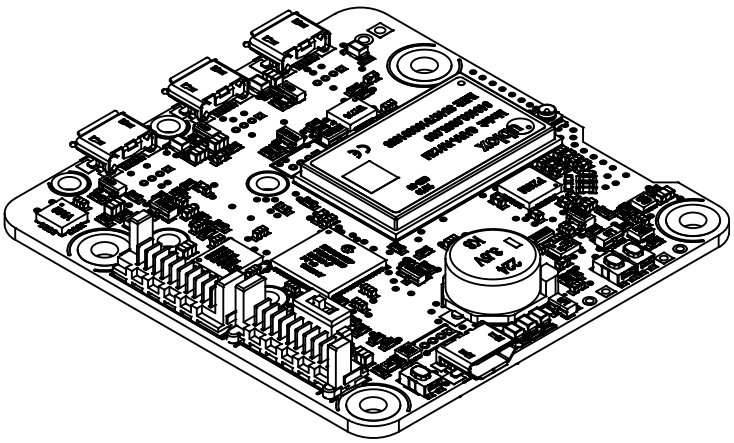
YoctoHub-GSM-4G

User's guide

Table of contents

1. Introduction

The YoctoHub-GSM-4G is a 60x58mm electronic module enabling you to drive other Yoctopuce modules through a cellular connection on a 4G (LTE-M or NB-IoT) or 2G network. For 4G, the YoctoHub-GSM-4G works on LTE bands 2, 3, 4, 5, 8, 12, 13 and 20 in LTE-M (LTE Cat M1) and NB-IoT (LTE Cat NB1) modes. For 2G, the YoctoHub-GSM-4G works on GSM bands 850 MHz, E-GSM 900 MHz, DCS 1800 MHz and PCS 1900 MHz (quad band), in GPRS and EGPRS (EDGE) modes.[1]



The YoctoHub-GSM-4G

The YoctoHub-GSM-4G is designed to be easily deployed and to not require any specific maintenance. In the opposite to a mini-computer, it does not have a complex operating system. Settings can be modified manually or automatically through USB. Therefore, the YoctoHub-GSM-4G is much more suited to industrialization than a mini-computer. However, you cannot run additional software written by the user on the YoctoHub-GSM-4G.

The YoctoHub-GSM-4G is not a standard USB hub with network access. Although it uses USB cables, its down ports use a proprietary protocol, much simpler than USB. It is therefore not possible to control, or even to power, standard USB devices with a YoctoHub-GSM-4G.

Yoctopuce thanks you for buying this YoctoHub-GSM-4G and sincerely hopes that you will be satisfied with it. The Yoctopuce engineers have put a large amount of effort to ensure that your

[1] For a detailed list of supported frequency bands by country, consult the Wikipedia pages https://en.wikipedia.org/wiki/LTE_frequency_bands and http://en.wikipedia.org/wiki/GSM_frequency_bands . Note that some countries have not yet deployed LTE-M and/or NB-IoT support.

YoctoHub-GSM-4G is easy to install anywhere and easy to use in any circumstance. If you are nevertheless disappointed with this device, do not hesitate to contact Yoctopuce support[2].

1.1. Optional accessories

The accessories below are not strictly necessary, but they might help you to get the better of your YoctoHub-GSM-4G.

Screws and spacers

In order to mount the Yocto-3D module, you can put small screws in the 3mm assembly holes, with a screw head no larger than 8mm. The best way is to use threaded spacers, which you can then mount wherever you want. You can find more details on this topic in the chapter about assembly and connections.

USB MicroB-MicroB cable

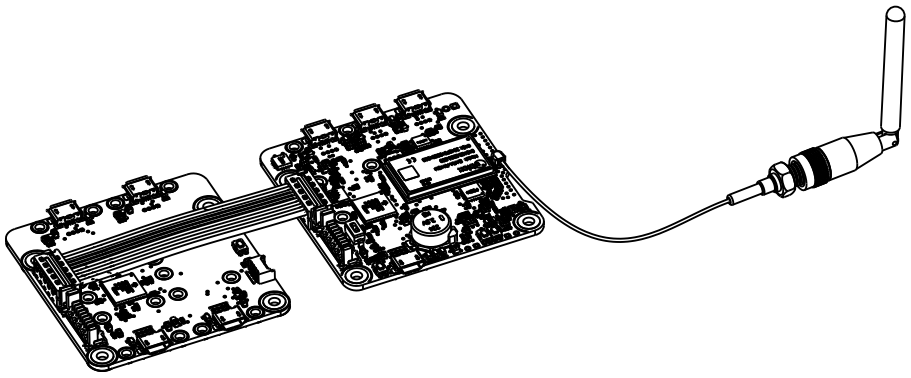
The YoctoHub-GSM-4G connectivity is mainly achieved through USB microB connectors. This means you will have to use USB cables with a microB connector on both ends. These cables are not very common, but you can order some from the Yoctopuce online shop[3].

1.27mm pitch connector

USB cable are very handy to quickly interconnect Yoctopuce devices. However, these use a lot of space. That's why there is, on the YoctoHub-GSM-4G PCB, a small footprint for 1.27 or 1.25mm pitch connectors near each USB connector. Soldering 1.27 pitch connector on these footprints allows to use a much more compact wiring. These 1.27mm connectors are quite standard and can be ordered from any electronic shop. Yoctopuce sells the *1.27-1.27-11* product, which is a set of connector with a 11cm long cable.

YoctoHub-Shield

The YoctoHub-GSM-4G features three down ports allowing to connect up to three sub-devices. However this capacity can be raised, thanks to the *YoctoHub-Shield*. Each shield add 4 new ports, and up to 10 shields can be daisy-chained to your YoctoHub-GSM-4G. See the YoctoHub-Shield documentation for more details.

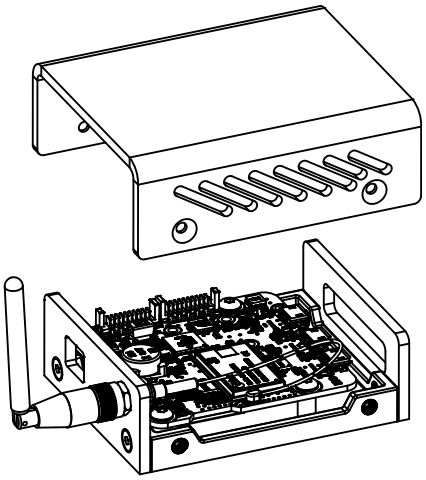


The YoctoHub-Shield adds more ports to your YoctoHub-GSM-4G.

Enclosures

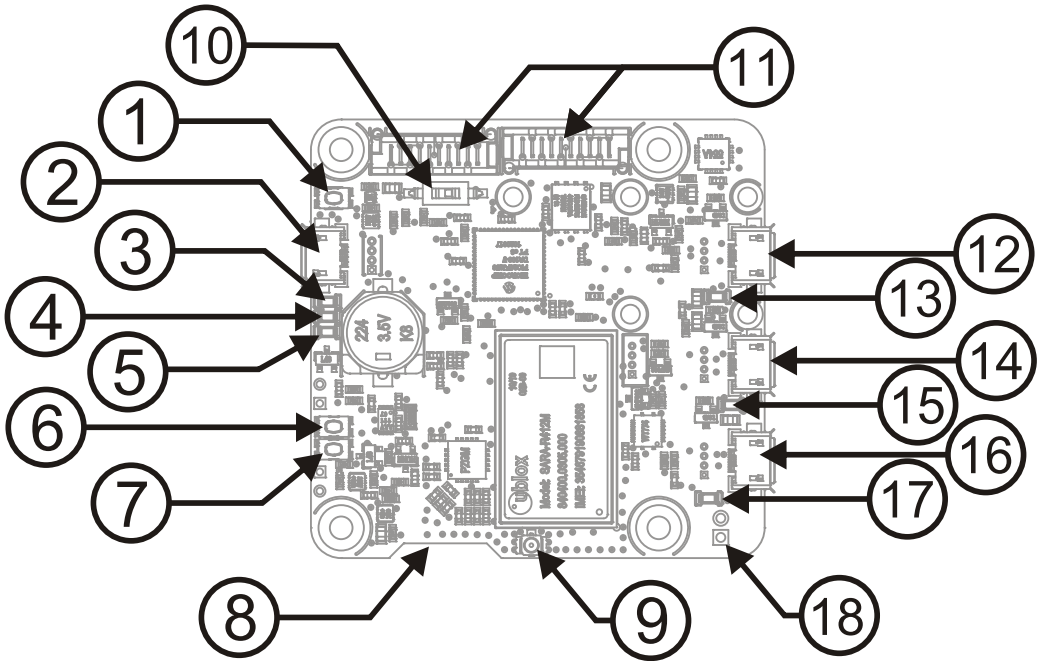
Your YoctoHub-GSM-4G has been designed to be installed as is in your project. Nevertheless, Yoctopuce sells enclosures specifically designed for Yoctopuce devices. More details are available on the Yoctopuce web site. The suggested enclosure model for your YoctoHub-GSM-4G is the YoctoBox-HubWlan-Transp.

[2] support@yoctopuce.com
[3] *USB-OTG-MicroB-MicroB-20* and *USB-OTG-MicroB-MicroB-100*



Your YoctoHub-GSM-4G can be installed in an enclosure.

2. Presentation



1: Yocto-button
2: Control + power USB port
3: Yocto-LED
4: Overload LED
5: Network transfer LED
6: Wake up button
7: Sleep button
8: SIM card holder (below)
9: Antenna connector
10: Sleep neutralization
11: Back connection
12: Down port 1
13: Down port 1 LED
14: Down port 2
15: Down port 2 LED
16: Down port 3
17: Down port 3 LED
18: Airplane mode trigger

2.1. The YoctoHub-GSM-4G components

Serial number

Each Yocto-module has a unique serial number assigned to it at the factory. For YoctoHub-GSM-4G modules, this number starts with YHUBGSM5. The module can be software driven using this serial number. The serial number cannot be modified.

Logical name

The logical name is similar to the serial number: it is a supposedly unique character string which allows you to reference your module by software. However, in the opposite of the serial number, the logical name can be modified at will. The advantage is to enable you to build several copies of the same project without needing to modify the driving software. You only need to program the same logical name in each copy. Warning: the behavior of a project becomes unpredictable when it contains several modules with the same logical name and when the driving software tries to access one of these modules through its logical name. When leaving the factory, modules do not have an assigned logical name. It is yours to define.

Yocto-button

The Yocto-button has two functionalities. First, it can activate the Yocto-beacon mode (see below under Yocto-led). Second, if you plug in a Yocto-module while keeping this button pressed, you can then reprogram its firmware with a new version. Note that there is a simpler UI-based method to update the firmware, but this one works even if the firmware on the module is incomplete or corrupted.

Yocto-led

Normally, the Yocto-led is used to indicate that the module is working smoothly. The Yocto-led then emits a low blue light which varies slowly, mimicking breathing. The Yocto-led stops breathing when the module is not communicating any more, as for instance when powered by a USB hub which is disconnected from any active computer.

When you press the Yocto-button, the Yocto-led switches to Yocto-beacon mode. It starts flashing faster with a stronger light, in order to facilitate the localization of a module when you have several identical ones. It is indeed possible to trigger off the Yocto-beacon by software, as it is possible to detect by software that a Yocto-beacon is on.

The Yocto-led has a third functionality, which is less pleasant: when the internal software which controls the module encounters a fatal error, the Yocto-led starts emitting an SOS in morse [1]. If this happens, unplug and re-plug the module. If it happens again, check that the module contains the latest version of the firmware and, if it is the case, contact Yoctopuce support[2].

Power / Control port

This port allows you to power the YoctoHub-GSM-4G and the modules connected to it with a simple USB charger. This port also allows you to control the YoctoHub-GSM-4G by USB, exactly like you can do it with a classic Yoctopuce module. It is particularly useful when you want to configure the YoctoHub-GSM-4G without knowing its IP address.

Down ports

You can connect up to three Yoctopuce modules on these ports. They will then be available as if they were connected to a computer running VirtualHub. Note that the protocol used between the YoctoHub-GSM-4G and the USB modules is not USB but a lighter proprietary protocol. Therefore, the YoctoHub-GSM-4G cannot manage devices other than Yoctopuce devices. A standard USB hub does not work either[3]. If you want to connect more than three Yoctopuce modules, just connect one or more YoctoHub-Shield[4] to the back ports.

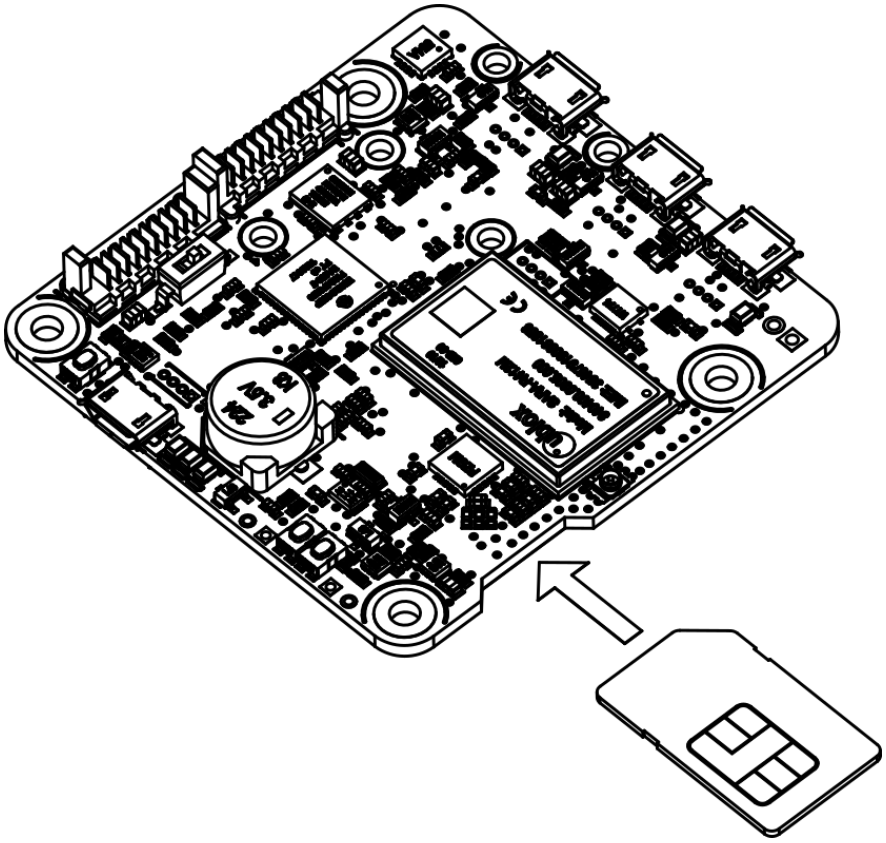
Warning: the USB connectors are simply soldered in surface and can be pulled out if the USB plug acts as a lever. In this case, if the tracks stayed in position, the connector can be soldered back with a good iron and flux to avoid bridges. Alternatively, you can solder a USB cable directly in the 1.27mm-spaced holes near the connector.

[1] short-short-short long-long-long short-short-short
[2] support@yoctopuce.com
[3] The Yoctopuce Micro-USB-Hub is a standard USB hub and does not work either.
[4] www.yoctopuce.com/FR/products/yoctohub-shield

The SIM card holder

To connect the YoctoHub-GSM-4G to a cellular network, you must insert in your YoctoHub-GSM-4G a SIM card, combined with a subscription allowing data transfers. The SIM card holder is designed for the most standard mini-SIM format, also known as 2FF. Adaptors enabling the use Micro-SIM or Nano-SIM cards can be found in any mobile phone store. The SIM card holder is of the *push-push* type: push to insert the SIM until it is in position and makes a small click. To remove the SIM card, don't pull it but push it a second time to eject it from the holder.

You must insert the SIM card with the metal contacts against the printed circuit.

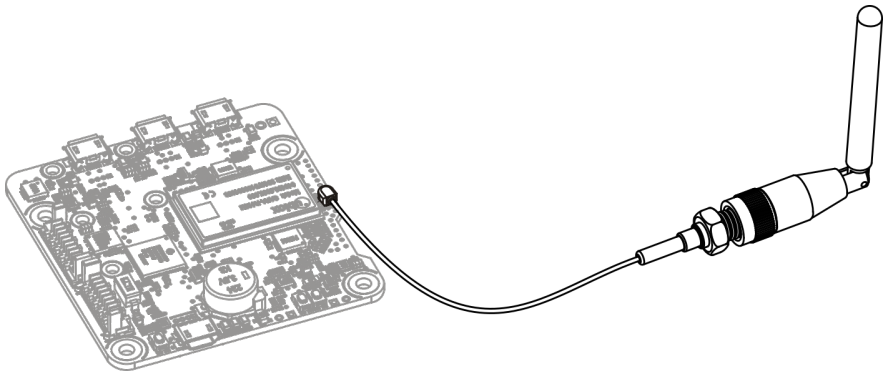


Direction of insertion of the SIM card in the YoctoHub-GSM-4G.

Antenna connector

The YoctoHub-GSM-4G includes an ultra miniature coaxial antenna connector (UFL). Take great care of the UFL connector. It is fragile and is not designed to support many connection/deconnection cycles. The YoctoHub-GSM-4G is sold with a small UFL cable to RP-SMA socket[5] and a corresponding RP-SMA plug antenna[6]. You can use another antenna of your choice, as long as it is designed for the frequency range used in your country for the cellular network and it has the correct connector. Be aware also that using a high-gain antenna may drive you to emit a signal stronger than the authorized norm in your country.

[5] Reverse polarity SMA: threaded on the outside with a plug in the center
[6] Threaded on the inside, jack in the center



Antenna connection

Overload led

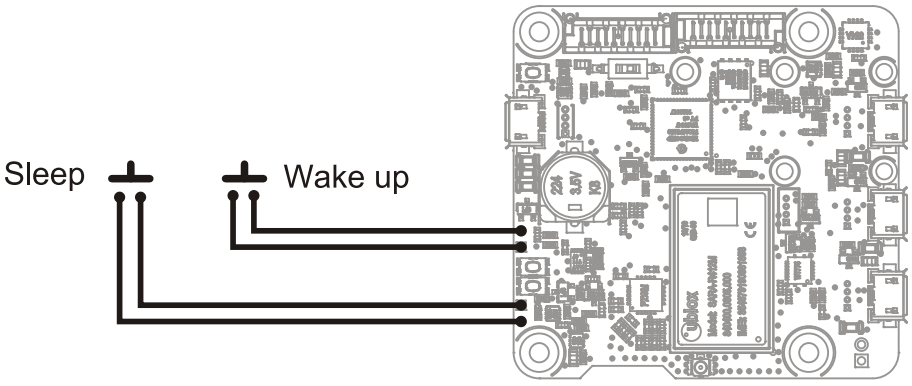
The YoctoHub-GSM-4G continuously monitors its power consumption. If it detects a global consumption of more that 2A, following an overload on one of the down ports for example, it automatically disables all the down ports and lights the overload led. To isolate the source of the issue, you can reactivate the ports one by one, monitoring the power consumption increase. Alternatively, if you know the source of the overload issue and know to have solved it, you can restart the YoctoHub-GSM-4G to enable all its ports at once.

Note that the overload led is a protection measure which can prevent overheating, but it is not a protection guarantee against shorts.

Sleep

On average, the YoctoHub-GSM-4G consumes about 0.5 Watt (50mA), to which you must add the connected modules consumption. But it is able to get into sleep to reduce its power consumption to a strict minimum, and to wake up at a precise time or when an outside contact is closed. This feature is very useful to build measuring installations working on a battery. When the YoctoHub-GSM-4G is in sleep mode, most of the electronics of the module as well as the connected Yoctopuce modules are switched off. This reduces the total consumption to 75 µW (15 µA).

Switching to sleep and waking up can be programmed based on a schedule, controlled by software, or controlled manually with two push buttons located on the YoctoHub-GSM-4G circuit. You can find there two pairs of contacts which enable you to shunt these two buttons.



Sleep and wake up button deviation.

The YoctoHub-GSM-4G includes a switch with which you can disable the sleep mode at the hardware level. This functionality is particularly useful when developing and debugging your project, as well as when updating the firmware.

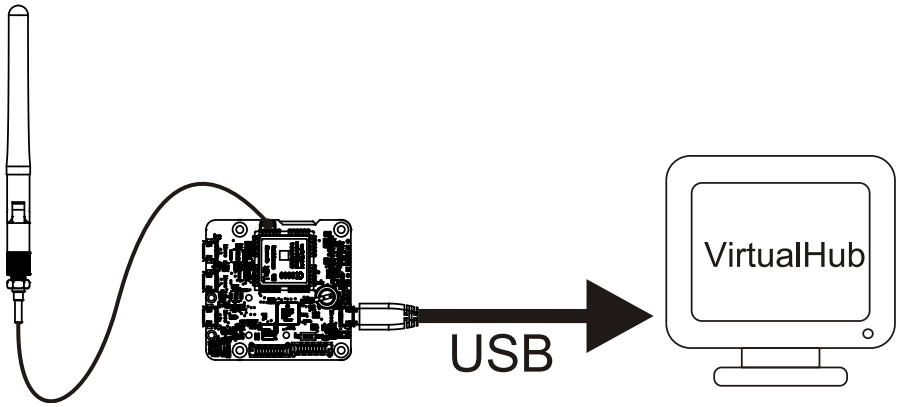
3. First steps

The aim of this chapter is to help you connect and configure your YoctoHub-GSM-4G for the first time.

3.1. Manual configuration

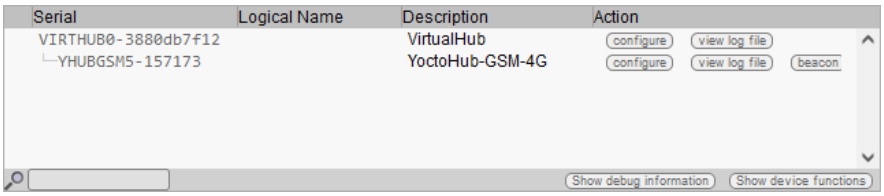
You can configure your YoctoHub-GSM-4G through its USB control port, by using VirtualHub[1].

Run VirtualHub on your preferred computer and connect it to the *power / control port* of the YoctoHub-GSM-4G. You need a USB A-MicroB cable.



Configuration: connecting your YoctoHub-GSM-4G by USB to a computer

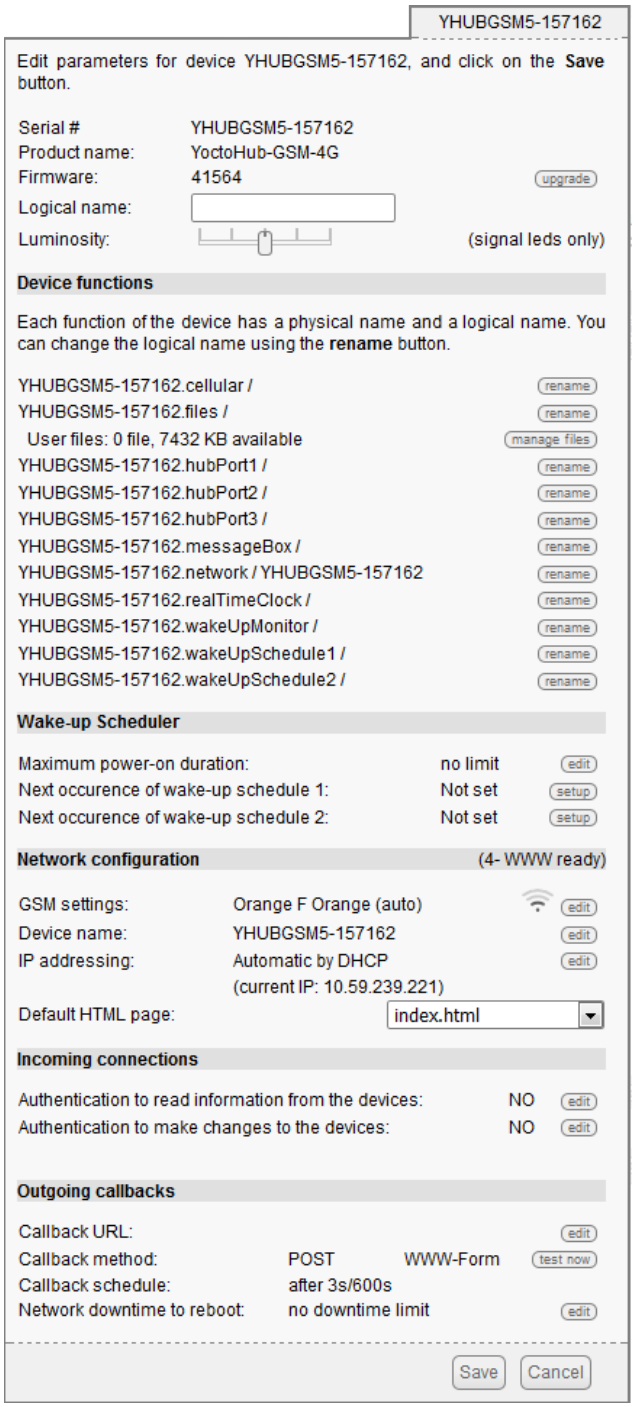
Launch your preferred browser on the URL of your VirtualHub. It usually is *http://127.0.0.1:4444*. You obtain the list of Yoctopuce modules connected by USB, among which your YoctoHub-GSM-4G.



