

VirtualHub

User's guide

---



# Table of contents

<b>1. Introduction</b> .....	<b>1</b>
<b>2. Installation</b> .....	<b>3</b>
2.1. <i>Installing from a .zip file</i> .....	3
2.2. <i>Installing with msi under Windows</i> .....	4
2.3. <i>Installing with apt_get under Linux</i> .....	4
2.4. <i>Linux and USB</i> .....	5
2.5. <i>USB access limitation</i> .....	6
2.6. <i>Command line parameters</i> .....	6
<b>3. Configuring and testing the modules</b> .....	<b>9</b>
3.1. <i>Locating the modules</i> .....	10
3.2. <i>Testing the modules</i> .....	10
3.3. <i>Configuring modules</i> .....	11
3.4. <i>Upgrading firmware</i> .....	11
3.5. <i>Accessing the sensor data logger</i> .....	13
<b>4. Using VirtualHub as a gateway</b> .....	<b>15</b>
4.1. <i>A gateway to work around USB access limitation</i> .....	15
4.2. <i>REST gateway</i> .....	16
4.3. <i>OpenMetrics gateway (Prometheus)</i> .....	16
<b>5. Access control</b> .....	<b>19</b>
5.1. <i>Admin access</i> .....	20
5.2. <i>User access</i> .....	20
5.3. <i>Access control and API</i> .....	20
<b>6. Outgoing connections</b> .....	<b>21</b>
6.1. <i>Configuration</i> .....	21
6.2. <i>HTTP Callbacks to 3rd-party services</i> .....	22
6.3. <i>Callbacks to an MQTT broker</i> .....	23
6.4. <i>Yocto-API callbacks</i> .....	25
6.5. <i>User defined HTTP callbacks</i> .....	26
6.6. <i>Names associated with posted values</i> .....	27

6.7. Scheduling callbacks .....	29
6.8. Tests .....	30
6.9. Spontaneous connections .....	30
<b>7. Optional components .....</b>	<b>31</b>
7.1. The Windows VirtualHub Control tool .....	31
7.2. Installing Yocto-Visualization (for web) .....	32
<b>8. VirtualHub 2.0 .....</b>	<b>33</b>
8.1. Incoming connections .....	33
8.2. Outgoing connections .....	33
8.3. IPv6 .....	33
Blueprint .....	35
8.4. Supported operating systems .....	35

# 1. Introduction

VirtualHub is a software destined mainly to manage USB devices conceived by Yoctopuce. It is a kind of toolbox which aims at

- providing access to USB devices from languages, such as Javascript and PHP, which do not allow you to control hardware layers of a computer.
- providing access to the USB devices through a network connection, and this from all the available languages.
- configuring and testing Yoctopuce USB devices.
- providing the required connectivity for interaction between Yoctopuce devices and cloud-based services.

VirtualHub is **not** mandatory for driving Yoctopuce USB devices with programming languages allowing to access to hardware layers, such as C++, C#, Python, Java, Android, Delphi, Visual Basic, LabView, and the command line API. With these languages, Yoctopuce USB devices can be driven **directly**, you do not even need a driver.

VirtualHub is available for Windows, macOS X, and Linux (both Intel and ARM) operating systems. It works in the same way on all three systems.



## 2. Installation

VirtualHub does not require a true installation. You can download it either as a .zip version for Windows, Linux and macOS, or as an .msi version for Windows, or even use `apt_get` under Linux.

### 2.1. Installing from a .zip file

The .zip file contains some .txt files and a simple executable file which can be located either at the same level as the .txt files, or in a directory corresponding to its architecture. Copy the executable file corresponding to your architecture wherever you want and run it from a command line. No driver is required.

VirtualHub needs to save some configuration parameters. They are saved in the `.virtualhub.dat` file, which is located in the user's `AppData\Roaming` directory under Windows and in the user's `homedir` under Linux and macOS. You can modify this behavior with the `-c` option on the command line.

**Under Windows**, if you do not wish to explicitly run VirtualHub each time you need it, you can install it as a service. To do so, open a command window with administrator privileges, run VirtualHub once with the `-i` option. It is then installed as a service and starts automatically. If you later change your mind, run it again in command line with the `-u` option to remove the service. Note that when VirtualHub runs as a service, it is launched by the `SYSTEM` special user and the directory used to save the configuration is `\windows\system32\config\systemprofile\AppData\Roaming`.

**Under Linux**, to allow users without privileges to communicate with the Yoctopuce devices, you must perform some modifications to the udev rules. You can do this by running the following command line:

```
sudo VirtualHub --install_udev_rule
```

Then you must restart the computer for the new rule to be taken into account (for more details about the changes performed on the udev rules, see section 2.4 below).

In addition, if you do not wish to explicitly run VirtualHub each time you need it, you can install it as a service. To do so, run the following command:

```
sudo VirtualHub -i
```

It is then installed as a system service and starts automatically on startup. If you later change your mind, run it again in command line with the `-u` option to remove the service.

## 2.2. Installing with msi under Windows

If you prefer to install VirtualHub using the .msi file, download it and double-click it to start the installer.

The VirtualHub installer is very simple. It asks you to accept the conditions of use of VirtualHub and which directory to use. By default, the installer uses the `C:\Program Files (x86)\Yoctopuce\VirtualHub\` directory. It performs the following tasks:

1. Extracts VirtualHub in the destination directory
2. Extracts the VirtualHub Control tool in the destination directory
3. Creates a Yoctopuce section in the Start menu
4. Creates a "VirtualHub UI" and a "VirtualHub Control" in this section
5. Adds VirtualHub to the PATH of the machine
6. Adds VirtualHub to the list of programs installed on the machine

At the end, you can run VirtualHub directly from the Start menu.

Thanks to the installer, VirtualHub is recognized as a full-fledged Windows program. In particular, you can check directly from the Windows control panel which version is installed, manage its installation through centralized tools, and uninstall it with one click from the control panel.

As with most installers, in most cases you can simply run the installer, click "Next" on each panel, and finally on "Install". The installer can discover if a previous version of VirtualHub is already installed and simply update the executable files with one click from the configuration panel.

The installer also adds a Windows specific tool, **VirtualHub Control**, for which you can find information in the section "The Windows VirtualHub Control tool" at the end of this guide.

## 2.3. Installing with apt\_get under Linux

The Yoctopuce APT repository enables you to install VirtualHub and other tools on any Linux distribution using `apt-get`. The most well-known distributions are Debian, Ubuntu, and Raspbian, but all the distributions based on those should work as well.

Before you can use `apt-get`, you must have added the Yoctopuce APT repository to the system. If it is not already the case, you must add it before installing VirtualHub.

### Add the Yoctopuce APT repository

1. Install the repository GPG key in the `/usr/share/keyrings/` directory.

```
wget -q -O - https://www.yoctopuce.com/apt/KEY.gpg | gpg --dearmor | sudo tee -a /usr/share/keyrings/yoctopuce.gpg > /dev/null
```

2. Add the Yoctopuce repository to the list of servers used by `apt-get`

```
echo 'deb [signed-by=/usr/share/keyrings/yoctopuce.gpg] https://www.yoctopuce.com/ apt/stable/' | sudo tee -a /etc/apt/sources.list.d/yoctopuce.list > /dev/null
```

### Installing VirtualHub

You must start by updating the list of available packages before running the installation per say.

```
sudo apt-get update
sudo apt-get install virtualhub
```

Finally, to allow users without privileges to communicate with the Yoctopuce devices, you must perform some modifications to the `udev` rules. You can do this by running the following command line:

```
sudo VirtualHub --install_udev_rule
```

Then you must restart the computer for the new rule to be taken into account.

In addition, if you do not wish to explicitly run VirtualHub each time you need it, you can install it as a service. To do so, run the following command:

```
sudo VirtualHub -i
```

It is then installed as a system service and starts automatically on startup. If you later change your mind, run it again in command line with the `-u` option to remove the service.

If you want to understand in details the changes performed on the `udev` rules, or perform them manually yourself, you can read the following paragraph. Otherwise, you can skip directly to the next section.

## 2.4. Linux and USB

To work correctly under Linux, VirtualHub needs to have write access to all the Yoctopuce USB peripherals. However, by default under Linux, USB privileges of the non-root users are limited to read access. To avoid having to run VirtualHub as root, you need to create a new `udev` rule to authorize one or several users to have write access to the Yoctopuce peripherals.

To add a new `udev` rule to your installation, you must add a file with a name following the "`##-arbitraryName.rules`" format, in the `/etc/udev/rules.d` directory. When the system is starting, `udev` reads all the files with a `.rules` extension in this directory, respecting the alphabetical order (for example, the `51-custom.rules` file is interpreted AFTER the `50-udev-default.rules` file).

The `50-udev-default` file contains the system default `udev` rules. To modify the default behavior, you therefore need to create a file with a name that starts with a number larger than 50, that will override the system default rules. Note that to add a rule, you need a root access on the system.

