YoctoHub-Wireless, User's guide

Table of contents

1. Introduction

The YoctoHub-Wireless module is a 60x58mm module enabling you to control other Yoctopuce modules through a wireless network connection. Seen from the outside, this module behaves exactly like a standard computer running a *VirtualHub*[1]: same interface, same functionalities.

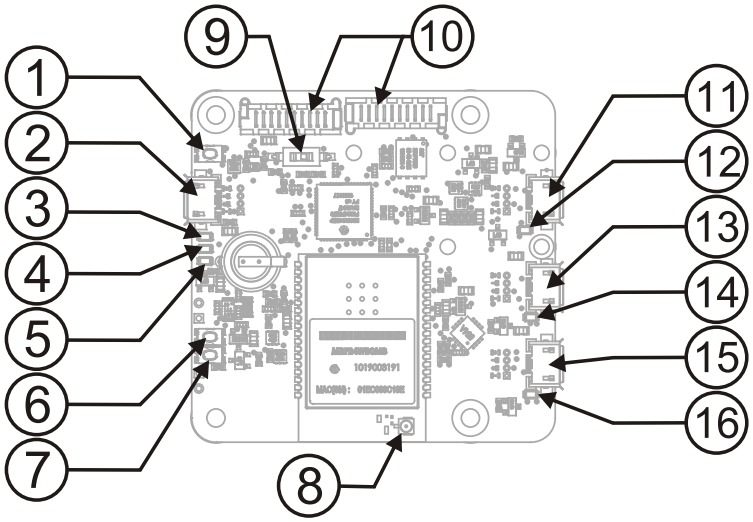
The YoctoHub-Wireless is designed to be easily deployed and to not require any specific maintenance. In the opposite to a mini-computer, it does not have a complex operating system. Some simple settings allow you to use it in many kinds of network environments. These settings can be modified manually or automatically through USB. Therefore, the YoctoHub-Wireless is much more suited to industrialization than a mini-computer. However, you cannot run additional software written by the user on the YoctoHub-Wireless.

The YoctoHub-Wireless 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-Wireless.

Yoctopuce thanks you for buying this YoctoHub-Wireless and sincerely hopes that you will be satisfied with it. The Yoctopuce engineers have put a large amount of effort to ensure that your YoctoHub-Wireless 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] http://www.yoctopuce.com/EN/virtualhub.php
[2] support@yoctopuce.com

2. Presentation



1: Yocto-button
2: Control and power USB port
3: Yocto-led
4: Overload led
5: Network transfer led
6: Wake up button
7: Sleep button
8: Antenna connector
9: Sleep neutralization
10: Back connection
11: Down port 1
12: Down port 1 led
13: Down port 2
14: Down port 2 led
15: Down port 3
16: Down port 3 led

2.1. The YoctoHub-Wireless components

Serial number

Each Yocto-module has a unique serial number assigned to it at the factory. For YoctoHub-Wireless modules, this number starts with YHUBWLN1. 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-Wireless and the modules connected to it with a simple USB charger. This port also allows you to control the YoctoHub-Wireless by USB, exactly like you can do it with a classic Yoctopuce module. It is particularly useful when you want to configure the YoctoHub-Wireless 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 a *VirtualHub*. Note that the protocol used between the YoctoHub-Wireless and the USB modules is not USB but a lighter proprietary protocol. Therefore, the YoctoHub-Wireless 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.

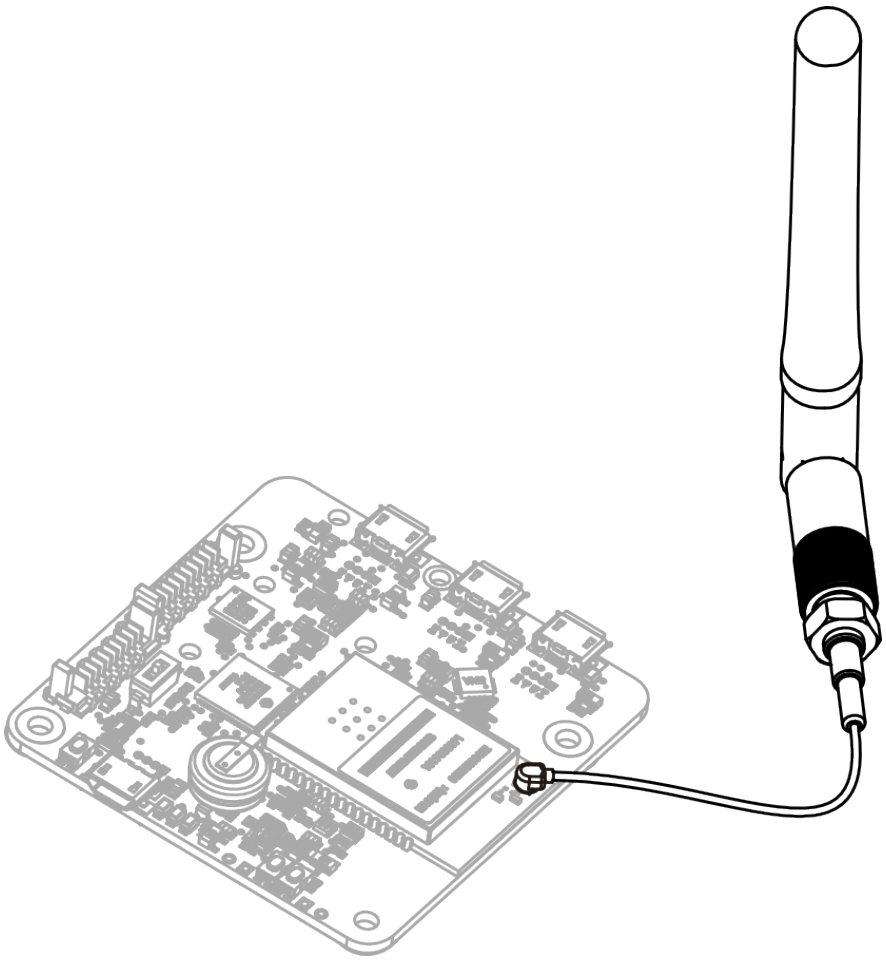
Warning: the USB connectors are simply soldered in surface and can be pulled out if the USB plug acts as a lever. In this case, if the tracks stayed in position, the connector can be soldered back with a good iron and flux to avoid bridges. Alternatively, you can solder a USB cable directly in the 1.27mm-spaced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

Antenna connector

The YoctoHub-Wireless includes an ultra miniature coaxial antenna connector (UFL). Take care of the UFL connector. It is fragile and is not designed to support many connection/deconnection cycles. The YoctoHub-Wireless is sold with a small UFL cable to RP-SMA socket (reverse polarity SMA: threaded on the outside with a plug in the center) and a corresponding RP-SMA plug antenna

[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

(threaded on the inside, jack in the center). You can use another antenna of your choice, as long as it is designed for the 2.4 GHz frequency range and it has the correct connector. Beware of the different variants of SMA connectors: there are antennas for each of the four combinations SMA/RP-SMA and plug/socket. Only an RP-SMA plug antenna can be used with the provided antenna cable. Beware also that using a high-gain antenna may drive you to emit a signal stronger than the authorized norm in your country.



Antenna connection

Overload led

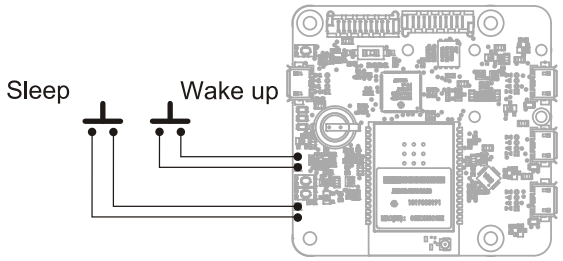
The YoctoHub-Wireless continuously monitors its power consumption. If it detects a global consumption of more that 2A, following an overload on one of the down ports for example, it automatically disables all the down ports and lights the overload led. To isolate the source of the issue, you can reactivate 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-Wireless 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

Usually, the YoctoHub-Wireless consumes about 0.5 Watt (110mA), to which you must add the connected module 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 functionality is very useful to build measuring installations working on a battery. When the YoctoHub-Wireless 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-Wireless circuit. You can find there two pairs of contacts which enable you to shunt these two buttons.



Sleep and wake up buttons deviation.

The YoctoHub-Wireless 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.

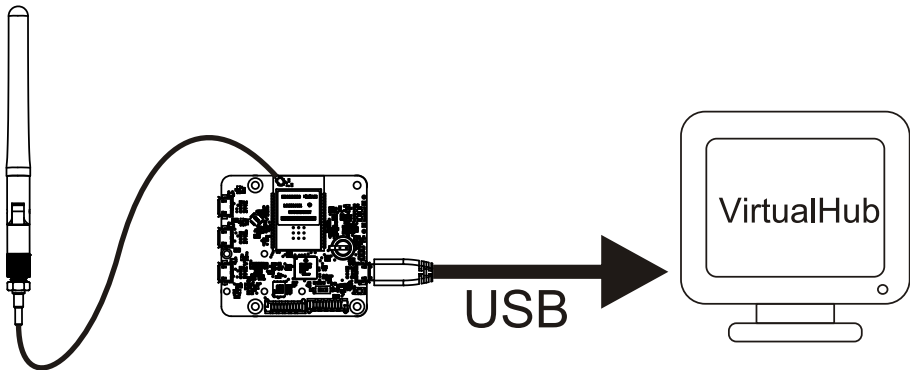
3. First steps

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

3.1. Manual configuration

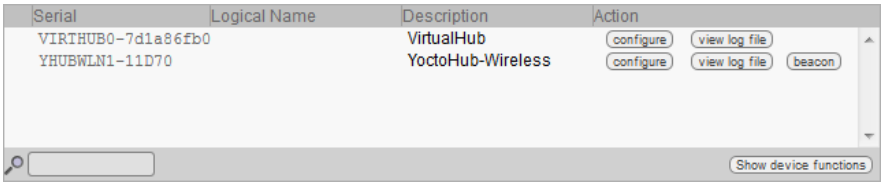
If you must use a specific network configuration, you can configure your YoctoHub-Wireless through its USB control port, by using the *VirtualHub*[1].

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



Configuration: connecting your YoctoHub-Wireless by USB to a computer

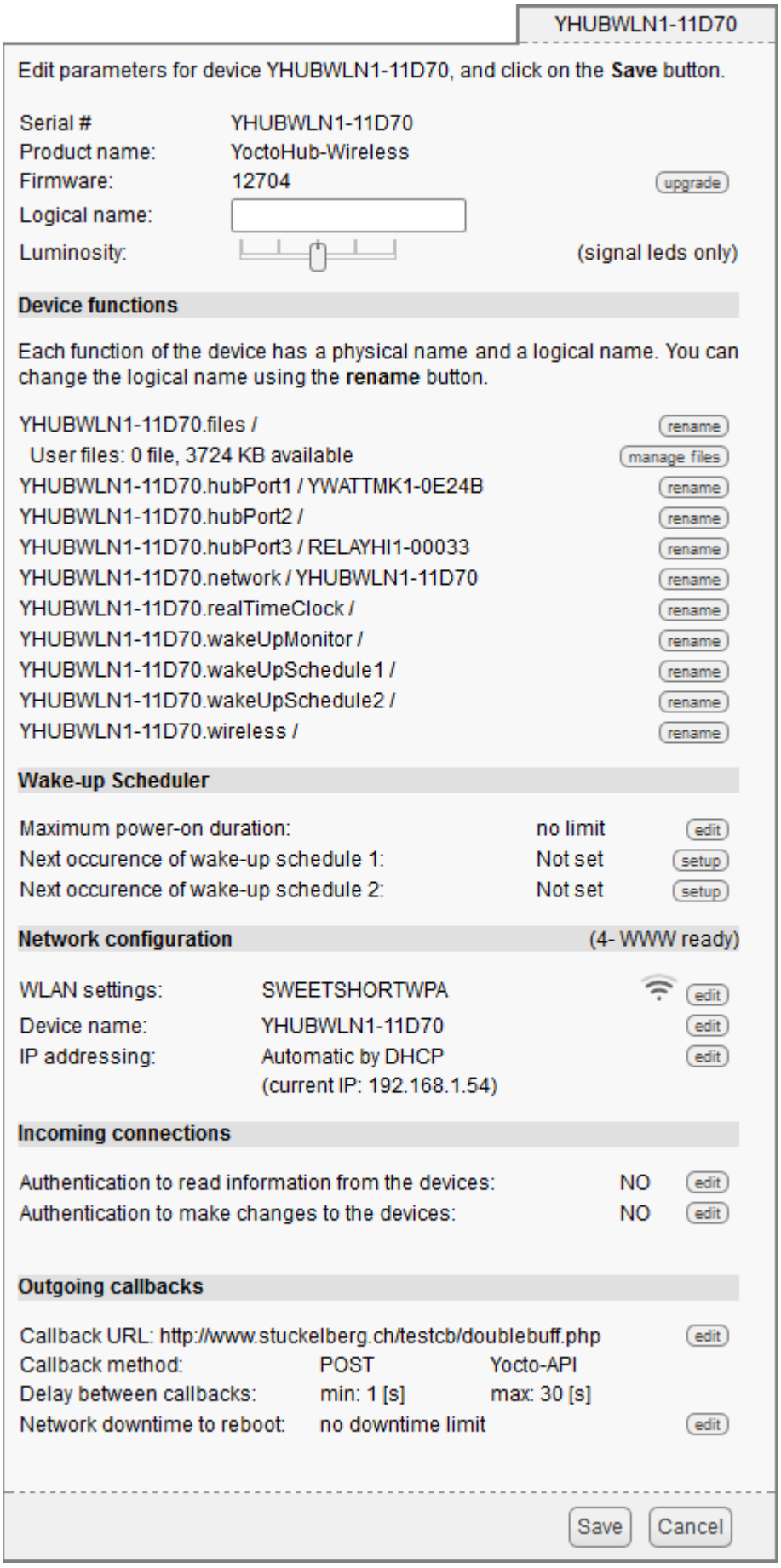
Launch your preferred browser[2] on the URL of your *VirtualHub*. It usually is *http://127.0.0.1:4444*. You obtain the list of Yoctopuce modules connected by USB, among which your YoctoHub-Wireless.



List of Yoctopuce modules connected by USB to your computer, among which your YoctoHub-Wireless

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

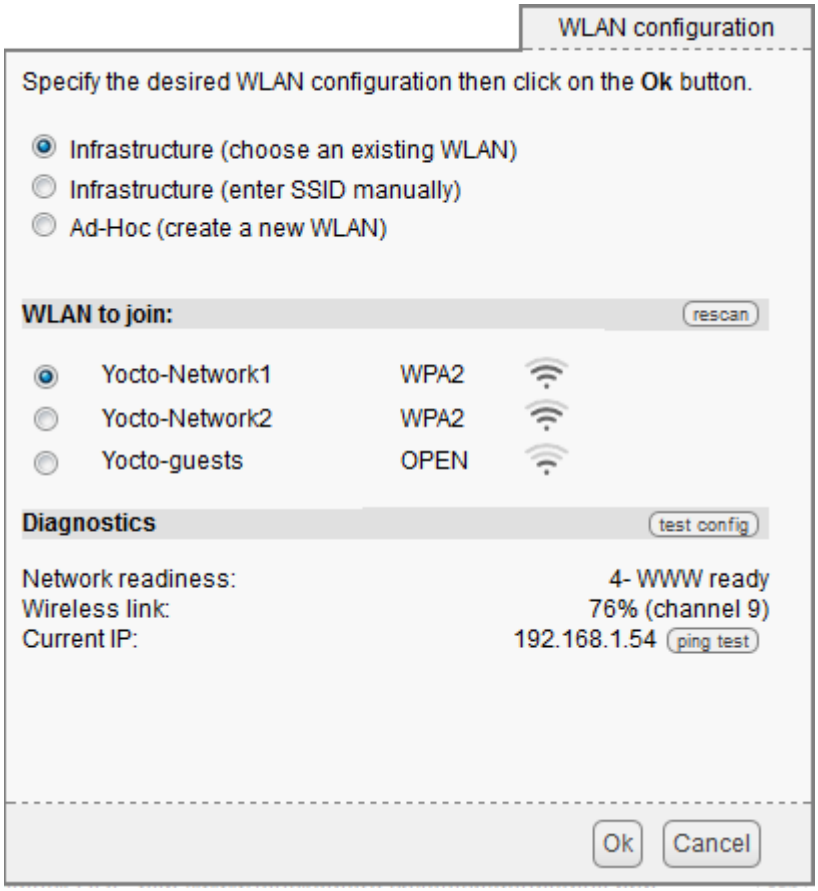
[1] http://www.yoctopuce.com/EN/virtualhub.php
[2] Except Opera



YoctoHub-Wireless module configuration window

Connection to the wireless network

You must first configure your YoctoHub-Wireless to enable it to connect itself to your wifi network. To do so, click on the **edit** button corresponding to **WLAN settings** in the **Network configuration** section. The configuration window of the wireless network shows up:



Wireless network configuration window.

You can then decide if you wish to connect your YoctoHub-Wireless to an existing network, or if you would rather manually enter the SSID of network you wish to use.

You can also configure the YoctoHub-Wireless for it to generate its own wireless network in *ad-hoc* mode. You can then connect yourself directly on the YoctoHub-Wireless without having to go through an infrastructure server (access point). However, be aware that the *ad-hoc* mode has important limitations compared to a real wifi network. In particular, devices under Android cannot connect themselves to it.

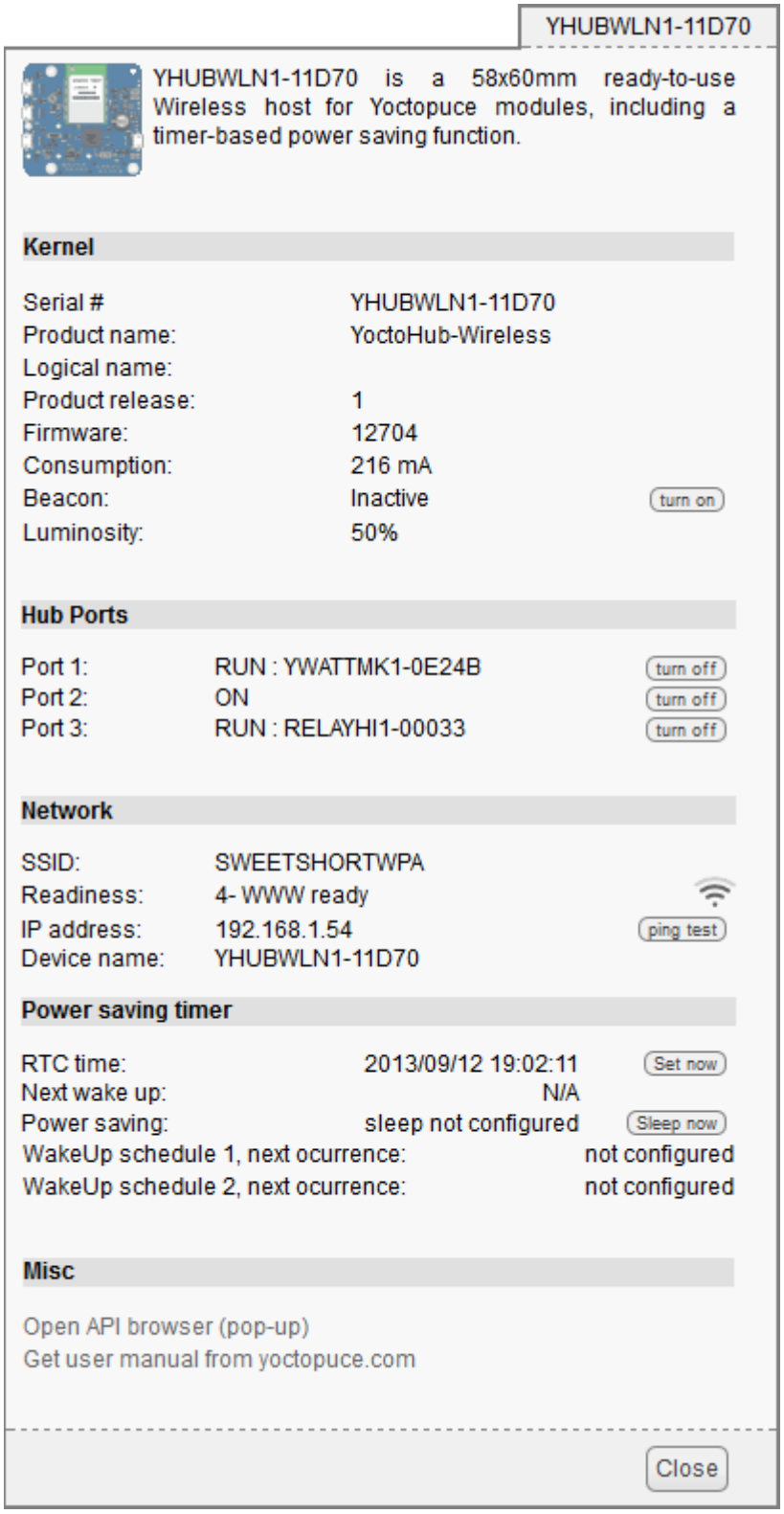
When you have set the wireless network parameters, and possibly tested them, you can click on the **OK** button to close this configuration window and go back to the main configuration window.

If needed, you can also configure which IP address must be assigned to the YoctoHub-Wireless. To do so, click on the **edit** button opposite to the **IP addressing** line in the main window.

You can then choose between a DHCP assigned IP address or a fixed IP address for your YoctoHub-Wireless module. The DHCP address is recommended in so much as this functionality is supported by most ADSL routers (its the default configuration). If you do not know what a DHCP server is but are used to connect machines on your network and to see them work without any problem, do not touch anything.

You can also choose the network name of your YoctoHub-Wireless. You can then access your YoctoHub-Wireless by using this name rather than its IP address. When the network part is configured, click on the *Save* button to save your changes and close the configuration window. These modifications are saved in the persistent memory of the YoctoHub-Wireless, they are kept even after the module has been powered off.

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



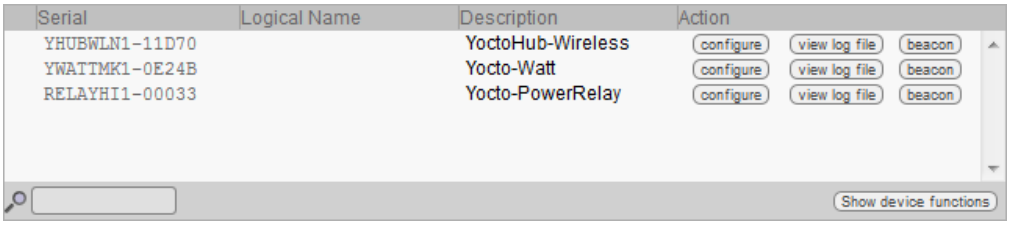
The YoctoHub-Wireless properties

This window contains a section indicating the state of the YoctoHub-Wireless network part. You can find there its MAC address, current IP address, and network name. This section also provides the state of the network connection. Possible states are:

- 0- search for link: The module is searching for a connection with the network. If this state persists, the sought wifi network is most likely not in the neighborhood.
- 1- network exists: The sought wifi network was detected.
- 2- network linked: The YoctoHub-Wireless did connect to the network.
- 3- DHCP ready: The local network is working (IP address obtained).
- 4- DNS ready: DNS server is reachable on the network.
- 5- WWW ready: The module has checked Internet connectivity by connecting itself to a time server (NTP).

When you have checked that your module does indeed have a valid IP address, you can close the property window, stop your *VirtualHub*, and disconnect your USB cable. They are not needed anymore.

From now on, you can access your YoctoHub-Wireless by typing its IP address directly in the address field of your preferred browser. The module answers to the standard HTTP port, but also to the 4444 port used by the *VirtualHub*. If your module IP address is *192.168.0.10*, you can therefore access it with the *http://192.168.0.10* URL.



The YoctoHub-Wireless interface is identical to that of a VirtualHub.

If you have assigned a name to your YoctoHub-Wireless, you can also use this name on the local network. For example, if your have used the *yoctohub* network name, you can contact the module with the *http://yoctohub* URL under Windows and the *http://yoctohub.local* URL under Mac OS X and Linux. Note that this technique is limited to the subnet of the YoctoHub-Wireless. If you want to contact the module by name from another network, you must use a classic DNS infrastructure.

3.2. Automated configuration

You can industrialize the YoctoHub-Wireless 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 [3].

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.

3.3. Connections

Power supply

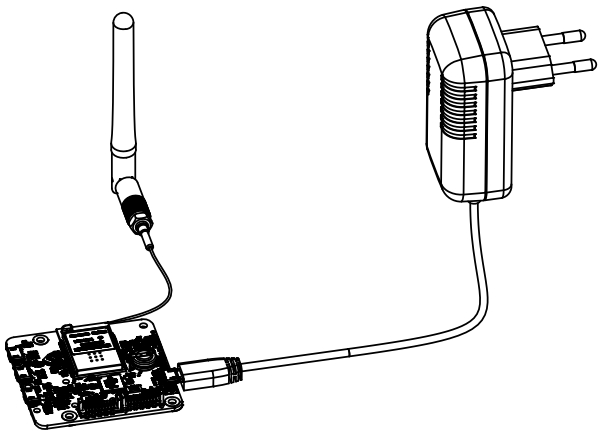
The YoctoHub-Wireless must be powered by the USB control socket.

USB

Simply connect a USB charger in the *power / control port* port, but make sure that the charger provides enough electric power. The YoctoHub-Wireless consumes about 110mA, to which you must

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

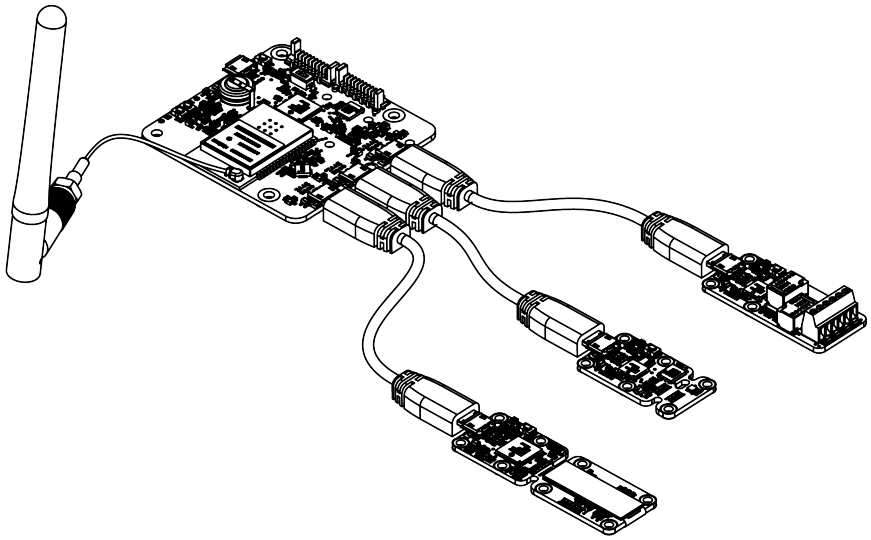
add the power consumption of each submodule. The YoctoHub-Wireless is designed to manage a maximum of 2A. Therefore, we recommend a USB charger able to deliver at least 2A. Moreover, you must make sure that the total power consumption of the set "hub + submodules" does not go above this limit.



The YoctoHub-Wireless can be powered by a regular USB charger

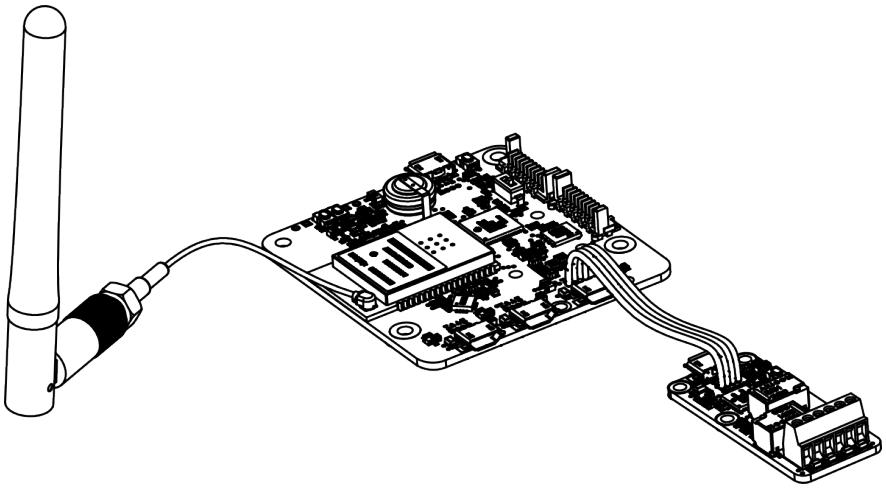
Sub-modules

The YoctoHub-Wireless 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.



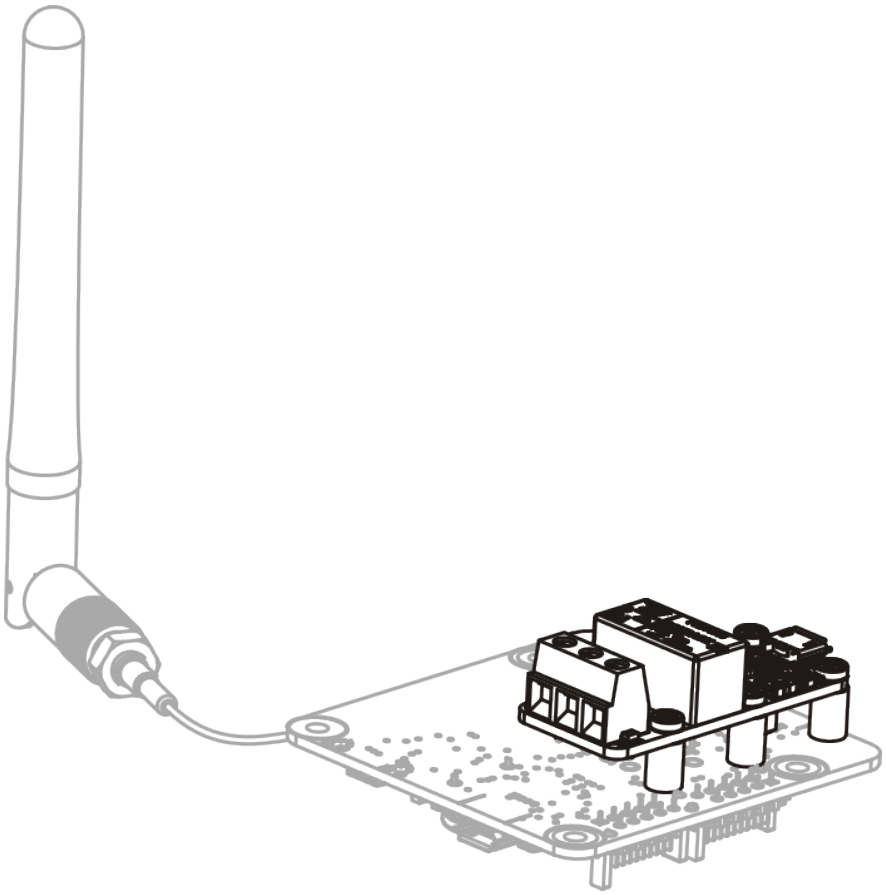
Connecting sub-modules with USB cables

Alternatively, you can connect your modules by directly soldering electric cables between the YoctoHub-Wireless 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-Wireless, 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-Wireless is designed so that you can fix a single width module directly on top of it. To do so, you need screws, spacers[4], and a 1.27mm pitch connector[5]. 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-Wireless is designed to drive only Yoctopuce modules. Indeed, the protocol used between the YoctoHub-Wireless 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-Wireless down ports, this port is automatically disabled to prevent damages to the device.