List of Yoctopuce modules connected by USB to your computer, among which your YoctoHub-GSM-4G

Click on the **configure** button corresponding to your YoctoHub-GSM-4G. You obtain the module configuration window. This window contains a **Network configuration** section.

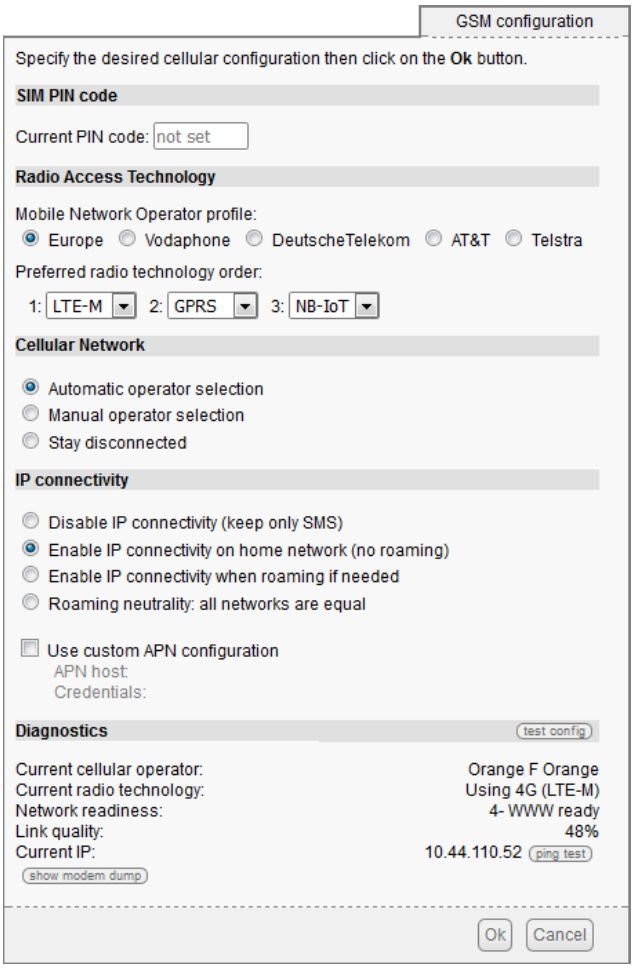
[1] http://www.yoctopuce.com/EN/virtualhub.php



YoctoHub-GSM-4G module configuration window

Connection to the cellular network

The first thing you must do is to configure your YoctoHub-GSM-4G so that it connects itself to your cellular network. To do so, click on the **edit** button corresponding to **GSM configuration** in the **Network settings** section and the cellular network configuration window opens:



Cellular network configuration window

If required, you can then enter the PIN code corresponding to the SIM card inserted in the YoctoHub-GSM-4G. Warning: you usually have only three chances to enter the correct PIN code. If you fail to do so, you will have to reset the SIM card with its PUK code.

As the YoctoHub-GSM-4G can connect to several types of cellular networks on several frequency bands, you must select a radio access technology and an operator profile, which defines the frequency bands to use as well as some more specific communication parameters. The available profiles are:

AT&T
: LTE-M (bands 2, 4, 5 and 12) and (E)GPRS

AT&T 2-4-12
: LTE-M (bands 2, 4 and 12) and (E)GPRS

Europe
: LTE-M (bands 3, 8, 20), NB-IoT (bands 3, 8, 20) and (E)GPRS

DT (Deutsche Telekom)
: LTE-M (bands 3, 8, 20), NB-IoT (bands 3, 8, 20) and (E)GPRS

Vodafone
: LTE-M (bands 3, 8, 20), NB-IoT (bands 3, 8, 20) and (E)GPRS

Orange France
: LTE-M (bands 3, 8, 20), NB-IoT (bands 3, 8, 20) and (E)GPRS

VIVO
: LTE-M (bands 3, 28) and NB-IoT (band 28)

TIM Brasil
: NB-IoT (bands 3 and 28)

Claro Brasil
: LTE-M (bands 3, 28) and NB-IoT (bands 3, 5 and 28)

For 2G, the YoctoHub-GSM-4G work on GSM bands 850 MHz, E-GSM 900 MHz, DCS 1800 MHz and PCS 1900 MHz (quad band), in GPRS and EGPRS (EDGE) modes.

If you plan to use one of the operators named in the profile list, select the corresponding profile. For the rest of Europe, select the generic Europe profile, which covers all other European countries. In addition to the profile choice, you can select your preferred technology (or technologies) to use when connecting, in order to speed-up connection time or maximize data transfer rate. If you only select a single technology, the others are not used at all, so make sure it is available in your environment. If you select multiple technologies, they are tried in the specified order, but it is still possible in some cases that the second choice is selected when it would have been possible to connect to the first choice.

In addition to the choice of radio access technology, you can specify the operator that you want to use. In many cases, the SIM card "knows" the provider it is designed for and you can simply keep the automatic selection. However, near country borders, the card may wish to use a more powerful but foreign signal, more expensive to use. To prevent this, you can select the provider of your local network manually. For the module to be able to connect to an operator on the LTE-M and NB-IoT networks, not only does the SIM card have to authorize it, but the APN must also be correctly configured (as explained below). Otherwise, the module will probably not even be able to attach itself to the operator, even just for the SMS function.

You can also specify in your YoctoHub-GSM-4G the context in which you want to enable the IP connection data transfer. You can either completely disable it if you are only interested in SMS use, or activate it on the SIM card network only, or even allow data use on other networks (roaming). Take care if you activate this latest option, because it can quickly generate significant costs! Please note, however, that in some cases, SMS may only be functional if the connection is functional. In addition, with some SIM cards, SMS may not be available not be available at all, depending on the contract provided by the operator which issued the SIM card.

Depending on your SIM card and on your cellular service provider, you may have to configure an APN (*Access Point Name*), corresponding to the Internet gateway on your cellular network. It is an arbitrary name, assigned by your provider, to which you must sometimes add a user name and a password. It is impossible to list in this user's guide the APN name to be used with each provider. But if you do a search on the Internet with the name of your cellular service provider and the "APN" keyword, you will very easily find the APN name to use as well as the potential user name and password. The apnchanger.org web site contains this information for the main providers in many countries.

When you have entered the network parameters and verified tested them, click on the **Ok** button to close this configuration window and come back to the main configuration window.

Difference between a cellular network and a standard network

Important: The cellular network to which your YoctoHub-GSM-4G is connected is not strictly equivalent to a standard Internet connection. Indeed, the IP address assigned to a cellular modem is almost always a *non routed* IP address. The YoctoHub-GSM-4G sees the whole Internet network, but Internet does not have a public address to contact it. This means that you cannot connect yourself remotely on your YoctoHub-GSM-4G from any computer, by simply typing its IP address in a web browser.

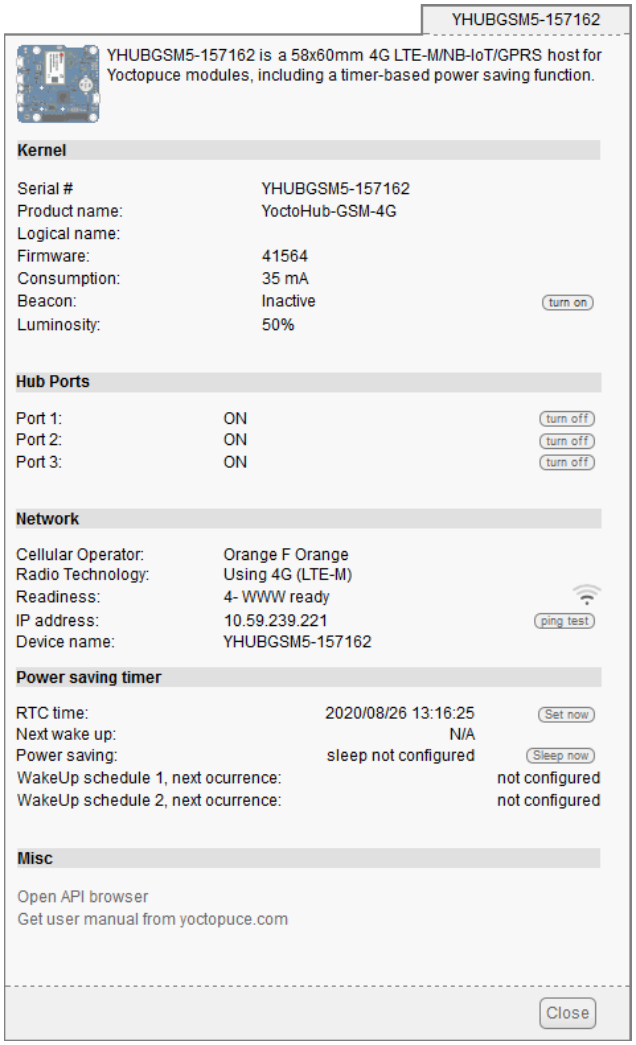
One solution to solve this issue is to obtain from your cellular service provided a SIM card enabling a direct connection through a virtual private network. This option often implies a consequent increase in communication costs.

Another solution is the free *VirtualHub (for web)* tool available on Yoctopuce web site. More information on how to use the YoctoHub-GSM-4G are available in chapter Usage.

3.2. Device state window

It is possible to check the device general state, such as logical name, Network state, Hub port states, and so on. Just go back to the device list.

Click on the serial number corresponding to your YoctoHub-GSM-4G. This opens your module property window:



The YoctoHub-GSM-4G properties

This window contains a section indicating the state of the YoctoHub-GSM-4G network part. You can find there, among other things, the connection type and its current IP address. This section also provides the state of the network connection. Possible states are:

- 0- The module does not find a cellular network. In this case, check that:
  - you are in a location where you can access a 4G (LTE-M or NB-IoT) or 2G network
  - you have inserted a SIM card authorized to connect to the selected network
  - you have configured the PIN code of the SIM card in the YoctoHub-GSM-4G
  - you have not disabled the radio mode (airplane mode)
  - you have configured the Radio Access Technology for the desired service on the YoctoHub-GSM-4G
  - the frequency band used by the network operator for this service is supported by the selected connection profile selected on the YoctoHub-GSM-4G
  - you have connected a cellular antenna compatible with this frequency band
- 1- network exists: a GSM cellular network was detected, but the module has not been accepted yet. If this state persists, check that your SIM is valid and corresponds to the selected cellular service provider.
- 2- network linked: the YoctoHub-GSM-4G did connect to the GSM network, but does not have an IP connection yet. If this state persists, check your APN settings.
- 3- LAN ready: the local network is operational (IP connection with the mobile service provider)
- 4- WWW ready: the module has checked connectivity with Internet by connecting itself to a time server (NTP).

If your YoctoHub-GSM-4G does indeed go into the *WWW Ready* state, it means that your internet cellular connection works correctly. You can then perform the next steps: Connect the wanted Yoctopuce modules (sensors and/or actuators) and configure the interactions with the outside.

3.3. Automated configuration

You can industrialize the YoctoHub-GSM-4G network configuration. You can find in the following chapters of this documentation the description of the programming functions enabling you to read the Ethernet address (MAC address) of a module, and to configure all of its network parameters.

The network configuration functions are also available as command lines, using the `YNetwork` utility software available in the command line programming library [2].

After having set some parameters by software, make sure to call the `saveToFlash()` function to ensure that the new settings are saved permanently in the module flash memory.

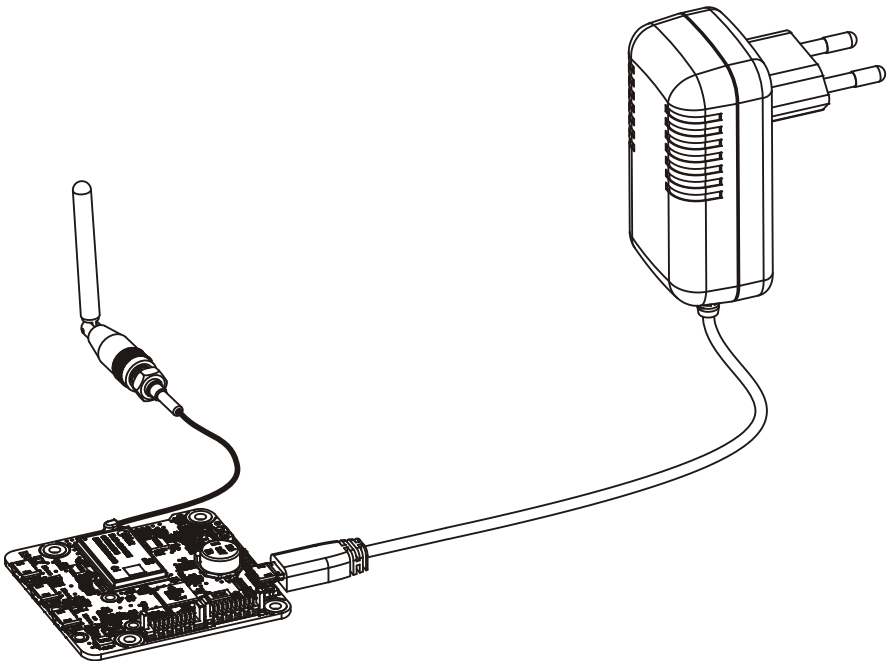
3.4. Connections

Power supply

The YoctoHub-GSM-4G must be powered by the USB control socket.

USB

Simply connect a USB charger in the *power / control port* port, but make sure that the charger provides enough electric power. The YoctoHub-GSM-4G consumes about 50mA, to which you must add the power consumption of each submodule. The YoctoHub-GSM-4G is designed to manage a maximum of 2A. Therefore, we recommend a USB charger able to deliver at least 2A. Moreover, you must make sure that the total power consumption of the set "hub + submodules" does not go above this limit.

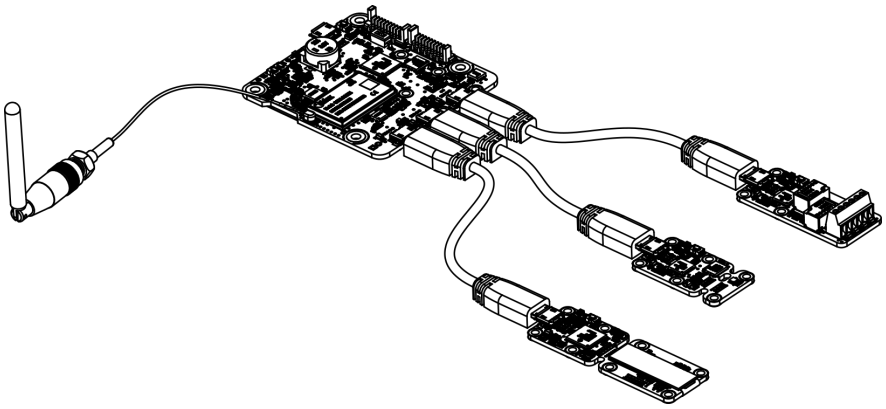


The YoctoHub-GSM-4G can be powered by a regular USB charger

Sub-modules

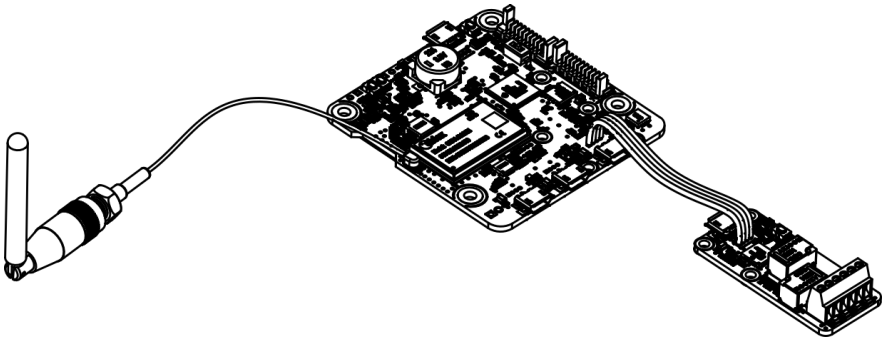
The YoctoHub-GSM-4G is able to drive all the Yoctopuce modules of the *Yocto* range. These modules can be directly connected to the down ports. They are automatically detected. For this, you need Micro-B Micro-B USB cables. Whether you use OTG cables or not does not matter.

[2] http://www.yoctopuce.com/EN/libraries.php



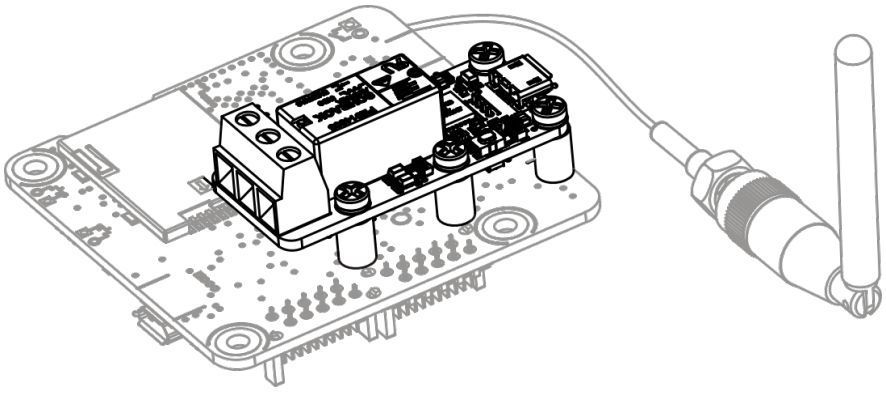
Connecting sub-modules with USB cables

Alternatively, you can connect your modules by directly soldering electric cables between the YoctoHub-GSM-4G and its sub-modules. Indeed, all the Yoctopuce modules have contacts designed for direct cabling. We recommend you to use solid coper ribbon cables, with a 1.27mm pitch. Solid copper ribbon cable is less supple than threaded cable but easier to solder. Pay particular attention to polarity: the YoctoHub-GSM-4G, like all modules in the Yoctopuce range, is not protected against polarity inversion. Such an inversion would likely destroy your devices. Make sure the positions of the square contacts on both sides of the cable correspond.



Sub-module connection with ribbon cable

The YoctoHub-GSM-4G is designed so that you can fix a single width module directly on top of it. To do so, you need screws, spacers[3], and a 1.27mm pitch connector[4]. You can thus transform your USB Yoctopuce module into a network module while keeping a very compact format.



Fixing a module directly on the hub

Beware, the YoctoHub-GSM-4G is designed to drive only Yoctopuce modules. Indeed, the protocol used between the YoctoHub-GSM-4G and the sub-modules is not USB but a much lighter proprietary protocol. If, by chance, you connect a device other than a Yoctopuce module on one of the YoctoHub-GSM-4G down ports, this port is automatically disabled to prevent damages to the device.

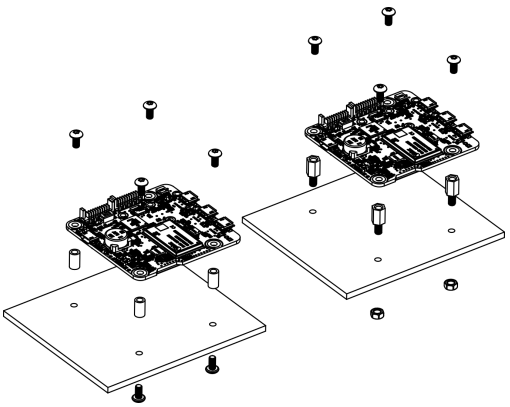
[3] http://www.yoctopuce.com/EN/products/accessories-and-connectors/fix-2-5mm
[4] http://www.yoctopuce.com/EN/products/accessories-and-connectors/board2board-127

4. Assembly

This chapter provides important information regarding the use of the YoctoHub-GSM-4G module in real-world situations. Make sure to read it carefully before going too far into your project if you want to avoid pitfalls.

4.1. Fixing

While developing your project, you can simply let the hub hang at the end of its cable. Check only that it does not come in contact with any conducting material (such as your tools). When your project is almost at an end, you need to find a way for your modules to stop moving around.



Examples of assembly on supports

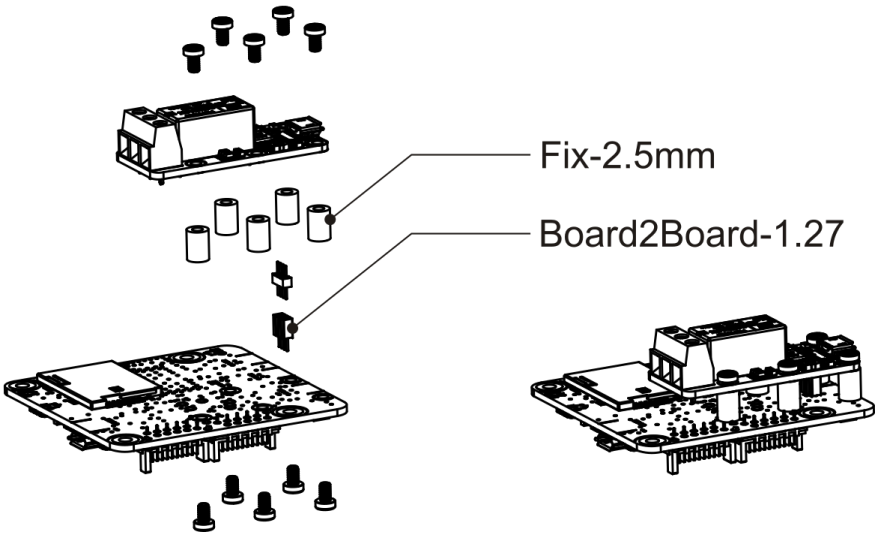
The YoctoHub-GSM-4G module contains 3mm assembly holes. You can use these holes for screws. The screw head diameter must not be larger than 8mm or the heads will damage the module circuits.

Make sure that the lower surface of the module is not in contact with the support. We recommend using spacers. You can fix the module in any position that suits you: however be aware that the YoctoHub-GSM-4G electronic components, in particular the network part, generate heat. You must not let this heat accumulate.

4.2. Fixing a sub-module

The YoctoHub-GSM-4G is designed so that you can screw a single width module directly on top of it. By single width, we mean modules with a 20mm width. All the single width modules have their 5 assembly holes and the USB socket in the same position. The sub-module can be assembled with screws and spacers. At the back of the YoctoHub-GSM-4G and sub-module USB connectors, there

are a set of 4 contacts enabling you to easily perform an electrical connection between the hub and the sub-module. If you do not feel sufficiently at ease with a soldering iron, you can also use a simple Micro-B Micro-B USB cable, OTG or not.



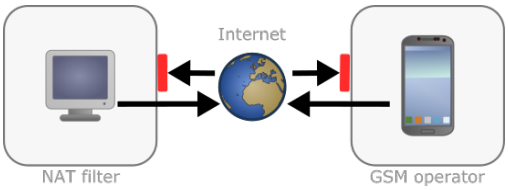
Fixing a module directly on the hub

Make sure to mount your module on the designed side, as illustrated above. The module 5 holes must correspond to the YoctoHub-GSM-4G 5 holes, and the square contact on the module must be connected to the square contact on the YoctoHub-GSM-4G down port. If you assemble a module on the other side or in another way, the connector polarity will be inverted and you risk to permanently damage your equipment.