In the `udev_conf` directory of the VirtualHub for Linux<sup>1</sup> archive, there are two rule examples which you can use as a basis.

### Example 1: 51-yoctopuce.rules

This rule provides all the users with read and write access to the Yoctopuce USB devices. Access rights for all other devices are not modified. If this scenario suits you, you only need to copy the `51-yoctopuce_all.rules` file into the `/etc/udev/rules.d` directory and to restart your system.

```
# udev rules to allow write access to all users
# for Yoctopuce USB devices
SUBSYSTEM=="usb", ATTR{idVendor}=="24e0", MODE="0666"
```

### Example 2: 51-yoctopuce\_group.rules

This rule authorizes the `yoctogroup` group to have read and write access to Yoctopuce USB peripherals. Access rights for all other peripherals are not modified. If this scenario suits you, you only need to copy the `51-yoctopuce_group.rules` file into the `/etc/udev/rules.d` directory and restart your system.

```
# udev rules to allow write access to all users of "yoctogroup"
# for Yoctopuce USB devices
```

<sup>1</sup> <http://www.yoctopuce.com/FR/virtualhub.php>

```
SUBSYSTEM=="usb", ATTR{idVendor}=="24e0", MODE="0664", GROUP="yoctogroup"
```

## 2.5. USB access limitation

Yoctopuce USB devices have a limitation: on a given machine, you can have only one application at any given time that natively controls them. And it so happens that VirtualHub counts as a native application. Therefore, if you try to run an application which natively controls Yoctopuce USB devices, make sure that VirtualHub is not running, neither in a command line, nor as a service.

Note that from a programming standpoint, you can easily work around this limitation by making sure that your application uses VirtualHub as a gateway to control the devices, rather than controlling them directly. To do so, you only need to change one parameter when calling `yRegisterHub`: instead of passing "usb" as parameter, you should use the IP address of the computer on which VirtualHub is running. Using VirtualHub as a gateway is described in the chapter "Using VirtualHub as a gateway".

```
YAPI.RegisterHub('localhost:4444',errmsg);
```

## 2.6. Command line parameters

VirtualHub accepts several parameters in the command line.

Option	Short description
-c	Indicates the path where the configuration file is saved
-d	Runs in the background
-F	Activates the Files mode
-g	Redirects the logs into a file
-h	Displays a brief help in English
-i	Installs the service
--install_udev_rule	Provides access to USB devices for all users (Linux)
-n	Uses the network interface
-o	Activates the OSControl interface
-p	Changes the port
-T	Uses a template to create the configuration file
-u	Uninstalls the service
-v	Indicates the version

### -c : Configuration file

Under Windows, when you start VirtualHub manually in a command window, it uses the configuration file in the following location:

```
C:\Users\username\AppData\Roaming\.virtualhub.dat
```

However, when VirtualHub is started by Windows as a service, it uses another location:

```
C:\Windows\System32\config\systemprofile\AppData\Roaming\.virtualhub.dat
```

Under Linux and macOS, the `.virtualhub.dat` configuration file is located directly in the user's home directory.

The `-c` option allows you to change this location. For example:

```
>virtualhub -c C:\tmp\mysetting.bin
```

Moreover, to find it more easily, the location of the configuration file is indicated in the logs.

### **-d : Starting in service/daemon**

Starts VirtualHub in the background. Only used when VirtualHub is running as a service.

### **-F : Activating the Files option**

Activates the Files interface for adding custom files to VirtualHub. The container is a .tar file and is created if it does not exist.

```
>virtualhub -F container.tar
```

### **-g : Saving debug information in a file**

Saves the debug information in a file.

Command line example:

```
>virtualhub -g tracefile.log
```

### **-h : Help**

Forces VirtualHub to display a brief help file.

### **-i : Installation as a service**

VirtualHub can run as a service, this option installs the service and starts it. Thus VirtualHub is available at all times, even if the machine reboots.

### **--install\_udev\_rule : Installing a UDEV rule**

By default, Linux does not give access to USB devices for standard users. To allow all users to communicate with Yoctopuce devices, you have to add a UDEV rule and reboot the machine. The installation of this rule can be performed with the option `--install_udev_rule`. **Do not forget to restart the machine** for the rule to be taken into account. Moreover, as this is a system modification, VirtualHub needs to be run with with supervisor rights using the `sudo` command.

Command line example :

```
>sudo VirtualHub --install_udev_rule
```

Once the machine is rebooted, VirtualHub can use the devices connected to the USB ports without `sudo`.

### **-n : Using the network interface with an IP address**

Uses the network interface with the IP address specified in the parameter.

### **-o : Activation of the osControl function**

Adds *osControl* functionality to VirtualHub, which enables you, among other things to remotely shut down the machine which is running VirtualHub using the Yoctopuce API.

### **-p : Port modification**

By default, VirtualHub uses TCP port 4444, this option allows you to use another one. For example:

```
>virtualhub -p 8889
```

### **-T : Creating a configuration file from a template**

If no configuration file is found, creates a configuration file based on the configuration template. This option allows you to use default settings other than factory settings (e.g. to impose a password at installation), but does not prevent the configuration from being changed later. The resulting configuration file is saved to the default location (see option `-c`).

Command line example :

```
>virtualhub -T config_template
```

### **-u : Uninstalling the service**

Uninstalls the service previously installed with the `-i` option

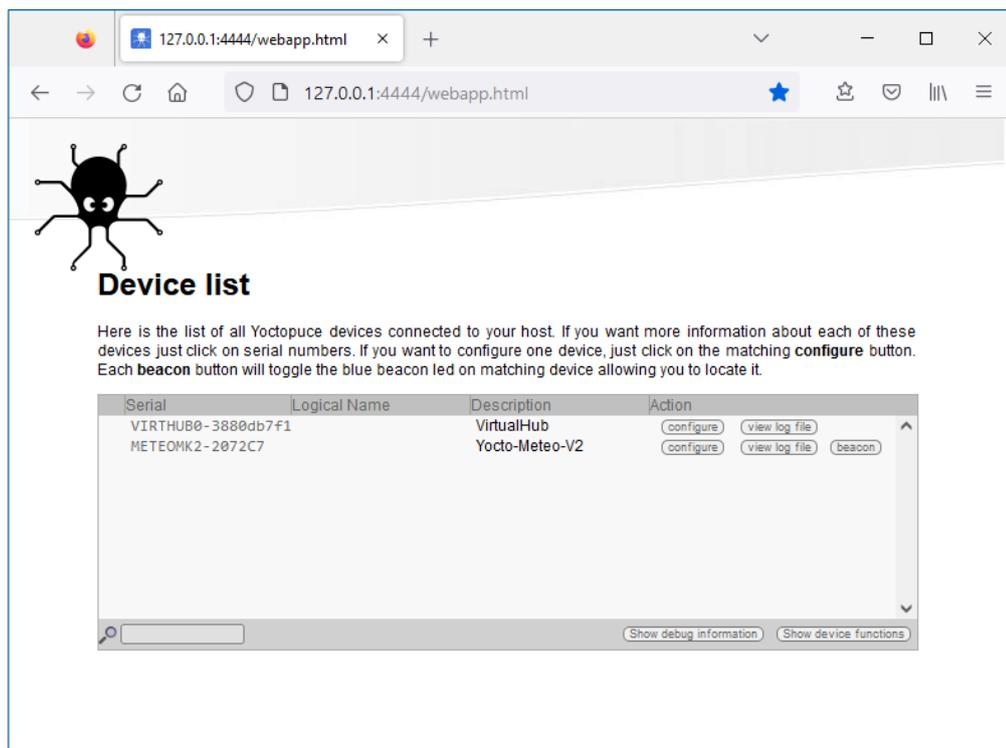
### **-v : Version**

Displays the VirtualHub version:

```
>virtualhub -v  
Version v1.0 (4237)
```

### 3. Configuring and testing the modules

Once installed and configured, VirtualHub allows you to test and configure your Yoctopuce modules. To do so, open your preferred web browser<sup>1</sup>. Connect in *HTTP* to port 4444 of the machine on which VirtualHub is running. If it is the local machine, use the `http://127.0.0.1:4444` address. The list of all the modules connected on the machine should appear.



*VirtualHub web interface*

At the bottom of the page, there are two buttons. The first button, **Show debug information**, enables you to display and then save all the information required to debug an issue linked to the VirtualHub, that is:

- The list of all detected modules

<sup>1</sup> The VirtualHub interface is regularly tested on Firefox, Chrome, Opera, and Brave. It probably also works on Safari.