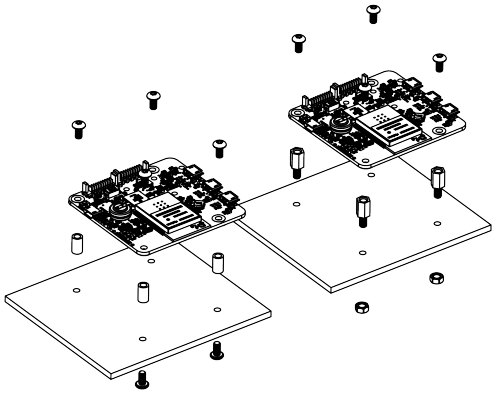
[4] http://www.yoctopuce.com/EN/products/accessories-and-connectors/fix-2-5mm
[5] http://www.yoctopuce.com/EN/products/accessories-and-connectors/board2board-127

4. Assembly

This chapter provides important information regarding the use of the YoctoHub-Wireless 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

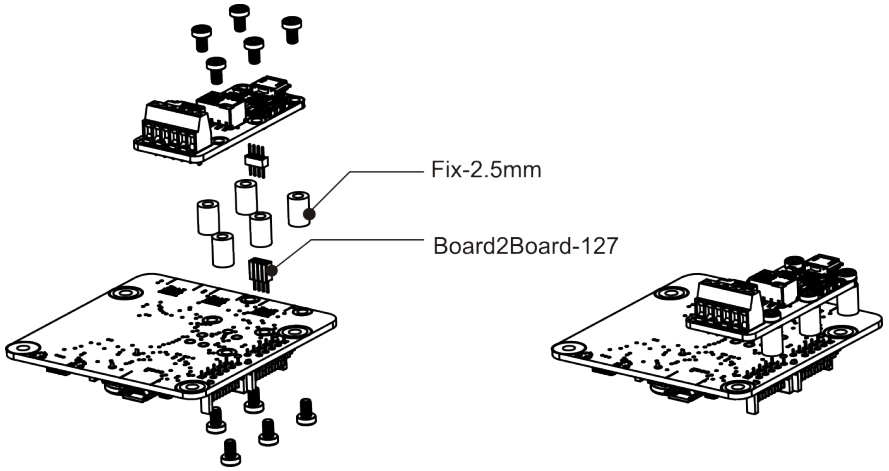
The YoctoHub-Wireless module contains 3mm assembly holes. You can use these holes for screws. The screw head diameter must not be larger than 8mm or the heads will damage the module circuits.

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-Wireless electronic components, in particular the network part, generate heat. You must not let this heat accumulate.

4.2. Fixing a sub-module

The YoctoHub-Wireless 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-Wireless 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.



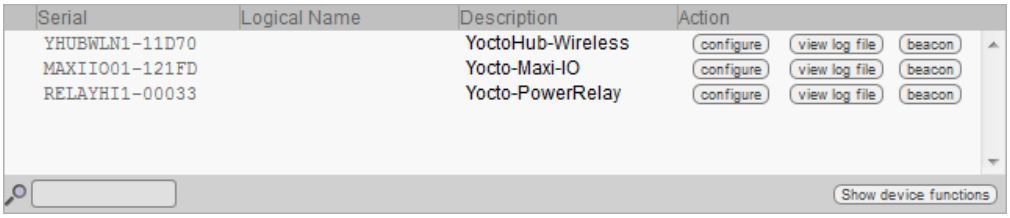
Fixing a module directly on the hub

Make sure to mount your module on the designed side, as illustrated above. The module 5 holes must correspond to the YoctoHub-Wireless 5 holes, and the square contact on the module must be connected to the square contact on the YoctoHub-Wireless 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-Wireless 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. Using the YoctoHub-Wireless

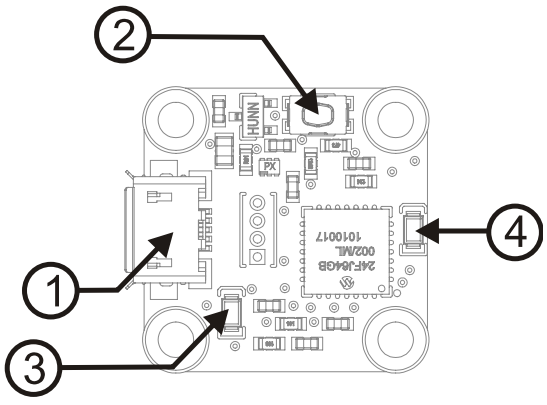
Apart from providing network access to the Yoctopuce module, the YoctoHub-Wireless enables you to test and configure your Yoctopuce modules. To do so, connect yourself to your YoctoHub-Wireless with your favorite web browser[1]. Use the IP address of the YoctoHub-Wireless or its network name, for example `http://192.168.0.10`. The list of the connected modules should appear.



YoctoHub-Wireless web interface

5.1. Locating the modules

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

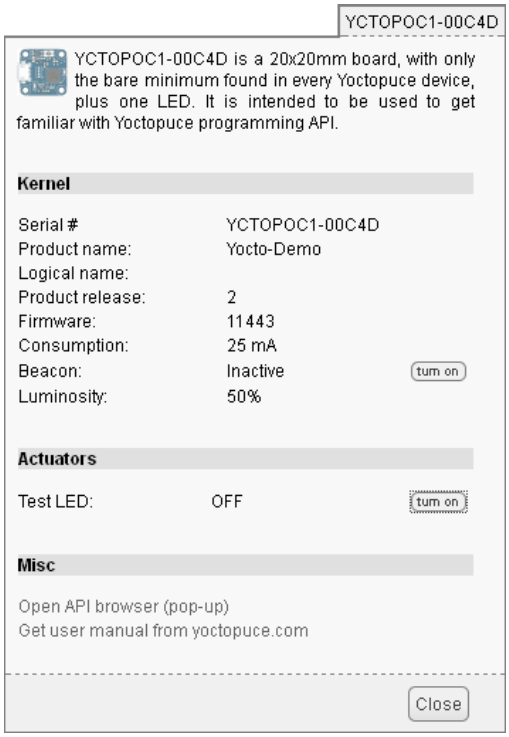


Yocto-button (1) and localization led (2) of the Yocto-Demo module. These two elements are usually placed in the same location, whatever the module.

[1] The YoctoHub-Wireless interface is regularly tested with Internet Explorer 6+, Firefox 3.5+, Chrome, and Safari. It does not work with Opera.

5.2. Testing the modules

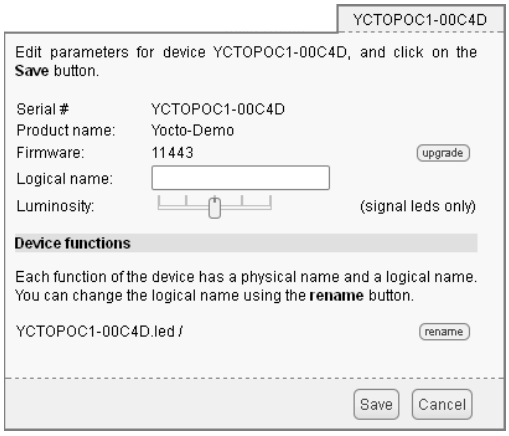
To test a module, simply click on the serial number of a module in the interface, a window specific to the module opens. This window generally allows you to activate the main functions of the module. Refer to the User's guide of the corresponding module for more details [2].



Property window of the Yocto-Demo module, obtained from the YoctoHub-Wireless interface

5.3. Configuring modules

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



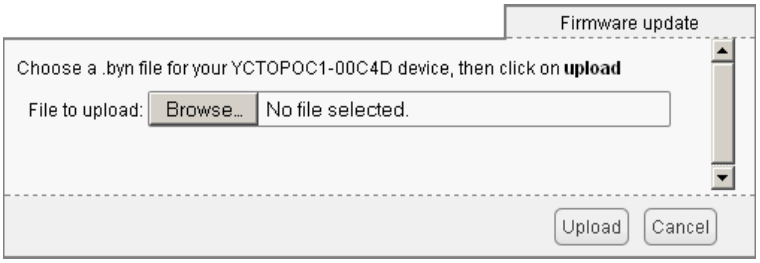
"Configuration" window of the Yocto-Demo module

[2] The YoctoHub-Wireless does not need to be more recent than the module you want to test and configure: all the elements specific to the module interfaces are kept in the module ROM, and not in the YoctoHub-Wireless.

5.4. Upgrading firmware

The Yoctopuce modules are in fact real computers, they even contain a small web server. And, as all computers, it is possible to update their control software (firmware). New firmware for each module are regularly published, they generally allow you to add new functionalities to the module, and/or to correct a hypothetical bug[3].

To update a module firmware, you must first get the new firmware. It can be downloaded from the module product page on the Yoctopuce web site[4]. The interface offers also a direct link if it detects that the firmware is not up-to-date [5]. Firmware is available as `.byn` files of a few tens of kilobytes. Save the one you are interested in on your local disk.



Firmware update window

When the firmware file is locally available, open the module **configuration** window and click on the **upgrade** button. The interface asks you to select the firmware file you wish to use. Enter the file name and click on **Upload**. From then on, everything is automatically performed: the YoctoHub-Wireless restarts the module in "update" mode, updates the firmware, then restarts the module in normal mode. The module configuration settings are kept. Do not disconnect the module during the update process.

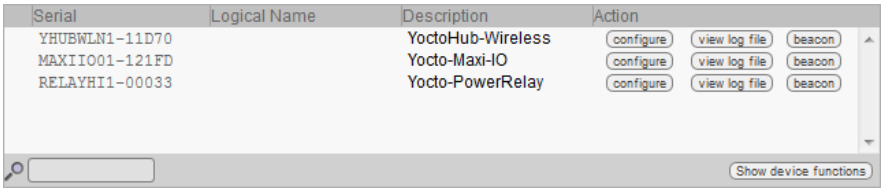
The YoctoHub-Wireless firmware can be updated in the same manner.

If control is lost during a firmware update (power failure or unwanted disconnection), it is always possible to manually force a firmware reload, even if the sub-module does not even appear in the YoctoHub-Wireless window. In this case, disconnect the module, and reconnect it while keeping the Yocto-button pressed. This starts the module in "update" mode. You can restart the firmware update process.

[3] Never trust people telling you that their software does not have bugs :-)
[4] www.yoctopuce.com
[5] On the condition that the interface could access the Yoctopuce web site.

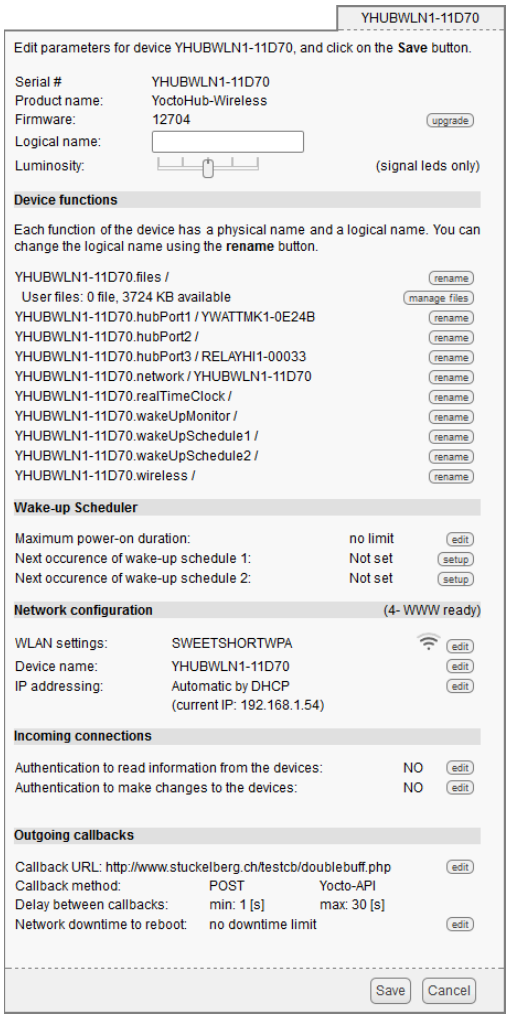
6. Access control

The YoctoHub-Wireless is able to perform access control to protect your Yoctopuce devices. Click on the **configure** button on the line matching the YoctoHub-Wireless in the user interface.



Click on the "configure" button on the first line

Then the configuration window for the YoctoHub-Wireless shows up.



The YoctoHub-Wireless configuration window.

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

6.1. Protected "admin" access

The *admin* password locks write access on the modules. When the admin password is set, only users using the admin login are allowed read and write access to the modules. The users using the *admin* login can configure the modules seen by this YoctoHub-Wireless as they wish.

6.2. Protected "user" access

The *user* password locks read access to the Yoctopuce modules. When set, any use without password becomes impossible. The *user* access type allows only read-only access to the modules, that is only to consult the states of the modules. If you simultaneously create "admin" and "user" access controls, users with a "user" login cannot modify the configuration of modules seen by this YoctoHub-Wireless.

If you configure an *admin* access, without configuring a *user* access, users are still able to read your device values without any password.

To set up YoctoHub-Wireless access, click the **edit** button on the line **Authentication to read the information from the devices** or **Authentication to write information to the devices**

6.3. Access control and API

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

```
yRegisterHub("admin:mypass@192.168.0.10:4444",errmsg);
```

6.4. Deleting passwords

If you forget your YoctoHub-Wireless password, the only way to regain control is to reset all the settings to the default value. To do so, find a USB cable for the YoctoHub-Wireless and connect it to a computer running the *VirtualHub*[1] while keeping the Yocto-button pressed. This forces the YoctoHub-Wireless to start in firmware update mode. It then appears in the *VirtualHub* below the module list. Click on its serial number and select a firmware file to load on the module. When the firmware is reloaded with this method, the module is reset to the factory settings, without access control.

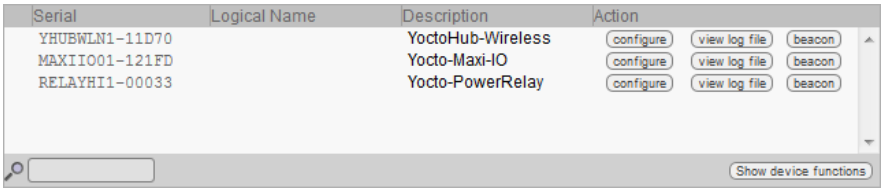
[1] http://www.yoctopuce.com/EN/virtualhub.php

7. Interaction with external services

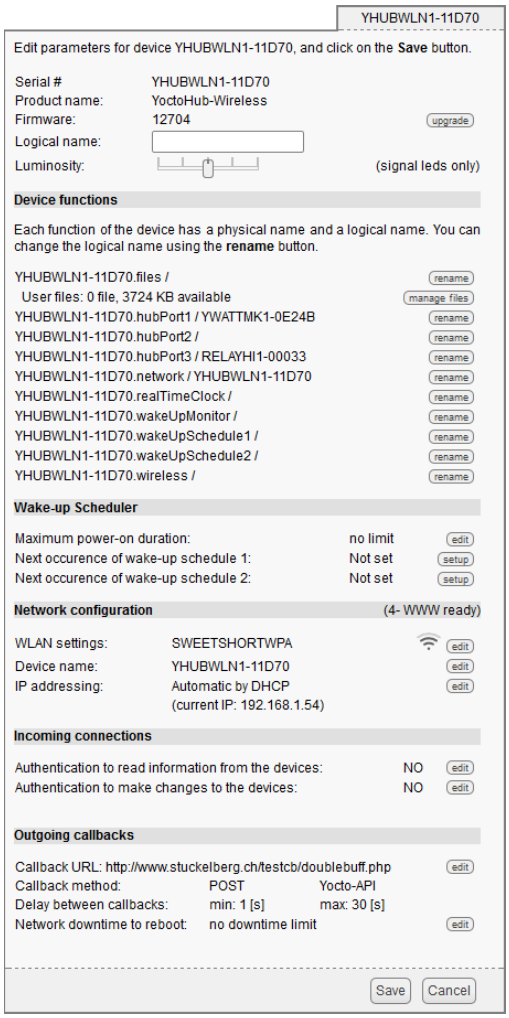
The YoctoHub-Wireless can publish the state of connected devices on any web server. The values are posted on a regular basis and each time one of them changes significantly. This feature enables you to interface your Yoctopuce devices with many web services.

7.1. Configuration

To use this feature, just click on the **configure** button located on the line matching the YoctoHub-Wireless on the main user interface. Then look for the **Outgoing calbacks** section and click on the **edit** button.



Just click on the "Configure" button on the first line.

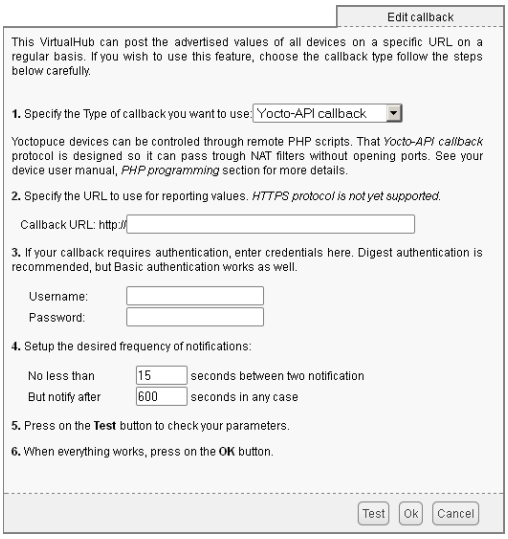


Then edit the "Outgoing callbacks" section.

The callback configuration window shows up. This window enables you to define how your YoctoHub-Wireless interacts with an external web server. Several interaction types are at your disposal.

7.2. User defined callback

The "*User defined callback*" allow you to fully customize the way the YoctoHub-Wireless interacts with an external web site. You need to provide the URL of the web server where you want the hub to post data. Note that only HTTP protocol is supported (no HTTPS).



The callback configuration window.

If you want to secure access to your callback script, you can setup a standard HTTP authentication. The YoctoHub-Wireless knows how to handle standard HTTP authentication schemes: simply fill in 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.

The YoctoHub-Wireless posts the advertised values[1] on a regular basis, and each time one of these values changes significantly. You can change the default delay between posts.

Tests

To help you debug the process, you can visualize with the YoctoHub-Wireless the answer to the callback sent by the web server. Click on the **test** button when all required fields are filled. When the result meets your expectations, close the debug window and then click on the "Ok" button.

Format

Values are posted in one of the following formats:

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

```
FUNCTION_NAME = VALUE
```

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

```
MODULE_NAME#HARDWARE_NAME = VALUE
```

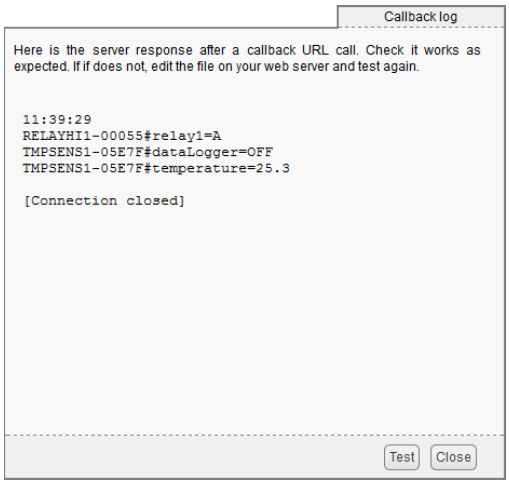
3. If no logical name has been set:

```
SERIAL_NUMBER#HARDWARE_NAME = VALUE
```

Here is a short PHP script allowing you to visualize the data posted by the callback and the result in the debug window:

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

[1] Advertised values are the ones you can see on the YoctoHub-Wireless main interface when you click on the *show functions* button.



Callback test results with a Yocto-PowerRelay and a Yocto-Temperature.

7.3. Yocto-API callback

The PHP Yoctopuce API is able to work on *callback* mode. This way, a PHP script can take control of Yoctopuce devices installed behind a NAT filter without having to open any port. Typically, this allows you to control Yoctopuce devices running on a LAN behind a private ADSL router from a public web site. The YoctoHub-Wireless then acts as a gateway. All you have to do is to define the PHP script URL and, if applicable, the credentials needed to access it. You can find more information about this callback mode in your Yoctopuce device User's guide.

7.4. Xively (previously Cosm)

Xively[2] is a paying cloud based service (with a free but limited development mode) enabling you to draw graphs with data coming from your Yoctopuce sensors. You can interface your Yoctopuce sensors with Xively without having to write a single line of code. To achieve this, you need to create a Xively account, then to define a feed ID and a API key on the Xively site. Then enter those two parameters in the YoctoHub-Wireless user interface. That's it. If needed, you will find more information about Xively in the Yoctopuce Blog[3]. Yoctopuce is not affiliated to Xively.

7.5. Thinkspeak

ThingSpeak[4] is another cloud service, but completely free. It also enables you to trace graphs with data coming from your Yoctopuce sensors. This service presents some feature limitations compared to Xively, but it has the advantage of not requiring any paying license. You can find on the Yoctopuce blog [5] how to configure your YoctoHub-Wireless to post data directly on ThinkSpeak. Yoctopuce is not affiliated to ThingSpeak.

[2] www.xively.com
[3] http://www.yoctopuce.com/EN/article/connect-your-sensors-to-the-cloud
[4] www.thingspeak.com
[5] http://www.yoctopuce.com/EN/article/cosm-alternatives-to-record-sensor-measurements

8. Programming

8.1. Accessing connected modules

The YoctoHub-Wireless behaves itself exactly like a computer running a *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-Wireless is located at the level of the *registerHub* function call. To use USB modules connected natively, the *registerHub* parameter is *usb*. To use modules connected to a YoctoHub-Wireless, you must simply replace this parameter by the IP address of the YoctoHub-Wireless. For instance, in Delphi:

```
YRegisterHub("usb",errmsg);
```

becomes

```
YRegisterHub("192.168.0.10",errmsg); // The hub IP address is 192.168.0.10
```

8.2. Controlling the YoctoHub-Wireless

From the programming API standpoint, the YoctoHub-Wireless is a module like the others. 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-Wireless, and so on.

Network

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

HubPort

This class enables you to manage the hub part. You can enable or disable the YoctoHub-Wireless 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-Wireless. The YoctoHub-Wireless contains a small file system which allows you to store, for example, a web application controlling the modules connected to the YoctoHub-Wireless.

WakeMonitor

This class enables you to monitor the sleep mode of the YoctoHub-Wireless.

WakeSchedule

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

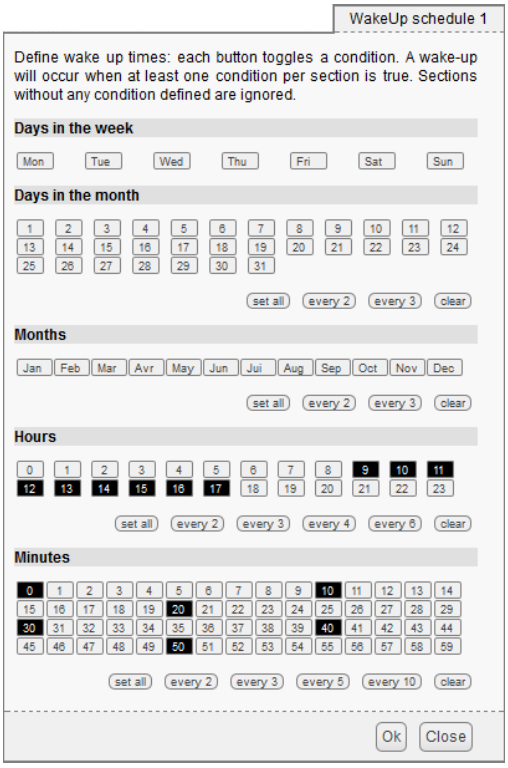
You can find some examples on how to drive the YoctoHub-Wireless by software in the Yoctopuce programming libraries, available free of charge on the Yoctopuce web site.

9. Sleep mode

The YoctoHub-Wireless 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.

9.1. Manual configuration of the wake ups

You can manually configure the wake up conditions by connecting yourself on the interface of the YoctoHub-Wireless. 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-Wireless interface the maximal wake up duration, after which the module automatically goes back to sleep. If your YoctoHub-Wireless is running on batteries, this ensures they do not empty even if no explicit sleep command is received.

9.2. 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

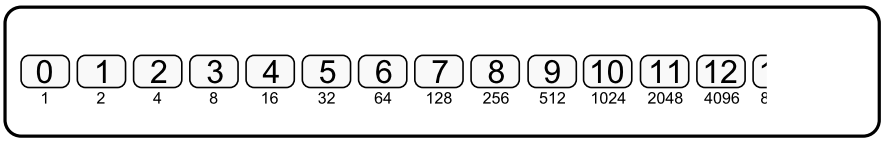
The *wakeUpMonitor* function manages wake ups and sleep periods, proper. It provides all the instant managing functionalities : instant wake up, instant sleep, computing the date of the next wake up, and so on...

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

wakeUpSchedule

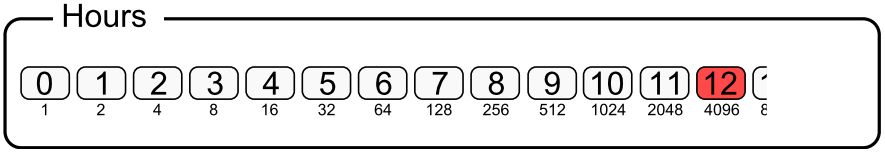
The *wakeUpSchedule* function enables you to program a wake up condition followed by a possible sleep. It includes five variables enabling you to define correspondences on minutes, hours, days of the week, days of the month, and months. These variables are integers where each bit defines a correspondence. 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-Wireless.

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.



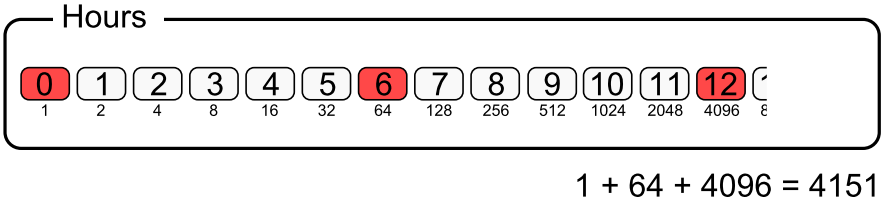
To each box is assigned a power of two

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



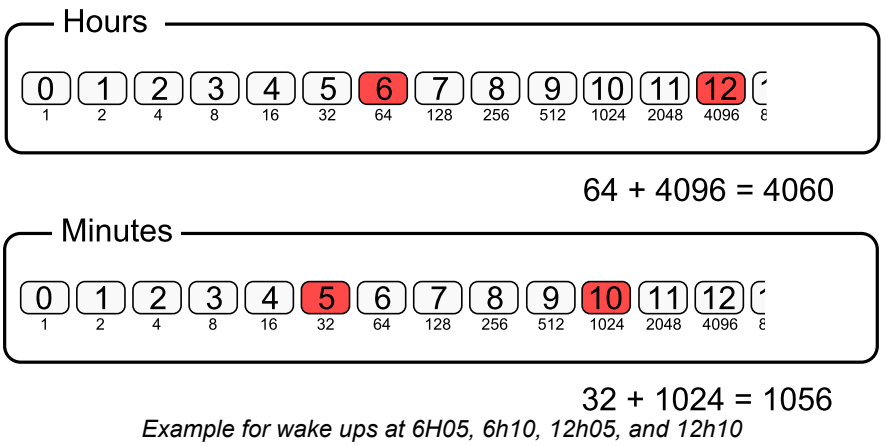
Example for a wake up at 12h

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

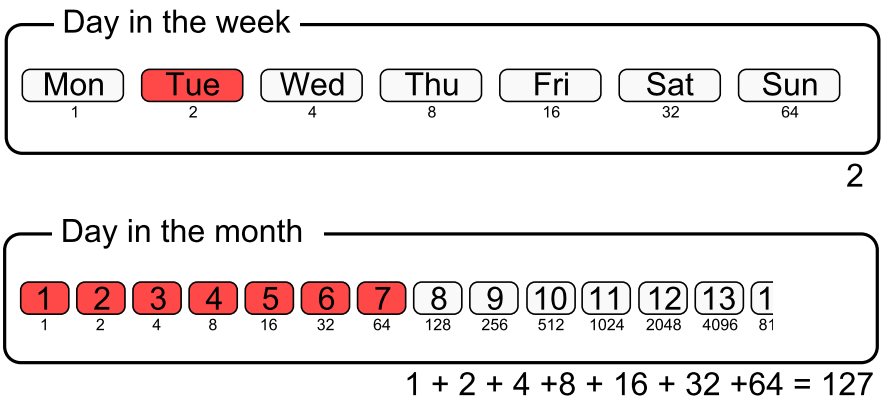
Variables can be combined. For a wake up to happen every day at 6h05, 6h10, 12h05, and 12h10, you must set the hours to 2^6 + 2^12 = 4060, minutes to 2^5 and 2^10 = 1056. Variables remaining at the zero value are ignored.



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

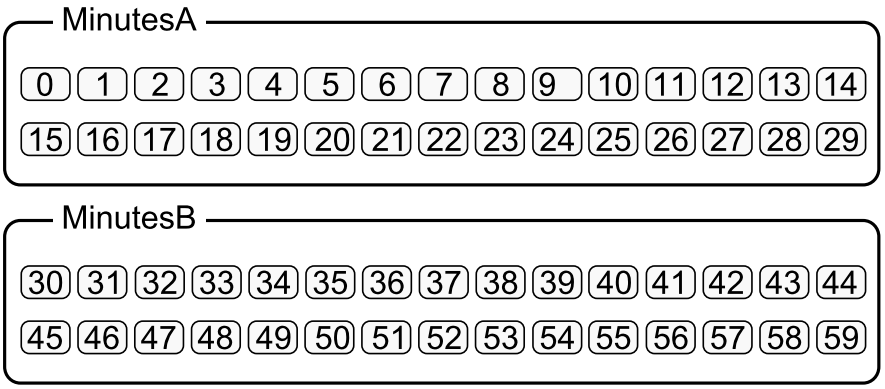
Note that if you want to program a wake up at 6h05 and 12h10, but not at 6h10 and 12h05, you need to use two distinct *wakeUpSchedule* functions.

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-Wireless includes two *wakeUpSchedule* functions, enabling you to program up to two independent wake up types.

10. Personalizing the web interface

Your YoctoHub-Wireless contains a small embedded file system, allowing it to store personalized files for its own use. You can manipulate the file system thanks to the *yocto_files* library. You can store there the files you want to. If need be, you can store a web application enabling you to manage modules connected to your YoctoHub-Wireless.

10.1. Using the file system

Interactive use

The YoctoHub-Wireless web interface provides a succinct interface to manipulate the content of the file system: simply click the *configuration* button corresponding to your module in the hub interface, then the *manage files* button. The files are listed and you can view them, erase them, or add new ones (downloads).

Because of its small size, the file system does not have an explicit concept of directories. You can nevertheless use the slash sign "/" inside file names to sort them as if they were in directories.