All the accessories necessary to fix a module on your YoctoHub-GSM-4G are relatively usual. You can find them on the Yoctopuce web site, as on most web sites selling electronic equipment. However, beware: the head of the screws used to assemble the sub-module must have a maximum head diameter of 4.5mm, otherwise they could damage the electronic components.

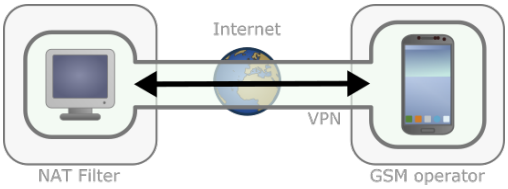
5. Usage

The YoctoHub-GSM-4G is the technical equivalent of a YoctoHub-Wireless for the GSM network. But a GSM network is far from being the equivalent of a Wifi network. The main difference resides at the level of addressing. Phone companies never assign public IP addresses to the terminals they manage, so to speak. Each terminal is located behind a type of NAT preventing it to be directly contacted from the outside. In plain text, a smart phone can access Internet, but it cannot be contacted directly from Internet, at least not with a standard subscription. The mechanism is similar to that of the NAT filter preventing anyone from directly accessing the machines you have at home. But here you cannot even add a port redirection on the router. This limitations applies to the YoctoHub-GSM-4G as well: with a standard mobile phone subscription, you are not likely to be able to directly contact a YoctoHub-GSM from your home computer.



Usually, the companies limit direct access from Internet to their terminals

Nevertheless, some mobile phone companies offer special subscriptions including a VPN with direct access to a fleet of terminals. With this type of subscription, you can directly contact the YoctoHub-GSM-4G as if it were a simple YoctoHub-Ethernet or YoctoHub-Wireless connected to your local network. But you must be aware that this type of subscription is usually reserved for large infrastructures, which puts them out of reach of private individuals. If you want to ask more details to your favorite phone company, the magic word to obtain the right person is "M2M", for machine-to-machine.



Phone companies sometimes offer a VPN service, enabling direct access to the terminals

5.1. Standard use, without VPN

General principles

If you cannot afford a direct access to your YoctoHub-GSM-4G, you can still use it. The YoctoHub-GSM-4G has the same callback system as the other Yoctopuce hubs. Thus, it can automatically post values from its sensors on several cloud services, such as Emoncms, Valarm, and so on. But above all, the callback API, enabling a PHP, Java or node.js server to drive the hub remotely, is also available.

To configure callbacks for your YoctoHub-GSM-4G, you must connect it by USB to a machine running VirtualHub and access its configuration page. In the **Outgoing callbacks** section of this page, click on **Edit** at the right of the **Callback URL** line.

Details of the different type of callbaks are described in chapter 7, "Outgoing Connections.

Special case of HTTP callback to VirtualHub (for Web)

VirtualHub (for Web) provides remote access to your Yoctopuce modules via the Internet. This software must be installed on a Web server, naturally before creating a callback pointing to it. As the installation of VirtualHub (for Web) is not the current topic, you can find installation procedure on our blog[1].

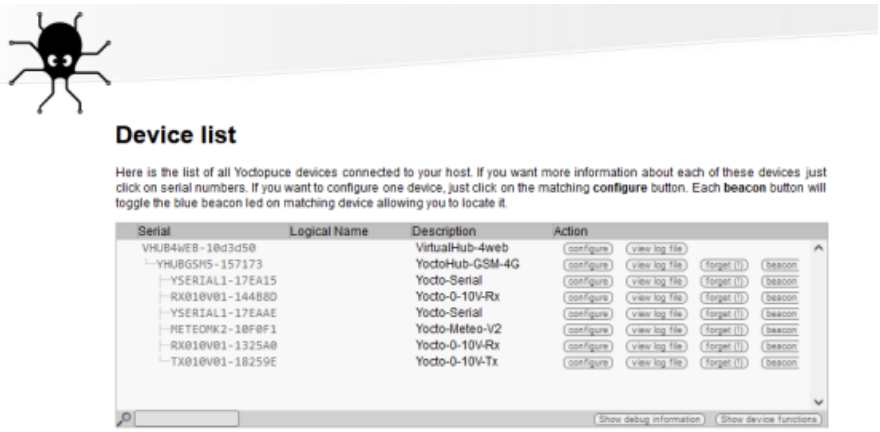
To connect a Yoctopuce system to VirtualHub (for Web), you just need to configure an HTTP callback on the YoctoHub-GSM-4G that controls it pointing to this Web server. During each callback, the VirtualHub (for Web) automatically downloads all the necessary information from the YoctoHub-GSM-4G, including the configuration of the modules, the latest sensor data, the log messages from the modules, and even the files needed for the user interface.

To configure HTTP callbacks on the YoctoHub-GSM-4G, start by making sure that the firmware of your YoctoHub-GSM-4G is at least version 51900, and that the modules connected to it are at least at least version 50730.

You can then configure your *outgoing callbacks* as follow:

- Type of callback: Yocto-API callback
- Callback URL: *_virtualhub-4web_instance_path_*/HTTPCallback
- Type of security: MD5 signature
- Password: *the same as you will set on the server for incoming callbacks*

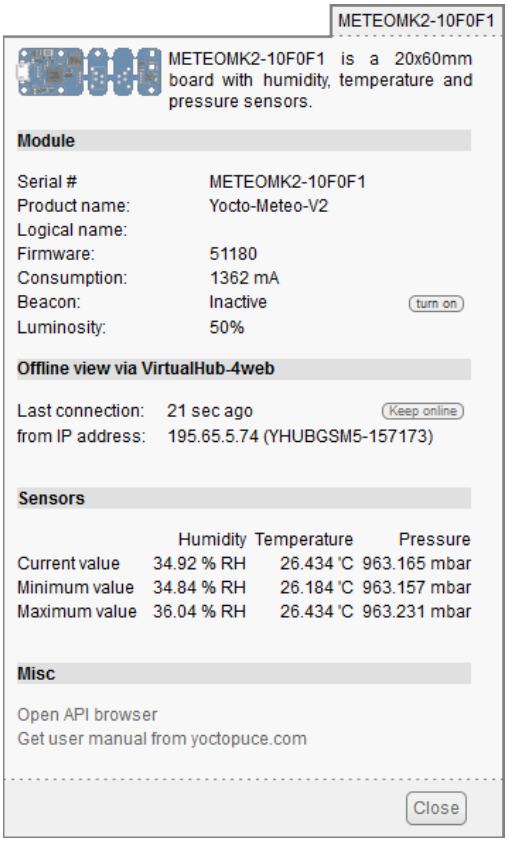
When you establish a connection with a web browser to the VirtualHub (for Web) server, you get an interface very similar to that of a regular YoctoHub or VirtualHub.



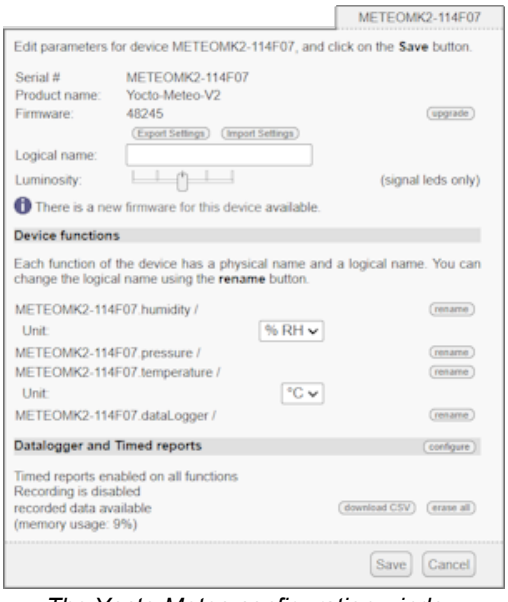
The VirtualHub (for Web) interface

[1] https://www.yoctopuce.com/EN/blog.php

You can go and view the status of each module, as it was the last time the module connected to the server. You can even change the module settings: the changes are applied the next time the module connects to the server.



The Yocto-Meteo details window



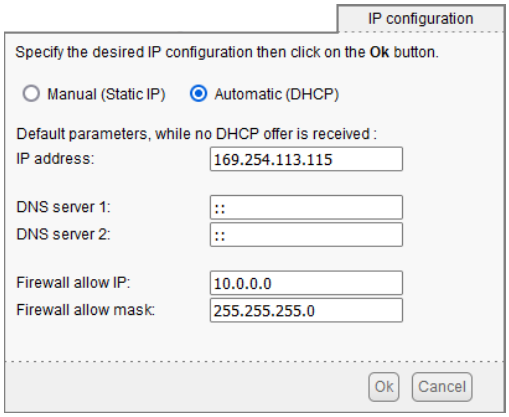
The Yocto-Meteo configuration window

When you use the VirtualHub (for Web), don't forget that the application of the settings is delayed to the next HTTP callback, which necessarily implies a slightly different behavior than a direct connection: for example, if you change an attribute and read back its value directly, you get the previous value until the time when the setting is actually applied to the module and transmitted back to VirtualHub (for Web).

5.2. Use with a SIM enabling VPN access

Some cellular service providers can conditionally provide a cellular connection with an internet link to a private address range, which is dedicated to you. This type of connection, dedicated to the *machine-to-machine* services, is generally restricted to businesses. It allows you to connect yourself remotely (through a virtual private network) to your YoctoHub-GSM-4G, which is otherwise impossible because each cellular phone is normally implicitly isolated behind a NAT filter.

If you do have such a connection, you can configure which IP address must be assigned to the YoctoHub-GSM-4G, and which IP address is authorized to contact it (*firewall* function). To do so, click on the **edit** opposite to the **IP addressing** line in the YoctoHub-GSM-4G configuration window. It is essential to configure these parameters correctly to be able to contact your module through its IP address. Otherwise, the firewall blocks any incoming connection.

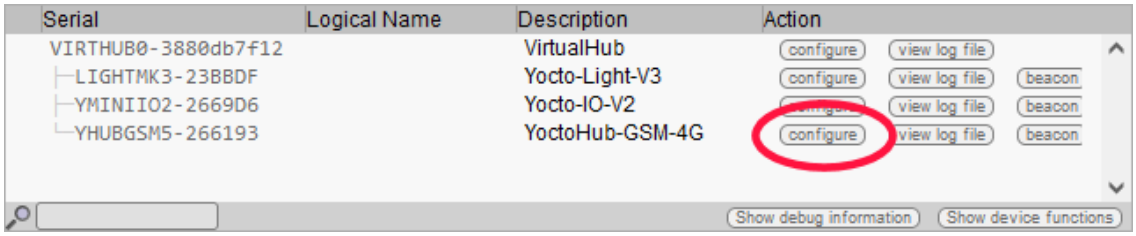


Editing the IP addressing parameters to enable direct connection to the YoctoHub-GSM-4G

When these settings are properly configured, you can access your YoctoHub-GSM-4G like any other YoctoHub.

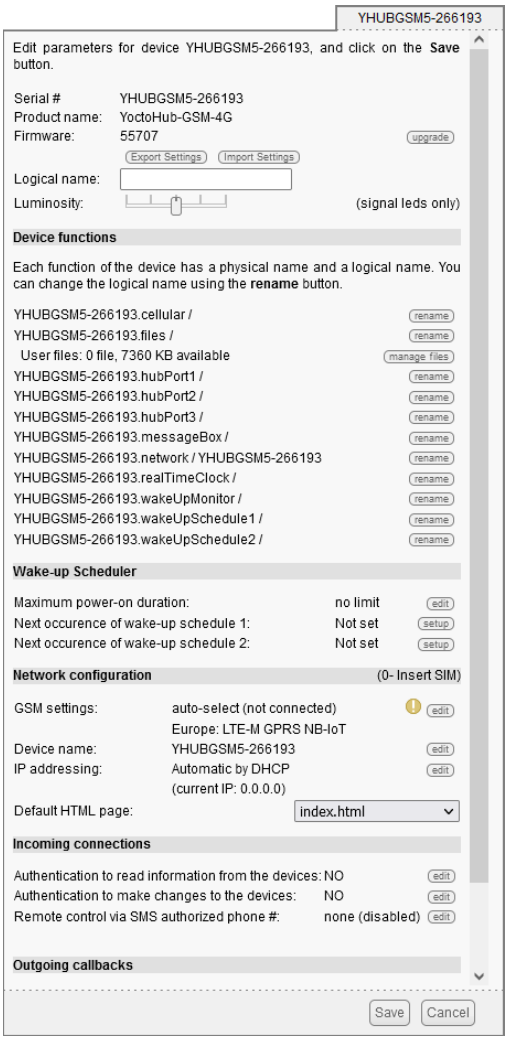
Access control

YoctoHub-GSM-4G is able to perform access control to protect your Yoctopuce devices. Click on the **Configure** button on the line matching YoctoHub-GSM-4G in the user interface.



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

This opens the configuration window for YoctoHub-GSM-4G



YoctoHub-GSM-4G configuration window

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

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 YoctoHub-GSM-4G as they want to.

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 YoctoHub-GSM-4G. *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.

Access control and API
Warning: access control has an impact on Yoctopuce API behavior when trying to connect to YoctoHub-GSM-4G with access control enabled. With Yoctopuce API, access control is handled at the `RegisterHub()` level. You need to provide the YoctoHub-GSM-4G address as follow: *login:password@adresse:port*, here is an exemple:

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

5.3. Data consumption

A large portion of the costs for a remote monitoring installation using the cellular network is linked to communication costs that the provider bills. It is therefore important to correctly estimate the amount of data transmitted in order to budget the system, to select the most appropriate subscription, and make the best technical choices. Here are a few elements which can help you to do so.

To give a concrete basis to this discussion, let's take as an example a small system made of a YoctoHub-GSM-4G, providing connectivity to three Yoctopuce modules:

- A Yocto-PT100 to measure the temperature
- A Yocto-4-20mA-Rx to measure two specific physical quantities
- A Yocto-GPS to locate the system (latitude, longitude, altitude, speed)

All together, there are therefore seven measures that we want to transmit.

If we want to monitor these seven measures every minute, we could imagine that, with an ideal system, this costs for example 4 bytes per measure, so roughly 30 bytes per minute.

Unfortunately, Internet is not that efficient for small quantities of data and we must add a good number of auxiliary data needed by the system for it to work.

We must minimally encapsulate the data in TCP/IP network packages, in order for them to reach the appropriate server, which adds at least 50 bytes per transmission. The exchange protocol used by the server can add a few hundreds of bytes per transmission. It is the case for example with the HTTP protocol, which uses very verbose text headers. The data themselves are often encoded in a more explicit way than a binary transmission, for example in JSON. This easily multiplies by 3 or 4 the quantity of data to be transmitted. Checking the continuity of the GSM connection in order to post the data without delay can also cost a few tens of bytes per minute, according the desired quality level.

This may seem somewhat vague, but fortunately, to make everything more precise, the YoctoHub-GSM-4G contains a transmitted and received byte counter. We don't guarantee that the number billed by the provider is exactly the same, as rounding can apply, but this allows us to have a more precise idea of data usage. To retrieve the counter values, you can simply call the following methods:

```
cellular.get_dataSent()
cellular.get_dataReceived()
```

If your YoctoHub-GSM-4G is connected by a USB cable to a computer, you can write a short script to retrieve these two values every minute and perform a few statistic computations. This is what we have done here to document the data usage in a few classic scenarios.

Consommation de base

When a YoctoHub-GSM-4G is connected to the TCP/IP network, its default behavior is to check its connectivity every minute, in order to initiate a new connection in case of problem. This behavior has a cost of 60 bytes per minute, in upload and download. If need be, you can space out the checking packets with the method

```
cellular.set_pingInterval(n_sec)
```

Moreover, the hub connects to an NTP server every 10 minutes to potentially adjust the real time clock, used to time stamp the measures. This represents 150 bytes per connection, in both directions.

Multiply by 60 minutes, 24 hours and 31 days, and you have a 6.7 MB/month consumption to maintain connectivity and time stamping 24/7, regardless of the number of sensors connected. You can naturally proportionally reduce this volume by putting the hub into sleep mode with the internal clock during some hours of the night, or by waking it up periodically only.

Sending the measures by simple HTTP callback

If you add a minimalist transmission of the current value of each sensor every minute with a simple HTTP GET request, with a CSV coding, you rise up to 0.6 KB for upload and 0.5 KB for download per minute, that is 50 MB/month for day and night operation. If you select the fancier alternative of transmission by HTTP POST with WWW-urlcoded form, you can even reach 70 MB/month.

These numbers correspond to our little test system with seven measured values. If you use more sensors, the data usage will raise a bit, but not proportionally since a large part of the data usage comes from the protocols overhead, which is independent of the number of sensors.

Connecting by Yocto-API HTTP callback

Using the Yocto-API HTTP callback protocol costs more. Indeed, then it's not only one value per sensor which is transmitted every minute, but the current state of each sensor, including minimal and maximal values, the measuring unit, the accuracy, the settings, uptime, and so on. And naturally, the advantage of this protocol is that it allows you to act on the modules as well, to change their configuration remotely, or trigger an actuator.

In its basic version, the Yocto-API HTTP protocol has a costs of 12.1 KM/minute for upload and 0.7 KB/minute for download, so a total of 580 MB/month. Fortunately, if we enable the optimization introduced for the PHP library, we can cut down the traffic to 300 MB/month.

Connexion par callback HTTP Yocto-API

L'utilisation du protocole de callback Yocto-API coûte plus cher. En effet, ce n'est pas qu'une valeur par capteur qui est transmise chaque minute, mais l'état courant de chaque capteur, y compris les valeurs minimales et maximales rencontrées, l'unité de mesure, la précision, les réglages, l'uptime, etc. Et bien sûr, l'intérêt de ce protocole est qu'il permet aussi d'agir sur les modules pour changer leur configuration à distance, ou actionner un actuateur.

Dans sa version de base, le protocole Yocto-API en http coûte pour notre système 12.1KB/minute en upload, et 0.7KB/minute en download, soit un total de 580 MB/mois. Hereusement, si on active l'optimisation introduite pour la librairie PHP, on peut ramener le trafic à 300 MB/mois.

Connecting by Yocto-API WebSocket callback

Using a persistent WebSocket channel avoids repeating the HTTP headers for each request. However, maintaining this channel open permanently, with a continuous publication of the measures, also has a non-negligible cost.

Without further precaution, a WebSocket permanent connection for our system costs between 3 and 5 KB/minute in upload, between 2 and 3 KB/minute in download, for a total of about 350 MB/month. As each new measure is transmitted instantaneously, the volume of transmitted data varies depending on measure variability. However, this allows you to generate as many simple HTTP callbacks to third-party services that you want, without additional GSM traffic.

In the same way, if you want to forward a WebSocket callback to an Yocto-API HTTP callback with VirtualHub (for web), you must take into account loading the full state of each sensor through the WebSocket channel, which represents an additional load. For a Yocto-API HTTP callback per

minute, we reach 500 MB/month. It's almost double from an optimized Yocto-API callback performed directly from the hub, but it has the advantage to allow you to take full control of the hub remotely.

Fortunately, we can do better than that. Indeed, the 350 MB/month used to transmit instantaneously and 24/7 the current value of each sensor are not necessarily useful, especially if you want to treat the data only once every minute. You simply have to disable on all the sensors the sending of instantaneous values and to use instead measures with a timed interval.

```
sensor.muteValueCallbacks()
sensor.set_reportFrequency("1/m")
```

Thanks to using time interval measures, we bring back the traffic to 1.3KB/min for download and 0.9KB/min for upload, so 100 MB/month only with a permanent WebSocket connection. Moreover, for each time interval, you receive not only the average value but also the min/max during the interval. And if there is a temporary GSM connectivity failure, if you have enabled the data loggers on the modules, you can precisely retrieve the missing data, as they are recorded simultaneously with the same granularity.

This solution requires however that you code your solution by exploiting the Yoctopuce library and its WebSocket callback API. If you are looking for something easier to set up, using VirtualHub (for web) and a simple HTTP callback to a third party service such as EmonCMS, you can also simply configure the sensors to propagate the average value per minute, rather than the instantaneous value.

```
sensor.set_advMode(YSensor.ADVMODE_PERIOD_AVG);
```

This probably enables you to benefit from the possibility of a remote connection to your hub, while consuming less than 100 MB/month.

Planning your data consumption carefully

GSM data usage differences between the different options are really significant. It's therefore worth it to take them into account as early as the design stage of the software solution that you are going to implement to read your sensors.

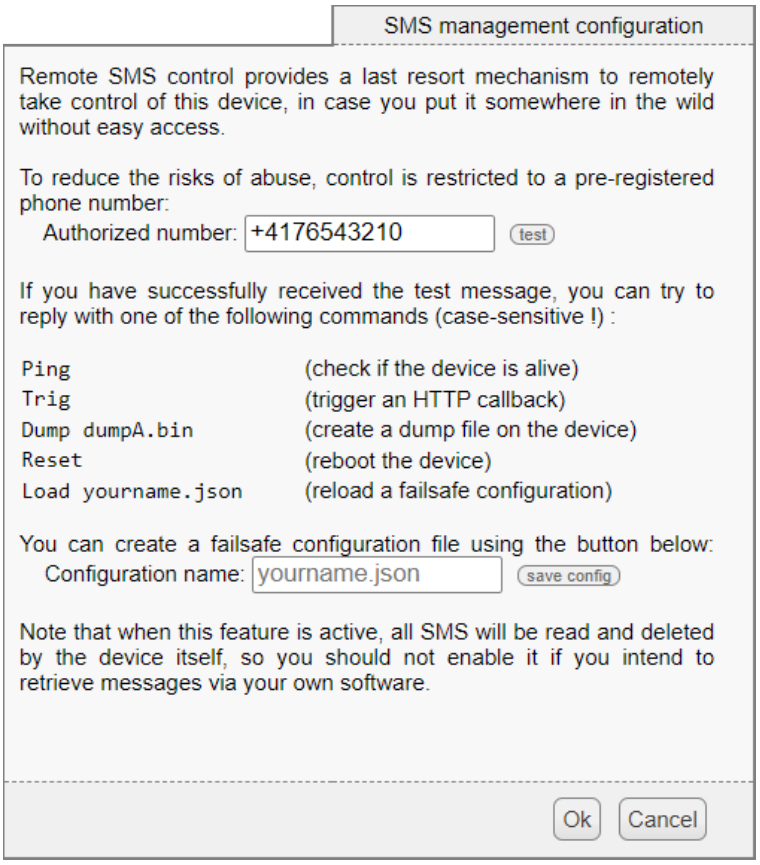
Remember as well that you can drastically reduce the data usage by spacing out the measures and/or disconnecting the hub from the network by putting it to sleep.

5.4. SMS access

The YoctoHub-GSM-4Gcan also receive control SMS. This way, if it can no longer connect to your server, but can still reach a GSM network, you may have a chance of getting it back.

Security

For security reasons, only SMS messages from a pre-configured number are taken into account. All others only produce a log message and are automatically deleted. This new setting can be found in the **Incoming connections** section of the configuration:



SMS control configuration interface

This window includes a button for sending a test message to the chosen number, which is doubly interesting. On the one hand, it lets you check that the SIM card supports sending SMS - not always the case with IoT packages. Secondly, it's the easiest way to find out your YoctoHub-GSM-4G's phone number, as the YoctoHub-GSM-4G itself has no way of informing you of it: in the vast majority of cases, the number is not stored on the SIM card, and the 4G standard has no command for requesting it from the operator.

With this test message, you can simply reply to the message on your phone with one of the commands below to check that your YoctoHub-GSM-4G is receiving your messages.

Recognized messages

SMS control only allows the following important operations. Here are the recognized commands:

Ping simply asks the YoctoHub-GSM-4G to confirm the reception of the message by SMS. If the YoctoHub-GSM-4G is configured to wake up periodically, it replies when it next wakes up - SMS messages are not received during deep sleep. This lets you know whether the YoctoHub-GSM-4G is still alive and able to connect to the cellular network.

