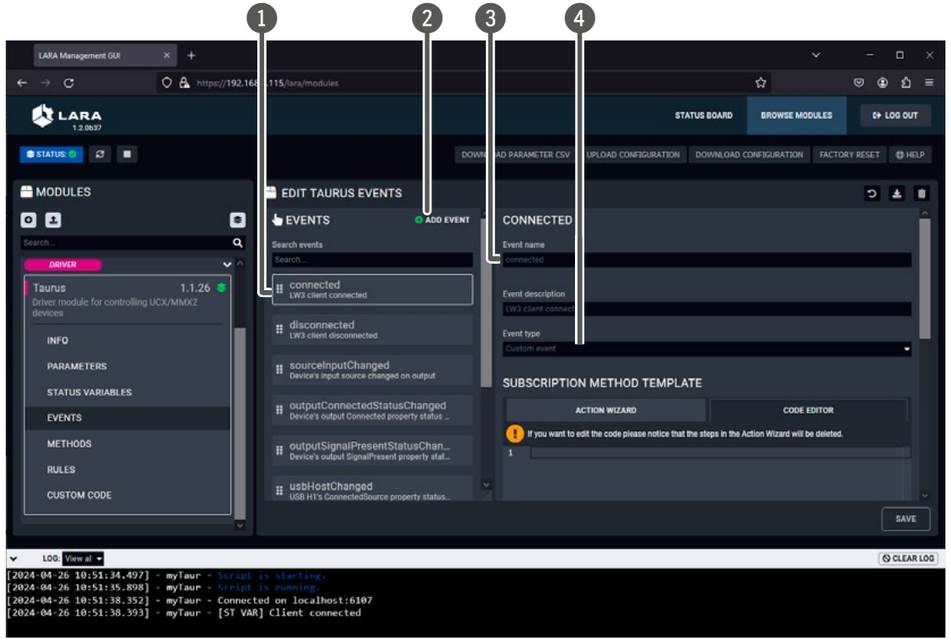
This option is to create a Rule that is triggered by a date/time-related occurance. If the current date/time matches with the set date/time setting (and if the Condition is fulfilled), the Action can be run – select the **Once** option.



The Trigger Panel

2.3.3. The Event

The Event means: something happened. Former LARA versions do not have Trigger but Event. The best way to use is to define a simple occurance as an Event and use the other Trigger types for more complicated cases.



1. **List of Events** — The factory default and the user-made Events are listed here. Factory Events cannot be edited/deleted.
2. **Add new Event** — User-defined Events can be added with this button.
3. **Name of the Event**
4. **Type of the Event** — The list of the available types depends on the Module. Factory default options:
   - Custom Event
   - Periodically dispatch Event
   - Dispatch Event at a specific time
   - HTML element clicked – this can be used for User Panel modules, see the User Module Descriptions chapter.

2.3.4. The Condition

> DEFINITION: The Condition is one ore more criteria that can be used together with the Trigger to allow running the Action step(s).

Optional, but if it has been set, it has to be fulfilled to run the Action. The way it works: if the Trigger occurs and the Condition can be detected **in that moment**, the Action can be launched. If the Condition cannot be detected, the Action will not be launched and the Rule process is finished. The process will start again if the Trigger occurs.

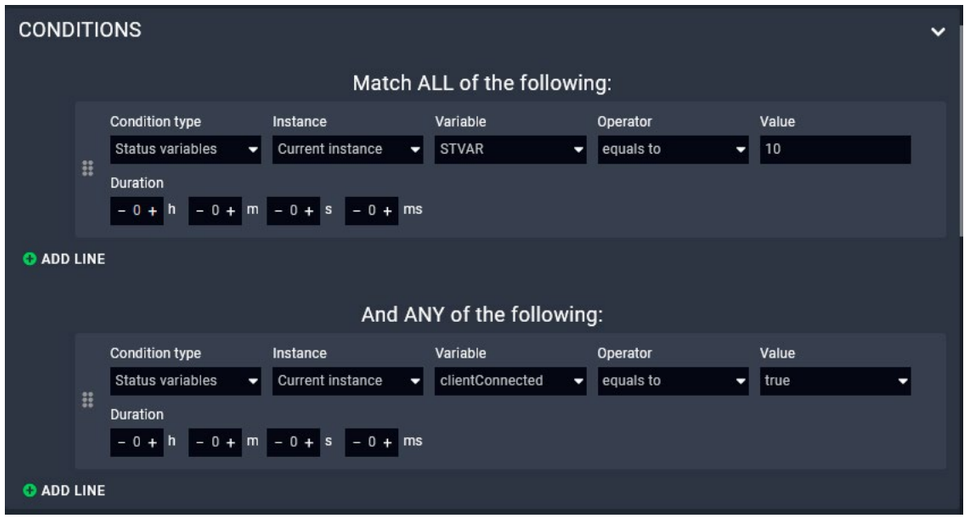
For defining Conditions the following options are available:

- Defining only **one Condition**,
- Defining more Conditions under 'Match ALL of the following' section (this means 'AND' logic): **all the defined Conditions** must be fulfilled at the same time to run the Action.
- Defining more Conditions under 'And ANY of the following' section (this means 'OR' logic): if **any of the defined Conditions** is fulfilled, the Action can be run.

Duration Setting: if set, the Condition must be detectable throughout the Duration time to launch the Action.

> **ATTENTION!** The Duration time is not measured from the Trigger but from **the last change** of the Status variable. This special feature is valid only if the **Condition type is Status variable**.

> **ATTENTION!** The logic connection between the two groups is 'AND'.



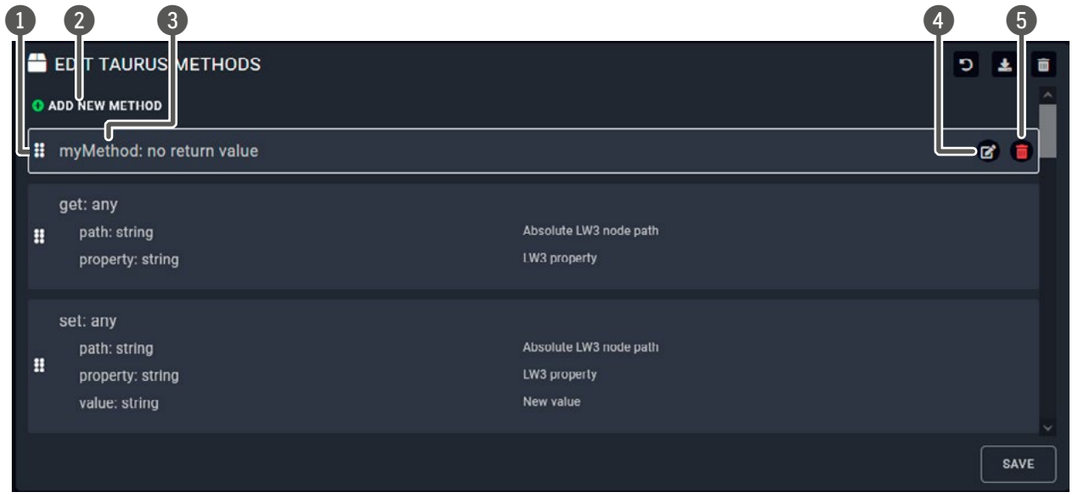
The Conditions Panel

2.3.5. The Method

DEFINITION: The Method is a pre-defined operation that can be executed as an Action of a Rule.

Methods are just like functions that can be called as an Action of a Rule. The Method can be a factory default Method or a user-defined custom Method. Some typical examples from the Taurus UCX/MMX2 Module:

- **GET**: querying a Parameter
- **SET**: setting a property
- **OPEN**: subscribing to a node path
- **CALLMETHOD**: calling an LW3 Method



The Method Editor Panel

1	**List of Methods**	The factory default and the user-made Methods are listed here.
2	**Add new Method**	User-defined Methods can be added with this button.
3	**Name of the Method**	
4	**Editing the Method**	Factory Methods cannot be edited. The icon is displayed if the mouse cursor is above the Method.
5	**Deleting the Method**	Factory Methods cannot be deleted. The icon is displayed if the mouse cursor is above the Method.

User-defined Methods

A Method can be added as follows:

Step 1. Press the **Add new Method** button in the Methods submenu.

Step 2. Press the ✎ button to edit the Method.

Step 3. Name the Method.

Step 4. **Add new Method Parameter** by the button (optional).

Step 5. Press the **Add Method step** button in the Code section and define the desired task with the Wizard. The available **Action types** are the same as described in connection with the Action steps – see the following section. Or select the Code editor and type a custom JavaScript code.

Step 6. Press the **Save** button.

INFO: The Method Parameter and the Return value can be defined via LARA GUI, the value can be assigned with the Events and Methods debug window. Even tough, the value assingment is not available with Wizard yet but with the Code Editor.

Testing a Method

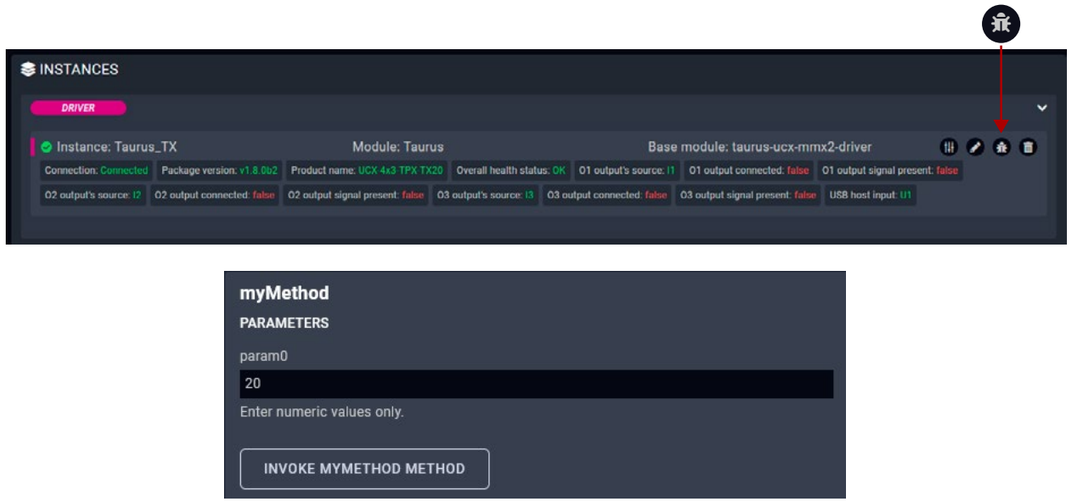
You can easily try if the defined Method works as it should. Closing the log window is recommended as it may cover the pop up window.

Step 1. Navigate to the **Status board**.

Step 2. Select the desired **Instance** and press the 🐞 button (see below). A new window will pop up showing the defined Events and Methods.

Step 3. Scroll to the **desired Method**. The desired Method Parameter values can be entered in the textboxes.

Step 4. Press the **Invoke <Method_name> Method** button.



Testing a Method

2.3.6. The Action

DEFINITION: The Action (step) is pre-defined task (operation) that is executed as the part of a Rule.

The Action is part of a Rule. Lightware-made Modules contain default Action types as follows:

- **Invoke Method**: a pre-defined Method (factory or user) can be run.
- **Log message**: a message to display in the log window (this can be a constant or a dynamic value).
- **Dispatch Event**: a pre-defined Event can be emulated (as if it has happened).
- **Wait**: setting a delay (in ms) before/after an Action step.
- **Display information on Status board**: a constant or a dynamic value (e.g. a Status variable) can be displayed in the log window; see the Displaying Information on the Status Board (Info Label) section.
- **Set Status Variable**: the value of a variable can be set.

User-defined Action Steps

The Action steps can be added as follows:

Step 1. Create the desired step as a **Method** (see the previous section).

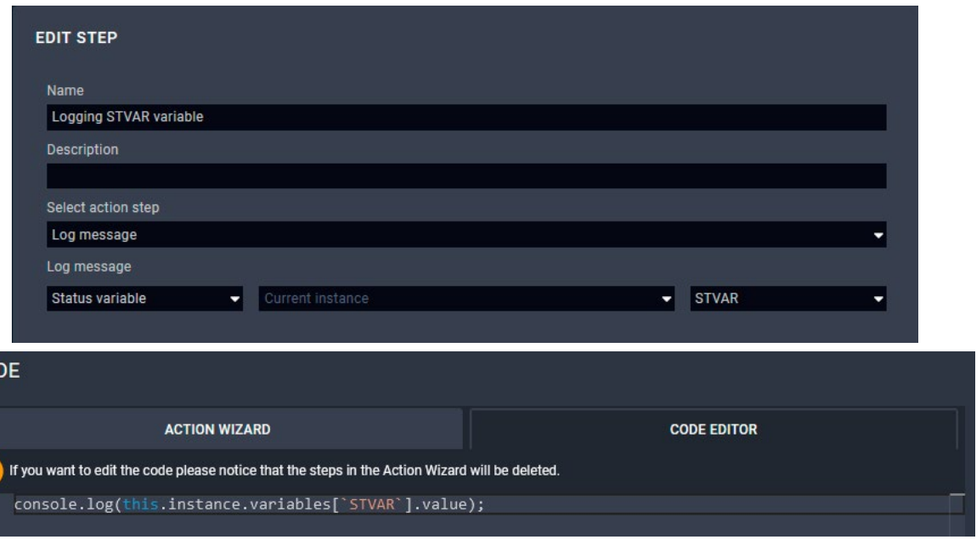
Step 2. Navigate to the desired Rule and set the Action: select the **Invoke Method** option.

Step 3. Select the previously set **Method** from the list.

Wizard vs. Custom Code

The Action step can be added with the **Action Wizard** or with the **Code Editor** as a JavaScript code (based on nodeJS programming language).

ATTENTION! If you change anything in the code, the wizard cannot be used to edit the Action step after that. Furthermore, if you use the wizard after the code editor, the new content would overwrite the code.