Programmed use

Use the *yocto_files* library to manage the file system. Basic functions are available:

- *upload* creates a new file on the module, with a content that you provide;
- *get_list* lists the files on the module, including their content size and CRC32;
- *download* retrieves in a variable the content of a file present on the module;
- *remove* erases a file from the module;
- *format* resets the file system to an empty, not fragmented state.

A piece of software using a well designed file system should always start by making sure that all the files necessary for its working are available on the module and, if needed, upload them on the module. We can thus transparently manage software updates and application deployment on new modules. To make file versions easier to detect, the *get_list* method returns for each file a 32 bit signature called CRC (Cyclic Redundancy Check) which identifies in a reliable manner the file content. Thus, if the file CRC corresponds, there is less than one chance over 4 billions that the content is not the correct one. You can even compute in advance in your software the CRC of the content you want, and therefore check it without having to download the files. The CRC function used by the Yoctopuce file system is the same as Ethernet, Gzip, PNG, etc. Its characteristic value for the nine character string "123456789" is 0xCBF43926.

HTTP use

You can access the files that your have downloaded on your YoctoHub-Wireless by HTTP at the root of the module (at the same level as the REST API). This allows you to load personalized HTML and Javascript interface pages, for example. You cannot, however, replace the content of a file preloaded on the module, you can only add new ones.

UI and optimisation

Since you can store files on the hub file system, you can easily build a web application to control the devices connected to the hub and store it directly on the hub. This is a very convenient way to build system remote controlled by tablets or smart phones. However the web server embedded in the hub have limited connectivity capabilities: only a few number of sockets can be opened at the same time. Since most web browsers tend to open as many connection as they can to load all elements in a web page, this might lead to very long loading time. To prevent this, try to keep your UI pages as compact as possible by embedding the javascript, CSS code and if possible, images in base64 code.

10.2. Limitations

The file system embedded on your YoctoHub-Wireless has some technical limitations:

- Its maximal storage space is 3.5Mo, allocated in blocks enabling to store up to about 800 files.
- Erasing a file does not necessarily immediately free all the space used by the file. The non freed space is completely reused if you create a new file with the same name, but not necessarily if you create files with a distinct name each time. For this reason, it is not recommended to automatically create files with distinct names.
- You can recover the complete non freed space with the *format* command which frees all the files.
- Each firmware update implicitly provokes a complete reformatting of the file system.
- As all flash memories, the memory used to store the files has a life of about 100'000 erasing cycles. It is enough, but it is not infinite. Make sure that you do not write and erase files uselessly and very quickly in a loop, or you may destroy your module.

11. High-level API Reference

This chapter summarizes the high-level API functions to drive your YoctoHub-Wireless. 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-Wireless.

11.1. Yocto-hub port interface

YHubPort objects provide control over the power supply for every YoctoHub port and 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:

```
js      <script type='text/javascript' src='yocto_hubport.js'></script>
nodejs  var yoctolib = require('yoctolib');
        var YHubPort = yoctolib.YHubPort;
php     require_once('yocto_hubport.php');
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;
py      from yocto_hubport import *
```

Global functions
yFindHubPort(**func**) Retrieves a Yocto-hub port for a given identifier.
yFirstHubPort() Starts the enumeration of Yocto-hub ports currently accessible.
`YHubPort` methods
hubport→describe() Returns a short text that describes unambiguously the instance of the Yocto-hub port in the form `TYPE(NAME)=SERIAL.FUNCTIONID`.
hubport→get_advertisedValue() Returns the current value of the Yocto-hub port (no more than 6 characters).
hubport→get_baudRate() Returns the current baud rate used by this Yocto-hub port, in kbps.
hubport→get_enabled() Returns true if the Yocto-hub port is powered, false otherwise.
hubport→get_errorMessage() Returns the error message of the latest error with the Yocto-hub port.
hubport→get_errorType() Returns the numerical error code of the latest error with the Yocto-hub port.
hubport→get_friendlyName() Returns a global identifier of the Yocto-hub port in the format `MODULE_NAME.FUNCTION_NAME`.
hubport→get_functionDescriptor() Returns a unique identifier of type `YFUN_DESCR` corresponding to the function.
hubport→get_functionId() Returns the hardware identifier of the Yocto-hub port, without reference to the module.
hubport→get_hardwareId() Returns the unique hardware identifier of the Yocto-hub port in the form `SERIAL.FUNCTIONID`.
hubport→get_logicalName() Returns the logical name of the Yocto-hub port.

hubport→get_module()

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

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 Yocto-hub port.

hubport→get_userData()

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

hubport→isOnline()

Checks if the Yocto-hub port is currently reachable, without raising any error.

hubport→isOnline_async(**callback**, **context**)

Checks if the Yocto-hub port is currently reachable, without raising any error (asynchronous version).

hubport→load(**msValidity**)

Preloads the Yocto-hub port cache with a specified validity duration.

hubport→load_async(**msValidity**, **callback**, **context**)

Preloads the Yocto-hub port cache with a specified validity duration (asynchronous version).

hubport→nextHubPort()

Continues the enumeration of Yocto-hub ports started using `yFirstHubPort()`.

hubport→registerValueCallback(**callback**)

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

hubport→set_enabled(**newval**)

Changes the activation of the Yocto-hub port.

hubport→set_logicalName(**newval**)

Changes the logical name of the Yocto-hub port.

hubport→set_userData(**data**)

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

hubport→wait_async(**callback**, **context**)

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

YHubPort.FindHubPort() YHubPort yFindHubPort()

Retrieves a Yocto-hub port for a given identifier.

```
js      function yFindHubPort( func)
nodejs  function FindHubPort( func)
php     function yFindHubPort( $func)
cpp     YHubPort* yFindHubPort( const string& func)
m       +(YHubPort*) FindHubPort :(NSString*) func
pas     function yFindHubPort( func: string): TYHubPort
vb      function yFindHubPort( ByVal func As String) As YHubPort
cs      YHubPort FindHubPort( string func)
java    YHubPort FindHubPort( String func)
py      def FindHubPort( func)
```

The identifier can be specified using several formats:

- FunctionLogicalName
- ModuleSerialNumber.FunctionIdentifier
- ModuleSerialNumber.FunctionLogicalName
- ModuleLogicalName.FunctionIdentifier
- ModuleLogicalName.FunctionLogicalName

This function does not require that the Yocto-hub port is online at the time it is invoked. The returned object is nevertheless valid. Use the method `YHubPort.isOnline()` to test if the Yocto-hub port is indeed online at a given time. In case of ambiguity when looking for a Yocto-hub port by logical name, no error is notified: the first instance found is returned. The search is performed first by hardware name, then by logical name.

Parameters :

func a string that uniquely characterizes the Yocto-hub port

Returns :

a `YHubPort` object allowing you to drive the Yocto-hub port.

YHubPort.FirstHubPort() YHubPort yFirstHubPort()

Starts the enumeration of Yocto-hub ports currently accessible.

```
js      function yFirstHubPort( )
nodejs  function FirstHubPort( )
php     function yFirstHubPort( )
cpp     YHubPort* yFirstHubPort( )
m       +(YHubPort*) FirstHubPort
pas     function yFirstHubPort( ): TYHubPort
vb      function yFirstHubPort( ) As YHubPort
cs      YHubPort FirstHubPort( )
java    YHubPort FirstHubPort( )
py      def FirstHubPort( )
```

Use the method `YHubPort.nextHubPort()` to iterate on next Yocto-hub ports.

Returns :

a pointer to a `YHubPort` object, corresponding to the first Yocto-hub port currently online, or a `null` pointer if there are none.

hubport→describe() YHubPort

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

```
js      function describe( )
nodejs  function describe( )
php     function describe( )
cpp     string describe( )
m       -(NSString*) describe
pas     function describe( ): string
vb      function describe( ) As String
cs      string describe( )
java    String describe( )
py      def describe( )
```

More precisely, `TYPE` is the type of the function, `NAME` it the name used for the first access to the function, `SERIAL` is the serial number of the module if the module is connected or `"unresolved"`, and `FUNCTIONID` is the hardware identifier of the function if the module is connected. For example, this method returns `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1` if the module is already connected or `Relay(BadCustomeName.relay1)=unresolved` if the module has not yet been connected. This method does not trigger any USB or TCP transaction and can therefore be used in a debugger.

Returns :

a string that describes the Yocto-hub port (ex: `Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1`)

hubport→get_advertisedValue() YHubPort
hubport→advertisedValue()

Returns the current value of the Yocto-hub port (no more than 6 characters).

```
js      function get_advertisedValue( )
nodejs  function get_advertisedValue( )
php     function get_advertisedValue( )
cpp     string get_advertisedValue( )
m       -(NSString*) advertisedValue
pas     function get_advertisedValue( ): string
vb      function get_advertisedValue( ) As String
cs      string get_advertisedValue( )
java    String get_advertisedValue( )
py      def get_advertisedValue( )
cmd     YHubPort target get_advertisedValue
```

Returns :
a string corresponding to the current value of the Yocto-hub port (no more than 6 characters).

On failure, throws an exception or returns `Y_ADVERTISEDVALUE_INVALID`.

hubport→get_baudRate() YHubPort
hubport→baudRate()

Returns the current baud rate used by this Yocto-hub port, in kbps.

```
js      function get_baudRate( )
nodejs  function get_baudRate( )
php     function get_baudRate( )
cpp     int get_baudRate( )
m       -(int) baudRate
pas     function get_baudRate( ): LongInt
vb      function get_baudRate( ) As Integer
cs      int get_baudRate( )
java    int get_baudRate( )
py      def get_baudRate( )
cmd     YHubPort target get_baudRate
```

The default value is 1000 kbps, but a slower rate may be used if communication problems are encountered.

Returns :

an integer corresponding to the current baud rate used by this Yocto-hub port, in kbps

On failure, throws an exception or returns `Y_BAUDRATE_INVALID`.

hubport→get_enabled() YHubPort
hubport→enabled()

Returns true if the Yocto-hub port is powered, false otherwise.

```
js      function get_enabled( )
nodejs  function get_enabled( )
php     function get_enabled( )
cpp     Y_ENABLED_enum get_enabled( )
m       -(Y_ENABLED_enum) enabled
pas     function get_enabled( ): Integer
vb      function get_enabled( ) As Integer
cs      int get_enabled( )
java    int get_enabled( )
py      def get_enabled( )
cmd     YHubPort target get_enabled
```

Returns :

either Y_ENABLED_FALSE or Y_ENABLED_TRUE, according to true if the Yocto-hub port is powered, false otherwise

On failure, throws an exception or returns Y_ENABLED_INVALID.

hubport→get_errorMessage() YHubPort
hubport→errorMessage()

Returns the error message of the latest error with the Yocto-hub port.

```
js      function get_errorMessage( )
nodejs  function get_errorMessage( )
php     function get_errorMessage( )
cpp     string get_errorMessage( )
m       -(NSString*) errorMessage
pas     function get_errorMessage( ): string
vb      function get_errorMessage( ) As String
cs      string get_errorMessage( )
java    String get_errorMessage( )
py      def get_errorMessage( )
```

This method is mostly useful when using the Yoctopuce library with exceptions disabled.

Returns :
a string corresponding to the latest error message that occured while using the Yocto-hub port object

hubport→get_errorType() YHubPort
hubport→errorType()

Returns the numerical error code of the latest error with the Yocto-hub port.

```
js      function get_errorType( )
nodejs  function get_errorType( )
php     function get_errorType( )
cpp     YRETCODE get_errorType( )
pas     function get_errorType( ): YRETCODE
vb      function get_errorType( ) As YRETCODE
cs      YRETCODE get_errorType( )
java    int get_errorType( )
py      def get_errorType( )
```

This method is mostly useful when using the Yoctopuce library with exceptions disabled.

Returns :
a number corresponding to the code of the latest error that occured while using the Yocto-hub port object

hubport→get_friendlyName() YHubPort
hubport→friendlyName()

Returns a global identifier of the Yocto-hub port in the format MODULE_NAME.FUNCTION_NAME.

```
js      function get_friendlyName( )
nodejs  function get_friendlyName( )
php     function get_friendlyName( )
cpp     string get_friendlyName( )
m       -(NSString*) friendlyName
cs      string get_friendlyName( )
java    String get_friendlyName( )
py      def get_friendlyName( )
```

The returned string uses the logical names of the module and of the Yocto-hub port if they are defined, otherwise the serial number of the module and the hardware identifier of the Yocto-hub port (for exemple: MyCustomName.relay1)

Returns :

a string that uniquely identifies the Yocto-hub port using logical names (ex: MyCustomName.relay1)

On failure, throws an exception or returns Y_FRIENDLYNAME_INVALID.

hubport→get_functionDescriptor() YHubPort
hubport→functionDescriptor()

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

js	function **get_functionDescriptor**( )
nodejs	function **get_functionDescriptor**( )
php	function **get_functionDescriptor**( )
cpp	YFUN_DESCR **get_functionDescriptor**( )
m	-(YFUN_DESCR) functionDescriptor
pas	function **get_functionDescriptor**( ): YFUN_DESCR
vb	function **get_functionDescriptor**( ) As YFUN_DESCR
cs	YFUN_DESCR **get_functionDescriptor**( )
java	String **get_functionDescriptor**( )
py	def **get_functionDescriptor**( )

This identifier can be used to test if two instances of YFunction reference the same physical function on the same physical device.

Returns :

an identifier of type YFUN_DESCR.

If the function has never been contacted, the returned value is Y_FUNCTIONDESCRIPTOR_INVALID.

hubport→get_functionId() YHubPort
hubport→functionId()

Returns the hardware identifier of the Yocto-hub port, without reference to the module.

```
js      function get_functionId( )
nodejs  function get_functionId( )
php     function get_functionId( )
cpp     string get_functionId( )
m       -(NSString*) functionId
vb      function get_functionId( ) As String
cs      string get_functionId( )
java    String get_functionId( )
py      def get_functionId( )
```

For example `relay1`

Returns :

a string that identifies the Yocto-hub port (ex: `relay1`)

On failure, throws an exception or returns `Y_FUNCTIONID_INVALID`.

hubport→get_hardwareId() YHubPort
hubport→hardwareId()

Returns the unique hardware identifier of the Yocto-hub port in the form SERIAL.FUNCTIONID.

```
js      function get_hardwareId( )
nodejs  function get_hardwareId( )
php     function get_hardwareId( )
cpp     string get_hardwareId( )
m       -(NSString*) hardwareId
vb      function get_hardwareId( ) As String
cs      string get_hardwareId( )
java    String get_hardwareId( )
py      def get_hardwareId( )
```

The unique hardware identifier is composed of the device serial number and of the hardware identifier of the Yocto-hub port (for example RELAYLO1-123456.relay1).

Returns :

a string that uniquely identifies the Yocto-hub port (ex: RELAYLO1-123456.relay1)

On failure, throws an exception or returns Y_HARDWAREID_INVALID.

hubport→get_logicalName() YHubPort
hubport→logicalName()

Returns the logical name of the Yocto-hub port.

```
js      function get_logicalName( )
nodejs  function get_logicalName( )
php     function get_logicalName( )
cpp     string get_logicalName( )
m       -(NSString*) logicalName
pas     function get_logicalName( ): string
vb      function get_logicalName( ) As String
cs      string get_logicalName( )
java    String get_logicalName( )
py      def get_logicalName( )
cmd     YHubPort target get_logicalName
```

Returns :
a string corresponding to the logical name of the Yocto-hub port.

On failure, throws an exception or returns `Y_LOGICALNAME_INVALID`.

hubport→get_module() YHubPort
hubport→module()

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

```
js      function get_module( )
nodejs  function get_module( )
php     function get_module( )
cpp     YModule * get_module( )
m       -(YModule*) module
pas     function get_module( ): TYModule
vb      function get_module( ) As YModule
cs      YModule get_module( )
java    YModule get_module( )
py      def get_module( )
```

If the function cannot be located on any module, the returned instance of YModule is not shown as on-line.

Returns :
an instance of YModule

hubport→get_module_async() YHubPort
hubport→module_async()

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

```
js      function get_module_async( callback, context)
nodejs  function get_module_async( callback, context)
```

If the function cannot be located on any module, the returned YModule object does not show as on-line.

This asynchronous version exists only in Javascript. It uses a callback instead of a return value in order to avoid blocking Firefox javascript VM that does not implement context switching during blocking I/O calls. See the documentation section on asynchronous Javascript calls for more details.

Parameters :

callback callback function that is invoked when the result is known. The callback function receives three arguments: the caller-specific context object, the receiving function object and the requested YModule object

context caller-specific object that is passed as-is to the callback function

Returns :

nothing : the result is provided to the callback.

hubport→get_portState() YHubPort
hubport→portState()

Returns the current state of the Yocto-hub port.

```
js      function get_portState( )
nodejs  function get_portState( )
php     function get_portState( )
cpp     Y_PORTSTATE_enum get_portState( )
m       -(Y_PORTSTATE_enum) portState
pas     function get_portState( ): Integer
vb      function get_portState( ) As Integer
cs      int get_portState( )
java    int get_portState( )
py      def get_portState( )
cmd     YHubPort target get_portState
```

Returns :

a value among `Y_PORTSTATE_OFF`, `Y_PORTSTATE_OVRLD`, `Y_PORTSTATE_ON`, `Y_PORTSTATE_RUN` and `Y_PORTSTATE_PROG` corresponding to the current state of the Yocto-hub port

On failure, throws an exception or returns `Y_PORTSTATE_INVALID`.

hubport→get_userData() YHubPort
hubport→userData()

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

js	function **get_userData**( )
nodejs	function **get_userData**( )
php	function **get_userData**( )
cpp	void * **get_userData**( )
m	-(void*) userData
pas	function **get_userData**( ): Tobject
vb	function **get_userData**( ) As Object
cs	object **get_userData**( )
java	Object **get_userData**( )
py	def **get_userData**( )

This attribute is never touched directly by the API, and is at disposal of the caller to store a context.

Returns :
the object stored previously by the caller.

hubport→isOnline() YHubPort

Checks if the Yocto-hub port is currently reachable, without raising any error.

```
js      function isOnline( )
nodejs  function isOnline( )
php     function isOnline( )
cpp     bool isOnline( )
m       -(BOOL) isOnline
pas     function isOnline( ): boolean
vb      function isOnline( ) As Boolean
cs      bool isOnline( )
java    boolean isOnline( )
py      def isOnline( )
```

If there is a cached value for the Yocto-hub port in cache, that has not yet expired, the device is considered reachable. No exception is raised if there is an error while trying to contact the device hosting the Yocto-hub port.

Returns :

`true` if the Yocto-hub port can be reached, and `false` otherwise

hubport→isOnline_async() YHubPort

Checks if the Yocto-hub port is currently reachable, without raising any error (asynchronous version).

```
js      function isOnline_async( callback, context)
nodejs  function isOnline_async( callback, context)
```

If there is a cached value for the Yocto-hub port in cache, that has not yet expired, the device is considered reachable. No exception is raised if there is an error while trying to contact the device hosting the requested function.

This asynchronous version exists only in Javascript. It uses a callback instead of a return value in order to avoid blocking the Javascript virtual machine.

Parameters :

callback callback function that is invoked when the result is known. The callback function receives three arguments: the caller-specific context object, the receiving function object and the boolean result

context caller-specific object that is passed as-is to the callback function

Returns :

nothing : the result is provided to the callback.

hubport→load() YHubPort

Preloads the Yocto-hub port cache with a specified validity duration.

```
js      function load( msValidity)
nodejs  function load( msValidity)
php     function load( $msValidity)
cpp     YRETCODE load( int msValidity)
m       -(YRETCODE) load : (int) msValidity
pas     function load( msValidity: integer): YRETCODE
vb      function load( ByVal msValidity As Integer) As YRETCODE
cs      YRETCODE load( int msValidity)
java    int load( long msValidity)
py      def load( msValidity)
```

By default, whenever accessing a device, all function attributes are kept in cache for the standard duration (5 ms). This method can be used to temporarily mark the cache as valid for a longer period, in order to reduce network trafic for instance.

Parameters :

msValidity an integer corresponding to the validity attributed to the loaded function parameters, in milliseconds

Returns :

YAPI_SUCCESS when the call succeeds.

On failure, throws an exception or returns a negative error code.

hubport→load_async() YHubPort

Preloads the Yocto-hub port cache with a specified validity duration (asynchronous version).

```
js      function load_async( msValidity, callback, context)
nodejs  function load_async( msValidity, callback, context)
```

By default, whenever accessing a device, all function attributes are kept in cache for the standard duration (5 ms). This method can be used to temporarily mark the cache as valid for a longer period, in order to reduce network trafic for instance.

This asynchronous version exists only in Javascript. It uses a callback instead of a return value in order to avoid blocking the Javascript virtual machine.

Parameters :

msValidity	an integer corresponding to the validity of the loaded function parameters, in milliseconds
callback	callback function that is invoked when the result is known. The callback function receives three arguments: the caller-specific context object, the receiving function object and the error code (or YAPI_SUCCESS)
context	caller-specific object that is passed as-is to the callback function

Returns :

nothing : the result is provided to the callback.

hubport→nextHubPort() YHubPort

Continues the enumeration of Yocto-hub ports started using `yFirstHubPort()`.

```
js      function nextHubPort( )
nodejs  function nextHubPort( )
php     function nextHubPort( )
cpp     YHubPort * nextHubPort( )
m       -(YHubPort*) nextHubPort
pas     function nextHubPort( ): TYHubPort
vb      function nextHubPort( ) As YHubPort
cs      YHubPort nextHubPort( )
java    YHubPort nextHubPort( )
py      def nextHubPort( )
```

Returns :

a pointer to a `YHubPort` object, corresponding to a Yocto-hub port currently online, or a `null` pointer if there are no more Yocto-hub ports to enumerate.

hubport→registerValueCallback() YHubPort

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

```
js      function registerValueCallback( callback)
nodejs  function registerValueCallback( callback)
php     function registerValueCallback( $callback)
cpp     int registerValueCallback( YHubPortValueCallback callback)
m       -(int) registerValueCallback : (YHubPortValueCallback) callback
pas     function registerValueCallback( callback: TYHubPortValueCallback): LongInt
vb      function registerValueCallback( ) As Integer
cs      int registerValueCallback( ValueCallback callback)
java    int registerValueCallback( UpdateCallback callback)
py      def registerValueCallback( callback)
```

The callback is invoked only during the execution of `ySleep` or `yHandleEvents`. This provides control over the time when the callback is triggered. For good responsiveness, remember to call one of these two functions periodically. To unregister a callback, pass a null pointer as argument.

Parameters :

callback the callback function to call, or a null pointer. The callback function should take two arguments: the function object of which the value has changed, and the character string describing the new advertised value.

hubport→set_enabled() YHubPort
hubport→setEnabled()

Changes the activation of the Yocto-hub port.

```
js      function set_enabled( newval)
nodejs  function set_enabled( newval)
php     function set_enabled( $newval)
cpp     int set_enabled( Y_ENABLED_enum newval)
m       -(int) setEnabled : (Y_ENABLED_enum) newval
pas     function set_enabled( newval: Integer): integer
vb      function set_enabled( ByVal newval As Integer) As Integer
cs      int set_enabled( int newval)
java    int set_enabled( int newval)
py      def set_enabled( newval)
cmd     YHubPort target set_enabled newval
```

If the port is enabled, the connected module is powered. Otherwise, port power is shut down.

Parameters :

newval either Y_ENABLED_FALSE or Y_ENABLED_TRUE, according to the activation of the Yocto-hub port

Returns :

YAPI_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

hubport→set_logicalName() YHubPort
hubport→setLogicalName()

Changes the logical name of the Yocto-hub port.

```
js      function set_logicalName( newval)
nodejs  function set_logicalName( newval)
php     function set_logicalName( $newval)
cpp     int set_logicalName( const string& newval)
m       -(int) setLogicalName : (NSString*) newval
pas     function set_logicalName( newval: string): integer
vb      function set_logicalName( ByVal newval As String) As Integer
cs      int set_logicalName( string newval)
java    int set_logicalName( String newval)
py      def set_logicalName( newval)
cmd     YHubPort target set_logicalName newval
```

You can use `yCheckLogicalName()` prior to this call to make sure that your parameter is valid. Remember to call the `saveToFlash()` method of the module if the modification must be kept.

Parameters :

newval a string corresponding to the logical name of the Yocto-hub port.

Returns :

`YAPI_SUCCESS` if the call succeeds.

On failure, throws an exception or returns a negative error code.

hubport→set_userData() YHubPort
hubport→setUserData()

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

```
js      function set_userData( data)
nodejs  function set_userData( data)
php     function set_userData( $data)
cpp     void set_userData( void* data)
m       -(void) setUserData : (void*) data
pas     procedure set_userData( data: Tobject)
vb      procedure set_userData( ByVal data As Object)
cs      void set_userData( object data)
java    void set_userData( Object data)
py      def set_userData( data)
```

This attribute is never touched by the API, and is at disposal of the caller to store a context.

Parameters :

data any kind of object to be stored

hubport→wait_async() YHubPort

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

```
js      function wait_async( callback, context)
nodejs  function wait_async( callback, context)
```

The callback function can therefore freely issue synchronous or asynchronous commands, without risking to block the Javascript VM.

Parameters :

callback callback function that is invoked when all pending commands on the module are completed. The callback function receives two arguments: the caller-specific context object and the receiving function object.

context caller-specific object that is passed as-is to the callback function

Returns :

nothing.

11.2. Wireless function interface

YWireless functions provides control over wireless network parameters and status for devices that are wireless-enabled.

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

```
js      <script type='text/javascript' src='yocto_wireless.js'></script>
nodejs  var yoctolib = require('yoctolib');
        var YWireless = yoctolib.YWireless;
php     require_once('yocto_wireless.php');
cpp     #include "yocto_wireless.h"
m       #import "yocto_wireless.h"
pas     uses yocto_wireless;
vb      yocto_wireless.vb
cs      yocto_wireless.cs
java    import com.yoctopuce.YoctoAPI.YWireless;
py      from yocto_wireless import *
```

Global functions

yFindWireless(**func**)

Retrieves a wireless lan interface for a given identifier.

yFirstWireless()

Starts the enumeration of wireless lan interfaces currently accessible.

YWireless methods

wireless→adhocNetwork(**ssid**, **securityKey**)

Changes the configuration of the wireless lan interface to create an ad-hoc wireless network, without using an access point.

wireless→describe()

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

wireless→get_advertisedValue()

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

wireless→get_channel()

Returns the 802.11 channel currently used, or 0 when the selected network has not been found.

wireless→get_detectedWlans()

Returns a list of YWlanRecord objects that describe detected Wireless networks.

wireless→get_errorMessage()

Returns the error message of the latest error with the wireless lan interface.

wireless→get_errorType()

Returns the numerical error code of the latest error with the wireless lan interface.

wireless→get_friendlyName()

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

wireless→get_functionDescriptor()

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

wireless→get_functionId()

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

wireless→get_hardwareId()

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

wireless→get_linkQuality()

Returns the link quality, expressed in percent.

wireless→get_logicalName()

Returns the logical name of the wireless lan interface.

wireless→get_message()

Returns the latest status message from the wireless interface.

wireless→get_module()

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

wireless→get_module_async(**callback**, **context**)

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

wireless→get_security()

Returns the security algorithm used by the selected wireless network.

wireless→get_ssid()

Returns the wireless network name (SSID).

wireless→get_userData()

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

wireless→isOnline()

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

wireless→isOnline_async(**callback**, **context**)

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

wireless→joinNetwork(**ssid**, **securityKey**)

Changes the configuration of the wireless lan interface to connect to an existing access point (infrastructure mode).

wireless→load(**msValidity**)

Preloads the wireless lan interface cache with a specified validity duration.

wireless→load_async(**msValidity**, **callback**, **context**)

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

wireless→nextWireless()

Continues the enumeration of wireless lan interfaces started using `yFirstWireless()`.

wireless→registerValueCallback(**callback**)

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

wireless→set_logicalName(**newval**)

Changes the logical name of the wireless lan interface.

wireless→set_userData(**data**)

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

wireless→wait_async(**callback**, **context**)

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

YWireless.FindWireless()
yFindWireless()

YWireless

Retrieves a wireless lan interface for a given identifier.

```
js      function yFindWireless( func)
nodejs  function FindWireless( func)
php     function yFindWireless( $func)
cpp     YWireless* yFindWireless( string func)
m       +(YWireless*) FindWireless : (NSString*) func
pas     function yFindWireless( func: string): TYWireless
vb      function yFindWireless( ByVal func As String) As YWireless
cs      YWireless FindWireless( string func)
java    YWireless FindWireless( String func)
py      def FindWireless( func)
```

The identifier can be specified using several formats:

- FunctionLogicalName
- ModuleSerialNumber.FunctionIdentifier
- ModuleSerialNumber.FunctionLogicalName
- ModuleLogicalName.FunctionIdentifier
- ModuleLogicalName.FunctionLogicalName

This function does not require that the wireless lan interface is online at the time it is invoked. The returned object is nevertheless valid. Use the method `YWireless.isOnline()` to test if the wireless lan interface is indeed online at a given time. In case of ambiguity when looking for a wireless lan interface by logical name, no error is notified: the first instance found is returned. The search is performed first by hardware name, then by logical name.

Parameters :

func a string that uniquely characterizes the wireless lan interface

Returns :

a `YWireless` object allowing you to drive the wireless lan interface.

YWireless.FirstWireless() YWireless yFirstWireless()

Starts the enumeration of wireless lan interfaces currently accessible.

```
js      function yFirstWireless( )
nodejs  function FirstWireless( )
php     function yFirstWireless( )
cpp     YWireless* yFirstWireless( )
m       +(YWireless*) FirstWireless
pas     function yFirstWireless( ): TYWireless
vb      function yFirstWireless( ) As YWireless
cs      YWireless FirstWireless( )
java    YWireless FirstWireless( )
py      def FirstWireless( )
```

Use the method `YWireless.nextWireless()` to iterate on next wireless lan interfaces.

Returns :

a pointer to a `YWireless` object, corresponding to the first wireless lan interface currently online, or a `null` pointer if there are none.

wireless→adhocNetwork() YWireless

Changes the configuration of the wireless lan interface to create an ad-hoc wireless network, without using an access point.

```
js      function adhocNetwork( ssid, securityKey)
nodejs  function adhocNetwork( ssid, securityKey)
php     function adhocNetwork( $ssid, $securityKey)
cpp     int adhocNetwork( string ssid, string securityKey)
m       -(int) adhocNetwork : (NSString*) ssid
                            : (NSString*) securityKey
pas     function adhocNetwork( ssid: string, securityKey: string): integer
vb      function adhocNetwork( ByVal ssid As String,
                               ByVal securityKey As String) As Integer
cs      int adhocNetwork( string ssid, string securityKey)
java    int adhocNetwork( String ssid, String securityKey)
py      def adhocNetwork( ssid, securityKey)
cmd     YWireless target adhocNetwork ssid securityKey
```

If a security key is specified, the network is protected by WEP128, since WPA is not standardized for ad-hoc networks. Remember to call the `saveToFlash()` method and then to reboot the module to apply this setting.

Parameters :

ssid	the name of the network to connect to
securityKey	the network key, as a character string

Returns :

`YAPI_SUCCESS` if the call succeeds.

On failure, throws an exception or returns a negative error code.

wireless→describe() YWireless

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

```
js      function describe( )
nodejs  function describe( )
php     function describe( )
cpp     string describe( )
m       -(NSString*) describe
pas     function describe( ): string
vb      function describe( ) As String
cs      string describe( )
java    String describe( )
py      def describe( )
```

More precisely, TYPE is the type of the function, NAME it the name used for the first access to the function, SERIAL is the serial number of the module if the module is connected or "unresolved", and FUNCTIONID is the hardware identifier of the function if the module is connected. For example, this method returns Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1 if the module is already connected or Relay(BadCustomeName.relay1)=unresolved if the module has not yet been connected. This method does not trigger any USB or TCP transaction and can therefore be used in a debugger.