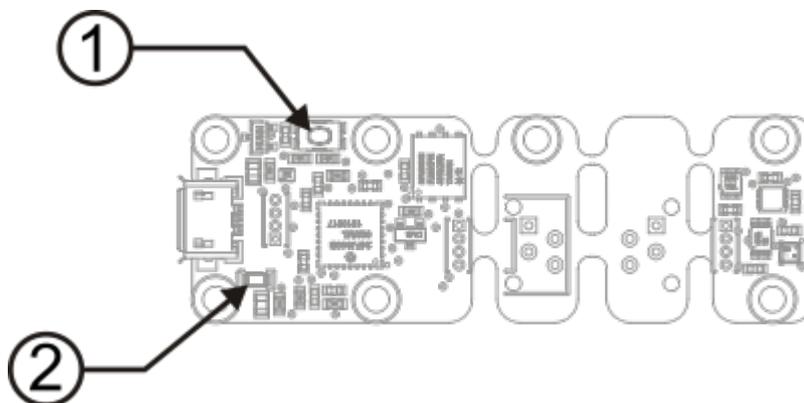
- The value of all the parameters of all the modules (without passwords).
- The logs of all the modules.
- The list of all the files uploaded on the modules, but not their content.
- The content of potential VirtualHub core dumps.

If you need to contact support, it is important to download this information and to send it with your request.

The second button, **Show device functions**, shows all the functions of the VirtualHub and of each of the modules connected to the VirtualHub.

### 3.1. Locating the modules

The main interface displays a line per connected module; if you have several modules of the same model, you can locate a specific module by clicking on the corresponding **beacon** button: it makes the blue led of the module start blinking and displays a blue disk at the beginning of the corresponding line in the interface. Pressing the Yocto-button of a connected module has the same effect.

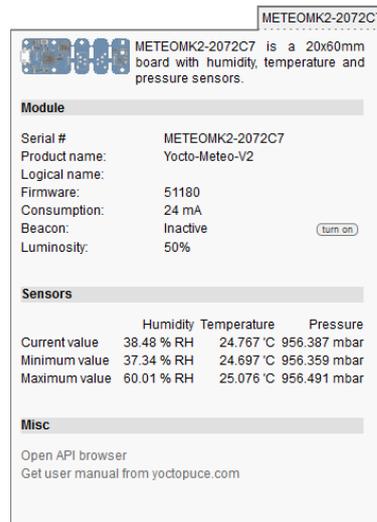


*Yocto-button (1) and localization led (2) of the Yocto-Meteo-V2 module. These two elements are always placed in the same location, whatever the module.*

### 3.2. Testing the modules

To test a module, simply click on the serial number of a module in the interface, a window specific to the module opens. This window generally allows you to activate the main functions of the module. Refer to the User's guide of the corresponding module for more details.

Usually, you do not need to have a version of the VirtualHub more recent than the module that you want to test/configure: most elements specific to the module interfaces are stored in the firmware of the modules themselves. There are however a few exceptions, so if you encounter an error in the web interface of a module, check if an update of VirtualHub is available and, if need be, install it. You may then need to reload the page in the browser with Shift-Reload, or to empty your browser cache, in order to force the update of the JavaScript code.



"Details" window of the Yocto-Meteo-V2 module

### 3.3. Configuring modules

You can configure a module by clicking on the corresponding **configure** button in the main interface. A window, specific to the module, then opens. This window allows you minimally to assign a logical name to the module and to update its firmware. Refer to the User's guide of the corresponding module for more details.



"Configuration" window of the Yocto-Meteo-V2 module

### 3.4. Upgrading firmware

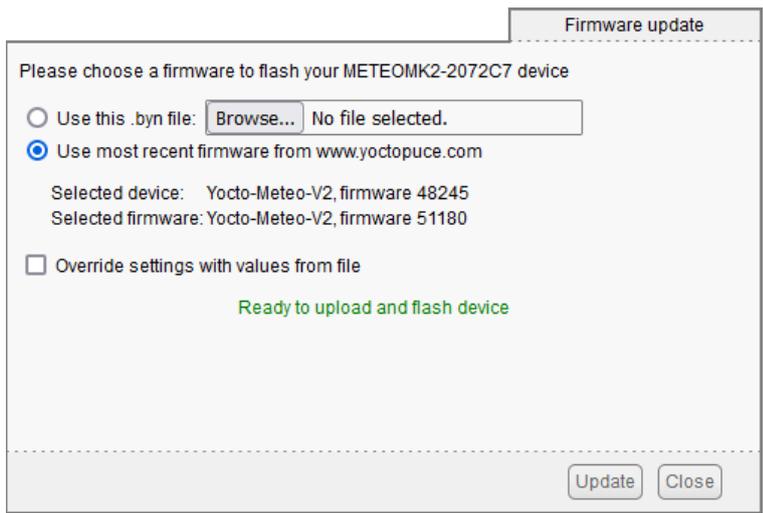
The Yoctopuce modules are in fact real computers, they even contain a small web server. And, as all computers, it is possible to update their control software (firmware). New firmware for each module are regularly published, they generally allow you to add new features to the module, and/or to correct a hypothetical bug<sup>2</sup>.

#### Recommended method

To update a module firmware, in the VirtualHub interface, open the configuration window of the module that you want to update, then click on the **upgrade** button. If you simply click on the **Update** button, VirtualHub uses the most recent version of the firmware published on the Yoctopuce web site and installs it.

VirtualHub also allows you to chose a .byn file that you have downloaded previously from the Yoctopuce web site, for example to reinstall a previous version.

<sup>2</sup> Never trust people telling you that their software does not have bugs :-)



Firmware update window

When you have clicked on the **Update** button, everything is automatic. VirtualHub restarts the module in "update" mode, updates the firmware, then restarts the module in regular mode. The module configuration settings are preserved. Do not disconnect the module during the update process.

### Alternative method

If a module update went wrong, in particular if the module was disconnected during the update process, there is a strong risk that it does not work anymore and that it does not appear in the module list. In this case, disconnect the module, wait a few seconds, and reconnect it while keeping the Yocto-button pressed. This starts the module in "update" mode. This working mode is protected against corruptions and should always be available. When the module is reconnected, request a refresh of the module list in the VirtualHub interface and your module should appear at the bottom of the interface. Click on it to update its firmware. This update method is a recovery method, it does not preserve the module settings.



The modules in "update" mode are listed in the interface

A regular update preserves the module configuration, while the alternative method with the Yocto-button resets the module with its factory settings. This means that you can also use this method to completely reinitialize a module.

### With a command line or by programming

All the command line tools can update Yoctopuce modules thanks to the `downloadAndUpdate` command. The module selection mechanism works like for a traditional command. The [target] is the name of the module that you want to update. You can also use the "any" or "all" aliases, or even a name list, where the names are separated by commas, without spaces.

The following example, using the Yoctopuce command line library, automatically downloads the latest available firmware files and updates all the Yoctopuce modules connected by USB:

```
C:\>YModule all downloadAndUpdate
ok: Yocto-PowerRelay RELAYHI1-266C8 (rev=15430) is up to date.
```

```
ok: 0 / 0 hubs in 0.000000s.
ok: 0 / 0 shields in 0.000000s.
ok: 1 / 1 devices in 0.130000s 0.130000s per device.
ok: All devices are now up to date.
C:\>
```

The next example installs the LIGHTMK3.51180.byn firmware, locally stored in C:\tmp\yfirmware folder, on the module with serial number LIGHTMK3-23BBDF. Firmware files are available for manual download from Yoctopuce web server.<sup>3</sup>

```
C:\>ymodule LIGHTMK3-23BBDF updateFirmware C:\tmp\yfirmware\LIGHTMK3.51180.byn
OK: LIGHTMK3-23BBDF.module.updateFirmware = 36% Wait for device.
OK: LIGHTMK3-23BBDF.module.updateFirmware = 50% Flash zone.
OK: LIGHTMK3-23BBDF.module.updateFirmware = 57% Flash zone.
OK: LIGHTMK3-23BBDF.module.updateFirmware = 64% Flash zone.
OK: LIGHTMK3-23BBDF.module.updateFirmware = 72% Flash zone.
OK: LIGHTMK3-23BBDF.module.updateFirmware = 73% Device info retrieved.
OK: LIGHTMK3-23BBDF.module.updateFirmware = 90% Firmware updated.
OK: LIGHTMK3-23BBDF.module.updateFirmware = 100% success.
OK: LIGHTMK3-23BBDF.module.updateFirmware = 100% Firmware Updated Successfully in: 14.305s.
```

By default, `downloadAndUpdate` performs only upgrades to a newer firmware, as the default setting of the optional second parameter `onlyNew` is true. If you want to downgrade your firmware to an older version, you must provide the older firmware as the first parameter and set the second parameter to false.

It is also possible to update your modules' firmware using the Yoctopuce programming library, in particular using `YModule.checkFirmware` and `YModule.updateFirmware` methods. For more information, please refer to the advanced programming chapters in the documentation for each module.

### 3.5. Accessing the sensor data logger

For all Yoctopuce sensors including a data logger, the configuration window contains a specific section enabling you to configure the recording and to modify the raw data saved in the data logger, module per module.