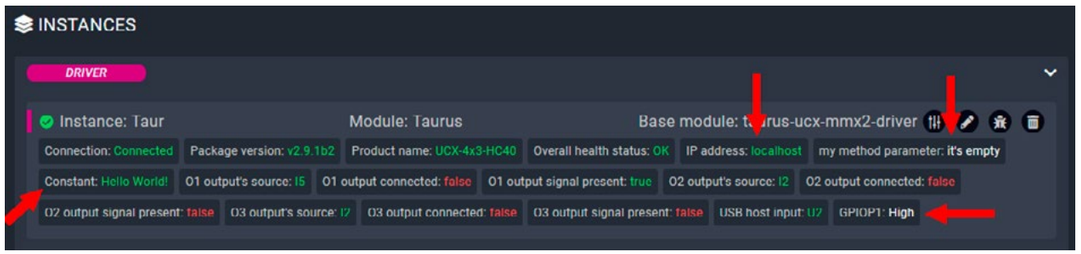


2.4. Info Labels

Factory default modules contain Info labels that are displayed on the Status Board but custom labels can also be added.

The following notes are referring to the labels:

- The order of the labels depends on the **starting sequence** of the Methods.
- Factory defined labels are being kept **up-to-date** continuously.
- The **custom labels** can be the following:
  - The value of a **Status variable** (e.g. GPIO P1 output value), see the Displaying Information on the Status Board (Info Label) section,
  - A **constant** value (e.g. 'Hello World!')
  - An **Instance parameter** (e.g. the IP address),
  - A **method parameter** (only if it is defined as a method step).



Custom Info Labels (marked) and the Default Labels

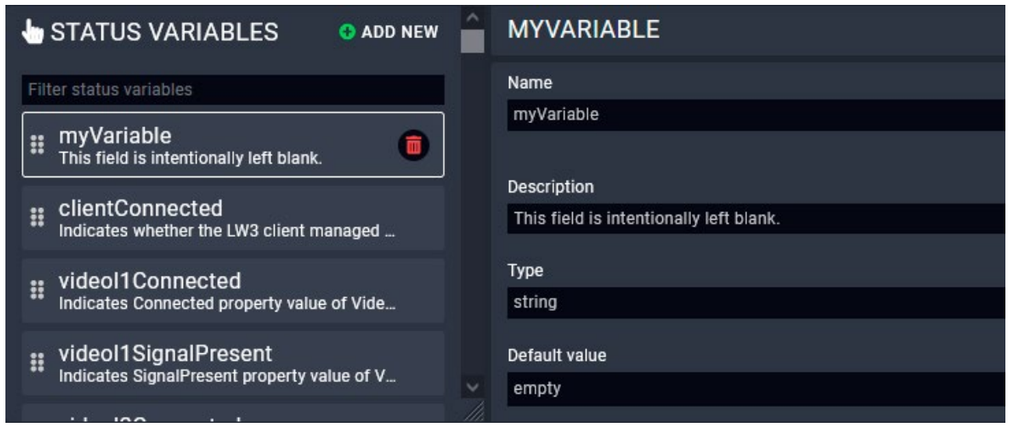
2.5. Status Variables

DEFINITION: The Status Variable can be used to store a piece of information. e.g. the value of an LW3 property.

Application Modes

- **Value assignment**: storing the value of a property,
- **Displaying Information** on the Status Board (Info Label),
- Using as a **reference in Rules**, see the The Trigger section.

INFO: LARA stores the timestamp of the last change of the variable. Time-sensitive Rules are based on that information (duration setting), see the Time-based Trigger section.



2.5.1. Value Assignment

Default Status Variables

The factory default Status variables are connected to LW3 properties and they are kept up-to-date in the background continuously.

For creating a unique Status variable and assigning the value, see the following section.

User-defined Status Variables

INFO: Creating the Status variable in the Logic Module is recommended.

You can create a new Status variable as follows:

Step 1. Navigate to the **Status variables** menu.
Step 2. Press the **Add new** button in the middle section.
Step 3. Fill the **Name** and **Type** fields; **Description** and **Default value** are optional.
Step 4. Press the **Save** button.

Variable types

- **number**: for storing numbers (decimals are accepted, **divided by a dot**).
- **string**: for storing characters and text.
- **boolean**: true/false.
- **json**: any kind of JSON object can be stored.
- **any:** any of the types above but without requirements.

ATTENTION! If you use comma as a decimal separator for a number, the value will be handled as a string.

Changing the Value of a User-defined Status Variable

The value of the Status variable can be changed with a Rule Action step or with a pre-defined Method. The value can be:

- **Constant**: a static value determined in this window,
- **Instance Parameter**: the value is coming from the selected Instance. (The Parameter is defined in the parent Module and the value can be different for each Instance. See the The Inside of the Module section.)
- **Status variable**: the value of another Status variable.
- **Method parameter** (only if the value is changed with a Method): the parameter defined in the module.

2.5.2. Displaying Information on the Status Board (Info Label)

Factory default modules contain Info labels that are displayed on the Status Board but custom labels can also be added. The label can get the value from a Status variable either (for other options please see the details in the Info Labels section) as follows:

You can display a Status variable as an Info label on the Status board as follows:

Step 1. **Create a Rule** and define a **Trigger**, or select an existing Rule.

Step 2. Press the **Add rule action step** button in the Action section (Wizard tab).

Step 3. Name the step and select the **Display information on Status Board** option.

Step 4. Type a **label**.

Step 5. Select the **Status variable** value type.

Step 6. Select the desired **Status variable**.

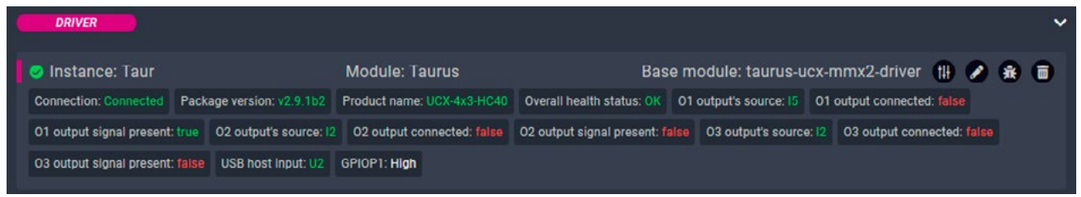
Step 7. Optionally set the **style** of the label.

Step 8. Press the **Save** button (this will close the pop up window) and press **Save** button in the main window, too.

INFO: The **Displaying information on the Status board** step can also be defined as a Method.

Example

The following example is about a case when a Status variable is selected from the default list and displayed on the Status Board as an Info label (**GPIOP1**).



Target

If the GPIO P1 output level is changed, the level is displayed and updated on the status board.

Operation

The label is displayed on the Status Board when LARA configuration is started. Once the output level of GPIO P1 is changed, the value is updated.

INFO: The 'GPIOP1' label appears when LARA is started. The Status variable is empty in the first moment (as this Status variable does not have default value). After that the Status variable is updated immediately. This 'value change' in the background would launch the 'Variable changed' Trigger type.

Preparation

- Taurus UCX/MMX2 Driver is defined under the **Modules** menu.
- Instance is created from the Module.
- The direction of the P1 port is set to **Output**.

Adding a New Rule

- Rule name: e.g. **Displaying GPIO P1 level**
- Trigger Type: **Variable changed**
- Variable Changed: **gpioP1Output**
- Action / **Add Rule Action step**
  - Action name: e.g. **GPIO1 update**
  - Select Action step: **Display information on Status Board**
  - Label: e.g. **GPIO1**
  - Value: **Status variable, gpioP1Output**
  - Style: as you wish

TIPS AND TRICKS: This Rule can easily be tested by toggling the GPIO Pin1 output level in LDC or web LDC.

2.6. Best Practices

How to Build Your Configuration

When building a LARA configuration with more, than one modules, it is worth to pay attention to:

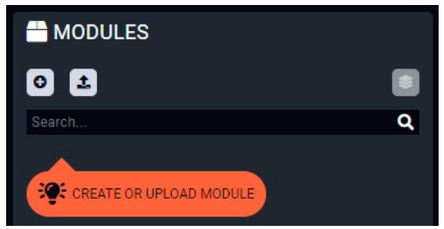
- Set the device-specific **Methods, Rules** and **Status Variables** in the **Driver** and **Script** Modules.
- Set the **Room automation related Rules** and **Status variables** in the **Logic** Module.
- When using a User Panel Module, set the **HTML-related Events** and **Rules** in the **User Panel** Module.

These principles help to create **re-usable Modules**.

Adding a New Module

As mentioned previously, the Driver Module is the interface to a device. A Module can be added on the **Browse Modules** page with:

- Pressing the button and selecting from the available list, or
- Pressing the button and adding a previously saved Module (as a ZIP file).



ATTENTION! The default Methods, Parameters and codes of the factory Modules cannot be edited/ deleted.

TIPS AND TRICKS: The modul (as a ZIP file) can be added via mouse drag&drop into this window.

Creating Instances

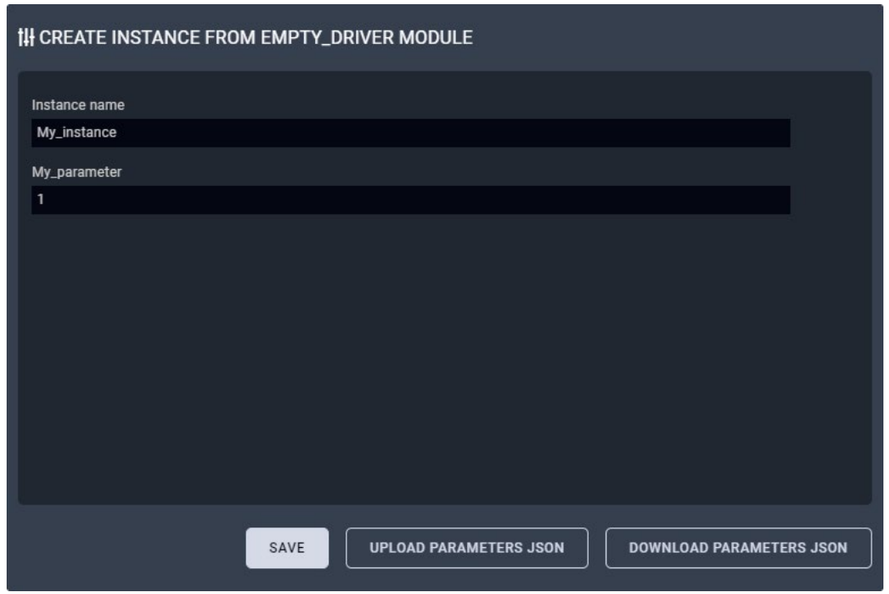
Step 1. Navigate to the **Browse Modules** page.

Step 2. Press the icon in the top of the **Modules** section.

Step 3. Type a **name** for the **Instance** and optionally set the value of the defined Parameters (if any).

Step 4. Press the **Save** button.

INFO: Please note that the name of the Instance cannot be changed in LARA after first saving.



The value of the Parameters can be set later as well:

Step 1. Navigate to the **Status Board**.

Step 2. Press the button in the end of the desired Instance line to open the **Parameter editor** window.

TIPS AND TRICKS: The Parameters and values of the Instance (seen on the screen) can be saved as a JSON file by the **download** button. Previously downloaded Parameters and values can be uploaded at this step by the **upload** button.

2.7. JavaScript Code Examples

The Modules may contain custom codes based on nodeJS programming language. Please find below the following JavaScript example codes that could help you write your custom codes.

Writing something to the log window (comments can be written after double slash)

```
console.log('Hello world!'); //this is a comment
```

Writing an error message to the output console (seen at the bottom part of the window, written in red)

```
console.error('Oh, no! Something terrible has happened!');
```

Accessing Instance Parameters

```
console.log('Device IP address is ', params.ip_address);
```

Waiting for 2000 milliseconds

```
await new Promise(r => setTimeout(r, '2000'));
```

Showing/updating information on status board with a given color

```
this.instance.updateStatus('Today weather', 'Sunny', {color: 'yellow'});
```

Emitting an Event from an Instance

```
this.instance.dispatchEvent('resulutionChanged', '1920x1080p60');
```

Running a custom code 10 seconds later

```
console.log('Enable the relay');
setTimeout(()=>{
  console.log('Disable the relay');
},10000);
```

Running a code every 5 seconds

```
setInterval(()=>{
  console.log('This is printed every 5 seconds');
},5000);
```

Running an Action when our Instance has fired an Event

```
this.instance.on('messageReceived', (message) => {
  console.log('A message has just been received:', message)
});
this.instance.dispatchEvent('messageReceived', 'Meow');
```

Running an Action when another Instance has fired an Event

```
this.getInstanceById('display').on('volumeChanged', (message) => {
  console.log('The display volume has just been changed:', message)
});
```

Invoking a Method from our own Instance

```
this.myFancyMethod(param1, param2);
```

Invoking a Method from another Instance

```
await InstanceApi.getInstanceById('display').send('powerOn');
```

Importing nodejs built-in Modules (see this documentation about the available Modules)

```
//import net Module
var net = require('net');
```

Writing a file

```
this.fs.writeFileSync(this.Instance.getLocalStoragePath()+'/data.txt', 'Hello world');
```

Reading a file

```
var data = this.fs.readFileSync(this.Instance.getLocalStoragePath()+'/data.txt');
```

More information and examples about file operations: https://nodejs.dev/learn/the-nodejs-fs-Module

Sending and receiving response to/from a client over TCP/IP

```
var net = require('net');
var client = new net.Socket();
client.connect(1337, '192.168.1.1', function() {
  console.log('Connected');
  client.write('Message from the Client.');
});

client.on('data', function(data) {
  console.log('Received: ' + data);
  client.destroy(); // terminate client after server's response
});

client.on('close', function() {   console.log('Connection closed');
});
```

Sending an HTTP GET message

```
const http = require('http');

http.get('http://192.168.0.1', (res) => {
  const { statusCode } = res;
  const contentType = res.headers['content-type'];
  if (statusCode !== 200) {
    console.log('Request Failed. Status Code: ', statusCode);
    res.resume();
    return;
  }
  res.setEncoding('utf8');
  let rawData = '';
  res.on('data', (chunk) => { rawData += chunk; });
  res.on('end', () => {
    console.log('Received: ', rawData);
  });
}).on('error', (e) => {
  console.error('Got error:',e);
});
```

2.8. External Links

The following documentations help becoming a more professional user of LARA:

The webpage of LARA:

https://lightware.com/lara

The nodeJS programming language – for creating custom codes:

https://nodejs.org/en/docs/

The markdown formatting – for editing a cool Module description:

https://daringfireball.net/projects/markdown/

Reserved Words – good to know when defining Parameters:

https://www.w3schools.com/js/js_reserved.asp

3

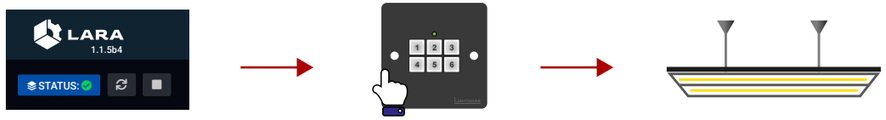
Sample Configuration

This chapter is about building up a whole configuration. The example starts from scratch as LARA contains no configuration.