Returns :
a string that describes the wireless lan interface (ex: Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1)

wireless→get_advertisedValue() YWireless
wireless→advertisedValue()

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

```
js      function get_advertisedValue( )
nodejs  function get_advertisedValue( )
php     function get_advertisedValue( )
cpp     string get_advertisedValue( )
m       -(NSString*) advertisedValue
pas     function get_advertisedValue( ): string
vb      function get_advertisedValue( ) As String
cs      string get_advertisedValue( )
java    String get_advertisedValue( )
py      def get_advertisedValue( )
cmd     YWireless target get_advertisedValue
```

Returns :

a string corresponding to the current value of the wireless lan interface (no more than 6 characters).

On failure, throws an exception or returns `Y_ADVERTISEDVALUE_INVALID`.

wireless→get_channel() YWireless
wireless→channel()

Returns the 802.11 channel currently used, or 0 when the selected network has not been found.

```
js      function get_channel( )
nodejs  function get_channel( )
php     function get_channel( )
cpp     int get_channel( )
m       -(int) channel
pas     function get_channel( ): LongInt
vb      function get_channel( ) As Integer
cs      int get_channel( )
java    int get_channel( )
py      def get_channel( )
cmd     YWireless target get_channel
```

Returns :

an integer corresponding to the 802.11 channel currently used, or 0 when the selected network has not been found

On failure, throws an exception or returns `Y_CHANNEL_INVALID`.

wireless→get_detectedWlans() YWireless
wireless→detectedWlans()

Returns a list of YWlanRecord objects that describe detected Wireless networks.

```
js      function get_detectedWlans( )
nodejs  function get_detectedWlans( )
php     function get_detectedWlans( )
cpp     vector<YWlanRecord> get_detectedWlans( )
m       -(NSMutableArray*) detectedWlans
pas     function get_detectedWlans( ): TYWlanRecordArray
vb      function get_detectedWlans( ) As List
cs      List<YWlanRecord> get_detectedWlans( )
java    ArrayList<YWlanRecord> get_detectedWlans( )
py      def get_detectedWlans( )
cmd     YWireless target get_detectedWlans
```

This list is not updated when the module is already connected to an acces point (infrastructure mode). To force an update of this list, `adhocNetwork()` must be called to disconnect the module from the current network. The returned list must be unallocated by the caller.

Returns :

a list of `YWlanRecord` objects, containing the SSID, channel, link quality and the type of security of the wireless network.

On failure, throws an exception or returns an empty list.

wireless→get_errorMessage() YWireless
wireless→errorMessage()

Returns the error message of the latest error with the wireless lan interface.

js	function **get_errorMessage**( )
nodejs	function **get_errorMessage**( )
php	function **get_errorMessage**( )
cpp	string **get_errorMessage**( )
m	-(NSString*) errorMessage
pas	function **get_errorMessage**( ): string
vb	function **get_errorMessage**( ) As String
cs	string **get_errorMessage**( )
java	String **get_errorMessage**( )
py	def **get_errorMessage**( )

This method is mostly useful when using the Yoctopuce library with exceptions disabled.

Returns :
a string corresponding to the latest error message that occured while using the wireless lan interface object

wireless→get_errorType() YWireless
wireless→errorType()

Returns the numerical error code of the latest error with the wireless lan interface.

```
js      function get_errorType( )
nodejs  function get_errorType( )
php     function get_errorType( )
cpp     YRETCODE get_errorType( )
pas     function get_errorType( ): YRETCODE
vb      function get_errorType( ) As YRETCODE
cs      YRETCODE get_errorType( )
java    int get_errorType( )
py      def get_errorType( )
```

This method is mostly useful when using the Yoctopuce library with exceptions disabled.

Returns :

a number corresponding to the code of the latest error that occured while using the wireless lan interface object

wireless→get_friendlyName() YWireless
wireless→friendlyName()

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

```
js      function get_friendlyName( )
nodejs  function get_friendlyName( )
php     function get_friendlyName( )
cpp     string get_friendlyName( )
m       -(NSString*) friendlyName
cs      string get_friendlyName( )
java    String get_friendlyName( )
py      def get_friendlyName( )
```

The returned string uses the logical names of the module and of the wireless lan interface if they are defined, otherwise the serial number of the module and the hardware identifier of the wireless lan interface (for exemple: `MyCustomName.relay1`)

Returns :

a string that uniquely identifies the wireless lan interface using logical names (ex: `MyCustomName.relay1`)

On failure, throws an exception or returns `Y_FRIENDLYNAME_INVALID`.

wireless→get_functionDescriptor() YWireless
wireless→functionDescriptor()

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

```
js      function get_functionDescriptor( )
nodejs  function get_functionDescriptor( )
php     function get_functionDescriptor( )
cpp     YFUN_DESCR get_functionDescriptor( )
m       -(YFUN_DESCR) functionDescriptor
pas     function get_functionDescriptor( ): YFUN_DESCR
vb      function get_functionDescriptor( ) As YFUN_DESCR
cs      YFUN_DESCR get_functionDescriptor( )
java    String get_functionDescriptor( )
py      def get_functionDescriptor( )
```

This identifier can be used to test if two instances of YFunction reference the same physical function on the same physical device.

Returns :

an identifier of type YFUN_DESCR.

If the function has never been contacted, the returned value is Y_FUNCTIONDESCRIPTOR_INVALID.

wireless→get_functionId() YWireless
wireless→functionId()

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

```
js      function get_functionId( )
nodejs  function get_functionId( )
php     function get_functionId( )
cpp     string get_functionId( )
m       -(NSString*) functionId
vb      function get_functionId( ) As String
cs      string get_functionId( )
java    String get_functionId( )
py      def get_functionId( )
```

For example `relay1`

Returns :

a string that identifies the wireless lan interface (ex: `relay1`)

On failure, throws an exception or returns `Y_FUNCTIONID_INVALID`.

wireless→get_hardwareId() YWireless
wireless→hardwareId()

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

```
js      function get_hardwareId( )
nodejs  function get_hardwareId( )
php     function get_hardwareId( )
cpp     string get_hardwareId( )
m       -(NSString*) hardwareId
vb      function get_hardwareId( ) As String
cs      string get_hardwareId( )
java    String get_hardwareId( )
py      def get_hardwareId( )
```

The unique hardware identifier is composed of the device serial number and of the hardware identifier of the wireless lan interface (for example RELAYLO1-123456.relay1).

Returns :

a string that uniquely identifies the wireless lan interface (ex: RELAYLO1-123456.relay1)

On failure, throws an exception or returns Y_HARDWAREID_INVALID.

wireless→get_linkQuality() YWireless
wireless→linkQuality()

Returns the link quality, expressed in percent.

```
js      function get_linkQuality( )
nodejs  function get_linkQuality( )
php     function get_linkQuality( )
cpp     int get_linkQuality( )
m       -(int) linkQuality
pas     function get_linkQuality( ): LongInt
vb      function get_linkQuality( ) As Integer
cs      int get_linkQuality( )
java    int get_linkQuality( )
py      def get_linkQuality( )
cmd     YWireless target get_linkQuality
```

Returns :
an integer corresponding to the link quality, expressed in percent

On failure, throws an exception or returns `Y_LINKQUALITY_INVALID`.

wireless→get_logicalName() YWireless
wireless→logicalName()

Returns the logical name of the wireless lan interface.

```
js      function get_logicalName( )
nodejs  function get_logicalName( )
php     function get_logicalName( )
cpp     string get_logicalName( )
m       -(NSString*) logicalName
pas     function get_logicalName( ): string
vb      function get_logicalName( ) As String
cs      string get_logicalName( )
java    String get_logicalName( )
py      def get_logicalName( )
cmd     YWireless target get_logicalName
```

Returns :
a string corresponding to the logical name of the wireless lan interface.

On failure, throws an exception or returns `Y_LOGICALNAME_INVALID`.

wireless→get_message() YWireless
wireless→message()

Returns the latest status message from the wireless interface.

```
js      function get_message( )
nodejs  function get_message( )
php     function get_message( )
cpp     string get_message( )
m       -(NSString*) message
pas     function get_message( ): string
vb      function get_message( ) As String
cs      string get_message( )
java    String get_message( )
py      def get_message( )
cmd     YWireless target get_message
```

Returns :
a string corresponding to the latest status message from the wireless interface

On failure, throws an exception or returns `Y_MESSAGE_INVALID`.

wireless→get_module() YWireless
wireless→module()

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

```
js      function get_module( )
nodejs  function get_module( )
php     function get_module( )
cpp     YModule * get_module( )
m       -(YModule*) module
pas     function get_module( ): TYModule
vb      function get_module( ) As YModule
cs      YModule get_module( )
java    YModule get_module( )
py      def get_module( )
```

If the function cannot be located on any module, the returned instance of YModule is not shown as on-line.

Returns :
an instance of YModule

wireless→get_module_async() YWireless
wireless→module_async()

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

```
js      function get_module_async( callback, context)
nodejs  function get_module_async( callback, context)
```

If the function cannot be located on any module, the returned YModule object does not show as on-line.

This asynchronous version exists only in Javascript. It uses a callback instead of a return value in order to avoid blocking Firefox javascript VM that does not implement context switching during blocking I/O calls. See the documentation section on asynchronous Javascript calls for more details.

Parameters :

callback callback function that is invoked when the result is known. The callback function receives three arguments: the caller-specific context object, the receiving function object and the requested YModule object

context caller-specific object that is passed as-is to the callback function

Returns :

nothing : the result is provided to the callback.

wireless→get_security() YWireless
wireless→security()

Returns the security algorithm used by the selected wireless network.

```
js      function get_security( )
nodejs  function get_security( )
php     function get_security( )
cpp     Y_SECURITY_enum get_security( )
m       -(Y_SECURITY_enum) security
pas     function get_security( ): Integer
vb      function get_security( ) As Integer
cs      int get_security( )
java    int get_security( )
py      def get_security( )
cmd     YWireless target get_security
```

Returns :

a value among `Y_SECURITY_UNKNOWN`, `Y_SECURITY_OPEN`, `Y_SECURITY_WEP`, `Y_SECURITY_WPA` and `Y_SECURITY_WPA2` corresponding to the security algorithm used by the selected wireless network

On failure, throws an exception or returns `Y_SECURITY_INVALID`.

wireless→get_ssid() YWireless
wireless→ssid()

Returns the wireless network name (SSID).

```
js      function get_ssid( )
nodejs  function get_ssid( )
php     function get_ssid( )
cpp     string get_ssid( )
m       -(NSString*) ssid
pas     function get_ssid( ): string
vb      function get_ssid( ) As String
cs      string get_ssid( )
java    String get_ssid( )
py      def get_ssid( )
cmd     YWireless target get_ssid
```

Returns :

a string corresponding to the wireless network name (SSID)

On failure, throws an exception or returns `Y_SSID_INVALID`.

wireless→get_userData() YWireless
wireless→userData()

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

```
js      function get_userData( )
nodejs  function get_userData( )
php     function get_userData( )
cpp     void * get_userData( )
m       -(void*) userData
pas     function get_userData( ): Tobject
vb      function get_userData( ) As Object
cs      object get_userData( )
java    Object get_userData( )
py      def get_userData( )
```

This attribute is never touched directly by the API, and is at disposal of the caller to store a context.

Returns :
the object stored previously by the caller.

wireless→isOnline() YWireless

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

```
js      function isOnline( )
nodejs  function isOnline( )
php     function isOnline( )
cpp     bool isOnline( )
m       -(BOOL) isOnline
pas     function isOnline( ): boolean
vb      function isOnline( ) As Boolean
cs      bool isOnline( )
java    boolean isOnline( )
py      def isOnline( )
```

If there is a cached value for the wireless lan interface in cache, that has not yet expired, the device is considered reachable. No exception is raised if there is an error while trying to contact the device hosting the wireless lan interface.

Returns :

`true` if the wireless lan interface can be reached, and `false` otherwise

wireless→isOnline_async() YWireless

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

```
js      function isOnline_async( callback, context)
nodejs  function isOnline_async( callback, context)
```

If there is a cached value for the wireless lan interface in cache, that has not yet expired, the device is considered reachable. No exception is raised if there is an error while trying to contact the device hosting the requested function.

This asynchronous version exists only in Javascript. It uses a callback instead of a return value in order to avoid blocking the Javascript virtual machine.

Parameters :

callback callback function that is invoked when the result is known. The callback function receives three arguments: the caller-specific context object, the receiving function object and the boolean result

context caller-specific object that is passed as-is to the callback function

Returns :

nothing : the result is provided to the callback.

wireless→joinNetwork() YWireless

Changes the configuration of the wireless lan interface to connect to an existing access point (infrastructure mode).

```
js      function joinNetwork( ssid, securityKey)
nodejs  function joinNetwork( ssid, securityKey)
php     function joinNetwork( $ssid, $securityKey)
cpp     int joinNetwork( string ssid, string securityKey)
m       -(int) joinNetwork : (NSString*) ssid
                           : (NSString*) securityKey
pas     function joinNetwork( ssid: string, securityKey: string): integer
vb      function joinNetwork( ByVal ssid As String,
                              ByVal securityKey As String) As Integer
cs      int joinNetwork( string ssid, string securityKey)
java    int joinNetwork( String ssid, String securityKey)
py      def joinNetwork( ssid, securityKey)
cmd     YWireless target joinNetwork ssid securityKey
```

Remember to call the `saveToFlash()` method and then to reboot the module to apply this setting.

Parameters :

ssid	the name of the network to connect to
securityKey	the network key, as a character string

Returns :

`YAPI_SUCCESS` if the call succeeds.

On failure, throws an exception or returns a negative error code.

wireless→load() YWireless

Preloads the wireless lan interface cache with a specified validity duration.

```
js      function load( msValidity)
nodejs  function load( msValidity)
php     function load( $msValidity)
cpp     YRETCODE load( int msValidity)
m       -(YRETCODE) load : (int) msValidity
pas     function load( msValidity: integer): YRETCODE
vb      function load( ByVal msValidity As Integer) As YRETCODE
cs      YRETCODE load( int msValidity)
java    int load( long msValidity)
py      def load( msValidity)
```

By default, whenever accessing a device, all function attributes are kept in cache for the standard duration (5 ms). This method can be used to temporarily mark the cache as valid for a longer period, in order to reduce network trafic for instance.

Parameters :

msValidity an integer corresponding to the validity attributed to the loaded function parameters, in milliseconds

Returns :

YAPI_SUCCESS when the call succeeds.

On failure, throws an exception or returns a negative error code.

wireless→load_async() YWireless

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

```
js      function load_async( msValidity, callback, context)
nodejs  function load_async( msValidity, callback, context)
```

By default, whenever accessing a device, all function attributes are kept in cache for the standard duration (5 ms). This method can be used to temporarily mark the cache as valid for a longer period, in order to reduce network trafic for instance.

This asynchronous version exists only in Javascript. It uses a callback instead of a return value in order to avoid blocking the Javascript virtual machine.

Parameters :

msValidity	an integer corresponding to the validity of the loaded function parameters, in milliseconds
callback	callback function that is invoked when the result is known. The callback function receives three arguments: the caller-specific context object, the receiving function object and the error code (or YAPI_SUCCESS)
context	caller-specific object that is passed as-is to the callback function

Returns :

nothing : the result is provided to the callback.

wireless→nextWireless() YWireless

Continues the enumeration of wireless lan interfaces started using `yFirstWireless()`.

```
js      function nextWireless( )
nodejs  function nextWireless( )
php     function nextWireless( )
cpp     YWireless * nextWireless( )
m       -(YWireless*) nextWireless
pas     function nextWireless( ): TYWireless
vb      function nextWireless( ) As YWireless
cs      YWireless nextWireless( )
java    YWireless nextWireless( )
py      def nextWireless( )
```

Returns :

a pointer to a `YWireless` object, corresponding to a wireless lan interface currently online, or a `null` pointer if there are no more wireless lan interfaces to enumerate.

wireless→registerValueCallback() YWireless

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

```
js      function registerValueCallback( callback)
nodejs  function registerValueCallback( callback)
php     function registerValueCallback( $callback)
cpp     int registerValueCallback( YWirelessValueCallback callback)
m       -(int) registerValueCallback : (YWirelessValueCallback) callback
pas     function registerValueCallback( callback: TYWirelessValueCallback): LongInt
vb      function registerValueCallback( ) As Integer
cs      int registerValueCallback( ValueCallback callback)
java    int registerValueCallback( UpdateCallback callback)
py      def registerValueCallback( callback)
```

The callback is invoked only during the execution of `ySleep` or `yHandleEvents`. This provides control over the time when the callback is triggered. For good responsiveness, remember to call one of these two functions periodically. To unregister a callback, pass a null pointer as argument.

Parameters :

callback the callback function to call, or a null pointer. The callback function should take two arguments: the function object of which the value has changed, and the character string describing the new advertised value.

wireless→set_logicalName() YWireless
wireless→setLogicalName()

Changes the logical name of the wireless lan interface.

```
js      function set_logicalName( newval)
nodejs  function set_logicalName( newval)
php     function set_logicalName( $newval)
cpp     int set_logicalName( const string& newval)
m       -(int) setLogicalName : (NSString*) newval
pas     function set_logicalName( newval: string): integer
vb      function set_logicalName( ByVal newval As String) As Integer
cs      int set_logicalName( string newval)
java    int set_logicalName( String newval)
py      def set_logicalName( newval)
cmd     YWireless target set_logicalName newval
```

You can use `yCheckLogicalName()` prior to this call to make sure that your parameter is valid. Remember to call the `saveToFlash()` method of the module if the modification must be kept.

Parameters :

newval a string corresponding to the logical name of the wireless lan interface.

Returns :

`YAPI_SUCCESS` if the call succeeds.

On failure, throws an exception or returns a negative error code.

wireless→set_userData() YWireless
wireless→setUserData()

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

```
js      function set_userData( data)
nodejs  function set_userData( data)
php     function set_userData( $data)
cpp     void set_userData( void* data)
m       -(void) setUserData : (void*) data
pas     procedure set_userData( data: Tobject)
vb      procedure set_userData( ByVal data As Object)
cs      void set_userData( object data)
java    void set_userData( Object data)
py      def set_userData( data)
```

This attribute is never touched by the API, and is at disposal of the caller to store a context.

Parameters :

data any kind of object to be stored

wireless→wait_async() YWireless

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

```
js      function wait_async( callback, context)
nodejs  function wait_async( callback, context)
```

The callback function can therefore freely issue synchronous or asynchronous commands, without risking to block the Javascript VM.

Parameters :

callback callback function that is invoked when all pending commands on the module are completed. The callback function receives two arguments: the caller-specific context object and the receiving function object.

context caller-specific object that is passed as-is to the callback function

Returns :

nothing.

11.3. Network function interface

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

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

```
js      <script type='text/javascript' src='yocto_network.js'></script>
nodejs  var yoctolib = require('yoctolib');
        var YNetwork = yoctolib.YNetwork;
php     require_once('yocto_network.php');
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;
py      from yocto_network import *
```

Global functions
yFindNetwork(**func**) Retrieves a network interface for a given identifier.
yFirstNetwork() Starts the enumeration of network interfaces currently accessible.
`YNetwork` methods
network→callbackLogin(**username**, **password**) Connects to the notification callback and saves the credentials required to log into it.
network→describe() Returns a short text that describes unambiguously the instance of the network interface in the form `TYPE(NAME)=SERIAL.FUNCTIONID`.
network→get_adminPassword() Returns a hash string if a password has been set for user "admin", or an empty string otherwise.
network→get_advertisedValue() Returns the current value of the network interface (no more than 6 characters).
network→get_callbackCredentials() Returns a hashed version of the notification callback credentials if set, or an empty string otherwise.
network→get_callbackEncoding() Returns the encoding standard to use for representing notification values.
network→get_callbackMaxDelay() Returns the maximum waiting time between two callback notifications, in seconds.
network→get_callbackMethod() Returns the HTTP method used to notify callbacks for significant state changes.
network→get_callbackMinDelay() Returns the minimum waiting time between two callback notifications, in seconds.
network→get_callbackUrl() Returns the callback URL to notify of significant state changes.
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).

network→get_errorMessage()

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

network→get_errorType()

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

network→get_friendlyName()

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

network→get_functionDescriptor()

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

network→get_functionId()

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

network→get_hardwareId()

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

network→get_ipAddress()

Returns the IP address currently in use by the device.

network→get_logicalName()

Returns the logical name of the network interface.

network→get_macAddress()

Returns the MAC address of the network interface.

network→get_module()

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

network→get_module_async(**callback**, **context**)

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

network→get_poeCurrent()

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

network→get_primaryDNS()

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

network→get_readiness()

Returns the current established working mode of the network interface.

network→get_router()

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

network→get_secondaryDNS()

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

network→get_subnetMask()

Returns the subnet mask currently used by the device.

network→get_userData()

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

network→get_userPassword()

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

network→get_wwwWatchdogDelay()

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

network→isOnline()

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

network→isOnline_async(**callback**, **context**)

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

network→load(msValidity)

Preloads the network interface cache with a specified validity duration.

network→load_async(msValidity, callback, context)

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

network→nextNetwork()

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

network→ping(host)

Pings str_host to test the network connectivity.

network→registerValueCallback(callback)

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

network→set_adminPassword(newval)

Changes the password for the "admin" user.

network→set_callbackCredentials(newval)

Changes the credentials required to connect to the callback address.

network→set_callbackEncoding(newval)

Changes the encoding standard to use for representing notification values.

network→set_callbackMaxDelay(newval)

Changes the maximum waiting time between two callback notifications, in seconds.

network→set_callbackMethod(newval)

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

network→set_callbackMinDelay(newval)

Changes the minimum waiting time between two callback notifications, in seconds.

network→set_callbackUrl(newval)

Changes the callback URL to notify significant state changes.

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).

network→set_logicalName(newval)

Changes the logical name of the network interface.

network→set_primaryDNS(newval)

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

network→set_secondaryDNS(newval)

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

network→set_userData(data)

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

network→set_userPassword(newval)

Changes the password for the "user" user.

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.

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.

network→useStaticIP(ipAddress, subnetMaskLen, router)

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

network→wait_async(callback, context)

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

YNetwork.FindNetwork() YNetwork yFindNetwork()

Retrieves a network interface for a given identifier.

```
js      function yFindNetwork( func)
nodejs  function FindNetwork( func)
php     function yFindNetwork( $func)
cpp     YNetwork* yFindNetwork( const string& func)
m       +(YNetwork*) FindNetwork :(NSString*) func
pas     function yFindNetwork( func: string): TYNetwork
vb      function yFindNetwork( ByVal func As String) As YNetwork
cs      YNetwork FindNetwork( string func)
java    YNetwork FindNetwork( String func)
py      def FindNetwork( func)
```

The identifier can be specified using several formats:

- FunctionLogicalName
- ModuleSerialNumber.FunctionIdentifier
- ModuleSerialNumber.FunctionLogicalName
- ModuleLogicalName.FunctionIdentifier
- ModuleLogicalName.FunctionLogicalName

This function does not require that the network interface is online at the time it is invoked. The returned object is nevertheless valid. Use the method `YNetwork.isOnline()` to test if the network interface is indeed online at a given time. In case of ambiguity when looking for a network interface by logical name, no error is notified: the first instance found is returned. The search is performed first by hardware name, then by logical name.

Parameters :

func a string that uniquely characterizes the network interface

Returns :

a `YNetwork` object allowing you to drive the network interface.

YNetwork.FirstNetwork() YNetwork
yFirstNetwork()

Starts the enumeration of network interfaces currently accessible.

```
js      function yFirstNetwork( )
nodejs  function FirstNetwork( )
php     function yFirstNetwork( )
cpp     YNetwork* yFirstNetwork( )
m       +(YNetwork*) FirstNetwork
pas     function yFirstNetwork( ): TYNetwork
vb      function yFirstNetwork( ) As YNetwork
cs      YNetwork FirstNetwork( )
java    YNetwork FirstNetwork( )
py      def FirstNetwork( )
```

Use the method `YNetwork.nextNetwork()` to iterate on next network interfaces.

Returns :

a pointer to a `YNetwork` object, corresponding to the first network interface currently online, or a `null` pointer if there are none.

network→callbackLogin() YNetwork

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

```
js      function callbackLogin( username, password)
nodejs  function callbackLogin( username, password)
php     function callbackLogin( $username, $password)
cpp     int callbackLogin( string username, string password)
m       -(int) callbackLogin : (NSString*) username : (NSString*) password
pas     function callbackLogin( username: string, password: string): integer
vb      function callbackLogin( ByVal username As String,
                                ByVal password As String) As Integer
cs      int callbackLogin( string username, string password)
java    int callbackLogin( String username, String password)
py      def callbackLogin( username, password)
cmd     YNetwork target callbackLogin username password
```

The password is not stored into the module, only a hashed copy of the credentials are saved. Remember to call the `saveToFlash()` method of the module if the modification must be kept.

Parameters :

username username required to log to the callback

password password required to log to the callback

Returns :

`YAPI_SUCCESS` if the call succeeds.

On failure, throws an exception or returns a negative error code.

network→describe() YNetwork

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

```
js      function describe( )
nodejs  function describe( )
php     function describe( )
cpp     string describe( )
m       -(NSString*) describe
pas     function describe( ): string
vb      function describe( ) As String
cs      string describe( )
java    String describe( )
py      def describe( )
```

More precisely, TYPE is the type of the function, NAME it the name used for the first access to the function, SERIAL is the serial number of the module if the module is connected or "unresolved", and FUNCTIONID is the hardware identifier of the function if the module is connected. For example, this method returns Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1 if the module is already connected or Relay(BadCustomeName.relay1)=unresolved if the module has not yet been connected. This method does not trigger any USB or TCP transaction and can therefore be used in a debugger.

Returns :

a string that describes the network interface (ex: Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1)

network→get_adminPassword() YNetwork
network→adminPassword()

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

```
js      function get_adminPassword( )
nodejs  function get_adminPassword( )
php     function get_adminPassword( )
cpp     string get_adminPassword( )
m       -(NSString*) adminPassword
pas     function get_adminPassword( ): string
vb      function get_adminPassword( ) As String
cs      string get_adminPassword( )
java    String get_adminPassword( )
py      def get_adminPassword( )
cmd     YNetwork target get_adminPassword
```

Returns :

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

On failure, throws an exception or returns `Y_ADMINPASSWORD_INVALID`.

network→get_advertisedValue() YNetwork
network→advertisedValue()

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

js	function **get_advertisedValue**( )
nodejs	function **get_advertisedValue**( )
php	function **get_advertisedValue**( )
cpp	string **get_advertisedValue**( )
m	-(NSString*) advertisedValue
pas	function **get_advertisedValue**( ): string
vb	function **get_advertisedValue**( ) As String
cs	string **get_advertisedValue**( )
java	String **get_advertisedValue**( )
py	def **get_advertisedValue**( )
cmd	YNetwork **target get_advertisedValue**

Returns :

a string corresponding to the current value of the network interface (no more than 6 characters).

On failure, throws an exception or returns `Y_ADVERTISEDVALUE_INVALID`.

network→get_callbackCredentials() YNetwork
network→callbackCredentials()

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

```
js      function get_callbackCredentials( )
nodejs  function get_callbackCredentials( )
php     function get_callbackCredentials( )
cpp     string get_callbackCredentials( )
m       -(NSString*) callbackCredentials
pas     function get_callbackCredentials( ): string
vb      function get_callbackCredentials( ) As String
cs      string get_callbackCredentials( )
java    String get_callbackCredentials( )
py      def get_callbackCredentials( )
cmd     YNetwork target get_callbackCredentials
```

Returns :

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

On failure, throws an exception or returns `Y_CALLBACKCREDENTIALS_INVALID`.

network→get_callbackEncoding() YNetwork
network→callbackEncoding()

Returns the encoding standard to use for representing notification values.

js	function **get_callbackEncoding**( )
nodejs	function **get_callbackEncoding**( )
php	function **get_callbackEncoding**( )
cpp	Y_CALLBACKENCODING_enum **get_callbackEncoding**( )
m	-(Y_CALLBACKENCODING_enum) callbackEncoding
pas	function **get_callbackEncoding**( ): Integer
vb	function **get_callbackEncoding**( ) As Integer
cs	int **get_callbackEncoding**( )
java	int **get_callbackEncoding**( )
py	def **get_callbackEncoding**( )
cmd	YNetwork **target get_callbackEncoding**

Returns :

a value among `Y_CALLBACKENCODING_FORM`, `Y_CALLBACKENCODING_JSON`, `Y_CALLBACKENCODING_JSON_ARRAY`, `Y_CALLBACKENCODING_CSV` and `Y_CALLBACKENCODING_YOCTO_API` corresponding to the encoding standard to use for representing notification values

On failure, throws an exception or returns `Y_CALLBACKENCODING_INVALID`.

network→get_callbackMaxDelay() YNetwork
network→callbackMaxDelay()

Returns the maximum waiting time between two callback notifications, in seconds.

js	function **get_callbackMaxDelay**( )
nodejs	function **get_callbackMaxDelay**( )
php	function **get_callbackMaxDelay**( )
cpp	int **get_callbackMaxDelay**( )
m	-(int) callbackMaxDelay
pas	function **get_callbackMaxDelay**( ): LongInt
vb	function **get_callbackMaxDelay**( ) As Integer
cs	int **get_callbackMaxDelay**( )
java	int **get_callbackMaxDelay**( )
py	def **get_callbackMaxDelay**( )
cmd	YNetwork **target get_callbackMaxDelay**

Returns :

an integer corresponding to the maximum waiting time between two callback notifications, in seconds

On failure, throws an exception or returns `Y_CALLBACKMAXDELAY_INVALID`.

network→get_callbackMethod() YNetwork
network→callbackMethod()

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

```
js      function get_callbackMethod( )
nodejs  function get_callbackMethod( )
php     function get_callbackMethod( )
cpp     Y_CALLBACKMETHOD_enum get_callbackMethod( )
m       -(Y_CALLBACKMETHOD_enum) callbackMethod
pas     function get_callbackMethod( ): Integer
vb      function get_callbackMethod( ) As Integer
cs      int get_callbackMethod( )
java    int get_callbackMethod( )
py      def get_callbackMethod( )
cmd     YNetwork target get_callbackMethod
```

Returns :

a value among `Y_CALLBACKMETHOD_POST`, `Y_CALLBACKMETHOD_GET` and `Y_CALLBACKMETHOD_PUT` corresponding to the HTTP method used to notify callbacks for significant state changes

On failure, throws an exception or returns `Y_CALLBACKMETHOD_INVALID`.

network→get_callbackMinDelay() YNetwork
network→callbackMinDelay()

Returns the minimum waiting time between two callback notifications, in seconds.

```
js      function get_callbackMinDelay( )
nodejs  function get_callbackMinDelay( )
php     function get_callbackMinDelay( )
cpp     int get_callbackMinDelay( )
m       -(int) callbackMinDelay
pas     function get_callbackMinDelay( ): LongInt
vb      function get_callbackMinDelay( ) As Integer
cs      int get_callbackMinDelay( )
java    int get_callbackMinDelay( )
py      def get_callbackMinDelay( )
cmd     YNetwork target get_callbackMinDelay
```

Returns :

an integer corresponding to the minimum waiting time between two callback notifications, in seconds

On failure, throws an exception or returns `Y_CALLBACKMINDELAY_INVALID`.

network→get_callbackUrl() YNetwork
network→callbackUrl()

Returns the callback URL to notify of significant state changes.

```
js      function get_callbackUrl( )
nodejs  function get_callbackUrl( )
php     function get_callbackUrl( )
cpp     string get_callbackUrl( )
m       -(NSString*) callbackUrl
pas     function get_callbackUrl( ): string
vb      function get_callbackUrl( ) As String
cs      string get_callbackUrl( )
java    String get_callbackUrl( )
py      def get_callbackUrl( )
cmd     YNetwork target get_callbackUrl
```

Returns :
a string corresponding to the callback URL to notify of significant state changes

On failure, throws an exception or returns `Y_CALLBACKURL_INVALID`.

network→get_discoverable() YNetwork
network→discoverable()

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

js	function **get_discoverable**( )
nodejs	function **get_discoverable**( )
php	function **get_discoverable**( )
cpp	Y_DISCOVERABLE_enum **get_discoverable**( )
m	-(Y_DISCOVERABLE_enum) discoverable
pas	function **get_discoverable**( ): Integer
vb	function **get_discoverable**( ) As Integer
cs	int **get_discoverable**( )
java	int **get_discoverable**( )
py	def **get_discoverable**( )
cmd	YNetwork **target** **get_discoverable**

Returns :

either `Y_DISCOVERABLE_FALSE` or `Y_DISCOVERABLE_TRUE`, according to the activation state of the multicast announce protocols to allow easy discovery of the module in the network neighborhood (uPnP/Bonjour protocol)

On failure, throws an exception or returns `Y_DISCOVERABLE_INVALID`.

network→get_errorMessage() YNetwork
network→errorMessage()

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

```
js      function get_errorMessage( )
nodejs  function get_errorMessage( )
php     function get_errorMessage( )
cpp     string get_errorMessage( )
m       -(NSString*) errorMessage
pas     function get_errorMessage( ): string
vb      function get_errorMessage( ) As String
cs      string get_errorMessage( )
java    String get_errorMessage( )
py      def get_errorMessage( )
```

This method is mostly useful when using the Yoctopuce library with exceptions disabled.