Click on the **configure** button of the **Datalogger and Timed reports** section:

**Data logger configuration**

You can choose which functions you want to record, and the frequency at which data should be recorded. Note that if you choose a recording rate higher than the effective sensor refresh rate, the same value will be recorded multiple time.

**Global settings**

Recording:  On  Off

Auto-start recording at power-on

Link recording to beacon button

**Configurable functions**

Function	Frequency	Recording	Reporting
humidity	1/s	<input checked="" type="checkbox"/>	<input type="checkbox"/>
pressure	1/s	<input checked="" type="checkbox"/>	<input type="checkbox"/>
temperature	1/s	<input checked="" type="checkbox"/>	<input type="checkbox"/>

*Data logger configuration window*

<sup>3</sup> [www.yoctopuce.com/EN/firmwares.php](http://www.yoctopuce.com/EN/firmwares.php)

You can also install as a VirtualHub plug-in the *Yocto-Visualization (for Web)* tool, which offers many more possibilities to visualize data as graphs, and to load a CSV file corresponding to all the connected sensors. For more details, read the section dedicated to the installation of *Yocto-Visualization (for Web)* at the end of this guide.

## 4. Using VirtualHub as a gateway

The less spectacular, but nevertheless the most useful, function of VirtualHub consists in providing a network gateway to control devices. First, it provides an access to languages such as JavaScript which, by nature, prevent you from accessing the physical resources of a machine. Second, it provides access to the devices through the network for all languages: Yoctopuce libraries are indeed able to connect themselves to VirtualHub through the network.

To use VirtualHub as a gateway, you need only to run it in a command line or as a service on the machine on which the devices that you want to control are connected. Applications wanting to connect themselves to VirtualHub must initialize the API by calling the `yRegisterHub` function with the IP address of the machine running VirtualHub, the default port is 4444. For example:

```
yRegisterHub("http://192.168.1.6:4444",errmsg);
```

If the application and VirtualHub run on the same machine, use the 127.0.0.1 address. Refer to the programming API documentation<sup>1</sup> for more details.

### 4.1. A gateway to work around USB access limitation

As mentioned in the "USB access limitation" section, only one application at a given time can have native access to Yoctopuce devices. This limitation is related to the fact that two different processes cannot talk to a USB device at the same time. Usually, this kind of problem is solved by a driver that takes care of the police work to prevent multiple processes fighting over the same device. But as you have probably noticed, Yoctopuce products do not use drivers. Therefore, the first process that manages to access the native mode keeps it for itself until `UnregisterHub` or `FreeApi` is called.

If your application tries to communicate in native mode with Yoctopuce devices, but that another application prevents you from accessing them, you receive the following error message:

```
Another process is already using yAPI
```

The solution is to use VirtualHub locally on your machine and to use it as a gateway for your applications. In this way, if all your applications use VirtualHub, you do not have conflicts anymore and you can access all your devices all the time.

To switch from native to network mode on your local machine, you only need to change the parameter when calling `YAPI.RegisterHub` and to indicate 127.0.0.1 instead of `usb`:

---

<sup>1</sup> <http://www.yoctopuce.com/EN/libraries>

```
YAPI.RegisterHub("usb",errmsg); // native USB mode
YAPI.RegisterHub("127.0.0.1",errmsg); // local network mode
```

## 4.2. REST gateway

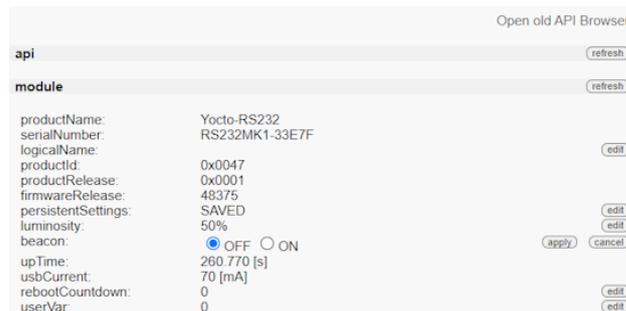
You can also use VirtualHub as a REST gateway. This consists in sending to the module HTTP requests through VirtualHub.

To test this feature, use the **Open API Browser** link available at the bottom of your module interface window, with a web browser:



Link to open the REST interface

In the window that opens, you can modify each attribute of the module, using first the **edit** button and then the **apply** button to apply the changes:



Manually changing an attribute

After you have performed a modification, if you go to the very bottom of the page, you can see the HTTP request which was sent to apply the requested modification:



Corresponding HTTP request

You can thus easily discover how to access the essential functions of our modules with HTTP requests: it is the whole interest of the REST interface. If the semantics of an attribute escapes you, you can always find explanations in the module user's guide.

## 4.3. OpenMetrics gateway (Prometheus)

You can also use VirtualHub as a data source for a Prometheus server, in order to centralize the measures from the Yoctopuce sensors in the same data base as the information on the state of IT infrastructures.

Prometheus data sources are called *exporters*: the idea is that each important system or service can make its vital data available to Prometheus, enabling the system administrator to collect and monitor them. Each *exporter* is an HTTP accessible URL, which provides data following the OpenMetrics standard: one measure per line, with conventions for naming and classifying, enabling system

administrators to find their way among the hundreds of measures at disposal when configuring the dashboard.

VirtualHub has the capacity to be a native OpenMetrics *exporter* through the REST interface. To obtain data in the OpenMetrics format, you simply need to load the `/api/services.om` URL of your VirtualHub, that is `http://127.0.0.1:4444/api/services.om`. For example, if you access this URL when a few sensors are connected locally, you obtain something akin to this (we modified the presentation to make reading easier, but in reality each measure fits on a single line):

```
yocto_temperature_advertisedValue{
  productName="Yocto-Thermocouple",
  serialNumber="THRMCPL1-16397A",
  deviceName="insideProbes",
  functionId="temperature1"} 21.78
yocto_temperature_advertisedValue{
  productName="Yocto-PT100",
  serialNumber="PT100MK1-BA496",
  functionId="temperature"} 28.57
yocto_temperature_advertisedValue{
  productName="Yocto-RangeFinder",
  serialNumber="YRNGFND1-1D1567",
  deviceName="rf",
  functionId="temperature1",
  functionName="rfTemp"} 25.13
yocto_lightSensor_advertisedValue{
  productName="Yocto-RangeFinder",
  serialNumber="YRNGFND1-1D1567",
  deviceName="rf",
  functionId="lightSensor1"} 56
yocto_rangeFinder_advertisedValue{
  productName="Yocto-RangeFinder",
  serialNumber="YRNGFND1-1D1567",
  deviceName="rf",
  functionId="rangeFinder1"} 1456
# EOF
```

To tell Prometheus to collect these data directly from VirtualHub, you must add to your `prometheus.yml` a file section of this type:

```
- job_name: "yoctohub_sensors"
  scrape_interval: 60s
  metrics_path: "/api/services.om"
  static_configs:
    - targets: ['127.0.0.1:4444']
```

As our OpenMetrics exporter is embedded straight into the VirtualHub REST interface, you can use it to obtain more detailed information for each sensor, using a URL pointing to a specific sensor. For example, if you connect a Yocto-Thermocouple to VirtualHub and then you assign it the logical name `tcProbes`, when you access the `/byName/tcProbes/api.om` URL, you obtain something like:

```
yocto_module_luminosity{...,functionId="module",functionName="tcProbes"} 50
yocto_module_beacon{...,functionId="module",functionName="tcProbes"} 0
yocto_module_usbCurrent_mA{...,functionId="module",functionName="tcProbes"} 23
yocto_module_rebootCountdown{...,functionId="module",functionName="tcProbes"} 0
yocto_module_userVar{...,functionId="module",functionName="tcProbes"} 0
yocto_temperature_currentValue_degC{[...],functionName="heatSink"} 21.99
yocto_temperature_lowestValue_degC{[...],functionName="heatSink"} 20.51
yocto_temperature_highestValue_degC{[...],functionName="heatSink"} 22.25
yocto_temperature_currentRawValue_degC{[...],functionName="heatSink"} 21.988
yocto_temperature_signalValue_mV{[...],functionName="heatSink"} -0.162
yocto_temperature_signalValue_mV{[...],functionId="temperature2"} 999.999
yocto_dataLogger_currentRunIndex{[...],functionId="dataLogger"} 0
yocto_dataLogger_autoStart{[...],functionId="dataLogger"} 0
yocto_dataLogger_beaconDriven{[...],functionId="dataLogger"} 0
yocto_dataLogger_usage{[...],functionId="dataLogger"} 0
# EOF
```

Thus, all the numerical attributes of the Yocto-Thermocouple are made available. You can therefore know the min/max values encountered, the voltage measured at the thermocouple terminal, and so on. Note also that in this case, the exported symbol includes the unit, as recommended by

OpenMetrics. When the module detects that an input is not connected (like the `temperature2` function above), metrics that cannot be computed are automatically deleted from the export so that Prometheus reports them as missing, rather than keeping the latest measured value.