3.1. Basic LED Toggle

What will happen?

A simple button (dry contact) is connected to GPIO PIN1 and an LED to PIN4 of the UCX Taurus device. When the button is pressed, the LED is toggled to be switched on or off.



LARA starts

- Taurus is connected
- GPIO P4 set to **output**
- Log message sent

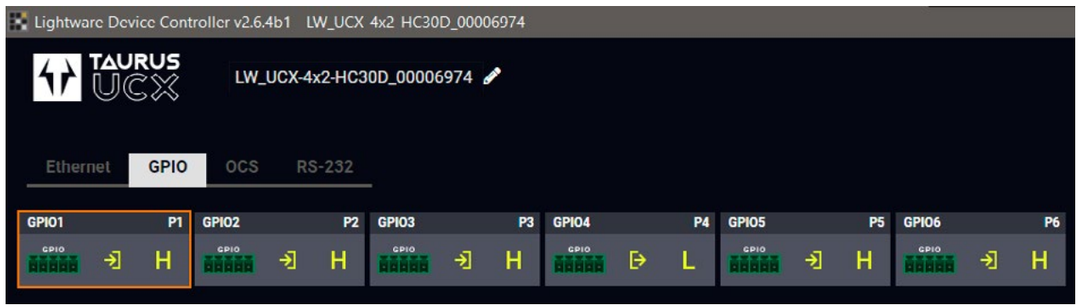
A button is pressed

- GPIO P1 becomes **low** for a moment

LED light is toggled

- GPIO P4 state is toggled from low to high or high to low
- Log message sent

TIPS AND TRICKS: If you do not have devices to connect to the GPIO, you can Trigger and monitor the state changes in LDC:

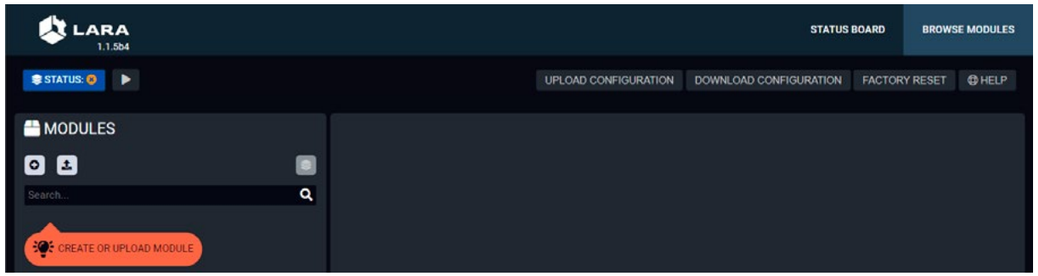


Configuration Steps

Step 1. Check the **GPIO pin directions** in LDC or web LDC and set the desired option.
Step 1. Creating the **Driver** Module(s).
Step 2. Creating the **Logic** Module.
Step 3. Creating **Instances** from the Modules.
Step 4. Creating **Events** in the Drived Module(s).
Step 5. Creating **Rules** in the Modules.
Step 6. **Starting** the configuration.

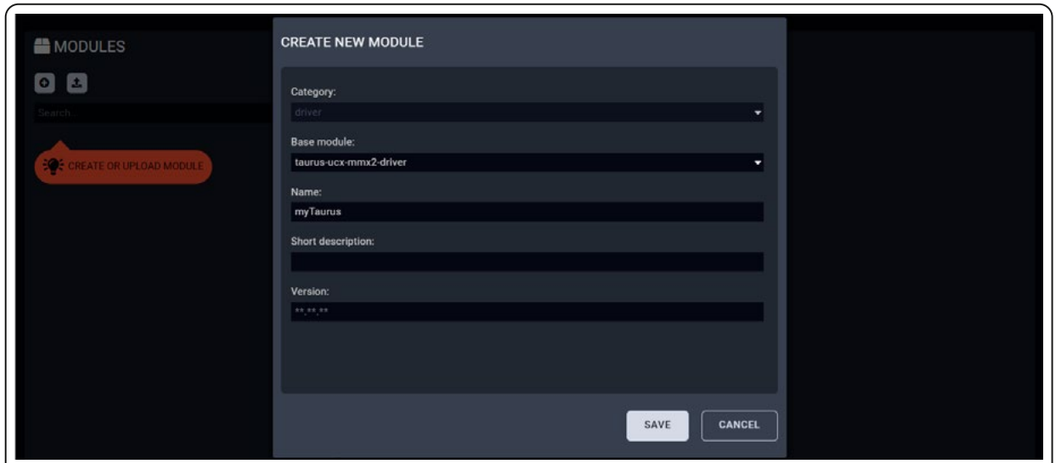
Step 1. Creating the Driver Module: *taurus-ucx-mmx2-Driver*

Add new Module



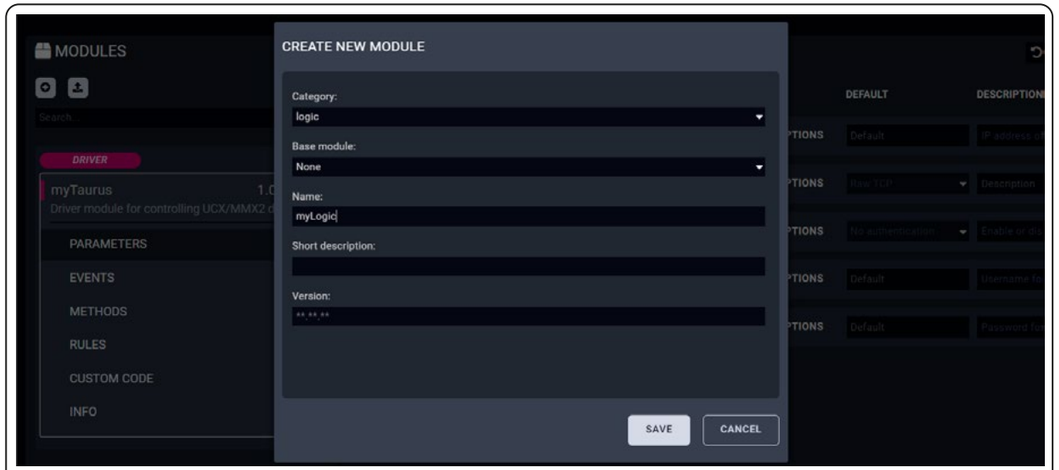
- Open LARA and navigate to the **Browse Modules** page.
- Create a **new module.**

Create Taurus Driver



- Select the **Category**: driver.
- Select the **Base module**: taurus-ucx-mmx2-driver.
- Type a **name** for the module.
- Press the **Save** button.

Step 2. Creating the Logic Module: *Logic*

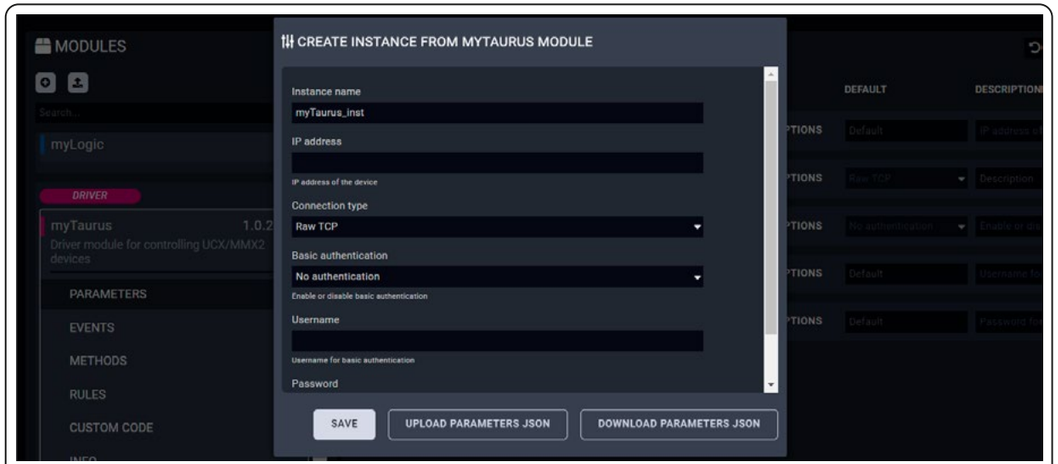


- Create a **new module** in the **Browse modules** page.
- Select the **Category**: logic.
- Type a **name** for the module.
- Press the **Save** button.

Step 3. Creating the Instances from the Modules

- Select the Taurus (myTaurus) driver in the **Browse modules** page.
- Press the icon.

Saving the *MyTaurus_inst* Instance

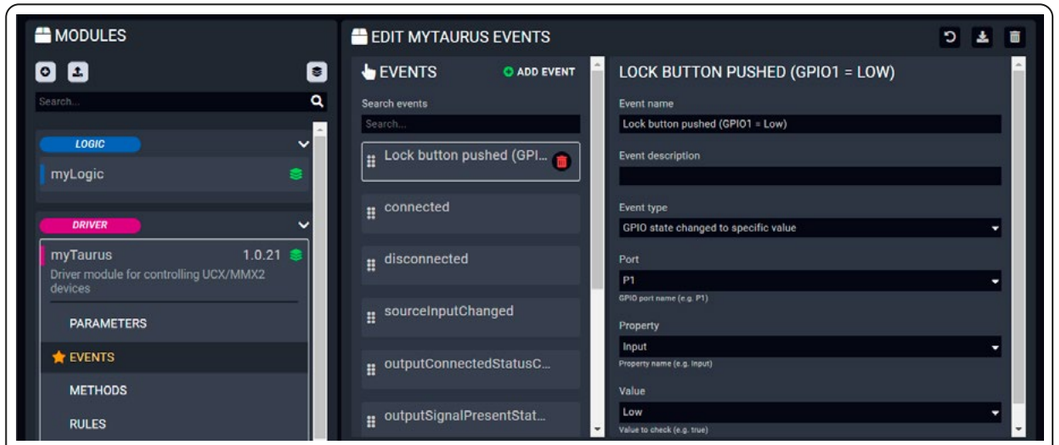


- Type a **name** for the instance.
- Type 'localhost' in the **IP address** box if this is the device that runs LARA.
- Press the **Save** button.

Saving the *myLogic_inst* Instance

- Select the logic module in the **Browse modules** page and press the icon.
- Type a **name** for the instance.
- Press the **Save** button.

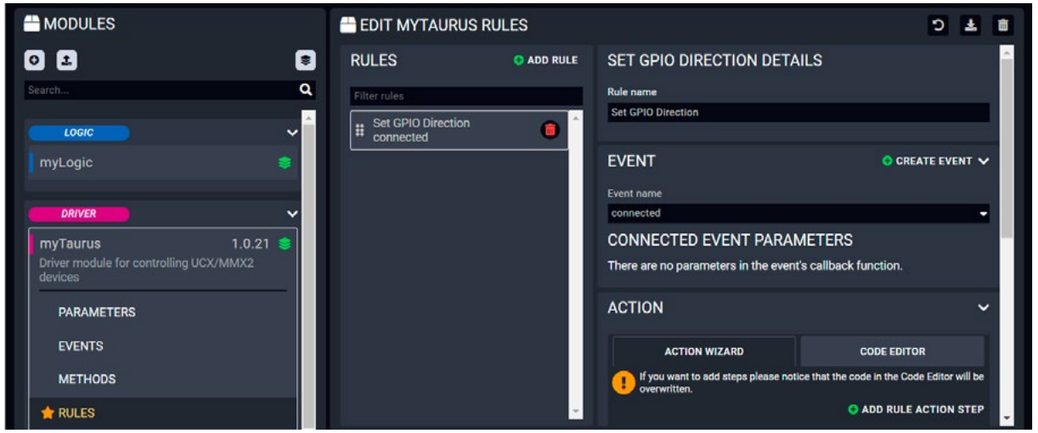
Step 4. Creating Event(s) in the Driver Module



- Select the **Taurus driver (myTaurus)** in the **Browse modules** page, then navigate to the **Events** page.
- Press the **Add event** button.
- Type a **name** for the event: 'Lock button pushed (GPIO1 = Low)'.
- Select the **event type**: 'GPIO state changed to specific value'.
- Select the **port**: 'P1'.
- Select the **property**: 'Input'.
- Select the **value**: 'Low'.
- Press the **Save** button.

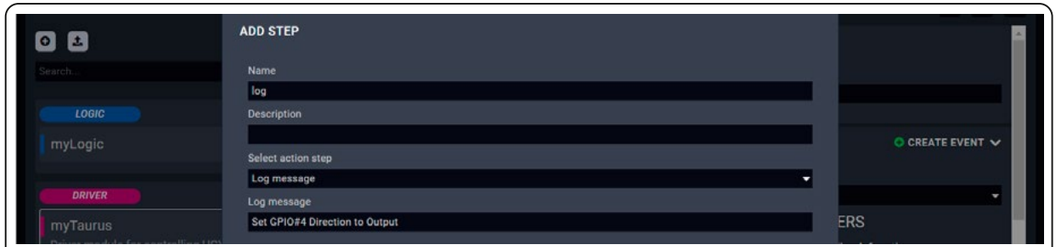
Step 5. Creating Rules in the Modules

Initializing the GPIO pin in the Taurus when LARA starts – *Taurus* Module



- Select the **Taurus driver (myTaurus)** in the **Browse modules** page, then navigate to the **Rules** page.
- Press the **Add rule** button.
- Type a **name** for the rule: 'Set GPIO Direction'.
- Select the **event**: 'connected'.
- Press the **Save** button.

Adding the first Action step: logging a short message

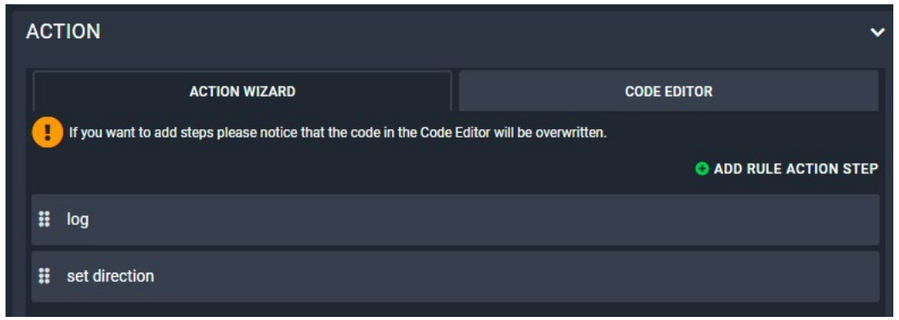


- Press the **Add rule action step** button.
- Type a **name** for the step: 'log'.
- Select the **action step**: 'Log message'.
- Type the desired text, e.g.: 'Set GPIO4 Direction to Output'.
- Press the **Save** button.