Returns :
a string corresponding to the latest error message that occured while using the network interface object

network→get_errorType() YNetwork
network→errorType()

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

```
js      function get_errorType( )
nodejs  function get_errorType( )
php     function get_errorType( )
cpp     YRETCODE get_errorType( )
pas     function get_errorType( ): YRETCODE
vb      function get_errorType( ) As YRETCODE
cs      YRETCODE get_errorType( )
java    int get_errorType( )
py      def get_errorType( )
```

This method is mostly useful when using the Yoctopuce library with exceptions disabled.

Returns :
a number corresponding to the code of the latest error that occured while using the network interface object

network→get_friendlyName() YNetwork
network→friendlyName()

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

```
js      function get_friendlyName( )
nodejs  function get_friendlyName( )
php     function get_friendlyName( )
cpp     string get_friendlyName( )
m       -(NSString*) friendlyName
cs      string get_friendlyName( )
java    String get_friendlyName( )
py      def get_friendlyName( )
```

The returned string uses the logical names of the module and of the network interface if they are defined, otherwise the serial number of the module and the hardware identifier of the network interface (for exemple: MyCustomName.relay1)

Returns :

a string that uniquely identifies the network interface using logical names (ex: MyCustomName.relay1)

On failure, throws an exception or returns Y_FRIENDLYNAME_INVALID.

network→get_functionDescriptor() YNetwork
network→functionDescriptor()

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

js	function **get_functionDescriptor**( )
nodejs	function **get_functionDescriptor**( )
php	function **get_functionDescriptor**( )
cpp	YFUN_DESCR **get_functionDescriptor**( )
m	-(YFUN_DESCR) functionDescriptor
pas	function **get_functionDescriptor**( ): YFUN_DESCR
vb	function **get_functionDescriptor**( ) As YFUN_DESCR
cs	YFUN_DESCR **get_functionDescriptor**( )
java	String **get_functionDescriptor**( )
py	def **get_functionDescriptor**( )

This identifier can be used to test if two instances of YFunction reference the same physical function on the same physical device.

Returns :

an identifier of type YFUN_DESCR.

If the function has never been contacted, the returned value is Y_FUNCTIONDESCRIPTOR_INVALID.

network→get_functionId() YNetwork
network→functionId()

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

```
js      function get_functionId( )
nodejs  function get_functionId( )
php     function get_functionId( )
cpp     string get_functionId( )
m       -(NSString*) functionId
vb      function get_functionId( ) As String
cs      string get_functionId( )
java    String get_functionId( )
py      def get_functionId( )
```

For example `relay1`

Returns :

a string that identifies the network interface (ex: `relay1`)

On failure, throws an exception or returns `Y_FUNCTIONID_INVALID`.

network→get_hardwareId() YNetwork
network→hardwareId()

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

```
js       function get_hardwareId( )
nodejs   function get_hardwareId( )
php      function get_hardwareId( )
cpp      string get_hardwareId( )
m        -(NSString*) hardwareId
vb       function get_hardwareId( ) As String
cs       string get_hardwareId( )
java     String get_hardwareId( )
py       def get_hardwareId( )
```

The unique hardware identifier is composed of the device serial number and of the hardware identifier of the network interface (for example RELAYLO1-123456.relay1).

Returns :

a string that uniquely identifies the network interface (ex: RELAYLO1-123456.relay1)

On failure, throws an exception or returns Y_HARDWAREID_INVALID.

network→get_ipAddress() YNetwork
network→ipAddress()

Returns the IP address currently in use by the device.

```
js      function get_ipAddress( )
nodejs  function get_ipAddress( )
php     function get_ipAddress( )
cpp     string get_ipAddress( )
m       -(NSString*) ipAddress
pas     function get_ipAddress( ): string
vb      function get_ipAddress( ) As String
cs      string get_ipAddress( )
java    String get_ipAddress( )
py      def get_ipAddress( )
cmd     YNetwork target get_ipAddress
```

The address may have been configured statically, or provided by a DHCP server.

Returns :

a string corresponding to the IP address currently in use by the device

On failure, throws an exception or returns `Y_IPADDRESS_INVALID`.

network→get_logicalName() YNetwork
network→logicalName()

Returns the logical name of the network interface.

```
js       function get_logicalName( )
nodejs   function get_logicalName( )
php      function get_logicalName( )
cpp      string get_logicalName( )
m        -(NSString*) logicalName
pas      function get_logicalName( ): string
vb       function get_logicalName( ) As String
cs       string get_logicalName( )
java     String get_logicalName( )
py       def get_logicalName( )
cmd      YNetwork target get_logicalName
```

Returns :
a string corresponding to the logical name of the network interface.

On failure, throws an exception or returns `Y_LOGICALNAME_INVALID`.

network→get_macAddress() YNetwork
network→macAddress()

Returns the MAC address of the network interface.

```
js      function get_macAddress( )
nodejs  function get_macAddress( )
php     function get_macAddress( )
cpp     string get_macAddress( )
m       -(NSString*) macAddress
pas     function get_macAddress( ): string
vb      function get_macAddress( ) As String
cs      string get_macAddress( )
java    String get_macAddress( )
py      def get_macAddress( )
cmd     YNetwork target get_macAddress
```

The MAC address is also available on a sticker on the module, in both numeric and barcode forms.

Returns :
a string corresponding to the MAC address of the network interface

On failure, throws an exception or returns `Y_MACADDRESS_INVALID`.

network→get_module() YNetwork
network→module()

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

```
js      function get_module( )
nodejs  function get_module( )
php     function get_module( )
cpp     YModule * get_module( )
m       -(YModule*) module
pas     function get_module( ): TYModule
vb      function get_module( ) As YModule
cs      YModule get_module( )
java    YModule get_module( )
py      def get_module( )
```

If the function cannot be located on any module, the returned instance of YModule is not shown as on-line.

Returns :
an instance of YModule

network→get_module_async() YNetwork
network→module_async()

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

```
  js  function get_module_async( callback, context)
nodejs function get_module_async( callback, context)
```

If the function cannot be located on any module, the returned YModule object does not show as online.

This asynchronous version exists only in Javascript. It uses a callback instead of a return value in order to avoid blocking Firefox javascript VM that does not implement context switching during blocking I/O calls. See the documentation section on asynchronous Javascript calls for more details.

Parameters :

callback callback function that is invoked when the result is known. The callback function receives three arguments: the caller-specific context object, the receiving function object and the requested YModule object

context caller-specific object that is passed as-is to the callback function

Returns :

nothing : the result is provided to the callback.

network→get_poeCurrent() YNetwork
network→poeCurrent()

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

js	function **get_poeCurrent**( )
nodejs	function **get_poeCurrent**( )
php	function **get_poeCurrent**( )
cpp	int **get_poeCurrent**( )
m	-(int) poeCurrent
pas	function **get_poeCurrent**( ): LongInt
vb	function **get_poeCurrent**( ) As Integer
cs	int **get_poeCurrent**( )
java	int **get_poeCurrent**( )
py	def **get_poeCurrent**( )
cmd	YNetwork **target get_poeCurrent**

The current consumption is measured after converting PoE source to 5 Volt, and should never exceed 1800 mA.

Returns :

an integer corresponding to the current consumed by the module from Power-over-Ethernet (PoE), in milli-amps

On failure, throws an exception or returns `Y_POECURRENT_INVALID`.

network→get_primaryDNS() YNetwork
network→primaryDNS()

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

```
js      function get_primaryDNS( )
nodejs  function get_primaryDNS( )
php     function get_primaryDNS( )
cpp     string get_primaryDNS( )
m       -(NSString*) primaryDNS
pas     function get_primaryDNS( ): string
vb      function get_primaryDNS( ) As String
cs      string get_primaryDNS( )
java    String get_primaryDNS( )
py      def get_primaryDNS( )
cmd     YNetwork target get_primaryDNS
```

Returns :
a string corresponding to the IP address of the primary name server to be used by the module

On failure, throws an exception or returns `Y_PRIMARYDNS_INVALID`.

network→get_readiness() YNetwork
network→readiness()

Returns the current established working mode of the network interface.

```
js      function get_readiness( )
nodejs  function get_readiness( )
php     function get_readiness( )
cpp     Y_READINESS_enum get_readiness( )
m       -(Y_READINESS_enum) readiness
pas     function get_readiness( ): Integer
vb      function get_readiness( ) As Integer
cs      int get_readiness( )
java    int get_readiness( )
py      def get_readiness( )
cmd     YNetwork target get_readiness
```

Level zero (DOWN_0) means that no hardware link has been detected. Either there is no signal on the network cable, or the selected wireless access point cannot be detected. Level 1 (LIVE_1) is reached when the network is detected, but is not yet connected. For a wireless network, this shows that the requested SSID is present. Level 2 (LINK_2) is reached when the hardware connection is established. For a wired network connection, level 2 means that the cable is attached at both ends. For a connection to a wireless access point, it shows that the security parameters are properly configured. For an ad-hoc wireless connection, it means that there is at least one other device connected on the ad-hoc network. Level 3 (DHCP_3) is reached when an IP address has been obtained using DHCP. Level 4 (DNS_4) is reached when the DNS server is reachable on the network. Level 5 (WWW_5) is reached when global connectivity is demonstrated by properly loading the current time from an NTP server.

Returns :

a value among `Y_READINESS_DOWN`, `Y_READINESS_EXISTS`, `Y_READINESS_LINKED`, `Y_READINESS_LAN_OK` and `Y_READINESS_WWW_OK` corresponding to the current established working mode of the network interface

On failure, throws an exception or returns `Y_READINESS_INVALID`.

network→get_router() YNetwork
network→router()

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

```
js      function get_router( )
nodejs  function get_router( )
php     function get_router( )
cpp     string get_router( )
m       -(NSString*) router
pas     function get_router( ): string
vb      function get_router( ) As String
cs      string get_router( )
java    String get_router( )
py      def get_router( )
cmd     YNetwork target get_router
```

Returns :
a string corresponding to the IP address of the router on the device subnet (default gateway)

On failure, throws an exception or returns `Y_ROUTER_INVALID`.

network→get_secondaryDNS() YNetwork
network→secondaryDNS()

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

```
js      function get_secondaryDNS( )
nodejs  function get_secondaryDNS( )
php     function get_secondaryDNS( )
cpp     string get_secondaryDNS( )
m       -(NSString*) secondaryDNS
pas     function get_secondaryDNS( ): string
vb      function get_secondaryDNS( ) As String
cs      string get_secondaryDNS( )
java    String get_secondaryDNS( )
py      def get_secondaryDNS( )
cmd     YNetwork target get_secondaryDNS
```

Returns :

a string corresponding to the IP address of the secondary name server to be used by the module

On failure, throws an exception or returns `Y_SECONDARYDNS_INVALID`.

network→get_subnetMask() YNetwork
network→subnetMask()

Returns the subnet mask currently used by the device.

```
js      function get_subnetMask( )
nodejs  function get_subnetMask( )
php     function get_subnetMask( )
cpp     string get_subnetMask( )
m       -(NSString*) subnetMask
pas     function get_subnetMask( ): string
vb      function get_subnetMask( ) As String
cs      string get_subnetMask( )
java    String get_subnetMask( )
py      def get_subnetMask( )
cmd     YNetwork target get_subnetMask
```

Returns :
a string corresponding to the subnet mask currently used by the device

On failure, throws an exception or returns `Y_SUBNETMASK_INVALID`.

network→get_userData() YNetwork
network→userData()

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

```
js      function get_userData( )
nodejs  function get_userData( )
php     function get_userData( )
cpp     void * get_userData( )
m       -(void*) userData
pas     function get_userData( ): Tobject
vb      function get_userData( ) As Object
cs      object get_userData( )
java    Object get_userData( )
py      def get_userData( )
```

This attribute is never touched directly by the API, and is at disposal of the caller to store a context.

Returns :
the object stored previously by the caller.

network→get_userPassword() YNetwork
network→userPassword()

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

```
js      function get_userPassword( )
nodejs  function get_userPassword( )
php     function get_userPassword( )
cpp     string get_userPassword( )
m       -(NSString*) userPassword
pas     function get_userPassword( ): string
vb      function get_userPassword( ) As String
cs      string get_userPassword( )
java    String get_userPassword( )
py      def get_userPassword( )
cmd     YNetwork target get_userPassword
```

Returns :

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

On failure, throws an exception or returns `Y_USERPASSWORD_INVALID`.

network→get_wwwWatchdogDelay() YNetwork
network→wwwWatchdogDelay()

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

js	function **get_wwwWatchdogDelay**( )
nodejs	function **get_wwwWatchdogDelay**( )
php	function **get_wwwWatchdogDelay**( )
cpp	int **get_wwwWatchdogDelay**( )
m	-(int) wwwWatchdogDelay
pas	function **get_wwwWatchdogDelay**( ): LongInt
vb	function **get_wwwWatchdogDelay**( ) As Integer
cs	int **get_wwwWatchdogDelay**( )
java	int **get_wwwWatchdogDelay**( )
py	def **get_wwwWatchdogDelay**( )
cmd	YNetwork **target** **get_wwwWatchdogDelay**

A zero value disables automated reboot in case of Internet connectivity loss.

Returns :

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

On failure, throws an exception or returns `Y_WWWWATCHDOGDELAY_INVALID`.

network→isOnline() YNetwork

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

```
js      function isOnline( )
nodejs  function isOnline( )
php     function isOnline( )
cpp     bool isOnline( )
m       -(BOOL) isOnline
pas     function isOnline( ): boolean
vb      function isOnline( ) As Boolean
cs      bool isOnline( )
java    boolean isOnline( )
py      def isOnline( )
```

If there is a cached value for the network interface in cache, that has not yet expired, the device is considered reachable. No exception is raised if there is an error while trying to contact the device hosting the network interface.

Returns :

`true` if the network interface can be reached, and `false` otherwise

network→isOnline_async() YNetwork

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

```
js      function isOnline_async( callback, context)
nodejs  function isOnline_async( callback, context)
```

If there is a cached value for the network interface in cache, that has not yet expired, the device is considered reachable. No exception is raised if there is an error while trying to contact the device hosting the requested function.

This asynchronous version exists only in Javascript. It uses a callback instead of a return value in order to avoid blocking the Javascript virtual machine.

Parameters :

callback callback function that is invoked when the result is known. The callback function receives three arguments: the caller-specific context object, the receiving function object and the boolean result

context caller-specific object that is passed as-is to the callback function

Returns :

nothing : the result is provided to the callback.

network→load() YNetwork

Preloads the network interface cache with a specified validity duration.

```
js      function load( msValidity)
nodejs  function load( msValidity)
php     function load( $msValidity)
cpp     YRETCODE load( int msValidity)
m       -(YRETCODE) load : (int) msValidity
pas     function load( msValidity: integer): YRETCODE
vb      function load( ByVal msValidity As Integer) As YRETCODE
cs      YRETCODE load( int msValidity)
java    int load( long msValidity)
py      def load( msValidity)
```

By default, whenever accessing a device, all function attributes are kept in cache for the standard duration (5 ms). This method can be used to temporarily mark the cache as valid for a longer period, in order to reduce network trafic for instance.

Parameters :

msValidity an integer corresponding to the validity attributed to the loaded function parameters, in milliseconds

Returns :

YAPI_SUCCESS when the call succeeds.

On failure, throws an exception or returns a negative error code.

network→load_async() YNetwork

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

```
js      function load_async( msValidity, callback, context)
nodejs  function load_async( msValidity, callback, context)
```

By default, whenever accessing a device, all function attributes are kept in cache for the standard duration (5 ms). This method can be used to temporarily mark the cache as valid for a longer period, in order to reduce network trafic for instance.

This asynchronous version exists only in Javascript. It uses a callback instead of a return value in order to avoid blocking the Javascript virtual machine.

Parameters :

msValidity	an integer corresponding to the validity of the loaded function parameters, in milliseconds
callback	callback function that is invoked when the result is known. The callback function receives three arguments: the caller-specific context object, the receiving function object and the error code (or YAPI_SUCCESS)
context	caller-specific object that is passed as-is to the callback function

Returns :

nothing : the result is provided to the callback.

network→nextNetwork() YNetwork

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

```
js      function nextNetwork( )
nodejs  function nextNetwork( )
php     function nextNetwork( )
cpp     YNetwork * nextNetwork( )
m       -(YNetwork*) nextNetwork
pas     function nextNetwork( ): TYNetwork
vb      function nextNetwork( ) As YNetwork
cs      YNetwork nextNetwork( )
java    YNetwork nextNetwork( )
py      def nextNetwork( )
```

Returns :

a pointer to a `YNetwork` object, corresponding to a network interface currently online, or a `null` pointer if there are no more network interfaces to enumerate.

network→ping() YNetwork

Pings str_host to test the network connectivity.

```
js      function ping( host)
nodejs  function ping( host)
php     function ping( $host)
cpp     string ping( string host)
m       -(NSString*) ping : (NSString*) host
pas     function ping( host: string): string
vb      function ping( ) As String
cs      string ping( string host)
java    String ping( String host)
py      def ping( host)
cmd     YNetwork target ping host
```

Sends four ICMP ECHO_REQUEST requests from the module to the target str_host. This method returns a string with the result of the 4 ICMP ECHO_REQUEST requests.

Parameters :

host the hostname or the IP address of the target

Returns :

a string with the result of the ping.

network→registerValueCallback() YNetwork

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

```
js      function registerValueCallback( callback)
nodejs  function registerValueCallback( callback)
php     function registerValueCallback( $callback)
cpp     int registerValueCallback( YNetworkValueCallback callback)
m       -(int) registerValueCallback : (YNetworkValueCallback) callback
pas     function registerValueCallback( callback: TYNetworkValueCallback): LongInt
vb      function registerValueCallback( ) As Integer
cs      int registerValueCallback( ValueCallback callback)
java    int registerValueCallback( UpdateCallback callback)
py      def registerValueCallback( callback)
```

The callback is invoked only during the execution of `ySleep` or `yHandleEvents`. This provides control over the time when the callback is triggered. For good responsiveness, remember to call one of these two functions periodically. To unregister a callback, pass a null pointer as argument.

Parameters :

callback the callback function to call, or a null pointer. The callback function should take two arguments: the function object of which the value has changed, and the character string describing the new advertised value.

network→set_adminPassword() YNetwork
network→setAdminPassword()

Changes the password for the "admin" user.

```
js      function set_adminPassword( newval)
nodejs  function set_adminPassword( newval)
php     function set_adminPassword( $newval)
cpp     int set_adminPassword( const string& newval)
m       -(int) setAdminPassword : (NSString*) newval
pas     function set_adminPassword( newval: string): integer
vb      function set_adminPassword( ByVal newval As String) As Integer
cs      int set_adminPassword( string newval)
java    int set_adminPassword( String newval)
py      def set_adminPassword( newval)
cmd     YNetwork target set_adminPassword newval
```

This password becomes instantly required to perform any change of the module state. If the specified value is an empty string, a password is not required anymore. Remember to call the `saveToFlash()` method of the module if the modification must be kept.

Parameters :

newval a string corresponding to the password for the "admin" user

Returns :

`YAPI_SUCCESS` if the call succeeds.

On failure, throws an exception or returns a negative error code.

network→set_callbackCredentials() YNetwork
network→setCallbackCredentials()

Changes the credentials required to connect to the callback address.

```
js      function set_callbackCredentials( newval)
nodejs  function set_callbackCredentials( newval)
php     function set_callbackCredentials( $newval)
cpp     int set_callbackCredentials( const string& newval)
m       -(int) setCallbackCredentials : (NSString*) newval
pas     function set_callbackCredentials( newval: string): integer
vb      function set_callbackCredentials( ByVal newval As String) As Integer
cs      int set_callbackCredentials( string newval)
java    int set_callbackCredentials( String newval)
py      def set_callbackCredentials( newval)
cmd     YNetwork target set_callbackCredentials newval
```

The credentials must be provided as returned by function `get_callbackCredentials`, in the form `username:hash`. The method used to compute the hash varies according to the the authentication scheme implemented by the callback, For Basic authentication, the hash is the MD5 of the string `username:password`. For Digest authentication, the hash is the MD5 of the string `username:realm:password`. For a simpler way to configure callback credentials, use function `callbackLogin` instead. Remember to call the `saveToFlash()` method of the module if the modification must be kept.

Parameters :

newval a string corresponding to the credentials required to connect to the callback address

Returns :

`YAPI_SUCCESS` if the call succeeds.

On failure, throws an exception or returns a negative error code.

network→set_callbackEncoding() YNetwork
network→setCallbackEncoding()

Changes the encoding standard to use for representing notification values.

js	function **set_callbackEncoding**(**newval**)
nodejs	function **set_callbackEncoding**(**newval**)
php	function **set_callbackEncoding**(**$newval**)
cpp	int **set_callbackEncoding**(Y_CALLBACKENCODING_enum **newval**)
m	-(int) setCallbackEncoding : (Y_CALLBACKENCODING_enum) **newval**
pas	function **set_callbackEncoding**(**newval**: Integer): integer
vb	function **set_callbackEncoding**(ByVal **newval** As Integer) As Integer
cs	int **set_callbackEncoding**(int **newval**)
java	int **set_callbackEncoding**(int **newval**)
py	def **set_callbackEncoding**(**newval**)
cmd	YNetwork **target set_callbackEncoding newval**

Parameters :

newval a value among `Y_CALLBACKENCODING_FORM`, `Y_CALLBACKENCODING_JSON`, `Y_CALLBACKENCODING_JSON_ARRAY`, `Y_CALLBACKENCODING_CSV` and `Y_CALLBACKENCODING_YOCTO_API` corresponding to the encoding standard to use for representing notification values

Returns :

`YAPI_SUCCESS` if the call succeeds.

On failure, throws an exception or returns a negative error code.

network→set_callbackMaxDelay() YNetwork
network→setCallbackMaxDelay()

Changes the maximum waiting time between two callback notifications, in seconds.

```
js      function set_callbackMaxDelay( newval)
nodejs  function set_callbackMaxDelay( newval)
php     function set_callbackMaxDelay( $newval)
cpp     int set_callbackMaxDelay( int newval)
m       -(int) setCallbackMaxDelay : (int) newval
pas     function set_callbackMaxDelay( newval: LongInt): integer
vb      function set_callbackMaxDelay( ByVal newval As Integer) As Integer
cs      int set_callbackMaxDelay( int newval)
java    int set_callbackMaxDelay( int newval)
py      def set_callbackMaxDelay( newval)
cmd     YNetwork target set_callbackMaxDelay newval
```

Parameters :

newval an integer corresponding to the maximum waiting time between two callback notifications, in seconds

Returns :

YAPI_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

network→set_callbackMethod() YNetwork
network→setCallbackMethod()

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

```
js      function set_callbackMethod( newval)
nodejs  function set_callbackMethod( newval)
php     function set_callbackMethod( $newval)
cpp     int set_callbackMethod( Y_CALLBACKMETHOD_enum newval)
m       -(int) setCallbackMethod : (Y_CALLBACKMETHOD_enum) newval
pas     function set_callbackMethod( newval: Integer): integer
vb      function set_callbackMethod( ByVal newval As Integer) As Integer
cs      int set_callbackMethod( int newval)
java    int set_callbackMethod( int newval)
py      def set_callbackMethod( newval)
cmd     YNetwork target set_callbackMethod newval
```

Parameters :

newval a value among `Y_CALLBACKMETHOD_POST`, `Y_CALLBACKMETHOD_GET` and `Y_CALLBACKMETHOD_PUT` corresponding to the HTTP method used to notify callbacks for significant state changes

Returns :

`YAPI_SUCCESS` if the call succeeds.

On failure, throws an exception or returns a negative error code.

network→set_callbackMinDelay() YNetwork
network→setCallbackMinDelay()

Changes the minimum waiting time between two callback notifications, in seconds.

```
js      function set_callbackMinDelay( newval)
nodejs  function set_callbackMinDelay( newval)
php     function set_callbackMinDelay( $newval)
cpp     int set_callbackMinDelay( int newval)
m       -(int) setCallbackMinDelay : (int) newval
pas     function set_callbackMinDelay( newval: LongInt): integer
vb      function set_callbackMinDelay( ByVal newval As Integer) As Integer
cs      int set_callbackMinDelay( int newval)
java    int set_callbackMinDelay( int newval)
py      def set_callbackMinDelay( newval)
cmd     YNetwork target set_callbackMinDelay newval
```

Parameters :

newval an integer corresponding to the minimum waiting time between two callback notifications, in seconds

Returns :

YAPI_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

network→set_callbackUrl() YNetwork
network→setCallbackUrl()

Changes the callback URL to notify significant state changes.

```
js      function set_callbackUrl( newval)
nodejs  function set_callbackUrl( newval)
php     function set_callbackUrl( $newval)
cpp     int set_callbackUrl( const string& newval)
m       -(int) setCallbackUrl : (NSString*) newval
pas     function set_callbackUrl( newval: string): integer
vb      function set_callbackUrl( ByVal newval As String) As Integer
cs      int set_callbackUrl( string newval)
java    int set_callbackUrl( String newval)
py      def set_callbackUrl( newval)
cmd     YNetwork target set_callbackUrl newval
```

Remember to call the `saveToFlash()` method of the module if the modification must be kept.

Parameters :

newval a string corresponding to the callback URL to notify significant state changes

Returns :

`YAPI_SUCCESS` if the call succeeds.

On failure, throws an exception or returns a negative error code.

network→set_discoverable() YNetwork
network→setDiscoverable()

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

```
js      function set_discoverable( newval)
nodejs  function set_discoverable( newval)
php     function set_discoverable( $newval)
cpp     int set_discoverable( Y_DISCOVERABLE_enum newval)
m       -(int) setDiscoverable : (Y_DISCOVERABLE_enum) newval
pas     function set_discoverable( newval: Integer): integer
vb      function set_discoverable( ByVal newval As Integer) As Integer
cs      int set_discoverable( int newval)
java    int set_discoverable( int newval)
py      def set_discoverable( newval)
cmd     YNetwork target set_discoverable newval
```

Parameters :

newval either `Y_DISCOVERABLE_FALSE` or `Y_DISCOVERABLE_TRUE`, according to the activation state of the multicast announce protocols to allow easy discovery of the module in the network neighborhood (uPnP/Bonjour protocol)

Returns :

`YAPI_SUCCESS` if the call succeeds.

On failure, throws an exception or returns a negative error code.

network→set_logicalName() YNetwork
network→setLogicalName()

Changes the logical name of the network interface.

```
js      function set_logicalName( newval)
nodejs  function set_logicalName( newval)
php     function set_logicalName( $newval)
cpp     int set_logicalName( const string& newval)
m       -(int) setLogicalName : (NSString*) newval
pas     function set_logicalName( newval: string): integer
vb      function set_logicalName( ByVal newval As String) As Integer
cs      int set_logicalName( string newval)
java    int set_logicalName( String newval)
py      def set_logicalName( newval)
cmd     YNetwork target set_logicalName newval
```

You can use `yCheckLogicalName()` prior to this call to make sure that your parameter is valid. Remember to call the `saveToFlash()` method of the module if the modification must be kept.

Parameters :

newval a string corresponding to the logical name of the network interface.

Returns :

`YAPI_SUCCESS` if the call succeeds.

On failure, throws an exception or returns a negative error code.

network→set_primaryDNS() YNetwork
network→setPrimaryDNS()

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

```
js      function set_primaryDNS( newval)
nodejs  function set_primaryDNS( newval)
php     function set_primaryDNS( $newval)
cpp     int set_primaryDNS( const string& newval)
m       -(int) setPrimaryDNS : (NSString*) newval
pas     function set_primaryDNS( newval: string): integer
vb      function set_primaryDNS( ByVal newval As String) As Integer
cs      int set_primaryDNS( string newval)
java    int set_primaryDNS( String newval)
py      def set_primaryDNS( newval)
cmd     YNetwork target set_primaryDNS newval
```

When using DHCP, if a value is specified, it overrides the value received from the DHCP server. Remember to call the `saveToFlash()` method and then to reboot the module to apply this setting.

Parameters :

newval a string corresponding to the IP address of the primary name server to be used by the module

Returns :

`YAPI_SUCCESS` if the call succeeds.

On failure, throws an exception or returns a negative error code.

network→set_secondaryDNS() YNetwork
network→setSecondaryDNS()

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

```
js      function set_secondaryDNS( newval)
nodejs  function set_secondaryDNS( newval)
php     function set_secondaryDNS( $newval)
cpp     int set_secondaryDNS( const string& newval)
m       -(int) setSecondaryDNS : (NSString*) newval
pas     function set_secondaryDNS( newval: string): integer
vb      function set_secondaryDNS( ByVal newval As String) As Integer
cs      int set_secondaryDNS( string newval)
java    int set_secondaryDNS( String newval)
py      def set_secondaryDNS( newval)
cmd     YNetwork target set_secondaryDNS newval
```

When using DHCP, if a value is specified, it overrides the value received from the DHCP server. Remember to call the `saveToFlash()` method and then to reboot the module to apply this setting.

Parameters :

newval a string corresponding to the IP address of the secondary name server to be used by the module

Returns :

`YAPI_SUCCESS` if the call succeeds.