To obtain these additional data per sensor, simply add to the `prometheus.yml` file a new section, referencing the sensor either by its serial number (*bySerial*) or by its logical name (*byName*):

```
- job_name: "thermocouple_probes"
  scrape_interval: 60s
  metrics_path: "/byName/tcProbes/api.om"
  static_configs:
    - targets: ['127.0.0.1:4444']
```

## 5. Access control

VirtualHub is able to perform access control to protect your Yoctopuce devices. Click on the **Configure** button on the line matching VirtualHub in the user interface.

Serial	Logical Name	Description	Action
VIRTHUB0-3880db7f1		VirtualHub	<input type="button" value="configure"/> <input type="button" value="view log file"/>
METEOMK2-2072C7		Yocto-Meteo-V2	<input type="button" value="configure"/> <input type="button" value="view log file"/> <input type="button" value="beacon"/>
LIGHTMK3-17F0AF		Yocto-Light-V3	<input type="button" value="configure"/> <input type="button" value="view log file"/> <input type="button" value="beacon"/>

Search:

Click on the **configure** button on the first line

This opens the configuration window for VirtualHub

VIRTHUB0-3880db7f12

Edit parameters for VIRTHUB0-3880db7f12, and click on the **Save** button.

Serial #: VIRTHUB0-3880db7f12  
Product name: VirtualHub  
Software version: 52892  
Logical name:

**Custom files**

Custom files: 0 file, 5620 KB available   
Default HTML page:

**Incoming connections**

Authentication to read information from the devices: NO   
Authentication to make changes to the devices: NO

**Outgoing callbacks**

Callback URL:    
Callback method: POST Yocto-API   
Callback schedule: every 60s  
Network downtime to reboot: no downtime limit

VirtualHub configuration window

Access control can be configured from the **Incoming connections** section. There are two distinct levels of access control

## 5.1. Admin access

The *admin* access locks write access to the Yoctopuce devices. When the admin password is set, only users using the admin login are allowed to configure the devices seen by VirtualHub as they want to.

## 5.2. User access

The *user* access locks all use of the Yoctopuce devices. When set, the *user* password prevents any user from consulting any device properties without the proper credentials.

If you configure a *user* password only, without configuring an *admin* password, all the user must give a password to access the modules, but once they are authenticated, they can also edit module configurations.

If you configure both *user* and *admin* passwords, users logging in with the *user* login are not able to modify the configurations of the modules seen by VirtualHub. *user* access enables access in read-only mode, that is only to consult the state of the modules. Only users using the *admin* login can change the configuration of the modules.

If you configure an *admin* access, without configuring a *user* access, users are still able to read your devices values without any password, but they wont be able to change any device setting. Only the users knowing the *admin* password can change module configurations.

## 5.3. Access control and API

Warning: access control has an impact on Yoctopuce API behavior when trying to connect to VirtualHub with access control enabled. With Yoctopuce API, access control is handled at the `RegisterHub()` level. You need to provide the VirtualHub address as follow: *login:password@adrese:port*, here is an exemple:

```
yRegisterHub("admin:mypass@127.0.0.1:4444",errmsg);
```

If you forget your VirtualHub password, the only way to regain control of your VirtualHub is to delete the VirtualHub configuration file (`.virtualhub.dat`).

## 6. Outgoing connections

VirtualHub is able to connect to external services to communicate the status of the modules connected to it.

VirtualHub knows how to post its data in the format accepted by some third-party cloud services, such as

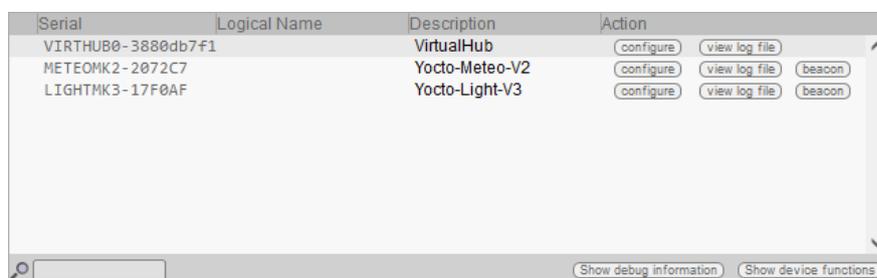
- Emoncms
- InfluxDB (versions 1.0 et 2.0)
- PRTG
- Valarm.net

VirtualHub can also connect to external services using advanced protocols that allow further interaction with the Yoctopuce devices, but which will require a little more knowledge to take advantage of them:

- MQTT
- Yocto-API

### 6.1. Configuration

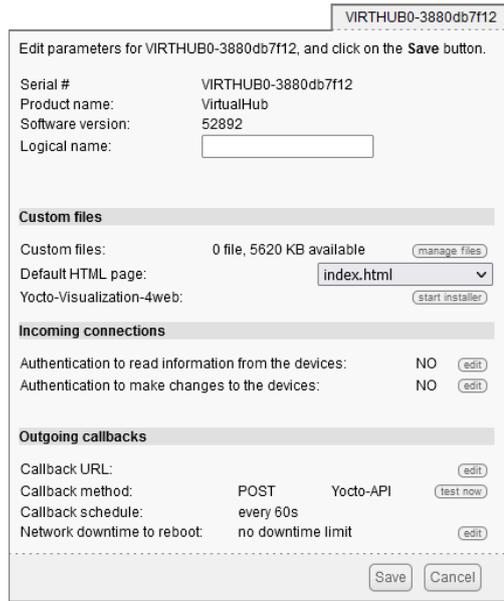
To use this feature, just click on the **configure** button located on the line matching VirtualHub on the interface. Then look for the **Outgoing callback** section and click on the **edit** button.



Serial	Logical Name	Description	Action
VIRTHUB0-3880db7f1		VirtualHub	<a href="#">configure</a> <a href="#">view log file</a>
METEOMK2-2072C7		Yocto-Meteo-V2	<a href="#">configure</a> <a href="#">view log file</a> <a href="#">beacon</a>
LIGHTMK3-17F0AF		Yocto-Light-V3	<a href="#">configure</a> <a href="#">view log file</a> <a href="#">beacon</a>

At the bottom of the interface, there is a search bar and two buttons: "Show debug information" and "Show device functions".

*Just click on the corresponding **configure***



Then edit the **Outgoing callbacks** section.

The callback configuration window shows up. This window allows you to define how VirtualHub interacts with an external web site. Several interaction types are at your disposal.

## 6.2. HTTP Callbacks to 3rd-party services

VirtualHub is able to post on external servers the values of the Yoctopuce sensors at regular intervals and/or whenever a value changes significantly. This feature allows you to store your measures and draw graphs without writing a single line of code.

Yoctopuce is in no way affiliated with these third-party services and can therefore neither guarantee their continuity nor propose improvements to these services.

### Emoncms

Emoncms is an open-source Cloud service enabling you to post Yoctopuce sensor data and then to visualize them. You can also install your own local server.

The parameters that you must provide are the Emoncms API key, the node number that you want to use, as well as the address of the Emoncms server if you use a local server.

You can personalize the names linked to the measures posted on Emoncms. For more details, see the section "Names associated with posted values" below.

### InfluxDB 1.0 and 2.0

InfluxDB is an open-source database specifically dedicated to storing temporal series of measures and events. Note that only local installations are supported. Indeed, the *InfluxDB Cloud* service is not supported as it requires an SSL connection.

Parameters for version 1.0 of InfluxDB are the address of the server and the name of the database.

Version 2.0 of InfluxDB uses a different API and Yoctopuce needs three parameters (organization, bucket, and token) as well as the server address.

You can personalize the names linked to the measures posted on InfluxDB. For more details, see the section "Names associated with posted values" below.

### PRTG

PRTG is a commercial solution designed to supervise systems and applications. You can store measures and obtain graphs from your sensors with this service.

The required parameters are the address of the PRTG server and the `token` enabling the identification of VirtualHub.

You can personalize the names linked to the measures posted on PRTG. For more details, see the section "Names associated with posted values" below.

### Valarm.net

Valarm is a professional cloud service that allows you to record data from Yoctopuce sensors but also allows you more elaborate functions such as the possibility to geolocate measures or to configure Yoctopuce modules remotely.

The only required parameter is a `Routing` code to identify VirtualHub.

## 6.3. Callbacks to an MQTT broker

MQTT is an Internet of Things protocol that allows sensors and actuators to communicate with each other via a central server called an *MQTT broker*. MQTT is particularly used in home automation, where it can federate many technologies to make them accessible to a central control system such as *Home Assistant*.