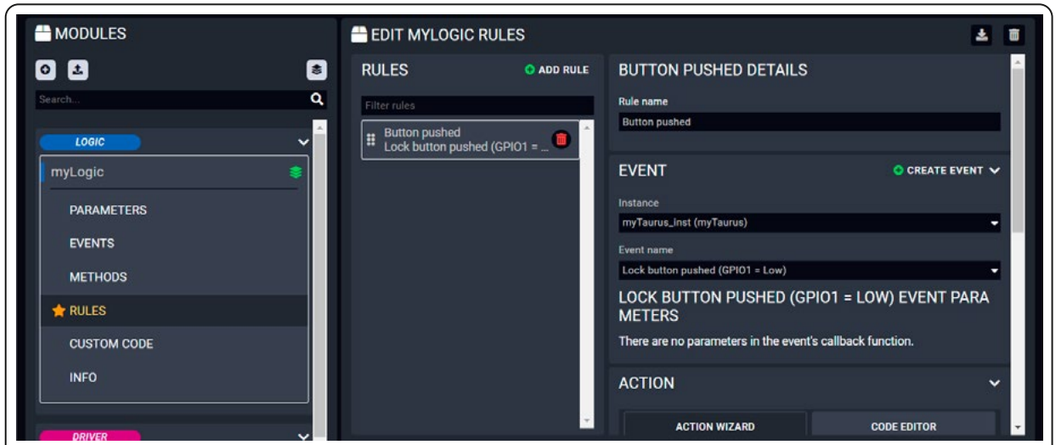
Adding the second Action step: setting the GPIO pin direction

- Press the **Add rule action step** button.
- Type a **name** for the step: 'set direction'.
- Select the **action step**: 'Invoke method'.
- Select the **method**: 'setGpioDirection'.
- Type the **port** number: 'P4'.
- Type the port **direction**: 'Output'.
- Press the **Save** button in this window.
- Press the **Save** button in the main window, too.

After that, the Action steps look like this:

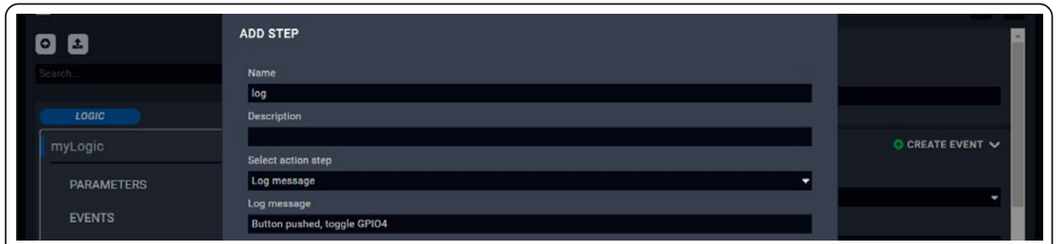


Toggle the output level of P4 – *Logic* Module



- Select the **Logic module** in the **Browse modules** page, then navigate to the **Rules** page.
- Press the **Add rule** button.
- Type a **name** for the rule: 'Button pushed'.
- Select the **instance**: 'myTaurus_inst'.
- Select the event name (that you have just created): 'Lock button pushed (GPIO1 = Low)'.
- Press the **Save** button.

Adding the first Action step: logging a short message



- Press the **Add rule action step** button.
- Type a **name** for the step: 'log'.
- Select the **action step**: 'Log message'.
- Type the desired **text**, e.g.: 'Button pushed, toggle GPIO4'.
- Press the **Save** button in this window.

Adding the second Action step: toggle the GPIO pin output level

- Press the **Add rule action step** button.
- Type a **name** for the step: 'toggle LED (GPIO4)'.
- Select the **action step**: 'Invoke method'.
- Select the **instance**: 'myTaurus_inst'.
- Select the **method**: 'callMethod'.
- Type the **LW3 node path** of the method: '/V1/MEDIA/GPIO/P4'.
- Type the **method**: 'toggle()'.
- Press the **Save** button in this window.
- Press the **Save** button in the main window, too.

Step 6. Starting the configuration

The Configuration is ready to use, run LARA!

TIPS AND TRICKS: This configuration can be downloaded from Lightware's Cloud storage, so you can compare your work with the solution.

4

Factory Module Descriptions

This chapter is about the Modules that have been developed by Lightware. The properties and the configuration steps are described in the coming sections.

4.1. Generic TCP/IP Device Module

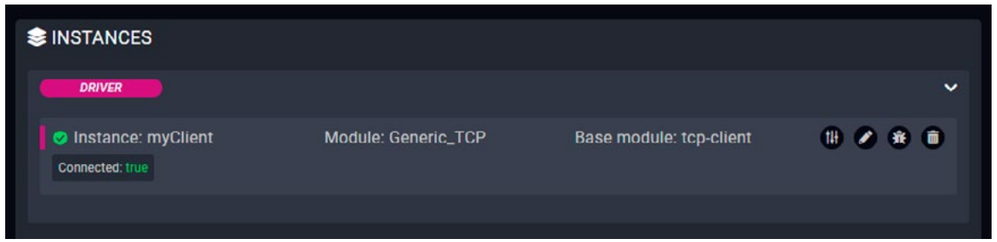
Introduction

This Module is for controlling any third-party device that can be queried/controlled over TCP/IP protocol.

Dashbord Content

The following status indicator is displayed on the Status board:

- Connection state of the device



Defined Parameters

- **ipAddressOrHost:** The IP address or the host name of the device.
- **portNumber**: Port to use when connecting.
- **permanent**: Boolean Parameter to determine if the connection should be kept open and not only for sending messages (receiving data is not available if false).
- **autoConnect**: Boolean Parameter to determine if client should automatically retry the connection if not connected.
- **autoConnectInterval**: Interval (in ms) for trying to connect.
- **frameDelimiter**: Delimiter to determine the end of a frame (e.g. `\r\n`).
- **frameDelimiterIsHex**: Specifying if the provided data is a hexadecimal string.
- **frameTimeout**: If the set time (in ms) has elapsed, the received data is considered as a frame.
- **keepAliveDelay**: Keep alive signal will be sent after the set interval time (in ms) has elapsed.

They can be referred in the JavaScript code as e.g. `params.portNumber`

Defined Events

- **connected**: The TCP/IP device with the specified ipAddress is connected.
- **disconnected**: The TCP/IP device with the specified ipAddress is disconnected.
- **error**: e.g. Problem in the data transmission, e.g. wrongly set frame delimiter. `'errormessage'` Parameter is defined in this Event for the error code.
- **frameReceived**: The received frame; `'frame Parameter'` is defined in this Event. It can be used to analyse the content of the received frame.

Defined Methods

- **send**: Sending a text message without a delimiter. `'message'` Parameter is defined in this Method.
- **sendFrame**: Sending a text message with a frame delimiter. `'message'` Parameter is defined in this Method.
- **sendHex**: Sending a message in hexadecimal format **without delimiter**. `'message'` Parameter is defined in this Method.

Defined Rules

No Rules are defined in this Module.

Custom Code

The code of the factory module in JavaScript.

> **ATTENTION!** If the GUI tools cannot solve a problem, custom code can be added to the end of the code, but please note that if you upgrade the driver module (blue button) the custom sections will be deleted.

4.2. Generic LW3 Device Module

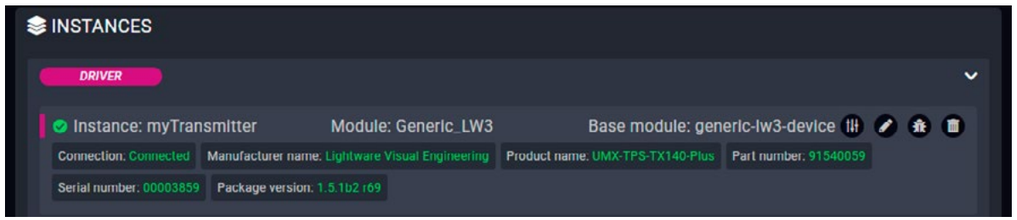
Introduction

This Module is for controlling a Lightware device that supports LW3 protocol. Please refer to the User's Manual of the desired Lightware device for whether it can be controlled with LW3 commands.

Dashbord Content

The following status indicators are displayed on the Status board:

- Connection state of the device
- Manufacturer name
- Product name
- Part number
- Serial number
- FW package version



Defined Parameters

- **ipAddress**
- **portNumber**

They can be referred in the JavaScript code as e.g. **`params.ipAddress`**

Defined Events

- **connected**: The LW3 device with the specified ipAddress is connected.
- **disconnected**: The LW3 device with the specified ipAddress is disconnected.
- **LW3 property changed**: The value of a specific LW3 property is changed to the defined value. **`'path'`**, **`'property'`** and **`'value'`** Parameters are defined in this Event.

Defined Methods

These methods can be used the same way as the ordinary LW3 GET, SET, OPEN, CLOSE and CALL commands.

- **get**: Querying the value of a specific property (with path). **`'path'`** and **`'property'`** Parameters are defined in this Method.
- **set**: Setting the value of a specific writable property (with path). **`'path'`**, **`'property'`** and **`'value'`** Parameters are defined in this Method.
- **open**: Subscribing to a specific node. It means that the user will get a notification if the property changes. **`'path'`** Parameter is defined in this Method.
- **close**: Unsubscribing from a specific node. **`'path'`** Parameter is defined in this Method.
- **callMethod**: Calling a specific LW3 Method; does not work for an LW3 property. **`'path'`** and **`'Method'`** Parameters are defined in this Method (Parameters are optional). Please note that wildchar (*) cannot be used in case of the **`'path'`** Parameter.

Defined Rules

No Rules are defined in this Module.

Custom Code

The code of the factory module in JavaScript.

> **ATTENTION!** If the GUI tools cannot solve a problem, custom code can be added to the end of the code, but please note that if you upgrade the driver module (blue button) the custom sections will be deleted.

4.3. Taurus UCX/MMX2 Driver Module

Introduction

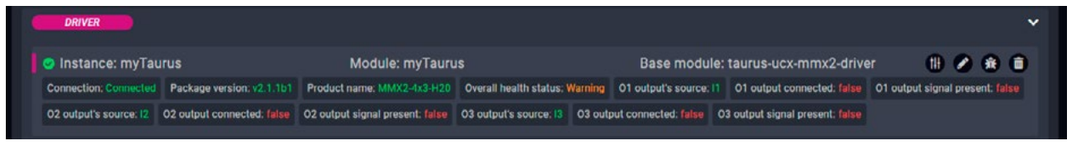
This Module is for controlling Lightware's UCX and MMX2 devices with a few Events and Methods. The Module covers the following models:

- UCX-2x1-HC30
- UCX-2x2-H30
- UCX-4x2-HC30
- UCX-4x2-HC30D
- UCX-4x3-HC40
- UCX-4x2-HC40
- UCX-4x3-HC40-BD
- MMX2-4x3-H20
- MMX2-4x1-H20
- UCX-4x3-TPX-TX20
- UCX-2x1-TPX-TX20

Dashbord Content

The following status indicators are displayed on the Status board in the row of the Instance:

- Connection state of the device
- Firmware package version
- Product name
- Overall health status
- Outputs' source
- Outputs' connection state
- Outputs' signal presence
- USB host input (if any)



Defined Parameters

- ipAddress
- connectionType
  - Raw TCP
  - Websocket
  - Secure Websocket
- authenticationType
  - no authentication
  - basic authentication
- username
- password

They can be referred in the JavaScript code as e.g. `params.ipAddress`

Defined Events

- **connected**: The UCX/MMX2 device with the specified ipAddress is connected.
- **disconnected**: The UCX/MMX2 device with the specified ipAddress is disconnected.
- **sourceInputChanged**: Another source is switched to the specified output. `'outoutPort'` and `'sourcePort'` Parameters are defined in this Event.
- **outputConnectedStatusChanged**: Sink device is connected/disconnected to/from the specified output. `'outputPort'` and `'status'` Parameters are defined in this Event.
- **outputSignalPresentStatusChanged**: The signal presence is changed on the specified output (e.g. signal is present/not present). `'outputPort'` and `'status'` Parameters are defined in this Event.
- **usbHostChanged**: Another USB host device is selected for the USB peripherals. `'sourcePort'` Parameter is defined in this Event to specify the USB input port.
- **overallHealthChanged**: The value of the /V1/MANAGEMENT/HEALTH/OverallHealthState property is changed. `'healthStatus'` Parameter is defined in the Event.
- **healthAlert**: The value of a specific Parameter under /V1/MANAGEMENT/HEALTH node is changed to the defined value. `'property'` and `'value'` Parameters are defined in this Event.
- **LW3 property changed:** The value of a specific LW3 property is changed to the defined value. `'path'`, `'property'` and `'value'` Parameters are defined in this Event.

Defined Methods

- **get**: Querying the value of a specific property (with path). `'path'` and `'property'` Parameters are defined in this Method.
- **set**: Setting the value of a specific writable property (with path). `'path'`, `'property'`and `'value'` Parameters are defined in this Method.
- **open**: Subscribing to a specific node. It means that the user will get a notification if the property changes. `'path'` Parameter is defined in this Method.
- **close**: Unsubscribing from a specific node. `'path'` Parameter is defined in this Method.
- **callMethod**: Calling a specific LW3 Method; does not work for an LW3 property. `'path'`, `'Method'` and `'Parameters'` are defined in this Method (Parameters are optional.)
- **switchCrosspoint**: Changing the crosspoint state of a specific layer (AUDIO, VIDEO or USB). Please note that USB is available only for specific models.
- **setGpioDirection**: Setting the direction (input/output) of a specific GPIO pin. `'port'` and `'direction'` Parameters are defined in this Method.
- **setGpioState**: Setting the state (low/high) of a specific GPIO pin. `'port'` and `'state'` Parameters are defined in this Method.
- **emulateEdid**: Setting an EDID (source) to emulate on a specific input port (destination). `'source'` and `'destination'` Parameters are defined in this Method. Source can be: factory, dynamic, user. Destination can be: emulated.
- **uploadEdid**: Uploading a HEX-format EDID to a User EDID slot. Target and data Parameters are defined in this Method.
- **changeAnalogAudioVolumeDB**: Setting the volume level of a specific analog audio output in dB. `'port'` and `'value'` Parameters are defined in this Method.
- **changeAnalogAudioVolumePercent**: Setting the volume level of a specific analog audio output in percentage. `'port'`and `'value'` Parameters are defined in this Method.