Trig asks the YoctoHub-GSM-4G to trigger an HTTP callback as soon as possible. In the event that you have set a too long interval between callbacks to allow callbacks, or programmed a sleep time too soon to allow the hub to callback, this may enable you to regain control of your YoctoHub-GSM-4G.

Dump xxx.bin asks the YoctoHub-GSM-4G to *dump* its current state into the `xxx.bin` file on flash memory, for further investigation. In cases where the YoctoHub-GSM-4G is no longer able to establish IP connectivity and you need to reboot it or even intervene on site, this dump may eventually enable Yoctopuce support to diagnose the cause of the problem.

Reset restarts the YoctoHub-GSM-4G.

Load filename.json asks the YoctoHub-GSM-4G to reload all its configuration in one go from the local `filename.json` file. To use this function, you must of course first have created this

configuration backup file on the YoctoHub-GSM-4G, using the button available on this same window, or using the *get_allSettings* function in the Yoctopuce library, for example. You can then switch your YoctoHub-GSM-4G to a predefined *backup configuration* in the event of a problem.

Note that if you need to remotely change your YoctoHub-GSM-4G's callback configuration or cell configuration, the safest way is to

1. predefine the new configuration on another YoctoHub-GSM-4G, so you can test it beforehand
2. then upload the configuration file to the remote YoctoHub-GSM-4G using the file manager (*manage files* function on VirtualHub for Web)
3. then activate the new configuration in one go with the SMS command **Load myconfig.json**

Indeed, if you try to change this kind of parameter directly via the VirtualHub (for Web) interface, the YoctoHub-GSM-4G may cut the callback as soon as the first setting is changed, and not complete the reconfiguration. Using the **Load** command ensures that all settings are applied in one go.

SMS and IoT portals

As mentioned above, some IoT SIMs do not offer direct SMS access to public cellular networks. However, it may be possible to send and receive SMS messages using the IoT SIM provider's web portal, which ultimately offers the same possibilities for using SMS control. Ask your IoT operator for details specific to your SIM.

If you're not sure which portal sender number to authorize in your YoctoHub-GSM-4G's configuration interface, simply give it a try by authorizing a "random" number. When the portal sends your first *ping* SMS, you'll see in the YoctoHub logs the phone number from which the "rejected" message originated.

6. Outgoing connections

YoctoHub-GSM-4G is able to connect to external services to communicate the status of the modules connected to it.

YoctoHub-GSM-4G knows how to post its data in the format accepted by some third-party cloud services, such as

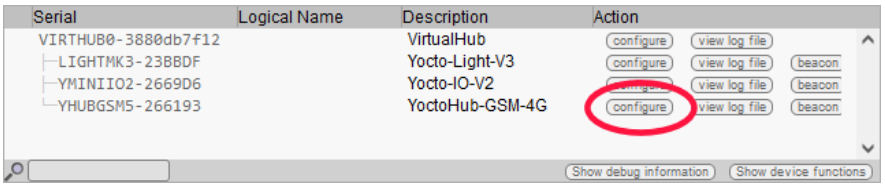
- Emoncms
- InfluxDB (versions 1.0 et 2.0)
- PRTG
- Valarm.net

YoctoHub-GSM-4G 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:

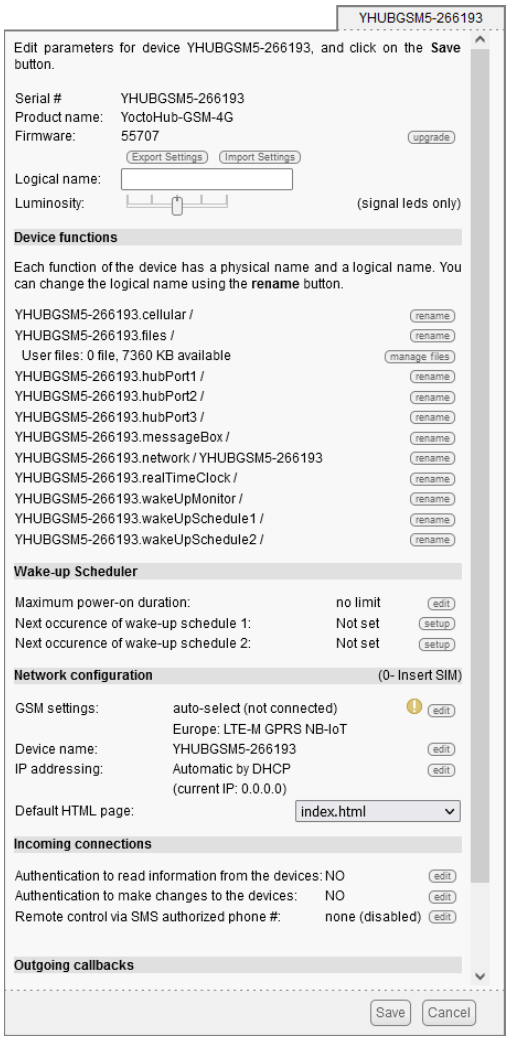
- MQTT
- Yocto-API

6.1. Configuration

To use this feature, just click on the **configure** button located on the line matching YoctoHub-GSM-4G on the interface. Then look for the **Outgoing calback** section and click on the **edit** button.



*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 YoctoHub-GSM-4G interacts with an external web site. Several interaction types are at your disposal.

6.2. HTTP Callbacks to 3rd-party services

YoctoHub-GSM-4G 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 YoctoHub-GSM-4G.

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 YoctoHub-GSM-4G.

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, YoctoHub-GSM-4G 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 YoctoHub-GSM-4G (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 [1].

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

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. YoctoHub-GSM-4G 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.

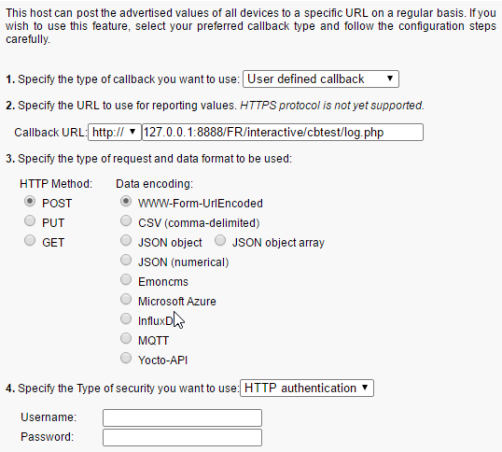
WebSockets provide access to a number of advanced features of Yoctopuce API that were unavailable in HTTP callback mode:

- 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 YoctoHub-GSM-4G sends information to the server. Note that only HTTP protocol is supported (no HTTPS).

[1] https://www.yoctopuce.com/EN/article/new-a-virtualhub-that-works-through-the-web



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. YoctoHub-GSM-4G knows how to handle standard HTTP authentication schemes: simply provide the user and 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 YoctoHub-GSM-4G (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": "YoctoHub-GSM-4G" }
```

Customization file for Emoncms

The basic format used by YoctoHub-GSM-4G 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 YoctoHub-GSM-4G 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 YoctoHub-GSM-4G 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 YoctoHub-GSM-4G 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 YoctoHub-GSM-4G 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 YoctoHub-GSM-4G 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 YoctoHub-GSM-4G 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

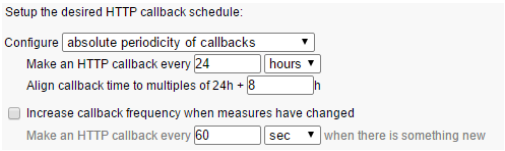
The callback scheduling section is present for all types of callbacks. It is the last section containing fields to be filled.

A first callback is always made just a few seconds after YoctoHub-GSM-4G 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, YoctoHub-GSM-4G 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

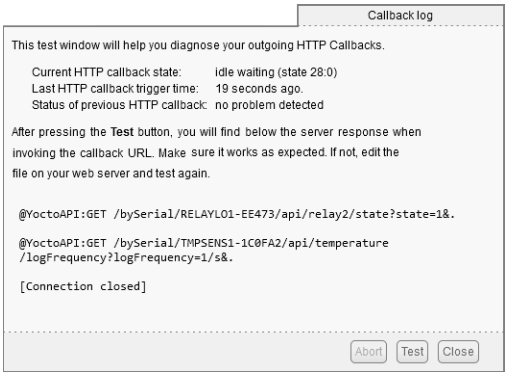
You can explicitly select if you want the callback frequency to vary when no new measure is detected. This enables you to lower the minimal transmission frequency to lower the quantity of data sent over the network if nothing is happening.

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, YoctoHub-GSM-4G 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, YoctoHub-GSM-4G occasionally tries to establish outgoing connections. These connections are the following:

- DNS connection: whenever YoctoHub-GSM-4G needs to connect to an external server, it makes a connection to a DNS server to obtain the IP address corresponding to the name of the server it is about to contact. If these DNS connections cannot be made, the default NTP server cannot be contacted and HTTP callbacks only work if they are defined by an explicit IP address.
- NTP connection: to keep its internal clock synchronized, the hub regularly connects to an NTP server. You can change the address of this server by modifying the "*ntpServer*" option in the "*network*" function. If these NTP connections cannot be made, the hub's internal clock starts to drift.
- Installing Yocto-Visualization (for web): When the user initiates the installation of Yocto-Visualization (for web) from the YoctoHub-GSM-4G interface, the hub automatically downloads the most recent installation file from www.yoctopuce.com
- Version test: every time the property or configuration window of a module is opened, YoctoHub-GSM-4G makes a request to 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, YoctoHub-GSM-4G makes a request to www.yoctopuce.com to retrieve the most recent firmware. This connection saves the user from having to download the latest firmware manually.

Note that the last two connections are in fact established by the web browser displaying the YoctoHub-GSM-4G user interface. Moreover, these connections are purely optional, if they cannot be established, the application continues to function normally.

Here is a table summarizing the exact parameters of these connections:

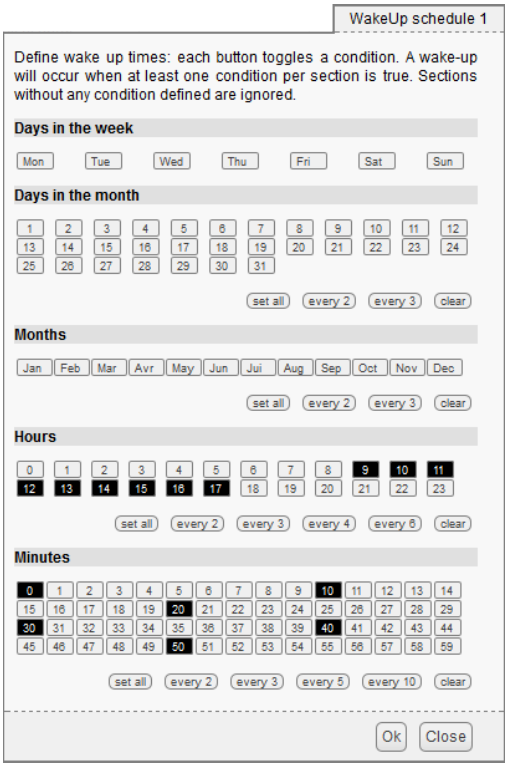
Justification	Protocol Port	Target address	How to disable
Name resolution	DNS UDP 53	configured DNS server	configuring the IP address of the NTP server and defining callbacks by IP address
Updating the clock	NTP UDP 123	1.yoctopuce.pool.ntp.org or configured server	setting the NTP server to 255.255.255.255
Installing YV4web	HTTP TCP 80	www.yoctopuce.com	do not use the quick link on the web interface
Version test	HTTPS TCP 443	www.yoctopuce.com	do not use the YoctoHub-GSM-4G web interface
Downloading firmware	HTTPS TCP 443	www.yoctopuce.com	do not update through the web interface

7. Sleep mode

The YoctoHub-GSM-4G includes a real time clock (RTC) powered by a super capacitor. This capacitor charges itself automatically when the module is powered. But it is able to keep time without any power for several days. This RTC is used to drive a sleep and wake up system to save power. You can configure the sleep system manually through an interface or drive it through software.

7.1. Manual configuration of the wake ups

You can manually configure the wake up conditions by connecting yourself on the interface of the YoctoHub-GSM-4G. In the **Wake-up scheduler** section of the main configuration window, click on the setup button corresponding to one of the "wakeup-schedule". This opens a window enabling you to schedule more or less regular wake ups. Select the boxes corresponding to the wanted occurrences. Empty sections are ignored.



Wake up configuration window: here every 10 minutes between 9h and 17h.

Likewise, you can configure directly in the YoctoHub-GSM-4G interface the maximal wake up duration, after which the module automatically goes back to sleep. If your YoctoHub-GSM-4G is running on batteries, this ensures they do not empty even if no explicit sleep command is received.

7.2. Configuring the wake up system by software

At the programming interface level, the wake up system is implemented with two types of functions: the *wakeUpMonitor* function and the *wakeUpSchedule* function.

wakeUpMonitor

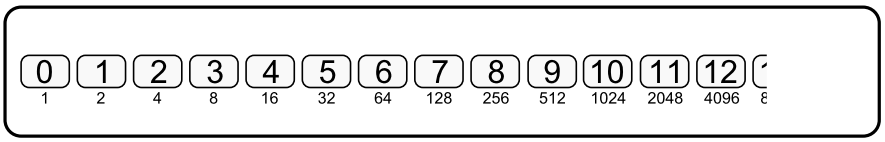
The *wakeUpMonitor* function manages wake ups and sleep periods, proper. It provides all the instant managing functionalities : instant wake up, instant sleep, computing the date of the next wake up, and so on...

The *wakeUpMonitor* function enables you also to define the maximum duration during which the YoctoHub-GSM-4G stays awake before automatically going back to sleep.

wakeUpSchedule

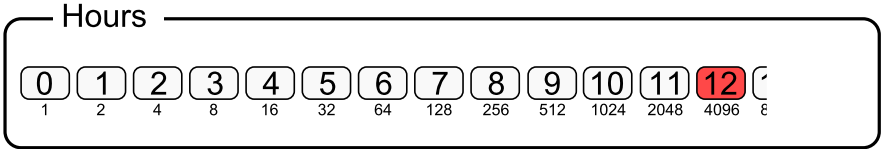
The *wakeUpSchedule* function enables you to program a wake up condition followed by a possible sleep. It includes five variables enabling you to define correspondences on minutes, hours, days of the week, days of the month, and months. These variables are integers where each bit defines a correspondence. Schematically, each set of minutes, hours, and days is represented as a set of boxes with each a coefficient which is a power of two, exactly like in the corresponding interface of the YoctoHub-GSM-4G.

For example, bit 0 for the hours corresponds to hour zero, bit 1 corresponds to hour 1, bit 2 to hour 2, and so on.



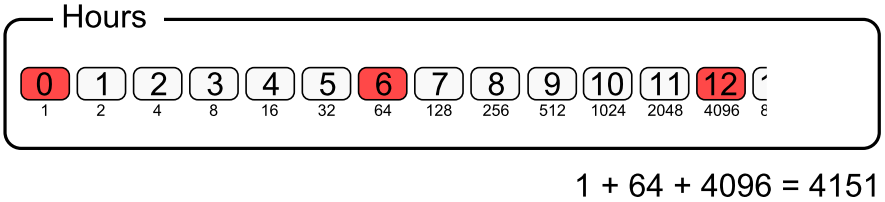
To each box is assigned a power of two

Thus, to program the YoctoHub-GSM-4G for it to wake up every day at noon, you must set bit 12 to 1, which corresponds to the value 2^12 = 4096.



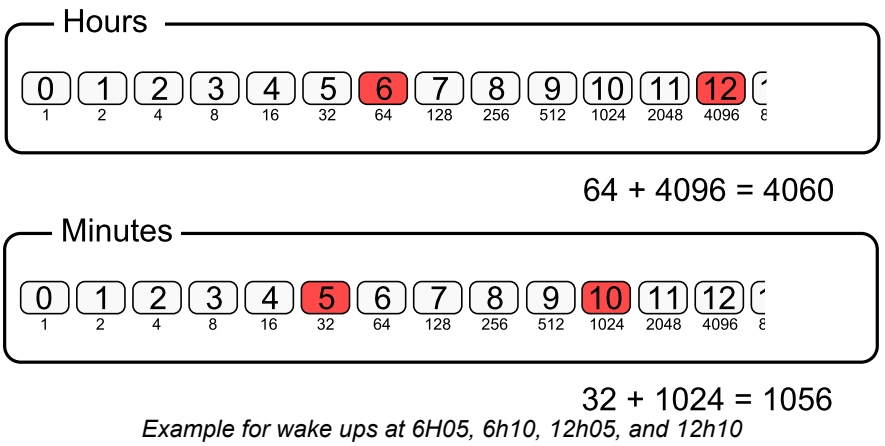
Example for a wake up at 12h

For the module to wake up at 0 hour, 6 hours, and 12 hours, you must set the 0, 6, and 12 bits to 1, which corresponds to the value 2^0 +2^6 +2^12 = 1 + 64 + 4096 = 4161



Example for wake ups at 0, 6, and 12h

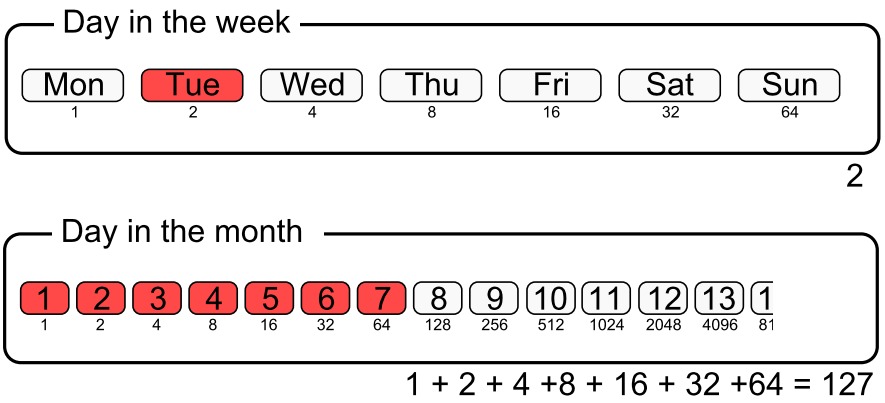
Variables can be combined. For a wake up to happen every day at 6h05, 6h10, 12h05, and 12h10, you must set the hours to 2^6 + 2^12 = 4060, minutes to 2^5 and 2^10 = 1056. Variables remaining at the zero value are ignored.



Example for wake ups at 6H05, 6h10, 12h05, and 12h10

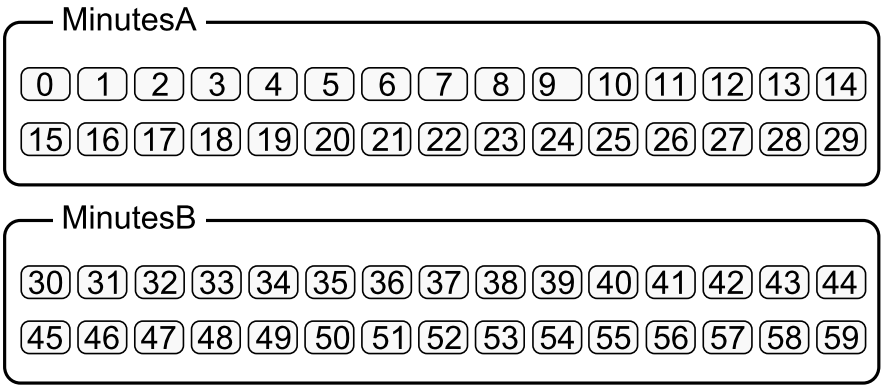
Note that if you want to program a wake up at 6h05 and 12h10, but not at 6h10 and 12h05, you need to use two distinct *wakeUpSchedule* functions.

This paradigm allows you to schedule complex wake ups. Thus, to program a wake up every first Tuesday of the month, you must set to 1 bit 1 of the days of the week and the first seven bits of the days of the month.



Example for a wake up every first Tuesday of the month

Some programming languages, among which JavaScript, do not support 64 bit integers. This is an issue for encoding minutes. Therefore, minutes are available both through a 64 bit integer *minutes* and two 32 bit integers *minutesA* and *minutesB*. These 32 bit integers are supposed to be available in any current programming language.



Minutes are also available in the shape of two 32 bit integers

The *wakeUpSchedule* function includes an additional variable to define the duration, in seconds, during which the module stays awake after a wake up. If this variable is set to zero, the modules stays awake.

The YoctoHub-GSM-4G includes two *wakeUpSchedule* functions, enabling you to program up to two independent wake up types.

8. Programming

8.1. Accessing connected modules

Without VPN, you can access the YoctoHub-GSM-4G functions by program only through its USB port, or through the HTTP callback API, using for example in PHP:

```
YAPI::RegisterHub("callback",errmsg);
```

In the rare cases when you can access your YoctoHub-GSM-4G through its IP address, it behaves itself exactly like a computer running VirtualHub. The only difference between a program using the Yoctopuce API with modules in native USB and the same program with Yoctopuce modules connected to a YoctoHub-GSM-4G is located at the level of the *RegisterHub* function call. To use modules connected to a YoctoHub-GSM-4G, you must use the IP address of the YoctoHub-GSM-4G when calling RegisterHub. For example, in PHP:

```
YAPI::RegisterHub("169.254.214.162",errmsg); // The hub IP address is
169.254.214.162
```

8.2. Controlling the YoctoHub-GSM-4G

From the programming API standpoint, the YoctoHub-GSM-4G is a module like any other when it is connected via USB. You can perfectly manage it from the Yoctopuce API. To do so, you need the following classes:

Module

This class, shared by all Yoctopuce modules, enables you to control the module itself. You can drive the Yocto-led, know the USB power consumption of the YoctoHub-GSM-4G, and so on.

Cellular

This class allows you to control the configuration of the YoctoHub-GSM-4G cellular network, in particular the radio access technology, the name of the cellular service provider, the PIN code to use the SIM card, and the APN parameters.

Network

This class enables you to manage the network part of the YoctoHub-GSM-4G. You can control the link state, read the MAC address, change the YoctoHub-GSM-4G IP address, know the power consumption on PoE, and so on.

Hub

This class provides the connection status between the programming library and the YoctoHubs or VirtualHubs that have been registered.

HubPort

This class enables you to manage the hub part. You can enable or disable the YoctoHub-GSM-4G ports, you can also know which module is connected to which port.

Files

This class enables you to access files stored in the flash memory of the YoctoHub-GSM-4G. The YoctoHub-GSM-4G contains a small file system which allows you to store, for example, a web application controlling the modules connected to the YoctoHub-GSM-4G.

WakeUpMonitor

This class enables you to monitor the sleep mode of the YoctoHub-GSM-4G.

WakeUpSchedule

This class enables you to schedule one or several wake ups for the YoctoHub-GSM-4G.

You can find some examples on how to drive the YoctoHub-GSM-4G by software in the Yoctopuce programming libraries, available free of charge on the Yoctopuce web site.

9. High-level API Reference

This chapter summarizes the high-level API functions to drive your YoctoHub-GSM-4G. Syntax and exact type names may vary from one language to another, but, unless otherwise stated, all the functions are available in every language. For detailed information regarding the types of arguments and return values for a given language, refer to the definition file for this language (`yocto_api.*` as well as the other `yocto_*` files that define the function interfaces).

For languages which support exceptions, all of these functions throw exceptions in case of error by default, rather than returning the documented error value for each function. This is by design, to facilitate debugging. It is however possible to disable the use of exceptions using the `yDisableExceptions()` function, in case you prefer to work with functions that return error values.

This chapter does not explain Yoctopuce programming concepts, in order to stay as concise as possible. You will find more details in the documentation of the devices you plan to connect to your YoctoHub-GSM-4G.

9.1. Class YModule

Global parameters control interface for all Yoctopuce devices

The `YModule` class can be used with all Yoctopuce USB devices. It can be used to control the module global parameters, and to enumerate the functions provided by each module.

In order to use the functions described here, you should include:

`js`	**<script type='text/javascript' src='yocto_api.js'></script>**
`cpp`	**#include "yocto_api.h"**
`m`	**#import "yocto_api.h"**
`pas`	**uses yocto_api;**
`vb`	**yocto_api.vb**
`cs`	**yocto_api.cs**
`java`	**import com.yoctopuce.YoctoAPI.YModule;**
`uwp`	**import com.yoctopuce.YoctoAPI.YModule;**
`py`	**from yocto_api import ***
`php`	**require_once('yocto_api.php');**
`ts`	**in HTML: import { YAPI, YErrorMsg, YModule, YSensor } from '../../dist/esm/yocto_api_browser.js';** **in Node.js: import { YAPI, YErrorMsg, YModule, YSensor } from 'yoctolib-cjs/yocto_api_nodejs.js';**
`es`	**in HTML: <script src="../../lib/yocto_api.js"></script>** **in node.js: require('yoctolib-es2017/yocto_api.js');**
`dnp`	**import YoctoProxyAPI.YModuleProxy**
`cp`	**#include "yocto_module_proxy.h"**
`vi`	**YModule.vi**
`ml`	**import YoctoProxyAPI.YModuleProxy"**

Global functions

YModule.FindModule(func)

Allows you to find a module from its serial number or from its logical name.

`cpp` `m` `pas` `vb` `cs` `java` `uwp` `py` `php` `ts` `es` `dnp`

YModule.FindModuleInContext(yctx, func)

Retrieves a module for a given identifier in a YAPI context.

`java` `uwp` `ts` `es`

YModule.FirstModule()

Starts the enumeration of modules currently accessible.

`cpp` `m` `pas` `vb` `cs` `java` `uwp` `py` `php` `ts` `es`

`YModule` properties

module→Beacon *[writable]*

State of the localization beacon.

`dnp`

module→FirmwareRelease *[read-only]*

Version of the firmware embedded in the module.

`dnp`

module→FunctionId *[read-only]*

Retrieves the hardware identifier of the *n*th function on the module.

dnp

module→HardwareId *[read-only]*

Unique hardware identifier of the module.

dnp

module→IsOnline *[read-only]*

Checks if the module is currently reachable.

dnp

module→LogicalName *[writable]*

Logical name of the module.

dnp

module→Luminosity *[writable]*

Luminosity of the module informative LEDs (from 0 to 100).

dnp

module→ProductId *[read-only]*

USB device identifier of the module.

dnp

module→ProductName *[read-only]*

Commercial name of the module, as set by the factory.

dnp

module→ProductRelease *[read-only]*

Release number of the module hardware, preprogrammed at the factory.

dnp

module→SerialNumber *[read-only]*

Serial number of the module, as set by the factory.

dnp

YModule methods

module→addFileToHTTPCallback(filename)

Adds a file to the uploaded data at the next HTTP callback.

cpp m pas vb cs java uwp py php ts es dnp cmd

module→checkFirmware(path, onlynew)

Tests whether the byn file is valid for this module.

cpp m pas vb cs java uwp py php ts es dnp cmd

module→clearCache()

Invalidates the cache.

cpp m pas vb cs java py php ts es

module→describe()

Returns a descriptive text that identifies the module.

cpp m pas vb cs java py php ts es

module→download(pathname)

Downloads the specified built-in file and returns a binary buffer with its content.

cpp m pas vb cs java uwp py php ts es dnp cmd

module→functionBaseType(functionIndex)

Retrieves the base type of the *n*th function on the module.

cpp pas vb cs java py php ts es

module→functionCount()

Returns the number of functions (beside the "module" interface) available on the module.

cpp m pas vb cs java py php ts es

module→functionId(functionIndex)

Retrieves the hardware identifier of the *n*th function on the module.

cpp m pas vb cs java py php ts es

module→functionName(functionIndex)

Retrieves the logical name of the *n*th function on the module.

cpp m pas vb cs java py php ts es

module→functionType(functionIndex)

Retrieves the type of the *n*th function on the module.

cpp pas vb cs java py php ts es

module→functionValue(functionIndex)

Retrieves the advertised value of the *n*th function on the module.

cpp m pas vb cs java py php ts es

module→get_allSettings()

Returns all the settings and uploaded files of the module.

cpp m pas vb cs java uwp py php ts es dnp cmd

module→get_beacon()

Returns the state of the localization beacon.

cpp m pas vb cs java uwp py php ts es dnp cmd

module→get_errorMessage()

Returns the error message of the latest error with this module object.

cpp m pas vb cs java py php ts es

module→get_errorType()

Returns the numerical error code of the latest error with this module object.

cpp m pas vb cs java py php ts es

module→get_firmwareRelease()

Returns the version of the firmware embedded in the module.

cpp m pas vb cs java uwp py php ts es dnp cmd

module→get_functionIds(funType)

Retrieve all hardware identifier that match the type passed in argument.

cpp m pas vb cs java uwp py php ts es dnp cmd

module→get_hardwareId()

Returns the unique hardware identifier of the module.

cpp m vb cs java py php ts es dnp pas uwp cmd

module→get_icon2d()

Returns the icon of the module.

cpp m pas vb cs java uwp py php ts es dnp cmd

module→get_lastLogs()

Returns a string with last logs of the module.

cpp m pas vb cs java uwp py php ts es dnp cmd

module→get_logicalName()

Returns the logical name of the module.

cpp m pas vb cs java uwp py php ts es dnp cmd

module→get_luminosity()

Returns the luminosity of the module informative LEDs (from 0 to 100).

cpp m pas vb cs java uwp py php ts es dnp cmd

module→get_parentHub()

Returns the serial number of the YoctoHub on which this module is connected.

cpp m pas vb cs java uwp py php ts es dnp cmd

module→get_persistentSettings()

Returns the current state of persistent module settings.

cpp m pas vb cs java uwp py php ts es dnp cmd

module→get_productId()

Returns the USB device identifier of the module.

cpp m pas vb cs java uwp py php ts es dnp cmd

module→get_productName()

Returns the commercial name of the module, as set by the factory.

cpp m pas vb cs java uwp py php ts es dnp cmd

module→get_productRelease()

Returns the release number of the module hardware, preprogrammed at the factory.

cpp m pas vb cs java uwp py php ts es dnp cmd

module→get_rebootCountdown()

Returns the remaining number of seconds before the module restarts, or zero when no reboot has been scheduled.

cpp m pas vb cs java uwp py php ts es dnp cmd

module→get_serialNumber()

Returns the serial number of the module, as set by the factory.

cpp m pas vb cs java uwp py php ts es dnp cmd

module→get_subDevices()

Returns a list of all the modules that are plugged into the current module.

cpp m pas vb cs java uwp py php ts es dnp cmd

module→get_upTime()

Returns the number of milliseconds spent since the module was powered on.

cpp m pas vb cs java uwp py php ts es dnp cmd

module→get_url()

Returns the URL used to access the module.

cpp m pas vb cs java uwp py php ts es dnp cmd

module→get_usbCurrent()

Returns the current consumed by the module on the USB bus, in milli-amps.

cpp m pas vb cs java uwp py php ts es dnp cmd

module→get_userData()

Returns the value of the userData attribute, as previously stored using method `set_userData`.

cpp m pas vb cs java py php ts es

module→get_userVar()

Returns the value previously stored in this attribute.

cpp m pas vb cs java uwp py php ts es dnp cmd

module→hasFunction(funcId)

Tests if the device includes a specific function.

cpp m pas vb cs java uwp py php ts es dnp cmd

module→isOnline()

Checks if the module is currently reachable, without raising any error.

cpp m pas vb cs java py php ts es dnp

module→isOnline_async(callback, context)

Checks if the module is currently reachable, without raising any error.

module→load(msValidity)

Preloads the module cache with a specified validity duration.

cpp m pas vb cs java py php ts es

module→load_async(msValidity, callback, context)

Preloads the module cache with a specified validity duration (asynchronous version).

module→log(text)

Adds a text message to the device logs.

cpp m pas vb cs java uwp py php ts es dnp cmd

module→nextModule()

Continues the module enumeration started using `yFirstModule()`.

cpp m pas vb cs java uwp py php ts es

module→reboot(secBeforeReboot)

Schedules a simple module reboot after the given number of seconds.

cpp m pas vb cs java uwp py php ts es dnp cmd

module→registerBeaconCallback(callback)

Register a callback function, to be called when the localization beacon of the module has been changed.

cpp m pas vb cs java uwp py php ts es

module→registerConfigChangeCallback(callback)

Register a callback function, to be called when a persistent settings in a device configuration has been changed (e.g.

cpp m pas vb cs java uwp py php ts es

module→registerLogCallback(callback)

Registers a device log callback function.

cpp m pas vb cs java uwp py php ts es

module→revertFromFlash()

Reloads the settings stored in the nonvolatile memory, as when the module is powered on.

cpp m pas vb cs java uwp py php ts es dnp cmd

module→saveToFlash()

Saves current settings in the nonvolatile memory of the module.

cpp m pas vb cs java uwp py php ts es dnp cmd

module→set_allSettings(settings)

Restores all the settings of the device.

cpp m pas vb cs java uwp py php ts es dnp cmd

module→set_allSettingsAndFiles(settings)

Restores all the settings and uploaded files to the module.

cpp m pas vb cs java uwp py php ts es dnp cmd

module→set_beacon(newval)

Turns on or off the module localization beacon.

cpp m pas vb cs java uwp py php ts es dnp cmd

module→set_logicalName(newval)

Changes the logical name of the module.

cpp m pas vb cs java uwp py php ts es dnp cmd

module→set_luminosity(newval)

Changes the luminosity of the module informative leds.

cpp m pas vb cs java uwp py php ts es dnp cmd

module→set_userData(data)

Stores a user context provided as argument in the userData attribute of the function.

cpp m pas vb cs java py php ts es

module→set_userVar(newval)

Stores a 32 bit value in the device RAM.

cpp m pas vb cs java uwp py php ts es dnp cmd

module→triggerConfigChangeCallback()

Triggers a configuration change callback, to check if they are supported or not.

cpp m pas vb cs java uwp py php ts es dnp cmd

module→triggerFirmwareUpdate(secBeforeReboot)

Schedules a module reboot into special firmware update mode.

cpp m pas vb cs java uwp py php ts es dnp cmd

module→updateFirmware(path)

Prepares a firmware update of the module.

cpp m pas vb cs java uwp py php ts es dnp cmd

module→updateFirmwareEx(path, force)

Prepares a firmware update of the module.

cpp m pas vb cs java uwp py php ts es dnp cmd

module→wait_async(callback, context)

Waits for all pending asynchronous commands on the module to complete, and invoke the user-provided callback function.

ts es

9.2. Class YCellular

Cellular interface control interface, available for instance in the YoctoHub-GSM-2G, the YoctoHub-GSM-3G-EU, the YoctoHub-GSM-3G-NA or the YoctoHub-GSM-4G

The `YCellular` class provides control over cellular network parameters and status for devices that are GSM-enabled. Note that TCP/IP parameters are configured separately, using class `YNetwork`.

In order to use the functions described here, you should include:

js	**<script type='text/javascript' src='yocto_cellular.js'></script>**
cpp	**#include "yocto_cellular.h"**
m	**#import "yocto_cellular.h"**
pas	**uses yocto_cellular;**
vb	**yocto_cellular.vb**
cs	**yocto_cellular.cs**
java	**import com.yoctopuce.YoctoAPI.YCellular;**
uwp	**import com.yoctopuce.YoctoAPI.YCellular;**
py	**from yocto_cellular import ***
php	**require_once('yocto_cellular.php');**
ts	**in HTML: import { YCellular } from '../../dist/esm/yocto_cellular.js';** **in Node.js: import { YCellular } from 'yoctolib-cjs/yocto_cellular.js';**
es	**in HTML: <script src="../../lib/yocto_cellular.js"></script>** **in node.js: require('yoctolib-es2017/yocto_cellular.js');**
dnp	**import YoctoProxyAPI.YCellularProxy**
cp	**#include "yocto_cellular_proxy.h"**
vi	**YCellular.vi**
ml	**import YoctoProxyAPI.YCellularProxy**

Global functions

YCellular.FindCellular(func)

Retrieves a cellular interface for a given identifier.

cpp m pas vb cs java uwp py php ts es dnp

YCellular.FindCellularInContext(yctx, func)

Retrieves a cellular interface for a given identifier in a YAPI context.

java uwp ts es

YCellular.FirstCellular()

Starts the enumeration of cellular interfaces currently accessible.

cpp m pas vb cs java uwp py php ts es

YCellular.FirstCellularInContext(yctx)

Starts the enumeration of cellular interfaces currently accessible.

java uwp ts es

YCellular.GetSimilarFunctions()

Enumerates all functions of type Cellular available on the devices currently reachable by the library, and returns their unique hardware ID.

dnp

`YCellular` properties

cellular→AdvertisedValue *[read-only]*

Short string representing the current state of the function.

dnp

cellular→Apn *[writable]*

Access Point Name (APN) to be used, if needed.

dnp

cellular→CellOperator *[read-only]*

Name of the cell operator currently in use.

dnp

cellular→EnableData *[writable]*

Condition for enabling IP data services (GPRS).

dnp

cellular→FriendlyName *[read-only]*

Global identifier of the function in the format `MODULE_NAME.FUNCTION_NAME`.

dnp

cellular→FunctionId *[read-only]*

Hardware identifier of the cellular interface, without reference to the module.

dnp

cellular→HardwareId *[read-only]*

Unique hardware identifier of the function in the form `SERIAL.FUNCTIONID`.

dnp

cellular→IsOnline *[read-only]*

Checks if the function is currently reachable.

dnp

cellular→LinkQuality *[read-only]*

Link quality, expressed in percent.

dnp

cellular→LockedOperator *[writable]*

Name of the only cell operator to use if automatic choice is disabled, or an empty string if the SIM card will automatically choose among available cell operators.

dnp

cellular→LogicalName *[writable]*

Logical name of the function.

dnp

cellular→Pin *[writable]*

N opaque string if a PIN code has been configured in the device to access the SIM card, or an empty string if none has been configured or if the code provided was rejected by the SIM card.

dnp

cellular→PingInterval *[writable]*

Automated connectivity check interval, in seconds.

dnp

cellular→RadioConfig *[writable]*

Type of protocol used over the serial line, as a string.

dnp

cellular→SerialNumber *[read-only]*

Serial number of the module, as set by the factory.

dnp

YCellular methods

cellular→_AT(cmd)

Sends an AT command to the GSM module and returns the command output.

cpp m pas vb cs java uwp py php ts es dnp cmd

cellular→clearCache()

Invalidates the cache.

cpp m pas vb cs java py php ts es

cellular→clearDataCounters()

Clear the transmitted data counters.

cpp m pas vb cs java uwp py php ts es dnp cmd

cellular→decodePLMN(mccmnc)

Returns the cell operator brand for a given MCC/MNC pair.

cpp m pas vb cs java uwp py php ts es dnp cmd

cellular→describe()

Returns a short text that describes unambiguously the instance of the cellular interface in the form `TYPE(NAME)=SERIAL.FUNCTIONID`.

cpp m pas vb cs java py php ts es

cellular→get_advertisedValue()

Returns the current value of the cellular interface (no more than 6 characters).

cpp m pas vb cs java uwp py php ts es dnp cmd

cellular→get_airplaneMode()

Returns true if the airplane mode is active (radio turned off).

cpp m pas vb cs java uwp py php ts es dnp cmd

cellular→get_apn()

Returns the Access Point Name (APN) to be used, if needed.

cpp m pas vb cs java uwp py php ts es dnp cmd

cellular→get_apnSecret()

Returns an opaque string if APN authentication parameters have been configured in the device, or an empty string otherwise.

cpp m pas vb cs java uwp py php ts es dnp cmd

cellular→get_availableOperators()

Returns the list detected cell operators in the neighborhood.

cpp m pas vb cs java uwp py php ts es dnp cmd

cellular→get_cellIdentifier()

Returns the unique identifier of the cellular antenna in use: MCC, MNC, LAC and Cell ID.

cpp m pas vb cs java uwp py php ts es dnp cmd

cellular→get_cellOperator()

Returns the name of the cell operator currently in use.

cpp m pas vb cs java uwp py php ts es dnp cmd

cellular→get_cellType()

Active cellular connection type.

cpp m pas vb cs java uwp py php ts es dnp cmd

cellular→get_communicationProfiles()

Returns the list available radio communication profiles, as a string array (YoctoHub-GSM-4G only).

cpp m pas vb cs java uwp py php ts es dnp cmd

cellular→get_dataReceived()

Returns the number of bytes received so far.

cpp m pas vb cs java uwp py php ts es dnp cmd

cellular→get_dataSent()

Returns the number of bytes sent so far.

cpp m pas vb cs java uwp py php ts es dnp cmd

cellular→get_enableData()

Returns the condition for enabling IP data services (GPRS).

cpp m pas vb cs java uwp py php ts es dnp cmd

cellular→get_errorMessage()

Returns the error message of the latest error with the cellular interface.

cpp m pas vb cs java py php ts es

cellular→get_errorType()

Returns the numerical error code of the latest error with the cellular interface.

cpp m pas vb cs java py php ts es

cellular→get_friendlyName()

Returns a global identifier of the cellular interface in the format MODULE_NAME.FUNCTION_NAME.

cpp m cs java py php ts es dnp

cellular→get_functionDescriptor()

Returns a unique identifier of type YFUN_DESCR corresponding to the function.

cpp m pas vb cs java py php ts es

cellular→get_functionId()

Returns the hardware identifier of the cellular interface, without reference to the module.

cpp m vb cs java py php ts es dnp

cellular→get_hardwareId()

Returns the unique hardware identifier of the cellular interface in the form SERIAL.FUNCTIONID.

cpp m vb cs java py php ts es dnp

cellular→get_imsi()

Returns the International Mobile Subscriber Identity (MSI) that uniquely identifies the SIM card.

cpp m pas vb cs java uwp py php ts es dnp cmd

cellular→get_linkQuality()

Returns the link quality, expressed in percent.

cpp m pas vb cs java uwp py php ts es dnp cmd

cellular→get_lockedOperator()

Returns the name of the only cell operator to use if automatic choice is disabled, or an empty string if the SIM card will automatically choose among available cell operators.

cpp m pas vb cs java uwp py php ts es dnp cmd

cellular→get_logicalName()

Returns the logical name of the cellular interface.

cpp m pas vb cs java uwp py php ts es dnp cmd

cellular→get_message()

Returns the latest status message from the wireless interface.

cpp m pas vb cs java uwp py php ts es dnp cmd

cellular→get_module()

Gets the `YModule` object for the device on which the function is located.

cpp m pas vb cs java py php ts es dnp

cellular→get_module_async(callback, context)

Gets the `YModule` object for the device on which the function is located (asynchronous version).

cellular→get_pin()

Returns an opaque string if a PIN code has been configured in the device to access the SIM card, or an empty string if none has been configured or if the code provided was rejected by the SIM card.

cpp m pas vb cs java uwp py php ts es dnp cmd

cellular→get_pingInterval()

Returns the automated connectivity check interval, in seconds.

cpp m pas vb cs java uwp py php ts es dnp cmd

cellular→get_radioConfig()

Returns the type of protocol used over the serial line, as a string.

cpp m pas vb cs java uwp py php ts es dnp cmd

cellular→get_serialNumber()

Returns the serial number of the module, as set by the factory.

cpp m pas vb cs java uwp py php ts es dnp cmd

cellular→get_userData()

Returns the value of the userData attribute, as previously stored using method `set_userData`.

cpp m pas vb cs java py php ts es

cellular→isOnline()

Checks if the cellular interface is currently reachable, without raising any error.

cpp m pas vb cs java py php ts es dnp

cellular→isOnline_async(callback, context)

Checks if the cellular interface is currently reachable, without raising any error (asynchronous version).

cellular→isReadOnly()

Indicates whether changes to the function are prohibited or allowed.

cpp m pas vb cs java uwp py php ts es dnp cmd

cellular→load(msValidity)

Preloads the cellular interface cache with a specified validity duration.

cpp m pas vb cs java py php ts es

cellular→loadAttribute(attrName)

Returns the current value of a single function attribute, as a text string, as quickly as possible but without using the cached value.

cpp m pas vb cs java uwp py php ts es dnp

cellular→load_async(msValidity, callback, context)

Preloads the cellular interface cache with a specified validity duration (asynchronous version).

cellular→muteValueCallbacks()

Disables the propagation of every new advertised value to the parent hub.

cpp m pas vb cs java uwp py php ts es dnp cmd

cellular→nextCellular()

Continues the enumeration of cellular interfaces started using `yFirstCellular()`.

cpp m pas vb cs java uwp py php ts es

cellular→quickCellSurvey()

Returns a list of nearby cellular antennas, as required for quick geolocation of the device.

cpp m pas vb cs java uwp py php ts es dnp cmd

cellular→registerValueCallback(callback)

Registers the callback function that is invoked on every change of advertised value.