On failure, throws an exception or returns a negative error code.

network→set_userData() YNetwork
network→setUserData()

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

```
js      function set_userData( data)
nodejs  function set_userData( data)
php     function set_userData( $data)
cpp     void set_userData( void* data)
m       -(void) setUserData : (void*) data
pas     procedure set_userData( data: Tobject)
vb      procedure set_userData( ByVal data As Object)
cs      void set_userData( object data)
java    void set_userData( Object data)
py      def set_userData( data)
```

This attribute is never touched by the API, and is at disposal of the caller to store a context.

Parameters :

data any kind of object to be stored

network→set_userPassword() YNetwork
network→setUserPassword()

Changes the password for the "user" user.

```
js      function set_userPassword( newval)
nodejs  function set_userPassword( newval)
php     function set_userPassword( $newval)
cpp     int set_userPassword( const string& newval)
m       -(int) setUserPassword : (NSString*) newval
pas     function set_userPassword( newval: string): integer
vb      function set_userPassword( ByVal newval As String) As Integer
cs      int set_userPassword( string newval)
java    int set_userPassword( String newval)
py      def set_userPassword( newval)
cmd     YNetwork target set_userPassword newval
```

This password becomes instantly required to perform any use of the module. If the specified value is an empty string, a password is not required anymore. Remember to call the `saveToFlash()` method of the module if the modification must be kept.

Parameters :

newval a string corresponding to the password for the "user" user

Returns :

`YAPI_SUCCESS` if the call succeeds.

On failure, throws an exception or returns a negative error code.

network→set_wwwWatchdogDelay() YNetwork
network→setWwwWatchdogDelay()

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

```
js      function set_wwwWatchdogDelay( newval)
nodejs  function set_wwwWatchdogDelay( newval)
php     function set_wwwWatchdogDelay( $newval)
cpp     int set_wwwWatchdogDelay( int newval)
m       -(int) setWwwWatchdogDelay : (int) newval
pas     function set_wwwWatchdogDelay( newval: LongInt): integer
vb      function set_wwwWatchdogDelay( ByVal newval As Integer) As Integer
cs      int set_wwwWatchdogDelay( int newval)
java    int set_wwwWatchdogDelay( int newval)
py      def set_wwwWatchdogDelay( newval)
cmd     YNetwork target set_wwwWatchdogDelay newval
```

A zero value disables automated reboot in case of Internet connectivity loss. The smallest valid non-zero timeout is 90 seconds.

Parameters :

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

Returns :

YAPI_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

network→useDHCP() YNetwork

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

```
js      function useDHCP( fallbackIpAddr, fallbackSubnetMaskLen, fallbackRouter)
nodejs  function useDHCP( fallbackIpAddr, fallbackSubnetMaskLen, fallbackRouter)
php     function useDHCP( $fallbackIpAddr, $fallbackSubnetMaskLen, $fallbackRouter)
cpp     int useDHCP( string fallbackIpAddr,
                     int fallbackSubnetMaskLen,
                     string fallbackRouter)
m       -(int) useDHCP : (NSString*) fallbackIpAddr
                       : (int) fallbackSubnetMaskLen
                       : (NSString*) fallbackRouter
pas     function useDHCP( fallbackIpAddr: string,
                          fallbackSubnetMaskLen: LongInt,
                          fallbackRouter: string): integer
vb      function useDHCP( ByVal fallbackIpAddr As String,
                          ByVal fallbackSubnetMaskLen As Integer,
                          ByVal fallbackRouter As String) As Integer
cs      int useDHCP( string fallbackIpAddr,
                     int fallbackSubnetMaskLen,
                     string fallbackRouter)
java    int useDHCP( String fallbackIpAddr,
                     int fallbackSubnetMaskLen,
                     String fallbackRouter)
py      def useDHCP( fallbackIpAddr, fallbackSubnetMaskLen, fallbackRouter)
cmd     YNetwork target useDHCP fallbackIpAddr fallbackSubnetMaskLen fallbackRouter
```

Until an address is received from a DHCP server, the module uses the IP parameters specified to this function. Remember to call the `saveToFlash()` method and then to reboot the module to apply this setting.

Parameters :

fallbackIpAddr	fallback IP address, to be used when no DHCP reply is received
fallbackSubnetMaskLen	fallback subnet mask length when no DHCP reply is received, as an integer (eg. 24 means 255.255.255.0)
fallbackRouter	fallback router IP address, to be used when no DHCP reply is received

Returns :

`YAPI_SUCCESS` if the call succeeds.

On failure, throws an exception or returns a negative error code.

network→useStaticIP() YNetwork

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

```
js      function useStaticIP( ipAddress, subnetMaskLen, router)
nodejs  function useStaticIP( ipAddress, subnetMaskLen, router)
php     function useStaticIP( $ipAddress, $subnetMaskLen, $router)
cpp     int useStaticIP( string ipAddress,
                         int subnetMaskLen,
                         string router)
m       -(int) useStaticIP : (NSString*) ipAddress
                           : (int) subnetMaskLen
                           : (NSString*) router
pas     function useStaticIP( ipAddress: string,
                              subnetMaskLen: LongInt,
                              router: string): integer
vb      function useStaticIP( ByVal ipAddress As String,
                              ByVal subnetMaskLen As Integer,
                              ByVal router As String) As Integer
cs      int useStaticIP( string ipAddress,
                         int subnetMaskLen,
                         string router)
java    int useStaticIP( String ipAddress,
                         int subnetMaskLen,
                         String router)
py      def useStaticIP( ipAddress, subnetMaskLen, router)
cmd     YNetwork target useStaticIP ipAddress subnetMaskLen router
```

Remember to call the `saveToFlash()` method and then to reboot the module to apply this setting.

Parameters :

ipAddress	device IP address
subnetMaskLen	subnet mask length, as an integer (eg. 24 means 255.255.255.0)
router	router IP address (default gateway)

Returns :

`YAPI_SUCCESS` if the call succeeds.

On failure, throws an exception or returns a negative error code.

network→wait_async() YNetwork

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

```
js      function wait_async( callback, context)
nodejs  function wait_async( callback, context)
```

The callback function can therefore freely issue synchronous or asynchronous commands, without risking to block the Javascript VM.

Parameters :

callback callback function that is invoked when all pending commands on the module are completed. The callback function receives two arguments: the caller-specific context object and the receiving function object.

context caller-specific object that is passed as-is to the callback function

Returns :

nothing.

11.4. Files function interface

The filesystem interface makes it possible to store files on some devices, 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>
nodejs  var yoctolib = require('yoctolib');
        var YFiles = yoctolib.YFiles;
php     require_once('yocto_files.php');
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;
py      from yocto_files import *
```

Global functions

yFindFiles(**func**)

Retrieves a filesystem for a given identifier.

yFirstFiles()

Starts the enumeration of filesystems currently accessible.

YFiles methods

files→describe()

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

files→download(**pathname**)

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

files→download_async(**pathname**, **callback**, **context**)

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

files→format_fs()

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

files→get_advertisedValue()

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

files→get_errorMessage()

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

files→get_errorType()

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

files→get_filesCount()

Returns the number of files currently loaded in the filesystem.

files→get_freeSpace()

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

files→get_friendlyName()

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

files→get_functionDescriptor()

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

files→get_functionId()

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

files→get_hardwareId()

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

files→get_list(**pattern**)

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

files→get_logicalName()

Returns the logical name of the filesystem.

files→get_module()

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

files→get_module_async(**callback**, **context**)

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

files→get_userData()

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

files→isOnline()

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

files→isOnline_async(**callback**, **context**)

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

files→load(**msValidity**)

Preloads the filesystem cache with a specified validity duration.

files→load_async(**msValidity**, **callback**, **context**)

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

files→nextFiles()

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

files→registerValueCallback(**callback**)

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

files→remove(**pathname**)

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

files→set_logicalName(**newval**)

Changes the logical name of the filesystem.

files→set_userData(**data**)

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

files→upload(**pathname**, **content**)

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

files→wait_async(**callback**, **context**)

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

YFiles.FindFiles() YFiles
yFindFiles()

Retrieves a filesystem for a given identifier.

```
js      function yFindFiles( func)
nodejs  function FindFiles( func)
php     function yFindFiles( $func)
cpp     YFiles* yFindFiles( string func)
m       +(YFiles*) FindFiles : (NSString*) func
pas     function yFindFiles( func: string): TYFiles
vb      function yFindFiles( ByVal func As String) As YFiles
cs      YFiles FindFiles( string func)
java    YFiles FindFiles( String func)
py      def FindFiles( func)
```

The identifier can be specified using several formats:

- FunctionLogicalName
- ModuleSerialNumber.FunctionIdentifier
- ModuleSerialNumber.FunctionLogicalName
- ModuleLogicalName.FunctionIdentifier
- ModuleLogicalName.FunctionLogicalName

This function does not require that the filesystem is online at the time it is invoked. The returned object is nevertheless valid. Use the method `YFiles.isOnline()` to test if the filesystem is indeed online at a given time. In case of ambiguity when looking for a filesystem by logical name, no error is notified: the first instance found is returned. The search is performed first by hardware name, then by logical name.

Parameters :

func a string that uniquely characterizes the filesystem

Returns :

a `YFiles` object allowing you to drive the filesystem.

YFiles.FirstFiles() YFiles
yFirstFiles()

Starts the enumeration of filesystems currently accessible.

```
js      function yFirstFiles( )
nodejs  function FirstFiles( )
php     function yFirstFiles( )
cpp     YFiles* yFirstFiles( )
m       +(YFiles*) FirstFiles
pas     function yFirstFiles( ): TYFiles
vb      function yFirstFiles( ) As YFiles
cs      YFiles FirstFiles( )
java    YFiles FirstFiles( )
py      def FirstFiles( )
```

Use the method `YFiles.nextFiles()` to iterate on next filesystems.

Returns :

a pointer to a `YFiles` object, corresponding to the first filesystem currently online, or a `null` pointer if there are none.

files→describe() YFiles

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

```
js      function describe( )
nodejs  function describe( )
php     function describe( )
cpp     string describe( )
m       -(NSString*) describe
pas     function describe( ): string
vb      function describe( ) As String
cs      string describe( )
java    String describe( )
py      def describe( )
```

More precisely, TYPE is the type of the function, NAME it the name used for the first access to the function, SERIAL is the serial number of the module if the module is connected or "unresolved", and FUNCTIONID is the hardware identifier of the function if the module is connected. For example, this method returns Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1 if the module is already connected or Relay(BadCustomeName.relay1)=unresolved if the module has not yet been connected. This method does not trigger any USB or TCP transaction and can therefore be used in a debugger.

Returns :

a string that describes the filesystem (ex: Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1)

files→download() YFiles

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

```
js      function download( pathname)
nodejs  function download( pathname)
php     function download( $pathname)
cpp     string download( string pathname)
m       -(NSData*) download : (NSString*) pathname
pas     function download( pathname: string): TByteArray
vb      function download( ) As Byte
py      def download( pathname)
cmd     YFiles target download pathname
```

Parameters :

pathname path and name of the file to download

Returns :

a binary buffer with the file content

On failure, throws an exception or returns an empty content.

files→download_async() YFiles

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

```
js      function download_async( pathname, callback, context)
nodejs  function download_async( pathname, callback, context)
```

This is the asynchronous version that uses a callback to pass the result when the donwload is completed.

Parameters :

pathname	path and name of the new file to load
callback	callback function that is invoked when the w The callback function receives three arguments: - the user-specific context object - the YFiles object whose download_async was invoked - a binary buffer with the file content
context	user-specific object that is passed as-is to the callback function

Returns :

nothing.

files→format_fs() YFiles

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

```
js      function format_fs( )
nodejs  function format_fs( )
php     function format_fs( )
cpp     int format_fs( )
m       -(int) format_fs
pas     function format_fs( ): LongInt
vb      function format_fs( ) As Integer
cs      int format_fs( )
java    int format_fs( )
py      def format_fs( )
cmd     YFiles target format_fs
```

All files previously uploaded are permanently lost.

Returns :

YAPI_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

files→get_advertisedValue() YFiles
files→advertisedValue()

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

```
js      function get_advertisedValue( )
nodejs  function get_advertisedValue( )
php     function get_advertisedValue( )
cpp     string get_advertisedValue( )
m       -(NSString*) advertisedValue
pas     function get_advertisedValue( ): string
vb      function get_advertisedValue( ) As String
cs      string get_advertisedValue( )
java    String get_advertisedValue( )
py      def get_advertisedValue( )
cmd     YFiles target get_advertisedValue
```

Returns :
a string corresponding to the current value of the filesystem (no more than 6 characters).

On failure, throws an exception or returns `Y_ADVERTISEDVALUE_INVALID`.

files→get_errorMessage() YFiles
files→errorMessage()

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

js	function **get_errorMessage**( )
nodejs	function **get_errorMessage**( )
php	function **get_errorMessage**( )
cpp	string **get_errorMessage**( )
m	-(NSString*) errorMessage
pas	function **get_errorMessage**( ): string
vb	function **get_errorMessage**( ) As String
cs	string **get_errorMessage**( )
java	String **get_errorMessage**( )
py	def **get_errorMessage**( )

This method is mostly useful when using the Yoctopuce library with exceptions disabled.

Returns :
a string corresponding to the latest error message that occured while using the filesystem object

files→get_errorType() YFiles
files→errorType()

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

```
js      function get_errorType( )
nodejs  function get_errorType( )
php     function get_errorType( )
cpp     YRETCODE get_errorType( )
pas     function get_errorType( ): YRETCODE
vb      function get_errorType( ) As YRETCODE
cs      YRETCODE get_errorType( )
java    int get_errorType( )
py      def get_errorType( )
```

This method is mostly useful when using the Yoctopuce library with exceptions disabled.

Returns :
a number corresponding to the code of the latest error that occured while using the filesystem object

files→get_filesCount() YFiles
files→filesCount()

Returns the number of files currently loaded in the filesystem.

```
js      function get_filesCount( )
nodejs  function get_filesCount( )
php     function get_filesCount( )
cpp     int get_filesCount( )
m       -(int) filesCount
pas     function get_filesCount( ): LongInt
vb      function get_filesCount( ) As Integer
cs      int get_filesCount( )
java    int get_filesCount( )
py      def get_filesCount( )
cmd     YFiles target get_filesCount
```

Returns :
an integer corresponding to the number of files currently loaded in the filesystem

On failure, throws an exception or returns `Y_FILESCOUNT_INVALID`.

files→get_freeSpace() YFiles
files→freeSpace()

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

```
js      function get_freeSpace( )
nodejs  function get_freeSpace( )
php     function get_freeSpace( )
cpp     int get_freeSpace( )
m       -(int) freeSpace
pas     function get_freeSpace( ): LongInt
vb      function get_freeSpace( ) As Integer
cs      int get_freeSpace( )
java    int get_freeSpace( )
py      def get_freeSpace( )
cmd     YFiles target get_freeSpace
```

Returns :

an integer corresponding to the free space for uploading new files to the filesystem, in bytes

On failure, throws an exception or returns Y_FREESPACE_INVALID.

files→get_friendlyName() YFiles
files→friendlyName()

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

```
js      function get_friendlyName( )
nodejs  function get_friendlyName( )
php     function get_friendlyName( )
cpp     string get_friendlyName( )
m       -(NSString*) friendlyName
cs      string get_friendlyName( )
java    String get_friendlyName( )
py      def get_friendlyName( )
```

The returned string uses the logical names of the module and of the filesystem if they are defined, otherwise the serial number of the module and the hardware identifier of the filesystem (for exemple: MyCustomName.relay1)

Returns :

a string that uniquely identifies the filesystem using logical names (ex: MyCustomName.relay1)

On failure, throws an exception or returns Y_FRIENDLYNAME_INVALID.

files→get_functionDescriptor() YFiles
files→functionDescriptor()

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

```
js      function get_functionDescriptor( )
nodejs  function get_functionDescriptor( )
php     function get_functionDescriptor( )
cpp     YFUN_DESCR get_functionDescriptor( )
m       -(YFUN_DESCR) functionDescriptor
pas     function get_functionDescriptor( ): YFUN_DESCR
vb      function get_functionDescriptor( ) As YFUN_DESCR
cs      YFUN_DESCR get_functionDescriptor( )
java    String get_functionDescriptor( )
py      def get_functionDescriptor( )
```

This identifier can be used to test if two instances of YFunction reference the same physical function on the same physical device.

Returns :

an identifier of type YFUN_DESCR.

If the function has never been contacted, the returned value is Y_FUNCTIONDESCRIPTOR_INVALID.

files→get_functionId() YFiles
files→functionId()

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

```
js      function get_functionId( )
nodejs  function get_functionId( )
php     function get_functionId( )
cpp     string get_functionId( )
m       -(NSString*) functionId
vb      function get_functionId( ) As String
cs      string get_functionId( )
java    String get_functionId( )
py      def get_functionId( )
```

For example `relay1`

Returns :

a string that identifies the filesystem (ex: `relay1`)

On failure, throws an exception or returns `Y_FUNCTIONID_INVALID`.

files→get_hardwareId() YFiles
files→hardwareId()

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

```
js      function get_hardwareId( )
nodejs  function get_hardwareId( )
php     function get_hardwareId( )
cpp     string get_hardwareId( )
m       -(NSString*) hardwareId
vb      function get_hardwareId( ) As String
cs      string get_hardwareId( )
java    String get_hardwareId( )
py      def get_hardwareId( )
```

The unique hardware identifier is composed of the device serial number and of the hardware identifier of the filesystem (for example RELAYLO1-123456.relay1).

Returns :

a string that uniquely identifies the filesystem (ex: RELAYLO1-123456.relay1)

On failure, throws an exception or returns Y_HARDWAREID_INVALID.

files→get_list() YFiles
files→list()

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

```
js      function get_list( pattern)
nodejs  function get_list( pattern)
php     function get_list( $pattern)
cpp     vector<YFileRecord> get_list( string pattern)
m       -(NSMutableArray*) list : (NSString*) pattern
pas     function get_list( pattern: string): TYFileRecordArray
vb      function get_list( ) As List
cs      List<YFileRecord> get_list( string pattern)
java    ArrayList<YFileRecord> get_list( String pattern)
py      def get_list( pattern)
cmd     YFiles target get_list pattern
```

Parameters :

pattern an optional filter pattern, using star and question marks as wildcards. When an empty pattern is provided, all file records are returned.

Returns :

a list of `YFileRecord` objects, containing the file path and name, byte size and 32-bit CRC of the file content.

On failure, throws an exception or returns an empty list.

files→get_logicalName() YFiles
files→logicalName()

Returns the logical name of the filesystem.

```
js      function get_logicalName( )
nodejs  function get_logicalName( )
php     function get_logicalName( )
cpp     string get_logicalName( )
m       -(NSString*) logicalName
pas     function get_logicalName( ): string
vb      function get_logicalName( ) As String
cs      string get_logicalName( )
java    String get_logicalName( )
py      def get_logicalName( )
cmd     YFiles target get_logicalName
```

Returns :
a string corresponding to the logical name of the filesystem.

On failure, throws an exception or returns `Y_LOGICALNAME_INVALID`.

files→get_module() YFiles
files→module()

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

```
js      function get_module( )
nodejs  function get_module( )
php     function get_module( )
cpp     YModule * get_module( )
m       -(YModule*) module
pas     function get_module( ): TYModule
vb      function get_module( ) As YModule
cs      YModule get_module( )
java    YModule get_module( )
py      def get_module( )
```

If the function cannot be located on any module, the returned instance of YModule is not shown as on-line.

Returns :
an instance of YModule

files→get_module_async() YFiles
files→module_async()

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

```
  js  function get_module_async( callback, context)
nodejs function get_module_async( callback, context)
```

If the function cannot be located on any module, the returned YModule object does not show as on-line.

This asynchronous version exists only in Javascript. It uses a callback instead of a return value in order to avoid blocking Firefox javascript VM that does not implement context switching during blocking I/O calls. See the documentation section on asynchronous Javascript calls for more details.

Parameters :

callback callback function that is invoked when the result is known. The callback function receives three arguments: the caller-specific context object, the receiving function object and the requested YModule object

context caller-specific object that is passed as-is to the callback function

Returns :

nothing : the result is provided to the callback.

files→get_userData() YFiles
files→userData()

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

js	function **get_userData**( )
nodejs	function **get_userData**( )
php	function **get_userData**( )
cpp	void * **get_userData**( )
m	-(void*) userData
pas	function **get_userData**( ): Tobject
vb	function **get_userData**( ) As Object
cs	object **get_userData**( )
java	Object **get_userData**( )
py	def **get_userData**( )

This attribute is never touched directly by the API, and is at disposal of the caller to store a context.

Returns :
the object stored previously by the caller.

files→isOnline() YFiles

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

```
js      function isOnline( )
nodejs  function isOnline( )
php     function isOnline( )
cpp     bool isOnline( )
m       -(BOOL) isOnline
pas     function isOnline( ): boolean
vb      function isOnline( ) As Boolean
cs      bool isOnline( )
java    boolean isOnline( )
py      def isOnline( )
```

If there is a cached value for the filesystem in cache, that has not yet expired, the device is considered reachable. No exception is raised if there is an error while trying to contact the device hosting the filesystem.

Returns :

`true` if the filesystem can be reached, and `false` otherwise

files→isOnline_async() YFiles

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

```
js      function isOnline_async( callback, context)
nodejs  function isOnline_async( callback, context)
```

If there is a cached value for the filesystem in cache, that has not yet expired, the device is considered reachable. No exception is raised if there is an error while trying to contact the device hosting the requested function.

This asynchronous version exists only in Javascript. It uses a callback instead of a return value in order to avoid blocking the Javascript virtual machine.

Parameters :

callback callback function that is invoked when the result is known. The callback function receives three arguments: the caller-specific context object, the receiving function object and the boolean result

context caller-specific object that is passed as-is to the callback function

Returns :

nothing : the result is provided to the callback.

files→load() YFiles

Preloads the filesystem cache with a specified validity duration.

```
js      function load( msValidity)
nodejs  function load( msValidity)
php     function load( $msValidity)
cpp     YRETCODE load( int msValidity)
m       -(YRETCODE) load : (int) msValidity
pas     function load( msValidity: integer): YRETCODE
vb      function load( ByVal msValidity As Integer) As YRETCODE
cs      YRETCODE load( int msValidity)
java    int load( long msValidity)
py      def load( msValidity)
```

By default, whenever accessing a device, all function attributes are kept in cache for the standard duration (5 ms). This method can be used to temporarily mark the cache as valid for a longer period, in order to reduce network trafic for instance.

Parameters :

msValidity an integer corresponding to the validity attributed to the loaded function parameters, in milliseconds

Returns :

YAPI_SUCCESS when the call succeeds.

On failure, throws an exception or returns a negative error code.

files→load_async() YFiles

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

```
js      function load_async( msValidity, callback, context)
nodejs  function load_async( msValidity, callback, context)
```

By default, whenever accessing a device, all function attributes are kept in cache for the standard duration (5 ms). This method can be used to temporarily mark the cache as valid for a longer period, in order to reduce network trafic for instance.

This asynchronous version exists only in Javascript. It uses a callback instead of a return value in order to avoid blocking the Javascript virtual machine.

Parameters :

msValidity	an integer corresponding to the validity of the loaded function parameters, in milliseconds
callback	callback function that is invoked when the result is known. The callback function receives three arguments: the caller-specific context object, the receiving function object and the error code (or YAPI_SUCCESS)
context	caller-specific object that is passed as-is to the callback function

Returns :

nothing : the result is provided to the callback.

files→nextFiles() YFiles

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

```
js      function nextFiles( )
nodejs  function nextFiles( )
php     function nextFiles( )
cpp     YFiles * nextFiles( )
m       -(YFiles*) nextFiles
pas     function nextFiles( ): TYFiles
vb      function nextFiles( ) As YFiles
cs      YFiles nextFiles( )
java    YFiles nextFiles( )
py      def nextFiles( )
```

Returns :

a pointer to a `YFiles` object, corresponding to a filesystem currently online, or a `null` pointer if there are no more filesystems to enumerate.

files→registerValueCallback() YFiles

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

```
js      function registerValueCallback( callback)
nodejs  function registerValueCallback( callback)
php     function registerValueCallback( $callback)
cpp     int registerValueCallback( YFilesValueCallback callback)
m       -(int) registerValueCallback : (YFilesValueCallback) callback
pas     function registerValueCallback( callback: TYFilesValueCallback): LongInt
vb      function registerValueCallback( ) As Integer
cs      int registerValueCallback( ValueCallback callback)
java    int registerValueCallback( UpdateCallback callback)
py      def registerValueCallback( callback)
```

The callback is invoked only during the execution of `ySleep` or `yHandleEvents`. This provides control over the time when the callback is triggered. For good responsiveness, remember to call one of these two functions periodically. To unregister a callback, pass a null pointer as argument.

Parameters :

callback the callback function to call, or a null pointer. The callback function should take two arguments: the function object of which the value has changed, and the character string describing the new advertised value.

files→remove() YFiles

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

```
js      function remove( pathname)
nodejs  function remove( pathname)
php     function remove( $pathname)
cpp     int remove( string pathname)
m       -(int) remove : (NSString*) pathname
pas     function remove( pathname: string): LongInt
vb      function remove( ) As Integer
cs      int remove( string pathname)
java    int remove( String pathname)
py      def remove( pathname)
cmd     YFiles target remove pathname
```

Because of filesystem fragmentation, deleting a file may not always free up the whole space used by the file. However, rewriting a file with the same path name will always reuse any space not freed previously. If you need to ensure that no space is taken by previously deleted files, you can use `format_fs` to fully reinitialize the filesystem.

Parameters :

pathname path and name of the file to remove.

Returns :

`YAPI_SUCCESS` if the call succeeds.

On failure, throws an exception or returns a negative error code.

files→set_logicalName() YFiles
files→setLogicalName()

Changes the logical name of the filesystem.

```
js      function set_logicalName( newval)
nodejs  function set_logicalName( newval)
php     function set_logicalName( $newval)
cpp     int set_logicalName( const string& newval)
m       -(int) setLogicalName : (NSString*) newval
pas     function set_logicalName( newval: string): integer
vb      function set_logicalName( ByVal newval As String) As Integer
cs      int set_logicalName( string newval)
java    int set_logicalName( String newval)
py      def set_logicalName( newval)
cmd     YFiles target set_logicalName newval
```

You can use `yCheckLogicalName()` prior to this call to make sure that your parameter is valid. Remember to call the `saveToFlash()` method of the module if the modification must be kept.

Parameters :

newval a string corresponding to the logical name of the filesystem.

Returns :

`YAPI_SUCCESS` if the call succeeds.

On failure, throws an exception or returns a negative error code.

files→set_userData() YFiles
files→setUserData()

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

```
js      function set_userData( data)
nodejs  function set_userData( data)
php     function set_userData( $data)
cpp     void set_userData( void* data)
m       -(void) setUserData : (void*) data
pas     procedure set_userData( data: Tobject)
vb      procedure set_userData( ByVal data As Object)
cs      void set_userData( object data)
java    void set_userData( Object data)
py      def set_userData( data)
```

This attribute is never touched by the API, and is at disposal of the caller to store a context.

Parameters :

data any kind of object to be stored

files→upload() YFiles

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

```
js      function upload( pathname, content)
nodejs  function upload( pathname, content)
php     function upload( $pathname, $content)
cpp     int upload( string pathname, string content)
m       -(int) upload : (NSString*) pathname
                      : (NSData*) content
pas     function upload( pathname: string, content: TByteArray): LongInt
vb      procedure upload( )
cs      int upload( string pathname)
java    int upload( String pathname)
py      def upload( pathname, content)
cmd     YFiles target upload pathname content
```

If a file already exists with the same path name, its content is overwritten.

Parameters :

pathname path and name of the new file to create

content binary buffer with the content to set

Returns :

YAPI_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

files→wait_async() YFiles

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

```
js      function wait_async( callback, context)
nodejs  function wait_async( callback, context)
```

The callback function can therefore freely issue synchronous or asynchronous commands, without risking to block the Javascript VM.

Parameters :

callback callback function that is invoked when all pending commands on the module are completed. The callback function receives two arguments: the caller-specific context object and the receiving function object.

context caller-specific object that is passed as-is to the callback function

Returns :

nothing.

11.5. WakeUpMonitor function interface

The WakeUpMonitor function handles globally all wake-up sources, as well as automated sleep mode.

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

```
js      <script type='text/javascript' src='yocto_wakeupmonitor.js'></script>
nodejs  var yoctolib = require('yoctolib');
        var YWakeUpMonitor = yoctolib.YWakeUpMonitor;
php     require_once('yocto_wakeupmonitor.php');
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;
py      from yocto_wakeupmonitor import *
```

Global functions
yFindWakeUpMonitor(**func**) Retrieves a monitor for a given identifier.
yFirstWakeUpMonitor() Starts the enumeration of monitors currently accessible.
`YWakeUpMonitor` methods
wakeupmonitor→describe() Returns a short text that describes unambiguously the instance of the monitor in the form `TYPE(NAME)=SERIAL.FUNCTIONID`.
wakeupmonitor→get_advertisedValue() Returns the current value of the monitor (no more than 6 characters).
wakeupmonitor→get_errorMessage() Returns the error message of the latest error with the monitor.
wakeupmonitor→get_errorType() Returns the numerical error code of the latest error with the monitor.
wakeupmonitor→get_friendlyName() Returns a global identifier of the monitor in the format `MODULE_NAME.FUNCTION_NAME`.
wakeupmonitor→get_functionDescriptor() Returns a unique identifier of type `YFUN_DESCR` corresponding to the function.
wakeupmonitor→get_functionId() Returns the hardware identifier of the monitor, without reference to the module.
wakeupmonitor→get_hardwareId() Returns the unique hardware identifier of the monitor in the form `SERIAL.FUNCTIONID`.
wakeupmonitor→get_logicalName() Returns the logical name of the monitor.
wakeupmonitor→get_module() Gets the `YModule` object for the device on which the function is located.
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)

wakeupmonitor→get_powerDuration()

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

wakeupmonitor→get_sleepCountdown()

Returns the delay before the next sleep period.

wakeupmonitor→get_userData()

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

wakeupmonitor→get_wakeUpReason()

Returns the latest wake up reason.

wakeupmonitor→get_wakeUpState()

Returns the current state of the monitor

wakeupmonitor→isOnline()

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

wakeupmonitor→isOnline_async(**callback**, **context**)

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

wakeupmonitor→load(**msValidity**)

Preloads the monitor cache with a specified validity duration.

wakeupmonitor→load_async(**msValidity**, **callback**, **context**)

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

wakeupmonitor→nextWakeUpMonitor()

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

wakeupmonitor→registerValueCallback(**callback**)

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

wakeupmonitor→resetSleepCountDown()

Resets the sleep countdown.

wakeupmonitor→set_logicalName(**newval**)

Changes the logical name of the monitor.

wakeupmonitor→set_nextWakeUp(**newval**)

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

wakeupmonitor→set_powerDuration(**newval**)

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

wakeupmonitor→set_sleepCountdown(**newval**)

Changes the delay before the next sleep period.

wakeupmonitor→set_userData(**data**)

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

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.

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.

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.

wakeupmonitor→wait_async(**callback**, **context**)

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

wakeupmonitor→wakeUp()

Forces a wake up.

YWakeUpMonitor.FindWakeUpMonitor() YWakeUpMonitor yFindWakeUpMonitor()

Retrieves a monitor for a given identifier.