For further details about the LW3 properties or Methods, please see the User's Manual of the device.

Defined Rules

No Rules are defined in this Module.

Custom Code

The code of the factory module in JavaScript.

> **ATTENTION!** If the GUI tools cannot solve a problem, custom code can be added to the end of the code, but please note that if you upgrade the driver module (blue button) the custom sections will be deleted.

4.4. Taurus CEC Driver Module

Introduction

This Module is for sending CEC commands (with REST API) to a sink (display) device connected to Lightware's UCX or MMX2 devices over HDMI output.

> INFO: Since LARA cannot access to a hardware directly, the REST API interface is used for this purpose. That's why the TCP IP port number and physical connection is necessary for the communication.

Dashbord Content

The following status indicators are displayed on the Status board in the row of the Instance:

- Connection state of the device
- Controlled output port
- Last CEC command timestamp (optional)
- Last CEC command status (optional)

Defined Parameters

- **outputPort**: Video output port of the UCX/MMX2 device for CEC command requests.
- **idlePollingFrequency**: Time in seconds for CEC polling on output (0 = no idle polling).
- **ipAddressOrHost**: The IP address or the host name of the device.
- **protocol**: Protocol to be used for connection.
- **portNumber**: Port number (80 for HTTP, 443 for HTTPS) to be used for connection.
- **authenticationType**: Enable or disable basic authentication.
- **username**: User name for basic authentication.
- **password**: Password for basic authentication.
- **customHeaders**: Setting custom headers with the following syntax: header-name:value.
- **cecMessageStats**: Enable the last sent CEC message statistics on the Status Board.
- **debugMode**: Enables the debug mode for sent CEC messages. All of the details of the messages will be printed to the console.

They can be referred in the JavaScript code as e.g. `params.outputPort.`

Defined Events

- **error**: The CEC command sending was not successful.
- **availableOnCEC**: The device responds to CEC REST requests.
- **unavailableOnCEC**: Device do not respond to CEC REST requests.

Defined Methods

- **powerOn**: Sending the Power On CEC command.
- **imageViewOn**: Sending the Image View On CEC command (use this if the Power On command does not work).
- **powerOff**: Sending the Power Off CEC command.
- **standby**: Sending the Standby CEC command (use this if the Power Off command does not work).
- **mute**: Sending the Mute CEC command.
- **unmute**: Sending the Unmute CEC command.
- **volumeUp**: Sending the Volume Up CEC command.
- **volumeDown**: Sending the Volume Down CEC command.
- **customCommand**: Sending a custom command via CEC. The command Parameter is defined for this purpose in this Method. The custom command can be a HEX string. Only hexadecimal characters are allowed, spaces can be used as delimiters.

Defined Rules

No Rules are defined in this Module.

Custom Code

The code of the factory module in JavaScript.

> **ATTENTION!** If the GUI tools cannot solve a problem, custom code can be added to the end of the code, but please note that if you upgrade the driver module (blue button) the custom sections will be deleted.

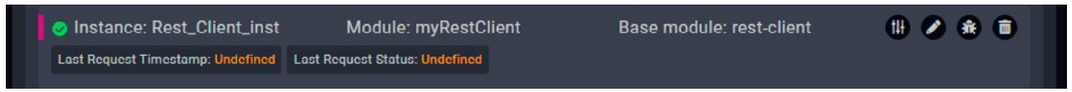
4.5. Generic Rest Client Driver Module

Introduction

This Module is for handling REST API-based communication.

Dashbord Content

- **Last request timestamp**: Last request status:



Defined Parameters

- **certAuthEnable**: Enabling certificate-based authentication. Disable this when communicating with Lightware devices via HTTPS.
  - true
  - false
- **ipAddressOrHost**: IP address or host name of device.
- **protocol**: Protocol to be used for connection:
  - HTTP
  - HTTPS
- **portNumber**: 80 for HTTP, 443 for HTTPS connection.
- **authenticationType**:
  - No authentication
  - Basic authentication
- **username**: user name for basic authentication.
- **password**: password for basic authentication.
- **customHeaders**: Setting custom headers with the following syntax: header-name:value.

They can be referred in the JavaScript code as e.g. `params.username`.

Defined Events

- **error**
- **responseReceived**

Defined Methods

- **put**: PUT Method. Defined Parameters:
  - path: string type
  - Parameters: JSON type (optional)
  - customHeaders: JSON type (optional)
- **post**: POST Method. Defined Parameters:
  - path: string type
  - Parameters: JSON type (optional)
  - customHeaders: JSON type (optional)
- **get**: GET Method. Defined Parameters:
  - path: string type
  - Parameters: JSON type (optional)
  - customHeaders: JSON type (optional)
- **del**: DEL Method. Defined Parameters:
  - path: string type
  - Parameters: JSON type (optional)
  - customHeaders: JSON type (optional)

Defined Rules

No Rules are defined in this Module.

Custom Code

The code of the factory module in JavaScript.

> **ATTENTION!** If the GUI tools cannot solve a problem, custom code can be added to the end of the code, but please note that if you upgrade the driver module (blue button) the custom sections will be deleted.

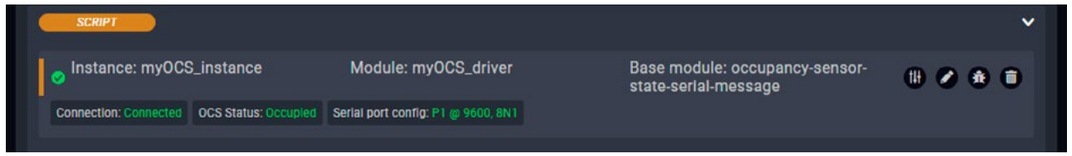
4.6. Occupancy Sensor and Serial Message Script Module

Introduction

This script Module is for Lightware Taurus and MMX2 series devices for sending serial messages based on the occupancy sensor status.

Dashbord Content

- **Connection state**: UCX/MMX2 device connection state
- **OCS Status**: low/high level state
- **Serial Port config**: optional



Defined Parameters

- **ipAddress**
- **connectionType**
  - Raw TCP
  - Websocket
  - Secure Websocket
- **authenticationType**
  - No authentication
  - Basic authentication
- **username**
- **password**
- **serialPort**: port number (e.g. P1)
- **baudRate**: Baud rate of the port (e.g. 9600)
- **stopBits**: stop bits (e.g. 1)
- **parity**: parity setting (e.g. None)
- **occupiedMessageType**:
  - string (it may contain unicode characters e.g. ¸\u000d for CR)
  - HEX
- **occupiedMessageInput**
- **freeMessageTimeout**: timeout in seconds.
- **freeMessageType**:
  - string (it may contain unicode characters e.g. ¸\u000d for CR)
  - HEX
- **freeMessageInput**: If type is HEX, these formats are accepted: A0B1... or A0 B1... or 0xA0 0xB1... or \xA0 \xB1...
- **showSerialPortConfig**: if enabled, the serial port connection status icon is displayed in the dashboard.

They can be referred in the JavaScript code as e.g. `params.ipAddress`

Defined Events

- **connected**
- **disconnected**
- **error**
- **ocsStatusChanged**

Defined Methods

No Methods are defined in this Module.

Defined Rules

No Rules are defined in this Module.

Custom Code

The code of the factory module in JavaScript.

> **ATTENTION!** If the GUI tools cannot solve a problem, custom code can be added to the end of the code, but please note that if you upgrade the driver module (blue button) the custom sections will be deleted.

4.7. Cisco Webex Module

Supported Cisco Devices

The following Cisco devices are supported by this Module:

- Cisco Webex Room Kit
- Cisco Webex Room Kit Mini
- Cisco Webex Room Kit Plus
- Cisco Webex Room Kit Pro
- Cisco Webex Codec Plus
- Cisco Webex Codec Pro
- Cisco Webex Desk Pro
- Cisco Webex Room 70 G2
- Cisco Telepresence SX20
- Cisco Telepresence SX80
- Cisco DX70
- Cisco DX80
- Cisco MX700
- Cisco MX800
- Room Bar
- Room Kit EQ
- Board Pro

The Purpose of This Module

The following main benefits can be achieved by using Taurus/MMX2 devices and LARA with this Module:

Bring Your Own Device (BYOD) mode: adding inputs to the Cisco Codec: thus more video sources (e.g. laptops) can be shared.

Camera share option: using the high quality camera and audio system of the Codec for other meeting purposes like a Zoom, Skype, etc. conference for a connected PC/laptop.

BYOD Mode

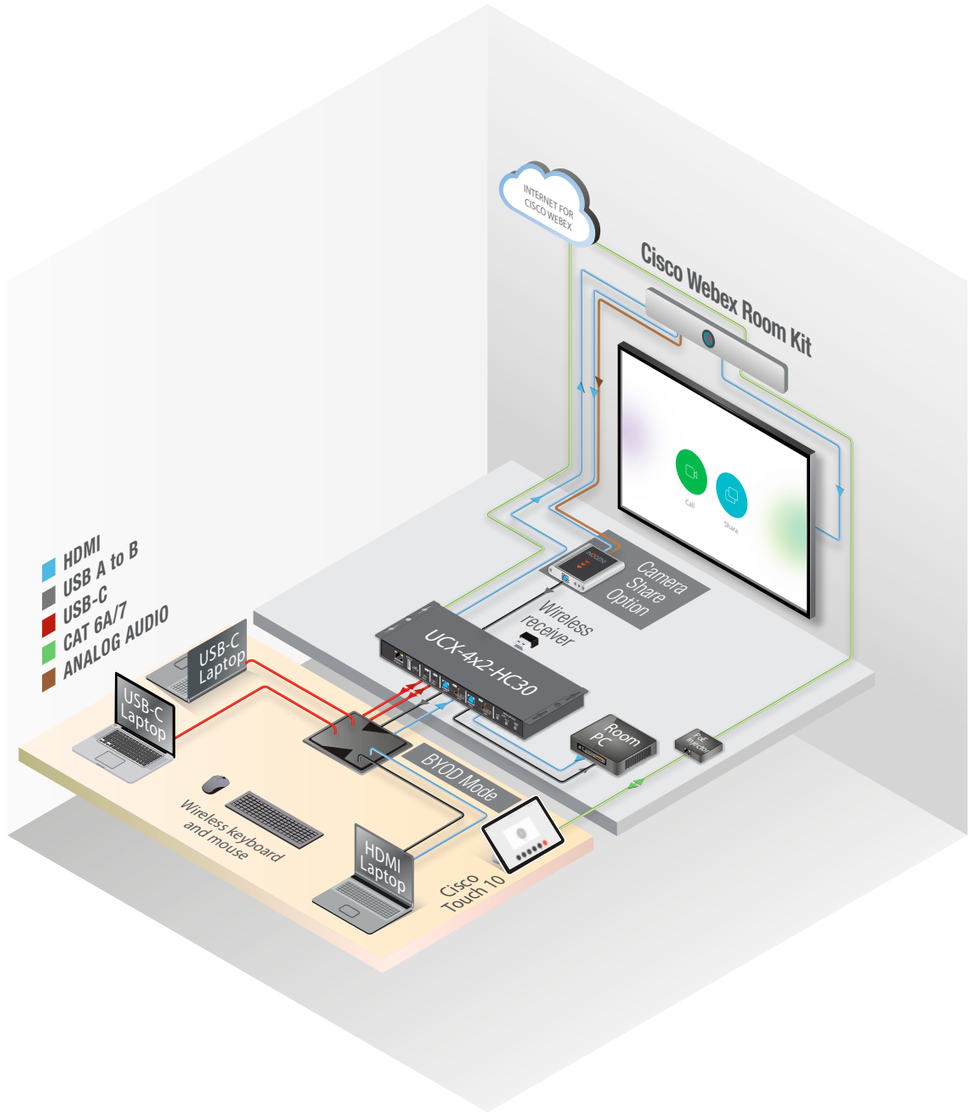
Cisco Codecs usually contain only one HDMI input. It can be increased by connecting a Taurus/MMX2 device to that input port, thus up to four devices can be connected. Furthermore, certain Taurus models contain USB-C ports, allowing another connection type. When video signal is detected on a video input port of the Taurus device, it will be visible on the UI of the Codec. After that the source can be selected as a BYOD source. Each input can be labelled with a unique name to display in the UI of the Codec.

Camera Share Option

It may happen that a video conference is taking place not via the Webex service, but with one of the particpant's laptop. The camera and the audio system of the Codec is much more suitable for this purpose than a laptop, so this feature means the camera and the audio devices can be shared to a PC/laptop connected over **HDMI and USB-B** or only **USB-C**.

4.7.1. Controlling Over Ethernet Connection

This chapter is about the settings when the Taurus and the Codec are connected via Ethernet.



Checklist for the Taurus/MMX2 Device

- Make sure that the Taurus and the Codec can **communicate with each other over Ethernet**.
- Check the **VLAN preset settings** of the Taurus (Control/Ethernet page in LDC): the Codec and the MCU must be in the same VLAN.
- Make sure the **autoselect is disabled** on the video output ports as well as on USB ports.
- Set the Video and USB-C ports to **umuted and unlocked** state.