cpp m pas vb cs java uwp py php ts es

cellular→sendPUK(puk, newPin)

Sends a PUK code to unlock the SIM card after three failed PIN code attempts, and setup a new PIN into the SIM card.

cpp m pas vb cs java uwp py php ts es dnp cmd

cellular→set_airplaneMode(newval)

Changes the activation state of airplane mode (radio turned off).

cpp m pas vb cs java uwp py php ts es dnp cmd

cellular→set_apn(newval)

Returns the Access Point Name (APN) to be used, if needed.

cpp m pas vb cs java uwp py php ts es dnp cmd

cellular→set_apnAuth(username, password)

Configure authentication parameters to connect to the APN.

cpp m pas vb cs java uwp py php ts es dnp cmd

cellular→set_dataReceived(newval)

Changes the value of the incoming data counter.

cpp m pas vb cs java uwp py php ts es dnp cmd

cellular→set_dataSent(newval)

Changes the value of the outgoing data counter.

cpp m pas vb cs java uwp py php ts es dnp cmd

cellular→set_enableData(newval)

Changes the condition for enabling IP data services (GPRS).

cpp m pas vb cs java uwp py php ts es dnp cmd

cellular→set_lockedOperator(newval)

Changes the name of the cell operator to be used.

cpp m pas vb cs java uwp py php ts es dnp cmd

cellular→set_logicalName(newval)

Changes the logical name of the cellular interface.

cpp m pas vb cs java uwp py php ts es dnp cmd

cellular→set_pin(newval)

Changes the PIN code used by the module to access the SIM card.

cpp m pas vb cs java uwp py php ts es dnp cmd

cellular→set_pingInterval(newval)

Changes the automated connectivity check interval, in seconds.

cpp m pas vb cs java uwp py php ts es dnp cmd

cellular→set_radioConfig(newval)

Changes the type of protocol used over the serial line.

cpp m pas vb cs java uwp py php ts es dnp cmd

cellular→set_userData(data)

Stores a user context provided as argument in the userData attribute of the function.

cpp m pas vb cs java py php ts es

cellular→unmuteValueCallbacks()

Re-enables the propagation of every new advertised value to the parent hub.

cpp m pas vb cs java uwp py php ts es dnp cmd

cellular→wait_async(callback, context)

Waits for all pending asynchronous commands on the module to complete, and invoke the user-provided callback function.

ts es

9.3. Class YNetwork

Network interface control interface, available for instance in the YoctoHub-Ethernet, the YoctoHub-GSM-4G, the YoctoHub-Wireless-SR or the YoctoHub-Wireless-n

`YNetwork` objects provide access to TCP/IP parameters of Yoctopuce devices that include a built-in network interface.

In order to use the functions described here, you should include:

es	**in HTML: <script src="../../lib/yocto_network.js"></script>** **in node.js: require('yoctolib-es2017/yocto_network.js');**
js	**<script type='text/javascript' src='yocto_network.js'></script>**
cpp	**#include "yocto_network.h"**
m	**#import "yocto_network.h"**
pas	**uses yocto_network;**
vb	**yocto_network.vb**
cs	**yocto_network.cs**
java	**import com.yoctopuce.YoctoAPI.YNetwork;**
uwp	**import com.yoctopuce.YoctoAPI.YNetwork;**
py	**from yocto_network import ***
php	**require_once('yocto_network.php');**
ts	**in HTML: import { YNetwork } from '../../dist/esm/yocto_network.js';** **in Node.js: import { YNetwork } from 'yoctolib-cjs/yocto_network.js';**
dnp	**import YoctoProxyAPI.YNetworkProxy**
cp	**#include "yocto_network_proxy.h"**
vi	**YNetwork.vi**
ml	**import YoctoProxyAPI.YNetworkProxy**

Global functions

YNetwork.FindNetwork(func)

Retrieves a network interface for a given identifier.

cpp m pas vb cs java uwp py php ts es dnp

YNetwork.FindNetworkInContext(yctx, func)

Retrieves a network interface for a given identifier in a YAPI context.

java uwp ts es

YNetwork.FirstNetwork()

Starts the enumeration of network interfaces currently accessible.

cpp m pas vb cs java uwp py php ts es

YNetwork.FirstNetworkInContext(yctx)

Starts the enumeration of network interfaces currently accessible.

java uwp ts es

YNetwork.GetSimilarFunctions()

Enumerates all functions of type Network available on the devices currently reachable by the library, and returns their unique hardware ID.

dnp

`YNetwork` properties

network→AdminPassword *[writable]*

Hash string if a password has been set for user "admin", or an empty string otherwise.

dnp

network→AdvertisedValue *[read-only]*

Short string representing the current state of the function.

dnp

network→CallbackCredentials *[writable]*

Hashed version of the notification callback credentials if set, or an empty string otherwise.

dnp

network→CallbackEncoding *[writable]*

Encoding standard to use for representing notification values.

dnp

network→CallbackInitialDelay *[writable]*

Initial waiting time before first callback notifications, in seconds.

dnp

network→CallbackMaxDelay *[writable]*

Waiting time between two HTTP callbacks when there is nothing new.

dnp

network→CallbackMethod *[writable]*

HTTP method used to notify callbacks for significant state changes.

dnp

network→CallbackMinDelay *[writable]*

Minimum waiting time between two HTTP callbacks, in seconds.

dnp

network→CallbackSchedule *[writable]*

HTTP callback schedule strategy, as a text string.

dnp

network→CallbackTemplate *[writable]*

Activation state of the custom template file to customize callback format.

dnp

network→CallbackUrl *[writable]*

Callback URL to notify of significant state changes.

dnp

network→DefaultPage *[writable]*

HTML page to serve for the URL "/"" of the hub.

dnp

network→Discoverable *[writable]*

Activation state of the multicast announce protocols to allow easy discovery of the module in the network neighborhood (uPnP/Bonjour protocol).

dnp

network→FriendlyName *[read-only]*

Global identifier of the function in the format `MODULE_NAME.FUNCTION_NAME`.

dnp

network→FunctionId *[read-only]*

Hardware identifier of the network interface, without reference to the module.

dnp

network→HardwareId *[read-only]*

Unique hardware identifier of the function in the form `SERIAL.FUNCTIONID`.

dnp

network→HttpPort *[writable]*

TCP port used to serve the hub web UI.

dnp

network→IpAddress *[read-only]*

IP address currently in use by the device.

dnp

network→IsOnline *[read-only]*

Checks if the function is currently reachable.

dnp

network→LogicalName *[writable]*

Logical name of the function.

dnp

network→MacAddress *[read-only]*

MAC address of the network interface.

dnp

network→NtpServer *[writable]*

IP address of the NTP server to be used by the device.

dnp

network→PrimaryDNS *[writable]*

IP address of the primary name server to be used by the module.

dnp

network→Readiness *[read-only]*

Current established working mode of the network interface.

dnp

network→SecondaryDNS *[writable]*

IP address of the secondary name server to be used by the module.

dnp

network→SerialNumber *[read-only]*

Serial number of the module, as set by the factory.

dnp

network→UserPassword *[writable]*

Hash string if a password has been set for "user" user, or an empty string otherwise.

dnp

network→WwwWatchdogDelay *[writable]*

Allowed downtime of the WWW link (in seconds) before triggering an automated reboot to try to recover Internet connectivity.

dnp

YNetwork methods

network→callbackLogin(username, password)

Connects to the notification callback and saves the credentials required to log into it.

cpp m pas vb cs java py php ts es dnp cmd

network→clearCache()

Invalidates the cache.

cpp m pas vb cs java py php ts es

network→describe()

Returns a short text that describes unambiguously the instance of the network interface in the form TYPE(NAME)=SERIAL.FUNCTIONID.

cpp m pas vb cs java py php ts es

network→get_adminPassword()

Returns a hash string if a password has been set for user "admin", or an empty string otherwise.

cpp m pas vb cs java uwp py php ts es dnp cmd

network→get_advertisedValue()

Returns the current value of the network interface (no more than 6 characters).

cpp m pas vb cs java uwp py php ts es dnp cmd

network→get_callbackCredentials()

Returns a hashed version of the notification callback credentials if set, or an empty string otherwise.

cpp m pas vb cs java uwp py php ts es dnp cmd

network→get_callbackEncoding()

Returns the encoding standard to use for representing notification values.

cpp m pas vb cs java uwp py php ts es dnp cmd

network→get_callbackInitialDelay()

Returns the initial waiting time before first callback notifications, in seconds.

cpp m pas vb cs java uwp py php ts es dnp cmd

network→get_callbackMaxDelay()

Returns the waiting time between two HTTP callbacks when there is nothing new.

cpp m pas vb cs java uwp py php ts es dnp cmd

network→get_callbackMethod()

Returns the HTTP method used to notify callbacks for significant state changes.

cpp m pas vb cs java uwp py php ts es dnp cmd

network→get_callbackMinDelay()

Returns the minimum waiting time between two HTTP callbacks, in seconds.

cpp m pas vb cs java uwp py php ts es dnp cmd

network→get_callbackSchedule()

Returns the HTTP callback schedule strategy, as a text string.

cpp m pas vb cs java uwp py php ts es dnp cmd

network→get_callbackTemplate()

Returns the activation state of the custom template file to customize callback format.

cpp m pas vb cs java uwp py php ts es dnp cmd

network→get_callbackUrl()

Returns the callback URL to notify of significant state changes.

cpp m pas vb cs java uwp py php ts es dnp cmd

network→get_currentDNS()

Returns the IP address of the DNS server currently used by the device.

cpp m pas vb cs java uwp py php ts es dnp cmd

network→get_defaultPage()

Returns the HTML page to serve for the URL "/"" of the hub.

cpp m pas vb cs java uwp py php ts es dnp cmd

network→get_discoverable()

Returns the activation state of the multicast announce protocols to allow easy discovery of the module in the network neighborhood (uPnP/Bonjour protocol).

cpp m pas vb cs java uwp py php ts es dnp cmd

network→get_errorMessage()

Returns the error message of the latest error with the network interface.

cpp m pas vb cs java py php ts es

network→get_errorType()

Returns the numerical error code of the latest error with the network interface.

cpp m pas vb cs java py php ts es

network→get_friendlyName()

Returns a global identifier of the network interface in the format `MODULE_NAME.FUNCTION_NAME`.

cpp m cs java py php ts es dnp

network→get_functionDescriptor()

Returns a unique identifier of type `YFUN_DESCR` corresponding to the function.

cpp m pas vb cs java py php ts es

network→get_functionId()

Returns the hardware identifier of the network interface, without reference to the module.

cpp m vb cs java py php ts es dnp

network→get_hardwareId()

Returns the unique hardware identifier of the network interface in the form `SERIAL.FUNCTIONID`.

cpp m vb cs java py php ts es dnp

network→get_httpPort()

Returns the TCP port used to serve the hub web UI.

cpp m pas vb cs java uwp py php ts es dnp cmd

network→get_ipAddress()

Returns the IP address currently in use by the device.

cpp m pas vb cs java uwp py php ts es dnp cmd

network→get_ipConfig()

Returns the IP configuration of the network interface.

cpp m pas vb cs java uwp py php ts es dnp cmd

network→get_logicalName()

Returns the logical name of the network interface.

cpp m pas vb cs java uwp py php ts es dnp cmd

network→get_macAddress()

Returns the MAC address of the network interface.

cpp m pas vb cs java uwp py php ts es dnp cmd

network→get_module()

Gets the `YModule` object for the device on which the function is located.

cpp m pas vb cs java py php ts es dnp

network→get_module_async(callback, context)

Gets the `YModule` object for the device on which the function is located (asynchronous version).

network→get_ntpServer()

Returns the IP address of the NTP server to be used by the device.

cpp m pas vb cs java uwp py php ts es dnp cmd

network→get_poeCurrent()

Returns the current consumed by the module from Power-over-Ethernet (PoE), in milliamps.

cpp m pas vb cs java uwp py php ts es dnp cmd

network→get_primaryDNS()

Returns the IP address of the primary name server to be used by the module.

cpp m pas vb cs java uwp py php ts es dnp cmd

network→get_readiness()

Returns the current established working mode of the network interface.

cpp m pas vb cs java uwp py php ts es dnp cmd

network→get_router()

Returns the IP address of the router on the device subnet (default gateway).

cpp m pas vb cs java uwp py php ts es dnp cmd

network→get_secondaryDNS()

Returns the IP address of the secondary name server to be used by the module.

cpp m pas vb cs java uwp py php ts es dnp cmd

network→get_serialNumber()

Returns the serial number of the module, as set by the factory.

cpp m pas vb cs java uwp py php ts es dnp cmd

network→get_subnetMask()

Returns the subnet mask currently used by the device.

cpp m pas vb cs java uwp py php ts es dnp cmd

network→get_userData()

Returns the value of the userData attribute, as previously stored using method `set_userData`.

cpp m pas vb cs java py php ts es

network→get_userPassword()

Returns a hash string if a password has been set for "user" user, or an empty string otherwise.

cpp m pas vb cs java uwp py php ts es dnp cmd

network→get_wwwWatchdogDelay()

Returns the allowed downtime of the WWW link (in seconds) before triggering an automated reboot to try to recover Internet connectivity.

cpp m pas vb cs java uwp py php ts es dnp cmd

network→isOnline()

Checks if the network interface is currently reachable, without raising any error.

cpp m pas vb cs java py php ts es dnp

network→isOnline_async(callback, context)

Checks if the network interface is currently reachable, without raising any error (asynchronous version).

network→isReadOnly()

Indicates whether changes to the function are prohibited or allowed.

cpp m pas vb cs java uwp py php ts es dnp cmd

network→load(msValidity)

Preloads the network interface cache with a specified validity duration.

cpp m pas vb cs java py php ts es

network→loadAttribute(attrName)

Returns the current value of a single function attribute, as a text string, as quickly as possible but without using the cached value.

cpp m pas vb cs java uwp py php ts es dnp

network→load_async(msValidity, callback, context)

Preloads the network interface cache with a specified validity duration (asynchronous version).

network→muteValueCallbacks()

Disables the propagation of every new advertised value to the parent hub.

cpp m pas vb cs java uwp py php ts es dnp cmd

network→nextNetwork()

Continues the enumeration of network interfaces started using `yFirstNetwork()`.

cpp m pas vb cs java uwp py php ts es

network→ping(host)

Pings host to test the network connectivity.

cpp m pas vb cs java uwp py php ts es dnp cmd

network→registerValueCallback(callback)

Registers the callback function that is invoked on every change of advertised value.

cpp m pas vb cs java uwp py php ts es

network→set_adminPassword(newval)

Changes the password for the "admin" user.

cpp m pas vb cs java uwp py php ts es dnp cmd

network→set_callbackCredentials(newval)

Changes the credentials required to connect to the callback address.

cpp m pas vb cs java uwp py php ts es dnp cmd

network→set_callbackEncoding(newval)

Changes the encoding standard to use for representing notification values.

cpp m pas vb cs java uwp py php ts es dnp cmd

network→set_callbackInitialDelay(newval)

Changes the initial waiting time before first callback notifications, in seconds.

cpp m pas vb cs java uwp py php ts es dnp cmd

network→set_callbackMaxDelay(newval)

Changes the waiting time between two HTTP callbacks when there is nothing new.

cpp m pas vb cs java uwp py php ts es dnp cmd

network→set_callbackMethod(newval)

Changes the HTTP method used to notify callbacks for significant state changes.

cpp m pas vb cs java uwp py php ts es dnp cmd

network→set_callbackMinDelay(newval)

Changes the minimum waiting time between two HTTP callbacks, in seconds.

cpp m pas vb cs java uwp py php ts es dnp cmd

network→set_callbackSchedule(newval)

Changes the HTTP callback schedule strategy, as a text string.

cpp m pas vb cs java uwp py php ts es dnp cmd

network→set_callbackTemplate(newval)

Enable the use of a template file to customize callbacks format.

cpp m pas vb cs java uwp py php ts es dnp cmd

network→set_callbackUrl(newval)

Changes the callback URL to notify significant state changes.

cpp m pas vb cs java uwp py php ts es dnp cmd

network→set_defaultPage(newval)

Changes the default HTML page returned by the hub.

cpp m pas vb cs java uwp py php ts es dnp cmd

network→set_discoverable(newval)

Changes the activation state of the multicast announce protocols to allow easy discovery of the module in the network neighborhood (uPnP/Bonjour protocol).

cpp m pas vb cs java uwp py php ts es dnp cmd

network→set_httpPort(newval)

Changes the the TCP port used to serve the hub web UI.

cpp m pas vb cs java uwp py php ts es dnp cmd

network→set_logicalName(newval)

Changes the logical name of the network interface.

cpp m pas vb cs java uwp py php ts es dnp cmd

network→set_ntpServer(newval)

Changes the IP address of the NTP server to be used by the module.

cpp m pas vb cs java uwp py php ts es dnp cmd

network→set_periodicCallbackSchedule(interval, offset)

Setup periodic HTTP callbacks (simplified function).

cpp m pas vb cs java uwp py php ts es dnp cmd

network→set_primaryDNS(newval)

Changes the IP address of the primary name server to be used by the module.

cpp m pas vb cs java uwp py php ts es dnp cmd

network→set_secondaryDNS(newval)

Changes the IP address of the secondary name server to be used by the module.

cpp m pas vb cs java uwp py php ts es dnp cmd

network→set_userData(data)

Stores a user context provided as argument in the userData attribute of the function.

cpp m pas vb cs java py php ts es

network→set_userPassword(newval)

Changes the password for the "user" user.

cpp m pas vb cs java uwp py php ts es dnp cmd

network→set_wwwWatchdogDelay(newval)

Changes the allowed downtime of the WWW link (in seconds) before triggering an automated reboot to try to recover Internet connectivity.

cpp m pas vb cs java uwp py php ts es dnp cmd

network→triggerCallback()

Trigger an HTTP callback quickly.

cpp m pas vb cs java uwp py php ts es dnp cmd

network→unmuteValueCallbacks()

Re-enables the propagation of every new advertised value to the parent hub.

cpp m pas vb cs java uwp py php ts es dnp cmd

network→useDHCP(fallbackIpAddr, fallbackSubnetMaskLen, fallbackRouter)

Changes the configuration of the network interface to enable the use of an IP address received from a DHCP server.

cpp m pas vb cs java uwp py php ts es dnp cmd

network→useDHCPauto()

Changes the configuration of the network interface to enable the use of an IP address received from a DHCP server.

cpp m pas vb cs java uwp py php ts es dnp cmd

network→useStaticIP(ipAddress, subnetMaskLen, router)

Changes the configuration of the network interface to use a static IP address.

cpp m pas vb cs java uwp py php ts es dnp cmd

network→wait_async(callback, context)

Waits for all pending asynchronous commands on the module to complete, and invoke the user-provided callback function.

ts es

9.4. Class YHub

In order to use the functions described here, you should include:

js	**<script type='text/javascript' src='yocto_module.js'></script>**
cpp	**#include "yocto_module.h"**
m	**#import "yocto_module.h"**
pas	**uses yocto_module;**
vb	**yocto_module.vb**
cs	**yocto_module.cs**
java	**import com.yoctopuce.YoctoAPI.YHub;**
uwp	**import com.yoctopuce.YoctoAPI.YHub;**
py	**from yocto_module import ***
php	**require_once('yocto_module.php');**
ts	**in HTML: import { YHub } from '../../dist/esm/yocto_module.js';** **in Node.js: import { YHub } from 'yoctolib-cjs/yocto_module.js';**
es	**in HTML: <script src="../../lib/yocto_module.js"></script>** **in node.js: require('yoctolib-es2017/yocto_module.js');**
dnp	**import YoctoProxyAPI.YHubProxy**
cp	**#include "yocto_module_proxy.h"**
ml	**import YoctoProxyAPI.YHubProxy**

Global functions

YHub.FirstHubInUse()

Starts the enumeration of hubs currently in use by the API.

cpp m vb cs java uwp py php ts es dnp

YHub.FirstHubInUseInContext(yctx)

Starts the enumeration of hubs currently in use by the API in a given YAPI context.

java uwp ts es

YHub methods

hub→get_connectionUrl()

Returns the URL currently in use to communicate with this hub.

cpp m pas vb cs java uwp py php ts es dnp

hub→get_errorMessage()

Returns the error message of the latest error with the hub.

cpp m pas vb cs java uwp py php ts es dnp

hub→get_errorType()

Returns the numerical error code of the latest error with the hub.

cpp m pas vb cs java uwp py php ts es dnp

hub→get_knownUrls()

Returns all known URLs that have been used to register this hub.

cpp m pas vb cs java uwp py php ts es dnp

hub→get_networkTimeout()

Returns the network connection delay for this hub.

cpp m pas vb cs java uwp py php ts es dnp

hub→get_registeredUrl()

Returns the URL that has been used first to register this hub.

cpp m pas vb cs java uwp py php ts es dnp

hub→get_serialNumber()

Returns the hub serial number, if the hub was already connected once.

cpp m pas vb cs java uwp py php ts es dnp

hub→get_userData()

Returns the value of the userData attribute, as previously stored using method `set_userData`.

cpp m pas vb cs java uwp py php ts es

hub→isInUse()

Tells if this hub is still registered within the API.

cpp m pas vb cs java uwp py php ts es dnp

hub→isOnline()

Tells if there is an active communication channel with this hub.

cpp m pas vb cs java uwp py php ts es dnp

hub→isReadOnly()

Tells if write access on this hub is blocked.

cpp m pas vb cs java uwp py php ts es dnp

hub→nextHubInUse()

Continues the module enumeration started using `YHub.FirstHubInUse()`.

cpp m pas vb cs java uwp py php ts es dnp

hub→set_networkTimeout(networkMsTimeout)

Modifies tthe network connection delay for this hub.

cpp m pas vb cs java uwp py php ts es dnp

hub→set_userData(data)

Stores a user context provided as argument in the userData attribute of the function.

cpp m pas vb cs java uwp py php ts es

9.5. Class YHubPort

YoctoHub slave port control interface, available for instance in the YoctoHub-Ethernet, the YoctoHub-GSM-4G, the YoctoHub-Shield or the YoctoHub-Wireless-n

The `YHubPort` class provides control over the power supply for slave ports on a YoctoHub. It provide information about the device connected to it. The logical name of a YHubPort is always automatically set to the unique serial number of the Yoctopuce device connected to it.

In order to use the functions described here, you should include:

es	**in HTML: <script src="../../lib/yocto_hubport.js"></script>** **in node.js: require('yoctolib-es2017/yocto_hubport.js');**
js	**<script type='text/javascript' src='yocto_hubport.js'></script>**
cpp	**#include "yocto_hubport.h"**
m	**#import "yocto_hubport.h"**
pas	**uses yocto_hubport;**
vb	**yocto_hubport.vb**
cs	**yocto_hubport.cs**
java	**import com.yoctopuce.YoctoAPI.YHubPort;**
uwp	**import com.yoctopuce.YoctoAPI.YHubPort;**
py	**from yocto_hubport import ***
php	**require_once('yocto_hubport.php');**
ts	**in HTML: import { YHubPort } from '../../dist/esm/yocto_hubport.js';** **in Node.js: import { YHubPort } from 'yoctolib-cjs/yocto_hubport.js';**
dnp	**import YoctoProxyAPI.YHubPortProxy**
cp	**#include "yocto_hubport_proxy.h"**
vi	**YHubPort.vi**
ml	**import YoctoProxyAPI.YHubPortProxy**

Global functions

YHubPort.FindHubPort(func)

Retrieves a YoctoHub slave port for a given identifier.

cpp m pas vb cs java uwp py php ts es dnp

YHubPort.FindHubPortInContext(yctx, func)

Retrieves a YoctoHub slave port for a given identifier in a YAPI context.

java uwp ts es

YHubPort.FirstHubPort()

Starts the enumeration of YoctoHub slave ports currently accessible.

cpp m pas vb cs java uwp py php ts es

YHubPort.FirstHubPortInContext(yctx)

Starts the enumeration of YoctoHub slave ports currently accessible.

java uwp ts es

YHubPort.GetSimilarFunctions()

Enumerates all functions of type HubPort available on the devices currently reachable by the library, and returns their unique hardware ID.

dnp

`YHubPort` properties

hubport→AdvertisedValue *[read-only]*