```
js      function yFindWakeUpMonitor( func)
nodejs  function FindWakeUpMonitor( func)
php     function yFindWakeUpMonitor( $func)
cpp     YWakeUpMonitor* yFindWakeUpMonitor( const string& func)
m       +(YWakeUpMonitor*) FindWakeUpMonitor :(NSString*) func
pas     function yFindWakeUpMonitor( func: string): TYWakeUpMonitor
vb      function yFindWakeUpMonitor( ByVal func As String) As YWakeUpMonitor
cs      YWakeUpMonitor FindWakeUpMonitor( string func)
java    YWakeUpMonitor FindWakeUpMonitor( String func)
py      def FindWakeUpMonitor( func)
```

The identifier can be specified using several formats:

- FunctionLogicalName
- ModuleSerialNumber.FunctionIdentifier
- ModuleSerialNumber.FunctionLogicalName
- ModuleLogicalName.FunctionIdentifier
- ModuleLogicalName.FunctionLogicalName

This function does not require that the monitor is online at the time it is invoked. The returned object is nevertheless valid. Use the method `YWakeUpMonitor.isOnline()` to test if the monitor is indeed online at a given time. In case of ambiguity when looking for a monitor by logical name, no error is notified: the first instance found is returned. The search is performed first by hardware name, then by logical name.

Parameters :

func a string that uniquely characterizes the monitor

Returns :

a `YWakeUpMonitor` object allowing you to drive the monitor.

YWakeUpMonitor.FirstWakeUpMonitor() YWakeUpMonitor yFirstWakeUpMonitor()

Starts the enumeration of monitors currently accessible.

js	function **yFirstWakeUpMonitor**( )
nodejs	function **FirstWakeUpMonitor**( )
php	function **yFirstWakeUpMonitor**( )
cpp	YWakeUpMonitor* **yFirstWakeUpMonitor**( )
m	+(YWakeUpMonitor*) **FirstWakeUpMonitor**
pas	function **yFirstWakeUpMonitor**( ): TYWakeUpMonitor
vb	function **yFirstWakeUpMonitor**( ) As YWakeUpMonitor
cs	YWakeUpMonitor **FirstWakeUpMonitor**( )
java	YWakeUpMonitor **FirstWakeUpMonitor**( )
py	def **FirstWakeUpMonitor**( )

Use the method `YWakeUpMonitor.nextWakeUpMonitor()` to iterate on next monitors.

Returns :

a pointer to a `YWakeUpMonitor` object, corresponding to the first monitor currently online, or a `null` pointer if there are none.

wakeupmonitor→describe() YWakeUpMonitor

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

```
js      function describe( )
nodejs  function describe( )
php     function describe( )
cpp     string describe( )
m       -(NSString*) describe
pas     function describe( ): string
vb      function describe( ) As String
cs      string describe( )
java    String describe( )
py      def describe( )
```

More precisely, TYPE is the type of the function, NAME it the name used for the first access to the function, SERIAL is the serial number of the module if the module is connected or "unresolved", and FUNCTIONID is the hardware identifier of the function if the module is connected. For example, this method returns Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1 if the module is already connected or Relay(BadCustomeName.relay1)=unresolved if the module has not yet been connected. This method does not trigger any USB or TCP transaction and can therefore be used in a debugger.

Returns :

a string that describes the monitor (ex: Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1)

wakeupmonitor→get_advertisedValue() YWakeUpMonitor
wakeupmonitor→advertisedValue()

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

```
js      function get_advertisedValue( )
nodejs  function get_advertisedValue( )
php     function get_advertisedValue( )
cpp     string get_advertisedValue( )
m       -(NSString*) advertisedValue
pas     function get_advertisedValue( ): string
vb      function get_advertisedValue( ) As String
cs      string get_advertisedValue( )
java    String get_advertisedValue( )
py      def get_advertisedValue( )
cmd     YWakeUpMonitor target get_advertisedValue
```

Returns :

a string corresponding to the current value of the monitor (no more than 6 characters).

On failure, throws an exception or returns `Y_ADVERTISEDVALUE_INVALID`.

wakeupmonitor→get_errorMessage() YWakeUpMonitor
wakeupmonitor→errorMessage()

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

```
js       function get_errorMessage( )
nodejs   function get_errorMessage( )
php      function get_errorMessage( )
cpp      string get_errorMessage( )
m        -(NSString*) errorMessage
pas      function get_errorMessage( ): string
vb       function get_errorMessage( ) As String
cs       string get_errorMessage( )
java     String get_errorMessage( )
py       def get_errorMessage( )
```

This method is mostly useful when using the Yoctopuce library with exceptions disabled.

Returns :
a string corresponding to the latest error message that occured while using the monitor object

wakeupmonitor→get_errorType() YWakeUpMonitor
wakeupmonitor→errorType()

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

```
js      function get_errorType( )
nodejs  function get_errorType( )
php     function get_errorType( )
cpp     YRETCODE get_errorType( )
pas     function get_errorType( ): YRETCODE
vb      function get_errorType( ) As YRETCODE
cs      YRETCODE get_errorType( )
java    int get_errorType( )
py      def get_errorType( )
```

This method is mostly useful when using the Yoctopuce library with exceptions disabled.

Returns :
a number corresponding to the code of the latest error that occured while using the monitor object

wakeupmonitor→get_friendlyName() YWakeUpMonitor
wakeupmonitor→friendlyName()

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

```
js      function get_friendlyName( )
nodejs  function get_friendlyName( )
php     function get_friendlyName( )
cpp     string get_friendlyName( )
m       -(NSString*) friendlyName
cs      string get_friendlyName( )
java    String get_friendlyName( )
py      def get_friendlyName( )
```

The returned string uses the logical names of the module and of the monitor if they are defined, otherwise the serial number of the module and the hardware identifier of the monitor (for exemple: MyCustomName.relay1)

Returns :

a string that uniquely identifies the monitor using logical names (ex: MyCustomName.relay1)

On failure, throws an exception or returns Y_FRIENDLYNAME_INVALID.

wakeupmonitor→get_functionDescriptor() YWakeUpMonitor
wakeupmonitor→functionDescriptor()

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

js	function **get_functionDescriptor**( )
nodejs	function **get_functionDescriptor**( )
php	function **get_functionDescriptor**( )
cpp	YFUN_DESCR **get_functionDescriptor**( )
m	-(YFUN_DESCR) functionDescriptor
pas	function **get_functionDescriptor**( ): YFUN_DESCR
vb	function **get_functionDescriptor**( ) As YFUN_DESCR
cs	YFUN_DESCR **get_functionDescriptor**( )
java	String **get_functionDescriptor**( )
py	def **get_functionDescriptor**( )

This identifier can be used to test if two instances of YFunction reference the same physical function on the same physical device.

Returns :

an identifier of type YFUN_DESCR.

If the function has never been contacted, the returned value is Y_FUNCTIONDESCRIPTOR_INVALID.

wakeupmonitor→get_functionId() YWakeUpMonitor
wakeupmonitor→functionId()

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

```
js      function get_functionId( )
nodejs  function get_functionId( )
php     function get_functionId( )
cpp     string get_functionId( )
m       -(NSString*) functionId
vb      function get_functionId( ) As String
cs      string get_functionId( )
java    String get_functionId( )
py      def get_functionId( )
```

For example `relay1`

Returns :

a string that identifies the monitor (ex: `relay1`)

On failure, throws an exception or returns `Y_FUNCTIONID_INVALID`.

wakeupmonitor→get_hardwareId() YWakeUpMonitor
wakeupmonitor→hardwareId()

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

js	function **get_hardwareId**( )
nodejs	function **get_hardwareId**( )
php	function **get_hardwareId**( )
cpp	string **get_hardwareId**( )
m	-(NSString*) hardwareId
vb	function **get_hardwareId**( ) As String
cs	string **get_hardwareId**( )
java	String **get_hardwareId**( )
py	def **get_hardwareId**( )

The unique hardware identifier is composed of the device serial number and of the hardware identifier of the monitor (for example RELAYLO1-123456.relay1).

Returns :

a string that uniquely identifies the monitor (ex: RELAYLO1-123456.relay1)

On failure, throws an exception or returns Y_HARDWAREID_INVALID.

wakeupmonitor→get_logicalName() YWakeUpMonitor
wakeupmonitor→logicalName()

Returns the logical name of the monitor.

```
js      function get_logicalName( )
nodejs  function get_logicalName( )
php     function get_logicalName( )
cpp     string get_logicalName( )
m       -(NSString*) logicalName
pas     function get_logicalName( ): string
vb      function get_logicalName( ) As String
cs      string get_logicalName( )
java    String get_logicalName( )
py      def get_logicalName( )
cmd     YWakeUpMonitor target get_logicalName
```

Returns :

a string corresponding to the logical name of the monitor.

On failure, throws an exception or returns `Y_LOGICALNAME_INVALID`.

wakeupmonitor→get_module() YWakeUpMonitor
wakeupmonitor→module()

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

```
js      function get_module( )
nodejs  function get_module( )
php     function get_module( )
cpp     YModule * get_module( )
m       -(YModule*) module
pas     function get_module( ): TYModule
vb      function get_module( ) As YModule
cs      YModule get_module( )
java    YModule get_module( )
py      def get_module( )
```

If the function cannot be located on any module, the returned instance of YModule is not shown as on-line.

Returns :
an instance of YModule

wakeupmonitor→get_module_async() YWakeUpMonitor
wakeupmonitor→module_async()

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

```
  js  function get_module_async( callback, context)
nodejs function get_module_async( callback, context)
```

If the function cannot be located on any module, the returned YModule object does not show as online.

This asynchronous version exists only in Javascript. It uses a callback instead of a return value in order to avoid blocking Firefox javascript VM that does not implement context switching during blocking I/O calls. See the documentation section on asynchronous Javascript calls for more details.

Parameters :

callback callback function that is invoked when the result is known. The callback function receives three arguments: the caller-specific context object, the receiving function object and the requested YModule object

context caller-specific object that is passed as-is to the callback function

Returns :

nothing : the result is provided to the callback.

wakeupmonitor→get_nextWakeUp() YWakeUpMonitor
wakeupmonitor→nextWakeUp()

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

```
js      function get_nextWakeUp( )
nodejs  function get_nextWakeUp( )
php     function get_nextWakeUp( )
cpp     s64 get_nextWakeUp( )
m       -(s64) nextWakeUp
pas     function get_nextWakeUp( ): int64
vb      function get_nextWakeUp( ) As Long
cs      long get_nextWakeUp( )
java    long get_nextWakeUp( )
py      def get_nextWakeUp( )
```

Returns :

an integer corresponding to the next scheduled wake up date/time (UNIX format)

On failure, throws an exception or returns `Y_NEXTWAKEUP_INVALID`.

wakeupmonitor→get_powerDuration() YWakeUpMonitor
wakeupmonitor→powerDuration()

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

```
js      function get_powerDuration( )
nodejs  function get_powerDuration( )
php     function get_powerDuration( )
cpp     int get_powerDuration( )
m       -(int) powerDuration
pas     function get_powerDuration( ): LongInt
vb      function get_powerDuration( ) As Integer
cs      int get_powerDuration( )
java    int get_powerDuration( )
py      def get_powerDuration( )
cmd     YWakeUpMonitor target get_powerDuration
```

Returns :

an integer corresponding to the maximal wake up time (in seconds) before automatically going to sleep

On failure, throws an exception or returns `Y_POWERDURATION_INVALID`.

wakeupmonitor→get_sleepCountdown() YWakeUpMonitor
wakeupmonitor→sleepCountdown()

Returns the delay before the next sleep period.

```
js      function get_sleepCountdown( )
nodejs  function get_sleepCountdown( )
php     function get_sleepCountdown( )
cpp     int get_sleepCountdown( )
m       -(int) sleepCountdown
pas     function get_sleepCountdown( ): LongInt
vb      function get_sleepCountdown( ) As Integer
cs      int get_sleepCountdown( )
java    int get_sleepCountdown( )
py      def get_sleepCountdown( )
cmd     YWakeUpMonitor target get_sleepCountdown
```

Returns :

an integer corresponding to the delay before the next sleep period

On failure, throws an exception or returns `Y_SLEEPCOUNTDOWN_INVALID`.

wakeupmonitor→get_userData() YWakeUpMonitor
wakeupmonitor→userData()

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

```
js      function get_userData( )
nodejs  function get_userData( )
php     function get_userData( )
cpp     void * get_userData( )
m       -(void*) userData
pas     function get_userData( ): Tobject
vb      function get_userData( ) As Object
cs      object get_userData( )
java    Object get_userData( )
py      def get_userData( )
```

This attribute is never touched directly by the API, and is at disposal of the caller to store a context.

Returns :
the object stored previously by the caller.

wakeupmonitor→get_wakeUpReason() YWakeUpMonitor
wakeupmonitor→wakeUpReason()

Returns the latest wake up reason.

js	function **get_wakeUpReason**( )
nodejs	function **get_wakeUpReason**( )
php	function **get_wakeUpReason**( )
cpp	Y_WAKEUPREASON_enum **get_wakeUpReason**( )
m	-(Y_WAKEUPREASON_enum) wakeUpReason
pas	function **get_wakeUpReason**( ): Integer
vb	function **get_wakeUpReason**( ) As Integer
cs	int **get_wakeUpReason**( )
java	int **get_wakeUpReason**( )
py	def **get_wakeUpReason**( )
cmd	YWakeUpMonitor **target get_wakeUpReason**

Returns :

a value among `Y_WAKEUPREASON_USBPOWER`, `Y_WAKEUPREASON_EXTPOWER`, `Y_WAKEUPREASON_ENDOFSLEEP`, `Y_WAKEUPREASON_EXTSIG1`, `Y_WAKEUPREASON_SCHEDULE1` and `Y_WAKEUPREASON_SCHEDULE2` corresponding to the latest wake up reason

On failure, throws an exception or returns `Y_WAKEUPREASON_INVALID`.

wakeupmonitor→get_wakeUpState() YWakeUpMonitor
wakeupmonitor→wakeUpState()

Returns the current state of the monitor

```
js       function get_wakeUpState( )
nodejs   function get_wakeUpState( )
php      function get_wakeUpState( )
cpp      Y_WAKEUPSTATE_enum get_wakeUpState( )
m        -(Y_WAKEUPSTATE_enum) wakeUpState
pas      function get_wakeUpState( ): Integer
vb       function get_wakeUpState( ) As Integer
cs       int get_wakeUpState( )
java     int get_wakeUpState( )
py       def get_wakeUpState( )
```

Returns :

either `Y_WAKEUPSTATE_SLEEPING` or `Y_WAKEUPSTATE_AWAKE`, according to the current state of the monitor

On failure, throws an exception or returns `Y_WAKEUPSTATE_INVALID`.

wakeupmonitor→isOnline() YWakeUpMonitor

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

```
js      function isOnline( )
nodejs  function isOnline( )
php     function isOnline( )
cpp     bool isOnline( )
m       -(BOOL) isOnline
pas     function isOnline( ): boolean
vb      function isOnline( ) As Boolean
cs      bool isOnline( )
java    boolean isOnline( )
py      def isOnline( )
```

If there is a cached value for the monitor in cache, that has not yet expired, the device is considered reachable. No exception is raised if there is an error while trying to contact the device hosting the monitor.

Returns :

`true` if the monitor can be reached, and `false` otherwise

wakeupmonitor→isOnline_async() YWakeUpMonitor

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

```
js      function isOnline_async( callback, context)
nodejs  function isOnline_async( callback, context)
```

If there is a cached value for the monitor in cache, that has not yet expired, the device is considered reachable. No exception is raised if there is an error while trying to contact the device hosting the requested function.

This asynchronous version exists only in Javascript. It uses a callback instead of a return value in order to avoid blocking the Javascript virtual machine.

Parameters :

callback callback function that is invoked when the result is known. The callback function receives three arguments: the caller-specific context object, the receiving function object and the boolean result

context caller-specific object that is passed as-is to the callback function

Returns :

nothing : the result is provided to the callback.

wakeupmonitor→load() YWakeUpMonitor

Preloads the monitor cache with a specified validity duration.

```
js      function load( msValidity)
nodejs  function load( msValidity)
php     function load( $msValidity)
cpp     YRETCODE load( int msValidity)
m       -(YRETCODE) load : (int) msValidity
pas     function load( msValidity: integer): YRETCODE
vb      function load( ByVal msValidity As Integer) As YRETCODE
cs      YRETCODE load( int msValidity)
java    int load( long msValidity)
py      def load( msValidity)
```

By default, whenever accessing a device, all function attributes are kept in cache for the standard duration (5 ms). This method can be used to temporarily mark the cache as valid for a longer period, in order to reduce network trafic for instance.

Parameters :

msValidity an integer corresponding to the validity attributed to the loaded function parameters, in milliseconds

Returns :

YAPI_SUCCESS when the call succeeds.

On failure, throws an exception or returns a negative error code.

wakeupmonitor→load_async() YWakeUpMonitor

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

```
js      function load_async( msValidity, callback, context)
nodejs  function load_async( msValidity, callback, context)
```

By default, whenever accessing a device, all function attributes are kept in cache for the standard duration (5 ms). This method can be used to temporarily mark the cache as valid for a longer period, in order to reduce network trafic for instance.

This asynchronous version exists only in Javascript. It uses a callback instead of a return value in order to avoid blocking the Javascript virtual machine.

Parameters :

msValidity	an integer corresponding to the validity of the loaded function parameters, in milliseconds
callback	callback function that is invoked when the result is known. The callback function receives three arguments: the caller-specific context object, the receiving function object and the error code (or YAPI_SUCCESS)
context	caller-specific object that is passed as-is to the callback function

Returns :

nothing : the result is provided to the callback.

wakeupmonitor→nextWakeUpMonitor() YWakeUpMonitor

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

js	function **nextWakeUpMonitor**( )
nodejs	function **nextWakeUpMonitor**( )
php	function **nextWakeUpMonitor**( )
cpp	YWakeUpMonitor * **nextWakeUpMonitor**( )
m	-(YWakeUpMonitor*) **nextWakeUpMonitor**
pas	function **nextWakeUpMonitor**( ): TYWakeUpMonitor
vb	function **nextWakeUpMonitor**( ) As YWakeUpMonitor
cs	YWakeUpMonitor **nextWakeUpMonitor**( )
java	YWakeUpMonitor **nextWakeUpMonitor**( )
py	def **nextWakeUpMonitor**( )

Returns :

a pointer to a `YWakeUpMonitor` object, corresponding to a monitor currently online, or a `null` pointer if there are no more monitors to enumerate.

wakeupmonitor→registerValueCallback() YWakeUpMonitor

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

```
js      function registerValueCallback( callback)
nodejs  function registerValueCallback( callback)
php     function registerValueCallback( $callback)
cpp     int registerValueCallback( YWakeUpMonitorValueCallback callback)
m       -(int) registerValueCallback : (YWakeUpMonitorValueCallback) callback
pas     function registerValueCallback( callback: TYWakeUpMonitorValueCallback): LongInt
vb      function registerValueCallback( ) As Integer
cs      int registerValueCallback( ValueCallback callback)
java    int registerValueCallback( UpdateCallback callback)
py      def registerValueCallback( callback)
```

The callback is invoked only during the execution of `ySleep` or `yHandleEvents`. This provides control over the time when the callback is triggered. For good responsiveness, remember to call one of these two functions periodically. To unregister a callback, pass a null pointer as argument.

Parameters :

callback the callback function to call, or a null pointer. The callback function should take two arguments: the function object of which the value has changed, and the character string describing the new advertised value.

wakeupmonitor→resetSleepCountDown() YWakeUpMonitor

Resets the sleep countdown.

js	function **resetSleepCountDown**( )
nodejs	function **resetSleepCountDown**( )
php	function **resetSleepCountDown**( )
cpp	int **resetSleepCountDown**( )
m	-(int) **resetSleepCountDown**
pas	function **resetSleepCountDown**( ): LongInt
vb	function **resetSleepCountDown**( ) As Integer
cs	int **resetSleepCountDown**( )
java	int **resetSleepCountDown**( )
py	def **resetSleepCountDown**( )
cmd	YWakeUpMonitor **target resetSleepCountDown**

Returns :

YAPI_SUCCESS if the call succeeds. On failure, throws an exception or returns a negative error code.

wakeupmonitor→set_logicalName() YWakeUpMonitor
wakeupmonitor→setLogicalName()

Changes the logical name of the monitor.

```
js      function set_logicalName( newval)
nodejs  function set_logicalName( newval)
php     function set_logicalName( $newval)
cpp     int set_logicalName( const string& newval)
m       -(int) setLogicalName : (NSString*) newval
pas     function set_logicalName( newval: string): integer
vb      function set_logicalName( ByVal newval As String) As Integer
cs      int set_logicalName( string newval)
java    int set_logicalName( String newval)
py      def set_logicalName( newval)
cmd     YWakeUpMonitor target set_logicalName newval
```

You can use `yCheckLogicalName()` prior to this call to make sure that your parameter is valid. Remember to call the `saveToFlash()` method of the module if the modification must be kept.

Parameters :

newval a string corresponding to the logical name of the monitor.

Returns :

`YAPI_SUCCESS` if the call succeeds.

On failure, throws an exception or returns a negative error code.

wakeupmonitor→set_nextWakeUp() YWakeUpMonitor
wakeupmonitor→setNextWakeUp()

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

```
js      function set_nextWakeUp( newval)
nodejs  function set_nextWakeUp( newval)
php     function set_nextWakeUp( $newval)
cpp     int set_nextWakeUp( s64 newval)
m       -(int) setNextWakeUp : (s64) newval
pas     function set_nextWakeUp( newval: int64): integer
vb      function set_nextWakeUp( ByVal newval As Long) As Integer
cs      int set_nextWakeUp( long newval)
java    int set_nextWakeUp( long newval)
py      def set_nextWakeUp( newval)
cmd     YWakeUpMonitor target set_nextWakeUp newval
```

Parameters :

newval an integer corresponding to the days of the week when a wake up must take place

Returns :

YAPI_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

wakeupmonitor→set_powerDuration() YWakeUpMonitor
wakeupmonitor→setPowerDuration()

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

```
js      function set_powerDuration( newval)
nodejs  function set_powerDuration( newval)
php     function set_powerDuration( $newval)
cpp     int set_powerDuration( int newval)
m       -(int) setPowerDuration : (int) newval
pas     function set_powerDuration( newval: LongInt): integer
vb      function set_powerDuration( ByVal newval As Integer) As Integer
cs      int set_powerDuration( int newval)
java    int set_powerDuration( int newval)
py      def set_powerDuration( newval)
cmd     YWakeUpMonitor target set_powerDuration newval
```

Parameters :

newval an integer corresponding to the maximal wake up time (seconds) before automatically going to sleep

Returns :

YAPI_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

wakeupmonitor→set_sleepCountdown() YWakeUpMonitor
wakeupmonitor→setSleepCountdown()

Changes the delay before the next sleep period.

```
js      function set_sleepCountdown( newval)
nodejs  function set_sleepCountdown( newval)
php     function set_sleepCountdown( $newval)
cpp     int set_sleepCountdown( int newval)
m       -(int) setSleepCountdown : (int) newval
pas     function set_sleepCountdown( newval: LongInt): integer
vb      function set_sleepCountdown( ByVal newval As Integer) As Integer
cs      int set_sleepCountdown( int newval)
java    int set_sleepCountdown( int newval)
py      def set_sleepCountdown( newval)
cmd     YWakeUpMonitor target set_sleepCountdown newval
```

Parameters :

newval an integer corresponding to the delay before the next sleep period

Returns :

YAPI_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

wakeupmonitor→set_userData() YWakeUpMonitor
wakeupmonitor→setUserData()

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

```
js      function set_userData( data)
nodejs  function set_userData( data)
php     function set_userData( $data)
cpp     void set_userData( void* data)
m       -(void) setUserData : (void*) data
pas     procedure set_userData( data: Tobject)
vb      procedure set_userData( ByVal data As Object)
cs      void set_userData( object data)
java    void set_userData( Object data)
py      def set_userData( data)
```

This attribute is never touched by the API, and is at disposal of the caller to store a context.

Parameters :

data any kind of object to be stored

wakeupmonitor→sleep() YWakeUpMonitor

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

```
js      function sleep( secBeforeSleep)
nodejs  function sleep( secBeforeSleep)
php     function sleep( $secBeforeSleep)
cpp     int sleep( int secBeforeSleep)
m       -(int) sleep : (int) secBeforeSleep
pas     function sleep( secBeforeSleep: LongInt): LongInt
vb      function sleep( ) As Integer
cs      int sleep( int secBeforeSleep)
java    int sleep( int secBeforeSleep)
py      def sleep( secBeforeSleep)
cmd     YWakeUpMonitor target sleep secBeforeSleep
```

Parameters :

secBeforeSleep number of seconds before going into sleep mode,

Returns :

YAPI_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

wakeupmonitor→sleepFor() YWakeUpMonitor

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.

```
js      function sleepFor( secUntilWakeUp, secBeforeSleep)
nodejs  function sleepFor( secUntilWakeUp, secBeforeSleep)
php     function sleepFor( $secUntilWakeUp, $secBeforeSleep)
cpp     int sleepFor( int secUntilWakeUp, int secBeforeSleep)
m       -(int) sleepFor : (int) secUntilWakeUp : (int) secBeforeSleep
pas     function sleepFor( secUntilWakeUp: LongInt,
                           secBeforeSleep: LongInt): LongInt
vb      function sleepFor( ) As Integer
cs      int sleepFor( int secUntilWakeUp, int secBeforeSleep)
java    int sleepFor( int secUntilWakeUp, int secBeforeSleep)
py      def sleepFor( secUntilWakeUp, secBeforeSleep)
cmd     YWakeUpMonitor target sleepFor secUntilWakeUp secBeforeSleep
```

The count down before sleep can be canceled with resetSleepCountDown.

Parameters :

secUntilWakeUp number of seconds before next wake up

secBeforeSleep number of seconds before going into sleep mode

Returns :

YAPI_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

wakeupmonitor→sleepUntil() YWakeUpMonitor

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.

```
js      function sleepUntil( wakeUpTime, secBeforeSleep)
nodejs  function sleepUntil( wakeUpTime, secBeforeSleep)
php     function sleepUntil( $wakeUpTime, $secBeforeSleep)
cpp     int sleepUntil( int wakeUpTime, int secBeforeSleep)
m       -(int) sleepUntil : (int) wakeUpTime : (int) secBeforeSleep
pas     function sleepUntil( wakeUpTime: LongInt,
                             secBeforeSleep: LongInt): LongInt
vb      function sleepUntil( ) As Integer
cs      int sleepUntil( int wakeUpTime, int secBeforeSleep)
java    int sleepUntil( int wakeUpTime, int secBeforeSleep)
py      def sleepUntil( wakeUpTime, secBeforeSleep)
cmd     YWakeUpMonitor target sleepUntil wakeUpTime secBeforeSleep
```

The count down before sleep can be canceled with resetSleepCountDown.

Parameters :

wakeUpTime wake-up datetime (UNIX format)

secBeforeSleep number of seconds before going into sleep mode

Returns :

YAPI_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

wakeupmonitor→wait_async() YWakeUpMonitor

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

```
js      function wait_async( callback, context)
nodejs  function wait_async( callback, context)
```

The callback function can therefore freely issue synchronous or asynchronous commands, without risking to block the Javascript VM.

Parameters :

callback callback function that is invoked when all pending commands on the module are completed. The callback function receives two arguments: the caller-specific context object and the receiving function object.

context caller-specific object that is passed as-is to the callback function

Returns :

nothing.

wakeupmonitor→wakeUp() YWakeUpMonitor

Forces a wake up.

```
js      function wakeUp( )
nodejs  function wakeUp( )
php     function wakeUp( )
cpp     int wakeUp( )
m       -(int) wakeUp
pas     function wakeUp( ): LongInt
vb      function wakeUp( ) As Integer
cs      int wakeUp( )
java    int wakeUp( )
py      def wakeUp( )
cmd     YWakeUpMonitor target wakeUp
```

11.6. WakeUpSchedule function interface

The WakeUpSchedule function 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:

```
js      <script type='text/javascript' src='yocto_wakeupschedule.js'></script>
nodejs  var yoctolib = require('yoctolib');
        var YWakeUpSchedule = yoctolib.YWakeUpSchedule;
php     require_once('yocto_wakeupschedule.php');
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;
py      from yocto_wakeupschedule import *
```

Global functions
yFindWakeUpSchedule(**func**) Retrieves a wake up schedule for a given identifier.
yFirstWakeUpSchedule() Starts the enumeration of wake up schedules currently accessible.
`YWakeUpSchedule` methods
wakeupschedule→describe() Returns a short text that describes unambiguously the instance of the wake up schedule in the form `TYPE(NAME)=SERIAL.FUNCTIONID`.
wakeupschedule→get_advertisedValue() Returns the current value of the wake up schedule (no more than 6 characters).
wakeupschedule→get_errorMessage() Returns the error message of the latest error with the wake up schedule.
wakeupschedule→get_errorType() Returns the numerical error code of the latest error with the wake up schedule.
wakeupschedule→get_friendlyName() Returns a global identifier of the wake up schedule in the format `MODULE_NAME.FUNCTION_NAME`.
wakeupschedule→get_functionDescriptor() Returns a unique identifier of type `YFUN_DESCR` corresponding to the function.
wakeupschedule→get_functionId() Returns the hardware identifier of the wake up schedule, without reference to the module.
wakeupschedule→get_hardwareId() Returns the unique hardware identifier of the wake up schedule in the form `SERIAL.FUNCTIONID`.
wakeupschedule→get_hours() Returns the hours scheduled for wake up.
wakeupschedule→get_logicalName() Returns the logical name of the wake up schedule.
wakeupschedule→get_minutes() Returns all the minutes of each hour that are scheduled for wake up.
wakeupschedule→get_minutesA()

Returns the minutes in the 00-29 interval of each hour scheduled for wake up.

wakeupschedule→get_minutesB()

Returns the minutes in the 30-59 intervalof each hour scheduled for wake up.

wakeupschedule→get_module()

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

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.

wakeupschedule→get_months()

Returns the months scheduled for wake up.

wakeupschedule→get_nextOccurence()

Returns the date/time (seconds) of the next wake up occurence

wakeupschedule→get_userData()

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

wakeupschedule→get_weekDays()

Returns the days of the week scheduled for wake up.

wakeupschedule→isOnline()

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

wakeupschedule→isOnline_async(**callback**, **context**)

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

wakeupschedule→load(**msValidity**)

Preloads the wake up schedule cache with a specified validity duration.

wakeupschedule→load_async(**msValidity**, **callback**, **context**)

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

wakeupschedule→nextWakeUpSchedule()

Continues the enumeration of wake up schedules started using `yFirstWakeUpSchedule()`.

wakeupschedule→registerValueCallback(**callback**)

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

wakeupschedule→set_hours(**newval**)

Changes the hours when a wake up must take place.

wakeupschedule→set_logicalName(**newval**)

Changes the logical name of the wake up schedule.

wakeupschedule→set_minutes(**bitmap**)

Changes all the minutes where a wake up must take place.

wakeupschedule→set_minutesA(**newval**)

Changes the minutes in the 00-29 interval when a wake up must take place.

wakeupschedule→set_minutesB(**newval**)

Changes the minutes in the 30-59 interval when a wake up must take place.

wakeupschedule→set_monthDays(**newval**)

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

wakeupschedule→set_months(**newval**)

Changes the months when a wake up must take place.

wakeupschedule→set_userData(**data**)

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

wakeupschedule→set_weekDays(newval)

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

wakeupschedule→wait_async(callback, context)

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

YWakeUpSchedule.FindWakeUpSchedule() YWakeUpSchedule yFindWakeUpSchedule()

Retrieves a wake up schedule for a given identifier.