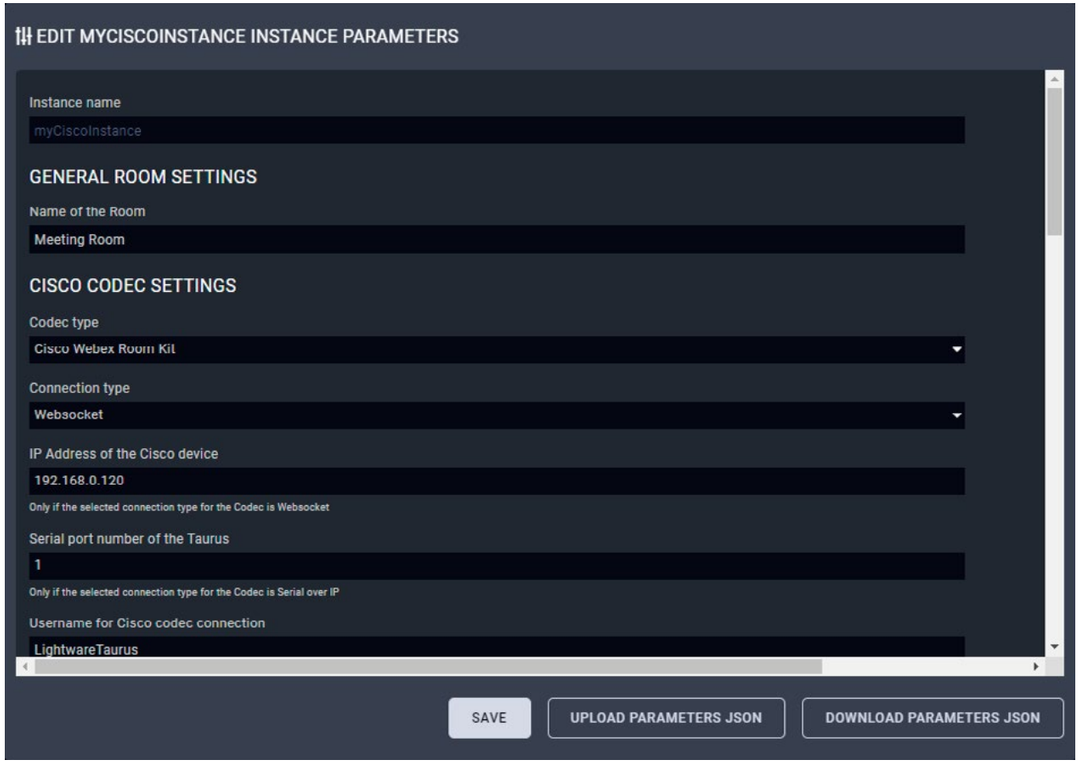
Checklist for the Codec

- **Websocket** has to be **enabled** (the integration works with wss (secured websocket) communication). Open the Codec **web interface:**
  - Settings → Configurations → NetworkServices → Websocket → **FollowHTTPService.**
  - Settings → Configurations → NetworkServices → HTTP → Mode → **HTTPS.**
  - **Save** the settings.
- Create a **user name** and set a **password** in the Codec. (Note them, these should be entered when configuring the Module in LARA.)
- Privileges should be **enabled** for:
  - **RoomControl**,
  - **Integrator**,
  - **Admin**.
- The following option shall be **disabled**:
  - **Require passphrase change on next user sign in.**
- If you install a **USB capture device** for camera share, e.g. Inogeni 4XKUSB3, connect it to the **last HDMI output** of the Codec.

Instance Parameters

Create an Instance from the Module '**ciscowebex**' and set the Instance Parameters as follows:

ATTENTION! Do not leave the value of the Parameter in 'empty' state if it can be selected from a drop-down menu.



Instance Name and Name of the Room

Set them as you wish.

Codec Type

Select the **Cisco model** you install.

Connection Type

Select **Websocket**.

IP Address of the Cisco Device

Type the IP address of the Codec.

Serial Port Number of the Taurus

N/A

Username for Cisco Codec Connection

As set in the Codec previously.

Password for Cisco Codec Connection

As set in the Codec previously.

Cisco Codec Output Port Connection Settings (#1/#2/#3)

DIFFERENCE: The available port number is Codec-dependent.

- If you have a **display device** connected to a certain output, set it to '**Display**'.
- If you connect a **USB capture device** to the last HDMI output for camera share option, set the '**Last HDMI Output as USB Capture**' setting to '**true**'. In this case the Output connection setting on that output does not matter.
- Set '**no connection**' for the output that is not used.

Taurus Settings

The default settings work, no need to change.

Taurus Input Port Settings

Set the labels that will be visible on the Cisco UI (BYOD selection). If you leave a field empty, that input will not be placed to the UI as a source. ASCII characters (space also) are accepted.

ATTENTION! For Cisco-compatibility reasons the HDCP will be disabled on the inputs that are defined in the BYOD selection.

Taurus Output Port Connections

Select '**Cisco Device Input #1**' for **Output#1**. The other output selectors are reserved for future developments.

Enable USB Parallel Switching

If set to true, the **USB** crosspoint will **follow the video** crosspoint.

Enable SignalPresent detection on inputs

If set to '**true**', the Taurus device will offer the **BYOD device selection** prompt when signal is present on the HDMI input.

4.7.2. Further Notices

Webex Mode

Webex mode is activated if:

- The Codec is started, or
- All devices are disconnected from the Codec.

INFO: **Half Wake** and **Standby** modes are active in Webex mode, but in BYOD mode these modes are inactive.

After Restart

If the Codec, the UCX/MMX2 or LARA is restarted for some reason, the system must recover in the same working state as it has been previously.

4.7.3. Known Issues

'Preset does not exist'

This message may appear in the log window when the Codec is a Cisco Room Kit Mini. This entry does not mean an error.

'Unknown widget: 'byodselector'

This message may appear in the log window when the Codec is a Cisco Room Kit Mini. This entry does not mean an error.

For camera sharing option

Please connect the USB capture device (or the RoomKit Mini USB-C cable) to the Taurus port USB-A #1:

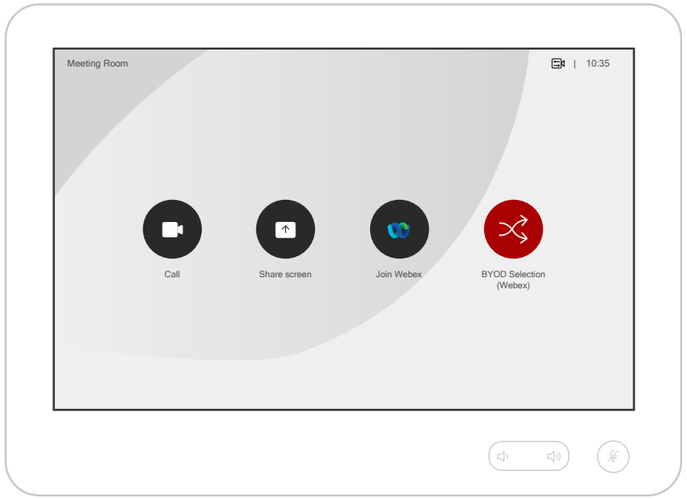
Unused Button

It may happen in case of Room Kit Mini and Room Kit Bar Codecs that the UI contains a button with the following label: 'Call from laptop'. This button has no function in this case.

4.7.4. Integration with the Cisco Codec

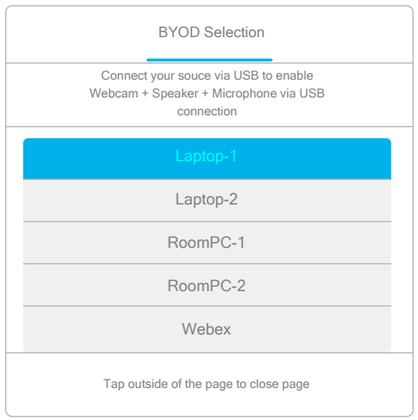
Main Menu

After a successful setup a new button will appear in the Main menu of the Codec: **BYOD Selection**.



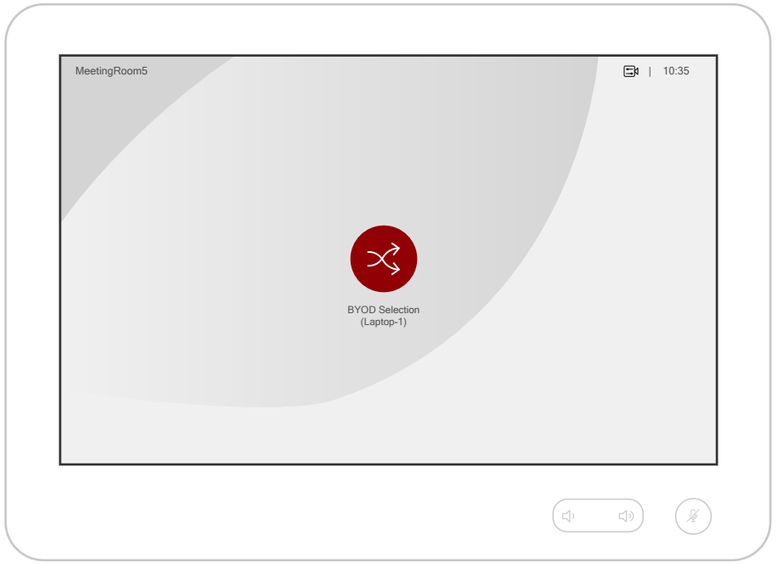
BYOD Selection

After pressing the '**BYOD Selection**' button, a window appears on the UI of the Codec. On the bottom of the screen the defined and available inputs can be seen with the labels set in the Instance *.



* If there is no USB layer in the Lightware device, the label below the 'BYOD selection' is: 'Connect your source via HDMI'.

When a source is selected:



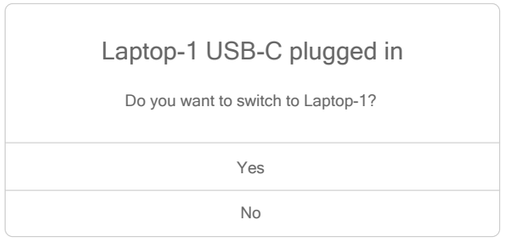
- The video signal of the selected source is **displayed on all outputs** of the Codec except the last HDMI output if a USB capture device is installed on that port.
- Not applicable buttons will be **hidden in the main menu**.
- The **USB crosspoint** of the Taurus UCX device **will follow** the video crosspoint state if the 'Enable USB parallel switching' option is 'true'.
- The '**SpeakerTrack**' feature of the camera is turned on – if it is supported.
- The **microphone signal** will be sent via the analog audio output of the Codec to the USB capture device. *
- The '**Do not disturb**' mode of the Codec will be switched on with default timeout (24 h).
- The automatic **standby mode** is deactivated in the Codec.

When selecting '**Webex**' again:

- The video output of the Codec will be the **default UI** content.
- Previously **hidden buttons will appear** again in the menu.
- The '**SpeakerTrack**' feature of the camera will be still on – if it is supported.
- The **analog audio output** of the Codec will be switched off. *
- The '**Do not disturb**' mode of the Codec will be switched off.
- The '**Halfwake**' mode must be activeted in the Codec after 2 minutes of idle time.

* Does not refer to Cisco Webex Room Kit Mini.

If a new source is connected to the UCX/MMX2 device, the following window appears:



5

User Module Descriptions

This chapter is about the Modules that have been developed by Lightware. The properties and the configuration steps are described in the coming sections.

5.1. Touchscreen UI Module

Introduction

The Module offers ready-to-use user interface templates for touchscreen devices, which provide controlling various type of meeting or huddle rooms.

Designs

- 2 different design sets
- Apply each design set for all above templates

Devices

The default templates contains the following devices and control tools:

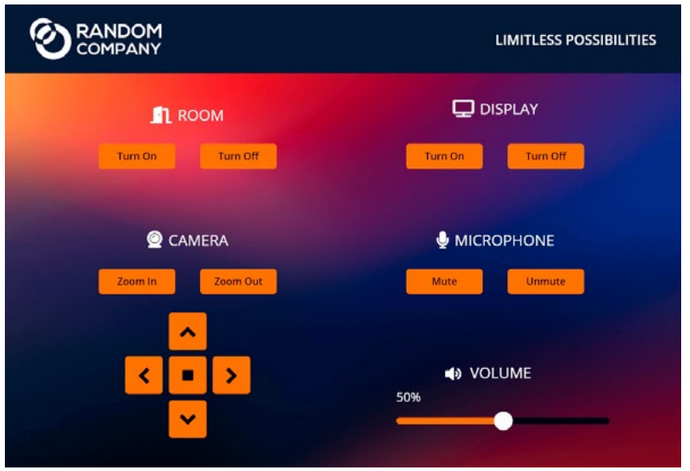
- **Room**: On / Off
- **Camera**: Move Up / Down / Right / Left / Zoom in / Zoom out
- **Microphone**: Mute / Unmute
- **Display**: Turn on / Turn off
- **Volume**: Increase / Decrease
- **Lights**: On / Off

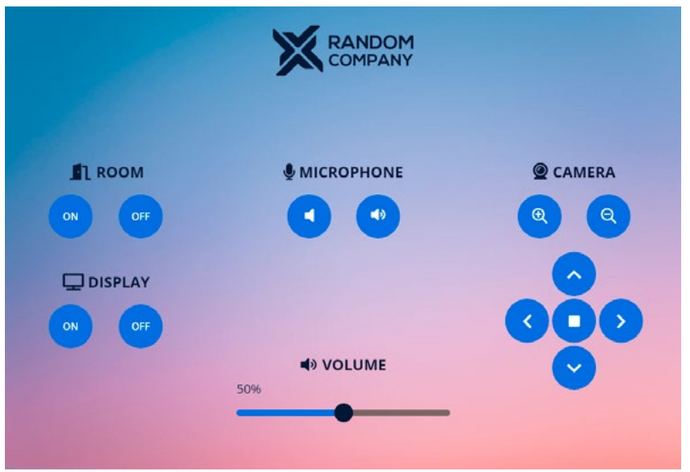
> TIPS AND TRICKS: The built-in templates can be customized freely and users can add new buttons and features to the touchpanel as well. See more personalization samples in the Personalizing of the UI section.