The basic parameters to provide for the MQTT callback configuration are the MQTT broker address, client ID, `root_topic` as well as authentication parameters. Note that encapsulation of the MQTT protocol in an SSL connection is not supported, which precludes its use with services like AWS IoT Core.

When an MQTT callback is active, VirtualHub is able to publish messages with the status of sensors and actuators, and receive command and configuration messages, allowing the central control system to fully interact with the devices.

The rest of this section describes in detail the supported MQTT messages. It will only be of interest to developers who want to build their own integration with Yoctopuce modules via MQTT. If you are just going to use *Home Assistant*, you can skip this section and thanks to the *MQTT Discovery* mechanism, your devices should automatically appear in *Home Assistant*.

### Common format of all messages

The `topic` of all messages starts with a common part, which identifies the Yoctopuce module and the particular function of this module concerned by the message. It has the following structure:

**`root_topic/deviceName/functionName`**

The `root_topic` can be freely configured, for example to the `yoctopuce` value. If you connect several hubs to the same MQTT *broker*, you can either use the same `root_topic` for all of them or a different topic per hub. Using a separate `root_topic` is recommended if you send a lot of commands to a hub via MQTT.

`deviceName` is the logical name you gave to the designated Yoctopuce module. If no logical name has been configured, the serial number of the module is used instead of the logical name (e.g. `METEOMK2-012345`).

`functionName` is the logical name you gave to the designated function. If no logical name has been configured, the function identifier is used (for example `genericSensor1`).

The advantage of using logical names rather than hardware names in the `topic` root is that you can identify modules and functions by their role and you don't have to explicitly tell the hardware ID of each module to the MQTT client that has to interact with these modules. The disadvantage is that if you decide to change the logical name of your modules or functions without thinking about it, the MQTT `topics` used must be changed accordingly.

### Topic `/api`: complete state of the function

Under `root_topic/deviceName/functionName/api`, each function publishes a JSON structure describing the complete state of the function, including all attributes. In this JSON encoding,

- booleans are represented by 0 and 1
- enumerated types are represented by numeric constants
- real numbers such as measures are represented as integers, after multiplication by 65536. They must therefore be divided by 65536.0 to obtain the actual value.

This message is issued when one of the following conditions occurs

- when the connection between the hub and the MQTT broker is established
- every five minutes
- after a change in the module configuration
- in response to the reception of a command by this function
- for the *module* function, when the *beacon* is activated or deactivated

### Base topic: current state

Under `root_topic/deviceName/functionName`, each function publishes a textual summary of its status. This is not JSON but a simple string, corresponding to the value of the *advertisedValue* attribute of the function. For example, for a sensor, it corresponds to the current value of the sensor, while for a relay it corresponds to the letter A or B depending on the switching state.

This message is issued when one of the following conditions occurs

- when the connection between the hub and the MQTT broker is established
- every five minutes
- every time the *advertisedValue* attribute changes

To avoid overloading the MQTT broker with changes in the current values of the sensors, it is possible to globally disable the sending of current value messages for the sensors only, in the MQTT configuration.

### Topics `/avg`, `/min`, `/max`: average and extreme values

Under `root_topic/deviceName/functionName/avg`, the sensor functions (subclasses of *Sensor*) periodically publish the average value observed during the previous time interval directly as a real number.

Under `root_topic/deviceName/functionName/min`, the minimum value observed during the previous time interval.

Under `root_topic/deviceName/functionName/max`, the maximum value observed during the previous time interval.

These messages are the direct equivalent of the *timed reports* documented in the user's guide of these modules. The time interval must have been previously configured in the *reportFrequency* attribute. Otherwise, these messages are not sent.

### Topics `/set/attributeName`: command sending and configuration

Under `root_topic/deviceName/functionName/set/attributeName`, it is possible to send messages to modify the attributes of functions, in order to change a function state or configuration. The value of the message corresponds to the new desired value, as is. The format is identical to the one used for the REST gateway of VirtualHub (see the REST gateway section of this guide).

For example, we could switch a relay by sending a message to the *topic*

```
yoctopuce/PumpRelay/relay1/set/state
```

with value 1.

We could also trigger a 1500ms pulse on the same relay by sending a message to the *topic*

```
yoctopuce/PumpRelay/relay1/set/pulseTimer
```

with value 1500.

To receive commands and configuration changes via MQTT, you must have explicitly enabled it in the MQTT configuration on the Yoctopuce hub. For safety, the basic behavior of the MQTT mode remains read-only.

### Topic */rdy*: availability status

Under `root_topic/deviceName/module/rdy`, the module function publishes a binary indication of the availability status of the device. The value is 1 when the module is online, and 0 when it is offline.

This message is published by the hub for its own module when one of the following conditions occurs

- when the hub establishes a connection with the MQTT broker
- when the hub disconnects from the MQTT broker

This message is published for devices other than the hub when one of the following conditions occurs

- when establishing the hub connection with the MQTT broker
- when a device is connected to the hub
- when a device is disconnected from the hub

To determine if a module is really reachable, it is therefore necessary to check its own */rdy* topic, to know if it has been disconnected, and the */rdy* topic of the hub, to know if the MQTT connection is active.

### MQTT discovery

In addition, special messages are published under the topic `homeassistant/` just after the connection of the hub with the MQTT broker is established, and repeated every 5 minutes, to allow the automatic detection of the proposed features, thanks to the *MQTT discovery* mechanism supported by Home Assistant and openHab.

## 6.4. Yocto-API callbacks

Yocto-API callbacks use a specific protocol defined by Yoctopuce, which allows a very advanced interaction with the Yoctopuce modules. Using languages such as PHP, TypeScript, JavaScript or Java, they allow the programmer of the web service to directly use the functions of the Yoctopuce programming library to interact with the modules that connect via HTTP callback. This allows you in particular to control from a public web site Yoctopuce modules installed behind a private ADSL router. You can for example switch the output of a relay depending on the value of a sensor, while keeping the complete control of the system on a web server.

Yoctopuce provides a free application that exploits to the maximum the possibilities of the Yocto-API callback on a PHP server: VirtualHub for Web. This web application allows you to interact remotely with modules that connect periodically via a Yocto-API callback. More information about VirtualHub for Web is available on the Yoctopuce blog <sup>1</sup>.

In Yocto-API or Yocto-API-JZON callback mode, you can choose between the "HTTP" and "WebSocket" protocols.

<sup>1</sup> <https://www.yoctopuce.com/EN/article/new-a-virtualhub-that-works-through-the-web>

## Yocto-API callbacks in WebSocket mode

The communication flow in a standard HTTP request is extremely simple: the client sends a request, and then listens for the server reply. It is not a real conversation, rather just a question and an answer. The HTTP client cannot respond to what the server says without restarting a new separate communication.

However there is a provision in the HTTP 1.1 standard for the client to request an upgrade of the communication protocol. One such protocol upgrade widely supported is called WebSockets and is defined in RFC 6455. This upgrade transforms the simple text-based request/reply link into a bidirectional messaging link, that can send arbitrary data in both direction.

Upgrading an HTTP connection to WebSockets requires that both sides support the upgrade. VirtualHub and programming libraries do support it. But in order to upgrade an HTTP callback connection into a WebSocket callback, you also need to ensure that your web server supports WebSocket connection upgrade. This is well supported on Java and Node.JS. This is however not the case currently for PHP implementation on Apache.

WebSockets provide access to a number of advanced features of Yoctopuce API that were unavailable in HTTP callback mode:

- a WebSocket callback can browse and retrieve data from any Yoctopuce sensor built-in data logger.
- a WebSocket callback can use bidirectional communication functions with serial devices, for instance to execute MODBUS queries over a Yocto-RS485.
- a WebSocket callback can use value change callbacks and timed report callbacks to receive real-time and averaged measures from sensors.
- a WebSocket callback can keep long connections between a hub and a server, and handle interactive behaviors, since API calls are made in real time.
- a WebSocket callback can even perform a remote firmware upgrade.

## 6.5. User defined HTTP callbacks

If none of the other options for HTTP callback configuration suits your needs, you can try specifying how the data should be sent yourself. User defined callback allows you to customize the way VirtualHub sends information to the server. Note that only HTTP protocol is supported (no HTTPS).

This host can post the advertised values of all devices to a specific URL on a regular basis. If you wish to use this feature, select your preferred callback type and follow the configuration steps carefully.