Short string representing the current state of the function.

dnp

hubport→Enabled *[writable]*

True if the YoctoHub port is powered, false otherwise.

dnp

hubport→FriendlyName *[read-only]*

Global identifier of the function in the format `MODULE_NAME.FUNCTION_NAME`.

dnp

hubport→FunctionId *[read-only]*

Hardware identifier of the YoctoHub slave port, without reference to the module.

dnp

hubport→HardwareId *[read-only]*

Unique hardware identifier of the function in the form `SERIAL.FUNCTIONID`.

dnp

hubport→IsOnline *[read-only]*

Checks if the function is currently reachable.

dnp

hubport→LogicalName *[writable]*

Logical name of the function.

dnp

hubport→PortState *[read-only]*

Current state of the YoctoHub port.

dnp

hubport→SerialNumber *[read-only]*

Serial number of the module, as set by the factory.

dnp

`YHubPort` methods

hubport→clearCache()

Invalidates the cache.

cpp m pas vb cs java py php ts es

hubport→describe()

Returns a short text that describes unambiguously the instance of the YoctoHub slave port in the form `TYPE(NAME)=SERIAL.FUNCTIONID`.

cpp m pas vb cs java py php ts es

hubport→get_advertisedValue()

Returns the current value of the YoctoHub slave port (no more than 6 characters).

cpp m pas vb cs java uwp py php ts es dnp cmd

hubport→get_baudRate()

Returns the current baud rate used by this YoctoHub port, in kbps.

cpp m pas vb cs java uwp py php ts es dnp cmd

hubport→get_enabled()

Returns true if the YoctoHub port is powered, false otherwise.

cpp m pas vb cs java uwp py php ts es dnp cmd

hubport→get_errorMessage()

Returns the error message of the latest error with the YoctoHub slave port.

cpp m pas vb cs java py php ts es

hubport→get_errorType()

Returns the numerical error code of the latest error with the YoctoHub slave port.

cpp m pas vb cs java py php ts es

hubport→get_friendlyName()

Returns a global identifier of the YoctoHub slave port in the format `MODULE_NAME.FUNCTION_NAME`.

cpp m cs java py php ts es dnp

hubport→get_functionDescriptor()

Returns a unique identifier of type `YFUN_DESCR` corresponding to the function.

cpp m pas vb cs java py php ts es

hubport→get_functionId()

Returns the hardware identifier of the YoctoHub slave port, without reference to the module.

cpp m vb cs java py php ts es dnp

hubport→get_hardwareId()

Returns the unique hardware identifier of the YoctoHub slave port in the form `SERIAL.FUNCTIONID`.

cpp m vb cs java py php ts es dnp

hubport→get_logicalName()

Returns the logical name of the YoctoHub slave port.

cpp m pas vb cs java uwp py php ts es dnp cmd

hubport→get_module()

Gets the `YModule` object for the device on which the function is located.

cpp m pas vb cs java py php ts es dnp

hubport→get_module_async(callback, context)

Gets the `YModule` object for the device on which the function is located (asynchronous version).

hubport→get_portState()

Returns the current state of the YoctoHub port.

cpp m pas vb cs java uwp py php ts es dnp cmd

hubport→get_serialNumber()

Returns the serial number of the module, as set by the factory.

cpp m pas vb cs java uwp py php ts es dnp cmd

hubport→get_userData()

Returns the value of the userData attribute, as previously stored using method `set_userData`.

cpp m pas vb cs java py php ts es

hubport→isOnline()

Checks if the YoctoHub slave port is currently reachable, without raising any error.

cpp m pas vb cs java py php ts es dnp

hubport→isOnline_async(callback, context)

Checks if the YoctoHub slave port is currently reachable, without raising any error (asynchronous version).

hubport→isReadOnly()

Indicates whether changes to the function are prohibited or allowed.

cpp m pas vb cs java uwp py php ts es dnp cmd

hubport→load(msValidity)

Preloads the YoctoHub slave port cache with a specified validity duration.

cpp m pas vb cs java py php ts es

hubport→loadAttribute(attrName)

Returns the current value of a single function attribute, as a text string, as quickly as possible but without using the cached value.

cpp m pas vb cs java uwp py php ts es dnp

hubport→load_async(msValidity, callback, context)

Preloads the YoctoHub slave port cache with a specified validity duration (asynchronous version).

hubport→muteValueCallbacks()

Disables the propagation of every new advertised value to the parent hub.

cpp m pas vb cs java uwp py php ts es dnp cmd

hubport→nextHubPort()

Continues the enumeration of YoctoHub slave ports started using `yFirstHubPort()`.

cpp m pas vb cs java uwp py php ts es

hubport→registerValueCallback(callback)

Registers the callback function that is invoked on every change of advertised value.

cpp m pas vb cs java uwp py php ts es

hubport→set_enabled(newval)

Changes the activation of the YoctoHub port.

cpp m pas vb cs java uwp py php ts es dnp cmd

hubport→set_logicalName(newval)

Changes the logical name of the YoctoHub slave port.

cpp m pas vb cs java uwp py php ts es dnp cmd

hubport→set_userData(data)

Stores a user context provided as argument in the userData attribute of the function.

cpp m pas vb cs java py php ts es

hubport→unmuteValueCallbacks()

Re-enables the propagation of every new advertised value to the parent hub.

cpp m pas vb cs java uwp py php ts es dnp cmd

hubport→wait_async(callback, context)

Waits for all pending asynchronous commands on the module to complete, and invoke the user-provided callback function.

ts es

9.6. Class YFiles

Filesystem control interface, available for instance in the Yocto-Color-V2, the Yocto-SPI, the YoctoHub-Ethernet or the YoctoHub-GSM-4G

The YFiles class is used to access the filesystem embedded on some Yoctopuce devices. This filesystem makes it possible for instance to design a custom web UI (for networked devices) or to add fonts (on display devices).

In order to use the functions described here, you should include:

js	**<script type='text/javascript' src='yocto_files.js'></script>**
cpp	**#include "yocto_files.h"**
m	**#import "yocto_files.h"**
pas	**uses yocto_files;**
vb	**yocto_files.vb**
cs	**yocto_files.cs**
java	**import com.yoctopuce.YoctoAPI.YFiles;**
uwp	**import com.yoctopuce.YoctoAPI.YFiles;**
py	**from yocto_files import ***
php	**require_once('yocto_files.php');**
ts	**in HTML: import { YFiles } from '../../dist/esm/yocto_files.js';** **in Node.js: import { YFiles } from 'yoctolib-cjs/yocto_files.js';**
es	**in HTML: <script src="../../lib/yocto_files.js"></script>** **in node.js: require('yoctolib-es2017/yocto_files.js');**
dnp	**import YoctoProxyAPI.YFilesProxy**
cp	**#include "yocto_files_proxy.h"**
vi	**YFiles.vi**
ml	**import YoctoProxyAPI.YFilesProxy**

Global functions

YFiles.FindFiles(func)

Retrieves a filesystem for a given identifier.

cpp m pas vb cs java uwp py php ts es dnp

YFiles.FindFilesInContext(yctx, func)

Retrieves a filesystem for a given identifier in a YAPI context.

java uwp ts es

YFiles.FirstFiles()

Starts the enumeration of filesystems currently accessible.

cpp m pas vb cs java uwp py php ts es

YFiles.FirstFilesInContext(yctx)

Starts the enumeration of filesystems currently accessible.

java uwp ts es

YFiles.GetSimilarFunctions()

Enumerates all functions of type Files available on the devices currently reachable by the library, and returns their unique hardware ID.

dnp

YFiles properties

files→AdvertisedValue *[read-only]*

Short string representing the current state of the function.

dnp

files→FilesCount *[read-only]*

Number of files currently loaded in the filesystem.

dnp

files→FriendlyName *[read-only]*

Global identifier of the function in the format `MODULE_NAME.FUNCTION_NAME`.

dnp

files→FunctionId *[read-only]*

Hardware identifier of the filesystem, without reference to the module.

dnp

files→HardwareId *[read-only]*

Unique hardware identifier of the function in the form `SERIAL.FUNCTIONID`.

dnp

files→IsOnline *[read-only]*

Checks if the function is currently reachable.

dnp

files→LogicalName *[writable]*

Logical name of the function.

dnp

files→SerialNumber *[read-only]*

Serial number of the module, as set by the factory.

dnp

YFiles methods

files→clearCache()

Invalidates the cache.

cpp m pas vb cs java py php ts es

files→describe()

Returns a short text that describes unambiguously the instance of the filesystem in the form `TYPE(NAME)=SERIAL.FUNCTIONID`.

cpp m pas vb cs java py php ts es

files→download(pathname)

Downloads the requested file and returns a binary buffer with its content.

cpp m pas vb cs java uwp py php ts es dnp cmd

files→download_async(pathname, callback, context)

Downloads the requested file and returns a binary buffer with its content.

files→fileExist(filename)

Test if a file exist on the filesystem of the module.

cpp m pas vb cs java uwp py php ts es dnp cmd

files→format_fs()

Reinitialize the filesystem to its clean, unfragmented, empty state.

cpp m pas vb cs java uwp py php ts es dnp cmd

files→get_advertisedValue()

Returns the current value of the filesystem (no more than 6 characters).

cpp m pas vb cs java uwp py php ts es dnp cmd

files→get_errorMessage()

Returns the error message of the latest error with the filesystem.

cpp m pas vb cs java py php ts es

files→get_errorType()

Returns the numerical error code of the latest error with the filesystem.

cpp m pas vb cs java py php ts es

files→get_filesCount()

Returns the number of files currently loaded in the filesystem.

cpp m pas vb cs java uwp py php ts es dnp cmd

files→get_freeSpace()

Returns the free space for uploading new files to the filesystem, in bytes.

cpp m pas vb cs java uwp py php ts es dnp cmd

files→get_friendlyName()

Returns a global identifier of the filesystem in the format `MODULE_NAME.FUNCTION_NAME`.

cpp m cs java py php ts es dnp

files→get_functionDescriptor()

Returns a unique identifier of type `YFUN_DESCR` corresponding to the function.

cpp m pas vb cs java py php ts es

files→get_functionId()

Returns the hardware identifier of the filesystem, without reference to the module.

cpp m vb cs java py php ts es dnp

files→get_hardwareId()

Returns the unique hardware identifier of the filesystem in the form `SERIAL.FUNCTIONID`.

cpp m vb cs java py php ts es dnp

files→get_list(pattern)

Returns a list of YFileRecord objects that describe files currently loaded in the filesystem.

cpp m pas vb cs java uwp py php ts es dnp cmd

files→get_logicalName()

Returns the logical name of the filesystem.

cpp m pas vb cs java uwp py php ts es dnp cmd

files→get_module()

Gets the `YModule` object for the device on which the function is located.

cpp m pas vb cs java py php ts es dnp

files→get_module_async(callback, context)

Gets the `YModule` object for the device on which the function is located (asynchronous version).

files→get_serialNumber()

Returns the serial number of the module, as set by the factory.

cpp m pas vb cs java uwp py php ts es dnp cmd

files→get_userData()

Returns the value of the userData attribute, as previously stored using method `set_userData`.

cpp m pas vb cs java py php ts es

files→isOnline()

Checks if the filesystem is currently reachable, without raising any error.

cpp m pas vb cs java py php ts es dnp

files→isOnline_async(callback, context)

Checks if the filesystem is currently reachable, without raising any error (asynchronous version).

files→isReadOnly()

Indicates whether changes to the function are prohibited or allowed.

cpp m pas vb cs java uwp py php ts es dnp cmd

files→load(msValidity)

Preloads the filesystem cache with a specified validity duration.

cpp m pas vb cs java py php ts es

files→loadAttribute(attrName)

Returns the current value of a single function attribute, as a text string, as quickly as possible but without using the cached value.

cpp m pas vb cs java uwp py php ts es dnp

files→load_async(msValidity, callback, context)

Preloads the filesystem cache with a specified validity duration (asynchronous version).

files→muteValueCallbacks()

Disables the propagation of every new advertised value to the parent hub.

cpp m pas vb cs java uwp py php ts es dnp cmd

files→nextFiles()

Continues the enumeration of filesystems started using `yFirstFiles()`.

cpp m pas vb cs java uwp py php ts es

files→registerValueCallback(callback)

Registers the callback function that is invoked on every change of advertised value.

cpp m pas vb cs java uwp py php ts es

files→remove(pathname)

Deletes a file, given by its full path name, from the filesystem.

cpp m pas vb cs java uwp py php ts es dnp cmd

files→set_logicalName(newval)

Changes the logical name of the filesystem.

cpp m pas vb cs java uwp py php ts es dnp cmd

files→set_userData(data)

Stores a user context provided as argument in the userData attribute of the function.

cpp m pas vb cs java py php ts es

files→unmuteValueCallbacks()

Re-enables the propagation of every new advertised value to the parent hub.

cpp m pas vb cs java uwp py php ts es dnp cmd

files→upload(pathname, content)

Uploads a file to the filesystem, to the specified full path name.

cpp m pas vb cs java uwp py php ts es dnp cmd

files→wait_async(callback, context)

Waits for all pending asynchronous commands on the module to complete, and invoke the user-provided callback function.

ts es

9.7. Class YRealTimeClock

Real-time clock control interface, available for instance in the YoctoHub-GSM-4G, the YoctoHub-Wireless-SR, the YoctoHub-Wireless-g or the YoctoHub-Wireless-n

The `YRealTimeClock` class provide access to the embedded real-time clock available on some Yoctopuce devices. It can provide current date and time, even after a power outage lasting several days. It is the base for automated wake-up functions provided by the WakeUpScheduler. The current time may represent a local time as well as an UTC time, but no automatic time change will occur to account for daylight saving time.

In order to use the functions described here, you should include:

es	**in HTML: <script src="../../lib/yocto_realtimeclock.js"></script>** **in node.js: require('yoctolib-es2017/yocto_realtimeclock.js');**
js	**<script type='text/javascript' src='yocto_realtimeclock.js'></script>**
cpp	**#include "yocto_realtimeclock.h"**
m	**#import "yocto_realtimeclock.h"**
pas	**uses yocto_realtimeclock;**
vb	**yocto_realtimeclock.vb**
cs	**yocto_realtimeclock.cs**
java	**import com.yoctopuce.YoctoAPI.YRealTimeClock;**
uwp	**import com.yoctopuce.YoctoAPI.YRealTimeClock;**
py	**from yocto_realtimeclock import ***
php	**require_once('yocto_realtimeclock.php');**
ts	**in HTML: import { YRealTimeClock } from '../../dist/esm/yocto_realtimeclock.js';** **in Node.js: import { YRealTimeClock } from 'yoctolib-cjs/yocto_realtimeclock.js';**
dnp	**import YoctoProxyAPI.YRealTimeClockProxy**
cp	**#include "yocto_realtimeclock_proxy.h"**
vi	**YRealTimeClock.vi**
ml	**import YoctoProxyAPI.YRealTimeClockProxy**

Global functions

YRealTimeClock.FindRealTimeClock(func)

Retrieves a real-time clock for a given identifier.

cpp m pas vb cs java uwp py php ts es dnp

YRealTimeClock.FindRealTimeClockInContext(yctx, func)

Retrieves a real-time clock for a given identifier in a YAPI context.

java uwp ts es

YRealTimeClock.FirstRealTimeClock()

Starts the enumeration of real-time clocks currently accessible.

cpp m pas vb cs java uwp py php ts es

YRealTimeClock.FirstRealTimeClockInContext(yctx)

Starts the enumeration of real-time clocks currently accessible.

java uwp ts es

YRealTimeClock.GetSimilarFunctions()

Enumerates all functions of type RealTimeClock available on the devices currently reachable by the library, and returns their unique hardware ID.

dnp

YRealTimeClock properties

realtimeclock→AdvertisedValue *[read-only]*

Short string representing the current state of the function.

dnp

realtimeclock→DisableHostSync *[writable]*

True if the automatic clock synchronization with host has been disabled, and false otherwise.

dnp

realtimeclock→FriendlyName *[read-only]*

Global identifier of the function in the format MODULE_NAME.FUNCTION_NAME.

dnp

realtimeclock→FunctionId *[read-only]*

Hardware identifier of the real-time clock, without reference to the module.

dnp

realtimeclock→HardwareId *[read-only]*

Unique hardware identifier of the function in the form SERIAL.FUNCTIONID.

dnp

realtimeclock→IsOnline *[read-only]*

Checks if the function is currently reachable.

dnp

realtimeclock→LogicalName *[writable]*

Logical name of the function.

dnp

realtimeclock→SerialNumber *[read-only]*

Serial number of the module, as set by the factory.

dnp

realtimeclock→UtcOffset *[writable]*

Number of seconds between current time and UTC time (time zone).

dnp

YRealTimeClock methods

realtimeclock→clearCache()

Invalidates the cache.

cpp m pas vb cs java py php ts es

realtimeclock→describe()

Returns a short text that describes unambiguously the instance of the real-time clock in the form TYPE(NAME)=SERIAL.FUNCTIONID.

cpp m pas vb cs java py php ts es

realtimeclock→get_advertisedValue()

Returns the current value of the real-time clock (no more than 6 characters).

cpp m pas vb cs java uwp py php ts es dnp cmd

realtimeclock→get_dateTime()

Returns the current time in the form "YYYY/MM/DD hh:mm:ss".

cpp m pas vb cs java uwp py php ts es dnp cmd

realtimeclock→get_disableHostSync()

Returns true if the automatic clock synchronization with host has been disabled, and false otherwise.

cpp m pas vb cs java uwp py php ts es dnp cmd

realtimeclock→get_errorMessage()

Returns the error message of the latest error with the real-time clock.

cpp m pas vb cs java py php ts es

realtimeclock→get_errorType()

Returns the numerical error code of the latest error with the real-time clock.

cpp m pas vb cs java py php ts es

realtimeclock→get_friendlyName()

Returns a global identifier of the real-time clock in the format `MODULE_NAME.FUNCTION_NAME`.

cpp m cs java py php ts es dnp

realtimeclock→get_functionDescriptor()

Returns a unique identifier of type `YFUN_DESCR` corresponding to the function.

cpp m pas vb cs java py php ts es

realtimeclock→get_functionId()

Returns the hardware identifier of the real-time clock, without reference to the module.

cpp m vb cs java py php ts es dnp

realtimeclock→get_hardwareId()

Returns the unique hardware identifier of the real-time clock in the form `SERIAL.FUNCTIONID`.

cpp m vb cs java py php ts es dnp

realtimeclock→get_logicalName()

Returns the logical name of the real-time clock.

cpp m pas vb cs java uwp py php ts es dnp cmd

realtimeclock→get_module()

Gets the `YModule` object for the device on which the function is located.

cpp m pas vb cs java py php ts es dnp

realtimeclock→get_module_async(callback, context)

Gets the `YModule` object for the device on which the function is located (asynchronous version).

realtimeclock→get_serialNumber()

Returns the serial number of the module, as set by the factory.

cpp m pas vb cs java uwp py php ts es dnp cmd

realtimeclock→get_timeSet()

Returns true if the clock has been set, and false otherwise.

cpp m pas vb cs java uwp py php ts es dnp cmd

realtimeclock→get_unixTime()

Returns the current time in Unix format (number of elapsed seconds since Jan 1st, 1970).

cpp m pas vb cs java uwp py php ts es dnp cmd

realtimeclock→get_userData()

Returns the value of the userData attribute, as previously stored using method `set_userData`.

cpp m pas vb cs java py php ts es

realtimeclock→get_utcOffset()

Returns the number of seconds between current time and UTC time (time zone).

cpp m pas vb cs java uwp py php ts es dnp cmd

realtimeclock→isOnline()

Checks if the real-time clock is currently reachable, without raising any error.

cpp m pas vb cs java py php ts es dnp

realtimeclock→isOnline_async(callback, context)

Checks if the real-time clock is currently reachable, without raising any error (asynchronous version).

realtimeclock→isReadOnly()

Indicates whether changes to the function are prohibited or allowed.

cpp m pas vb cs java uwp py php ts es dnp cmd

realtimeclock→load(msValidity)

Preloads the real-time clock cache with a specified validity duration.

cpp m pas vb cs java py php ts es

realtimeclock→loadAttribute(attrName)

Returns the current value of a single function attribute, as a text string, as quickly as possible but without using the cached value.

cpp m pas vb cs java uwp py php ts es dnp

realtimeclock→load_async(msValidity, callback, context)

Preloads the real-time clock cache with a specified validity duration (asynchronous version).

realtimeclock→muteValueCallbacks()

Disables the propagation of every new advertised value to the parent hub.

cpp m pas vb cs java uwp py php ts es dnp cmd

realtimeclock→nextRealTimeClock()

Continues the enumeration of real-time clocks started using `yFirstRealTimeClock()`.

cpp m pas vb cs java uwp py php ts es

realtimeclock→registerValueCallback(callback)

Registers the callback function that is invoked on every change of advertised value.

cpp m pas vb cs java uwp py php ts es

realtimeclock→set_disableHostSync(newval)

Changes the automatic clock synchronization with host working state.

cpp m pas vb cs java uwp py php ts es dnp cmd

realtimeclock→set_logicalName(newval)

Changes the logical name of the real-time clock.

cpp m pas vb cs java uwp py php ts es dnp cmd

realtimeclock→set_unixTime(newval)

Changes the current time.

cpp m pas vb cs java uwp py php ts es dnp cmd

realtimeclock→set_userData(data)

Stores a user context provided as argument in the userData attribute of the function.

cpp m pas vb cs java py php ts es

realtimeclock→set_utcOffset(newval)

Changes the number of seconds between current time and UTC time (time zone).

cpp m pas vb cs java uwp py php ts es dnp cmd

realtimeclock→unmuteValueCallbacks()

Re-enables the propagation of every new advertised value to the parent hub.

cpp m pas vb cs java uwp py php ts es dnp cmd

realtimeclock→wait_async(callback, context)

Waits for all pending asynchronous commands on the module to complete, and invoke the user-provided callback function.

ts es

9.8. Class YWakeUpMonitor

Wake-up monitor control interface, available for instance in the YoctoHub-GSM-4G, the YoctoHub-Wireless-SR, the YoctoHub-Wireless-g or the YoctoHub-Wireless-n

The `YWakeUpMonitor` class handles globally all wake-up sources, as well as automated sleep mode.

In order to use the functions described here, you should include:

es	**in HTML: <script src="../../lib/yocto_wakeupmonitor.js"></script>** **in node.js: require('yoctolib-es2017/yocto_wakeupmonitor.js');**
js	**<script type='text/javascript' src='yocto_wakeupmonitor.js'></script>**
cpp	**#include "yocto_wakeupmonitor.h"**
m	**#import "yocto_wakeupmonitor.h"**
pas	**uses yocto_wakeupmonitor;**
vb	**yocto_wakeupmonitor.vb**
cs	**yocto_wakeupmonitor.cs**
java	**import com.yoctopuce.YoctoAPI.YWakeUpMonitor;**
uwp	**import com.yoctopuce.YoctoAPI.YWakeUpMonitor;**
py	**from yocto_wakeupmonitor import ***
php	**require_once('yocto_wakeupmonitor.php');**
ts	**in HTML: import { YWakeUpMonitor } from '../../dist/esm/yocto_wakeupmonitor.js';** **in Node.js: import { YWakeUpMonitor } from 'yoctolib-cjs/yocto_wakeupmonitor.js';**
dnp	**import YoctoProxyAPI.YWakeUpMonitorProxy**
cp	**#include "yocto_wakeupmonitor_proxy.h"**
vi	**YWakeUpMonitor.vi**
ml	**import YoctoProxyAPI.YWakeUpMonitorProxy**

Global functions

YWakeUpMonitor.FindWakeUpMonitor(func)

Retrieves a wake-up monitor for a given identifier.

cpp m pas vb cs java uwp py php ts es dnp

YWakeUpMonitor.FindWakeUpMonitorInContext(yctx, func)

Retrieves a wake-up monitor for a given identifier in a YAPI context.

java uwp ts es

YWakeUpMonitor.FirstWakeUpMonitor()

Starts the enumeration of wake-up monitors currently accessible.

cpp m pas vb cs java uwp py php ts es

YWakeUpMonitor.FirstWakeUpMonitorInContext(yctx)

Starts the enumeration of wake-up monitors currently accessible.

java uwp ts es

YWakeUpMonitor.GetSimilarFunctions()

Enumerates all functions of type WakeUpMonitor available on the devices currently reachable by the library, and returns their unique hardware ID.

dnp

`YWakeUpMonitor` properties

wakeupmonitor→AdvertisedValue *[read-only]*

Short string representing the current state of the function.

dnp

wakeupmonitor→FriendlyName *[read-only]*

Global identifier of the function in the format MODULE_NAME.FUNCTION_NAME.

dnp

wakeupmonitor→FunctionId *[read-only]*