5.1.1. Huddle Room Template

Devices in the template:

- Room
- Camera
- Microphone
- Display
- Volume slider

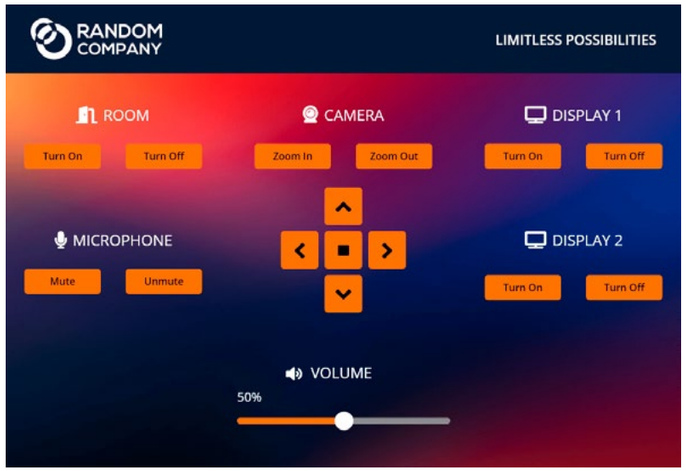


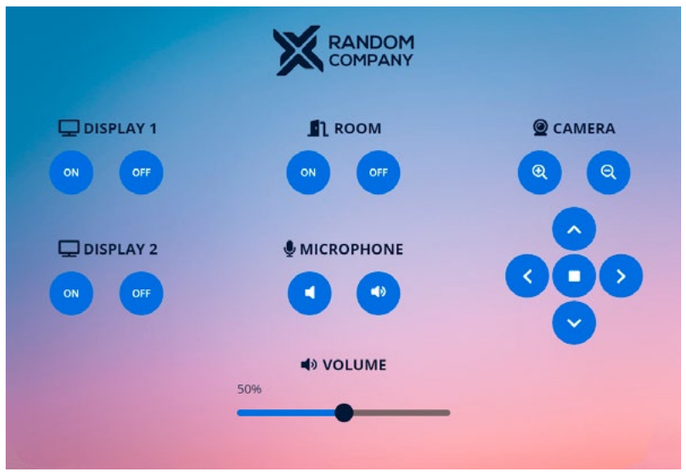


5.1.2. Meeting Room Template

Devices in the template:

- Room
- Camera
- Microphone
- Two displays
- Volume slider

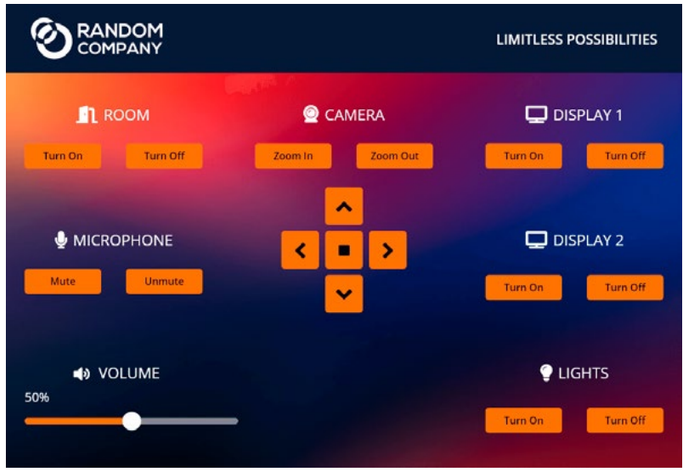


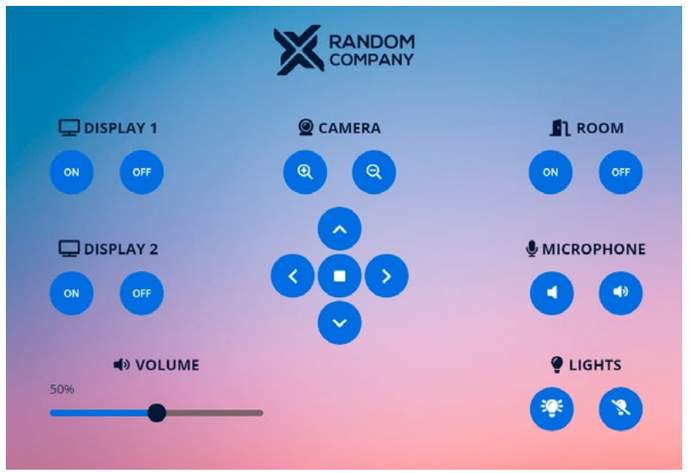


5.1.3. Meeting Room with Light Control Template

Devices in the template:

- Room
- Camera
- Microphone
- Two displays
- Volume slider
- Light

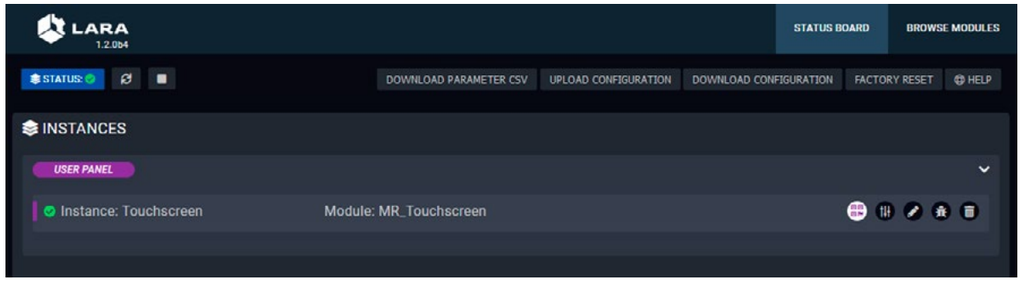




5.1.4. Dashboard Content

The following status indicator is displayed on the Status board:

- QR code and link to the Touchscreen UI

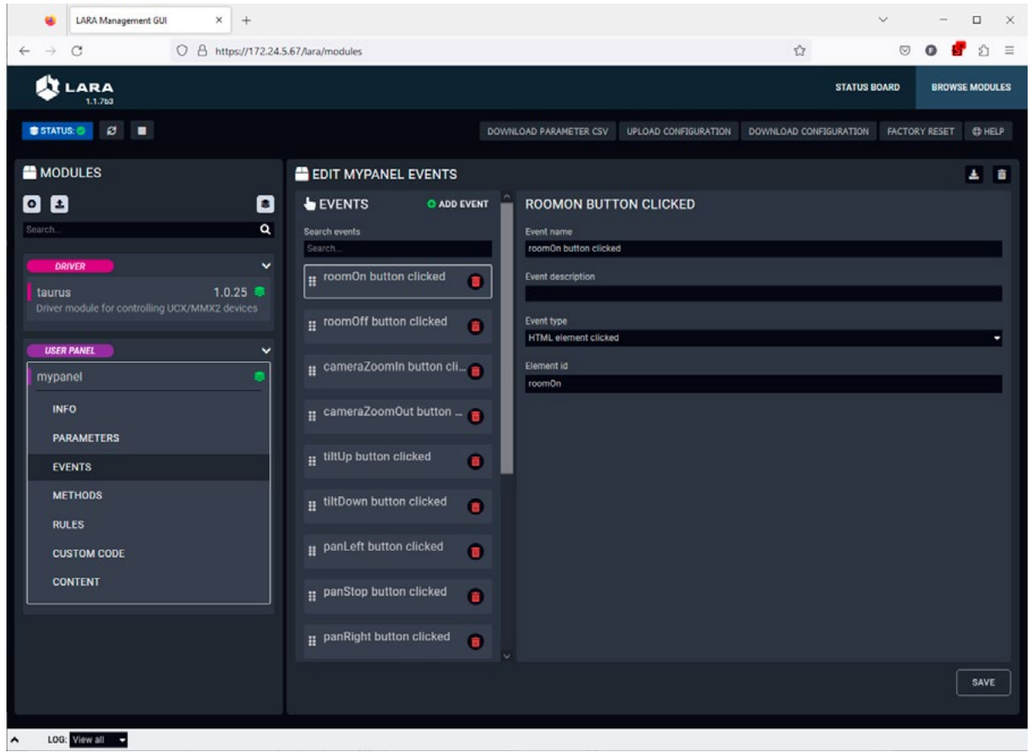


Clicking on the icon pops up the scannable QR code and the link of the touchscreen UI.

5.1.5. Editing of the Module

Click on the icon to edit the Parameters of the Module and to get more information about it.

The following sections are available in the user panel Module:



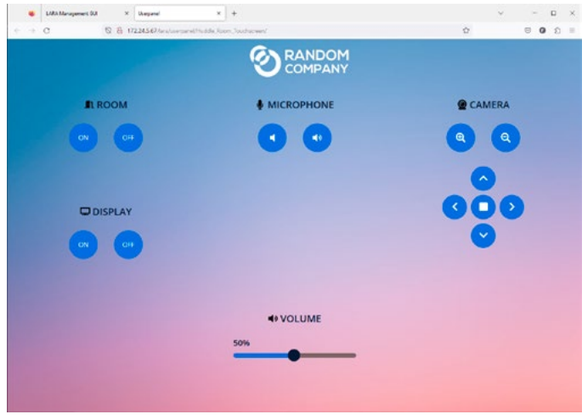
- **Info**: it can be filled with basic user information about the user panel like author, version, etc.
- **Parameters**: Parameters can be added to the Module which can be used for the user panel. User defined Parameters can be used in Javascript code only.
- **Events**: predefined Events can be cusomized or new Events can be added to the user panel interActions and a specific change (user interAction or Parameter change) that can be used as a Trigger.
- **Methods**: Methods can be added to Module and Actions to run when an Event is triggered.
- **Rules**: predefined Rules can be cusomized or new Rules can be added to the user panel interActions. In case of User Modules, the Rule can be arranged in the Module, Logic Module is not necessary.
- **Custom code**: custom user code in javascript can be uploaded.
- **Content**: the original HTML and CSS code of the chosen template can be reviewed and customized here.

5.1.6. Personalizing of the UI

This section shows how to personalize the touchscreen UI templates through some simple examples. The **Huddle Room Light template** will be used in the following samples.

Changing of the Company Logo

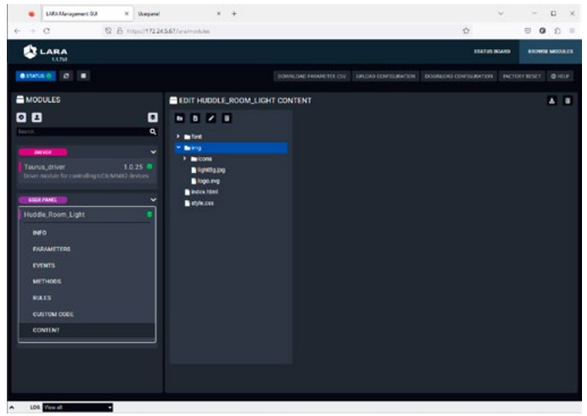
The template contains a logo which is displayed on the touchscreen UI by default. The logo can be changed easily in few simple steps.



Huddle Room Light template with the default company logo

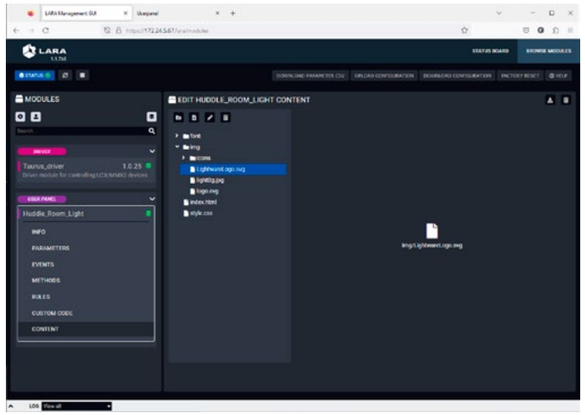
Uploading the New Logo

Step 1. Edit the user panel Module by clicking on the icon. Go to the **Content** section and select the **img** folder.



TIPS AND TRICKS: The content or the files can be uploaded by drag&drop, too.

Step 2. Upload the new company logo to the **img** folder. Click on the **Upload file** icon. Browse the new logo file and upload it. Recommended file extension is **.svg**. In our example it is the LightwareLogo.svg.

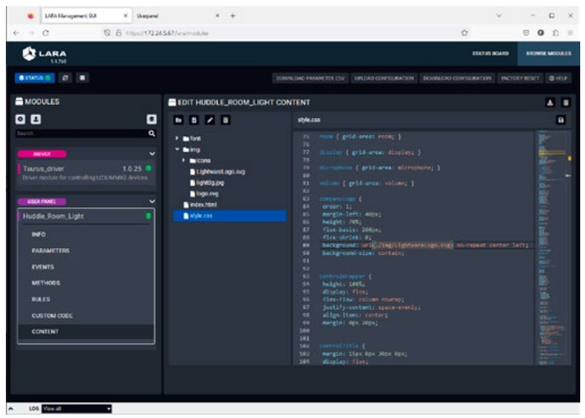


Editing of the CSS File

Step 3. Select the **style.css** and find the row in the code where the logo.svg is actually linked. In our example it is in the row 89. Edit the link of the logo file to the new one like in our example:

Original code: `background: url(./img/logo.svg) no-repeat center left;`

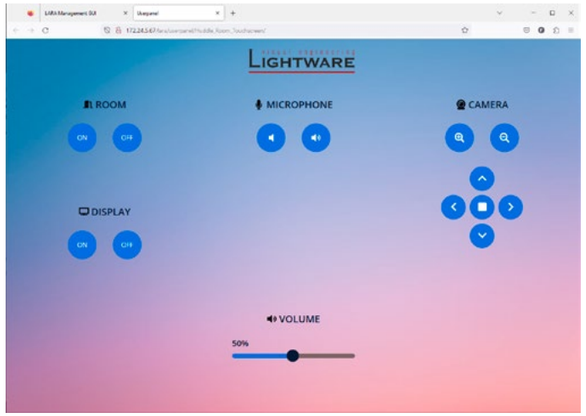
Edited code: `background: url(./img/LightwareLogo.svg) no-repeat center left;`



Step 4. Save the style.css by clicking on the **Save file** icon.

Refreshing of the Touchscreen UI

Step 5. Go the touchscreen panel UI and refresh it in the web browser. The new company logo appears immediately.



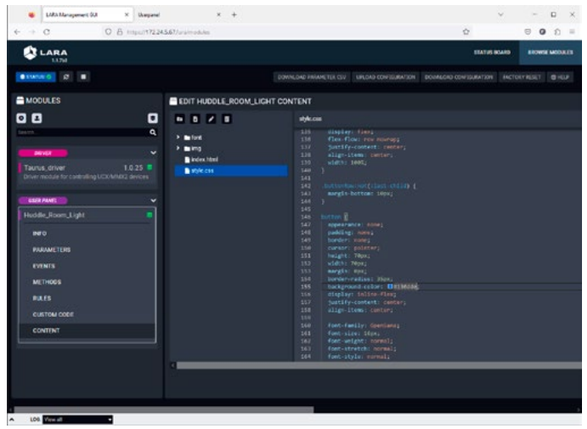
Touchscreen UI with the new company logo

Changing the Color of the Buttons

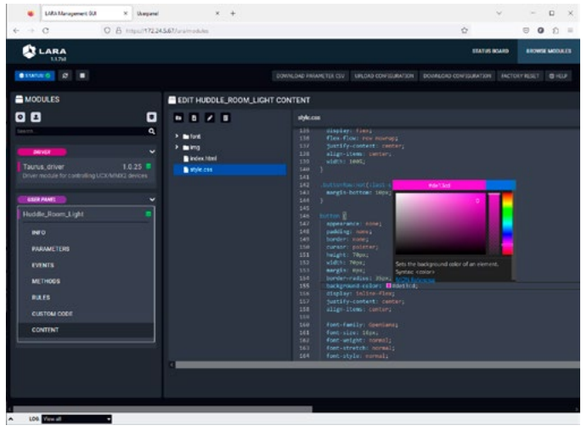
The next few steps show how to change the color of the buttons.