1. Specify the type of callback you want to use:

2. Specify the URL to use for reporting values. *HTTPS protocol is not yet supported.*  
 Callback URL:

3. Specify the type of request and data format to be used:

HTTP Method:	Data encoding:
<input checked="" type="radio"/> POST	<input checked="" type="radio"/> WWW-Form-UrlEncoded
<input type="radio"/> PUT	<input type="radio"/> CSV (comma-delimited)
<input type="radio"/> GET	<input type="radio"/> JSON object
	<input type="radio"/> JSON object array
	<input type="radio"/> JSON (numerical)
	<input type="radio"/> Emoncms
	<input type="radio"/> Microsoft Azure
	<input type="radio"/> InfluxDB
	<input type="radio"/> MQTT
	<input type="radio"/> Yocto-API

4. Specify the Type of security you want to use:

Username:

Password:

The callback configuration window

If you want to secure access to your callback script, you can setup a standard HTTP authentication on the web server. VirtualHub knows how to handle standard HTTP authentication schemes: simply provide the user and password fields needed to access the URL. Both "Basic" and "Digest" authentication are supported. However, "Digest" authentication is highly recommended, since it uses a challenge mechanism that avoids sending the password itself over the Internet, and prevents replays.

As an example, here is a PHP script enabling you to visualize in the debug window the content of the data posted by a user-defined HTTP callback in POST and WWW-Form-UrlEncoded mode.

```
<?php
Print(Date('H:i:s')."\\r\\n");
foreach ($_POST as $key => $value) {
    Print("$key=$value\\r\\n");
}
?>
```

You can personalize the names linked to the measures posted by a user-defined HTTP callback. For more details, see the section "Names associated with posted values" below.

## 6.6. Names associated with posted values

With the exception of Yocto-API callbacks which give access to all the information about the Yoctopuce modules, callbacks are designed to transmit only the most important information to the server, associating each value with a name that makes it easy to link it to its origin.

### Basic behavior

Standard behavior is to transmit the value of the `advertisedValue` attribute for each function present on the Yoctopuce modules. The automatically associated name for each value follows this logic:

1. If the function has been given a logical name:

```
FUNCTION_LOGICAL_NAME = VALUE
```

2. If the module has been given a logical name, but not the function:

```
MODULE_NAME.HARDWARE_NAME = VALUE
```

3. If no logical name was set:

```
SERIAL_NUMBER.HARDWARE_NAME = VALUE
```

The easiest way to customize the names associated with the values is to configure the desired name as the logical name of the function, or otherwise as the logical name of the module itself.

Here is an example of data posted by a user-defined HTTP callback in POST mode and *JSON (numerical)* format for a system with a Yocto-Watt where each function has an explicit logical name (for example `VoltageDC`) and a Yocto-Meteo-V2 where the module itself was assigned the `Ambient` logical name:

```
{ "timestamp":1678276738,
  "CurrentAC":0
, "CurrentDC":0
, "VoltageAC":0
, "VoltageDC":0
, "Power":0
, "Ambient.temperature":22.17
, "Ambient.pressure":949.36
, "Ambient.humidity":30
}
```

### Advanced customization with a file

If you wish to further customize the format of the transmitted data, to specifically select which attribute of which module should be sent under which name, or to add contextual information, it is also possible, but it is a little more complicated. To do so, you need to create a template file defining the exact format of the data to be sent. The content and the name of this file is specific to each type of HTTP callback, and each case is explained individually below.

The common point to all the template files is that their content is sent as is to the server, with the exception of expressions included between *backquotes* (or *backticks*, character ```, ASCII code 96)

which are evaluated by the REST gateway of VirtualHub (see the REST gateway section of this guide). For example, if the template file contains the text:

```
{ "origin": "`/api/module/productName`" }
```

then the content actually posted is

```
{ "origin": "VirtualHub" }
```

## Customization file for Emoncms

The basic format used by VirtualHub for Emoncms has the form below. Note that the data is transmitted in the URL, therefore in a single line, but it has been put here on several lines to make it easier to read.

```
time=1678277614
&json={
  "CurrentAC":0,
  "CurrentDC":0,
  "VoltageAC":0,
  "VoltageDC":0,
  "Power":0,
  "Ambient.temperature":22.28,
  "Ambient.pressure":949.54,
  "Ambient.humidity":29.7
}
```

To customize the format of the data sent to Emoncms, you need to create a format template file on VirtualHub with the name `EMONCMS_cb.fmt`.

This file should be a single line, with no carriage returns, and start with a string of type `&json=`. For example, to post only the absolute and relative humidity, you could use (without carriage return!):

```
&json={
  "absoluteHumidity"="/byName/Ambient/api/humidity/absHum",
  "relativeHumidity"="/byName/Ambient/api/humidity/relHum"
}
```

## Customization file for InfluxDB

The basic format used by VirtualHub for InfluxDB has the format shown below. It associates all the values to a *yoctopuce* measure base and adds a *name* tag with the network name of VirtualHub and an *ip* tag with its IP address. Then each value is posted in a field whose name follows the basic convention described above. The data is transmitted by a CSV POST, in a single line, but it has been put here on several lines to facilitate reading.

```
yoctopuce,name=VIRTHUB0-12345678,ip=192.168.1.10
CurrentAC=0,CurrentDC=0,VoltageAC=0,VoltageDC=0,Power=0,
Ambient_temperature=22.5,Ambient_pressure=948.63,Ambient_humidity=29.4
1678281649
```

To customize the format of the data sent to InfluxDB, you must create on VirtualHub a format template file with the `INFLUXDB_cb.fmt` (version 1.0) or `INFLUXDB_V2_cb.fmt` (version 2.0) name.

This file can have several lines if you wish, which allows you to use different tags for different measures, or even to split measures over several databases.

For example, to post for absolute humidity and relative humidity, but with distinct tags, you could use the following format file:

```
humidity,type=relative,location=Library relHum=`/byName/Ambient/api/humidity/relHum`
humidity,type=absolute,location=Library absHum=`/byName/Ambient/api/humidity/absHum`
```

Note: InfluxDB servers accept carriage return only in the UNIX format (character `\n`, also called LF). If you edit the file on a Windows machine, take care to use a text editor able to not add the Windows carriage return (`\r\n`, also called CR LF).

## Customization file for PRTG

The basic format used by VirtualHub for PRTG has the format below. It posts each value to a channel whose name follows the basic convention described earlier. The data is transmitted by a CSV POST, in a single line, but here they have been put on several lines to facilitate reading.

```
{ "prtg": { "result": [
  { "channel": "CurrentAC", "value": "0", "float": "1", "DecimalMode": "All" }
  , { "channel": "CurrentDC", "value": "0", "float": "1", "DecimalMode": "All" }
  , { "channel": "VoltageAC", "value": "0", "float": "1", "DecimalMode": "All" }
  , { "channel": "VoltageDC", "value": "0", "float": "1", "DecimalMode": "All" }
  , { "channel": "Power", "value": "0", "float": "1", "DecimalMode": "All" }
  , { "channel": "Ambient.temperature", "value": "22.48", "float": "1", "DecimalMode": "All" }
  , { "channel": "Ambient.pressure", "value": "948.68", "float": "1", "DecimalMode": "All" }
  , { "channel": "Ambient.humidity", "value": "29.7", "float": "1", "DecimalMode": "All" }
  ] } }
```

To customize the format of the data sent to PRTG, you must create a format template file on VirtualHub with the `PRTG_cb.fmt` name.

This file must have at least the same first line and last line as the example above. However, the description of the channels can be completely customized.

For example, to post absolute humidity and relative humidity simultaneously, but in two distinct channels, you could use the following format file:

```
{ "prtg": { "result": [
  { "channel": "relHum", "value": "`/byName/Ambient/api/humidity/relHum`",
    "float": "1", "DecimalMode": "All" },
  { "channel": "absHum", "value": "`/byName/Ambient/api/humidity/absHum`",
    "float": "1", "DecimalMode": "All" }
  ] } }
```

## 6.7. Scheduling callbacks

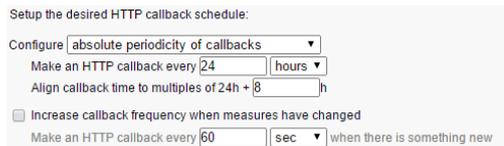
The callback scheduling section is present for all types of callbacks. It is the last section containing fields to be filled.

A first callback is always made just a few seconds after VirtualHub starts up. Subsequent callbacks are made according to the scheduling settings, which determine the callback frequency.

You can select between two planning methods: either by configuring the time interval between two subsequent callbacks, or by specifying an absolute periodicity, to obtain callbacks at fixed times. You can specify the interval between callbacks in seconds, minutes, or hours.

The `interval between subsequent callbacks` option allows you to specify the delay between each callback. That is, if you configure an interval of 5 minutes, VirtualHub waits 5 minutes before triggering the next callback. If the first callback is triggered at 12:03, the next one is executed at 12:08, and so on.

The `absolute periodicity of callbacks` option allows you to configure callbacks at a fixed time. That is, the callback is triggered every multiple of the configured delay. For example a delay of 5 minutes triggers callbacks at 8h00, 8h05, 8h10, and so on. Note that in this mode, it is also possible to specify an offset from the configured delay. For example with a 24 hour delay, it is possible to use an offset of 8h to trigger the callback every day at 8h in the morning.