Hardware identifier of the wake-up monitor, without reference to the module.

dnp

wakeupmonitor→HardwareId *[read-only]*

Unique hardware identifier of the function in the form SERIAL.FUNCTIONID.

dnp

wakeupmonitor→IsOnline *[read-only]*

Checks if the function is currently reachable.

dnp

wakeupmonitor→LogicalName *[writable]*

Logical name of the function.

dnp

wakeupmonitor→NextWakeUp *[writable]*

Next scheduled wake up date/time (UNIX format).

dnp

wakeupmonitor→PowerDuration *[writable]*

Maximal wake up time (in seconds) before automatically going to sleep.

dnp

wakeupmonitor→SerialNumber *[read-only]*

Serial number of the module, as set by the factory.

dnp

YWakeUpMonitor methods

wakeupmonitor→clearCache()

Invalidates the cache.

cpp m pas vb cs java py php ts es

wakeupmonitor→describe()

Returns a short text that describes unambiguously the instance of the wake-up monitor in the form TYPE(NAME)=SERIAL.FUNCTIONID.

cpp m pas vb cs java py php ts es

wakeupmonitor→get_advertisedValue()

Returns the current value of the wake-up monitor (no more than 6 characters).

cpp m pas vb cs java uwp py php ts es dnp cmd

wakeupmonitor→get_errorMessage()

Returns the error message of the latest error with the wake-up monitor.

cpp m pas vb cs java py php ts es

wakeupmonitor→get_errorType()

Returns the numerical error code of the latest error with the wake-up monitor.

cpp m pas vb cs java py php ts es

wakeupmonitor→get_friendlyName()

Returns a global identifier of the wake-up monitor in the format MODULE_NAME.FUNCTION_NAME.

cpp m cs java py php ts es dnp

wakeupmonitor→get_functionDescriptor()

Returns a unique identifier of type YFUN_DESCR corresponding to the function.

cpp m pas vb cs java py php ts es

wakeupmonitor→get_functionId()

Returns the hardware identifier of the wake-up monitor, without reference to the module.

cpp m vb cs java py php ts es dnp

wakeupmonitor→get_hardwareId()

Returns the unique hardware identifier of the wake-up monitor in the form SERIAL.FUNCTIONID.

cpp m vb cs java py php ts es dnp

wakeupmonitor→get_logicalName()

Returns the logical name of the wake-up monitor.

cpp m pas vb cs java uwp py php ts es dnp cmd

wakeupmonitor→get_module()

Gets the YModule object for the device on which the function is located.

cpp m pas vb cs java py php ts es dnp

wakeupmonitor→get_module_async(callback, context)

Gets the YModule object for the device on which the function is located (asynchronous version).

wakeupmonitor→get_nextWakeUp()

Returns the next scheduled wake up date/time (UNIX format).

cpp m pas vb cs java uwp py php ts es dnp cmd

wakeupmonitor→get_powerDuration()

Returns the maximal wake up time (in seconds) before automatically going to sleep.

cpp m pas vb cs java uwp py php ts es dnp cmd

wakeupmonitor→get_serialNumber()

Returns the serial number of the module, as set by the factory.

cpp m pas vb cs java uwp py php ts es dnp cmd

wakeupmonitor→get_sleepCountdown()

Returns the delay before the next sleep period.

cpp m pas vb cs java uwp py php ts es dnp cmd

wakeupmonitor→get_userData()

Returns the value of the userData attribute, as previously stored using method set_userData.

cpp m pas vb cs java py php ts es

wakeupmonitor→get_wakeUpReason()

Returns the latest wake up reason.

cpp m pas vb cs java uwp py php ts es dnp cmd

wakeupmonitor→get_wakeUpState()

Returns the current state of the monitor.

cpp m pas vb cs java uwp py php ts es dnp cmd

wakeupmonitor→isOnline()

Checks if the wake-up monitor is currently reachable, without raising any error.

cpp m pas vb cs java py php ts es dnp

wakeupmonitor→isOnline_async(callback, context)

Checks if the wake-up monitor is currently reachable, without raising any error (asynchronous version).

wakeupmonitor→isReadOnly()

Indicates whether changes to the function are prohibited or allowed.

cpp m pas vb cs java uwp py php ts es dnp cmd

wakeupmonitor→load(msValidity)

Preloads the wake-up monitor cache with a specified validity duration.

cpp m pas vb cs java py php ts es

wakeupmonitor→loadAttribute(attrName)

Returns the current value of a single function attribute, as a text string, as quickly as possible but without using the cached value.

cpp m pas vb cs java uwp py php ts es dnp

wakeupmonitor→load_async(msValidity, callback, context)

Preloads the wake-up monitor cache with a specified validity duration (asynchronous version).

wakeupmonitor→muteValueCallbacks()

Disables the propagation of every new advertised value to the parent hub.

cpp m pas vb cs java uwp py php ts es dnp cmd

wakeupmonitor→nextWakeUpMonitor()

Continues the enumeration of wake-up monitors started using `yFirstWakeUpMonitor()`.

cpp m pas vb cs java uwp py php ts es

wakeupmonitor→registerValueCallback(callback)

Registers the callback function that is invoked on every change of advertised value.

cpp m pas vb cs java uwp py php ts es

wakeupmonitor→resetSleepCountDown()

Resets the sleep countdown.

cpp m pas vb cs java uwp py php ts es dnp cmd

wakeupmonitor→set_logicalName(newval)

Changes the logical name of the wake-up monitor.

cpp m pas vb cs java uwp py php ts es dnp cmd

wakeupmonitor→set_nextWakeUp(newval)

Changes the days of the week when a wake up must take place.

cpp m pas vb cs java uwp py php ts es dnp cmd

wakeupmonitor→set_powerDuration(newval)

Changes the maximal wake up time (seconds) before automatically going to sleep.

cpp m pas vb cs java uwp py php ts es dnp cmd

wakeupmonitor→set_sleepCountdown(newval)

Changes the delay before the next sleep period.

cpp m pas vb cs java uwp py php ts es dnp cmd

wakeupmonitor→set_userData(data)

Stores a user context provided as argument in the userData attribute of the function.

cpp m pas vb cs java py php ts es

wakeupmonitor→sleep(secBeforeSleep)

Goes to sleep until the next wake up condition is met, the RTC time must have been set before calling this function.

cpp m pas vb cs java uwp py php ts es dnp cmd

wakeupmonitor→sleepFor(secUntilWakeUp, secBeforeSleep)

Goes to sleep for a specific duration or until the next wake up condition is met, the RTC time must have been set before calling this function.

cpp m pas vb cs java uwp py php ts es dnp cmd

wakeupmonitor→sleepUntil(wakeUpTime, secBeforeSleep)

Go to sleep until a specific date is reached or until the next wake up condition is met, the RTC time must have been set before calling this function.

cpp m pas vb cs java uwp py php ts es dnp cmd

wakeupmonitor→unmuteValueCallbacks()

Re-enables the propagation of every new advertised value to the parent hub.

cpp m pas vb cs java uwp py php ts es dnp cmd

wakeupmonitor→wait_async(callback, context)

Waits for all pending asynchronous commands on the module to complete, and invoke the user-provided callback function.

ts es

wakeupmonitor→wakeUp()

Forces a wake up.

cpp m pas vb cs java uwp py php ts es dnp cmd

9.9. Class YWakeUpSchedule

Wake up schedule control interface, available for instance in the YoctoHub-GSM-4G, the YoctoHub-Wireless-SR, the YoctoHub-Wireless-g or the YoctoHub-Wireless-n

The `YWakeUpSchedule` class implements a wake up condition. The wake up time is specified as a set of months and/or days and/or hours and/or minutes when the wake up should happen.

In order to use the functions described here, you should include:

es	**in HTML: <script src="../../lib/yocto_wakeupschedule.js"></script>** **in node.js: require('yoctolib-es2017/yocto_wakeupschedule.js');**
js	**<script type='text/javascript' src='yocto_wakeupschedule.js'></script>**
cpp	**#include "yocto_wakeupschedule.h"**
m	**#import "yocto_wakeupschedule.h"**
pas	**uses yocto_wakeupschedule;**
vb	**yocto_wakeupschedule.vb**
cs	**yocto_wakeupschedule.cs**
java	**import com.yoctopuce.YoctoAPI.YWakeUpSchedule;**
uwp	**import com.yoctopuce.YoctoAPI.YWakeUpSchedule;**
py	**from yocto_wakeupschedule import ***
php	**require_once('yocto_wakeupschedule.php');**
ts	**in HTML: import { YWakeUpSchedule } from '../../dist/esm/yocto_wakeupschedule.js';** **in Node.js: import { YWakeUpSchedule } from 'yoctolib-cjs/yocto_wakeupschedule.js';**
dnp	**import YoctoProxyAPI.YWakeUpScheduleProxy**
cp	**#include "yocto_wakeupschedule_proxy.h"**
vi	**YWakeUpSchedule.vi**
ml	**import YoctoProxyAPI.YWakeUpScheduleProxy**

Global functions

YWakeUpSchedule.FindWakeUpSchedule(func)

Retrieves a wake up schedule for a given identifier.

cpp m pas vb cs java uwp py php ts es dnp

YWakeUpSchedule.FindWakeUpScheduleInContext(yctx, func)

Retrieves a wake up schedule for a given identifier in a YAPI context.

java uwp ts es

YWakeUpSchedule.FirstWakeUpSchedule()

Starts the enumeration of wake up schedules currently accessible.

cpp m pas vb cs java uwp py php ts es

YWakeUpSchedule.FirstWakeUpScheduleInContext(yctx)

Starts the enumeration of wake up schedules currently accessible.

java uwp ts es

YWakeUpSchedule.GetSimilarFunctions()

Enumerates all functions of type WakeUpSchedule available on the devices currently reachable by the library, and returns their unique hardware ID.

dnp

`YWakeUpSchedule` properties

wakeupschedule→AdvertisedValue *[read-only]*

Short string representing the current state of the function.

dnp

wakeupschedule→FriendlyName *[read-only]*

Global identifier of the function in the format `MODULE_NAME.FUNCTION_NAME`.

dnp

wakeupschedule→FunctionId *[read-only]*

Hardware identifier of the wake up schedule, without reference to the module.

dnp

wakeupschedule→HardwareId *[read-only]*

Unique hardware identifier of the function in the form `SERIAL.FUNCTIONID`.

dnp

wakeupschedule→Hours *[writable]*

Hours scheduled for wake up.

dnp

wakeupschedule→IsOnline *[read-only]*

Checks if the function is currently reachable.

dnp

wakeupschedule→LogicalName *[writable]*

Logical name of the function.

dnp

wakeupschedule→MinutesA *[writable]*

Minutes in the 00-29 interval of each hour scheduled for wake up.

dnp

wakeupschedule→MinutesB *[writable]*

Minutes in the 30-59 interval of each hour scheduled for wake up.

dnp

wakeupschedule→MonthDays *[writable]*

Days of the month scheduled for wake up.

dnp

wakeupschedule→Months *[writable]*

Months scheduled for wake up.

dnp

wakeupschedule→NextOccurence *[read-only]*

Date/time (seconds) of the next wake up occurrence.

dnp

wakeupschedule→SecondsBefore *[writable]*

Number of seconds to anticipate wake-up time to allow the system to power-up.

dnp

wakeupschedule→SerialNumber *[read-only]*

Serial number of the module, as set by the factory.

dnp

wakeupschedule→WeekDays *[writable]*

Days of the week scheduled for wake up.

dnp

YWakeUpSchedule methods

wakeupschedule→clearCache()

Invalidates the cache.

cpp m pas vb cs java py php ts es

wakeupschedule→describe()

Returns a short text that describes unambiguously the instance of the wake up schedule in the form TYPE(NAME)=SERIAL.FUNCTIONID.

cpp m pas vb cs java py php ts es

wakeupschedule→get_advertisedValue()

Returns the current value of the wake up schedule (no more than 6 characters).

cpp m pas vb cs java uwp py php ts es dnp cmd

wakeupschedule→get_errorMessage()

Returns the error message of the latest error with the wake up schedule.

cpp m pas vb cs java py php ts es

wakeupschedule→get_errorType()

Returns the numerical error code of the latest error with the wake up schedule.

cpp m pas vb cs java py php ts es

wakeupschedule→get_friendlyName()

Returns a global identifier of the wake up schedule in the format MODULE_NAME.FUNCTION_NAME.

cpp m cs java py php ts es dnp

wakeupschedule→get_functionDescriptor()

Returns a unique identifier of type YFUN_DESCR corresponding to the function.

cpp m pas vb cs java py php ts es

wakeupschedule→get_functionId()

Returns the hardware identifier of the wake up schedule, without reference to the module.

cpp m vb cs java py php ts es dnp

wakeupschedule→get_hardwareId()

Returns the unique hardware identifier of the wake up schedule in the form SERIAL.FUNCTIONID.

cpp m vb cs java py php ts es dnp

wakeupschedule→get_hours()

Returns the hours scheduled for wake up.

cpp m pas vb cs java uwp py php ts es dnp cmd

wakeupschedule→get_logicalName()

Returns the logical name of the wake up schedule.

cpp m pas vb cs java uwp py php ts es dnp cmd

wakeupschedule→get_minutes()

Returns all the minutes of each hour that are scheduled for wake up.

cpp m pas vb cs java uwp py php ts es dnp cmd

wakeupschedule→get_minutesA()

Returns the minutes in the 00-29 interval of each hour scheduled for wake up.

cpp m pas vb cs java uwp py php ts es dnp cmd

wakeupschedule→get_minutesB()

Returns the minutes in the 30-59 interval of each hour scheduled for wake up.

cpp m pas vb cs java uwp py php ts es dnp cmd

wakeupschedule→get_module()

Gets the `YModule` object for the device on which the function is located.

cpp m pas vb cs java py php ts es dnp

wakeupschedule→get_module_async(callback, context)

Gets the `YModule` object for the device on which the function is located (asynchronous version).

wakeupschedule→get_monthDays()

Returns the days of the month scheduled for wake up.

cpp m pas vb cs java uwp py php ts es dnp cmd

wakeupschedule→get_months()

Returns the months scheduled for wake up.

cpp m pas vb cs java uwp py php ts es dnp cmd

wakeupschedule→get_nextOccurence()

Returns the date/time (seconds) of the next wake up occurrence.

cpp m pas vb cs java uwp py php ts es dnp cmd

wakeupschedule→get_secondsBefore()

Returns the number of seconds to anticipate wake-up time to allow the system to power-up.

cpp m pas vb cs java uwp py php ts es dnp cmd

wakeupschedule→get_serialNumber()

Returns the serial number of the module, as set by the factory.

cpp m pas vb cs java uwp py php ts es dnp cmd

wakeupschedule→get_userData()

Returns the value of the userData attribute, as previously stored using method `set_userData`.

cpp m pas vb cs java py php ts es

wakeupschedule→get_weekDays()

Returns the days of the week scheduled for wake up.

cpp m pas vb cs java uwp py php ts es dnp cmd

wakeupschedule→isOnline()

Checks if the wake up schedule is currently reachable, without raising any error.

cpp m pas vb cs java py php ts es dnp

wakeupschedule→isOnline_async(callback, context)

Checks if the wake up schedule is currently reachable, without raising any error (asynchronous version).

wakeupschedule→isReadOnly()

Indicates whether changes to the function are prohibited or allowed.

cpp m pas vb cs java uwp py php ts es dnp cmd

wakeupschedule→load(msValidity)

Preloads the wake up schedule cache with a specified validity duration.

cpp m pas vb cs java py php ts es

wakeupschedule→loadAttribute(attrName)

Returns the current value of a single function attribute, as a text string, as quickly as possible but without using the cached value.

cpp m pas vb cs java uwp py php ts es dnp

wakeupschedule→load_async(msValidity, callback, context)

Preloads the wake up schedule cache with a specified validity duration (asynchronous version).

wakeupschedule→muteValueCallbacks()

Disables the propagation of every new advertised value to the parent hub.

cpp m pas vb cs java uwp py php ts es dnp cmd

wakeupschedule→nextWakeUpSchedule()

Continues the enumeration of wake up schedules started using `yFirstWakeUpSchedule()`.

cpp m pas vb cs java uwp py php ts es

wakeupschedule→registerValueCallback(callback)

Registers the callback function that is invoked on every change of advertised value.

cpp m pas vb cs java uwp py php ts es

wakeupschedule→set_hours(newval)

Changes the hours when a wake up must take place.

cpp m pas vb cs java uwp py php ts es dnp cmd

wakeupschedule→set_logicalName(newval)

Changes the logical name of the wake up schedule.

cpp m pas vb cs java uwp py php ts es dnp cmd

wakeupschedule→set_minutes(bitmap)

Changes all the minutes where a wake up must take place.

cpp m pas vb cs java uwp py php ts es dnp cmd

wakeupschedule→set_minutesA(newval)

Changes the minutes in the 00-29 interval when a wake up must take place.

cpp m pas vb cs java uwp py php ts es dnp cmd

wakeupschedule→set_minutesB(newval)

Changes the minutes in the 30-59 interval when a wake up must take place.

cpp m pas vb cs java uwp py php ts es dnp cmd

wakeupschedule→set_monthDays(newval)

Changes the days of the month when a wake up must take place.

cpp m pas vb cs java uwp py php ts es dnp cmd

wakeupschedule→set_months(newval)

Changes the months when a wake up must take place.

cpp m pas vb cs java uwp py php ts es dnp cmd

wakeupschedule→set_secondsBefore(newval)

Changes the number of seconds to anticipate wake-up time to allow the system to power-up.

cpp m pas vb cs java uwp py php ts es dnp cmd

wakeupschedule→set_userData(data)

Stores a user context provided as argument in the userData attribute of the function.

cpp m pas vb cs java py php ts es

wakeupschedule→set_weekDays(newval)

Changes the days of the week when a wake up must take place.

cpp m pas vb cs java uwp py php ts es dnp cmd

wakeupschedule→unmuteValueCallbacks()

Re-enables the propagation of every new advertised value to the parent hub.

cpp m pas vb cs java uwp py php ts es dnp cmd

wakeupschedule→wait_async(callback, context)

Waits for all pending asynchronous commands on the module to complete, and invoke the user-provided callback function.

ts es

10. Troubleshooting

10.1. Where to start?

If it is the first time that you use a Yoctopuce module and you do not really know where to start, have a look at the Yoctopuce blog. There is a section dedicated to beginners [1].

10.2. Programming examples don't seem to work

Most of Yoctopuce API programming examples are command line programs and require some parameters to work properly. You have to start them from your operating system command prompt, or configure your IDE to run them with the proper parameters. [2].

10.3. Linux and USB

To work correctly under Linux, the library 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 the library 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[3] archive, there are two rule examples which you can use as a basis.

[1] see: http://www.yoctopuce.com/EN/blog_by_categories/for-the-beginners
[2] see: http://www.yoctopuce.com/EN/article/about-programming-examples
[3] http://www.yoctopuce.com/FR/virtualhub.php

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

10.4. ARM Platforms: HF and EL

There are two main flavors of executable on ARM: HF (Hard Float) binaries, and EL (EABI Little Endian) binaries. These two families are not compatible at all. The compatibility of a given ARM platform with of one of these two families depends on the hardware and on the OS build. ArmHL and ArmEL compatibility problems are quite difficult to detect. Most of the time, the OS itself is unable to make a difference between an HF and an EL executable and will return meaningless messages when you try to use the wrong type of binary.

All pre-compiled Yoctopuce binaries are provided in both formats, as two separate ArmHF et ArmEL executables. If you do not know what family your ARM platform belongs to, just try one executable from each family.

10.5. Powered module but invisible for the OS

If your YoctoHub-GSM-4G is connected by USB, if its blue led is on, but if the operating system cannot see the module, check that you are using a true USB cable with data wires, and not a charging cable. Charging cables have only power wires.

10.6. Another process named xxx is already using yAPI

If when initializing the Yoctopuce API, you obtain the "*Another process named xxx is already using yAPI*" error message, it means that another application is already using Yoctopuce USB modules. On a single machine only one process can access Yoctopuce modules by USB at a time. You can easily work around this limitation by using VirtualHub and the network mode [4].

10.7. Disconnections, erratic behavior

If your YoctoHub-GSM-4G behaves erratically and/or disconnects itself from the USB bus without apparent reason, check that it is correctly powered. Avoid cables with a length above 2 meters. If needed, insert a powered USB hub [5] [6].

[4] see: http://www.yoctopuce.com/EN/article/error-message-another-process-is-already-using-yapi
[5] see: http://www.yoctopuce.com/EN/article/usb-cables-size-matters
[6] see: http://www.yoctopuce.com/EN/article/how-many-usb-devices-can-you-connect

10.8. After a failed firmware update, the device stopped working

If a firmware update of your YoctoHub-GSM-4G fails, it is possible that the module is no longer working. If this is the case, plug in your module while holding down the Yocto-Button. The Yocto-LED should light up brightly and remain steady. Release the button. Your YoctoHub-GSM-4G should then appear at the bottom of the VirtualHub user interface as a module waiting to be flashed. This operation also reverts the module to its factory configuration.

10.9. Registering VirtualHub disconnects another instance

If, when performing a call to RegisterHub() with a VirtualHub address, another previously registered VirtualHub disconnects, make sure the machine running these VirtualHubs do not have the same *Hostname*. Same *Hostname* can happen very easily when the operating system is installed from a monolithic image, Raspberry Pi are the best example. The Yoctopuce API uses serial numbers to communicate with devices and VirtualHub serial numbers are created on the fly based the *hostname* of the machine running VirtualHub.

10.10. Dropped commands

If, after sending a bunch of commands to a Yoctopuce device, you are under the impression that the last ones have been ignored, a typical example is a quick and dirty program meant to configure a device, make sure you used a YAPI.FreeAPI() at the end of the program. Commands are sent to Yoctopuce modules asynchronously thanks to a background thread. When the main program terminates, that thread is killed no matter if some command are left to be sent. However API.FreeAPI () waits until there is no more command to send before freeing the API resources and returning.

10.11. Can't contact sub devices by USB

The point of the YoctoHub-GSM-4G is to provide network access to connected sub-devices, it does not behave like a common USB hub. The YoctoHub-GSM-4G's USB port is just meant for power and Hub configuration. Access to sub device is only possible through a network connection.

10.12. Network Readiness stuck at 3- LAN ready

Check your outbound Internet connectivity and make sure you don't have an invalid callback set in the YoctoHub-GSM-4G configuration

10.13. Damaged device

Yoctopuce strives to reduce the production of electronic waste. If you believe that your YoctoHub-GSM-4G is not working anymore, start by contacting Yoctopuce support by e-mail to diagnose the failure. Even if you know that the device was damaged by mistake, Yoctopuce engineers might be able to repair it, and thus avoid creating electronic waste.

Waste Electrical and Electronic Equipment (WEEE) If you really want to get rid of your YoctoHub-GSM-4G, do not throw it away in a trash bin but bring it to your local WEEE recycling point. In this way, it will be disposed properly by a specialized WEEE recycling center.

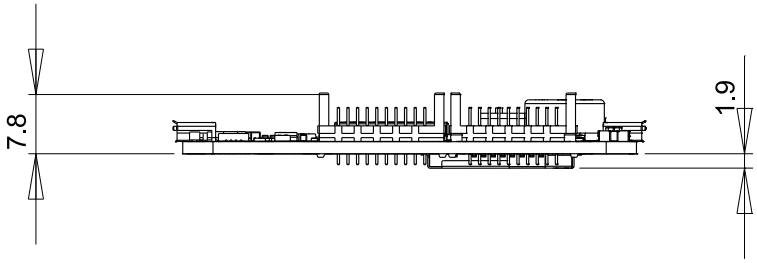
11. Characteristics

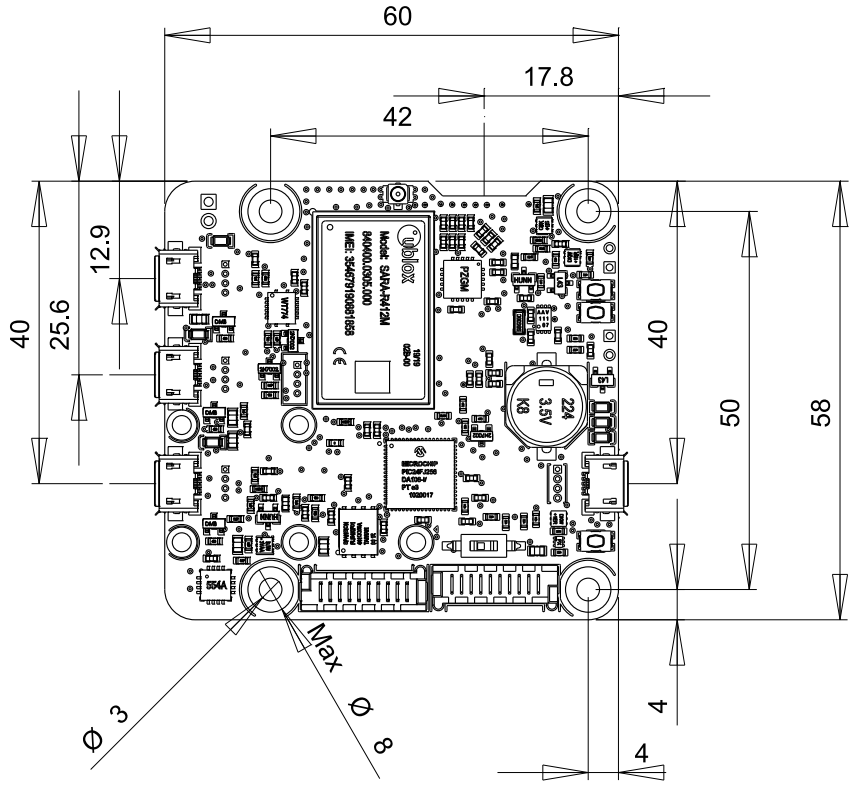
You can find below a summary of the main technical characteristics of your YoctoHub-GSM-4G module.

Product ID	YHUBGSM5
Hardware release†	
USB connector	micro-B
Thickness	9.5 mm
Width	58 mm
Length	60 mm
Weight	33.2 g
Chipset	u-Blox SARA-R412M-02B
Protection class, according to IEC 61140	class III
Normal operating temperature	5...40 °C
Extended operating temperature‡	-20...70 °C
USB consumption	50 mA
RoHS compliance	RoHS III (2011/65/UE+2015/863)
USB Vendor ID	0x24E0
USB Device ID	0x0034
Suggested enclosure	YoctoBox-HubWlan-Transp
Harmonized tariff code	9032.9000
Made in	Switzerland

† These specifications are for the current hardware revision. Specifications for earlier revisions may differ.

‡ The extended temperature range is defined based on components specifications and has been tested during a limited duration (1h). When using the device in harsh environments for a long period of time, we strongly advise to run extensive tests before going to production.





All dimensions are in mm
Toutes les dimensions sont en mm

YoctoHub-GSM-4G

A4

Scale
1:1
Echelle