Editing of the CSS File

Step 1. Select the **style.css** and find the row in the code where the background color of the buttons are defined. In our example it is in the row 155.

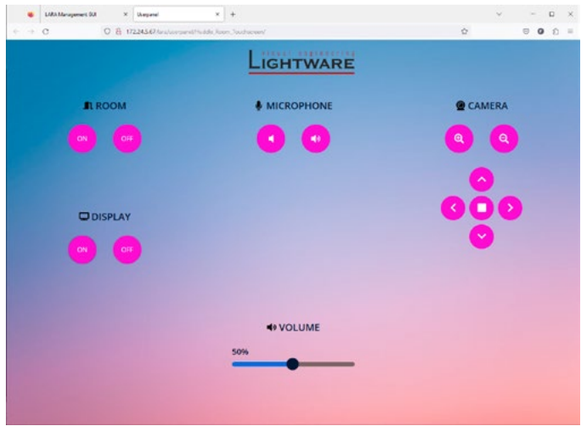


Step 2. Type the wished RGB code after the # character or select it in the color picker pop-up window.



Step 3. Save the style.css by clicking on the **Save file** icon.

Step 4. Go the touchscreen panel UI and refresh it in the web browser. The new button color appears immediately.



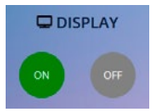
Adding Feedback Indicators to the Buttons

A feedback from the UI that makes it visible that the interaction actually happened is a valid user requirement. The following section shows how to do it in the touchscreen panel Module.

Example Description

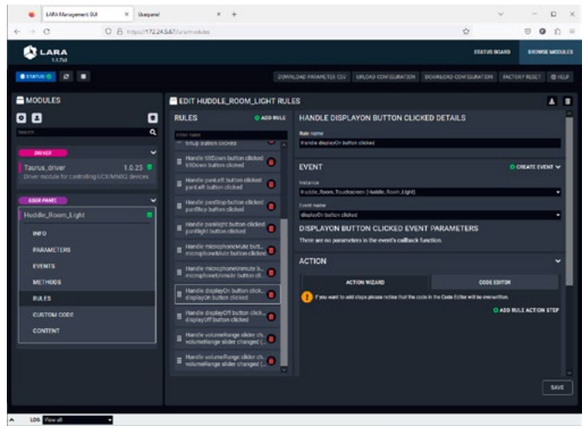
The Display On button shall change its color to green when it is clicked and parallel, the Display Off button shall change its color to grey.

When Display Off button is clicked, its color shall change to green and parallel, the Display On button shall change its color to grey.



Adding Rule Action Steps

Step 1. Edit the user panel Module by clicking on the ✎ icon. Navigate to the **Rules** section and select the **Handle displayOn button clicked** Rule.



Step 2. Click on the ⊕ **Add Rule Action step** button. Fill the fields with the following values. Click on the **Save** button when it is done.

Name	DisplayOn color green
Description	Color of the Display On button changes to green when it is clicked.
Select Action step	Set HTML Element CSS Style
HTML Element ID	displayOn
CSS Property name	background-color
CSS Property value	green

Step 3. Click on the ⊕ **Add Rule Action step** button again. Fill the fields with the following values. Click on the **Save** button when it is done.

Name	DisplayOff color grey
Description	Color of the Display Off button changes to grey when Display On is active.
Select Action step	Set HTML Element CSS Style
HTML Element ID	displayOff
CSS Property name	background-color
CSS Property value	grey

Step 4. Navigate to the **Rules** section and select the **Handle displayOff button clicked** Rule.

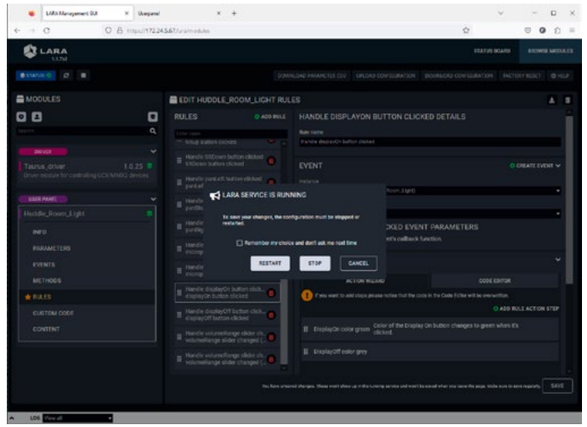
Step 5. Click on the ⊕ **Add Rule Action step** button. Fill the fields with the following values. Click on the **Save** button when it is done.

Name	DisplayOff color green
Description	Color of the Display Off button changes to green when it is clicked.
Select Action step	Set HTML Element CSS Style
HTML Element ID	displayOff
CSS Property name	background-color
CSS Property value	green

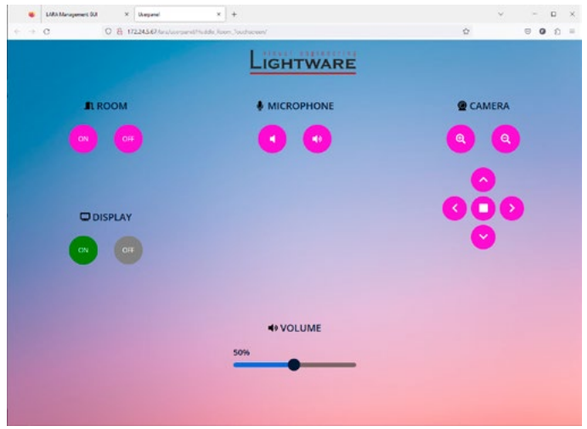
Step 6. Click on the ⊕ **Add Rule Action step** button again. Fill the fields with the following values. Click on the **Save** button when it is done.

Name	DisplayOn color grey
Description	Color of the Display On button changes to grey when Display Off is active.
Select Action step	Set HTML Element CSS Style
HTML Element ID	displayOn
CSS Property name	background-color
CSS Property value	grey

Step 7. Click on the **Save** button to apply the recent changes. Accept to **Restart** LARA service.



Step 8. Go the touchscreen panel UI. After restarting the LARA service the UI will be refreshed automatically and the new button style appears.



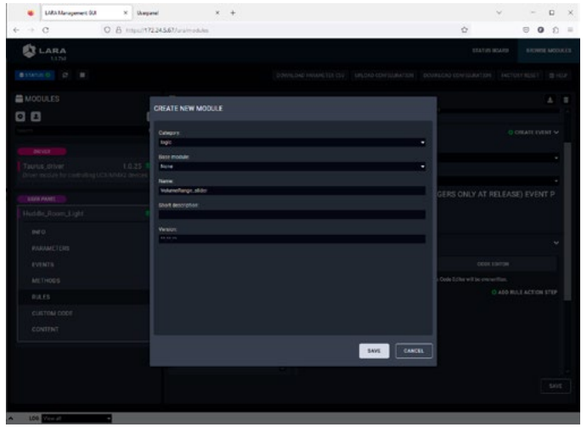
TIPS AND TRICKS: The example above is one possibility to customize the image of the touchscreen panel. Every element that has CSS property can be re-colored / resized / added new element (for example border around the buttons) etc. Only your imagination can set the limit.

How to Use the Value of the VolumeRange Slider Changed Event

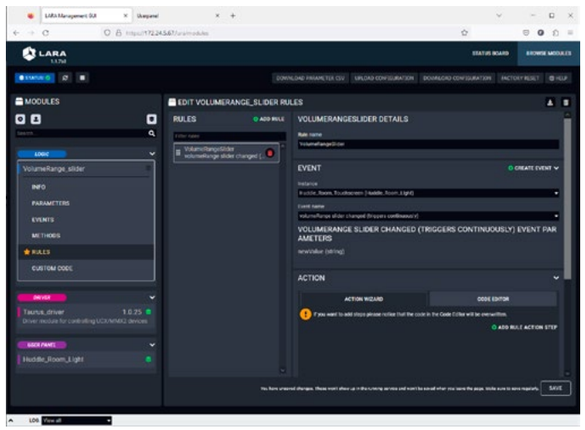
Disclaimer: due to a bug, the slider of the HTML templates do not work as it should. The following workaround helps to have a correct configuration – until the bug is corrected. The phenomenon is that the slider does not show the correct setting when the slider is moved. The root cause is in the Rule editor's Action wizard: the values written into the Parameter input fields (for example a Method's Parameter input field) are handled as constant values, thus an Event Parameter cannot be passed to an invoked Method via wizard. This section is about a workaround to solve that issue.

Creating of a Logic Module

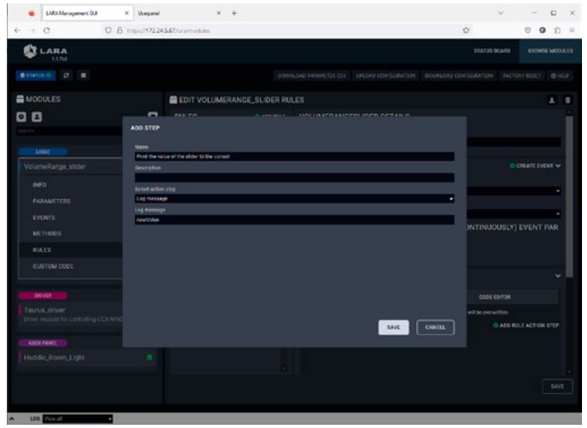
Step 1. Click on the ⊕ button under the Modules section to create a new logic Module. Select the **logic** category and type a custom name. Finally click on the **Save** button.



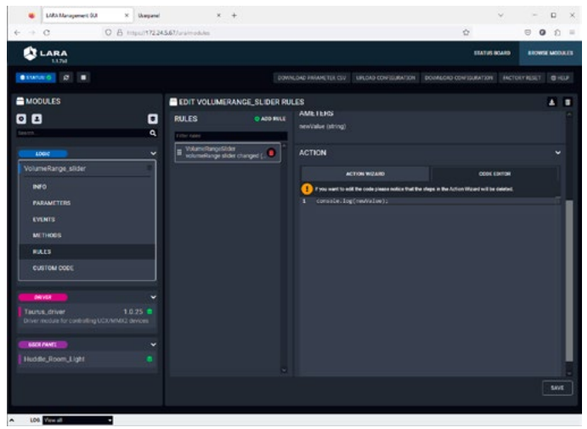
Step 2. Create a new Rule in the logic Module by clicking on the ⊕ **Add Rule** button. Select the created userpanel Instance and the **volumeRange slider changed (triggers continuously)** Event in the Event section. Finally click on the **Save** button.



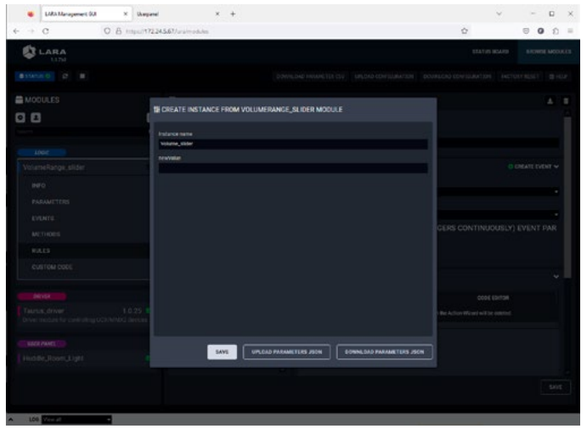
Step 3. Add an Action step to the Rule by clicking on the **Add Rule Action step** button. Type a custom name and choose the **Log message** for example. In the "Log message" input, type in the Parameter name of the previously selected Event, which is **newValue**, and click on **Save** button.



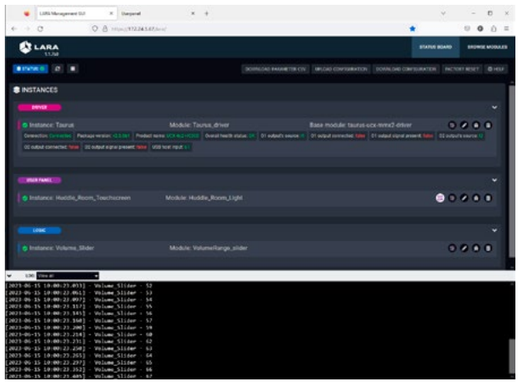
Step 4. Switch to **Code Editor** in the Actions section. **Remove the apostrophes** from the console.log function call. Finally click on the **Save** button.

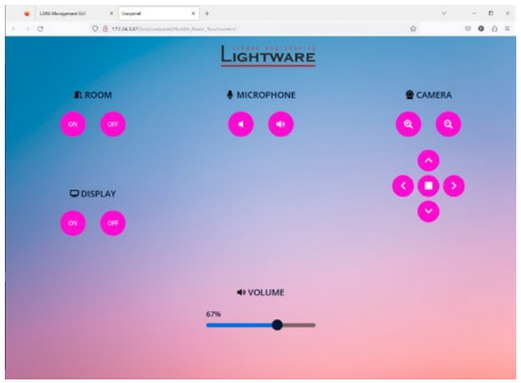


Step 5. Create an Instance by clicking on the **Create new Instance for selected Module** icon. Type a custom name and click on **Save** button.



Step 6. It's working!





Document Revision History

Rev.	Release date	Changes	Editor
v1.0	22-03-2023	Initial version	Laszlo Zsedenyi
v1.1	03-10-2023	User Module chapter (Touchscreen) added; minor corrections and updates.	Tamas Forgacs Laszlo Zsedenyi
v1.2	06-05-2024	Chapter 1-3 have been revised; new Rule-handling and Status variable sections added.	Laszlo Zsedenyi

Contact Us

sales@lightware.com

+36 1 255 3800

support@lightware.com

+36 1 255 3810

Lightware Visual Engineering PLC.

Peterdy 15, Budapest H-1071, Hungary

www.lightware.com