The callback scheduling parameters

You can explicitly select if you want the callback frequency to vary when no new measure is detected. This enables you to lower the minimal transmission frequency to lower the quantity of data sent over the network if nothing is happening.

Be aware that if you setup many hubs to each make an callback at the same time, you may create a load peak on your web server. So make sure to use an offset parameter to balance load over time.

### 6.8. Tests

In order to help you debug the process, VirtualHub can show you the answers to the callback sent by the web server. In the callback configuration window, click on the **test** button when all required fields are filled to open the test window.

The window shows you the actual state of the callback system to enable you to see, for example if a callback is currently running on the web server. In any case, while this window is open, no HTTP callback is automatically triggered. You can trigger callbacks manually by pressing on the **Test** button. Once triggered, the window displays the information returned by the web service, as in the example below:



The result of the test callback with a Yocto-PowerRelay and a Yocto-Temperature

If the result meets your expectations, close the debug window and then click on the **OK** button.

### 6.9. Spontaneous connections

On top of the connections linked to outgoing callbacks described in the sections above, VirtualHub occasionally tries to establish outgoing connections. These connections are the following:

- Installing Yocto-Visualization (for web): When the user initiates the installation of Yocto-Visualization (for web) from the VirtualHub interface, the hub automatically downloads the most recent installation file from [www.yoctopuce.com](http://www.yoctopuce.com)
- Version test: every time the property or configuration window of a module is opened, VirtualHub makes a request to [www.yoctopuce.com](http://www.yoctopuce.com) to check if the module firmware is up to date. This query contains only the serial number of the module and is used to tell the user if a new firmware is available.
- Firmware download: Each time the firmware update window is opened, VirtualHub makes a request to [www.yoctopuce.com](http://www.yoctopuce.com) to retrieve the most recent firmware. This connection saves the user from having to download the latest firmware manually.

Note that these connections are in fact established by the web browser displaying the VirtualHub user interface. Moreover, these connections are purely optional, if they cannot be established, the application continues to function normally.

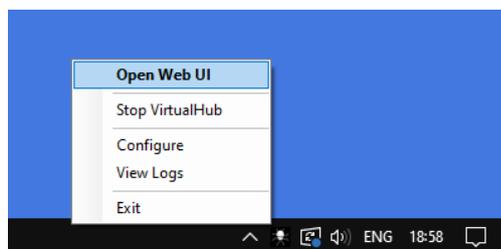
## 7. Optional components

### 7.1. The Windows VirtualHub Control tool

This tool is automatically installed on Windows when VirtualHub is set up using the .msi installer. It is an application running in the task bar which enables you to control VirtualHub. With it, you can:

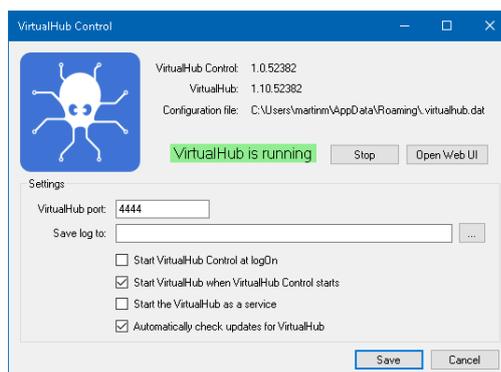
- Access the VirtualHub web interface
- Start/stop VirtualHub
- Obtain the VirtualHub logs
- Configure VirtualHub

A simple click on the icon brings up a menu with the available operations, while a double-click displays the VirtualHub web interface in the default web browser.



*Operations directly available from the taskbar*

The **Configure** menu brings up the VirtualHub configuration window. In this window, you can find some information about VirtualHub such as the version number and the location of configuration or log files.

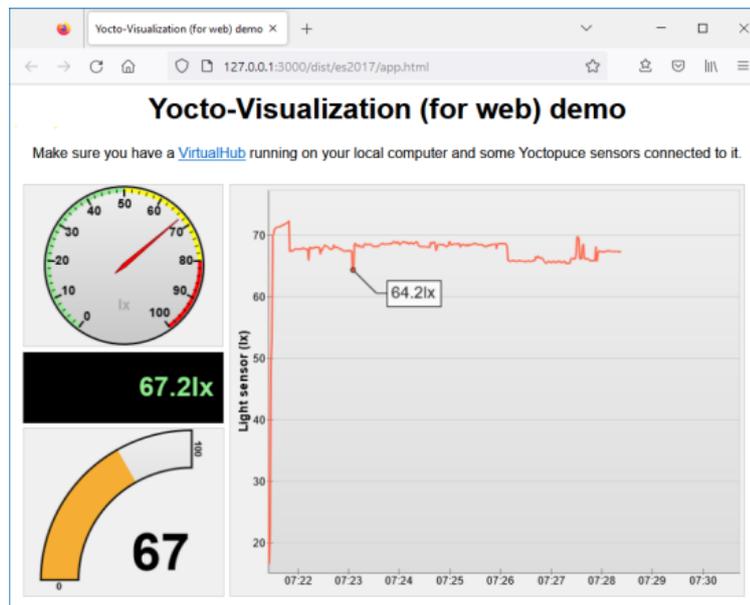


*VirtualHub configuration window*

This window also allows you to configure the VirtualHub startup options. It is possible to choose the port used by VirtualHub, as well as the location of the log files. You can also configure VirtualHub to run as a Windows service. In fact, this window allows you to specify some of the parameters that were previously passed on the command line.

## 7.2. Installing Yocto-Visualization (for web)

Yocto-Visualization (for web) is a small web application that allows you to easily visualize the values measured by Yoctopuce sensors.

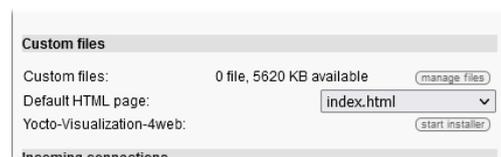


*Yocto-Visualization (for web) interface*

You can install Yocto-Visualization (for web) as a VirtualHub plug-in, directly through the VirtualHub web configuration interface. To do so, VirtualHub must be launched with the `-F` option, i.e. the Files interface must be active and therefore allow the addition of custom files in VirtualHub. Do not forget that this option takes as argument the name of the "container" file in tar format in which the custom files will be stored.

To install Yocto-Visualization (for web) from the VirtualHub web interface

1. In the VirtualHub interface, open the VirtualHub configuration window by clicking on **configure**.
2. In the **Custom files** section, opposite to `Yocto-Visualization-4web`, click on **start installer**.
3. Then you only need to follow the steps by clicking on **Next** repeatedly and then on **OK**. Simply make sure that in the port is present only once in the **Hub address** field.



*The Custom files section from the VirtualHub configuration window*

To use Yocto-Visualization (for web), please refer to the blog posts on the Yoctopuce web site.

## 8. VirtualHub 2.0

At the time of writing (March 2023) of this user's guide, VirtualHub 2.0 is a BETA version. VirtualHub 2.0 includes HTTPS support.

Despite the major version change, the new VirtualHub 2.0 remains compatible with all Yoctopuce components. In fact, from the outside the 2.0 version is identical to version 1.0, but internally a large part of the code has been rewritten.

### 8.1. Incoming connections

The main new feature is the addition of SSL/TLS support which allows VirtualHub to accept incoming HTTPS connections. From now on, VirtualHub accepts regular HTTP connections on port 4444 and also HTTPS connections on port 4443. You can access VirtualHub in a secure and encrypted way with your browser using the `https://myhostname:4443` URL.

Naturally, you can configure these ports with options on the command line:

- the `-p` option enables you to change the HTTP port
- the `-P` option enables you to change the HTTPS port
- the `--no_https` option completely disables SSL/TLS support
- the `--only_https` option forces the use of HTTPS on all incoming connections

### 8.2. Outgoing connections

VirtualHub 2.0 also includes SSL/TLS support for outgoing connections. It is therefore possible to configure an Outgoing HTTP callback on a server that requires an HTTPS connection, such as InfluxDB OSS v2.0.

To use this option, when configuring an outbound callback, open the drop-down menu to select the connection type and choose the `https://` option.

### 8.3. IPv6

By default, VirtualHub 2.0 operates in IPv4 only and will therefore not respond to requests from IPv6 interfaces. If you want to use IPv6, you need to launch VirtualHub with one of the following two options: `--enable_ipv6` or `--only_ipv6`.

The `--enable_ipv6` option activates the IPv6 network stack, but keeps the IPv4 stack active. In other words, VirtualHub can use both IPv6 and IPv4 network interfaces. The `--only_ipv6` option forces the use of IPv6 network interfaces.



## 8.4. Supported operating systems

At the time of writing this documentation, VirtualHub version 2.0 is a beta version. Only the following operating systems and architectures are supported:

- Windows Intel, 32-bit and 64-bit
- Linux Intel, 32-bit and 64-bit
- Linux ARM, 64-bit
- macOS