```
js      function yFindWakeUpSchedule( func)
nodejs  function FindWakeUpSchedule( func)
php     function yFindWakeUpSchedule( $func)
cpp     YWakeUpSchedule* yFindWakeUpSchedule( const string& func)
m       +(YWakeUpSchedule*) FindWakeUpSchedule :(NSString*) func
pas     function yFindWakeUpSchedule( func: string): TYWakeUpSchedule
vb      function yFindWakeUpSchedule( ByVal func As String) As YWakeUpSchedule
cs      YWakeUpSchedule FindWakeUpSchedule( string func)
java    YWakeUpSchedule FindWakeUpSchedule( String func)
py      def FindWakeUpSchedule( func)
```

The identifier can be specified using several formats:

- FunctionLogicalName
- ModuleSerialNumber.FunctionIdentifier
- ModuleSerialNumber.FunctionLogicalName
- ModuleLogicalName.FunctionIdentifier
- ModuleLogicalName.FunctionLogicalName

This function does not require that the wake up schedule is online at the time it is invoked. The returned object is nevertheless valid. Use the method `YWakeUpSchedule.isOnline()` to test if the wake up schedule is indeed online at a given time. In case of ambiguity when looking for a wake up schedule by logical name, no error is notified: the first instance found is returned. The search is performed first by hardware name, then by logical name.

Parameters :

func a string that uniquely characterizes the wake up schedule

Returns :

a `YWakeUpSchedule` object allowing you to drive the wake up schedule.

YWakeUpSchedule.FirstWakeUpSchedule() YWakeUpSchedule yFirstWakeUpSchedule()

Starts the enumeration of wake up schedules currently accessible.

js	function **yFirstWakeUpSchedule**( )
nodejs	function **FirstWakeUpSchedule**( )
php	function **yFirstWakeUpSchedule**( )
cpp	YWakeUpSchedule* **yFirstWakeUpSchedule**( )
m	+(YWakeUpSchedule*) **FirstWakeUpSchedule**
pas	function **yFirstWakeUpSchedule**( ): TYWakeUpSchedule
vb	function **yFirstWakeUpSchedule**( ) As YWakeUpSchedule
cs	YWakeUpSchedule **FirstWakeUpSchedule**( )
java	YWakeUpSchedule **FirstWakeUpSchedule**( )
py	def **FirstWakeUpSchedule**( )

Use the method `YWakeUpSchedule.nextWakeUpSchedule()` to iterate on next wake up schedules.

Returns :

a pointer to a `YWakeUpSchedule` object, corresponding to the first wake up schedule currently online, or a `null` pointer if there are none.

wakeupschedule→describe() YWakeUpSchedule

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

```
js      function describe( )
nodejs  function describe( )
php     function describe( )
cpp     string describe( )
m       -(NSString*) describe
pas     function describe( ): string
vb      function describe( ) As String
cs      string describe( )
java    String describe( )
py      def describe( )
```

More precisely, TYPE is the type of the function, NAME it the name used for the first access to the function, SERIAL is the serial number of the module if the module is connected or "unresolved", and FUNCTIONID is the hardware identifier of the function if the module is connected. For example, this method returns Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1 if the module is already connected or Relay(BadCustomeName.relay1)=unresolved if the module has not yet been connected. This method does not trigger any USB or TCP transaction and can therefore be used in a debugger.

Returns :
a string that describes the wake up schedule (ex: Relay(MyCustomName.relay1)=RELAYLO1-123456.relay1)

wakeupschedule→get_advertisedValue() YWakeUpSchedule
wakeupschedule→advertisedValue()

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

```
js      function get_advertisedValue( )
nodejs  function get_advertisedValue( )
php     function get_advertisedValue( )
cpp     string get_advertisedValue( )
m       -(NSString*) advertisedValue
pas     function get_advertisedValue( ): string
vb      function get_advertisedValue( ) As String
cs      string get_advertisedValue( )
java    String get_advertisedValue( )
py      def get_advertisedValue( )
cmd     YWakeUpSchedule target get_advertisedValue
```

Returns :

a string corresponding to the current value of the wake up schedule (no more than 6 characters).

On failure, throws an exception or returns `Y_ADVERTISEDVALUE_INVALID`.

wakeupschedule→get_errorMessage() YWakeUpSchedule
wakeupschedule→errorMessage()

Returns the error message of the latest error with the wake up schedule.

js	function **get_errorMessage**( )
nodejs	function **get_errorMessage**( )
php	function **get_errorMessage**( )
cpp	string **get_errorMessage**( )
m	-(NSString*) errorMessage
pas	function **get_errorMessage**( ): string
vb	function **get_errorMessage**( ) As String
cs	string **get_errorMessage**( )
java	String **get_errorMessage**( )
py	def **get_errorMessage**( )

This method is mostly useful when using the Yoctopuce library with exceptions disabled.

Returns :
a string corresponding to the latest error message that occured while using the wake up schedule object

wakeupschedule→get_errorType() YWakeUpSchedule
wakeupschedule→errorType()

Returns the numerical error code of the latest error with the wake up schedule.

```
js      function get_errorType( )
nodejs  function get_errorType( )
php     function get_errorType( )
cpp     YRETCODE get_errorType( )
pas     function get_errorType( ): YRETCODE
vb      function get_errorType( ) As YRETCODE
cs      YRETCODE get_errorType( )
java    int get_errorType( )
py      def get_errorType( )
```

This method is mostly useful when using the Yoctopuce library with exceptions disabled.

Returns :
a number corresponding to the code of the latest error that occured while using the wake up schedule object

wakeupschedule→get_friendlyName() YWakeUpSchedule
wakeupschedule→friendlyName()

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

```
js      function get_friendlyName( )
nodejs  function get_friendlyName( )
php     function get_friendlyName( )
cpp     string get_friendlyName( )
m       -(NSString*) friendlyName
cs      string get_friendlyName( )
java    String get_friendlyName( )
py      def get_friendlyName( )
```

The returned string uses the logical names of the module and of the wake up schedule if they are defined, otherwise the serial number of the module and the hardware identifier of the wake up schedule (for exemple: `MyCustomName.relay1`)

Returns :

a string that uniquely identifies the wake up schedule using logical names (ex: `MyCustomName.relay1`)

On failure, throws an exception or returns `Y_FRIENDLYNAME_INVALID`.

wakeupschedule→get_functionDescriptor() YWakeUpSchedule
wakeupschedule→functionDescriptor()

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

js	function **get_functionDescriptor**( )
nodejs	function **get_functionDescriptor**( )
php	function **get_functionDescriptor**( )
cpp	YFUN_DESCR **get_functionDescriptor**( )
m	-(YFUN_DESCR) functionDescriptor
pas	function **get_functionDescriptor**( ): YFUN_DESCR
vb	function **get_functionDescriptor**( ) As YFUN_DESCR
cs	YFUN_DESCR **get_functionDescriptor**( )
java	String **get_functionDescriptor**( )
py	def **get_functionDescriptor**( )

This identifier can be used to test if two instances of YFunction reference the same physical function on the same physical device.

Returns :

an identifier of type YFUN_DESCR.

If the function has never been contacted, the returned value is Y_FUNCTIONDESCRIPTOR_INVALID.

wakeupschedule→get_functionId() YWakeUpSchedule
wakeupschedule→functionId()

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

js	function **get_functionId**( )
nodejs	function **get_functionId**( )
php	function **get_functionId**( )
cpp	string **get_functionId**( )
m	-(NSString*) functionId
vb	function **get_functionId**( ) As String
cs	string **get_functionId**( )
java	String **get_functionId**( )
py	def **get_functionId**( )

For example `relay1`

Returns :

a string that identifies the wake up schedule (ex: `relay1`)

On failure, throws an exception or returns `Y_FUNCTIONID_INVALID`.

wakeupschedule→get_hardwareId() YWakeUpSchedule
wakeupschedule→hardwareId()

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

```
js      function get_hardwareId( )
nodejs  function get_hardwareId( )
php     function get_hardwareId( )
cpp     string get_hardwareId( )
m       -(NSString*) hardwareId
vb      function get_hardwareId( ) As String
cs      string get_hardwareId( )
java    String get_hardwareId( )
py      def get_hardwareId( )
```

The unique hardware identifier is composed of the device serial number and of the hardware identifier of the wake up schedule (for example RELAYLO1-123456.relay1).

Returns :

a string that uniquely identifies the wake up schedule (ex: RELAYLO1-123456.relay1)

On failure, throws an exception or returns Y_HARDWAREID_INVALID.

wakeupschedule→get_hours() YWakeUpSchedule
wakeupschedule→hours()

Returns the hours scheduled for wake up.

```
js      function get_hours( )
nodejs  function get_hours( )
php     function get_hours( )
cpp     int get_hours( )
m       -(int) hours
pas     function get_hours( ): LongInt
vb      function get_hours( ) As Integer
cs      int get_hours( )
java    int get_hours( )
py      def get_hours( )
cmd     YWakeUpSchedule target get_hours
```

Returns :
an integer corresponding to the hours scheduled for wake up

On failure, throws an exception or returns `Y_HOURS_INVALID`.

wakeupschedule→get_logicalName() YWakeUpSchedule
wakeupschedule→logicalName()

Returns the logical name of the wake up schedule.

```
js      function get_logicalName( )
nodejs  function get_logicalName( )
php     function get_logicalName( )
cpp     string get_logicalName( )
m       -(NSString*) logicalName
pas     function get_logicalName( ): string
vb      function get_logicalName( ) As String
cs      string get_logicalName( )
java    String get_logicalName( )
py      def get_logicalName( )
cmd     YWakeUpSchedule target get_logicalName
```

Returns :
a string corresponding to the logical name of the wake up schedule.

On failure, throws an exception or returns `Y_LOGICALNAME_INVALID`.

wakeupschedule→get_minutes() YWakeUpSchedule
wakeupschedule→minutes()

Returns all the minutes of each hour that are scheduled for wake up.

```
js      function get_minutes( )
nodejs  function get_minutes( )
php     function get_minutes( )
cpp     s64 get_minutes( )
m       -(s64) minutes
pas     function get_minutes( ): int64
vb      function get_minutes( ) As Long
cs      long get_minutes( )
java    long get_minutes( )
py      def get_minutes( )
cmd     YWakeUpSchedule target get_minutes
```

wakeupschedule→get_minutesA() YWakeUpSchedule
wakeupschedule→minutesA()

Returns the minutes in the 00-29 interval of each hour scheduled for wake up.

```
js      function get_minutesA( )
nodejs  function get_minutesA( )
php     function get_minutesA( )
cpp     int get_minutesA( )
m       -(int) minutesA
pas     function get_minutesA( ): LongInt
vb      function get_minutesA( ) As Integer
cs      int get_minutesA( )
java    int get_minutesA( )
py      def get_minutesA( )
cmd     YWakeUpSchedule target get_minutesA
```

Returns :

an integer corresponding to the minutes in the 00-29 interval of each hour scheduled for wake up

On failure, throws an exception or returns `Y_MINUTESA_INVALID`.

wakeupschedule→get_minutesB() YWakeUpSchedule
wakeupschedule→minutesB()

Returns the minutes in the 30-59 intervalof each hour scheduled for wake up.

```
js      function get_minutesB( )
nodejs  function get_minutesB( )
php     function get_minutesB( )
cpp     int get_minutesB( )
m       -(int) minutesB
pas     function get_minutesB( ): LongInt
vb      function get_minutesB( ) As Integer
cs      int get_minutesB( )
java    int get_minutesB( )
py      def get_minutesB( )
cmd     YWakeUpSchedule target get_minutesB
```

Returns :

an integer corresponding to the minutes in the 30-59 intervalof each hour scheduled for wake up

On failure, throws an exception or returns `Y_MINUTESB_INVALID`.

wakeupschedule→get_module() YWakeUpSchedule
wakeupschedule→module()

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

```
js      function get_module( )
nodejs  function get_module( )
php     function get_module( )
cpp     YModule * get_module( )
m       -(YModule*) module
pas     function get_module( ): TYModule
vb      function get_module( ) As YModule
cs      YModule get_module( )
java    YModule get_module( )
py      def get_module( )
```

If the function cannot be located on any module, the returned instance of YModule is not shown as on-line.

Returns :
an instance of YModule

wakeupschedule→get_module_async() YWakeUpSchedule
wakeupschedule→module_async()

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

```
js      function get_module_async( callback, context)
nodejs  function get_module_async( callback, context)
```

If the function cannot be located on any module, the returned YModule object does not show as online.

This asynchronous version exists only in Javascript. It uses a callback instead of a return value in order to avoid blocking Firefox javascript VM that does not implement context switching during blocking I/O calls. See the documentation section on asynchronous Javascript calls for more details.

Parameters :

callback callback function that is invoked when the result is known. The callback function receives three arguments: the caller-specific context object, the receiving function object and the requested YModule object

context caller-specific object that is passed as-is to the callback function

Returns :

nothing : the result is provided to the callback.

wakeupschedule→get_monthDays() YWakeUpSchedule
wakeupschedule→monthDays()

Returns the days of the month scheduled for wake up.

```
js      function get_monthDays( )
nodejs  function get_monthDays( )
php     function get_monthDays( )
cpp     int get_monthDays( )
m       -(int) monthDays
pas     function get_monthDays( ): LongInt
vb      function get_monthDays( ) As Integer
cs      int get_monthDays( )
java    int get_monthDays( )
py      def get_monthDays( )
cmd     YWakeUpSchedule target get_monthDays
```

Returns :
an integer corresponding to the days of the month scheduled for wake up

On failure, throws an exception or returns `Y_MONTHDAYS_INVALID`.

wakeupschedule→get_months() YWakeUpSchedule
wakeupschedule→months()

Returns the months scheduled for wake up.

```
js      function get_months( )
nodejs  function get_months( )
php     function get_months( )
cpp     int get_months( )
m       -(int) months
pas     function get_months( ): LongInt
vb      function get_months( ) As Integer
cs      int get_months( )
java    int get_months( )
py      def get_months( )
cmd     YWakeUpSchedule target get_months
```

Returns :

an integer corresponding to the months scheduled for wake up

On failure, throws an exception or returns `Y_MONTHS_INVALID`.

wakeupschedule→get_nextOccurence() YWakeUpSchedule
wakeupschedule→nextOccurence()

Returns the date/time (seconds) of the next wake up occurence

js	function **get_nextOccurence**( )
nodejs	function **get_nextOccurence**( )
php	function **get_nextOccurence**( )
cpp	s64 **get_nextOccurence**( )
m	-(s64) nextOccurence
pas	function **get_nextOccurence**( ): int64
vb	function **get_nextOccurence**( ) As Long
cs	long **get_nextOccurence**( )
java	long **get_nextOccurence**( )
py	def **get_nextOccurence**( )

Returns :

an integer corresponding to the date/time (seconds) of the next wake up occurence

On failure, throws an exception or returns `Y_NEXTOCCURENCE_INVALID`.

wakeupschedule→get_userData() YWakeUpSchedule
wakeupschedule→userData()

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

js	function **get_userData**( )
nodejs	function **get_userData**( )
php	function **get_userData**( )
cpp	void * **get_userData**( )
m	-(void*) userData
pas	function **get_userData**( ): Tobject
vb	function **get_userData**( ) As Object
cs	object **get_userData**( )
java	Object **get_userData**( )
py	def **get_userData**( )

This attribute is never touched directly by the API, and is at disposal of the caller to store a context.

Returns :
the object stored previously by the caller.

wakeupschedule→get_weekDays() YWakeUpSchedule
wakeupschedule→weekDays()

Returns the days of the week scheduled for wake up.

```
js      function get_weekDays( )
nodejs  function get_weekDays( )
php     function get_weekDays( )
cpp     int get_weekDays( )
m       -(int) weekDays
pas     function get_weekDays( ): LongInt
vb      function get_weekDays( ) As Integer
cs      int get_weekDays( )
java    int get_weekDays( )
py      def get_weekDays( )
cmd     YWakeUpSchedule target get_weekDays
```

Returns :

an integer corresponding to the days of the week scheduled for wake up

On failure, throws an exception or returns `Y_WEEKDAYS_INVALID`.

wakeupschedule→isOnline() YWakeUpSchedule

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

```
js      function isOnline( )
nodejs  function isOnline( )
php     function isOnline( )
cpp     bool isOnline( )
m       -(BOOL) isOnline
pas     function isOnline( ): boolean
vb      function isOnline( ) As Boolean
cs      bool isOnline( )
java    boolean isOnline( )
py      def isOnline( )
```

If there is a cached value for the wake up schedule in cache, that has not yet expired, the device is considered reachable. No exception is raised if there is an error while trying to contact the device hosting the wake up schedule.

Returns :

`true` if the wake up schedule can be reached, and `false` otherwise

wakeupschedule→isOnline_async() YWakeUpSchedule

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

```
js      function isOnline_async( callback, context)
nodejs  function isOnline_async( callback, context)
```

If there is a cached value for the wake up schedule in cache, that has not yet expired, the device is considered reachable. No exception is raised if there is an error while trying to contact the device hosting the requested function.

This asynchronous version exists only in Javascript. It uses a callback instead of a return value in order to avoid blocking the Javascript virtual machine.

Parameters :

callback callback function that is invoked when the result is known. The callback function receives three arguments: the caller-specific context object, the receiving function object and the boolean result

context caller-specific object that is passed as-is to the callback function

Returns :

nothing : the result is provided to the callback.

wakeupschedule→load() YWakeUpSchedule

Preloads the wake up schedule cache with a specified validity duration.

```
js      function load( msValidity)
nodejs  function load( msValidity)
php     function load( $msValidity)
cpp     YRETCODE load( int msValidity)
m       -(YRETCODE) load : (int) msValidity
pas     function load( msValidity: integer): YRETCODE
vb      function load( ByVal msValidity As Integer) As YRETCODE
cs      YRETCODE load( int msValidity)
java    int load( long msValidity)
py      def load( msValidity)
```

By default, whenever accessing a device, all function attributes are kept in cache for the standard duration (5 ms). This method can be used to temporarily mark the cache as valid for a longer period, in order to reduce network trafic for instance.

Parameters :

msValidity an integer corresponding to the validity attributed to the loaded function parameters, in milliseconds

Returns :

YAPI_SUCCESS when the call succeeds.

On failure, throws an exception or returns a negative error code.

wakeupschedule→load_async() YWakeUpSchedule

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

```
js      function load_async( msValidity, callback, context)
nodejs  function load_async( msValidity, callback, context)
```

By default, whenever accessing a device, all function attributes are kept in cache for the standard duration (5 ms). This method can be used to temporarily mark the cache as valid for a longer period, in order to reduce network trafic for instance.

This asynchronous version exists only in Javascript. It uses a callback instead of a return value in order to avoid blocking the Javascript virtual machine.

Parameters :

msValidity	an integer corresponding to the validity of the loaded function parameters, in milliseconds
callback	callback function that is invoked when the result is known. The callback function receives three arguments: the caller-specific context object, the receiving function object and the error code (or YAPI_SUCCESS)
context	caller-specific object that is passed as-is to the callback function

Returns :

nothing : the result is provided to the callback.

wakeupschedule→nextWakeUpSchedule() YWakeUpSchedule

Continues the enumeration of wake up schedules started using `yFirstWakeUpSchedule()`.

js	function **nextWakeUpSchedule**( )
nodejs	function **nextWakeUpSchedule**( )
php	function **nextWakeUpSchedule**( )
cpp	YWakeUpSchedule * **nextWakeUpSchedule**( )
m	-(YWakeUpSchedule*) **nextWakeUpSchedule**
pas	function **nextWakeUpSchedule**( ): TYWakeUpSchedule
vb	function **nextWakeUpSchedule**( ) As YWakeUpSchedule
cs	YWakeUpSchedule **nextWakeUpSchedule**( )
java	YWakeUpSchedule **nextWakeUpSchedule**( )
py	def **nextWakeUpSchedule**( )

Returns :

a pointer to a `YWakeUpSchedule` object, corresponding to a wake up schedule currently online, or a `null` pointer if there are no more wake up schedules to enumerate.

wakeupschedule→registerValueCallback() YWakeUpSchedule

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

```
js      function registerValueCallback( callback)
nodejs  function registerValueCallback( callback)
php     function registerValueCallback( $callback)
cpp     int registerValueCallback( YWakeUpScheduleValueCallback callback)
m       -(int) registerValueCallback : (YWakeUpScheduleValueCallback) callback
pas     function registerValueCallback( callback: TYWakeUpScheduleValueCallback): LongInt
vb      function registerValueCallback( ) As Integer
cs      int registerValueCallback( ValueCallback callback)
java    int registerValueCallback( UpdateCallback callback)
py      def registerValueCallback( callback)
```

The callback is invoked only during the execution of `ySleep` or `yHandleEvents`. This provides control over the time when the callback is triggered. For good responsiveness, remember to call one of these two functions periodically. To unregister a callback, pass a null pointer as argument.

Parameters :

callback the callback function to call, or a null pointer. The callback function should take two arguments: the function object of which the value has changed, and the character string describing the new advertised value.

wakeupschedule→set_hours() YWakeUpSchedule
wakeupschedule→setHours()

Changes the hours when a wake up must take place.

```
js      function set_hours( newval)
nodejs  function set_hours( newval)
php     function set_hours( $newval)
cpp     int set_hours( int newval)
m       -(int) setHours : (int) newval
pas     function set_hours( newval: LongInt): integer
vb      function set_hours( ByVal newval As Integer) As Integer
cs      int set_hours( int newval)
java    int set_hours( int newval)
py      def set_hours( newval)
cmd     YWakeUpSchedule target set_hours newval
```

Parameters :

newval an integer corresponding to the hours when a wake up must take place

Returns :

YAPI_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

wakeupschedule→set_logicalName() YWakeUpSchedule
wakeupschedule→setLogicalName()

Changes the logical name of the wake up schedule.

```
js      function set_logicalName( newval)
nodejs  function set_logicalName( newval)
php     function set_logicalName( $newval)
cpp     int set_logicalName( const string& newval)
m       -(int) setLogicalName : (NSString*) newval
pas     function set_logicalName( newval: string): integer
vb      function set_logicalName( ByVal newval As String) As Integer
cs      int set_logicalName( string newval)
java    int set_logicalName( String newval)
py      def set_logicalName( newval)
cmd     YWakeUpSchedule target set_logicalName newval
```

You can use `yCheckLogicalName()` prior to this call to make sure that your parameter is valid. Remember to call the `saveToFlash()` method of the module if the modification must be kept.

Parameters :

newval a string corresponding to the logical name of the wake up schedule.

Returns :

`YAPI_SUCCESS` if the call succeeds.

On failure, throws an exception or returns a negative error code.

wakeupschedule→set_minutes() YWakeUpSchedule
wakeupschedule→setMinutes()

Changes all the minutes where a wake up must take place.

```
js      function set_minutes( bitmap)
nodejs  function set_minutes( bitmap)
php     function set_minutes( $bitmap)
cpp     int set_minutes( s64 bitmap)
m       -(int) setMinutes : (s64) bitmap
pas     function set_minutes( bitmap: int64): LongInt
vb      function set_minutes( ) As Integer
cs      int set_minutes( long bitmap)
java    int set_minutes( long bitmap)
py      def set_minutes( bitmap)
cmd     YWakeUpSchedule target set_minutes bitmap
```

Parameters :

bitmap Minutes 00-59 of each hour scheduled for wake up.

Returns :

YAPI_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

wakeupschedule→set_minutesA() YWakeUpSchedule
wakeupschedule→setMinutesA()

Changes the minutes in the 00-29 interval when a wake up must take place.

```
js      function set_minutesA( newval)
nodejs  function set_minutesA( newval)
php     function set_minutesA( $newval)
cpp     int set_minutesA( int newval)
m       -(int) setMinutesA : (int) newval
pas     function set_minutesA( newval: LongInt): integer
vb      function set_minutesA( ByVal newval As Integer) As Integer
cs      int set_minutesA( int newval)
java    int set_minutesA( int newval)
py      def set_minutesA( newval)
cmd     YWakeUpSchedule target set_minutesA newval
```

Parameters :

newval an integer corresponding to the minutes in the 00-29 interval when a wake up must take place

Returns :

YAPI_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

wakeupschedule→set_minutesB() YWakeUpSchedule
wakeupschedule→setMinutesB()

Changes the minutes in the 30-59 interval when a wake up must take place.

js	function **set_minutesB**(**newval**)
nodejs	function **set_minutesB**(**newval**)
php	function **set_minutesB**(**$newval**)
cpp	int **set_minutesB**(int **newval**)
m	-(int) setMinutesB : (int) **newval**
pas	function **set_minutesB**(**newval**: LongInt): integer
vb	function **set_minutesB**(ByVal **newval** As Integer) As Integer
cs	int **set_minutesB**(int **newval**)
java	int **set_minutesB**(int **newval**)
py	def **set_minutesB**(**newval**)
cmd	YWakeUpSchedule **target set_minutesB newval**

Parameters :

newval an integer corresponding to the minutes in the 30-59 interval when a wake up must take place

Returns :

YAPI_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

wakeupschedule→set_monthDays() YWakeUpSchedule
wakeupschedule→setMonthDays()

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

```
js      function set_monthDays( newval)
nodejs  function set_monthDays( newval)
php     function set_monthDays( $newval)
cpp     int set_monthDays( int newval)
m       -(int) setMonthDays : (int) newval
pas     function set_monthDays( newval: LongInt): integer
vb      function set_monthDays( ByVal newval As Integer) As Integer
cs      int set_monthDays( int newval)
java    int set_monthDays( int newval)
py      def set_monthDays( newval)
cmd     YWakeUpSchedule target set_monthDays newval
```

Parameters :

newval an integer corresponding to the days of the month when a wake up must take place

Returns :

YAPI_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

wakeupschedule→set_months() YWakeUpSchedule
wakeupschedule→setMonths()

Changes the months when a wake up must take place.

```
js      function set_months( newval)
nodejs  function set_months( newval)
php     function set_months( $newval)
cpp     int set_months( int newval)
m       -(int) setMonths : (int) newval
pas     function set_months( newval: LongInt): integer
vb      function set_months( ByVal newval As Integer) As Integer
cs      int set_months( int newval)
java    int set_months( int newval)
py      def set_months( newval)
cmd     YWakeUpSchedule target set_months newval
```

Parameters :

newval an integer corresponding to the months when a wake up must take place

Returns :

YAPI_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

wakeupschedule→set_userData() YWakeUpSchedule
wakeupschedule→setUserData()

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

```
js      function set_userData( data)
nodejs  function set_userData( data)
php     function set_userData( $data)
cpp     void set_userData( void* data)
m       -(void) setUserData : (void*) data
pas     procedure set_userData( data: Tobject)
vb      procedure set_userData( ByVal data As Object)
cs      void set_userData( object data)
java    void set_userData( Object data)
py      def set_userData( data)
```

This attribute is never touched by the API, and is at disposal of the caller to store a context.

Parameters :

data any kind of object to be stored

wakeupschedule→set_weekDays() YWakeUpSchedule
wakeupschedule→setWeekDays()

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

```
js      function set_weekDays( newval)
nodejs  function set_weekDays( newval)
php     function set_weekDays( $newval)
cpp     int set_weekDays( int newval)
m       -(int) setWeekDays : (int) newval
pas     function set_weekDays( newval: LongInt): integer
vb      function set_weekDays( ByVal newval As Integer) As Integer
cs      int set_weekDays( int newval)
java    int set_weekDays( int newval)
py      def set_weekDays( newval)
cmd     YWakeUpSchedule target set_weekDays newval
```

Parameters :

newval an integer corresponding to the days of the week when a wake up must take place

Returns :

YAPI_SUCCESS if the call succeeds.

On failure, throws an exception or returns a negative error code.

wakeupschedule→wait_async() YWakeUpSchedule

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

```
js      function wait_async( callback, context)
nodejs  function wait_async( callback, context)
```

The callback function can therefore freely issue synchronous or asynchronous commands, without risking to block the Javascript VM.

Parameters :

callback callback function that is invoked when all pending commands on the module are completed. The callback function receives two arguments: the caller-specific context object and the receiving function object.

context caller-specific object that is passed as-is to the callback function

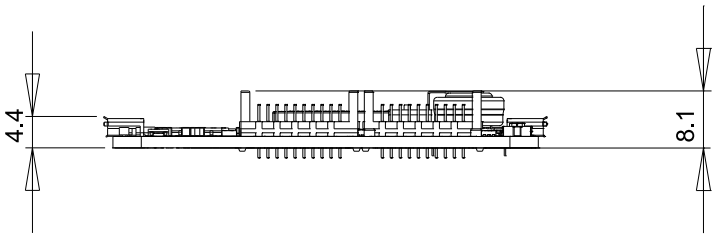
Returns :

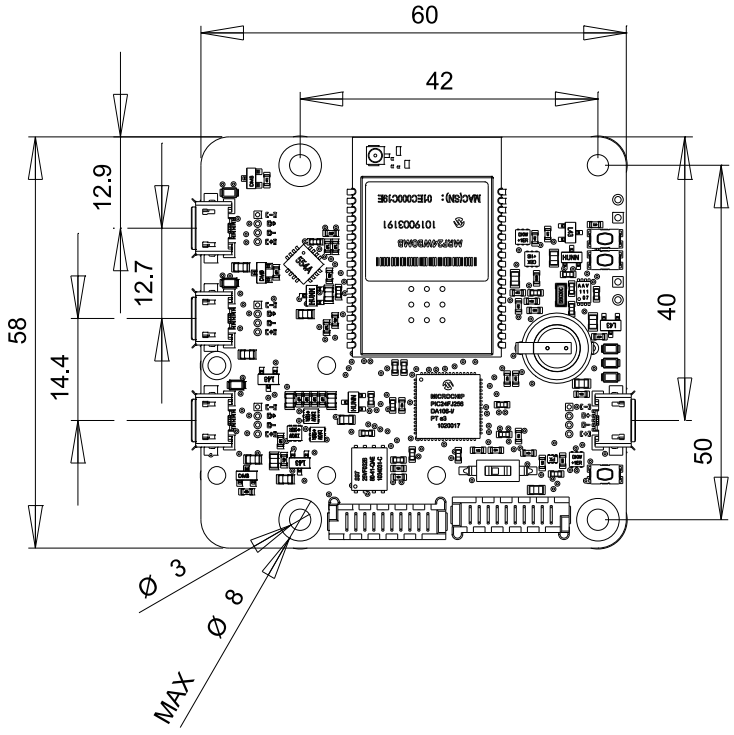
nothing.

12. Characteristics

You can find below a summary of the main technical characteristics of your YoctoHub-Wireless module.

Thickness	8.1 mm
Width	58 mm
Length	60 mm
Weight	30.1 g
USB connector	micro-B
Channels	3 ports
Max Current (continuous)	2 A
USB consumption	110 mA
Supported Operating Systems	Windows, Linux (Intel + ARM), Mac OS X, Android
Network connection	802.11b
Drivers	no driver needed
API / SDK / Libraries (USB+TCP)	C++, Objective-C, C#, VB .NET, Delphi, Python, Java/Android
API / SDK / Libraries (TCP only)	Javascript, Node.js, PHP, Java
RoHS	yes
USB Vendor ID	0x24E0
USB Device ID	0x0023
Suggested enclosure	YoctoBox-HubWlan-Transp





All dimensions are in mm
Toutes les dimensions sont en mm

YoctoHub-Wireless

A4

Scale
1:1
Echelle

Index

H

I

J

L

M

N